

University of Nevada, Reno

Smart-Cloud: A Framework for Cloud Native Applications Development

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Computer Science and Engineering

by

Jalal Hasan Ahmed Al Kiswani

Dr. Sergiu M. Dascalu/ Dissertation Advisor
Dr. Frederick C. Harris, Jr./ Dissertation Co-advisor

May, 2019

Copyright © by Jalal Kiswani
All Rights Reserved



THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

JALAL HASAN AHMED AL KISWANI

Entitled

Smart-Cloud: A Framework For Rapid Cloud Application Development

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Dr. Sergiu M. Dascalu, Advisor

Dr. Frederick C. Harris, Jr., Co-advisor

Dr. Feng Yan, Committee Member

Dr. Sage Hiibel, Committee Member

Dr. Scotty Strachan, Graduate School Representative

David W. Zeh, Ph. D., Dean, Graduate School
May, 2019

Abstract

The development of modern software applications requires the adoption of cutting-edge techniques, technologies, frameworks, tools, and infrastructure. Even though there are many options to choose in every category, the common characteristics are faster-delivery, lightweight and scalable applications, and lower cost. In addition, the automation of testing, deployment, and operations have become significant. All of these new characteristics would not be possible without Cloud-Computing, where services are delivered in three different flavors, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). However, developing high-quality cloud-based software applications that exploit the full benefits of cloud computing is challenging and costly, since it requires a long learning curve and it is challenged by the lack of expertise. In this dissertation, we propose Smart-Cloud, a framework for developing high-quality cloud applications. The primary goals of the framework are to enable the development of such applications to be more rapid, efficient, reliable, and straightforward. Smart-Cloud consists of a novel combination of reusable components that includes API's, services, and application framework. These components were designed and developed using cutting-edge architectural styles, best practices, and patterns such as the microservices architectural style and the twelve-factor-app. The framework adopts the metadata-driven design to enable dynamic and static generation of single artifacts, end-to-end features, and full applications. Moreover, it consists of a cloud-based platform that utilizes the model-driven development approach to enable the use of the framework without writing code. The framework also follows the software product lines approach of enabling users to develop new projects efficiently. Furthermore, a PMS case study is presented, which is a publication management system developed using the framework without writing any code. Moreover, we discuss the results of the user study that we conducted with 36 experienced professionals from industry and academia, who assessed the framework and provided their evaluation and feedback.

Dedication

I dedicate this dissertation to my father, my mother, my brothers, my wife, and my children; without all of them, I wouldn't be the person that I have become today.

Acknowledgments

I would like to sincerely thank my co-advisers Dr. Sergiu M. Dascalu and Dr. Frederick C. Harris, Jr. for their continuous guidance and support; I am speechless for all what they did for me to be a better person, student, and researcher. In addition, I would like to thank my committee members: Dr. Scotty Strachan, Dr. Feng Yan, and Dr. Sage Hiibel for their time and efforts being part of this committee.

This material is based in part upon work supported by the National Science Foundation under grant number(s) IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Table of Contents

Abstract	i
Dedication	ii
Acknowledgments	iii
List of Tables	vii
List of Figures	viii
1 Introduction	1
2 Background	7
2.1 Cloud Computing	7
2.1.1 History and Evolution	10
2.1.2 Deployment Models	14
2.1.3 Service Delivery Models	15
2.1.4 Cloud Providers	17
2.1.5 General Benefits of Cloud Computing	19
2.1.6 Challenges, Barriers and Risks	21
2.1.7 Impact on Hardware Business	22
2.1.8 Cloud Adoption	24
2.2 Software Architecture	25
2.3 Software Product Lines	27
3 Literature Review and Related Work	30
3.1 Design Models of Cloud Applications	32
3.2 Application's Clients	34
3.3 Cloud vs Traditional On-premise Applications	35
3.4 Design and Architecture of Cloud Applications	36
3.4.1 Monolithic Applications Architecture	37
3.4.2 Service-Oriented Architecture	37
3.4.3 Microservices Architecture	38
3.4.4 Cloud-Native Applications	40
3.5 Main Characteristics of Cloud Applications	40

3.6	DevOps and Cloud Applications Development Process	44
3.7	Benefits of Cloud Applications	45
3.8	Challenges of Cloud Applications	46
3.9	Strategy for Moving to the Cloud and Customer's Motivation	47
3.10	Industry-based Related Work	48
3.11	Related Work Discussion	48
4	Framework Overview	50
4.1	High-level Overview	50
4.2	High-level Architecture	52
4.3	Significance	53
4.4	Features and Characteristics	54
4.5	Technology	55
5	Design, Architecture, and Implementation	56
5.1	Framework Architecture	56
5.1.1	Libraries and APIs	57
5.1.2	App Logic Services	63
5.1.3	Cross-cutting Services	63
5.1.4	Platform Apps (Clowiz Platform)	66
5.1.5	Database Design	68
5.2	Framework User-Interface	70
5.3	Framework Implementation	73
6	Case Study: Publication Management System	74
6.1	PMS Overview	74
6.2	Implementation	75
6.2.1	Create PMS Application	75
6.2.2	Design PMS App	75
6.2.3	Design PMS Pages	76
6.2.4	PMS Prview	78
6.2.5	Download PMS Source Code	79
6.2.6	Import PMS Project into IDE	80
6.2.7	Launch PMS Locally	80
6.3	PMS Deployment to the Cloud	81
7	Evaluation and Discussion	87
7.1	User Study	87
7.1.1	Participants	87
7.1.2	Methodology	89
7.2	Results and Discussion	92
7.2.1	PMS Results and Evaluation	92
7.2.2	Users Study Results	94

7.3	Discussion	97
8	Conclusion and Future Work	102
8.1	Research Question Revisited	102
8.2	An Overall Characterization of Smart-Cloud	102
8.3	Cloud Computing	103
8.4	Concluding Remarks and Future Work	105
	Appendix	106
A	User Manual	107
A.1	Introduction	107
A.2	Home	108
A.3	Authentication App	109
A.3.1	Sign Up	109
A.3.2	Reset Password	111
A.3.3	Log In	111
A.4	CodeGen App	112
A.4.1	CodeGen Metadata	113
A.4.2	Fields Data Types	114
A.4.3	Technologies	115
A.5	FeatureGen App	124
A.5.1	FeatureGen Metadata	125
A.5.2	Fields Data Types	126
A.5.3	FeatureGen Technology	126
A.5.4	Feature Page Preview	127
A.5.5	FeatureGen Generated Code	127
A.6	AppGen App	127
A.6.1	App Manager	128
A.6.2	App Designer	131
A.6.3	Page Designer	133
B	Excerpts from Source Code	136
C	Clowiz Full Questionnaire	149
	Bibliography	155

List of Tables

2.1	Total worldwide spending on cloud services forecast	8
5.1	Metadata API classes	60
5.2	The list of generators supported by CodeGen	67
7.1	The results of implementing PMS using Smart-Cloud	93
7.2	The results of cloud application questions	94
7.3	The results of Clowiz platform questions	95
7.4	The results of the most significant app in Clowiz question	95
A.1	Fields metadata attributes	135
B.1	Source code excerpts summary	136

List of Figures

1.1	Examples of monolithic software systems	2
1.2	Smart-Cloud high-level architecture	5
1.3	Native Cloud Applications Reference Architecture (NCARA)	5
2.1	Simplified cloud infrastructure [52]	9
2.2	Providers and users of cloud services [6]	10
2.3	Mainframes time-sharing [95]	12
2.4	Cloud computing evolution [80]	13
2.5	Cloud computing services delivery models [58]	16
2.6	Top cloud services providers [107]	18
2.7	Automated vs traditional applications scalability comparisons [124]	20
2.8	Cloud computing barriers ranking by users in 2011 [58]	23
2.9	Cloud computing challenges in 2018[107]	23
2.10	Cloud adoption as of 2017 [58]	25
2.11	Software product lines and time-to-market [71]	27
2.12	A Software Product Line example	28
3.1	Cloud computing services worldwide spending [66]	30
3.2	Software taxonomy [28]	31
3.3	Tenancy-models in cloud applications [54]	33
3.4	A pyramid of modern cloud native applications [31]	39
3.5	Cloud native applications external environment [103]	41
3.6	An example of DevOps team structure [16]	45
4.1	High-level architecture of Smart-Cloud	52
5.1	Medium-level architecture of Smart-Cloud	57
5.2	Metadata API Class Diagram	59
5.3	Exporter API Package Diagram	61
5.4	Exporter API Core Package Class Diagram	61
5.5	Exporter API Commons Package Class Diagram	62

5.6	Exporter API Technologies Package Class Diagram	62
5.7	Metadata Service Class Diagram	64
5.8	DevOps Service Class Diagram	65
5.9	Metadata Service Class Diagram	65
5.10	AppGen Development Process	68
5.11	AppGen Status Workflow	69
5.12	Entity Relationship Diagram of Smart-Cloud	69
5.13	Clowiz Platform Home Page	70
5.14	Clowiz Authentication Page	71
5.15	CodeGen App Page	71
5.16	FeatureGen App Page	72
5.17	AppGen App Page	72
5.18	AppGen App Page	73
6.1	PMS use case diagram	75
6.2	PMS class diagram	76
6.3	PMS project in AppGen manager	77
6.4	PMS project in AppGen designer	77
6.5	PMS University page design	78
6.6	PMS Department page design	78
6.7	PMS Author page design	79
6.8	PMS Publication page design	79
6.9	PMS preview	80
6.10	PMS Maven import	81
6.11	PMS project structure	82
6.12	PMS console output	82
6.13	PMS Home page	83
6.14	PMS universities management page	83
6.15	PMS departments management page	84
6.16	PMS authors management page	84
6.17	PMS publication management page	85
6.18	PMS App on PCF	85
6.19	PMS MySql Service	86
6.20	PMS URL Routing	86
6.21	PMS Production on PCF	86
7.1	Experience levels of the user study participants	88
7.2	Academic levels of the participants in the user study	88
7.3	Involvement of the participants in software development activities	89
7.4	Results of Clowiz usability	95
7.5	Results of the understanding of AppGen process	96

7.6	Results of Clowiz reducing the cost of cloud applications development	96
7.7	Results of generated code quality of Clowiz	97
7.8	Results of using Clowiz in day-to-day work	97
7.9	Results of Clowiz overall quality of generated applications	98
7.10	Shortage in experienced cloud application's developers	100
A.1	Clowiz Home Page	109
A.2	Authentication App	110
A.3	Create an Account	110
A.4	Sign Up Confirmation	111
A.5	Reset Password Form	111
A.6	Login Form	112
A.7	CodeGen App Page	113
A.8	CodeGen metadata	114
A.9	Java Class Exporter in CodeGen	116
A.10	Java Class with Lombok Exporter in CodeGen	116
A.11	JPA Entity Exporter in CodeGen	117
A.12	JPA Entity with Lombok Exporter in CodeGen	118
A.13	JSF Managed Bean Exporter in CodeGen	118
A.14	PrimeFaces View Exporter in CodeGen	119
A.15	PrimeFaces with Backend Binding Exporter in CodeGen	119
A.16	HTML Only Exporter in CodeGen	120
A.17	HTML with Bootstrap exporter	121
A.18	HTML Full Page with Bootstrap Exporter in CodeGen	122
A.19	H2 Database Exporter in CodeGen	122
A.20	MySQL Exporter in CodeGen	123
A.21	Microsoft SQL Server Exporter in CodeGen	123
A.22	Oracle Exporter in CodeGen	124
A.23	FeatureGen Page	125
A.24	FeatureGen Metadata	126
A.25	Clowiz AppGen Home Page	129
A.26	AppGen App Designer	132
A.27	AppGen Page Designer	133

Chapter 1

Introduction

Organizations from all domains implement software applications with different scales for both internal and external use. For example, governments implement software systems to enable internal administration management with thousands of users. At the same time, they might implement an online software system to enable smart-government online services for citizens and residents. Another example is that a scientific community may implement a group scoped system for scientific big data management while enabling visualization features for external entities.

The monolithic approach of software development was the dominant model, in particular, building software applications as a single deployable unit. In the past, this approach was practical since software size was relatively small and consisted of a lower number of software components and functionality. In addition, this approach was common due to its convenience and ease of development. Moreover, the systems were only used by a limited number of users, with relatively long-term and stable requirements. Figure 1.1 shows some examples of monolithic applications.

However, things have changed, with the Internet, smartphones, big data, cloud computing, and the startups' ecosystem. Time to market and response to new requirements have become critical. Scalability has reached limits that were never possible and required before, with hundreds of millions of concurrent users accessing the same service at the same time. In addition, the Internet of Things (IoT) has also affected this wave, which increased the number of devices accessing online services exponentially.

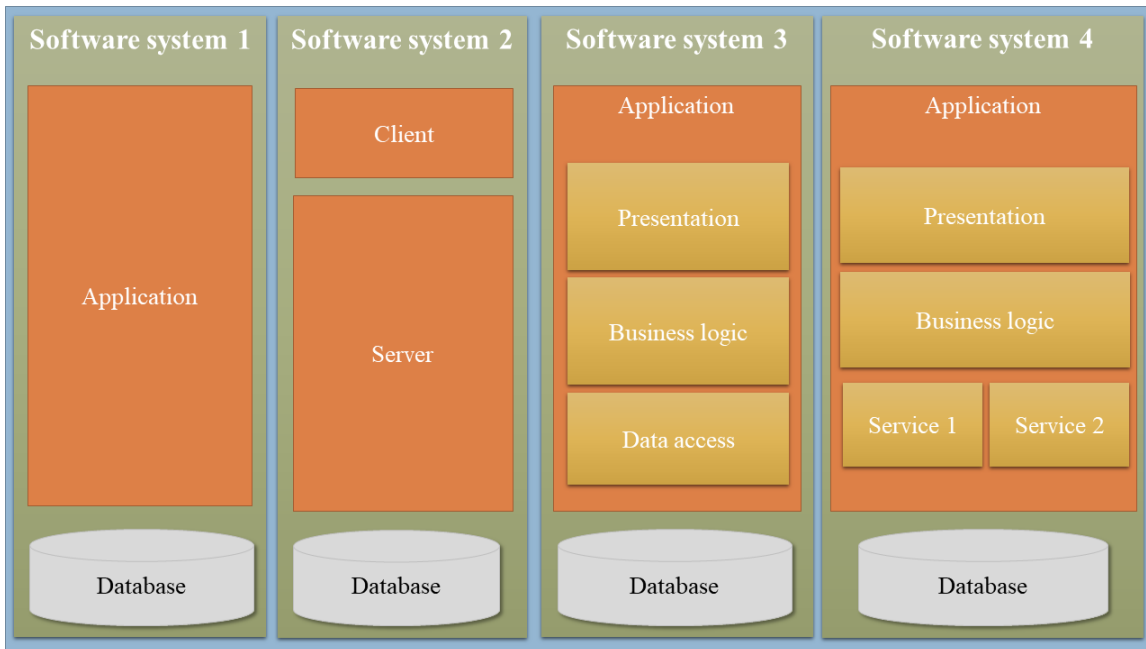


Figure 1.1: Examples of monolithic software systems

With all these factors, the monolithic approach of software applications development has become a serious bottleneck. In essence, it may affect an organization's own existence. The high-risks produced by following this approach include the high cost of implementation and scalability, heavyweight testing and deployment processes, and complexity of development. Moreover, factors such as global software engineering -where remotely located developers are working on the same project, and the need to use different technologists in the same project (e.g., Angular for front-end, Python for presentation tier, and Spring-Boot for backend) increased the challenge of selecting the monolithic approach, where the same technology is the main theme for most of the applications. Furthermore, various front-end technologies such as mobile devices, browsers, desktop applications, and IoT devices requires lighter communication and full separation between front-end and back-end technologies.

Even with the existence of techniques and concepts such as Object Oriented Programming (OOP), design patterns, reusable components development, application framework, and software product lines (SPL), building the modern requirements of a software system is still challenging.

Service Oriented Architecture (SOA) has been an approach of software architecture and development driven by academia and practitioners. In particular, it is a software construction model that aims to change how the software systems are built. Its main idea is based on separating an application into smaller self-contained components, encapsulated and independently deployable, which communicate over a standard protocol. The commonly used protocol for SOA is Simple Object Access Protocol (SOAP) based web services. However, SOA didn't get high traction because of its heavy-weight nature, in particular, communication, development and configuration complexity, tools and technologies. Moreover, the common use of SOA applications was for medium to large-scale information systems that require a relational database as the main persistence mechanism of the application's data. In fact, a unified instance of the database was used, which limited flexibility, where developers were constrained in their design decisions and created a bottleneck on architectural and database design decisions.

Consequently, a new architectural style has evolved: Microservices Architecture (MsA). In MsA, a software application is built as small, lightweight, reusable and self-deployable components that communicate over a lightweight protocol. In fact, these components can be built using any technology. Moreover, every service has its own business logic, storage engine, and persistent mechanism. Therefore, developers have full decision control over the design, technology, and implementation of their services. The commonly used architectural style in MsA is the Representational State Transfer (REST) style that operates over HTTP protocol. The components that are fully comply to the REST style are called RESTful web-services, which was created as a lightweight alternative of the SOAP web-services. In particular, it works as a layer that operates directly over the HTTP protocol [40].

Microservices style overcomes many issues and challenges over the monolithic approach and SOA. However, it requires a careful early architecture and design, mainly for the services and their interfaces. In fact, these decisions may affect the overall success of the project.

Even though MsA can be implemented in traditional on-site applications, its full advantage can directly be gained if implemented in Cloud Applications. Cloud Applications are the applications deployed on Cloud Computing environments such as Infrastructure as a Service (IaaS) or Platform as a Service (PaaS), and publicly accessible over the Internet by the intended users. As explained earlier, adopting MsA on the cloud is more efficient than Monolithic and SOA. In addition, implementing the quality attributes of scalability, high availability, auditability, monitorability, and traceability will achieve the full benefits of Cloud Computing. In fact, including these quality attributes along with MsA in an application is termed as Cloud Native Applications (CNA).

MsA and CNA are relatively new terms, and we think it's considered a new green area for research and development. In fact, having an approach that can enable building CNA applications based on MsA to be repeatable, reliable, secure and rapid will open the door for new opportunities and potential. In particular, developing different types of systems such as Information Systems, big data management or any other types of cloud applications is becoming more rapid and cost-effective.

However, developing high quality cloud-based applications is challenging and costly. In particular, (i) the long learning curve, (ii) the lack of special expertise, (iii) the immaturity of technologies and tools, (v) the limited reusability, (iv) and portability, are all factors that characterize the risks of developing such applications.

For those reasons, we try to answer the following research question of this dissertation: *Is there an approach that enables the development of cloud based applications to be: rapid, efficient, and straightforward?*

As an answer for this question we propose Smart-Cloud. In particular, Smart-Cloud is an architectural and design approach for cloud application development. In fact, it utilizes concepts from reusable software components, applications framework, and software product lines (SPL). The high-level architecture of Smart-Cloud is shown in Figure 1.2.

During the different phases of work presented in this dissertation, best practices,

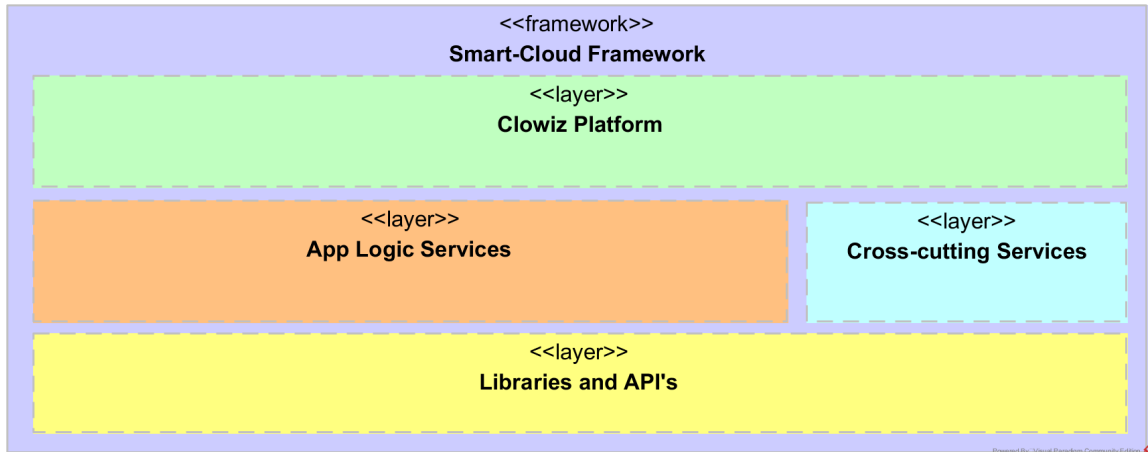


Figure 1.2: Smart-Cloud high-level architecture

patterns, tactics, and techniques of software architecture and design are extensively used. In addition to the Smart-Cloud framework, we propose a Native Cloud Application Reference Architecture (NCARA) for developing Cloud Native Applications based on Microservices. NSCRA is shown in Figure 1.3.

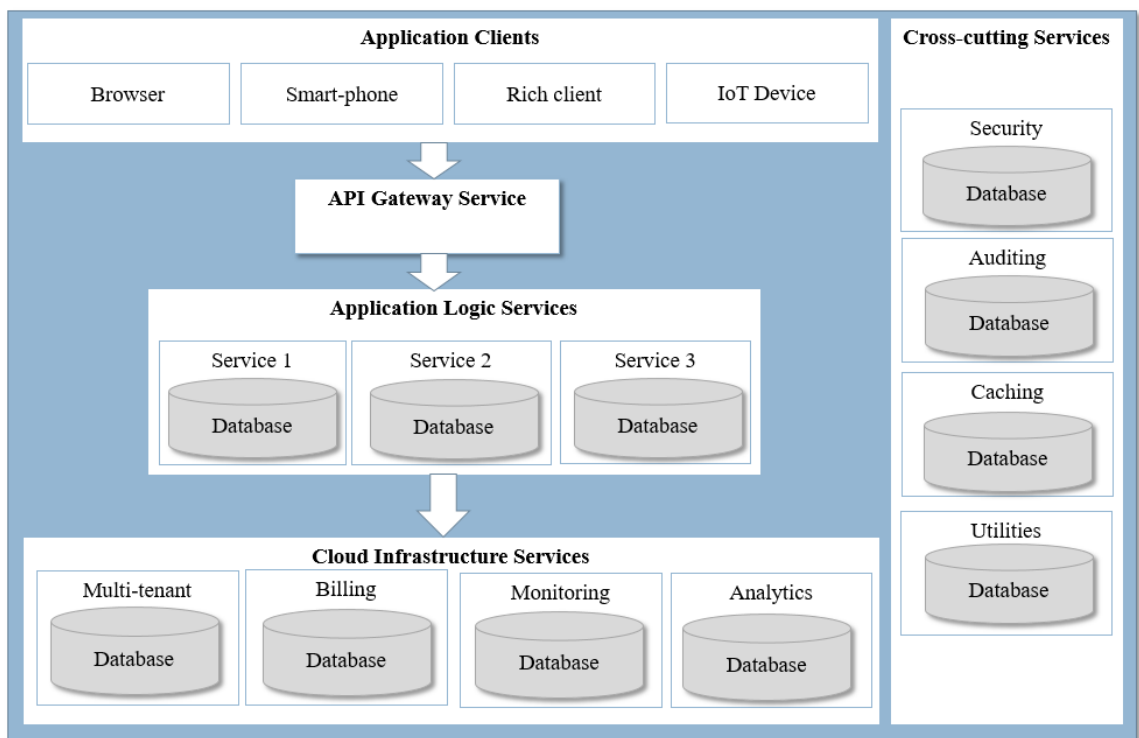


Figure 1.3: Native Cloud Applications Reference Architecture (NCARA)

The proposed work has been fully architected and designed using the Unified Modeling Language (UML). In addition, components are implemented using Spring Java technology-based framework. Moreover, Continuous Integration (CI), Test Automation, Continuous Delivery (CD) implemented using Jenkins [65], Maven [5], JUnit [4], and other modern software engineering practices have also been used.

The validation of Smart-Cloud is based on three phases. First, the implementation of the full backend of Clowiz, where Clowiz is a proposed model-driven development platform that enables application developers to build cloud-based information systems in an interactive approach without any coding needs. Second, the implementation a PMS case study, which is a publication management system that is fully developed using the Clowiz platform without writing code. Thirdly, a comprehensive user study that was conducted for more than 30 professional from industry and academia, which shows how the framework can reduce the development time and achieve the required quality attributes.

All the implementation artifacts will be published as an open source project on GitHub. These artifacts include source code, user manual, and application programming interfaces (API) documentation.

This dissertation is organized into eight chapters and two appendices. This chapter covers the introduction. Chapter 2 discusses the background. Chapter 3 presents the literature review and related work. Chapters 4, 5, 6 present the proposed work, its design and architecture, and application case study. The evaluation of the work is discussed in Chapter 7. Chapter 8 concludes the dissertation and identifies several directions of future work. Finally, several appendices such as User Manual, Excerpts from the Source Code, and the full questionnaire are all presented in Appendix A, B, and C.

Chapter 2

Background

As discussed in Chapter 1, the work presented in this dissertation is to propose Smart-Cloud, a framework to enable repeatable, reliable, efficient and rapid development of native cloud applications. This chapter aims to provide the reader with the required information to understand the following chapters through background about cloud computing, software architecture, and software product lines.

2.1 Cloud Computing

Cloud computing has made tremendous changes and improvements in the information technology industry, where using 1,000 servers on the cloud for one hour is cheaper than using one server for 1,000 hours [7]. In fact, without cloud computing, many startup companies would not even exist [112] or would not achieve their current economies of scale [115]. As shown in Table 2.1, worldwide spending on public cloud services was almost \$210 billion in 2016, and it is expected to reach \$383 billion by 2020 according to Gartner news [100].

Cloud computing is one of the important innovations in information technology. It changed the way services are delivered to end users and affected almost every industry and discipline. Startup companies do not need to worry about investing in large data centers and hardware anymore. Software developers can start building software applications on top of platforms that can enable rapid-web application development, which can be deployed immediately. Enterprise organizations do not need

Table 2.1: Total worldwide spending on cloud services forecast

Year	Spending (Millions of dollars)
2016	\$209,244
2017	\$246,841
2018	\$287,820
2019	\$332,723
2020	\$383,355

to buy expensive software that may become stale with time and need large operational and maintenance costs. In addition, mobile Internet will be 10 times more spread than the traditional desktop Internet, connecting more than 10 billion devices such as smart-phones and sensors [11]. Moreover, it will enable saving costs and delegate liabilities [52].

Cloud computing has opened new opportunities, trends, and needs such as mobile interactive applications, parallel application processes, the large growth in data size, the rise of analytics, and the need of putting the data near applications, real-time decisions, and the Internet of Things.

Even though many definitions introduced in this article are about cloud computing, we find the following definition to be the most comprehensive:

A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet [38].

Cloud computing has five main characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service [81]. Figure 2.1 shows the simplified cloud infrastructure.

Even though cloud computing started to get high traction after 2005 [52], the idea was discussed in the mid of the 20th century as Utility Computing. History and evolution are presented in Section 2.1.1. Furthermore, the section includes the

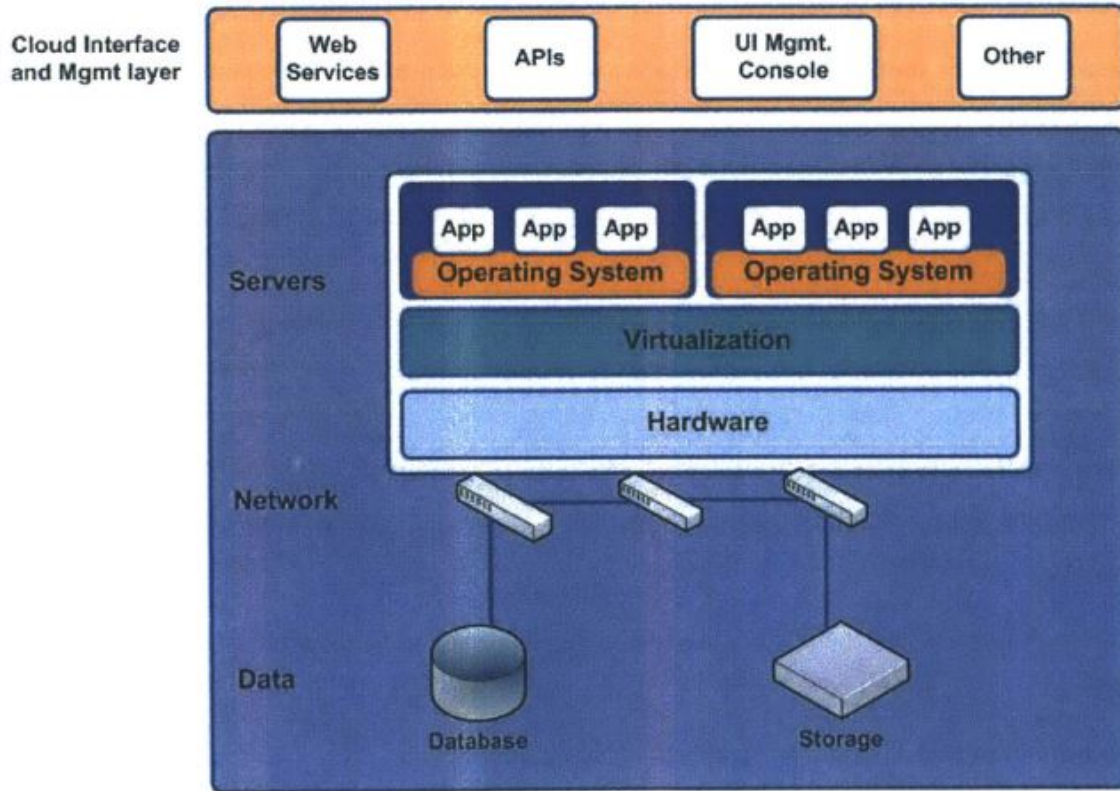


Figure 2.1: Simplified cloud infrastructure [52]

enablers that caused the massive growth in this technology.

Cloud can be deployed on different models, private cloud, community cloud, public cloud, and hybrid cloud. The main difference between these deployment models is the target audience and public accessibility. More details about cloud deployment models are discussed in Section 2.1.2. However, for the rest of this dissertation, the focus is on public and community clouds (assuming community is publicly accessible).

With cloud computing, several data centers will be turned into a single pool of computing utilities, which will enable the illusion of infinite resources. Cloud computing can deliver services in three different models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The sum of these services, data centers they operate on, and the software that runs and manages these data centers, is called “The Cloud” [7]. These three service delivery models are

discussed in detail in Section 2.1.3.

As shown in Figure 2.2, entities on which providers of cloud services operate are called cloud providers, and the entities in which users consume those services are called cloud users. Dedicated discussion about cloud providers, the advantages of being one, major providers and their classifications are presented in Section 2.1.4.

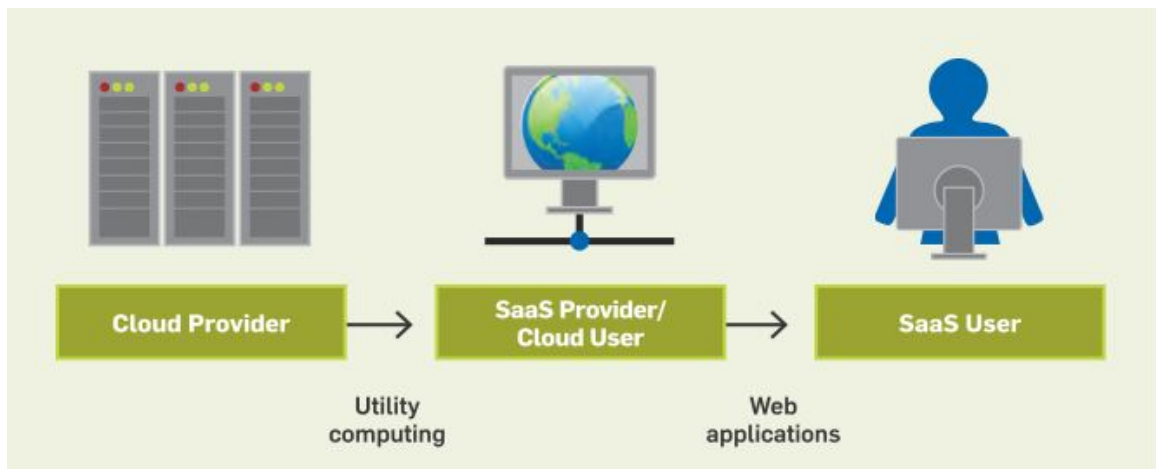


Figure 2.2: Providers and users of cloud services [6]

Furthermore, and since every technology introduces advantages and benefits on one side, there are disadvantages and challenges on the other side. Benefits of cloud computing are presented in Section 2.1.5, while challenges, barriers, and risks are discussed in Section 2.1.6 for both service providers and users.

Impact on the hardware industry is discussed in Section 2.1.7. Finally, the adoption of cloud computing is presented in Section 2.1.8.

2.1.1 History and Evolution

Users of utility services such as electricity and telephones employ services based on a model called a subscription. In the subscription model, the billing amount is based on service usage. The usage is measured by specific metrics based on the type of service. Usage-based billing called the pay-as-you-go model. The rationale behind this approach of service delivery is based on resources and services shared among different customers. In particular, this service delivery will reduce the overall cost

of resources and services and enable providing services at competitive prices. For example, the sharing of electrical lines connecting a whole neighborhood can reduce the cost by having only very short distance of lines to be dedicated for individuals, which will enable a lower cost of service.e. On the other hand, having an electricity line for a person who lives in his own large farm may be very expensive, since it may require dedicated expensive resources.

Offering information technology services on the utility-based subscription model is not a new idea. In fact, the first discussion about offering computing as a utility started in 1961 when John McCarthy said that computation might be offered as a utility someday [38]. Later on, Parkhill discussed in 1966 on how a revolution in distribution and utilization of computer power may enable social changes and opportunities of human development [96]. In particular, he discussed benefits, challenges, and future directions and potential of computing utility. Parkhill said that data transmission, expensive hardware, and limited hardware capabilities were the main barriers. Furthermore, he discussed working to overcome these barriers to open the door for new opportunities of eliminating the need of local storage and allow direct memory-to-memory communication between two remotely located computers, and enable faster growth in distributed computing. In addition, this may enable better teaching and information sharing.

Meanwhile, companies like IBM were working on the time-sharing technique [34] as shown in Figure 2.3. In time-sharing, an expensive computer such as mainframe was enabled to share computing resource with many users. However, due to the limitations described earlier (e.g., expensive hardware), this was not executed on a large scale and was based on a single organization or community.

As shown in Figure 2.4, the evolution of personal computers in the 1980's has made computers available for common people. This was one of the motivations to create the next big thing, the Internet. In the 1990's, a global network of networks was built based on a standard protocol such as TCP/IP, SSL, HTTP, FTP, SMPT, or POP. This global network was named the Internet and has affected all aspects of

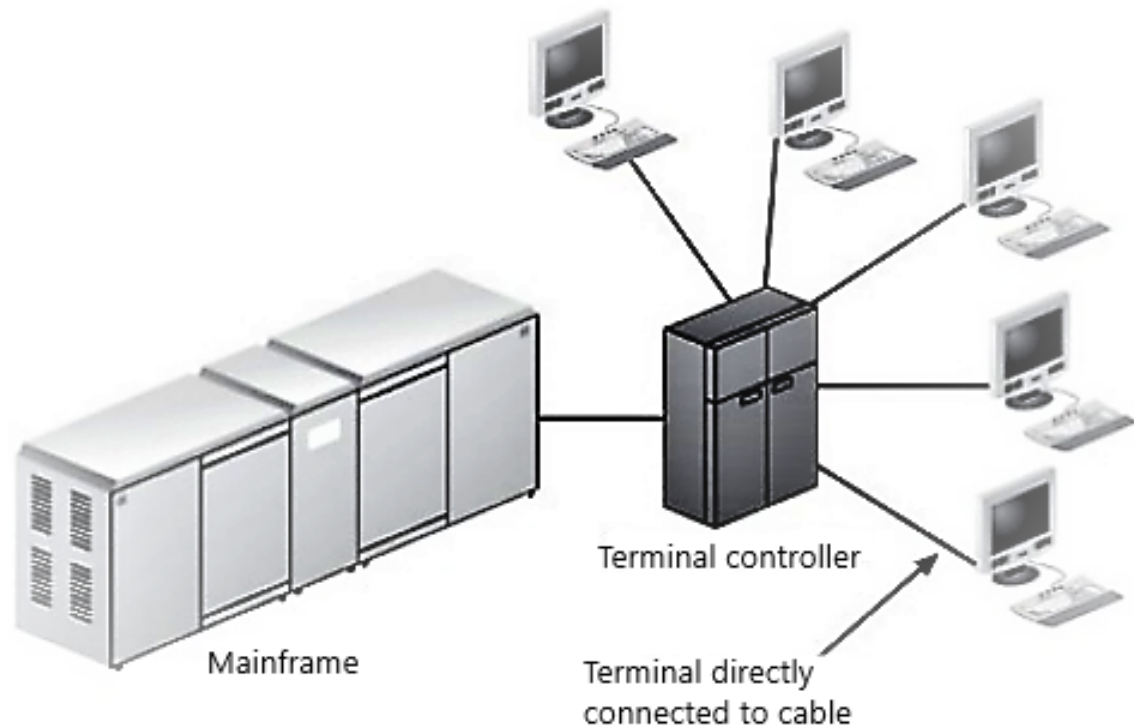


Figure 2.3: Mainframes time-sharing [95]

our lives. Meanwhile, the communication technology and hardware were improving as well, which enabled cheaper and faster computing and data transmission.

Since the late 1990s, businesses, academia, governments, and the general public started to offer and use services over the Internet. In the beginning, these services were mainly offering software applications over the Internet. In 2000-2002, Intel failed to offer computing services for organizations and companies, because it required negotiations and long-term contracting, which did not enable the growth of their approach [7].

Meanwhile, grid computing started to gain some traction to utilize the ability to distribute tasks between commodity computers that are available in different geographical locations, and offer computing power on demand [20, 38]. In fact, grid computing was later developed into cloud computing with the support of virtualization [14].

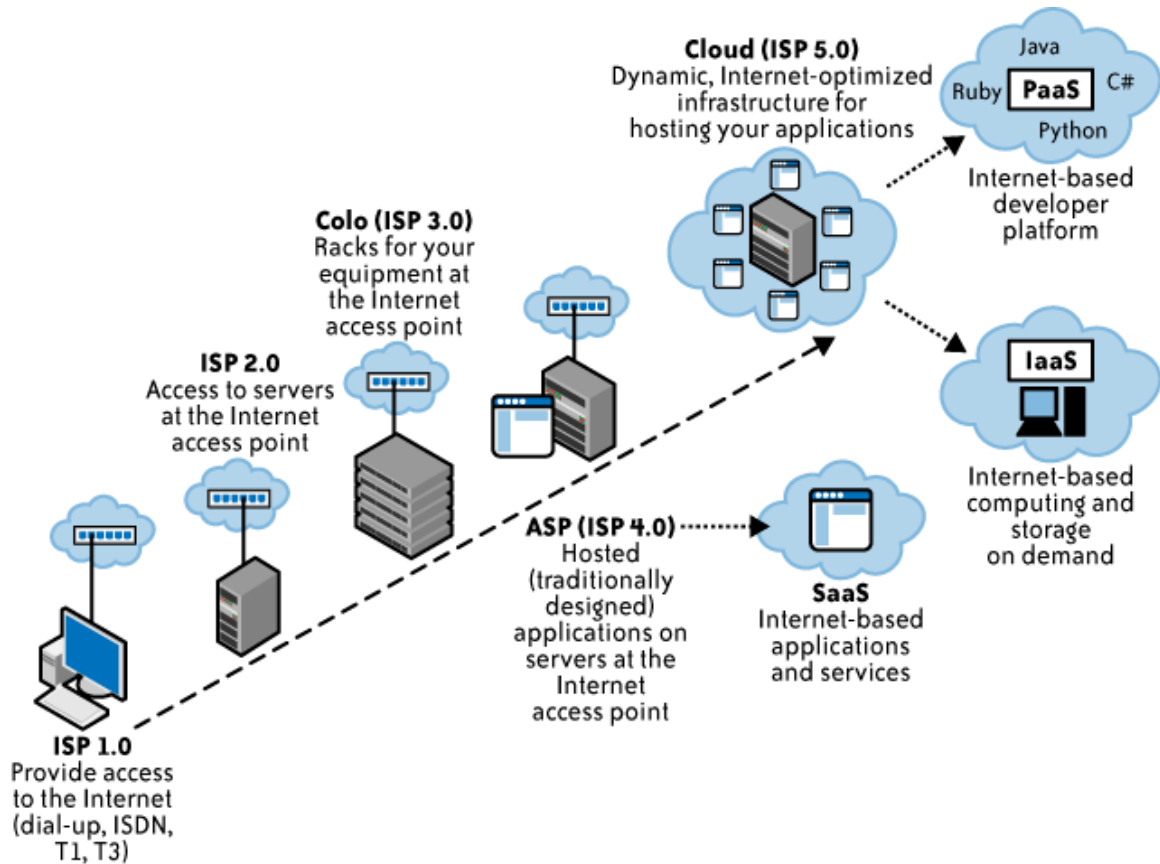


Figure 2.4: Cloud computing evolution [80]

In 2003, Jim Gray, the manager of Microsoft Research Lab in San-Francisco, expressed his opinion about the future of distributed computing, where software can be available in different physical locations and communication with other software over standard protocols. He discussed how the free services provided by internet service providers were actually not free and were paid for by advertisements. In addition, he discussed the high cost of ownership, which reached \$1 trillion per year. An exciting point in his report is that in 2002, at Google, only 25 operation staff were managing a two-petabyte database and distributed it over 10,000 server using automation tools. In fact, this was the main reason that Google was generating profits since they had lower operational costs. This applies also to other giant vendors such as Yahoo and Hotmail. He discussed how the future of the Internet would depend on computer-to-computer interaction. However, the advertisement model, which was the

main revenue stream at that time, was not sufficient, and companies need to invent a new business model that leverages the new trend [53].

Later, smart-phones started to be everywhere, which increased the number of internet users to orders of magnitudes. Sensors and smart-device industries were growing, and the Internet of Things (IoT) became a major discipline. On the other hand, the economic crisis in 2008 motivated all type of entities to look into cheaper and more efficient ways of doing their businesses. Moreover, space has become a serious issue for corporates, where the quarter of their data centers are out of space [112].

From the technical perspective, the drop-in hardware prices along with the increase in computation power and storage capacity enabled building data centers from commodity computers [38]. In addition, this was helped by the decrease in the cost of utilities, and communication of small to medium data centers [112]. Moreover, the operational cost was reduced due to the availability of automation software tools. Furthermore, wireless device adoption and smartphones removed the dependency of poor infrastructure and enabled even developing countries to be part of the revolution. Moreover, resource pooling using virtualization technology enabled the lower cost and more scalability [7, 52, 58]. Also, adoption of free open-source software, global workforces, and agile software processes played important roles in this evolution [52].

From economical perspectives, customer behaviors have changed from buying expensive long-term services and assets to subscription low-price services with low commitments. In particular, they were moving from capital and asset expenses (CapEx) to operational expenses (OpEx). In addition, the economic crisis in 2008 played a major role where organizations started to look at alternatives to reduce the cost of doing business [58].

2.1.2 Deployment Models

Cloud computing can be deployed in different deployment models, based on the target audience and users. These models are (i) private cloud, (ii) community cloud, (iii)

public cloud, (iv) and hybrid cloud.

Private cloud is a data center or set of data centers that are provisioned for and used by only one organization or corporation. The users of this model are the organization's internal or external users. In fact, this model is an organization very similar to a traditional on-premise data-center; however, the utilization of cloud computing technologies (e.g., virtualization), and the possibility to be run and be managed by third party organizations are key differences [81]. An example of this type of cloud is a virtual data center built by a bank to deploy a core banking system.

A community cloud has a broader audience beyond the same organization. It serves a group of users who share common interests and concerns. For example, satellite images datasets and tools may be deployed on specific data centers managed by NASA or the National Science Foundation in the USA, to enable domain and data scientists to perform big-data analytics on these data sets [81].

In the public cloud, services are open to the general public on the pay-as-you-go model for open use. An example of this is the public hosting providers, which can enable businesses to deploy their Internet applications and support rapid scalability and monitoring [81]. This is the most popular type of cloud deployment models among individuals and small-medium businesses. On the other hand, hybrid cloud is more preferred for enterprises [106].

2.1.3 Service Delivery Models

Cloud computing provides services in different service delivery models. Even though there is a unified classification for these services, for now, we will go through NIST service models, which are Infrastructure as a Service, Platform as a Service, and Software as a Service [81]. These delivery models are shown in Figure 2.5.

Infrastructure as a Service (IaaS) provides the lowest level of services from cloud providers to cloud user (consumers). IaaS includes services such as computation, storage, networks and any other fundamental infrastructure. In this model, cloud users will be able to install and deploy arbitrary software including operating systems and

Public Cloud Spectrum: Stack View

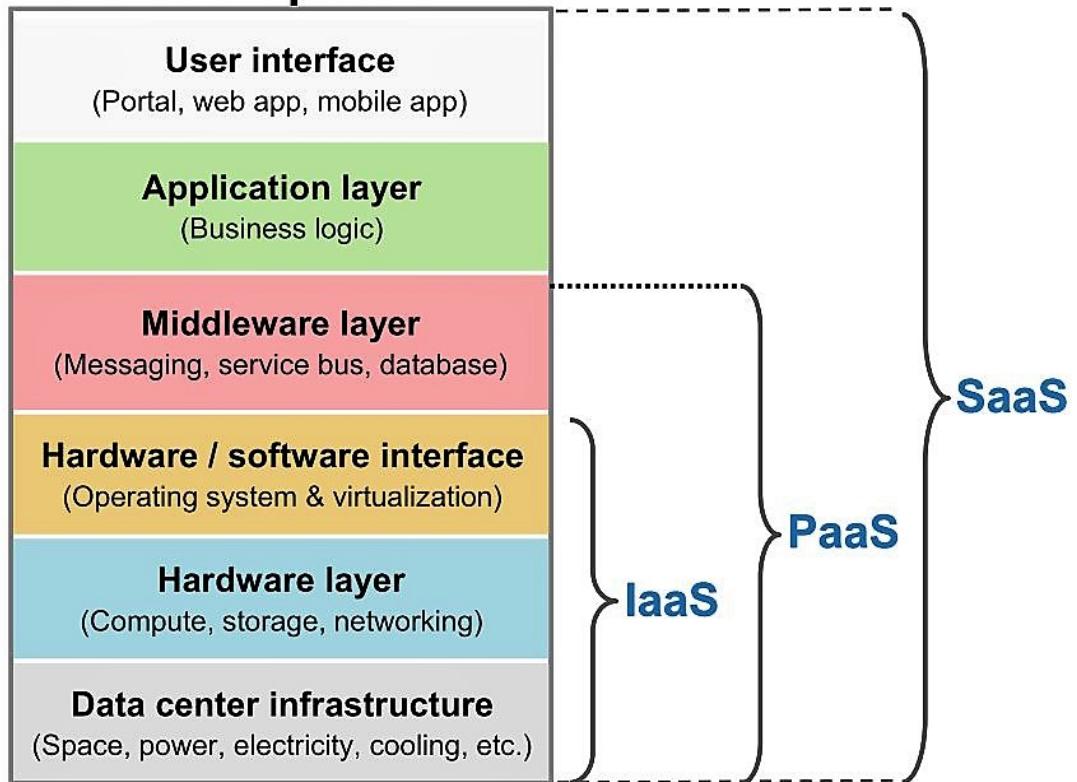


Figure 2.5: Cloud computing services delivery models [58]

applications. However, users do not have the ability to manage the underline cloud infrastructure. In particular, it is a collection of hardware and software that enables the special characteristics of cloud computing. IaaS includes virtual machines, servers, storage, load balancer, a network [52], and may include an operating system [106].

On the top of IaaS, Platform as a Service (PaaS) is provided. PaaS is a service that enables PaaS users to build and/or deploy their applications using language, libraries, services and tools provided by PaaS providers on a cloud platform. As we might expect, PaaS users do not have any control over the platform or cloud infrastructure. However, they have control over their application's management [81]. PaaS may include execution runtime, database, web servers, and development tools [52, 106].

Software as a Service (SaaS) is defined as the applications provided over the In-

ternet [7]. Another definition is the capability given by application providers (SaaS providers) to consumers (SaaS users) to use applications' running over a cloud infrastructure. In this model, SaaS users cannot control the underlying infrastructure, the platform or the applications; however, a possible configuration for the user's specific instances may be provided [81]. A research group that works for IBM argues that SaaS should be a single instance of a software system, serving a large group of customers over the Internet, built on top of a multi-tenancy platform [54]. Some examples of SaaS are CRM, email, virtual desktops, and games [52].

However, many other flavors of services are provided, especially in industry, such as Container as a Service [106], Testing as a Service, Database as a Service, Security as a Service, and even Metadata as a Service [27]. All these types of services are sometimes grouped as to XaaS, where X is anything as a service [7, 108].

2.1.4 Cloud Providers

Low-level service delivery models such as infrastructure as a service require a large investment in properties, hardware, software, and operations. However, many technology giants such as Google, Amazon, and Microsoft already had these infrastructures in their data centers to be able to provide their services to their customers. The availability of these assets gave those companies an edge over other companies, by leveraging their current investments and thus getting a higher market share. In addition, this was one of the main enablers of Cloud computing as discussed in Section 2.5. The companies that provide low-level services mainly related to hardware are called cloud providers. Figure 2.6 shows the list of top cloud providers in 2018, by RightScale [107].

Being a cloud provider can bring many benefits. Mainly, depending on the economies of scale concept, long-term financial stability and high revenue are possible. In fact, based on the dynamics of innovations theory by Utterback [122], being part of the innovation will give the chance to dominate the market. On the other hand, new innovations may, directly and indirectly, affect other businesses. For ex-

Area	AWS	Azure	Google	IBM
% Adoption	68%	58%	19%	15%
YoY Growth in Adoption	15%	35%	26%	50%
% Adoption in Beginners	47%	49%	18%	14%
% with Footprint >50 VMs	58%	44%	17%	14%
YoY Growth in Footprint > 50 VMs	14%	38%	42%	56%

AWS leads
 Other vendors lead AWS

Source: RightScale 2018 State of the Cloud Report

Figure 2.6: Top cloud services providers [107]

ample, after offering the cloud infrastructure for a fraction of the price, hardware sales started to drop dramatically. In fact, this has forced hardware manufacturers such as Dell and HP to collaborate with research labs on cloud computing and start investing in this field to catch the wave.

Choosing a data center's location to provide low-level cloud services depends on many factors. For example, choosing locations with low-price property prices, low labor cost, and lower taxes will ensure offering the service at more competitive prices, while leveraging higher profit margins. On the other hand, to ensure the reliability of service, location selection criteria should include the quality of infrastructure and utilities, such as Internet speed and electricity reliability.

However, based on the physical theories, shipping photons over fibers optics is cheaper than shipping electricity [7], and cooling is still a challenge and consumes most of the electricity cost. Many pieces of research are being conducted on this issue, for example, Google has applied for a patent for a water-based data center. Their patent application shows data centers on large ships utilizing the sea motion and water in both electricity generation and cooling [112]. In addition, setting up the data centers in cold areas might be an option; however, infrastructure availability may be a limitation. Moreover, current data center's design for modularity, power,

and cooling requires new innovations [56].

2.1.5 General Benefits of Cloud Computing

Cloud computing provides benefits for both service providers and service customers.

A general benefit for service providers is to utilize the economies of scale concept, which means having a large customer base with recurring revenue.

From the IaaS perspective, as discussed in Section 2.6, the providers can gain many benefits such as leverage current investments and defend their franchise. From PaaS perspective, PaaS providers will have higher opportunities of customers lock in, which will reduce the risk of customers turn-over.

A general benefit for service users is the reduction in the overall cost, and delegation of liabilities to service providers [52]. Moving from CapEx to OpEx will get better tax benefits as well as reducing the operational and administration costs. Liability delegation is achieved by transferring risks such as security, availability (e.g. DDos attack), and legal liabilities to service providers. In fact, the cost of protecting the cloud is less than the cost of attacking it, since the attackers will require huge resources and bandwidth to be able to attack a cloud data center, which results in not being feasible in most of the cases for the attackers. In addition, it is possible to reduce the risk of procuring a service that does not achieve the business goals by leveraging the test-before-you-buy flexibility, since most of the cloud services are coming with no long-term commitment.

Furthermore, cloud computing can protect against the risk of load mis-estimation. In fact, the elasticity provided by cloud computing enables efficient scalability, which can make the service more reliable since the demand of service may be affected by many uncontrolled events such as news and holidays. Additionally, with cloud computing, there is no need for a large capital investment in hardware or operations, especially for startup companies. Figure 2.7 shows the difference between automated elasticity in scalability conditions compared to traditional non-automated solutions, and how automated elasticity can almost reach the actual demand without consuming

higher resources.

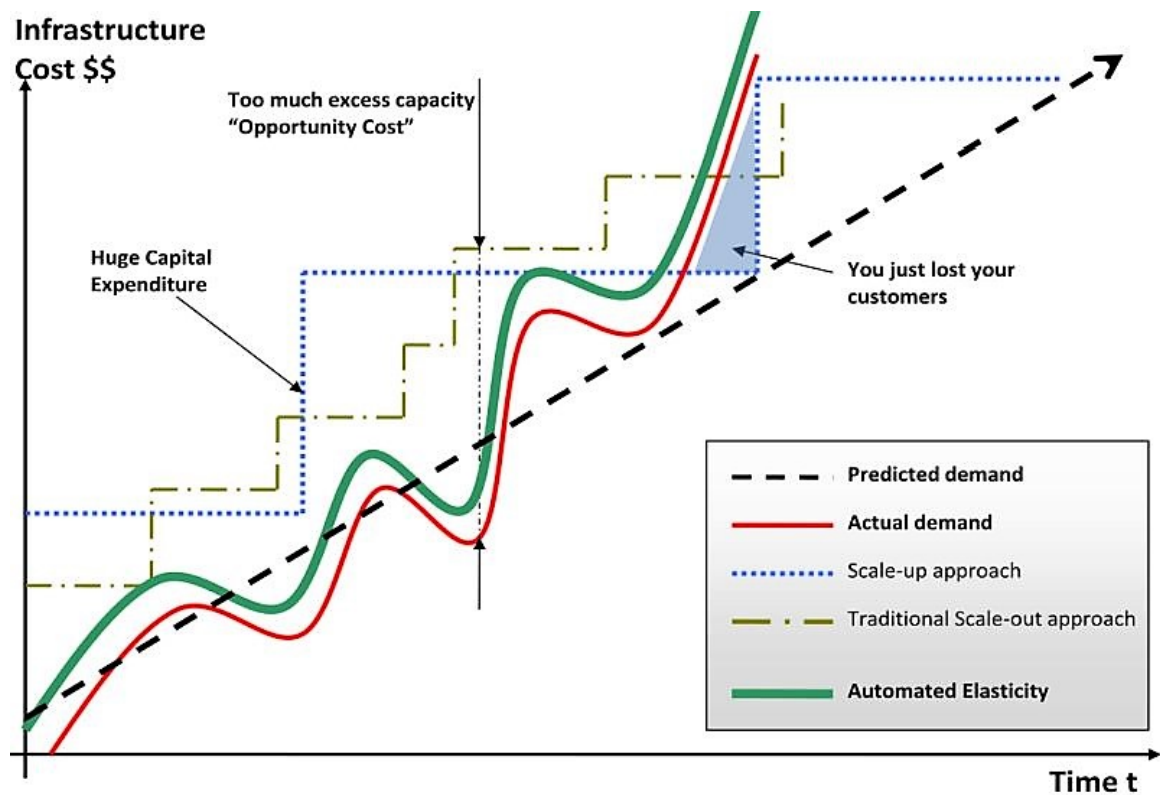


Figure 2.7: Automated vs traditional applications scalability comparisons [124]

From an operational perspective, cloud computing can benefit service users by enabling them to access the service anytime anywhere. In addition, collaboration and data sharing are easier. However, the risk of data security is discussed in the challenges which are covered in Section 2.1.6. In addition, having more control over servers and installed applications, where 30% of on-premise servers for an enterprise have applications that nobody knows about, and they follow the rule of “Let’s pull the plug and see who calls” [112].

From a business perspective, since the largest cost for enterprises is people, with about 1 employee for 100 servers [56], automated elasticity can also contribute to reducing the operational labor cost.

2.1.6 Challenges, Barriers and Risks

Even though cloud computing has many advantages, it comes with challenges and barriers. One of the disadvantages is that it is a single point of failure, where most of the cloud providers use the same infrastructure and the same software, in which any introduced bugs or issues can affect the all service users. Obviously, bugs in large-scale distributed systems used by cloud providers are hard to debug and fix. Another disadvantage is performance unpredictability, since shared virtualization resources and dynamic elasticity may affect the performance of other users. Moreover, the provision of new services is still not happening in real-time. Finally, technical up-to-date expertise in cloud technology in general and in cloud services management and administration in particular, is relatively not easy to find [58, 106].

One of the main challenges of cloud computing is data transfer, which is still a bottleneck. For example, moving large datasets is still cheaper and faster with traditional mail service such as FedEx. Moreover, another challenge is internal organizational and business policies, which enforce usage of internal data centers.

In addition, many customers have concerns in regards to security [24], data lock in, data confidentiality [45], data auditability, and control loss [52]. In fact, data can be exposed in multiple scenarios, it can be exposed during upload, while in the cloud, as well as during backup and restore [67].

From a business perspective, pricing uncertainty and cost model complexity, may not fit with some organization's policies such as government. Fate sharing between service users and service providers is becoming more common, where a failure in the cloud provider's service may affect the reputation of service users as well [7]. In addition, since cloud services are based on heavy marketing campaigns, this may not be appropriate for some type of businesses such as some scientific disciplines. Furthermore, lowering consultancy revenue is another concern for some service providers.

Some broad license agreements are a critical risk, which may enable service providers to terminate the service any time for any reason, without any customers

communication or feedback [45].

Moreover, politics and regulations can affect the evolution of cloud computing. For example, Internet neutrality [119], political and governmental conflicts may cause a suspension of services. In addition, some compliance regulations may put some special constraints, such as the European Union Data Protection act [42], where all European customers data should be saved in data centers located in the European Union [52].

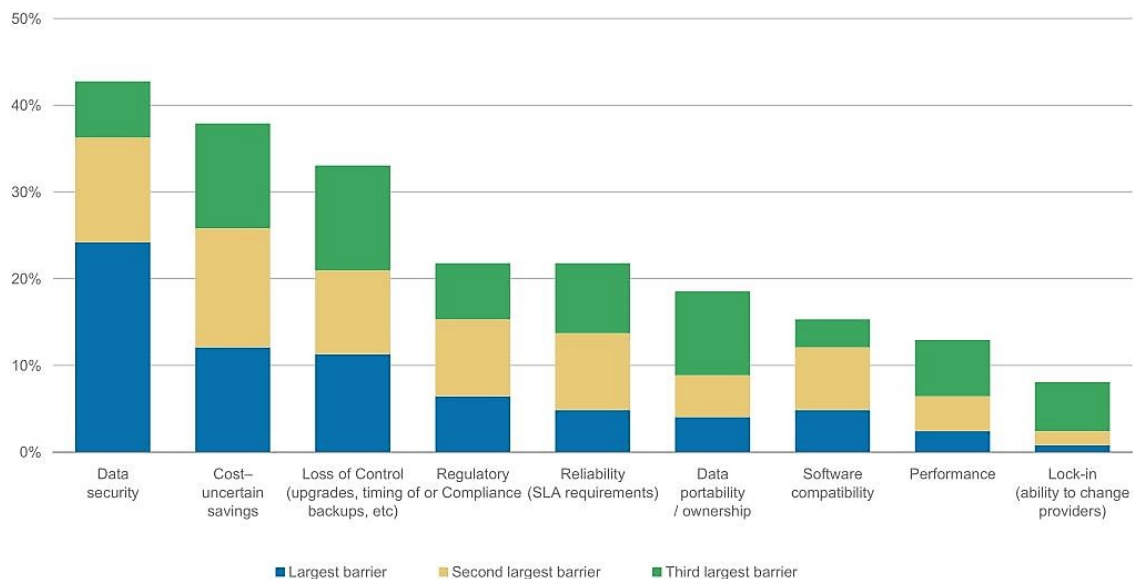
Another barrier may be the rejection from freeware and privacy advocates such as Richard Stallman, founder of the Free Software Foundation [43] and the creator of the computer operating system GNU [49]. In his interview with the Guardian in 2008 [55], he argued that cloud computing is a trap for enforcing people to buy proprietary licenses, and the main reason for its adoption is the marketing campaigns.

In addition, availability is still a challenge, as services issues such as service suspension by Amazon and Google [37, 38, 112] add some concerns. Furthermore, disasters may destroy infrastructure and interrupt service for days or perhaps weeks. On the other hand, service suspension may be caused by the unavailability of Internet access for some special places like underground halls or airplanes.

Figure 2.8 show the barriers ranking for cloud users based on a study done in 2011 by Morgan Stanley financial services firm [58]. Based on this study, data security is the main concern, followed by cost uncertainty, loss of control, regulatory or compliance requirements, reliability, data portability/ownership, software compatibility, performance, and finally lock-in. Even after 7 years from that report, and as shown in Figure 2.9, a report by RightScale published in 2018 shows many similarities in the challenges with the dramatic arise of *lack of resources and expertise* challenge [107].

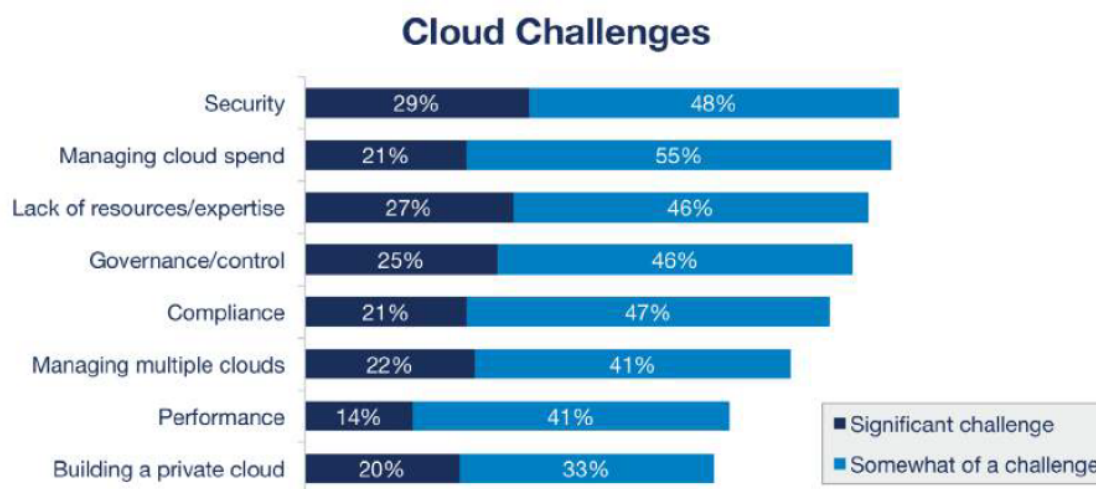
2.1.7 Impact on Hardware Business

Cloud computing has affected many businesses directly and indirectly. Computer hardware industry has directly been affected in many aspects. In particular, the behavior changes of customers toward OpEx instead of CapEx has affected hardware



Source: AlphaWiseSM, Morgan Stanley Research

Figure 2.8: Cloud computing barriers ranking by users in 2011 [58]



Source: RightScale 2018 State of the Cloud Report

Figure 2.9: Cloud computing challenges in 2018[107]

sales [58]. Customers now prefer virtual infrastructure instead of physical hardware for many reasons. Reasons include the illusion of infinite resources available on demand and eliminating upfront investments [7]. On the other hand, cloud provider's orders of hardware have increased and are sold by a scale of containers. For example,

about 2,500 servers were delivered by a 13-meter shipping container, then they were installed in a new Microsoft data center in Chicago and the center was up and running in only four days, including electricity and water supply for cooling and network setup [112].

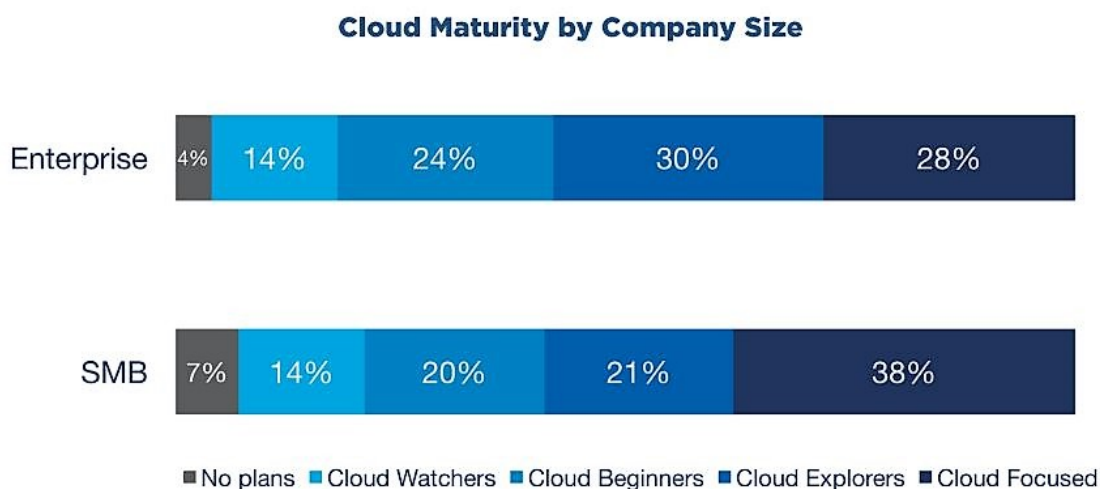
In the hardware labs, power saving features are now one of the leading topics, to reduce the operational cost of data centers. In addition, more compatibility and utilization with virtualization technology is required. Furthermore, improving communication technologies in both the routers and media connections is important.

2.1.8 Cloud Adoption

As discussed in the previous sections, cloud computing has many benefits that can reduce risks and increase profits. In fact, cloud computing utilization is highly recommended over private data centers in many scenarios, in particular when the demand of service varies with time. This allows a benefit from elasticity and ensures efficient utilization of resources. In addition, cloud computing is recommended when the demand of service is not known in advance, such as the growing demand of new service or needs of startup companies. Furthermore, cloud computing is preferred with batch analysis jobs, which most likely will get results faster. Moreover, running out of space for new data centers may be a strong motivation to adopt the cloud. Finally, resource limitations such as the inability to provide extra utility power for cooling [112] is another motivation to move to cloud computing.

Based on the cloud maturity model of Rightscale [106], cloud users are classified in four categories: (i) cloud watchers, (ii) cloud beginners, (iii) cloud explorers, and (iv) cloud focused. Cloud watchers still have not adopted any cloud technologies yet and are still in the evaluation phase; however, they are planning a cloud strategy. On the other hand, cloud beginners started to do some experiments for cloud services such as proof of concepts or running these services on a small scale. The third category is cloud explorers, consisting of the users who have adopted cloud computing in serious work for multiple projects, and they have the required expertise to use and manage

their cloud services. However, they are still exploring new opportunities to expand their business on the cloud. Finally, there are focused users, where the business is heavily based on cloud computing, and they work on cost and optimization for their cloud infrastructure. Figure 2.10 shows the cloud adoption percentage as of 2017.



Source: RightScale 2017 State of the Cloud Report

Figure 2.10: Cloud adoption as of 2017 [58]

2.2 Software Architecture

Bridging the gap between software system requirements and the final product is the main problem that the software engineering discipline tries to solve [47].

In any medium-to-large scale software system, the architecture of any system may dramatically affect its success or failure. In fact, developing good architecture can help achieve the project's overall success, and it can be disastrous if done incorrectly [46].

Software architecture is a set of structures which allow you to think about a system, which comprises software elements, relationships among them, and their properties [47]. Software architecture is significant for achieving the desired software quality attributes [15]. Quality attributes is a new term for non-functional requirements [113]. Most of the quality attributes are emergent properties (i.e., can be detected and arise only after putting the system into the final operational environment). Decompos-

ing the system into smaller components is a key activity in the architectural design process. In fact, it is significant for achieving important quality attributes such as modifiability, availability, reliability, scalability, and security [21]. Moreover, the architecture can define what changes and modifications can be made on the system and what can not be done.

Software architecture is used for many purposes. It can be used as part of the design process in the Software Development Life Cycle(SDLC) in the so-called Architectural Design [21]. In this case, the output of architectural design is an input for the construction phase. Moreover, architecture can be used for documenting software systems. In particular, diagrams and figures are easier and more efficient to communicate with other entities such as stakeholders. Also, it can be used for training and knowledge transfer. In addition, it can explain the rationale behind the chosen architectural and design decisions. Furthermore, it can be used to evaluate current systems and their qualities [15].

Even though software architecture is relatively not a new topic, it is still considered an immature field [46]. In particular, its implementation is still challenging for many reasons. Some of these reasons are expertise, high initial cost, and management awareness of the long-term value of implementing it.

The software architecture process can be classified into three categories: Informal, Semi-Formal, and Formal. Informal architecture is mainly having the minimum process, which includes architecture and design of the most significant components using non-standardized diagramming techniques (e.g., white-board sketches) with minimal documentation. On the other hand, the Formal has a complete process following a standard diagramming and documentation. Moreover, in between, is the Semi-Formal one.

Implementing software architecture appropriately in the SDLC can bring many benefits. It can enable a better understanding of the system and supports reusing the of system's components. Furthermore, it can enable more efficient and higher quality construction and development. Moreover, the evolution of software and software

management can be more effective.

2.3 Software Product Lines

Software product lines (SPL) is an approach in software development that can cut-down cost, increase quality, and reduce risks. It is based on building common assets that can be used for building software products that share common attributes and behaviors. In fact, SPL can reduce the software development cost for similar products for more than 90% [71]. In fact, as shown in Figure 2.11, SPL can reduce time to market dramatically after developing and maturing the core assets.

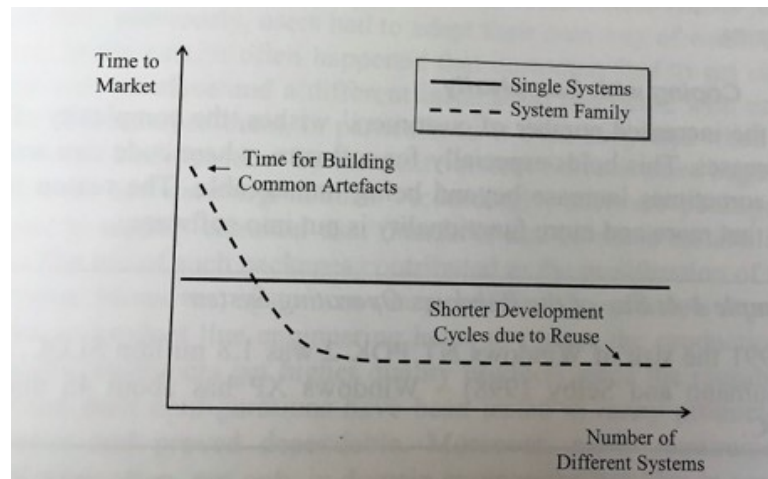


Figure 2.11: Software product lines and time-to-market [71]

SPL development process includes: defining the commonality across different software products to be developed by an organization, architecture, and development of these functionalities, and keeping the door open for customization through parameterization or inheritance.

An example of SPL is shown in Figure 2.12 built from Financial, Reporting Management, and Collaboration systems developed by the same organization. In this example, all the commonalities of the three systems are extracted into a separate project in what is called the Core Assets. While every project is developed by its own team, core assets are developed and maintained by a dedicated team, the Core

Team.

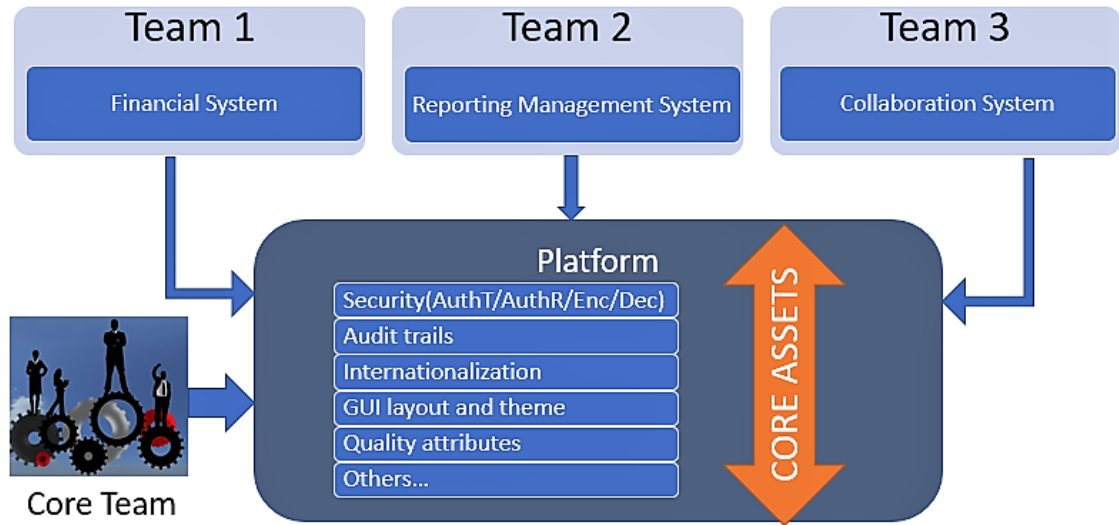


Figure 2.12: A Software Product Line example

The importance of SPL in software engineering is that it can support its main objectives of reducing the gap between the requirements and final deliverables while maintaining the quality attributes.

There are many challenges faced by researchers working in this field. First, there is a need for reducing the cost of collecting commonalities across common products to be developed in advance. In addition, there is a lack of architects' expertise in terms of who can analyze, design and develop such product lines. Moreover, there is a need for awareness by top management and stakeholders to invest in product-line asset development as a long-term investment, and how to reduce the time to market.

To standardize SPL and make it more implementable by organizations, SPL framework was presented by SEI (SPLFw) [76]. In particular, the framework covers many aspects of SPL, such as the value of adopting software product lines, best practices, recommendations, and challenges.

According to SPLFw, the SPL approach is “a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” [76]. Based on the framework, SPL development can pro-

duce many benefits from different perspectives, which include large-scale productivity, improved time to market, and reduced risk of unavailability of human resources, and the ability to have customization features relatively quickly and with lower cost while maintaining the quality attributes.

The framework acts as one-stop-shop for all required information about software product lines research. In fact, it is based on information from collaboration partners, surveys, and intensive-continuous research. In particular, it includes the purpose, definition, benefits, and costs of product line development. Moreover, it includes the essential activities, best practices, technical management, and organizational management.

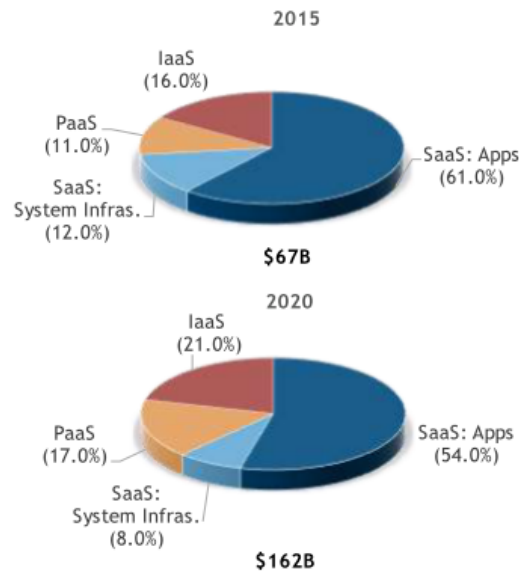
Chapter 3

Literature Review and Related Work

This chapter includes the literature review for the work presented in this dissertation. In particular, it includes an intensive review of cloud applications and the industry related work such as NetFlix and Cloud-Foundry.

Software as a Service (SaaS) is one of the service delivery models of cloud computing. In fact, SaaS is projected to have 54% spending share from cloud computing services by 2020 [66], as shown in Figure 3.1.

Worldwide Spending on Public Cloud Computing by Type (\$B)



Source: IDC, 2016

Figure 3.1: Cloud computing services worldwide spending [66]

As shown in Figure 3.2 software is a very generic concept, which can include operating systems, programming languages, tools, and applications [28]. Hence, these different categories of software can be part of any cloud service delivery model (i.e. IaaS, PaaS, and SaaS), in this dissertation Software as a Service term is used interchangeably with Cloud Applications. In fact, a discussion about a proposed new taxonomy for service delivery is presented in the discussion and future work section.

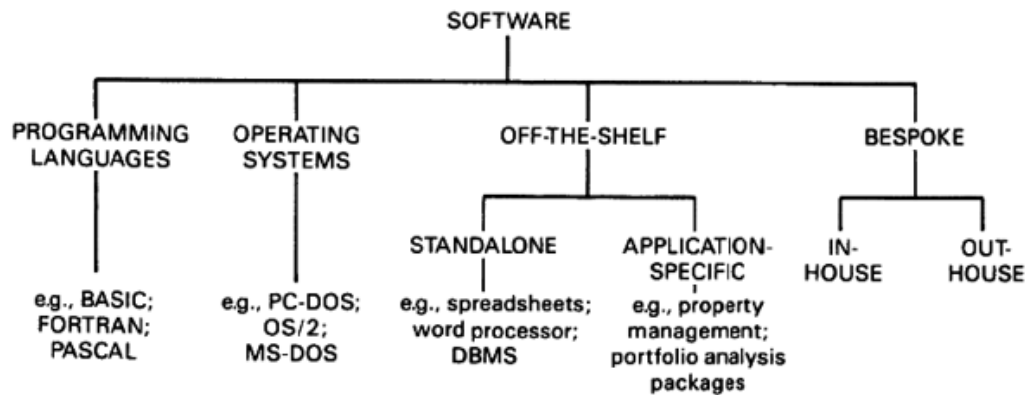


Figure 3.2: Software taxonomy [28]

Generally, cloud applications are defined as applications delivered over the Internet [7]. Cloud applications are most likely to run over PaaS (discussed in Section 2.1.3) [118]. A more detailed definition is the functionality provided over the Internet for a large group of clients, based on multi-tenant platform, with a single instance of software [54]. In fact, the multi-tenant support can also be provided at the application level. The design models of cloud applications are discussed in more details in Section 3.1. Section 3.2 presents the different types of clients that can access cloud services. Cloud applications are different from traditional on-premise applications in many aspects, and these differences are discussed in Section 3.3. The design and architecture of cloud applications are presented in Section 3.4. New patterns and techniques for developing cloud applications are required and give advantages over traditional monolithic applications, as discussed in Sections 3.4.3 and 3.4.4. Benefits of cloud applications are discussed in Section 3.7, and related challenges are presented

in Section 3.8. A strategy for moving to the cloud is presented in Section 3.9.

3.1 Design Models of Cloud Applications

Most Cloud applications are designed based on the economies-of-scale model, where many users use the same applications, in what is multi-tenant applications [19, 58, 72]. Cloud applications can be categorized based on their multi-tenancy nature into four design models:

1. **Single Instance Single Tenant (SIST):** In SIST, every tenant (i.e. client) has a dedicated separate instance of the application, including any files and database schema. In this case, billing and instance realization of the application are managed by the underlying platform or batch executions jobs. This separate instance for every client approach comes with the main benefits of development simplicity and reducing risks of data consistency. However, scalability cost and ability, operations and support for different clients in different instances. In addition, different versions for different clients can increase the probability of losing control on upgrades and features road-maps.
2. **Single Instance Multi Tenant (SIMT):** SIMT, is maybe the most popular model because it can gain higher benefits and lower operation and support costs. In this approach, application providers build one system that is used by all clients at the same time, support one version, and utilize fewer resources for operations. In this approach, all clients also use the same database or data repository, which can enable enhanced reporting and statistics for future decision making by service providers. On the other hand, building such application adds complexity to the software development process, debugging is harder and introduces common risks to all clients, such as quality of service for one client may affect other tenants, which can dramatically affect a service provider's reputation, and fate sharing [74]. In fact, it can reach the point where it can be a maintenance nightmare [19]. Generally, SIMT is appropriate for small to

medium business clients [54], and it can achieve economies of scale with extensive resource sharing. However, multi-tenant for mission and business -critical systems (e.g. core banking) is very risky, since the disadvantages of multi-tenant design discussed earlier may produce higher risks that are hard to recover from business and reputation perspectives. For example, if an upgrade caused a data corruption or inconsistency, it may result in producing invalid financial results and transactions, something that can not be forgiven in the financial industries.

3. **Multi Instance Single Tenant (MIST):** MIST maybe implemented for re-sellers or corporates where every subsidiary company or division is considered a separate entity as well as a separate customer.
4. **Multi Instance Multi Tenant (MIMT):** MIMT can be used as a mix of the previous design models. Figure 3.3 shows the differences between SIMT and MIMT.

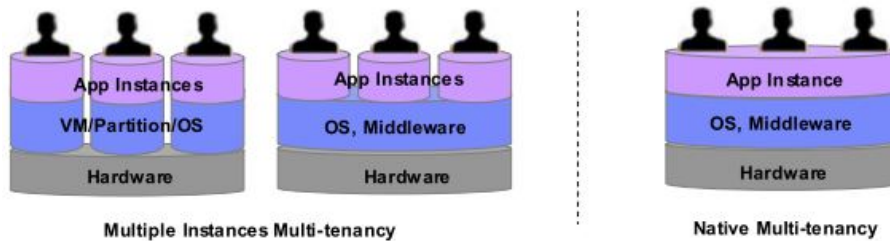


Figure 3.3: Tenancy-models in cloud applications [54]

The number of users, business-size (i.g. small, medium, or enterprise), and domain (i.e. government, telecommunications, financial, etc.) of customers are important factors to consider in designing a model to implement. If a low-number of large-enterprise customers are targeted, single SIST may be taken into consideration and can be profitable, since this type of a client can pay a higher subscription, and can not sacrifice or compromise quality, security, and risks. On the other hand, as application for a large-number of Small-Medium Enterprises (SME), SIMT may be

preferred [18]. Furthermore, the level of configurability and customizability has a significant impact on the chosen solution [84].

Obviously, a small number of users from small-size companies may not be appropriate for cloud applications to generate profits since the economies-of-scale factor is not available in this case. In addition, these categories of customers are not willing to pay higher prices compared to enterprise customers. However, it may work for non-profit organizations such as research-oriented and scientific applications if funded by organizations such as NSF [88].

Whenever the model is used, it should be fully transparent to the customers. Based on the framework presented by an IBM team in 2007, they proposed that the cloud applications should provide the isolation of (i) security, (ii) performance, (iii) availability, and (iv) administration [54].

3.2 Application's Clients

Most cloud applications consist of server-side components (i.e. application backend) and client-side components (e.g. front-end). Since the beginning of cloud evolution, HTML -Internet standard front-end technology- was the main used technology. Internet browsers have been the most widely used HTML clients to communicate with application backends. Desktop widgets became another form of clients [112]. Meanwhile, and with the rise of mobile devices and Internet of Things, new browsers, desktop applications, and sensors applications were added to the client's stack. Consequently, the complexity of building Internet-based applications has increased, and the need for more sophisticated front-end, a new term was introduced: Rich-Internet-Applications (RIA).

RIA is commonly based on utilizing client-side features that depend on JavaScript, HTML5, and CSS3. In particular, the Asynchronous JavaScript And XMLHttpRequest technology (AJAX) is the main enablers for modern interactive web-based applications. Even though it has been the common approach, the user-experience of browser-based applications was not sufficient for scaling cloud applications and could

not be used by non-technical people, which forced the way to native applications. In particular, native applications are developed using a programming language to build applications that can utilize native platform components. An example of native applications are applications built using the Java programming language to create Android platform apps. Another example is using Objective-C or Swift to build Apple IOS apps.

Another form of application clients is desktop widgets that communicate with back-end cloud services, such as weather or stock widgets [112].

The new trends of having different front-end clients require new architecture, design patterns, and tactics [57]. Design and Architecture of cloud applications are discussed in Section 3.4.

3.3 Cloud vs Traditional On-premise Applications

On-premise traditional applications are the software application instances that are designed to be installed on a client's environment (e.g. local data centers or computers) on the client's premises. The local installation of these applications includes all the application dependent artifacts and software systems, such as web servers, application servers, and databases. In on-premise applications, dedicated support for clients is provided, with different versions installed in every client's environment. In addition, there is no resource or access sharing with other clients. However, some form of integration with other systems, and external access to the client's services may be provided from that on-premise deployment.

The common licensing model for traditional applications is perpetual-licensing, where clients can use the software without any time limitations, and cost can be a one-time cost [23]. In this model, the cost can be accurately estimated from the beginning. In addition, in this approach, clients have almost full control over the applications and its data. On the other hand, cloud applications can be licensed on pay as you go or rental licensing models [77, 90, 125].

There are many disadvantages and challenges for on-premise applications. Firstly,

clients pay a relatively large amount of money for licensing compared to cloud applications. Secondly, on-premise applications require special upfront consulting and implementation costs [58]. In addition, long implementation time is one of the main risks. In particular, hardware procurement and installation, software environment configurations and setup, application deployment, and on-site implementation are all causing implementation delays. Moreover, most of the time support and upgrades are not included in the initial cost of the system.

Although cloud applications sound tempting, they are not fit for all types of applications [77]. For example, the traditional approach is more appropriate for real-time stock trading which requires microsecond precision [7], since performance is not guaranteed like on-premise deployment because of the sharing nature of cloud applications [58].

Adding to that, the on-premise approach can make a perfect sense in organizations working in sensitive domains, such as governmental or financial organizations. In fact, some domain compliance regulations and procedures require only the internal existence of their applications. For example, central banks in some countries enforce the core-banking to be locally installed and managed to ensure availability and data privacy.

On-premise software is normally built in a monolithic fashion. However, cloud applications are designed and built based on self-independent services [112], in what is called Microservice architecture [10]. A comparison between Monolithic vs Microservices architectures is discussed in Section 3.4.3.

3.4 Design and Architecture of Cloud Applications

In all categories of software engineering processes (i.e. waterfall, agile or component-reuse), the design is a significant phase [113]. Architecture is the core part of the design. Software architecture is an abstract, technology-neutral, representation of software systems elements, their relations and how they interact with each other. Moreover, architecture is important to deliver the quality attributes (i.e. non-functional

requirements) of software systems [15]. Furthermore, the architecture can be used as an input for development, documenting, and evaluating software [46].

This section discusses the evolution of cloud applications, starting from Monolithic applications, followed by Service Oriented Architecture (SOA), then Microservice architecture, to cloud native applications.

3.4.1 Monolithic Applications Architecture

Since the beginning of computer software development disciplines, building applications was mainly done using the monolithic approach. In particular, in the monolithic approach all the components of a software application are built as a single unit that should be compiled and deployed as a single instance on the edit-compile-link concept [120], and most likely with the same programming language or technology. Even though this type of software architecture is easier for software developers to understand, develop, deploy and operate, it has many disadvantages. These disadvantages include: full system compilation is required for any change, having all the teamwork on same technology or programming language, and harder horizontal scalability because of the application's heaviness. Another major issue with the monolithic architecture is that the system is a single point of failure, where a single error in the application can take the whole system down.

3.4.2 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an approach used to overcome some of the monolithic application's limitations [120]. In particular, SOA is about decomposing applications into smaller-unit (services) that integrate and are composed at runtime with each other using a standard protocol. Various XML based web-services protocols are used as standard protocols, including SOAP, WSDL, and UDDI [75]. Even though it was an elegant approach based on standard technologies, it didn't get high-traction from small-medium organizations because of its complexity, protocol overhead, and heaviness of the final full system. In addition, having remotely located services was

not practical since the units of most SOA applications were communicating with the same data stores (e.g. database), which caused many performance and reliability issues.

With all the limitations and issues discussed, new factors led to new requirements being needed. These factors include: (i) smart-phones and IoT require new lightweight yet interactive front-end technologies, (ii) entrepreneurship-wave and startup companies require faster time-to-market and lower development cost, (iii) cloud computing requires economies of scale models and scalability with minimal hardware and infrastructure cost. All these factors caused the innovation of many new technologies that have disrupted the software industry. Those new technologies have also created another issue of lack of human-resources.

On the other hand, there is a conflicted misunderstanding about the relationship between SOA and SaaS. In fact, SOA is a software construction model, while SaaS is a software delivery model [75, 118].

3.4.3 Microservices Architecture

To overcome all the constraints, limitations, and disadvantages of the monolithic approach and SOA, and to achieve the requirements enforced by the discussed new trends, Microservices architecture was innovated. As shown in Figure 3.4 the microservices architecture is a modern way of building cloud-based software applications, in which software applications are decomposed into small light components (services) that communicate with each other over light protocols and light messages exchange. The most commonly used protocol is Representational State Transfer (REST), which is a light-weight text-based solution over the HTTP protocol [36]. JavaScript Object Notation (JSON) is the common message format used by inter-services and service to front communications [26]. In Microservices, every service may be developed using different technology, must access its own data-store, and may not access any other service's data-stores directly, only over the exchange protocol. In fact, directly accessing other service's data-store directly will increase coupling and reduce the service's

portability. Moreover, these services are also independently deployable [41]. Also, horizontal scalability of Microservices is light and more efficient than other architectures. In fact, scalability is performed on the service level, where the service which has more load will be replicated on another instance, and there is no need to replicate the whole system. Application containers such as docker are the main enablers for this feature.

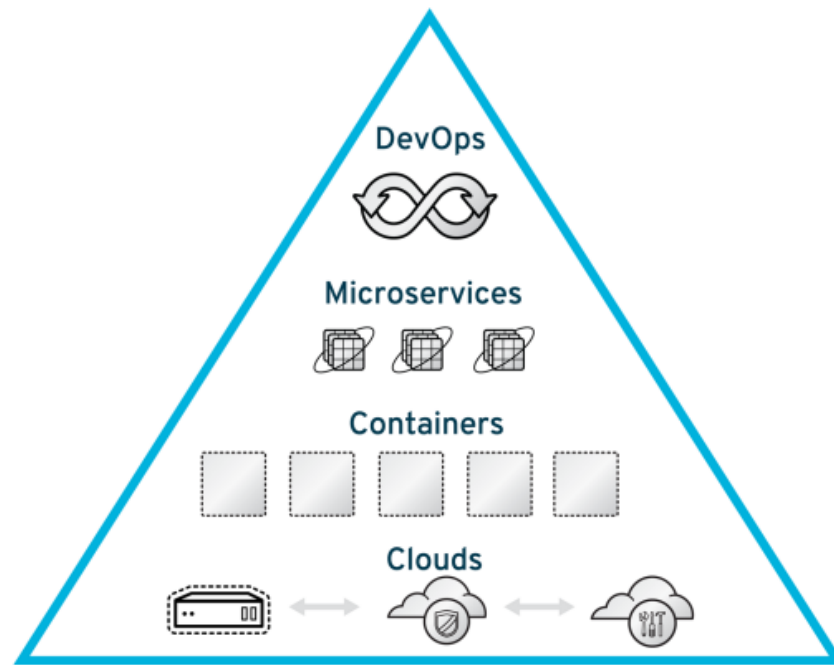


Figure 3.4: A pyramid of modern cloud native applications [31]

Furthermore, the microservices-based architecture can also come with other “dividends”, such as enabling innovations for developers, since they have full control over the design of their microservice, they can easily replace components and enable freedom of testing [68].

Even though the microservices style is similar to SOA in many aspects, such as decomposing software into smaller-deployable parts and communicating over a standard protocol [124], the lightness of communication based on REST, messages based on JSON, and separate data stores, might be the main differences.

Even though the microservices architecture solves many issues and problems, and creates potential for new opportunities, it introduces new complexity issues. In particular, special expertise is required to design and architect software solutions based on the microservices architecture. In addition, there is a complexity of integrating the services, testing, and deploy them. Moreover, monitoring and supporting services at runtime by the operations and support team is harder than supporting single processes applications such as the monolithic-based applications.

To reduce some of the risks of microservices architecture, intensive automation is required. In particular, automation can be achieved by applying automation software infrastructures such as Continuous Integration (CI), Continuous Delivery (CD), Test-Driven-Development (TDD), standard projects structure, and other [70].

3.4.4 Cloud-Native Applications

The complexity introduced by the microservices architecture is discussed in Section 3.4.3, has led to a new term: Cloud Native Applications (CNA). CNA are portable applications that exploit the full benefits of cloud computing without being dependent on a specific cloud provider or infrastructure [103]. Features such as services scalability, registry, binding, orchestration, and monitoring are supported out of the box. However, a platform is required to act as middle-ware and application server for those services.

In addition, as shown in Figure 3.5, cloud-native applications integrate well with CD, Microservices, DevOps, and Containers.

3.5 Main Characteristics of Cloud Applications

Cloud applications have special characteristics that make them different from traditional applications in many aspects [118]. These main characteristics are:

1. **Scalability up & down (Elasticity):** In traditional applications, the scalability requirement includes scaling-up, so that a system should be able to handle

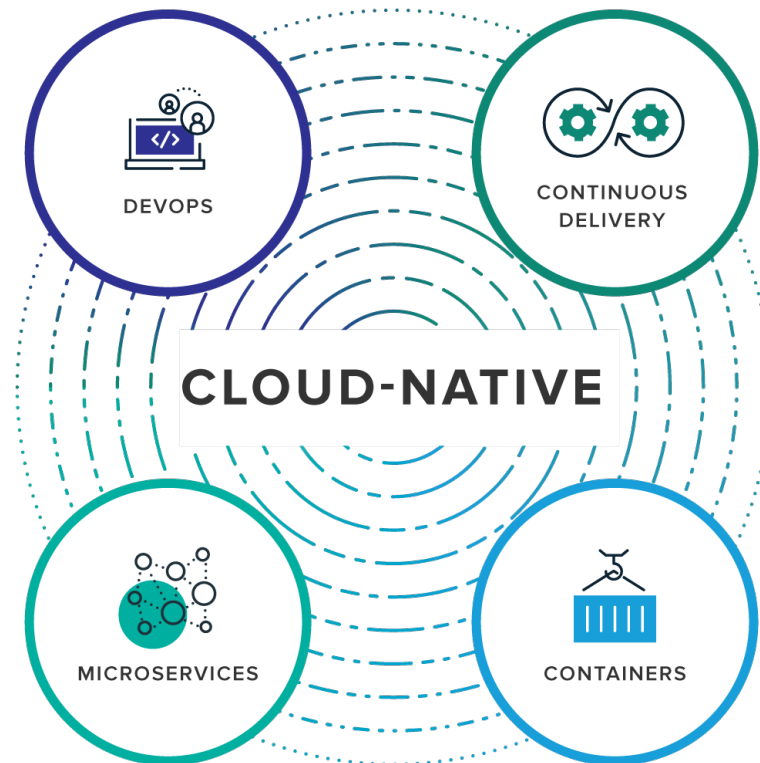


Figure 3.5: Cloud native applications external environment [103]

a larger number of users if required, without modifying the software’s code. This was normally achieved by vertical scalability [111]. In particular, vertical scalability is achieved by increasing systems resources, such as memory, storage or computing power [123]. On the other hand, horizontal scalability is widely used by enterprises, by adding extra nodes to the application cluster [38], but it is not common in small-medium organizations and businesses since it is relatively expensive, and not easy to configure and manage.

Even though scaling-up is important in cloud applications to utilize large data volumes and a vast list of services [46], scaling down is also significant, because it will minimize resource utilization, which reduces the cost for service users [7, 124]. In addition, horizontal scalability is lighter and more cost-effective than vertical-scalability. In fact, application containers such as Docker [62] are the main enablers for lighter and more cost efficient horizontal scalability. These new trends (horizontal scalability and application containerization) led to the in-

novation of a new architecture, microservices architecture, which was discussed in Section 3.4.3.

Designing an application for exact scalability needs is still a challenge, where having under-utilization, even with a small percentage, increases the cost more than actually needed, and over-utilization makes the services slower, and cause service users to look for alternatives [33]. Furthermore, in-advance testing and benchmarking of the cloud-application scalability can reduce the risk of downtime or load mis-estimation [44, 117].

2. **Support for different front-end technologies:** Currently, trends such as IoT and mobile devices have created the need for supporting widely different types of application clients (e.g. smart-phones, smart-cars, smart-televisions). A special design and architecture should be taken into consideration to support different types of front-ends without the need of modifying the back-end.
3. **Usage Metrics:** Since most cloud applications are based on subscriptions and pay-as-you-go models, usage metrics should be taken into consideration from the beginning since they will be the base for financial billing [7, 112].
4. **Monitoring:** Application monitoring is required to directly ensure that expected quality attributes are being met at runtime, especially in non-normal scalability conditions, such as holidays for e-commerce platforms, or breaking stock market news for real-time trading applications. It may include frequent health checks, heart-beats, and resources visualizations.
5. **Offline support:** Even though Internet services have become more accessible and reliable over the years, customers still have access difficulties to the Internet in many locations and places (e.g. airplanes, underground floors, trains). In addition, with the rise of IoT, scientific devices and sensors may be deployed in some remote locations (e.g. deserts, mountains, oceans), which also may not have an available or reliable Internet connection. Consequently, an offline

support feature is important. Having this feature gives service users the ability to use the service while disconnected from the Internet, which can be synchronized -once re-connected- later with the server. The offline support feature is critical for many applications, such as word editing tools, projects management applications, and IoT devices.

6. **Configurability:** In multi-tenant cloud applications, clients should have the illusion of separate application instances, while service providers may, and most likely should, maintain single instance to be able to maintain only one version and to achieve economies of scale. Designing the applications to be configurable and parameterized at runtime is important. In fact, having the quality attribute of configurability can reduce support cost, and give more customization and preference features for clients, which can increase customer traction and reduce their turn-over [75]. Furthermore, variability modeling from Software Product Lines (SPL) [71] can also be implemented to achieve the configurability quality attribute [83][85].
7. **Data locality:** The decision on whether to pull or keep data on the cloud requires special attention and balance between performance, data transfer cost, and usability. In particular, data locality is important to improve performance. For example, keeping data on a server may be efficient for server-side processing (e.g. search, filters), however, it might be more efficient to pull data to client-side for visualization applications. Nevertheless, in general, data-locality can achieve better usability and processing performance [53].
8. **Quality of Service:** Finding a way for separating the quality of service for multi-tenant services is important to ensure a reliable service and the separation of the shared-fate issue discussed in Section 2.1.6.

The dynamic nature of cloud computing and the difference between physical environments and virtualized cloud environments plays an important role in distinguishing between the architectures of traditional and cloud applications [124].

3.6 DevOps and Cloud Applications Development Process

Developing and managing cloud applications has caused a serious issue of mis-coordination between development and operation. In fact, the microservices architecture is a main reason for the increase in this issue, as discussed in Section 3.4.3. To overcome this issue and conflicts a new term was created: DevOps. The main concept about DevOps is that “you built it, you run it”, where application/service developers are also responsible for supporting and maintaining their applications/services while in production [59] and reducing the friction that appears while in deployment and operation phases [9]. Another approach of DevOps is that the development and operation teams work closely with each other to reduce the gap and taking ownership of the project overall success [21]. In addition, this decreases the time between changing a system and reflecting that change into the live environment [16]. As of 2018, 84% of enterprises are adopting this approach. In fact, 30% of these companies implement this approach on a company-wide policy [106].

In reference to the high diversity of roles involved in cloud application development (e.g. security, networks, business), DevOp has four main perspectives: (i) culture of collaboration where all team members from the different project life cycle stages have the required knowledge about the project, (ii) automation, continuous delivery, and deployment pipelines, (iii) high-level and accessible measurement and metrics, and (iv) and sharing of knowledge, development, tools, techniques and other aspects that can enable the required understanding for the system [60, 61]. Moreover, the knowledge, skills, and ability used in developing modern web-based applications were discussed in what is called “grounded theory” [12]. Figure 3.6 shows how DevOps changed the traditional structure of software development teams.

Moreover, the cloud has changed the role of the System Administrator to a Virtual System Administrator, where there is no need for any cabling/wiring, or server installation required or any other manual activity, it is all now done through

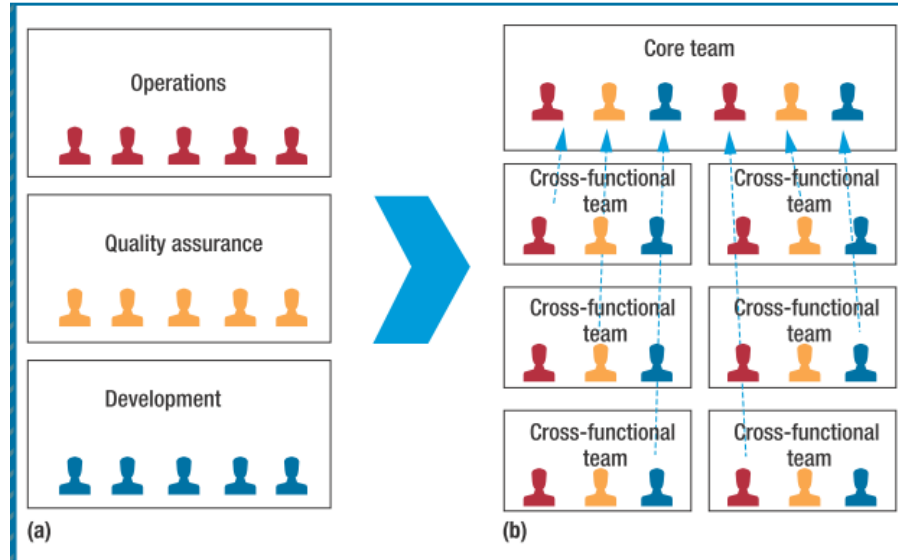


Figure 3.6: An example of DevOps team structure [16]

a web console that enables the network and the system to be administrated and managed virtually. This led to software developers and systems administrators being more collaborative and having more interaction [124].

Since DevOps is considered a culmination for what the agile method started [87], with both encouraging running software over writing-documentation [113], the main disadvantages of applying this approach is the risk produced when the developer leaves and not enough documentation is available.

3.7 Benefits of Cloud Applications

The general benefits of cloud computing were discussed in Section 2.1.5, which also apply to cloud applications. However, in this section, benefits of cloud applications, in particular, are presented for both service providers and service users.

Common advantages are the almost zero upfront investment, just-in-time infrastructure, and reduced time to market [124].

For service providers, running single versions will simplify maintenance, and lower customer support consequently reduces the cost of operations; also, it can reduce the research cost. Furthermore, this will give the ability for small non-risky updates.

From an operational perspective, service installation and deployment are easier, especially when utilizing the appropriate software infrastructure [70]. On the other hand, and from an economic perspective, since organizations are not willing to pay a large amount of money for software anymore [112], providing applications on the cloud will enable application providers to take advantage of this change in customer behavior to make more traction and profit. Moreover, software piracy is impossible, which is another major advantage for service providers [90].

The utilization of cloud applications can bring many advantages to service users. Firstly, low cost may be the first important factor. Secondly, data security may be better than on-premise applications especially in small businesses, where most likely there is no dedicated support team to operate and support on-premise applications. Furthermore, in general, cloud applications have better quality than on-premise applications and you always have access to the latest stable version of the system [23]. In fact, SaaS providers should invest in building higher quality software to ensure the increase of customers retention [48, 79].

3.8 Challenges of Cloud Applications

Cloud applications have many advantages; however, it also introduces many challenges and issues. Debugging a cloud application is not as easy as traditional application debugging. In addition, the support of multi-tenant discussed in Section 3.1, and adopting cloud native applications properties discussed in Section 3.4.4 introduce an extra complexity for the application development, deployment and management [54].

Furthermore, even though building cloud applications based on economies of scale model sounds tempting, marketing cost is the main challenge for customer acquisition. In fact, in 2012, even though with 90K customers, and a revenue of \$2.3 billion per quarter, their profit margin for SalesForce was negative because of the high cost of sales and marketing to attract and keep customers [52].

Furthermore, the competition between service providers makes the customers more selective and the decision to switch to another service provider is easier than

ever. Increasing the service cancellation cost of customers may be a solution, however, customers will not continue if the service is poor, or may sacrifice that extra cost if they found better quality elsewhere, so working on the application quality is significant to reduce customer turnover [79]. In fact, service providers need 12 months subscription on average to cover the expenses of a single customer [48].

Moreover, data integration and interoperability are challenging and include many concerns. These concerns and challenges include difficulty in large data transmission, from both security and bandwidth perspectives; data integrity and support of transaction across the cloud; expensive data change detection; controlling data quality; and determining the original source of data [78].

3.9 Strategy for Moving to the Cloud and Customer's Motivation

Cloud applications are widely used; however, as discussed in Cloud Adoption in Section 2.1.8, cloud watchers and cloud explorers need a strategy to move to the cloud. A proposed migration strategy is discussed in this section.

Moving applications to the cloud may involve multiple phases: (i) migrating non-mission critical software such as CRM, payroll, and recruitment, (ii) transaction between buyers and suppliers such as procurement, logistics, and supply chain, (iii) business critical such as startup financial applications and software development, and (iv) software as a service which may include spam-filters, and integration with on-premise software. Characteristics of this type of applications are low data security and low privacy concerns, little integration, and customization.

Encouraging customers who use on-premise applications to move to the cloud may not be easy. So special pricing and campaigns may be a good technique to follow. Because of lean regulations and policies, small to medium businesses may be a better target for cloud applications. In addition, changing the attitude towards more and enhanced customer care and service is very important.

3.10 Industry-based Related Work

Companies such as NetFlix and Pivotal led the cloud application development portability by adopting microservices architecture. They created many open source projects, tools, and API's that have been widely used in the industry. For example, NetFlix created Eureka [86], which is currently used by Amazon Web Services (AWS) to locate services and support load balancing and failover. Other examples are Spring Boot and Spring Cloud created by Pivotal [102, 114]. For simple application development, Dropwizard is used to create high-performance, lightweight microservices and back-end applications that support configuration, metrics, logging and operational tools out of the box [29].

In regards to multi-tenancy support, it is most likely implemented in code like techniques such as Hibernate Interceptors [63]. However, this approach is generally designed for other purposes such as logging and auditing.

Some platforms such as Pivotal Cloud Foundry try to enable portable support for cloud applications across different IaaS providers [101]. These platforms provide many services out of the box, such as metrics and monitoring.

3.11 Related Work Discussion

Based on the intensive review presented in this dissertation, many limitations and issues surround web application development. Mainly, there is the requirement for special expertise to be able to architect, design and develop such applications. In addition, there is a long learning curve to understand the concepts, tools, and techniques especially by junior to mid-level developers. Moreover, features such as multi-tenant support, auditing, metrics, and services register and lookup are distributed across different tools, which may cause consistency and portability issues.

To the best of our knowledge, no comprehensive framework that enables easy to learn, rapid, and portable cloud application development is available. In addition, we could not find any approach in which the user-interface dynamic generation is

supported or adopted in developing microservice based information systems.

We think that the availability of a framework that can enable building high-quality cloud-based applications with a minimal learning curve will be beneficial for both industry and academia. In addition, utilizing the metadata driven dynamic user interface generation, especially in cloud-based information systems, in a portable way will help organizations move to the cloud faster, and will help reduce both development and operational costs.

Chapter 4

Framework Overview

This chapter includes an overview of the proposed framework, Smart-Cloud. It starts with a high-level overview, followed by the high-level architecture. Furthermore, the main features and characteristics are discussed. Finally, the software infrastructure, technology-stack, and tools that have been used are all presented.

4.1 High-level Overview

Smart-Cloud is an application development framework that enables rapid application development of data-intensive cloud-based applications. It could be adopted in both the monolithic or the microservices architectural-style based applications. The framework is designed as a set of APIs, components, services, and cloud-apps that integrate with each other to provide general features for data-intensive application's development in general, and cloud-based applications in particular.

The proposed services in the framework are designed to be served as both microservices or APIs. In particular, every service could be reused as an independent self-deployable unit that manages its own data storage, or could be served directly through API calls within the same process. The rationale behind this design decision, and as discussed in Sections 3.4.3 and 3.4.1, choosing between the microservices or the monolithic approaches depends on many factors and constraints, however, designing the applications based on only one of them is risky, since changing the style in an advanced phase in the project may require significant changes to the whole system.

In fact, a recommendation by Martin Fowler is to start by the monolithic architecture approach, then enhance to the microservices one. Based on his article, he says:

As I hear stories about teams using a microservices architecture, I've noticed a common pattern. 1- Almost all the successful microservice stories have started with a monolith that got too big and was broken up. 2- Almost all the cases where I've heard of a system that was built as a microservice system from scratch, it has ended up in serious trouble [39].

One of the significant features of Smart-Cloud that it enables rapid application development (RAD). RAD is achieved by utilizing the model-driven development approach combined with a metadata-driven one.

The adoption of the metadata-driven approach in Smart-Cloud to develop cloud-based data-intensive applications, where most of the user-interface and back-end functionalities are designed to manage application's data, is due to the repetitive nature of such applications. In particular, the process of developing data-management functionality is a straightforward process and most likely will be repeated across most of the application's functionalities, but for different set of data. That consumes a lot of time, and may cause consistency and reliability issues, especially for large-scale enterprise level applications. To overcome these issues, the metadata-driven approach is used, where a single metadata entry for an entity is used to generate the full artifacts required by the application without writing code. The generated artifacts follow the Model-View-Controller MVC design pattern. These artifacts include: the user-interface (view), model, and controller, along with the database script required for the persistence of the data to be managed.

However, the metadata-driven approach alone is not enough, since the management of these metadata is also a time-consuming process, and error-prone, and may cause another issue of the maintainability of these metadata. That is where the model-driven development approach comes.

In Smart-Cloud, the model-driven development approach was implemented to

enable efficient creation and management of the metadata required for artifacts generation. Where users will be able to generate single artifacts, end-to-end artifacts, or end-to-end full applications with few clicks. The model-driven approach in Smart-Cloud has been implemented under the umbrella of Clowiz platform, the cloud wizard.

Clowiz platform is a cloud-based platform that enable software engineers, full-stack software developers, and even non-programmers to create software artifacts or full applications without writing code. Clowiz consists of three apps: CodeGen, FeatureGen, and AppGen.

The CodeGen app is used by software developers to generate a single unit of artifacts, such as Java class, HTML page, or SQL create-table script. On the other hand, FeatureGen is designed to generate an end-to-end feature that includes all the MVC artifacts to make a page fully functional. Finally, AppGen, is designed to enable, theoretically, anybody with or without technical programming skills to develop cloud-based applications using cloud-based model-driven design tools.

4.2 High-level Architecture

This section presents the high-level architecture of the Smart-Cloud framework which is shown in Figure 4.1.

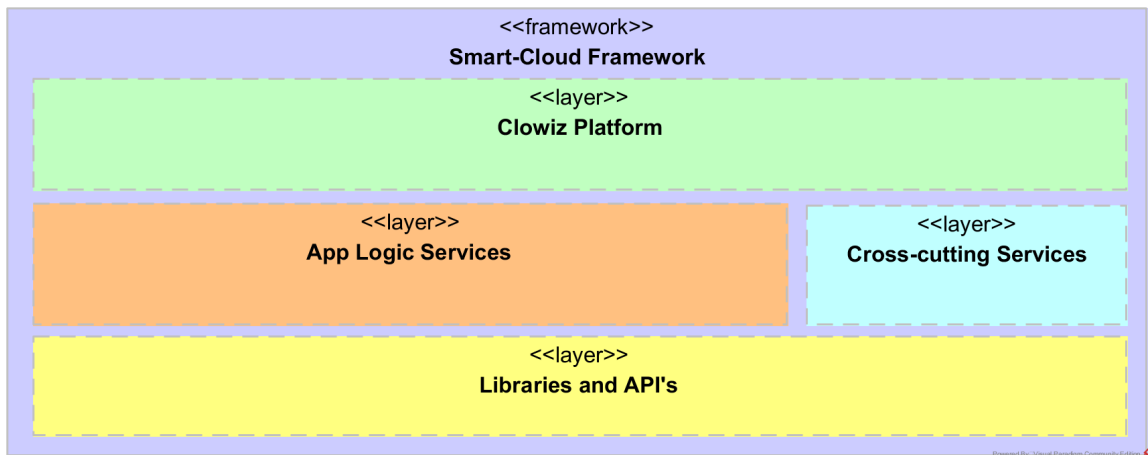


Figure 4.1: High-level architecture of Smart-Cloud

The framework consists of the following layers:

- **Libraries and API's:** This layer includes a set of reusable API's that are used by the other layers and components to reduce the development time, and increase software quality by providing a set of encapsulated and well-tested components, such as commons utilities, data access, web components, and metadata management functionalities.
- **Cross-Cutting Services:** These services include generic services that are re-used by the other application's services, such as security, account management, and logging.
- **App Logic Services:** These services include the services related to the core logic of the framework, which includes metadata and DevOps services.
- **Clowiz Platform:** This includes the apps that included Clowiz platform's apps, which are CodeGen, FeatureGen, and AppGen.

4.3 Significance

In Smart-Cloud, we aim to reduce the development time of building cloud-based applications. In addition, we aim to reduce the risk of failure of such application's projects by eliminating the dependency on highly skilled developers or software architects who have particular expertise in developing such applications. Moreover, we aim to give developers the ability to focus only on the domain business logic, by re-using the ready-made software API's, components, and services provided by Smart-Cloud out of the box. Furthermore, Smart-Cloud shall encourage organizations to consider migration from legacy on-premise information systems to native cloud applications early, in order to utilize the full benefits of cloud computing, increase work efficiency, and reduce the cost.

4.4 Features and Characteristics

Smart-Cloud has many features that can be beneficial for researchers and practitioners. These features are categorized into architecture, development, quality, user-experience, and operational.

From an architectural perspective, the proposed framework can be the base of various data-driven cloud-based applications. In fact, it can also be used for other categories of cloud applications. In addition, the reuse of the API's, components, and services from Smart-Cloud can enable more effective architecture by adopting the “don't reinvent the wheel” concept.

From a development perspective, focusing on programming the domain business logic only, by generating most of the required artifacts, and providing features out of the box, such as multi-tenant support and auditing will increase developers' productivity and reduce the overall cost of software development and the risk of failure.

On the other hand, having ready-made tested components that were built based on the test-driven development approach can ensure higher-software quality. Furthermore, the generation of the software artifacts using Clowiz platform, provide a higher consistency and centralized bug-fixing and features enhancements.

From a user-experience perspective, the consistency achieved by the generation of the user-interface enhances the user-experience for the users of developed-systems. In particular, having all the views appear in the same layout, same theme, and same validation using the same alerts messages enable a faster learning curve and more usage efficiency.

From an operational perspective, enabling auditing and logging features, additional to providing out-of-the-box dashboards, will support a real-time monitoring and actions from the operation team.

On the organization's wide point of view, Smart-Cloud enables building high-quality cloud-based software systems with a lower-cost and minimal resources.

4.5 Technology

Cutting-edge technologies, platforms, tools, and techniques were adopted in the development of Smart-Cloud framework. From technology point-of-view, the following have been used: JavaSE, JSF, PrimeFaces, SpringBoot, Spring Framework, JPA, Hibernate, HTML5, CSS3, and Bootstrap. Spring-Tools-Suite (STS) has been the used IDE, and Maven as the standard project model. MySQL has been used as the database management system. For diagramming and modeling, we used Visual-Paradigm community edition. For the version control and source code management, GitHub has been used. For the test-driven development, Mockito, JUnit, and Selenium were extensively used. The API's and components artifacts have been published to the Maven central repository, to enable easier dependency management and re-usability. Pivotal Cloud Foundry (PCF) was the selected platform to host the production version of Clowiz platform. PCF was chosen after extensive proofs-of-concept and testing on different hosting providers, which are Amazon Web Services, Heroku, and PCF. The primary reasons for choosing PCF are: ease of configuration and deployment, transparent scalability, pricing model, and the reliable integration with STS IDE. For Continuous Integration and Continuous Delivery (CI/CD), pipelines were implemented on Travis-CI online platform, to deploy to the production directly from the source code after a comprehensive pipeline that ensures full unit and integration testing on the staging environment, followed by the deployment to the production one. Finally, AsciiDoctor was used to write the online user-guide and user-manual of Clowiz, and TexStudio has been used to write this dissertation.

Chapter 5

Design, Architecture, and Implementation

In the previous chapter, we introduced Smart-Cloud framework, its high-level architecture, features and characteristics, significance, and the technologies and infrastructure adopted in all the phases of the implementation.

This chapter includes the medium and low-level architectural design of the framework. In particular, we present a comprehensive discussion about the Libraries, APIs, services, apps, their internal design, and how they compose the framework as a whole. UML diagrams are extensively used to model the architecture. Furthermore, we discuss the implementation, followed by samples of the framework user-interface.

5.1 Framework Architecture

The Smart-Cloud framework consists of different layers, where every layer composes the components that fall under the same category. As discussed in Section 4.2, these layers are *Libraries and APIs*, *Application Logic Services*, *Cross-cutting Services*, and *Clowiz Platform apps*. As shows in Figure 5.1, the Libraries and APIs layer consist of reusable components that are used across the whole framework. These libraries include util, data, web, metadata, and exporter API's. The Application Logic layer composes the services required for the core logic of the platform; these services are the Metadata and DevOps services. Cross-cutting Services layer includes the services that are utilized by the other services; these services include Security, Accounts man-

agement, Logging, and Notifications. Finally, Clowiz Platform layer includes the apps that enable the end-users of the platform to generate code, features, or full applications without writing code, using the metadata-driven combined with model-driven approaches.

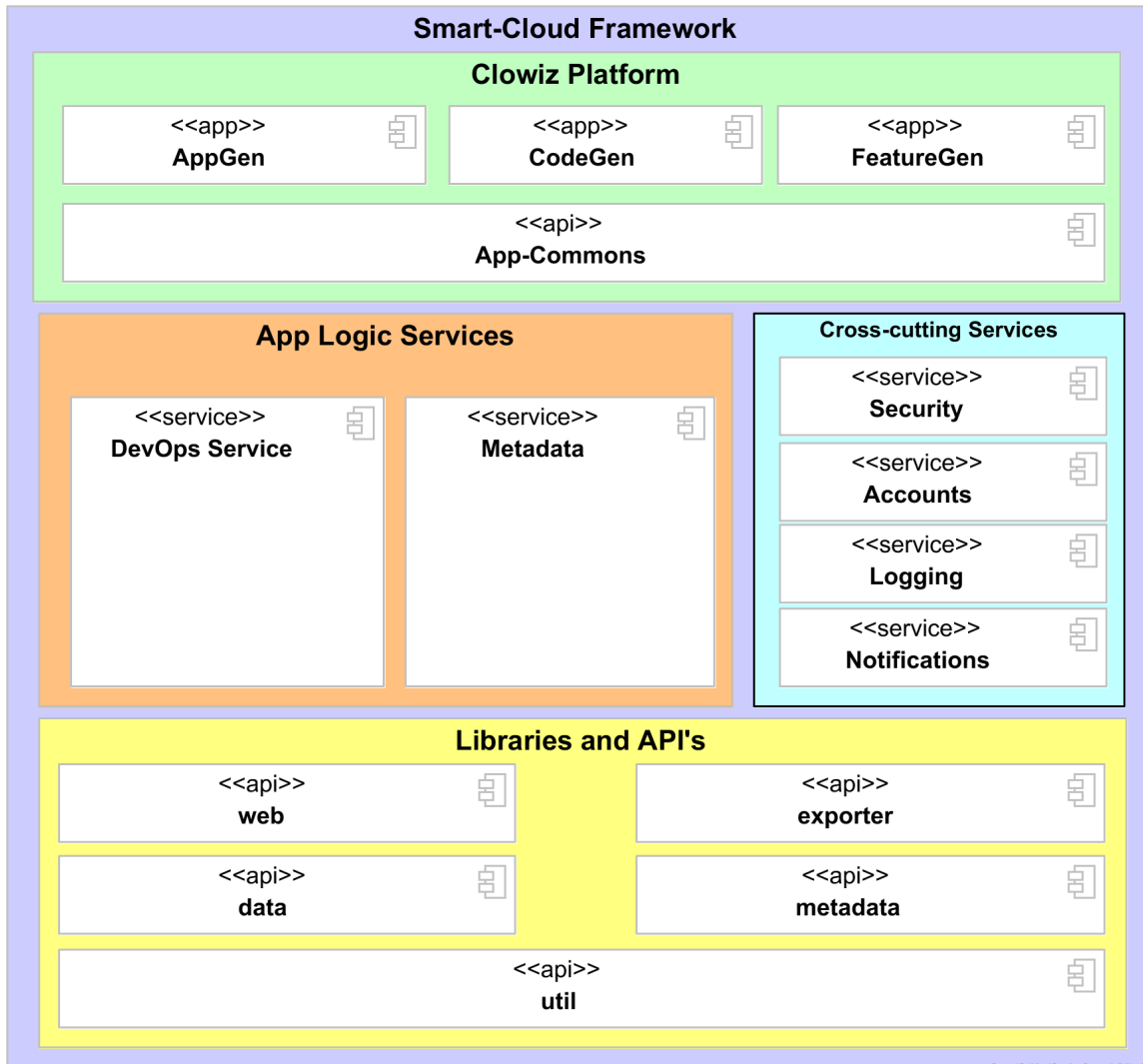


Figure 5.1: Medium-level architecture of Smart-Cloud

5.1.1 Libraries and APIs

This layer of the framework consists of reusable libraries that increase developers productivity. In addition, it reduces the risk of failure, which has been achieved by

implementing the test-driven development approach for these components.

Util Library

This library includes a set of generic-level reusable components, classes, and methods that could be reused in any application that uses the Java technology. It includes more 20 packages that serve different purposes such as compression, validation, templates, caching, security, mail, and testing. Most of these utilities provide functionalities through a single call to a single utility method. For example, compressing and uncompressing files are provided through simple calls to `public static void unzip(InputStream in, String destDirectory)` and `public static Path unzip(InputStream in)` respectively.

Data Library

This library is designed and implemented to enable easier and more efficient development of applications that require persistence and retrieval of data from a relational database. In particular, it includes utilities that support reliable and efficient support to plain SQL development inside Java applications (JDBC), and other utilities that support the Object Relational Mapping (ORM), where normal classes can be mapped directly with database tables without the need of writing SQL code inside Java code. This library solves common problems that face developers who develop such components. These problems include the absence of a common configuration file for both techniques and unified resources pooling; in addition, it solves the problem of resource management, where many developers forget or do not know how to release database resources properly.

Metadata API

This API includes the metadata API and its related components. This API is the core of the metadata-driven functionalities provided by the whole framework.

As shown in Figure 5.2, the metadata API consists of classes which represent the metadata required for generating the end-to-end artifacts of software applications.

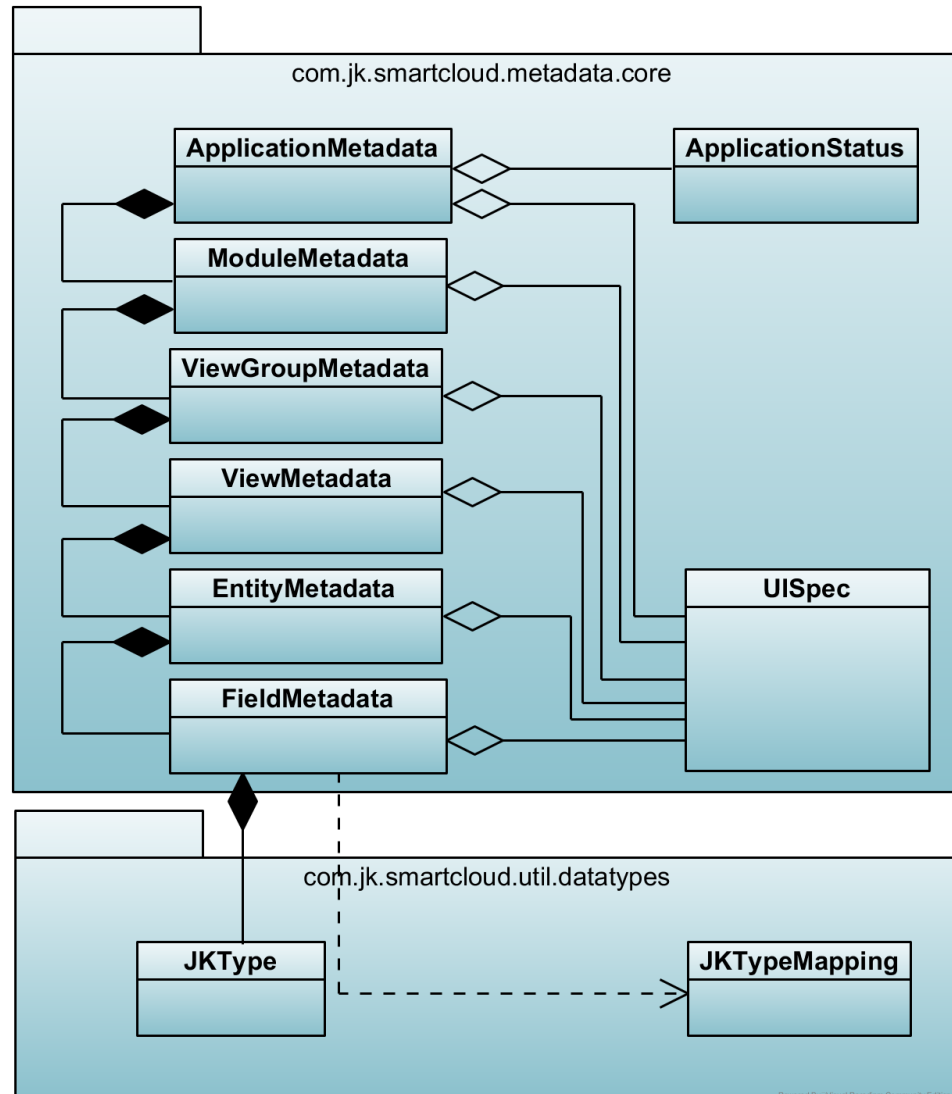


Figure 5.2: Metadata API Class Diagram

The description of the classes shown in the Metadata class diagram is shown in Table 5.1.

Exporter API

This API includes a set of classes that export metadata instances, which shall be created by the framework users using the metadata API, into the final software artifacts. The package diagram of this API is shown in Figure 5.3.

The *core* package in the API is shown in Figure 5.4. This package includes the

Table 5.1: Metadata API classes

Class	Description
<i>ApplicationMetadata</i>	This class contains the application level metadata such as name, status, home page, locale. Also, it is the root of the module's metadata. Instances of this metadata class are used to generate full applications and artifacts.
<i>ModuleMetadata</i>	Contains the metadata of a module (subsystem). A module is the container of view-groups metadata. This metadata is used to generate the tabbed modules menu in the generated applications. In addition, it is important in classifying and building the tree of artifacts to follow a convenient structure.
<i>ViewGroupMetadata</i>	View group metadata class contains metadata of a collection of views (e.g., pages). Instances of View groups are rendered as menu-bars in the final application generation.
<i>ViewMetadata</i>	This metadata includes an individual view metadata. Internally, it composes an entity metadata, that is required to generate the MVC artifacts for this view.
<i>EntityMetadata</i>	This metadata is general purposes metadata instance that could be exported various formats of source-code artifacts. These artifacts maybe views, controllers, models, database tables, and others. EntityMetadata consists of a list of metadata fields and Id-field.
<i>FieldMetadata</i>	This class contains the metadata for individual fields that included in an entity metadata. It consists of the field name, datatype, null-ability, and many others.
<i>UISpec</i>	This class includes the metadata required for user interface artifacts generation. This metadata includes width, height, background color, and foreground color.
<i>JKType</i>	This class encapsulates the information of datatype mapping for an individual datatype. For example, it may contain the mapping code for SQL datatype of varchar2, and the String class datatype in Java.
<i>JKTypeMapping</i>	This class includes the mapping registry of all the datatypes that are used with converting a datatype to another one in all the supported technologies in the framework.

primary classes of the exporters, such as `ApplicationExporter` and `EntityMetadataExporter` interfaces. These classes should be implemented to enable exporting the metadata into new applications for a specific technology stack, and their internal

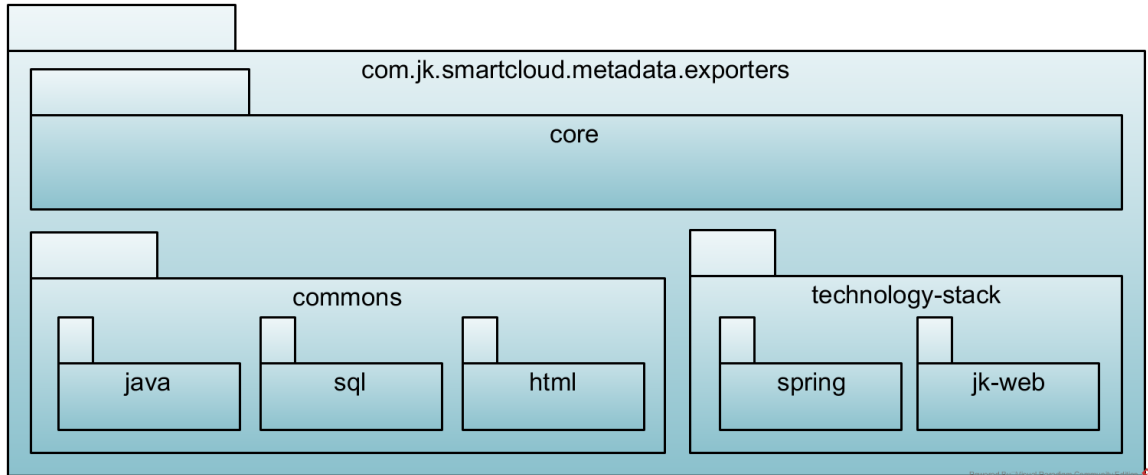


Figure 5.3: Exporter API Package Diagram

artifacts.

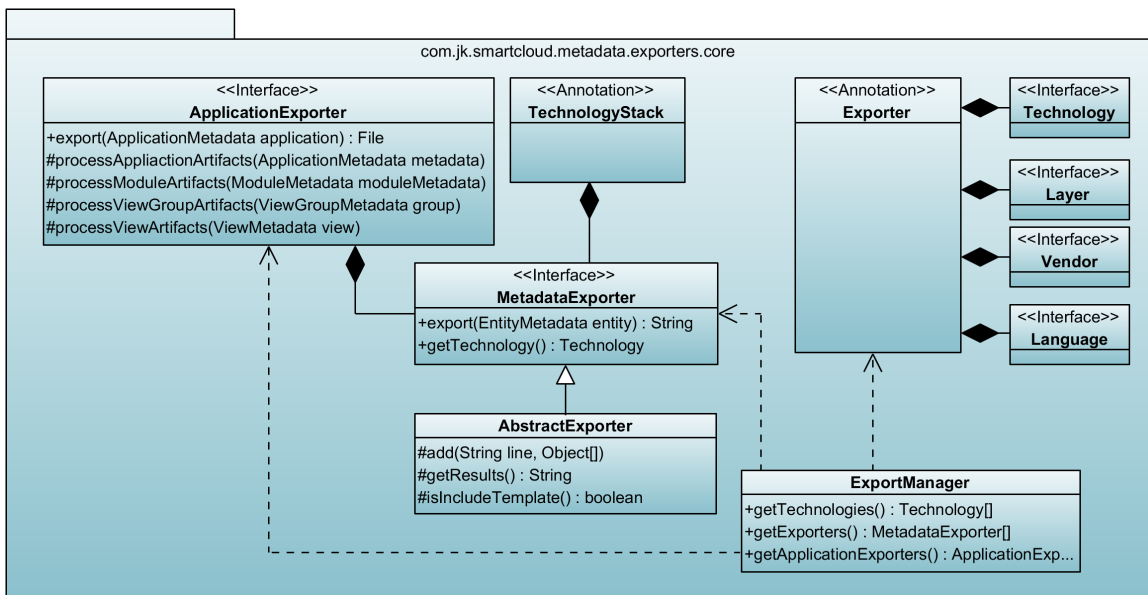


Figure 5.4: Exporter API Core Package Class Diagram

The *commons* package of this API includes the common exporters, that are shared in different technologies, and how they are related to the *core* package. The details of this package are shown in Figure 5.5. Every exporter in this package is used to generate individual software artifacts, such as Java class, HTML page, or SQL create script.

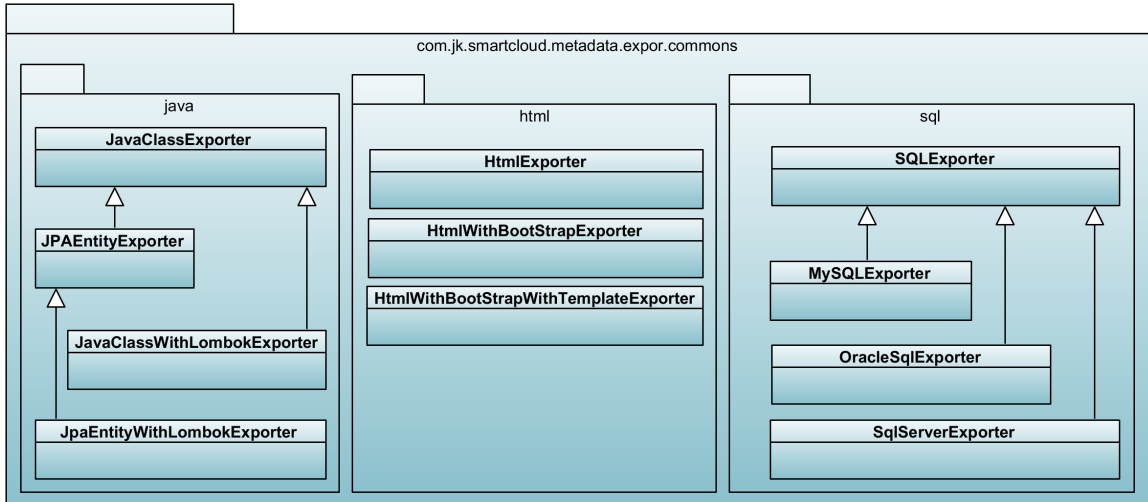


Figure 5.5: Exporter API Commons Package Class Diagram

As shown in Figure 5.6, the technologies package includes exporters that generate end-to-end applications for a specific technology stack. This package currently supports generating applications in JavaEE and Spring framework.

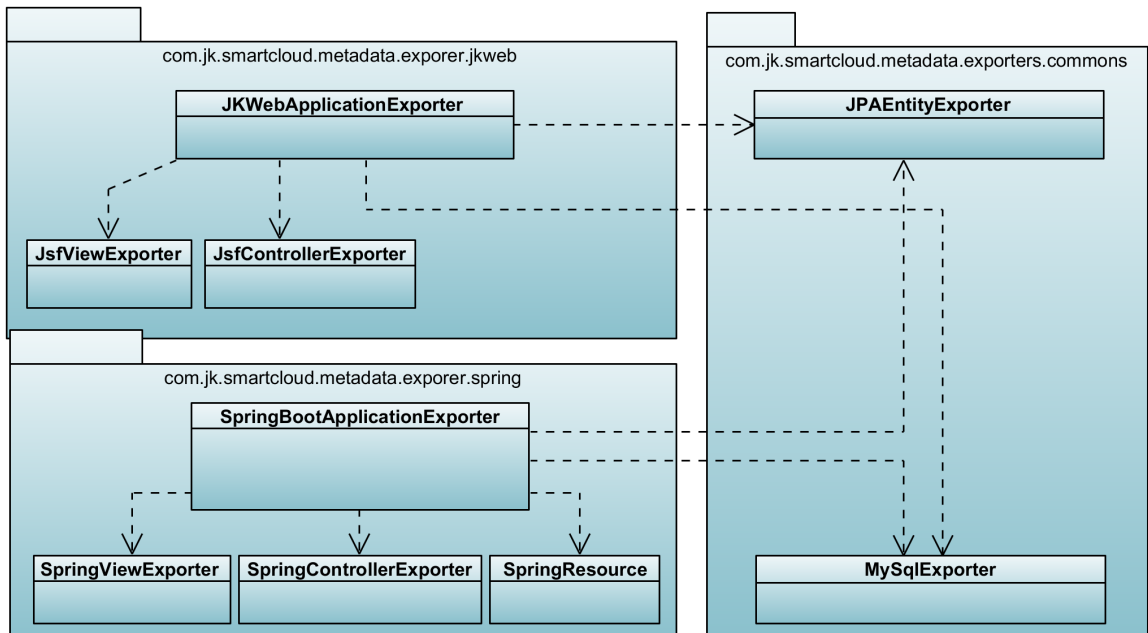


Figure 5.6: Exporter API Technologies Package Class Diagram

5.1.2 App Logic Services

App Logic Services layer consists of services that are required by the higher-level applications of the framework. These services include the Metadata and DevOps services.

Metadata Service

This service provides full management of all the categories of metadata, synchronizing it with a relational database engine, and exposing its functionalities either as a high-level API or a microservice. The details of this service are shown in Figure 5.7. To provide apps with the ability to use this service without persisting information to the database, `FakeMetadataService` class was designed, where it extends the `MetadataService` class, and override some methods to change its behavior into an in-memory data-structure instead of relational database persistence. This class is used in the unit testing and in the guest-mode support, where users can try the full functionality without the need of signing in or register to Clowiz platform.

DevOps service

This service includes the features that enable the generation and management of the applications created by the end users. In particular, it includes the functionality required to generate, download, or preview applications. The `DevOps` service class is shown in Figure 5.8. The service is currently used by the AppGen app.

5.1.3 Cross-cutting Services

This layer includes the services that are not coupled to a specific functionality or layer and could be reused across any other service in any layer. In addition, it could also be reused by other applications in other domains, since they are not domain-specific services. These services include security, accounts management, logging, and notifications.

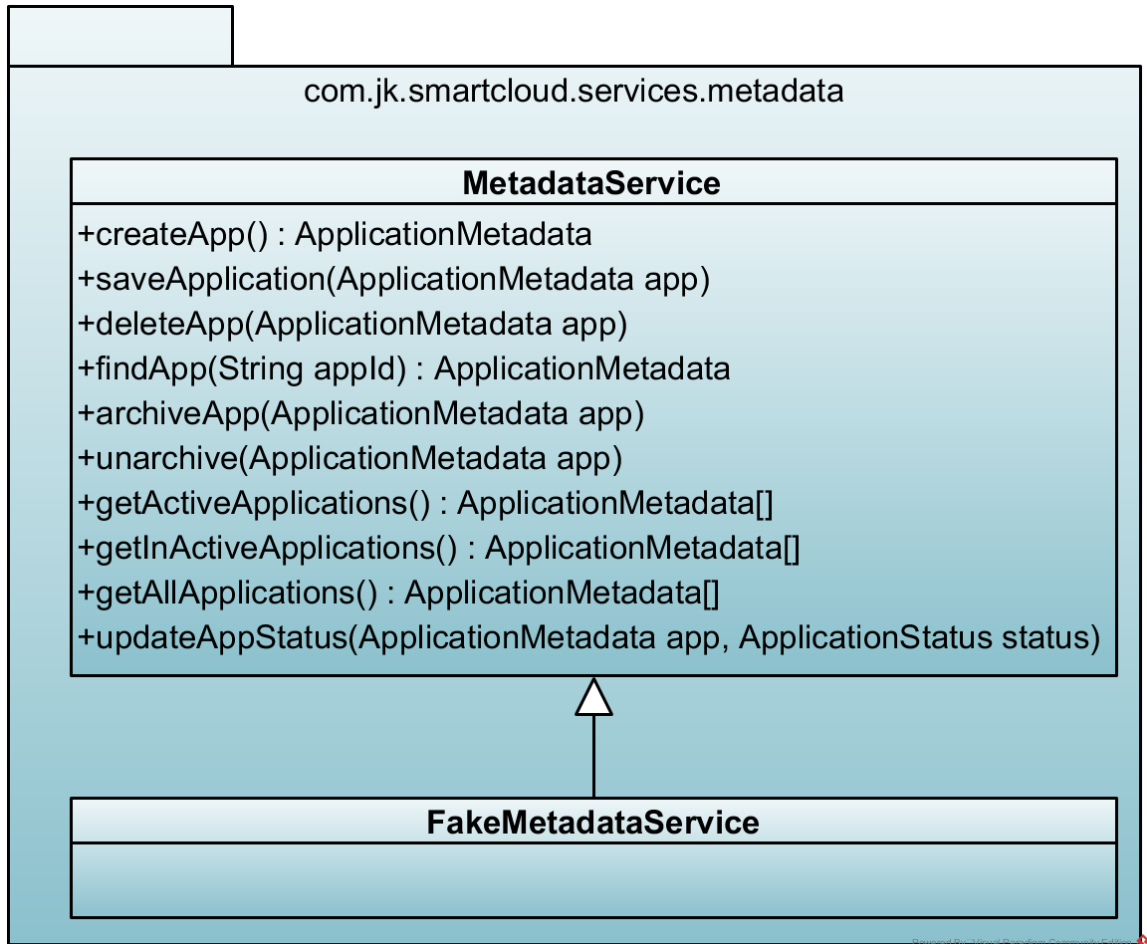


Figure 5.7: Metadata Service Class Diagram

Security Service

This service includes the security, authentication, and authorization functionalities. It also contains token management to enable authentication in different security contexts, and wrappers to the accounts service for the actual user's management.

Account Service

The account service includes features that enable users to create accounts, reset their passwords, and manage their information. The details of the accounts service are shown in Figure 5.9. In the current design of the framework, the accounts service is used by the security service, where every account represents an authenticated person

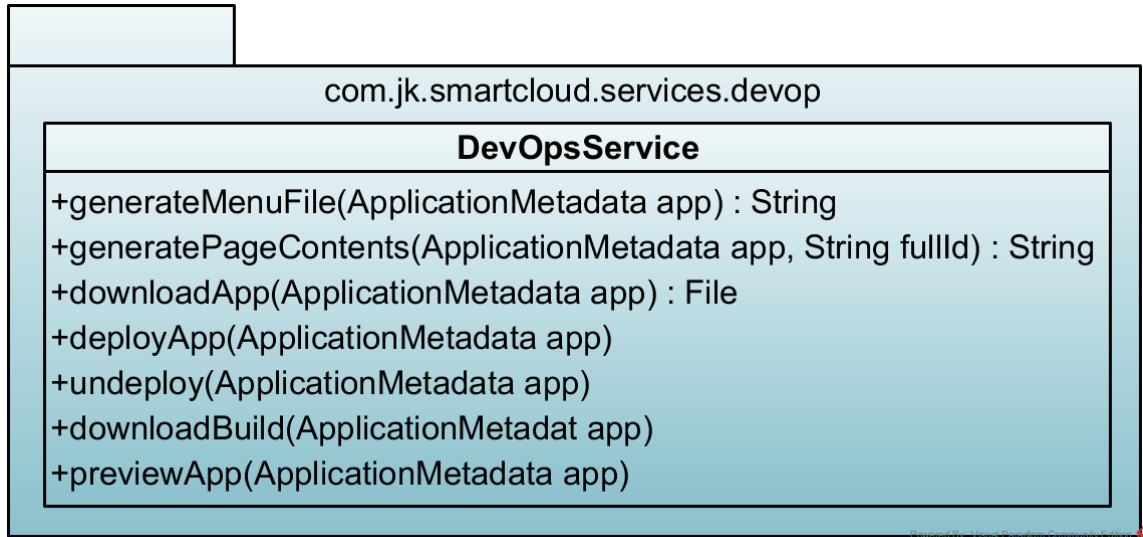


Figure 5.8: DevOps Service Class Diagram

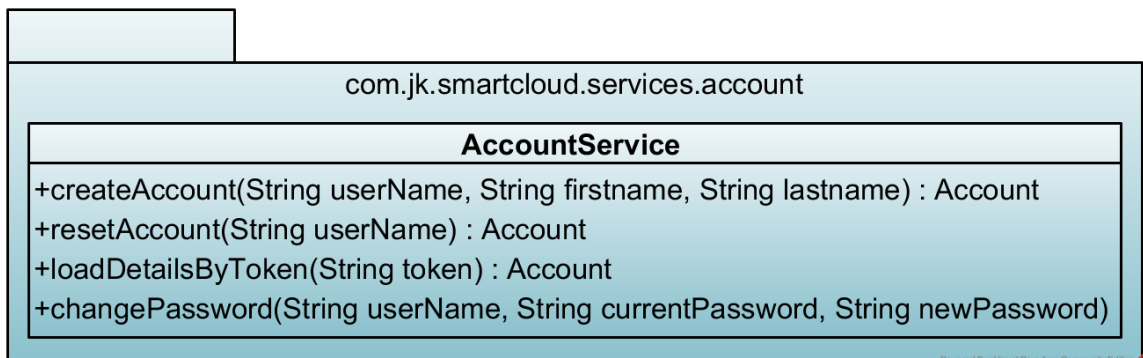


Figure 5.9: Metadata Service Class Diagram

who is allowed to access the system.

Logging Service

This service includes features that enable tracking and statistics of the system wide-usage. The information collected by this service will help to identify the users-behavior in using the apps of this framework.

Notification Service

The notification service currently provides email notifications for the users about their important activities working on the system. For example, it is used to send the

required authentication information for the end users when they sign-up or reset their passwords.

5.1.4 Platform Apps (Clowiz Platform)

To enable the end users to fully utilize the features of the framework in a user-friendly and an efficient way, model-driven development apps were created to enable metadata management, and software artifacts generation through a cloud-based web interface. These apps are CodeGen, FeatureGen, and AppGen.

CodeGen

CodeGen enables users to create source code for a single artifact. The app currently supports more than 15 generators that export the metadata into cutting-edge artifacts. These artifacts include Java classes, JPA Entities with Lombok support, full HTML pages with Bootstrap support, and full database script for some of the most popular database engines, Oracle, SQL Server, and MySQL. The list of generators supported by CodeGen are shown in Table 5.2.

FeatureGen

FeatureGen enables users to generate end-to-end features based on the MVC design pattern. The generated artifacts consist of the view, model, controller, and database script. The FeatureGen generates artifacts for the supported technology-stacks in the framework, which currently are JavaEE and Spring Framework.

AppGen

AppGen enables the user to create, design, preview, download, and deploy cloud-based applications without writing code. Users can fully design all the application's aspects as metadata. The process of building applications using AppGen is shown in the activity diagram shown in Figure 5.10.

Applications created using AppGen start with *In Development Phase* status. When a user deploys one of his/her apps, the app status will be updated to *Deployed*.

Table 5.2: The list of generators supported by CodeGen

Generator	Description
JPA Entity	Java class that has the Java Persistence API annotation for database mapping.
JPA with Lombok	Java class that has the Java Persistence API. In this generator, Lombok is used to reduce the size of generated code by avoiding the need of generating setters and getters, which are generated by Lombok processor at compile time.
JSF Managed Bean	Java class that contains Java Server Faces Controller functionality.
Java Class	Normal Java class with setters and getters.
Java Class with Lombok	Normal Java class without setters and getters, supported by the Lombok library.
PrimeFaces View Only	JSF view based on PrimeFaces widget components. This code doesn't include binding with back-end controller.
PrimeFaces with Binding	JSF view based on PrimeFaces widget components including back-end binding with a JSF controller.
HTML Full Page	Generates full HTML page including the commons css and JavaScript frameworks such as bootstrap, jquery, Google fonts, and fonts-awesome.
HTML Only	Generates HTML code only that contains input fields and labels for the designed metadata, however this doesn't include any styles.
HTML with Bootstrap	Generates HTML form with all input fields and labels. In addition, it includes Bootstrap framework styles.
H2 SQL Structure	Generates create table SQL statement for H2 database.
SQL Server Structure	Generates create table SQL statement for Microsoft SQL Server database.
Oracle SQL Structure	Generates create table SQL statement for Oracle database.
MySQL Structure	Generates create table SQL statement for MySQL database.

Users can un-deploy apps so that they will return to the *In Development Phase* status. Apps could also be archived, un-archived, and permanently deleted. The Full workflow is shown in Figure 5.11.

A full-detailed user-guide for the framework apps is provided in Appendix A.

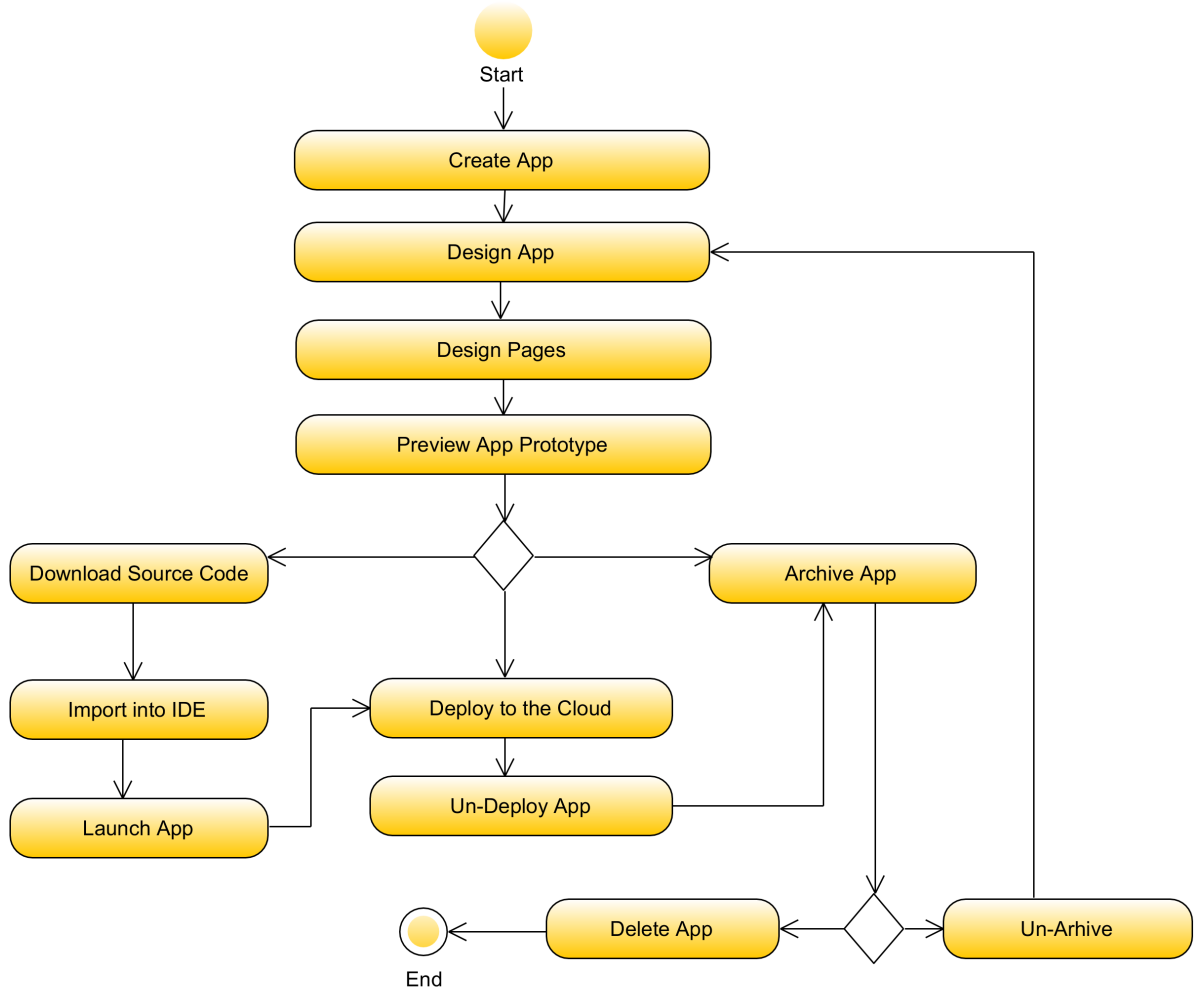


Figure 5.10: AppGen Development Process

5.1.5 Database Design

To enable users to have their work persisted, the different services are designed to communicate with a relational database through the *data* API in the *Libraries and APIs* layer. The current implementation uses MySQL as the database management system. However, any database that supports Java Database Connectivity (JDBC) specifications can be used without changing code. The complete Entity-Relationship Diagram (ERD) of Smart-Cloud framework is shown in Figure 5.12.

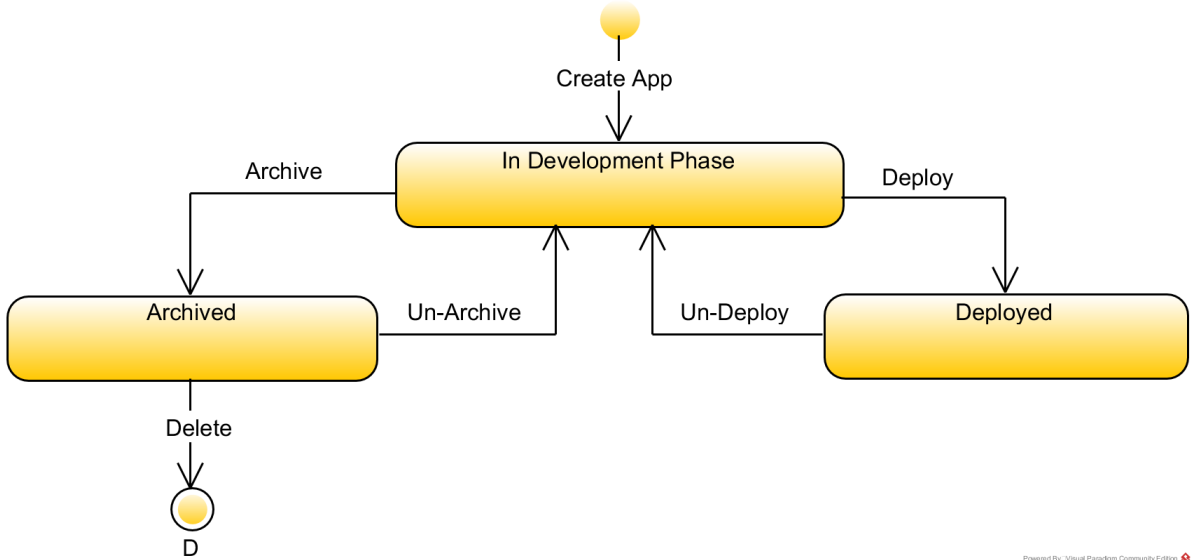


Figure 5.11: AppGen Status Workflow

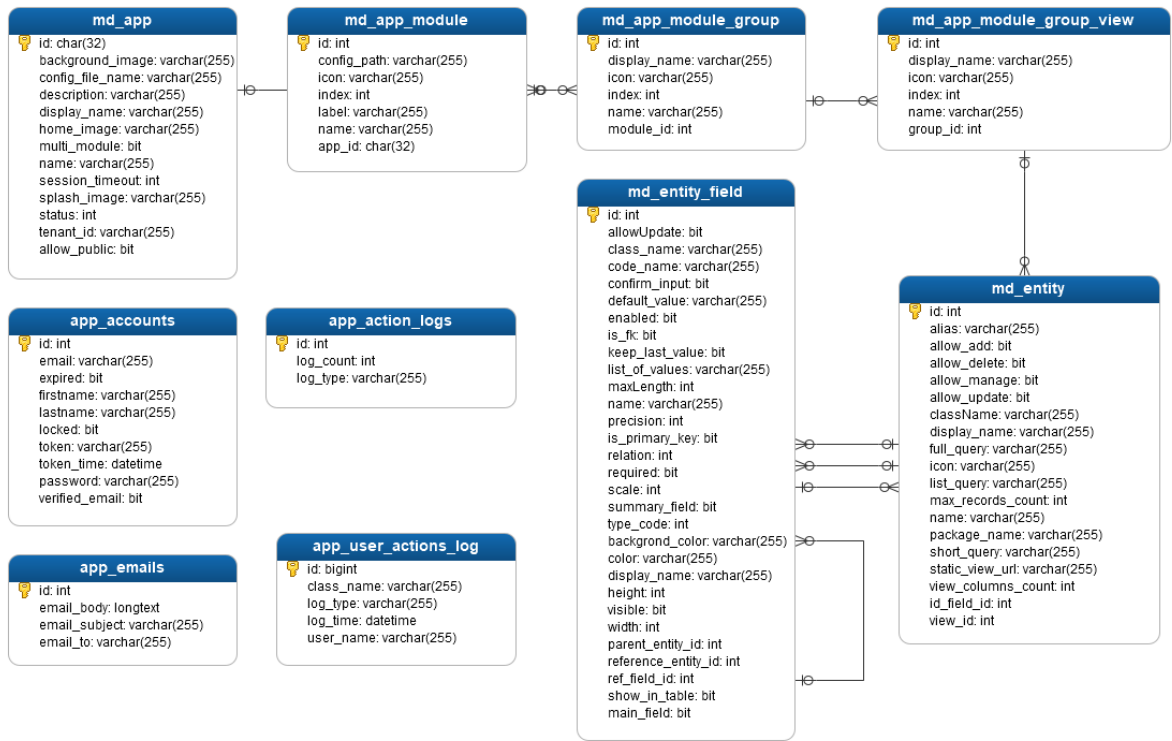


Figure 5.12: Entity Relationship Diagram of Smart-Cloud

5.2 Framework User-Interface

The user interface-views of the Smart-Cloud framework is provided as part of in Clowiz platform. As shown in Figure 5.13, the home page of the platform has main links to CodeGen, FeatureGen, and AppGen apps. In addition, it includes links to the documentation and accounts management.

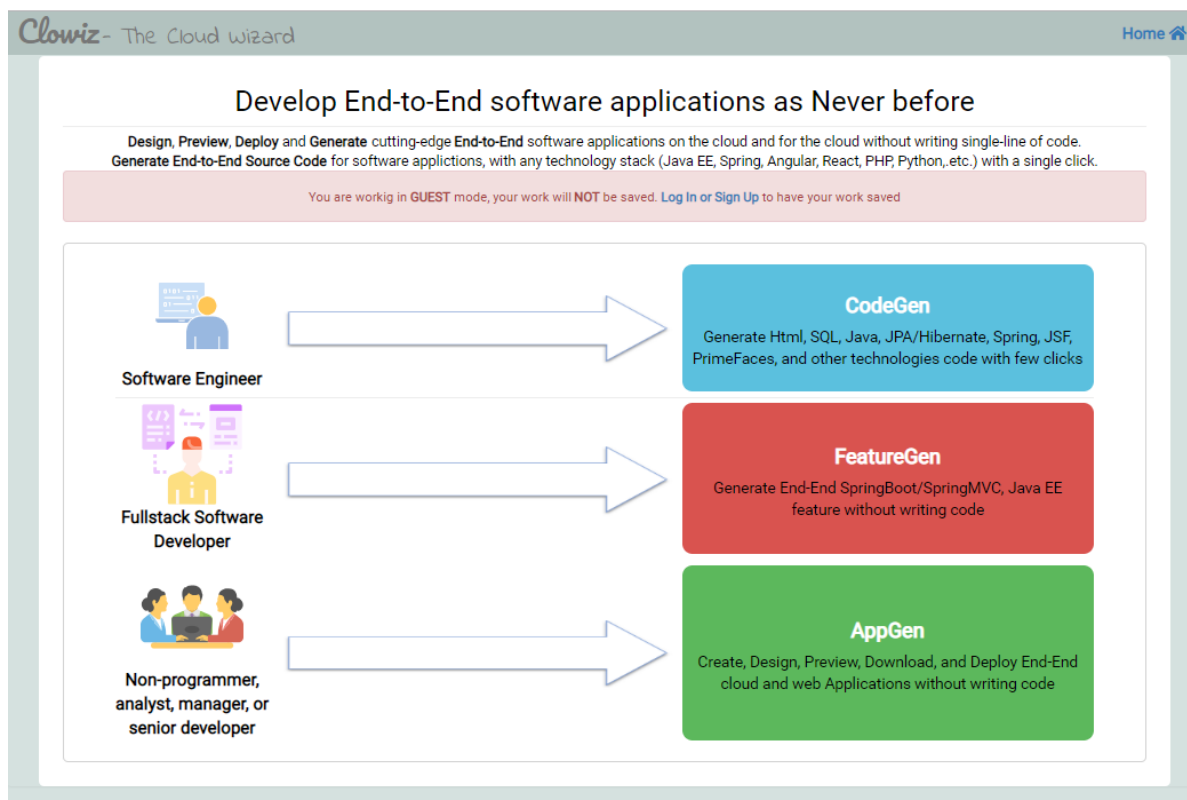


Figure 5.13: Clowiz Platform Home Page

Figure 5.14 shows the authentication page of the platform. It consists of the *Login*, *Sign up*, and *Reset Password* features.

The CodeGen page is shown in Figure 5.15, where the users can configure the metadata of the artifacts followed by selecting the desired code-generator (exporter). Then the generated code will be shown in the middle center of the view.

FeatureGen is shown in Figure 5.16. In this app, users shall input the metadata of the pages that would like to generate full feature to; then a full page preview is

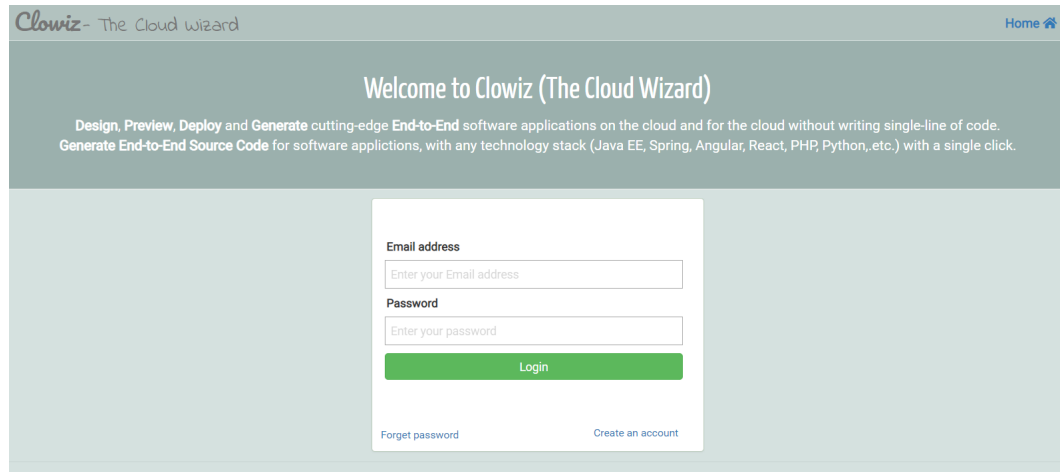


Figure 5.14: Clowiz Authentication Page

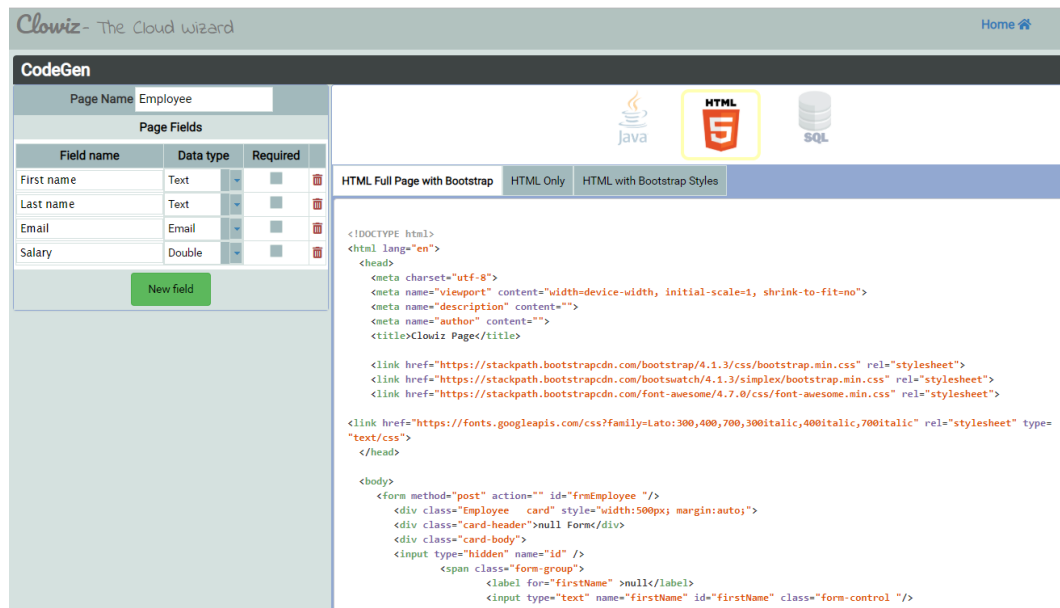


Figure 5.15: CodeGen App Page

shown. Also, the end-to-end MVC source code for that page is shown at the bottom of the view.

AppGen is shown in Figure 5.17, where users can apply the actions discussed in the AppGen process in Section 5.1.4.

The screenshot shows the Clowiz FeatureGen interface. At the top, it displays the page name 'Employee' and 4 UI columns. Below this is a table for 'Page Data Fields' with columns for Field name, Data type, Required, and Delete. The fields listed are Number (Integer), Name (Text), Email (Email), and Salary (Double). To the right is a 'Preview of (Employee)' showing an 'Employee Form' with input fields for Number, Name, Email, and Salary. At the bottom, there are four code snippets: PrimeFaces View with Back-end Binding (XML), JSF Managed Bean (Java), JPA with Lombok (Java), and MySQL SQL Structure (SQL).

Field name	Data type	Required	Delete
Number	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Name	Text	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Email	Email	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Salary	Double	<input checked="" type="checkbox"/>	<input type="checkbox"/>

```

<DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:cc="http://xmlns.jcp.org/jsp/jstl:composition">
<h:form id="frmEmployee">
<p:autoUpdate />
<p:messages />
<p:panelGrid columns="4" id="model" s
<f:facet name="header">#{msg.get('Emp
<p:outputLabel value="#{msg.get('N
<p:inputText type="number" id="num

```

```

package com.app.controllers;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import com.jk.web.controllers.*;
import com.app.models.Employee;

@ManagedBean(name = "mbEmployee")
@ViewScoped
public class MB_Employee extends JKMC

```

```

package com.app.models;

import lombok.Data;
import javax.persistence.*;

@Entity
@Table(name="employee")
@Data
public class Employee{

    @Id
    @Column(name="id")

```

```

create table `employee` (
`id` integer not null auto_increment,
`number` integer NOT NULL,
`name` varchar(250) NOT NULL,
`email` varchar(250) NOT NULL,
`salary` double precision NOT NULL,
primary key (id)
);

```

Figure 5.16: FeatureGen App Page

The screenshot shows the Clowiz AppGen interface. At the top, it displays 'Hi, Jalal!'. Below this is a 'Create New Software Application' button. The main area shows a list of applications with their phases and actions:

- Publication Management System** (In Development phase): Modules (1) Page Groups (2) Pages/Views (4). Actions: Design, Preview, Source Code, Deploy, Archive.
- ERP System** (In Deployment phase): Modules (1) Page Groups (3) Pages/Views (6). Action: Undeploy.
- Smart - HRMS** (In Development phase): Modules (1) Page Groups (2) Pages/Views (3). Actions: Design, Preview, Source Code, Deploy, Archive.
- Demo App** (In Archiving phase): Modules (1) Page Groups (3) Pages/Views (3). Actions: Un-archive, Delete.

Disclaimer: Clowiz (Cloud-Wizard) is under heavy development and is still in Alpha testing phase, [more details](#). Clowiz is part of the dissertation of Jalal Kiswani, a PhD candidate at the CSE Department of University of Nevada, Reno. Email: Jalal@Clowiz.com [Read more](#)

Figure 5.17: AppGen App Page

5.3 Framework Implementation

As discussed in Section 4.5, different technologies from the Java ecosystem were used, specifically, JavaSE, JSF, PrimeFaces, SpringBoot, Spring Framework, JPA, Hibernate. Spring-Tools-Suite (STS) was used as the integrated development environment for the implementation of the framework components, and Maven as the standard project object model. Figure 5.18 shows the project structure of Smart-Cloud implementation.

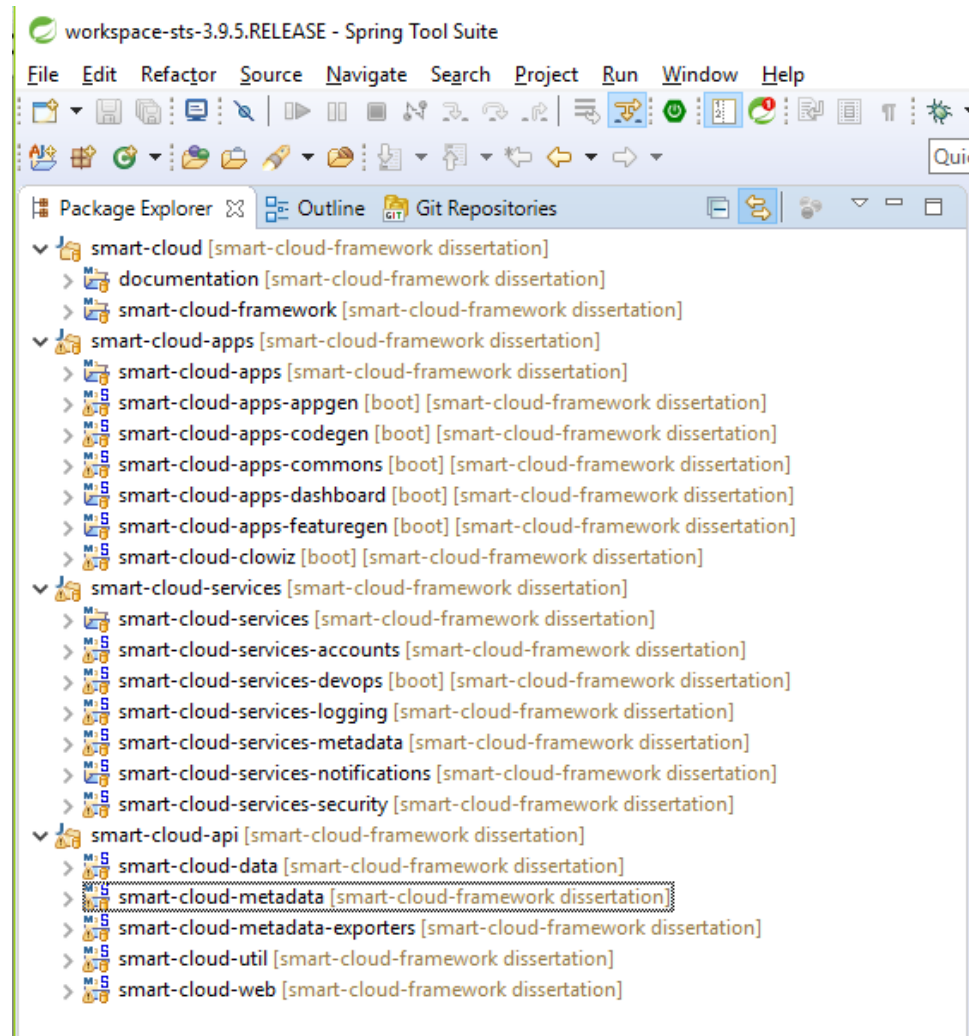


Figure 5.18: AppGen App Page

Chapter 6

Case Study: Publication Management System

As a case study of Smart-Cloud framework, the chapter includes a full implementation of a Publication Management System (PMS) using Smart-Cloud framework. PMS was entirely designed using Clowiz platform, the model-driven app-suites of the framework. In particular, all the artifacts were designed and generated using the cloud-based design tools without writing any line of code.

The following sections include the requirement and design of PMS system, followed by a step-by-step implementation, starting from the application creation to the final deployment to Pivotal-Cloud Foundry.

6.1 PMS Overview

PMS is an application that enables authors, mainly in the academia and the research fields, to manage their publications more effectively, by providing more convenient access to their publications information and resources. PMS is considered a data-intensive application (i.e., information system), where data management is the core functionality of the system.

Figure 6.1 shows the Use-Case diagram of PMS, which includes the management of *Publications* as main use case, which requires the management of *Universities*, *Departments*, and *Authors* data as sub-use cases.

As shown in Figure 6.2, the data model of PMS is designed as a UML class

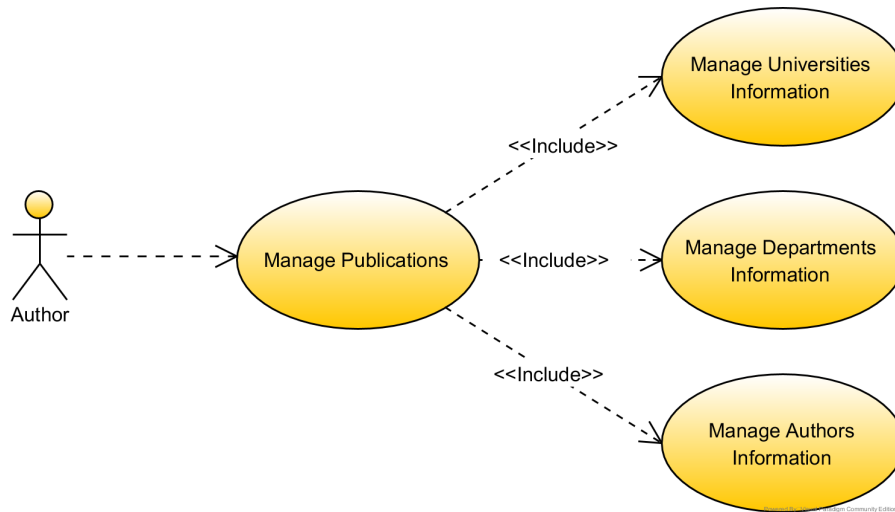


Figure 6.1: PMS use case diagram

diagram. The diagram includes all the required classes covered by the use cases diagram, which include *University* and *Department*, *Author*, and *Publications*.

6.2 Implementation

The process of PMS development on the platform starts follows the proposed process discussed in Section 5.1.4.

6.2.1 Create PMS Application

The process starts by creating a new application on AppGen, where users navigate to the AppGen URL at <https://www.clowiz.com/app-manager/>, followed by clicking on *Created New Software Application* button. The created application is shown in Figure 6.3.

6.2.2 Design PMS App

After the application is created, the high-level artifacts of the application were designed using the application-designer of AppGen. In particular, it includes creating the menus and the pages. We create two page-groups, *Config* and *Authors*. In the *Config* page-group, we added two pages, *University*, which manages the universi-

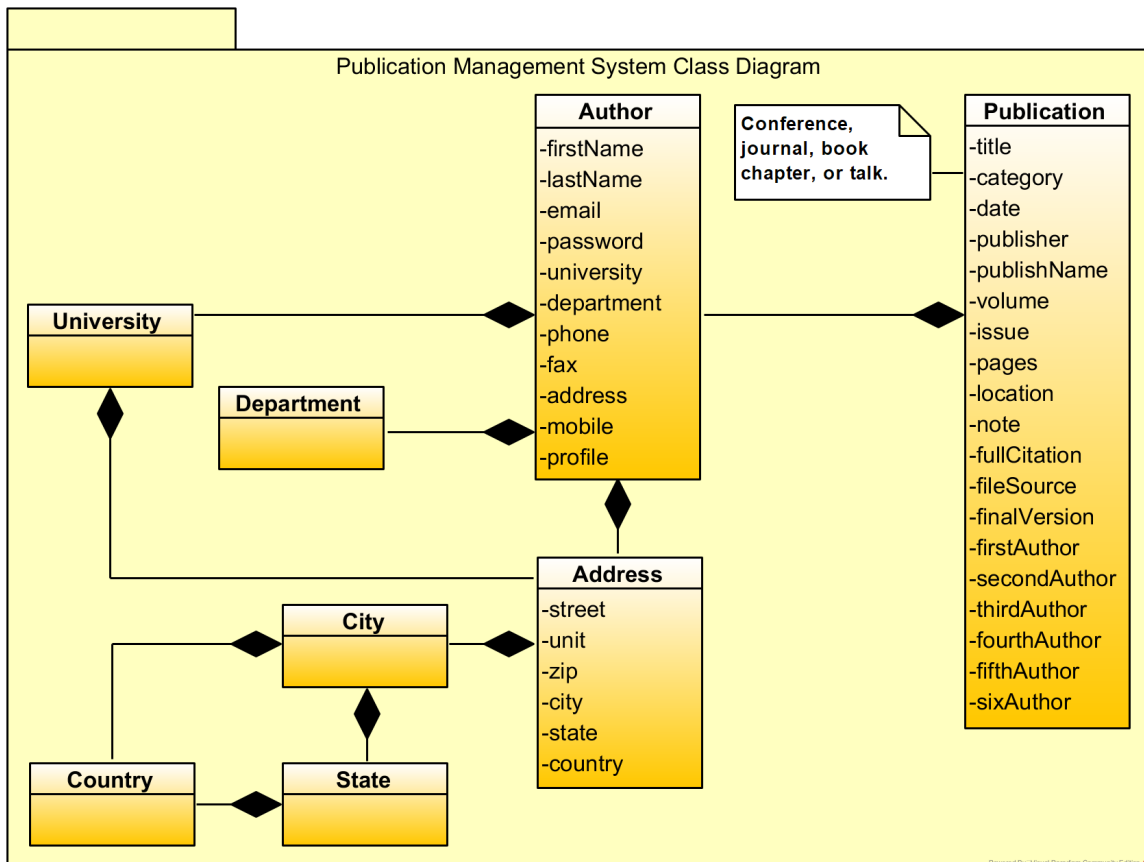


Figure 6.2: PMS class diagram

ties data, and *Department*, which manages the Departments data. In the second view group, *Authors*, two pages were created, *Author* and *Publications*. The Author page manages the authors' data, and the publication page manages all the publications information. The pages and pages-group design is shown in Figure 6.4.

6.2.3 Design PMS Pages

The pages design of *University*, *Department*, and *Author* are shown in Figures 6.5, 6.6, and 6.7 respectively. Even though the University and Department pages only include a field for the name, which could be simply replaced by a field with *List of Values* in the *Publication* page, this design is more generalized and flexible to allow using the system without frequent changes to the code, thus making it highly configurable.

The design of the publication page consists of many advanced metadata con-

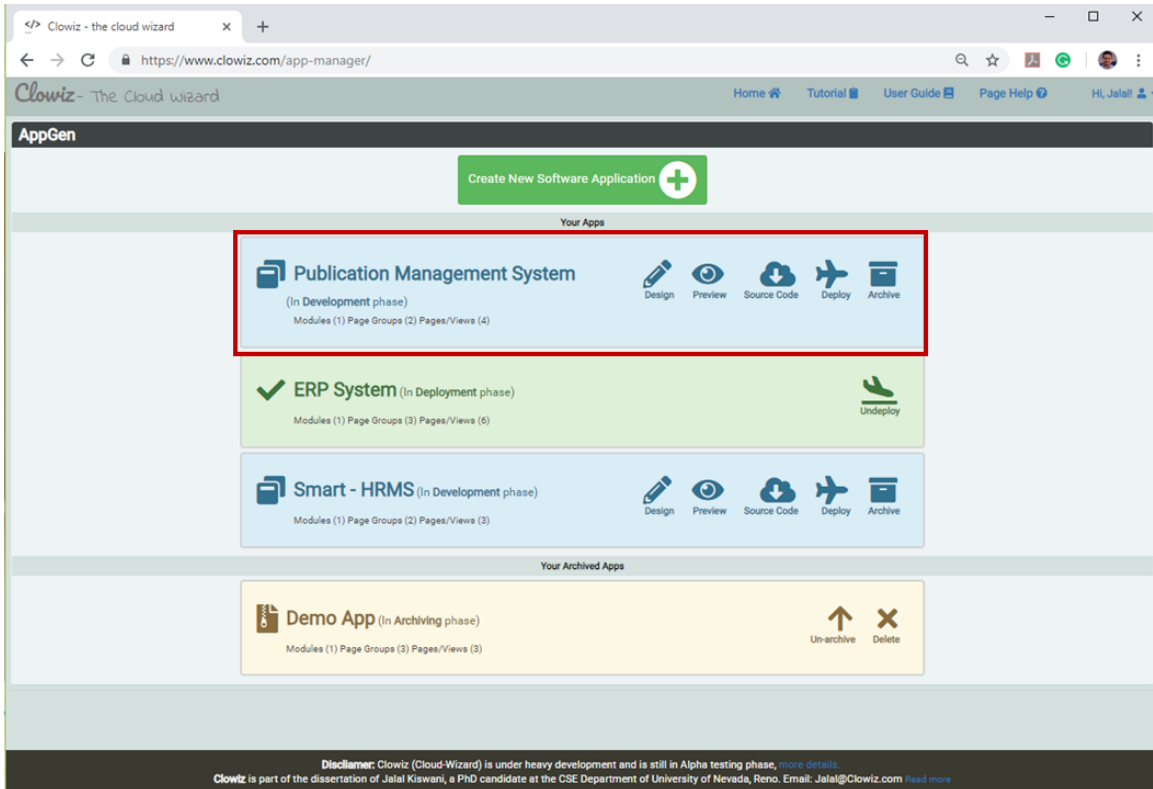


Figure 6.3: PMS project in AppGen manager

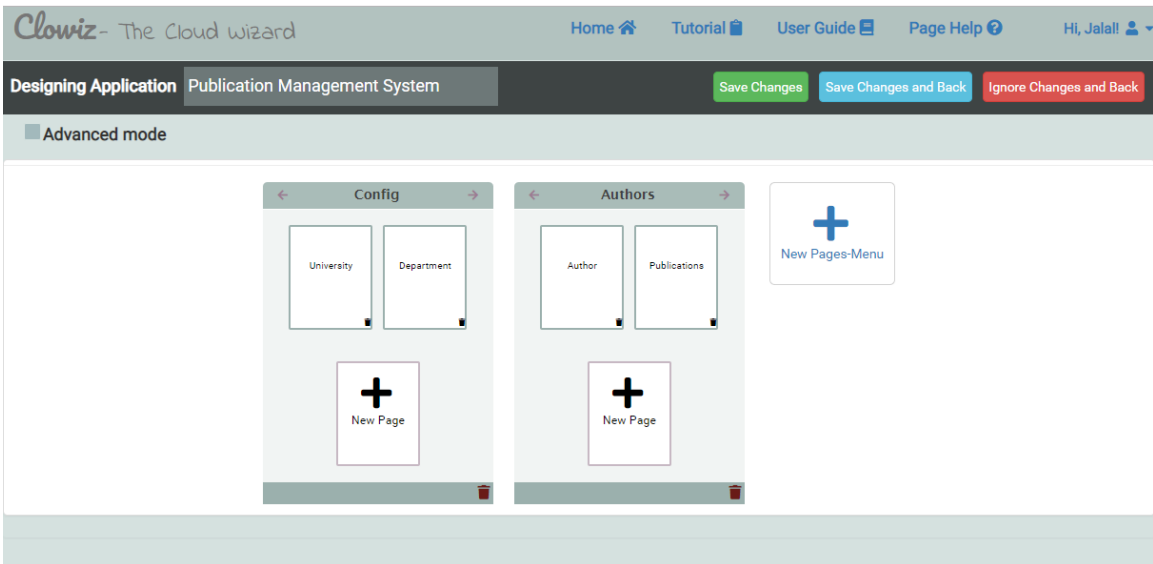


Figure 6.4: PMS project in AppGen designer

figurations. As shown in Figure 6.8, it consists of a *List of Value* attribute for the *Category* field, that shows a list contains *Conference*, *Journal*, *Book*, or *Talk*. In

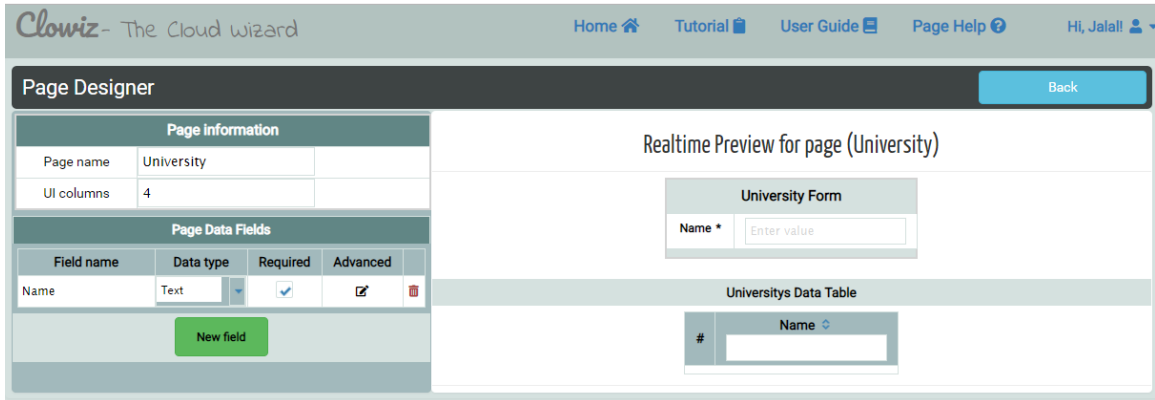


Figure 6.5: PMS University page design

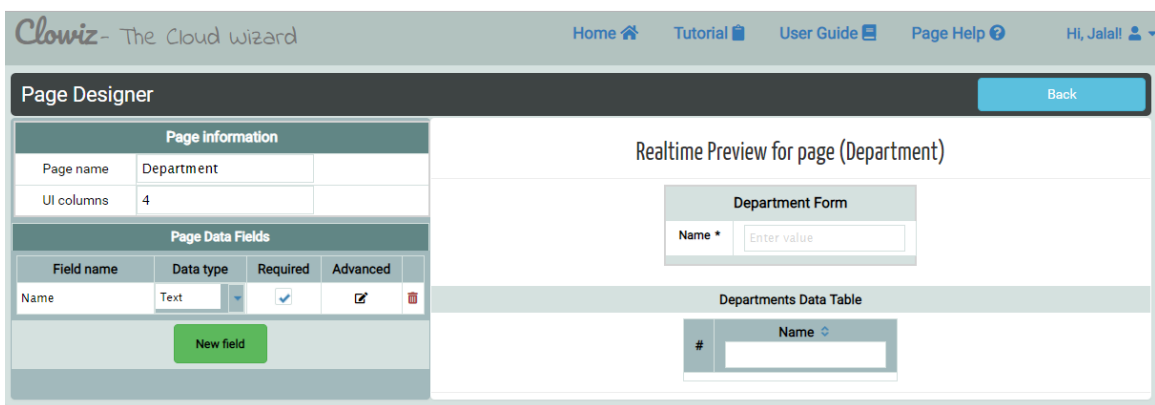


Figure 6.6: PMS Department page design

addition, it consists of *Binary* fields such as the *Final Version* and the *Full Source*. Moreover, it consists of relations with all the other pages, that includes University, Department, and Author.

6.2.4 PMS Prview

After the pages-design of the system, users can return to the AppGen home page and preview a static prototype for the application, which does not include any functionality or data, as shown in Figure 6.9.

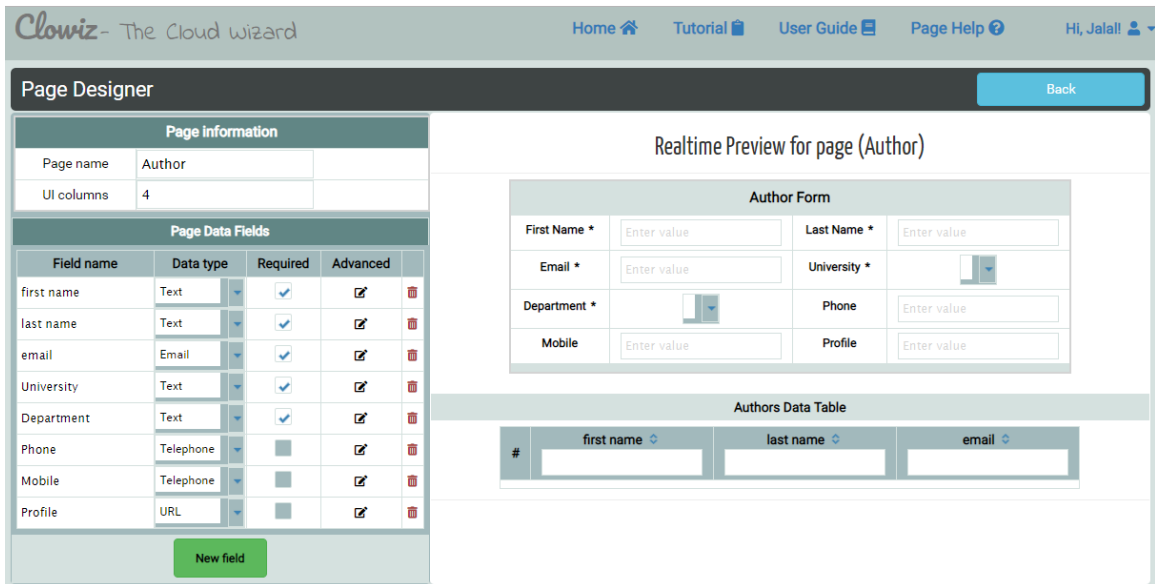


Figure 6.7: PMS Author page design

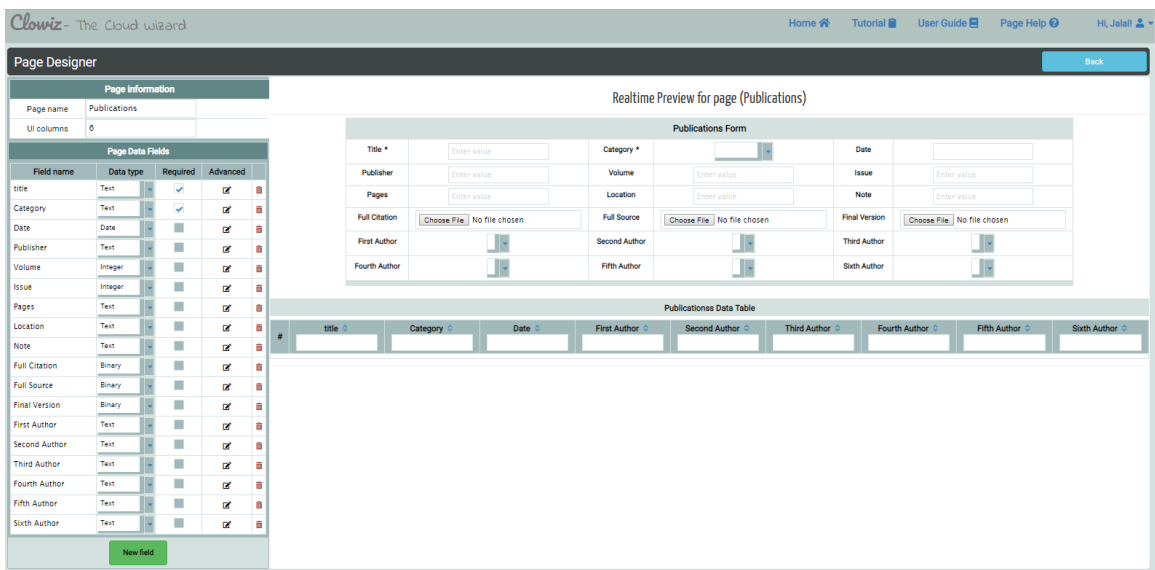


Figure 6.8: PMS Publication page design

6.2.5 Download PMS Source Code

To download the full source code of the PMS application, users shall click on the *Source Code* icon in the AppGen home page; this will trigger generating the full software artifacts, compressing it, then downloading it to the user machine as a compressed file named *publication-management-system.zip*.

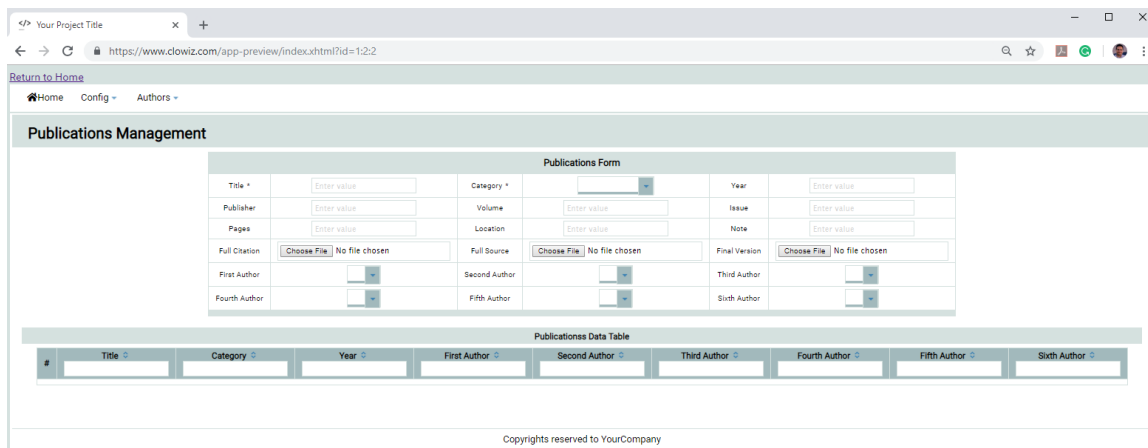


Figure 6.9: PMS preview

6.2.6 Import PMS Project into IDE

When the application was downloaded to our local file system, we extracted the compressed file, then imported it as Java Maven project using Spring-Tool-Suite (STS) IDE, an Eclipse based IDE. Figure 6.10 shows the Maven import view from STS.

After importing the project, the structure of the generated projects shows configuration files, Java, and web artifacts. The artifacts are shown in Figure 6.11.

6.2.7 Launch PMS Locally

For a smooth and straightforward execution, the generated code includes a *Main* class that includes a call to an embedded java web-server that is bundled in the framework. Running this class as a stand-alone application will initialize the framework, launch the web server, and launch a browser window for the applications. The console output of the running application is shown in Figure 6.12.

When the application is launched, the home page of the PMS system is shown, which consists of the product-name, version, summary, header, and footer. In addition, it includes a menu bar and menu items that include links to all the other pages. The home page of PMS is shown in Figure 6.13.

In the *Config* menu, when clicking on the university or departments menu items it

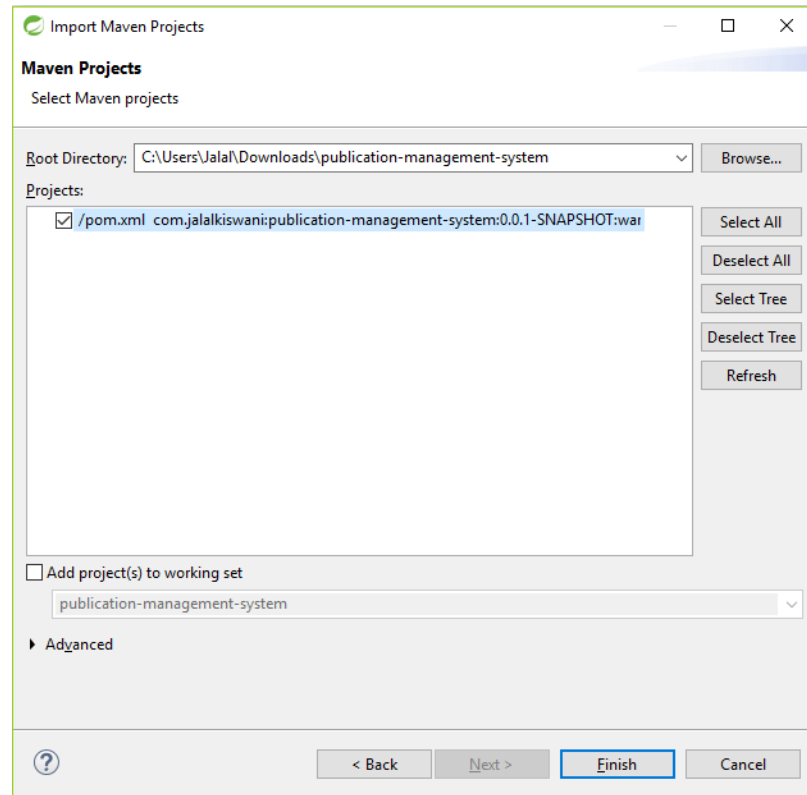


Figure 6.10: PMS Maven import

will show full management pages that manage their information. The design including a demo data are shown in Figures in 6.14 and 6.15.

Under the *Authors* menu, Author and Publication management pages can be accessed. The full design and dummy data of these pages are shown in Figures 6.16 and 6.17.

Note: The implementation of Address, City, State, and Country classes were not included for the sake of simplicity in documentation and the reader.

6.3 PMS Deployment to the Cloud

After the full testing of the PMS system, we used STS IDE to deploy PMS system to Pivotal Cloud Foundry (PCF). The live version of PMS can be accessed at <https://pms.clowiz.com>.

As shown in Figure 6.18, PMS currently deployed on a single instance with 1GB

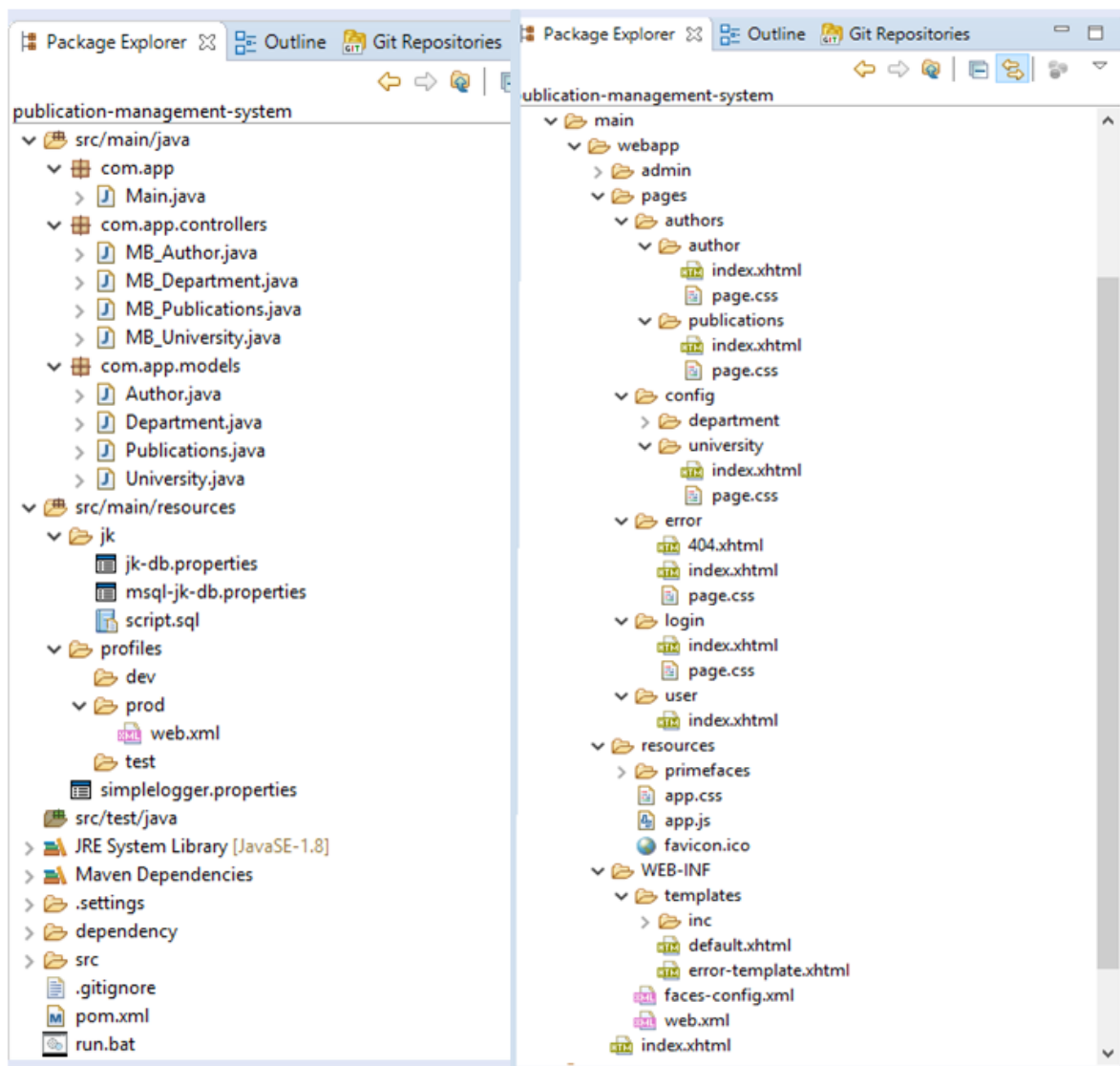


Figure 6.11: PMS project structure

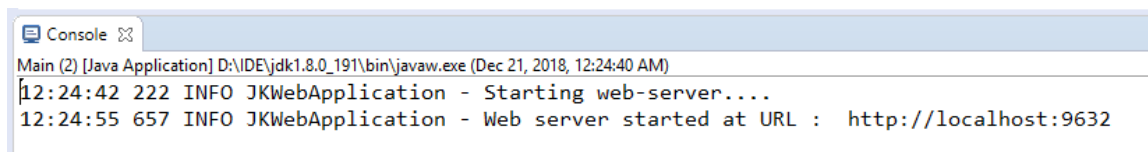


Figure 6.12: PMS console output

of memory and 1GB of storage. Processing power is transparent from developers and handled internally by the Cloud Foundry platform.

Based on the 12-factors-app recommendations, resources should be attached at

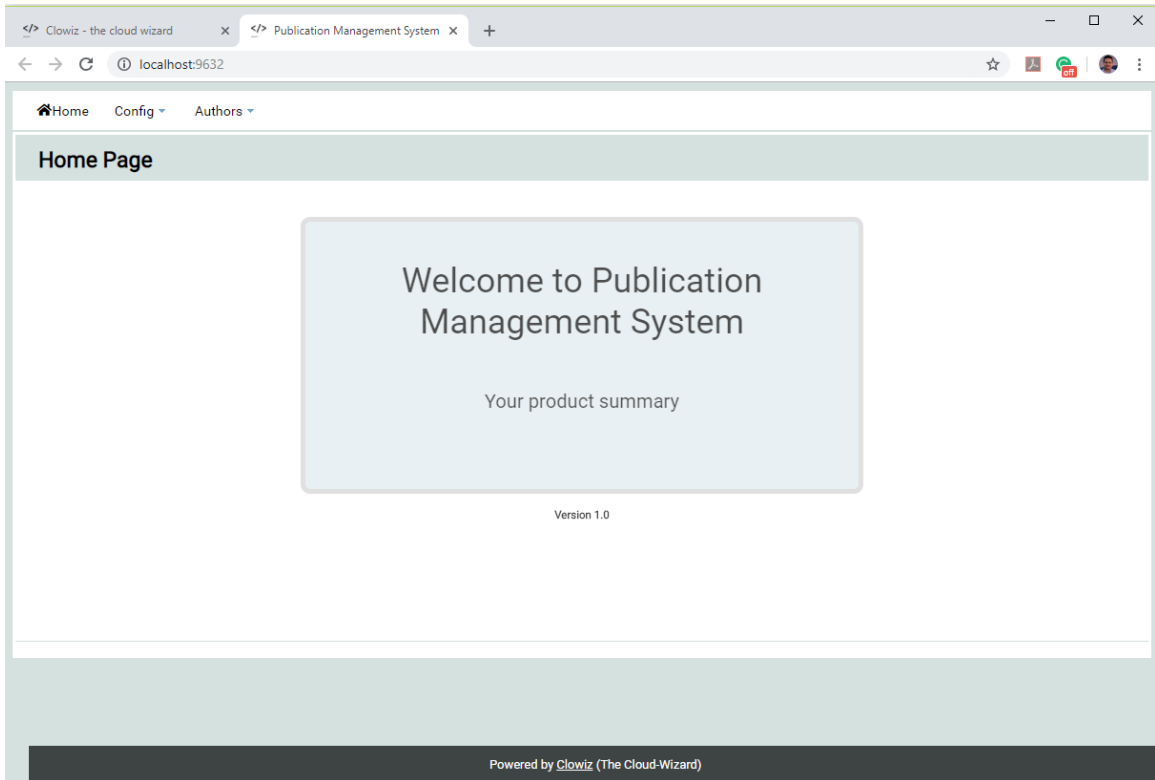


Figure 6.13: PMS Home page

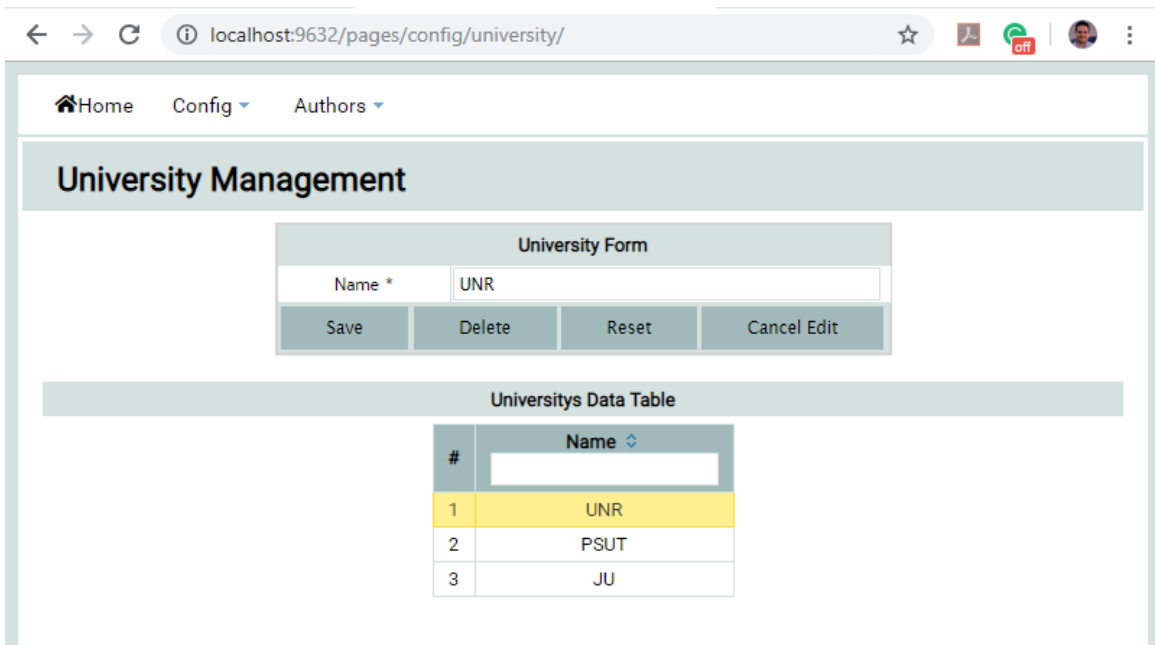


Figure 6.14: PMS universities management page

Home Config Authors

Department Management

Department Form

Name *	CSE
--------	-----

Save Delete Reset Cancel Edit

Departments Data Table

#	Name
1	CSE
2	Marketing
3	Business

Figure 6.15: PMS departments management page

Home Config Authors

Author Management

Author Form

First Name *	Sergiu	Last Name *	Dascalu
Email *	dascalus@cse.unr.edu	University *	UNR
Department *	CSE	Phone	Enter value
Mobile	Enter value	Profile	Enter value

Edit Reset

Authors Data Table

#	first name	last name	email
1	Jalal	Kiswani	jalal@nevada.unr.edu
2	Frederick	Harris	fredh@cse.unr.edu
3	Sergiu	Dascalu	dascalus@cse.unr.edu

Figure 6.16: PMS authors management page

run-time by the hosting environment, and since Smart-Cloud follows the 12-factors app, MySQL instance is attached to the application at run-time as shown in Figure 6.19.

Home Config Authors

Publications Management

Publications Form

Title *	Smart-Cloud: A Framework for Cloud Native		Category *	Journal	Year	2019
Publisher	Enter value	Volume	Enter value	Issue	Enter value	
Pages	Enter value	Location	Enter value		Note	Enter value
Full Citation	<input type="button" value="Choose File"/> No file chosen	Full Source	<input type="button" value="Choose File"/> No file chosen		Final Version	<input type="button" value="Choose File"/> No file chosen
First Author	Kiswani	Second Author	Dascalu	Third Author	Harris	
Fourth Author		Fifth Author		Sixth Author		

Publications Data Table									
#	Title	Category	Year	First Author	Second Author	Third Author	Fourth Author	Fifth Author	Sixth Author
1	Survey of Cloud Applications: Past, Current and Future	Journal	2019	Jalal Kiswani	Sergiu Dascalu	Muhanna Muhanna	Frederick Harris	-	-
2	Smart-Cloud: A Framework for Cloud Native Applications Development	Journal	2019	Jalal Kiswani	Sergiu Dascalu	Frederick Harris	-	-	-
3	Clowiz: A Model-driven Development Platform for Cloud-based Information Systems	Conference	2018	Jalal Kiswani	Sergiu Dascalu	Muhanna Muhanna	Frederick Harris	-	-
4	Cloud-RA: A Reference Architecture for Cloud Based information Systems	Conference	2018	Jalal Kiswani	Sergiu Dascalu	Frederick Harris	-	-	-
5	Let's VR: A Multiplayer Framework for Virtual Reality	Conference	2018	Alex Hansen	Kurt Andersen	Brittany Sievert	Jalal Kiswani	Sergiu Dascalu	Frederick Harris
6	Smart-EIS: An End to End Enterprise Information Systems Framework	Journal	2018	Jalal Kiswani	Muhanna Muhanna	-	-	-	-
7	Using metadata in optimizing the design and development of enterprise information systems	Conference	2017	Jalal Kiswani	Muhanna Muhanna	Abdullah Qusef	-	-	-
8	Software Infrastructure to Reduce the Cost and Time of Building Enterprise Software Applications: Practices and Case Studies	Conference	2017	Jalal Kiswani	Muhanna Muhanna	Sergiu Dascalu	Frederick Harris	-	-
9	Project Manager Roles in Software Information Systems: Case Studies from Jordan	Conference	2016	Jalal Kiswani	Abdullah Qusef	-	-	-	-

Figure 6.17: PMS publication management page

Pivotal Web Services Search apps, services, spaces, & orgs press [J] kiswanij@gmail.com

Home Marketplace

Home / kiswanij / production / pms

APP pms Running [VIEW APP](#)

Overview Service (1) Routes (2) Logs Tasks Settings Buildpack: N/A

Last Push: 02:40 AM 12/21/18 **App Summary**

Updated app kiswanij@gmail.com 12/21/2018 at 02:40:22 AM	Instances / Allocated 1 / 1	Memory / Allocated 0.25 / 1.00 GB	Disk / Allocated 0.19 / 1.00 GB
---	--------------------------------	--------------------------------------	------------------------------------

Mapped route to app
kiswanij@gmail.com 12/21/2018 at 02:40:22 AM

Started app
kiswanij@gmail.com 12/21/2018 at 02:38:44 AM

Updated app
kiswanij@gmail.com 12/21/2018 at 02:34:41 AM

Updated app
kiswanij@gmail.com 12/21/2018 at 02:34:39 AM

Mapped route to app

Processes and Instances [View in PCF Metrics](#)

web

Instances	1	Memory Allocated	1 GB	Disk Allocated	1 GB	<input type="button" value="SCALE"/>
Autoscaling <input type="checkbox"/>						
#	CPU	Memory	Disk	Uptime		
0	0%	251.43 MB	192.74 MB	23 hr 46 min		

Tools Docs Support Blog Status

Figure 6.18: PMS App on PCF

Deploying to PCF should include routing configurations, to be able to access apps and services externally. Figure 6.20 shows the configured routings for PMS, which are <http://pms.clowiz.com> and <http://pms2.cfapps.io>. The final deployed version of PMS is shown in Figure 6.21.

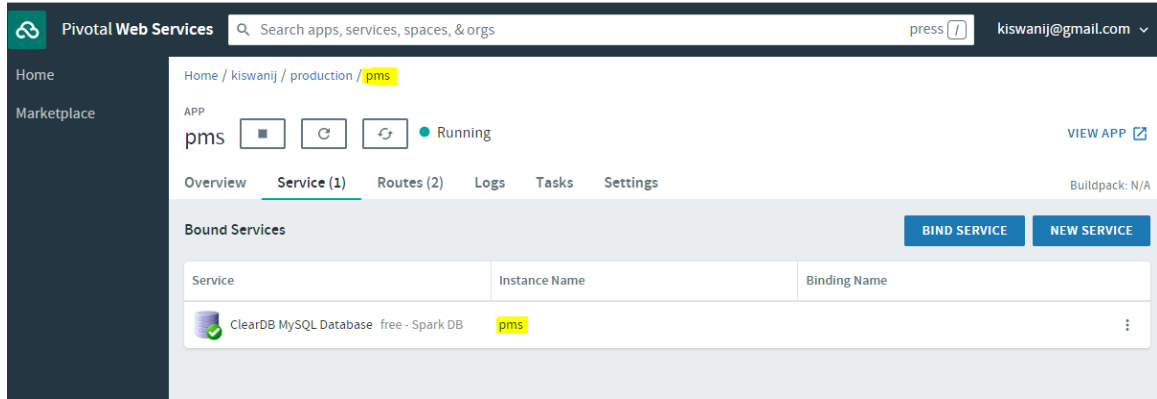


Figure 6.19: PMS MySQL Service

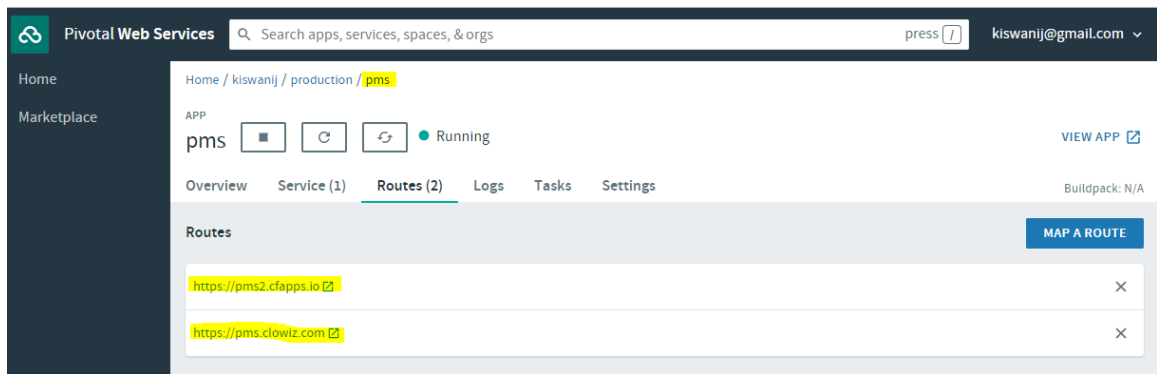


Figure 6.20: PMS URL Routing

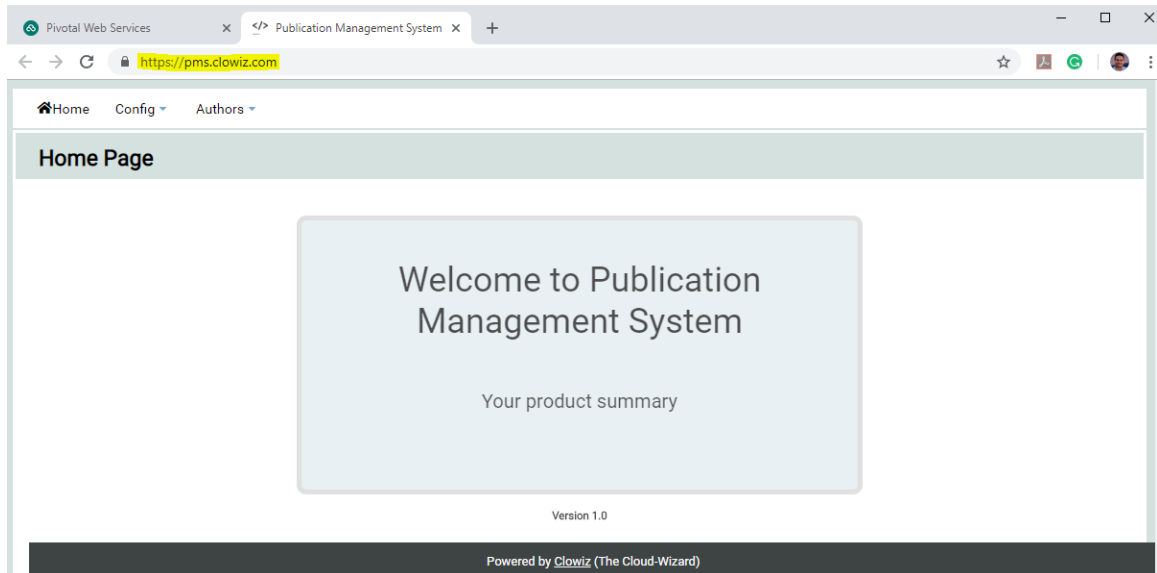


Figure 6.21: PMS Production on PCF

Chapter 7

Evaluation and Discussion

This chapter includes the evaluation and discussion of Smart-Cloud framework, which are based on a user study conducted for 36 professionals from industry and academia, and the case-study presented in Chapter 6.

7.1 User Study

To evaluate Smart-Cloud framework, we designed and conducted a user study that involved academics from the software engineering discipline and professionals from the software industry field. The main goals of the study were to answer the following three questions:

- Does the work presented in this research solve a significant problem which are worth attention?
- Does the work presented here contribute to solving this problem?
- What have been missed in this work, and what could be done better?

7.1.1 Participants

The user study is based on an experiment conducted between the period of Monday 24 to 31 December, 2018. It included 36 software technologies practitioners and academics with various seniority levels, positions, and education. In addition, the participants come from different regions all-over the world. As shown in Figure 7.1,

the survey included a wide range of participants with different professionalism levels. In addition, participants come from different academic backgrounds, as shown in Figure 7.2. Moreover, Figure 7.3 shows the participants day-to-day involvement in software development projects activities.

What is your experience level in the software development field?

36 responses

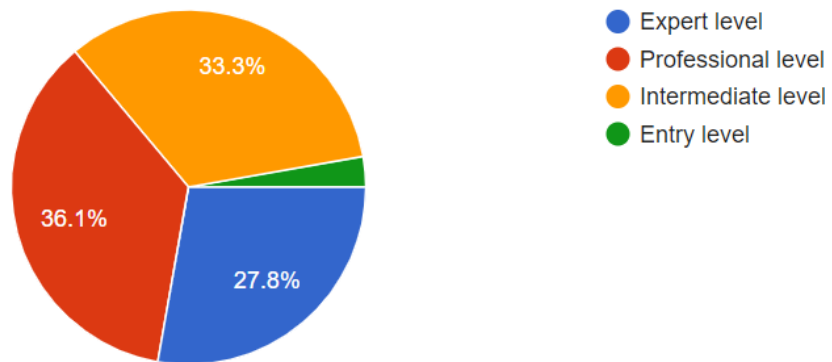


Figure 7.1: Experience levels of the user study participants

What is your highest academic degree?

36 responses

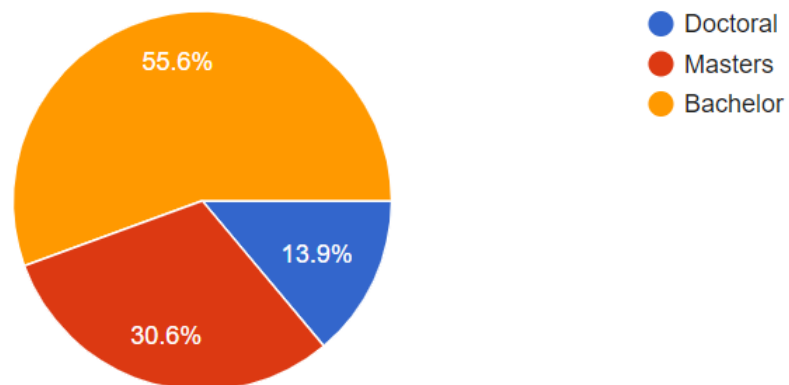


Figure 7.2: Academic levels of the participants in the user study

What is your current day to day work in software development projects? (please check all that apply)

36 responses

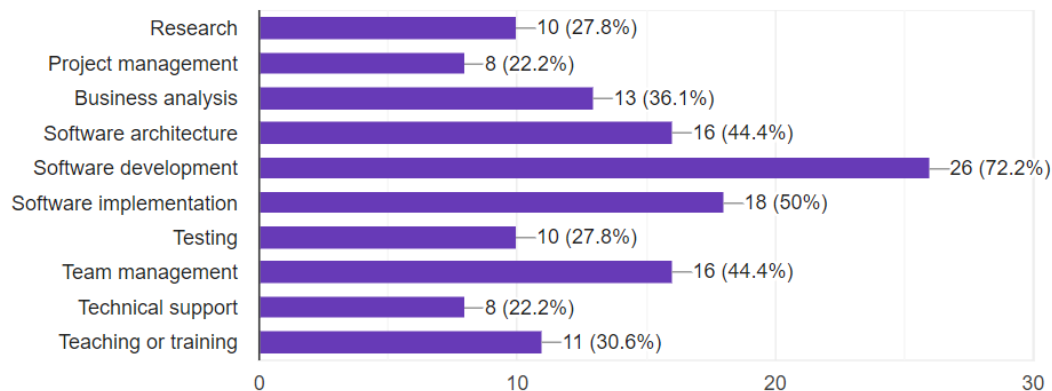


Figure 7.3: Involvement of the participants in software development activities

The job titles of the participants included: Solution Managers, System architects, Associate and Assistant Professors, IT Directors, System Analysts, Ph.D. Students, Implementation Managers, Services Manager, Integration Specialist, Technical Team Leader, Head of QA, Software Engineers, Senior Software Engineers, and Software Development Consultants. The total years of experience of the participants range from 3-20 years of experience.

7.1.2 Methodology

The experiment consists of three instructions the participants shall follow: (i) reading an online tutorial about Clowiz platform, (ii) implementing the steps in the tutorial or trying the platform randomly, then (iii) completing an online survey.

To measure and evaluate the user's behaviors, all the transactions and actions performed on the platform were recorded automatically for every participant. These transactions were later used in the analysis to map the user responses with the actual experiment of the system.

Clowiz Tutorial

The first instruction in the user study was to read an online tutorial about Clowiz platform, which was published on the Clowiz platform website at <https://doc.clowiz.com/tutorial.html>.

The tutorial consists of an introduction about the Clowiz platform in General, and CodeGen, FeatureGen, and AppGen applications in particular.

The CodeGen section includes instructions to create *Java JPA Entity*, *Full HTML Page with BootStrap*, and *MySQL Table Structure*. Based on the instructions, users are able to generate these software artifacts with few clicks and by modifying the metadata values of the desired fields.

In the FeatureGen section in the tutorial, the same fields that were used in the CodeGen experiment, had also been used in the FeatureGen one, where end-to-end source code were generated based on the *Java Web Stack*. The stack includes *PrimeFaces*, *Java Server Faces*, and *Java Persistence API with Hibernate implementation*.

In the last experiment, the users were asked to develop a simple application using AppGen. The requested application represents a simple human resources management system, which consists of *Departments*, *Job Titles*, and *Employee Profiles* management pages. The development process includes creating, designing, preview, downloading, importing, then launching the project locally as described in Section 5.1.4.

Moreover, to give the participants more flexibility, they were also given an option of reading the tutorial, then trying out the platform randomly. However, and to have a clear understanding of the user's behaviors of using the platform, all the transactions and the interaction with the system were recorded automatically, which enabled distinguishing between the participants who followed the instructions in the tutorial, and who randomly worked on the platform, and also who scanned the tutorial and didn't try the platform.

Smart-Cloud Survey

After following the instructions, and to complete the experiment, the participants were asked to complete an online survey.

The survey organized into five sections: *Introductory Information*, *Participant Information*, *Cloud Applications*, *Clowiz Platform*, and ,finally, *Suggestions and Recommendations*.

- **Introductory Information** This section included an introduction about the user study, followed by the consent required by the IRB process of UNR. After that, it includes the email address of the participant, then (optionally) his/her name, and if they could be contacted for future feedback or questions if needed.
- **Participant Information** This section aims to get general information about the participants including their overall experience in the software development industry, higher academic degree, current job titles, years of experience, and what are their day-to-day activities working on software development projects.
- **Cloud Applications** This section included questions about cloud applications significance, cost, and limitations. In particular, the first question was whether choosing cloud applications is more beneficial than the traditional approaches for green-field projects (i.e., new projects with minimal constraints). The second question is to verify if there a shortage in experienced software engineers who can develop high-quality cloud-based applications. The third, fourth, and fifth questions are about the cost of software development, maintainability, and operation of cloud vs. traditional applications respectively. The last question in this section is about the risk of uplifting traditional applications (i.e., migrating applications to the cloud).
- **Clowiz Platform** This section included questions about the Clowiz platform. Clowiz platform was selected since it exposes the features of Smart-Cloud framework into a user-friendly model-driven development approach. Similar to asking

users to evaluate Google search engine through the simple web interface provided over the internet.

The first two questions include the learning-curve and the usability-level of Clowiz platform. The third, fourth, and sixth questions ask the participant if the platform can reduce the cost of cloud-based application's development, and whether the generated code is high-quality and maintainable. Question five asks the users if they would use Clowiz for their day-to-day work if it could generate the artifacts in the technology they use. The last question in this section, asks about what could be the most significant app from the platform (CodeGen, FeatureGen, or AppGen) based on the participants' experience and requirements.

- **Suggestions and Recommendations** In the last section of the survey, there were three questions. Firstly, we asked the participants about what could be done better; secondly, what new features could be considered in the future; thirdly, we asked them if they have any comments or other suggestions.

A full copy of the questionnaire can be found in Appendix C.

7.2 Results and Discussion

This section includes the results and discussion of the evaluations performed for Smart-Cloud framework, which includes the results and evaluation of the PMS case study presented in Chapter 6, and the results and evaluation of the user study presented earlier.

7.2.1 PMS Results and Evaluation

Smart-Cloud was used to develop an end-to-end Publication Management System (PMS). The front-end model-driven platform, Clowiz, was used to enable rapid application development for the application. The results are shown in Table 7.1.

Table 7.1: The results of implementing PMS using Smart-Cloud

Artifact(s)	Efforts
Project Structure	Statically generated by the framework, it includes creating a Maven project object model (i.e., pom file) and files structure. In addition, it includes a proposed unique structure that enables easier styling and maintaining of pages.
Dependencies	Single dependency of the smart-cloud web framework, which includes all the required utilities and API's to enable faster and efficient development, such as the ManagedBeanWithOrmSupport, and JKObjectDataAccess classes.
Theme resources	Statically generated by the framework, which includes the templates, CSS styles, themes, JavaScript, images, icons, and others.
Configuration files	Statically generated by the framework, which consists of development and live configurations.
View pages	Statically generated by the framework, which are based on the PrimeFaces and JSF components.
Controllers code	Statically generated code that extends a reused class provided by the framework.
Models code	Statically generated by the framework, which are JPA entities that are mapped directly to a database.
Data access code	Provided out of the box from the framework as easy to use APIs.
Web Server	Provided out of the box as embedded web-server, to enable rapid development and execution.
Database	Provided out of the box H2 embedded database to enable faster execution and rapid application development.

Clearly, implementing the framework can reduce the development time by providing all artifacts shown in Table 7.1 out of the box, without writing any single line code. Having all these artifacts as a base for a new software project could be considered a software product line approach, where such approach can reduce the

development time of similar projects up to 90%, as discussed in Section 2.3, which in our case are data-intensive applications.

7.2.2 Users Study Results

As a part of the user study conducted to evaluate Smart-Cloud, the participants were asked to complete an online survey. The survey included questions about cloud applications. The questions and their responses are shown in Table 7.2. The results of Clowiz platform questions are shown in Table 7.3. Moreover, the bar charts of these results are shown in Figures 7.4, 7.5, 7.6, and 7.7. The question is about Clowiz apps, and what participants think is the most significant apps. The results of this question are shown in Table 7.4.

Table 7.2: The results of cloud application questions

Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree	Average
<i>Ranking Value</i>	(1)	(2)	(3)	(4)	(5)	
For new software applications, choosing a cloud-based approach can be more beneficial than choosing traditional approaches	-	1 (2.8%)	3 (8.3%)	19 (52.8%)	11 (30.6%)	4.18
There is a shortage of experienced software engineers who can develop high-quality cloud applications	-	-	10 (27.8%)	16 (44.4%)	10 (27.8%)	4
The cost of software development of cloud based applications is higher than cost of developing traditional software	1 (2.8%)	18 (50%)	6 (16.7%)	10 (27.8%)	-	2.71
The maintainability cost of cloud based applications is higher than that of traditional software applications	4 (11.1%)	19 (52.8%)	4 (11.1%)	6 (16.7%)	2 (5.6%)	2.72
The operational cost of cloud based applications is higher than that of traditional software applications	3 (8.3%)	15 (41.7%)	5 (13.9%)	10 (27.8%)	2 (5.6%)	2.80
Migrating applications developed using traditional approaches to be cloud-based is an expensive and risky process	1 (2.8%)	5 (13.9%)	8 (22.2%)	17 (47.2%)	2 (5.6%)	3.42

In addition to cloud applications and the platform questions, we asked the participants if they have any comments, suggestions, or feedback. The summary of these comments included asking for supporting other technologies, such as Angular, React, and Solr. In addition, they proposed integration other technologies and tools such as GitHub, Amazon AWS, and Docker-Hub. Moreover, they proposed to generate a microservices skeleton for systems as a whole, then for every service individually.

Table 7.3: The results of Clowiz platform questions

Question / Ranking Value	(1)	(2)	(3)	(4)	(5)	(6)	(7)	Average
I find it easy to learn and work on Clowiz	-	1 (2.8%)	-	1 (2.8%)	4 (11.1%)	13 (36.1%)	17 (47.2%)	6.19
I understand the process of developing cloud applications using Clowiz	-	1 (2.8%)	-	5 (13.9%)	6 (16.7%)	12 (33.3%)	12 (33.3%)	5.78
Clowiz platform can reduce the development cost of creating cloud-based applications	-	-	2 (5.6%)	1 (2.8%)	6 (16.7%)	11 (30.6%)	16 (44.4%)	6.06
The code generated by Clowiz is high-quality and maintainable	-	1 (2.8%)	1 (2.8%)	-	8 (22.2%)	15 (41.7%)	11 (30.6%)	5.89
I would use Clowiz features if it could generate the code in my day-to-day working programming language and technology	-	2 (5.6%)	2 (5.6%)	4 (11.1%)	5 (13.9%)	12 (33.3%)	11 (30.6%)	5.56
Clowiz can generate good quality end-to-end cloud-based code, features, and applications	-	1 (2.8%)	1 (2.8%)	5 (13.9%)	5 (13.9%)	14 (38.9%)	10 (27.8%)	5.67

I find it easy to learn and work on Clowiz

36 responses

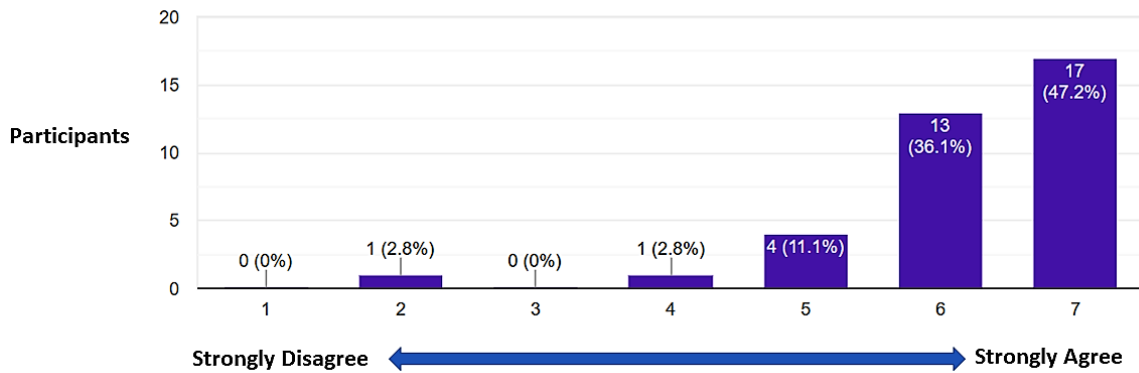


Figure 7.4: Results of Clowiz usability

Table 7.4: The results of the most significant app in Clowiz question

Application	Number of participants think its more significant	Percentage
CodeGen	13	39%
FeatureGen	6	18%
AppGen	14	42%

I understand the process of developing cloud applications using Clowiz.

36 responses

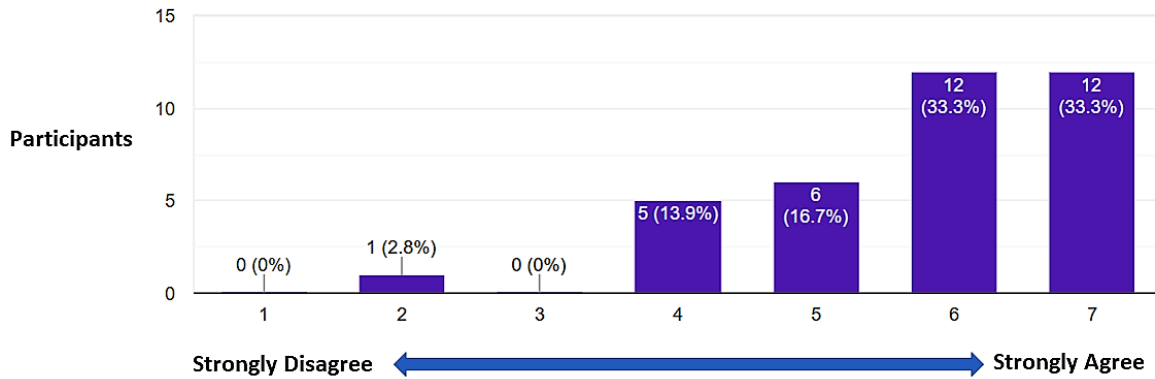


Figure 7.5: Results of the understanding of AppGen process

Clowiz platform can reduce the development cost of creating cloud-based applications.

36 responses

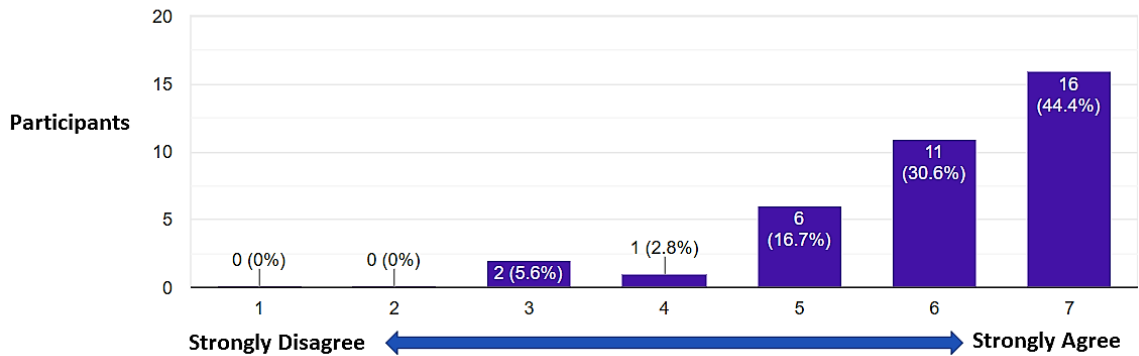


Figure 7.6: Results of Clowiz reducing the cost of cloud applications development

To enable faster generation, some of them proposed exporting and importing the metadata into JSON format. Some performance tuning and user experience requests were also proposed, such as having keyboard shortcuts, to enable faster entry for the metadata. Also, to get more flexibility, many of them asked about the possibility of enabling the customization of the generated artifacts, themes, and styles before downloading the projects.

The code generated by Clowiz is high-quality and maintainable

36 responses

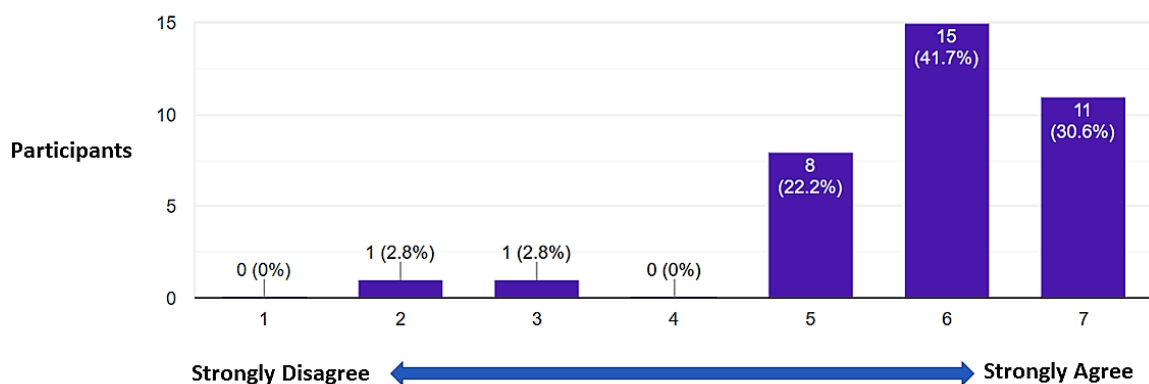


Figure 7.7: Results of generated code quality of Clowiz

I would use Clowiz features if it could generate the code in my day-to-day working programming language and technology.

36 responses

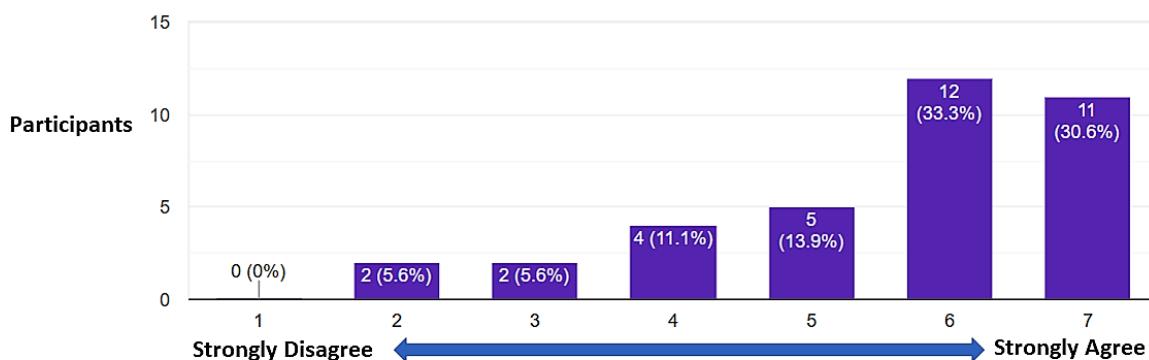


Figure 7.8: Results of using Clowiz in day-to-day work

7.3 Discussion

Nowadays, the cloud-based approach of software applications development is one of the important current trends in the software engineering industry. However, we believe that there are many challenges surround the adoption of this approach in many organizations.

Clowiz can generate good quality end-to-end cloud-based code, features, and applications.

36 responses

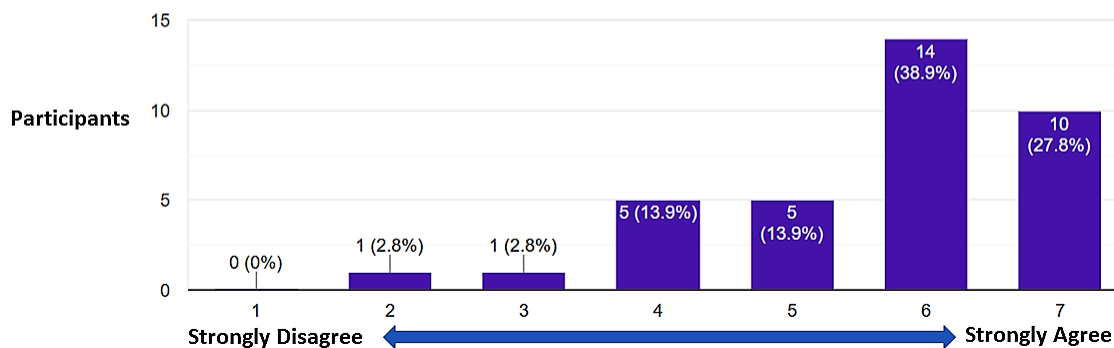


Figure 7.9: Results of Clowiz overall quality of generated applications

However, before trying to identify these problems, we wanted to be sure that building applications following the cloud approach is the preferred way in green-field projects over the traditional on-premises approach in organizations in different domains. So based on the results of the user study, more than 82% of the participants say that choosing cloud applications can be more beneficial than the traditional approach in new software projects, with an average of 4.18/5, where 5 is Strongly Agree and 1 is Strongly Disagree.

In addition, and as shown in Table 7.2, around 50% of participants think that the development, maintainability, and operational cost of cloud applications is lower than traditional the approaches. As shown in the table, the average of the responses are 2.71, 2.72, and 2.8 respectively, where lower values means that the cloud is cheaper. The goal behind designing the survey for these questions in an opposite direction is to ensure that the participants are completing the survey with rational data and not rushing the answers.

The above results shows the significance of cloud computing and cloud applications in reducing the cost of development, maintenance, and operations, which answers the first question we raised in Section 7.1: *Does the work presented in this*

research solve a significant problem which are worth attention?

However, since averages of development, maintainability, and operational cost are all close to the median, we think that these questions require more investigation.

On the hand, all the previous questions were related to new projects in a green field situation. So, we also wanted to check if migrating current applications to be cloud-based is considered a risky process for the organizations, and the results show that more than 40% of the participants think it is risky to uplift on-premise applications (migrate applications to the cloud), with an average of 3.42/5. However, we also think that this requires more investigation, since based on the feedback of some participants, privacy, compliance, and security concerns are still dominant.

So the question is, if most of the participants think that cloud applications are more beneficial than the monolithic based, then why there are still many on-primes applications in many organizations?. The answer to this question is shown in Figure 7.10, where more than 71% of the participants say that there is a shortage in experienced software engineers who can develop high-quality cloud software applications, with an average of 4/5 of the participants think that there is a shortage in this area, which confirms the results of RightScale report published in 2018, which discussed in Section 2.1.6, that the lack of resources and expertise has become a major barrier for adopting cloud computing.

We believe that the above results, show that there is a need for an approach that enables developing cloud-based applications in an efficient and effective way, without requiring particular expertise, is significant and worth attention.

As a solution for the presented problem, we proposed Smart-Cloud framework, which is a metadata-driven approach that enables rapid applications development of cloud-based applications, without the need for special expertise in software engineering. Furthermore, we developed the Clowiz platform, which is a cloud-based model-driven-development approach built on the top of the framework to enable more efficient usage of the framework, without the need of writing code to use it.

However, we wanted to be sure that people from the software engineering industry

There is a shortage of experienced software engineers who can develop high-quality cloud applications

36 responses

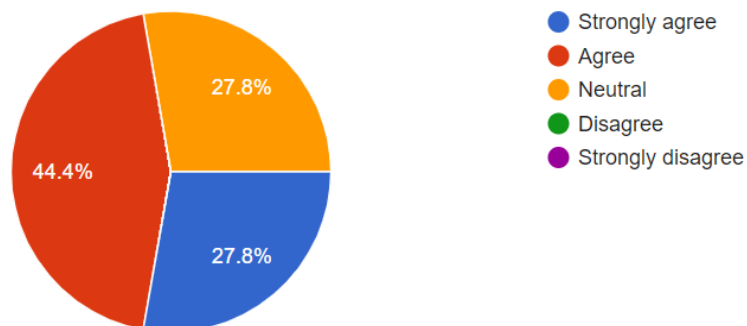


Figure 7.10: Shortage in experienced cloud application's developers

from different seniority levels, and different backgrounds, are able to use the platform without any training and with a short period of time and learning curve. So based on the user study, and by only reading a tutorial that requires 10-20 minutes, more than 87% say that the platform is easy to learn, with an average of 6.19/7, and more than 76% of them understand the process of development with an average of 5.78/7, where 7 is Strongly Agree and 1 is Strongly Disagree.

Even though a short learning curve is significant in any approach, the quality, and the willingness to adopt this approach by the participants can indicate if the approach is practical enough to solve real-life problems or not. Based on the user-study, more than 93% of the participants say that the code generated by the platform is high-quality and maintainable with an average of 5.89/7. Also, 76% of them would use the platform in their day-to-day work if it can generate the code in the technology they use, with an average of 5.56/7.

Moreover, more than 90% of the participants say that the platform can reduce the development cost of cloud-based applications, with an average of 6/7.

On the other hand, we also wanted to know what features of the platform could be more useful to the participants. We found that all the people who fully followed

the tutorial say that AppGen is the more significant application with a percentage of 38.9%, while 36.1% think CodeGen is the most significant; however, only 16.7% think that FeatureGen is the most important one.

The feedback of the open-ended questions was promising, where most of the participants were positive with the experience. Some of the comments were “Really amazed by this concept and implementation”, “Your work is definitely a step in the right direction of software development future”, “It is a well developed creative idea that can be converted into a revolutionary product that helps millions of developers and give entry to non-developer into the development world”, and “As a developer I find it very easy to use Clowiz platform and a great opportunity to boost my development process”.

Moreover, some of the participants proposed to include new technologies, tool integration, and user-experience features. Proposed technologies included Angular, React, Microservices, and others. Features included integration with GitHub, Amazon AWS, and Docker. User experience suggestions included keyboard shortcuts, and lighter response by disabling the real-time code generation.

However, there were some comments about small glitches “As it is still in Alpha phase, there are some small glitches here and there,” so we worked on those bugs and solved them, such as the error when trying to delete a field from the FeatureGen application.

Chapter 8

Conclusion and Future Work

This chapter concludes the works presented in this dissertation, where the research question is revisited, an overall characterization of Smart-Cloud is presented, concluding remarks are provided, and future directions are identified.

8.1 Research Question Revisited

As discussed in Chapter 1, the research question of this dissertation is: *Is there an approach that enables the development of cloud-based applications to be: rapid, efficient, and straightforward?*

We believe that with Smart-Cloud we have been able to provide an approach that answers and solved this research question. In particular, we have been able to achieve rapidity by adopting the model-driven and metadata-driven development approaches. Moreover, efficiency was achieved by providing a comprehensive set of components that are reusable enough to eliminate the need of third-party libraries and APIs. Furthermore, straightforwardness was delivered by having a high-level abstraction and functionalities that reduce the learning curve and development time.

8.2 An Overall Characterization of Smart-Cloud

The Smart-Cloud framework can be characterized by being a cloud-based, reusable, model-driven, metadata-driven, and a product line approach.

The Smart-Cloud components are designed and developed following the twelve-

factor-app recommendation for modern cloud software architecture. Also, the framework provides transparent support of cloud functionality out of the box, such as the multi-tenant support and automatic binding of relational database resources on cloud environments. Moreover, the model-driven development functionality provided on the cloud by Clowiz platform is also deployed on the cloud.

Also, the framework consists of a novel combination of reusable API's, services, and application framework to increase developer's productivity on all levels.

Furthermore, it utilizes the model-driven development approach, where users can generate single artifacts, end-to-end features, or full applications using cloud-based design tools without writing code.

The code generation provided by the framework is based on the metadata-driven design, where the developers configure only the attributes and specifications of their artifacts or applications, then using the framework this metadata could be exported into the desired artifacts.

Finally, to reduce the time required for starting new projects, the framework was designed to provide a software product line approach for developers to enable a more robust creation of new projects.

8.3 Cloud Computing

Cloud computing was extensively discussed. The history and evolution of cloud computing were presented, and how the expensive hardware and infrastructure, along with the absence of economies of scale might be the main reasons for delaying its adoption. This was followed by how the Internet, low-cost commodity-computers based data centers, smart-phones, and economic crisis played essential roles in moving forward in cloud computing, and offering computer services as utilities. The main advantages of cloud computing were also presented, such as reducing the total cost of ownership, time to market, and liabilities delegation. On the other hand, disadvantages and challenges were discussed, such as security, loss of control, regulations and political conflicts. Moreover, the effect of cloud computing on startups, economic

disciplines, and hardware businesses were also examined.

Furthermore, the common service delivery models were presented, IaaS, PaaS, and SaaS. However, we think SaaS term is misused, and service delivery models require a standardized new taxonomy. In particular, software is a very generic term that includes operating systems, platforms, applications, and even virtualization technologies such as hypervisors. Consequently, all service delivery models are SaaS in some way. The main issue with this is the future regulations of taxation, billing, and licenses may be based on the categories of the software provided.

Moreover, cloud applications were presented, including a multi-tenancy taxonomy, application clients, and a comparison with on-premise applications. In addition, design, architecture, and process concepts and their rationale were discussed including the microservices architecture, native cloud applications, and DevOps. Also, benefits, challenges, and strategies for moving to applications on the cloud were overviewed.

To the best of our knowledge, the taxonomy of cloud applications presented in this dissertation: (i) Single-Instance Single-Tenant(SIST), (ii) Single-Instance Multi-Tenant(SIMT), (iii) Multi-instance Single-Tenant, and (iv) Multi-Instance Multi-tenant, is novel, and may open the door for new styles, patterns, and techniques and business models to build cloud applications.

Even though we think the background and the literature review presented in this dissertation is comprehensive enough as an introductory survey for cloud computing and cloud applications, we believe that having more details about the architectural styles and patterns for building cloud applications can also be beneficial for software architects and developers. Moreover, a further discussion about PaaS platforms will enable them to choose whether to build on one of the available options, build a new platform on top of an existing one, or even build a new domain-specific one.

The main issue with the currently available cloud computing offerings and technologies is the lack of standardization, which increases the risk of service provider lock. Even though this can be mitigated by creating an abstract layer between service users and providers, this increases the development cost, and may introduce

buggy features, and will not allow full utilization of services provided. In fact, we think Amazon is leading the de-facto standardization of cloud computing following the Dominant Design Concept [122]. However, this situation is risky since its long-term stability is not guaranteed and increasing the number of proprietary services, technologies, and protocols are more likely.

We believe that standardization is significant since it can address many constraints, risks, and challenges, and can enable more user traction. In addition, it may eliminate privacy and data constraints issues, and enable interoperability.

On the other hand, since the design and architecture of cloud applications is challenging and requires particular expertise, the utilization of native cloud platforms such as Pivotal Cloud Foundry [114] and the work presented in some reports [9, 99] may reduce the cost and enable proper utilization of cloud resources. In fact, PaaS worldwide spending is expected to increase from 11% in 2015 to 17% in 2020, which may be a sign for the need of supporting native cloud applications out of the box [66]. However, the low-maturity of this field and its lack of standardization open the door for future research on new designs and methods of native cloud application platforms.

8.4 Concluding Remarks and Future Work

In this dissertation, we presented Smart-Cloud, an application framework that aims to reduce the development cost of modern cloud-based software applications in general and data-intensive applications in particular, while maintaining a high level of quality and standardization. The framework consists of libraries, APIs, services, and end-user applications. These application utilize the model-driven development approach to enable rapid and efficient application development using the framework without writing code.

For evaluation and validation, a PMS case study of publication management system was presented and discussed, to ensure that the validation could be performed by following the AppGen process presented in Section 5.1.4. Also, a user study was conducted which involved 36 professionals from industry and academia. The goals of

the user study were to ensure that the problem we try to solve is significant, and to verify if we solve it right. The user study was an IRB approved by the University of Nevada, Reno (IRB #1362116-1). The results of the user study are promising, and the feedback from the participants contributed to identifying some future directions of this work.

Smart-Cloud can be the basis for future work in the cloud space and Internet-based applications. Some future directions may include the support for Big Data management and Big Data analysis web-based projects. In addition, it could be useful to enhance the framework to support applications that are not data intensive. For example, bringing rapid application development to domains such as Artificial Intelligence (AI), Virtual Reality (VR), Internet of Things (IoT), and game development.

Furthermore, enabling integration through standard interfaces can make Smart-Cloud interoperable and integrable with other systems.

Finally, supporting other technologies, such as Angular and React, and integration with cloud tools and services such as GitHub and AWS, can also be beneficial.

Appendix A

User Manual

This appendix includes the user guide of Clowiz platform. It consists of a description about Clowiz in general, followed by comprehensive information about by Clowiz applications (apps): CodeGen, FeatureGen, and AppGen.

A.1 Introduction

Clowiz is a cloud-based platform that aims to increase the productivity of teams working on developing software projects, and reducing the cost and time of developing high-quality software applications.

Clowiz utilizes the model-driven development approach, where users can generate and develop software artifacts using visual design tools without writing code. The code generation is based on metadata information provided by the platform's users.

- **CodeGen App:** An Online code generator that enables users to generate a single artifact unit of code at a time, such as Java class, HTML page, or MySQL SQL script. This app is designed to increase the productivity of software engineers, web developers, and database developers so that they can have their day to day artifacts generated rapidly without writing code.
- **FeatureGen App:** An online end-to-end code generator, that enables full-stack software developers to develop all the artifacts required for a fully functional page. FeatureGen app follows the MVC design pattern, where it generates the code for the view (page), model, controller, and database script.

- **AppGen App:** AppGen is an online application development app, that enables technical and non-technical people to develop software applications without writing code. With AppGen, users can create applications, design the artifacts, preview a full prototype, and download the end-to-end full source code for their projects, without writing a single line of code. With AppGen, designing applications is a straightforward process, which includes, configuring the application modules (sub-apps), page-groups (menus), and pages (views). Designing the pages is also as simple as configuring the fields of that page (metadata).

To evaluate Clowiz apps, or creating quick proof of concepts, users can work in the *Guest Mode* without creating accounts or signing in. However, in this case, nothing of their work will be saved.

It is highly recommended that every user login to the platform before start creating artifacts or apps, especially in the AppGen app.

The following sub-sections include the details of the applications of Clowiz.

A.2 Home

Clowiz Home page is the main entry for the platform, which can be accessed at <http://www.clowiz.com> URL.

As shown in Figure A.1, the home page of Clowiz consists of links for the primary platform's applications (apps), which are:

- CodeGen App
- FeatureGen App
- AppGen App

IMPORTANT: As discussed in the Introduction section, it is highly recommended that Authentication App to the platform before doing any serious development, to

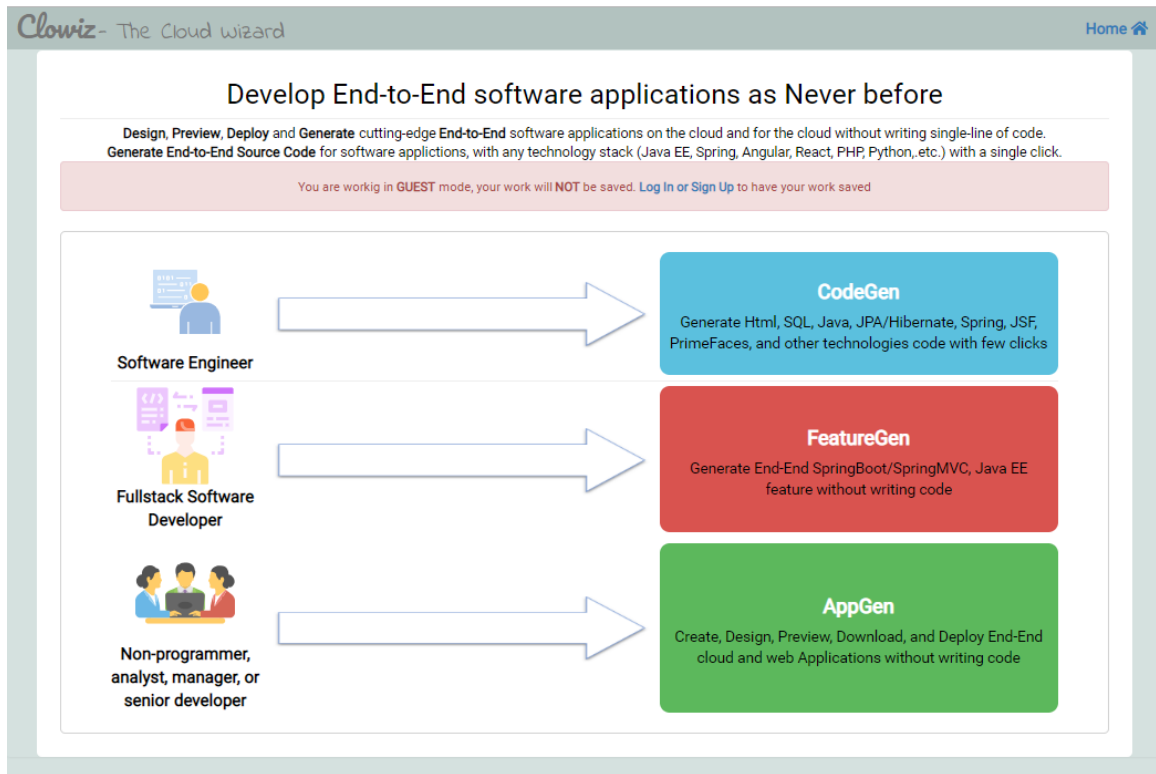


Figure A.1: Clowiz Home Page

avoid losing their work. The *Guest Mode* was only created to enable users to experiment and to do quick proofs-of-concept on the platform, and it is not intended for production work.

A.3 Authentication App

In the Authentication App of Clowiz, users can Sign Up, Reset Password, Log In, as shown in Figure A.2. To access the authentication app, navigate to <https://www.clowiz.com/login>.

A.3.1 Sign Up

To be able to save their work and get the full benefits of the platform, users shall have an account in Clowiz, and Log In to the system.

- Click on **Create an Account**

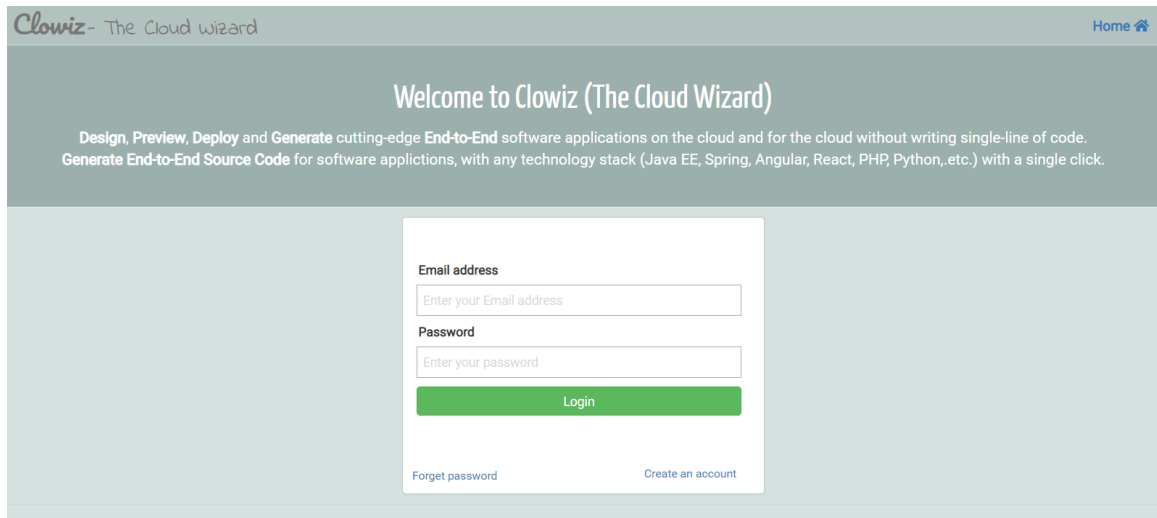


Figure A.2: Authentication App

- Fill the required information (Email, First name, and Last name) as shown in Figure A.3.

The image shows a "Create account" form. The form is titled "Create account" and has a close button (X) in the top right corner. It contains three required input fields: "Email *", "First name *", and "Last name *". Each field is represented by a text input box. Below the input fields is a blue "Create account now" button. At the bottom of the form, there is a message: "Yes, it's that simple!".

Figure A.3: Create an Account

- Check your Email As shown in Figure A.4, after completing the required information, an email contains the password will be delivered to the specified

Email.

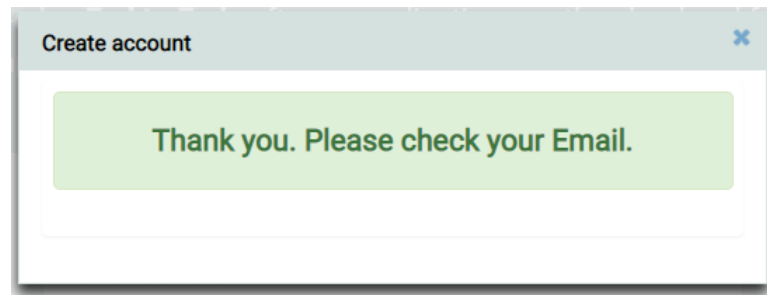


Figure A.4: Sign Up Confirmation

A.3.2 Reset Password

If a user forgets his/her password, they can click on Forgot Password link in the Authentication app, enter their email address, then click on the 'Next' button to reset their passwords, as shown in Figure A.5. If the email address already exists as a registered account, a new password will be created and sent to the email; otherwise, the system will show “We cannot find this Email in our records. Create an Account?” message.

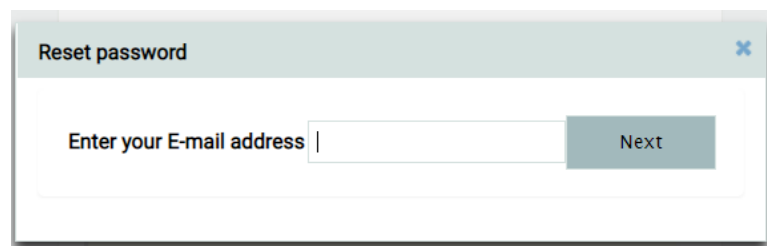


Figure A.5: Reset Password Form

A.3.3 Log In

To login to the system, in the Authentication app, the user should enter his/her email and password, then click **Login**, as shown in Figure A.6.

The image shows a login form with a light green border. It contains the following elements:

- Email address**: A label above a text input field with the placeholder text "Enter your Email address".
- Password**: A label above a text input field with the placeholder text "Enter your password".
- Login**: A prominent green button with white text.
- Forget password**: A blue text link at the bottom left.
- Create an account**: A blue text link at the bottom right.

Figure A.6: Login Form

A.4 CodeGen App

CodeGen is an app that generates source code artifacts for different technologies using the model-driven development approach. To access the CodeGen app, navigate to <https://www.clowiz.com/code-generator>.

TIP: The artifacts created by FeatureGen is just for ad-hoc needs. To have the work saved and organized in a more modular way, users should consider using AppGen App. As shown in Figure A.7, the view consists of four main components:

- The *Metadata* on the left side.
- The *Technologies* section on the center top side.
- The *Exporters* of the selected technology.
- The *Generated Code* in the middle center of the view.

TIP: CodeGen does not provide a full project structure and configurations, so users will need to include the generated artifacts into their projects manually. For a more robust and comprehensive option, users may consider using AppGen App to create a fully configured end-to-end applications. CodeGen automatically generates code based on the information that the users configured in the metadata data section in the left side of the page.

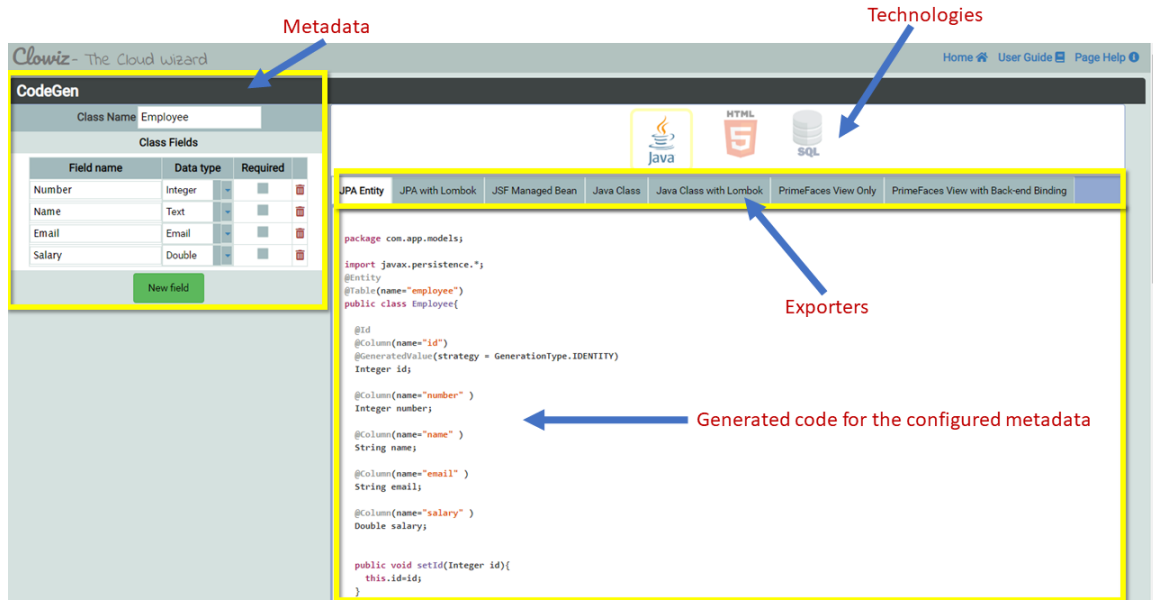


Figure A.7: CodeGen App Page

A.4.1 CodeGen Metadata

As shown in Figure A.8, the metadata required to generate source code in CodeGen is quite simple.

- **[Unit] name:** is the name of the unit to be generated, which depends on the selected technology. For example, if the selected exporter is Java, the unit name will be the Class name, and if the selected exporter is MySQL, then the unit name will be the table name.
- **Fields:** the fields are the part of the unit to be generated. The actual meaning of these fields depends on the selected exporter, for example, in Java classes, this means the instance variables, while in the HTML exporters, it means the HTML input fields and their assigned labels.

Add Field

To add new fields to the metadata, click on the **New Field** button, this will automatically add a field to the list with the name automatically set on that field.

Class Name			
Clowiz Code			
Class Fields			
Field name	Data type	Required	
First name	Text	<input type="checkbox"/>	
Last name	Text	<input type="checkbox"/>	
Email	Email	<input type="checkbox"/>	
Salary	Double	<input type="checkbox"/>	
<input type="button" value="New field"/>			

Figure A.8: CodeGen metadata

Modify Field

To change the field name, just change the value directly in the input box that contains the current field name. In addition, a user can change the field data type, and configures whether it is required or not.

Delete Field

To delete a field, click on the trash icon on the last column in the metadata table.

A.4.2 Fields Data Types

Every field should have a data type that will be used by the exporter to generate the code correctly. By default the data type of the fields is text.

- Text
- Integer
- Float
- Double

- Boolean
- Binary
- Month
- Time
- Timestamp
- Date
- Email
- Password
- Telephone
- URL

A.4.3 Technologies

CodeGen supports different technologies that can be selected by the platform's user. Every technology consists of different exporters (generators), that can generate software artifacts for the configured metadata. Currently, CodeGen supports the Java, HTML, and SQL technologies.

Java

Java technology support in CodeGen consists of the following exporters:

Java Class Exporter This exporter generates java class based on the metadata, where the fields in the metadata represent the instance variables in the Java class.

As shown in Figure A.9, the generated code follows the best practices of java naming and convention. In addition, it follows the best practices of encapsulation by restricting access to the instance variables and expose them using setters and getters.

The screenshot shows the Clowiz CodeGen interface. On the left, the 'Class Name' is 'Employee'. Below it, a table lists 'Class Fields':

Field name	Data type	Required
First name	Text	<input type="checkbox"/>
Last name	Text	<input type="checkbox"/>
Email	Email	<input type="checkbox"/>
Salary	Double	<input type="checkbox"/>

Below the table is a 'New field' button. The main area shows the 'Java Class' exporter selected. The generated code is:

```
package com.app.models;

public class Employee{

    Integer id;
    String firstName;
    String lastName;
    String email;
    Double salary;

    public void setId(Integer id){
        this.id=id;
    }

    public Integer getId(){
        return this.id;
    }

    public void setFirstName(String firstName){
        this.firstName=firstName;
    }

    public String getFirstName(){
        return this.firstName;
    }
}
```

Figure A.9: Java Class Exporter in CodeGen

Java Class with Lombok Exporter This exporter generates java class with Lombok support based on the metadata, where the fields in the metadata represent the instance variables in the Java class. As shown in Figure A.10, the generated code does not include the setters and getters for the instance variables because Lombok will generate them for free during the compilation phase.

The screenshot shows the Clowiz CodeGen interface. On the left, the 'Class Name' is 'Employee'. Below it, a table lists 'Class Fields':

Field name	Data type	Required
First name	Text	<input type="checkbox"/>
Last name	Text	<input type="checkbox"/>
Email	Email	<input type="checkbox"/>
Salary	Double	<input type="checkbox"/>

Below the table is a 'New field' button. The main area shows the 'Java Class with Lombok' exporter selected. The generated code is:

```
package com.app.models;

import lombok.Data;
@Data
public class Employee{

    Integer id;
    String firstName;
    String lastName;
    String email;
    Double salary;
}
```

Figure A.10: Java Class with Lombok Exporter in CodeGen

JPA Entity Exporter JPA Entity exporter exports Java classes including the full annotation required by Java Persistence API, which could be used to persist objects of this class to a relational database.

JPA Entity exporter is shown in Figure A.11.

The screenshot shows the Clowiz CodeGen interface. At the top, it says "Clowiz - The Cloud Wizard" and "Home". Below that, the "CodeGen" section is active. The "Class Name" is "Employee". Under "Class Fields", there is a table with the following data:

Field name	Data type	Required
First name	Text	<input type="checkbox"/>
Last name	Text	<input type="checkbox"/>
Email	Email	<input type="checkbox"/>
Salary	Double	<input type="checkbox"/>

Below the table is a "New field" button. To the right, there are icons for Java, HTML, and SQL. Below these icons, there are tabs for "JPA Entity", "JPA with Lombok", "JSF Managed Bean", "Java Class", "Java Class with Lombok", and "PrimeFaces View Only". The "JPA Entity" tab is selected, and the generated code is displayed in a text area:

```
package com.app.models;

import javax.persistence.*;
@Entity
@Table(name="employee_")
public class Employee{

    @Id
    @Column(name="id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer id;

    @Column(name="first_name" )
    String firstName;

    @Column(name="last_name" )
    String lastName;

    @Column(name="email" )
    String email;

    @Column(name="salary" )
    Double salary;
}
```

Figure A.11: JPA Entity Exporter in CodeGen

JPA Entity with Lombok Exporter JPA with Lombok exporter exports the entity into JPA entity with Lombok support, which can reduce the lines of code dramatically by eliminating the need of writing setters and getters. Figure A.12 shows JPA with Lombok exporter.

JSF Managed Bean Exporter JSF Managed Bean exporter exports the entity into annotated JSF managed bean Java class that extends *JKManagedBeanWithOrmSupport* class which is provided by the Smart-Cloud framework, which includes all the required functionalities to enable high-quality CRUD operation in the view. The exporter is shown in Figure A.13.

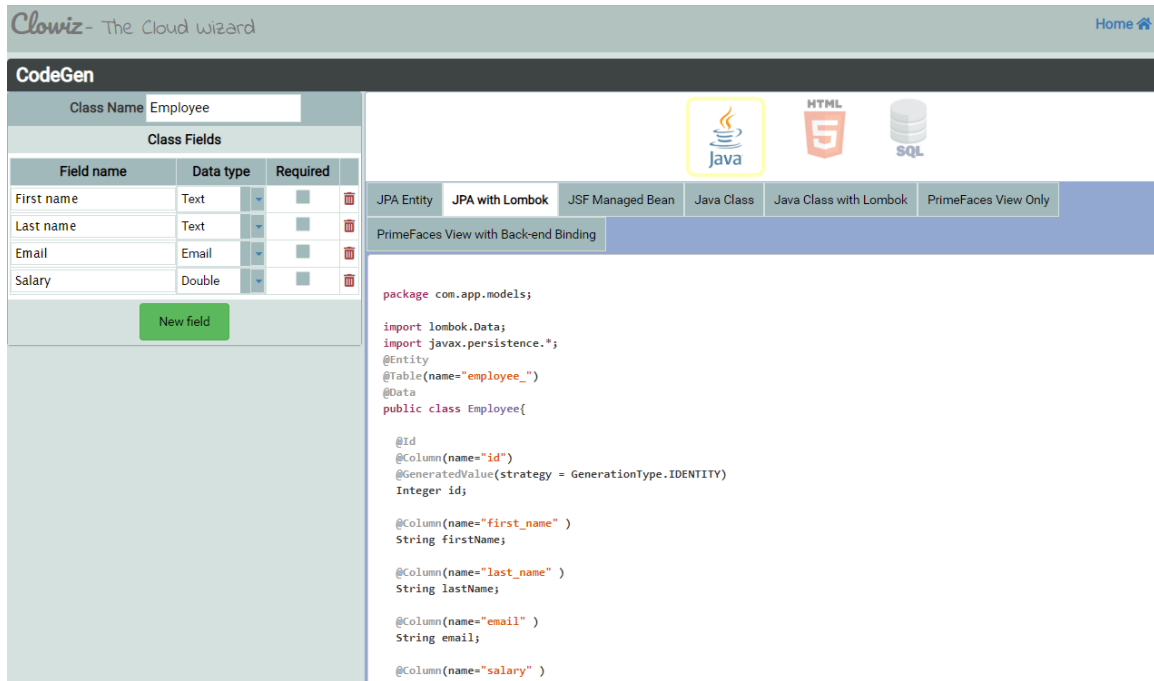


Figure A.12: JPA Entity with Lombok Exporter in CodeGen

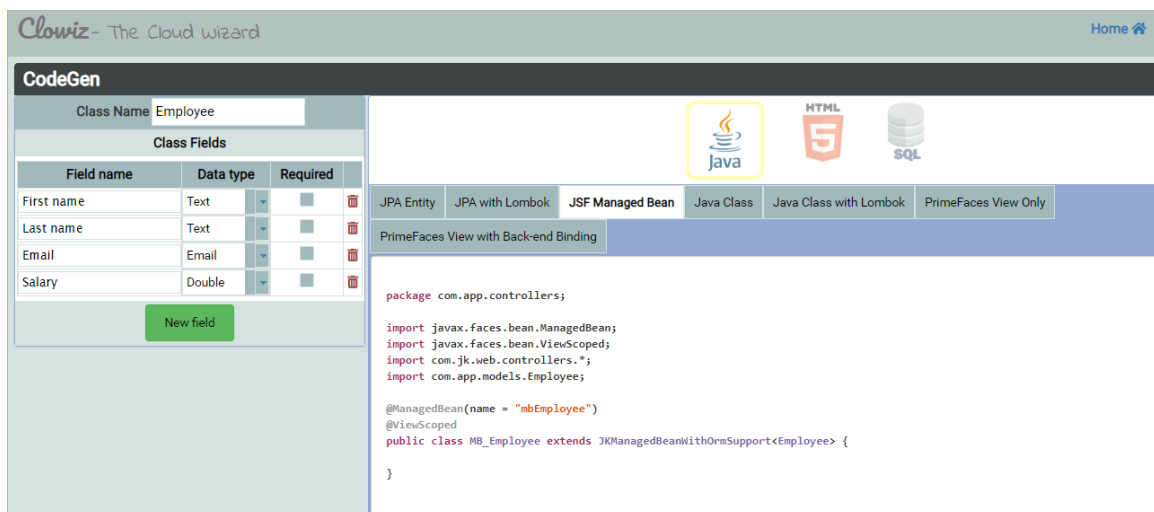


Figure A.13: JSF Managed Bean Exporter in CodeGen

PrimeFaces View Only Exporter This exporter used to generate PrimeFaces view code only. PrimeFaces is a web-based user interface widgets that enable rich user experience for JSF based applications. As shown in Figure A.14, this exporter does not include binding the user interface components with the backend model or controller.

The screenshot shows the Clowiz CodeGen interface. On the left, the 'View Name' is 'Employee'. Below it, a table lists 'View Fields':

Field name	Data type	Required
First name	Text	<input type="checkbox"/>
Last name	Text	<input type="checkbox"/>
Email	Email	<input type="checkbox"/>
Salary	Double	<input type="checkbox"/>

Below the table is a 'New field' button. The main area shows the 'PrimeFaces View with Back-end Binding' selected. The generated code is as follows:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html" xmlns:p="http://primefaces.org/ui" xmlns:
"http://xmlns.jcp.org/jsf/facelets"
xmlns:c="http://xmlns.jcp.org/jsf/jstl/core" xmlns:f="http://xmlns.jcp.org/jsf/core">
<ui:composition>
<h:form>
<p:autoUpdate />
<p:messages />
<p:panelGrid columns="4" id="model" style="margin:auto">
<f:facet names="header">Employee Form</f:facet>
<p:outputLabel value="First name" for="firstName"/>
<p:inputText type="text" id="firstName" placeholder="Enter value"/>

<p:outputLabel value="Last name" for="lastName"/>
<p:inputText type="text" id="lastName" placeholder="Enter value"/>

<p:outputLabel value="Email" for="email"/>
<p:inputText type="email" id="email" placeholder="Enter value"/>

<p:outputLabel value="Salary" for="salary"/>
<p:inputText type="number" id="salary" placeholder="Enter value"/>
</p:panelGrid>
</h:form>
</ui:composition>
</html>
```

Figure A.14: PrimeFaces View Exporter in CodeGen

PrimeFaces with Backend Exporter PrimeFaces exporter with Backend binding includes generating PrimeFaces view and the binding code with the model and the backend. As shown in Figure A.15.

The screenshot shows the Clowiz CodeGen interface. On the left, the 'View Name' is 'Employee'. Below it, a table lists 'View Fields':

Field name	Data type	Required
First name	Text	<input type="checkbox"/>
Last name	Text	<input type="checkbox"/>
Email	Email	<input type="checkbox"/>
Salary	Double	<input type="checkbox"/>

Below the table is a 'New field' button. The main area shows the 'PrimeFaces View with Back-end Binding' selected. The generated code is as follows:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html" xmlns:p="http://primefaces.org/ui" xmlns:ui=
"http://xmlns.jcp.org/jsf/facelets"
xmlns:c="http://xmlns.jcp.org/jsf/jstl/core" xmlns:f="http://xmlns.jcp.org/jsf/core">
<ui:composition>
<h:form id="frmEmployee">
<p:autoUpdate />
<p:messages />
<p:panelGrid columns="4" id="model" style="margin:auto">
<f:facet name="header">#{msg.get('Employee')} Form</f:facet>
<p:outputLabel value="#{msg.get('First name')}" for="firstName"/>

<p:inputText type="text" id="firstName" value="#{mbEmployee.model.firstName}" disabled="#{mbEmployee.readOnlyMode}" readOnly=
"#{mbEmployee.readOnlyMode}" placeholder="Enter value"/>

<p:outputLabel value="#{msg.get('Last name')}" for="lastName"/>

<p:inputText type="text" id="lastName" value="#{mbEmployee.model.lastName}" disabled="#{mbEmployee.readOnlyMode}" readOnly=
"#{mbEmployee.readOnlyMode}" placeholder="Enter value"/>

<p:outputLabel value="#{msg.get('Email')}" for="email"/>

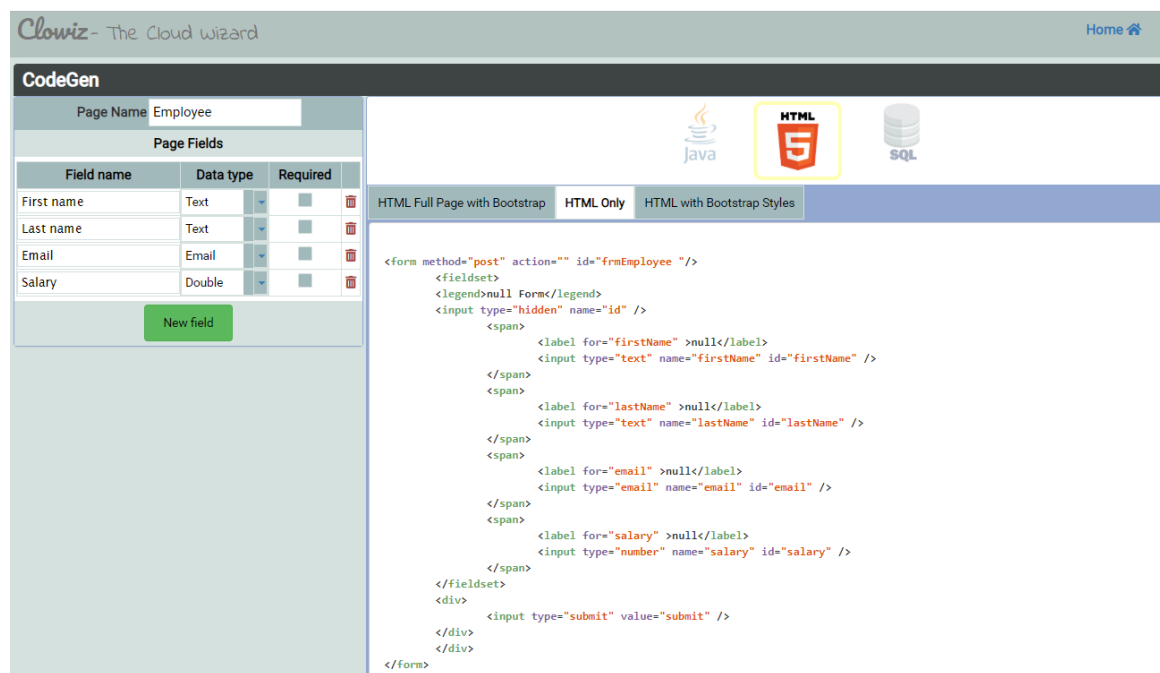
<p:inputText type="email" id="email" value="#{mbEmployee.model.email}" disabled="#{mbEmployee.readOnlyMode}" readOnly=
"#{mbEmployee.readOnlyMode}" placeholder="Enter value"/>
</p:panelGrid>
</h:form>
</ui:composition>
</html>
```

Figure A.15: PrimeFaces with Backend Binding Exporter in CodeGen

HTML

The HTML technology in CodeGen includes exporters for: HTML Only, HTML with Bootstrap Styles, and HTML full page with Bootstrap.

HTML Only Exporter In this exporter, CodeGen will generate the required input HTML fields based on the field's metadata data type. The HTML input tags will be surrounded by fieldset and form tags. The HTML Only exporter is shown in Figure A.16.



The screenshot shows the CodeGen interface with the 'Page Name' set to 'Employee'. The 'Page Fields' table is as follows:

Field name	Data type	Required
First name	Text	<input checked="" type="checkbox"/>
Last name	Text	<input checked="" type="checkbox"/>
Email	Email	<input checked="" type="checkbox"/>
Salary	Double	<input checked="" type="checkbox"/>

The 'HTML Only' exporter is selected, and the generated HTML code is shown below:

```
<form method="post" action="" id="frmEmployee ">
  <fieldset>
    <legend>null Form</legend>
    <input type="hidden" name="id" />
    <span>
      <label for="firstName" >null</label>
      <input type="text" name="firstName" id="firstName" />
    </span>
    <span>
      <label for="lastName" >null</label>
      <input type="text" name="lastName" id="lastName" />
    </span>
    <span>
      <label for="email" >null</label>
      <input type="email" name="email" id="email" />
    </span>
    <span>
      <label for="salary" >null</label>
      <input type="number" name="salary" id="salary" />
    </span>
  </fieldset>
  <div>
    <input type="submit" value="submit" />
  </div>
</form>
```

Figure A.16: HTML Only Exporter in CodeGen

HTML with Bootstrap Exporter This exporter generates a complete HTML form with its input components and labels components attached to these input fields. In addition, it includes the CSS styles of the Bootstrap framework on every component as needed. HTML with Bootstrap Exporter in CodeGen shows in Figure A.17.

HTML Full Page with Bootstrap Exporter This exporter generates full HTML page the includes:

The screenshot shows the Clowiz CodeGen interface. On the left, the 'Page Name' is 'Employee'. Below it, a table lists 'Page Fields':

Field name	Data type	Required
First name	Text	<input type="checkbox"/>
Last name	Text	<input type="checkbox"/>
Email	Email	<input type="checkbox"/>
Salary	Double	<input type="checkbox"/>

Below the table is a 'New field' button. On the right, there are icons for Java, HTML, and SQL. A navigation bar at the top right has 'Home' and a 'CodeGen' header. Below the navigation bar, three tabs are visible: 'HTML Full Page with Bootstrap', 'HTML Only', and 'HTML with Bootstrap Styles' (which is selected). The main area displays the generated HTML code for the form:

```
<form method="post" action="" id="frmEmployee" />
<div class="Employee card" style="width:500px; margin:auto;">
<div class="card-header">null Form</div>
<div class="card-body">
<input type="hidden" name="id" />
<span class="form-group">
<label for="firstName" >null</label>
<input type="text" name="firstName" id="firstName" class="form-control" />
</span>
<span class="form-group">
<label for="lastName" >null</label>
<input type="text" name="lastName" id="lastName" class="form-control" />
</span>
<span class="form-group">
<label for="email" >null</label>
<input type="email" name="email" id="email" class="form-control" />
</span>
<span class="form-group">
<label for="salary" >null</label>
<input type="number" name="salary" id="salary" class="form-control" />
</span>
</div>
<div class="card-footer">
<input type="submit" value="submit" />
</div>
</form>
```

Figure A.17: HTML with Bootstrap exporter

- Bootstrap framework.
- JQuery framework.
- Bootstrap theme.
- Fonts Awesome library.
- Selected Google Fonts.
- Complete HTML form for the configured metadata.

HTML Full Page with Exporter is shown in Figure A.18.

SQL

The SQL technology section consists of exporters that generate scripts for various relational database engines such as MySQL, H2, SQL Server, PostgreSQL, and Oracle.

The screenshot shows the Clowiz CodeGen interface. The page name is "Employee". The "Page Fields" table is as follows:

Field name	Data type	Required
First name	Text	<input type="checkbox"/>
Last name	Text	<input type="checkbox"/>
Email	Email	<input type="checkbox"/>
Salary	Double	<input type="checkbox"/>

The "HTML Full Page with Bootstrap" exporter is selected, and the generated HTML code is displayed in the right pane. The code includes Bootstrap and Font Awesome links and a form for the Employee page.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">
  <title>Clowiz Pages</title>

  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" rel="stylesheet">
  <link href="https://stackpath.bootstrapcdn.com/bootswatch/4.1.3/simplex/bootstrap.min.css" rel="stylesheet">
  <link href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css" rel="stylesheet">

  <link href="https://fonts.googleapis.com/css?family=Lato:300,400,700,300italic,400italic,700italic" rel="stylesheet" type="text/css">
</head>
<body>
  <form method="post" action="" id="frmEmployee ">
    <div class="Employee card" style="width:500px; margin:auto;">
      <div class="card-header"><null Form</div>
      <div class="card-body">
        <input type="hidden" name="id" />
        <span class="form-group">
          <label for="firstName" >null</label>
          <input type="text" name="firstName" id="firstName" class="form-control" />
        </span>
      </div>
    </div>
  </form>
</body>
</html>
```

Figure A.18: HTML Full Page with Bootstrap Exporter in CodeGen

H2 Database Exporter This exporter will generate the script that creates a database table for H2 database, which is a lightweight database engine that is heavily used in proofs-of-concept, unit testing, and integration testing, as shown in Figure A.19.

The screenshot shows the Clowiz CodeGen interface. The table name is "Clowiz Code". The "Table Fields" table is as follows:

Field name	Data type	Required
first name	Text	<input type="checkbox"/>
last name	Text	<input type="checkbox"/>
email	Email	<input type="checkbox"/>
salary	Double	<input type="checkbox"/>

The "H2 SQL Structure" exporter is selected, and the generated SQL code is displayed in the right pane. The code creates a table named "clowiz_code" with the following structure:

```
create table "clowiz_code"(
  "id" integer generated by default as identity,
  "first_name" varchar(250),
  "last_name" varchar(250),
  "email" varchar(250),
  "salary" double,
);
```

Figure A.19: H2 Database Exporter in CodeGen

MySQL Database Exporter This exporter generates a create table statement for the configured metadata with the specific syntax of MySQL database. MySQL exporter is shown in Figure A.20.

The screenshot shows the CodeGen interface for the MySQL Database Exporter. The table name is 'Clowiz Code'. The table fields are:

Field name	Data type	Required
first name	Text	<input type="checkbox"/>
last name	Text	<input type="checkbox"/>
email	Email	<input type="checkbox"/>
salary	Double	<input type="checkbox"/>

The generated SQL code for MySQL is:

```
create table `clowiz_code` (
  `id` integer not null auto_increment,
  `first_name` varchar(250),
  `last_name` varchar(250),
  `email` varchar(250),
  `salary` double precision,
  primary key (id)
);
```

Figure A.20: MySQL Exporter in CodeGen

Microsoft Database Exporter The exporter is generating the SQL script for Microsoft SQL server database. It generates full create table statement, as shown in Figure A.21.

The screenshot shows the CodeGen interface for the Microsoft Database Exporter. The table name is 'Clowiz Code'. The table fields are:

Field name	Data type	Required
first name	Text	<input type="checkbox"/>
last name	Text	<input type="checkbox"/>
email	Email	<input type="checkbox"/>
salary	Double	<input type="checkbox"/>

The generated SQL code for Microsoft SQL Server is:

```
create table [clowiz_code](
  [id] int identity not null,
  [first_name] varchar(250),
  [last_name] varchar(250),
  [email] varchar(250),
  [salary] double precision,
  primary key (id)
);
```

Figure A.21: Microsoft SQL Server Exporter in CodeGen

Oracle Database Exporter Oracle database exporter used to generate the script that is compatible with Oracle database. It generates full create table statement, as shown in Figure A.22.



Figure A.22: Oracle Exporter in CodeGen

A.5 FeatureGen App

FeatureGen is one of the core apps in Clowiz platform that generates end-to-end artifacts for a page. It can be accessed at <https://www.clowiz.com/feature-generator>.

FeatureGen does not provide a full project structure and configurations, so users will need to include the generated artifacts into their projects manually. For a more robust and comprehensive option, users may consider using AppGen App to create a fully configured end-to-end applications. The usage of FeatureGen is as simple as configuring that fields (metadata) that compose the page (view), then a realtime preview for the page is shown in the middle of the page, and the full generated source code artifacts are shown in the bottom of the view.

As shown in Figure A.23, the FeatureGen page consists of the following sections:

- Metadata.
- Technology Stack.
- Page Preview.
- Generated code.

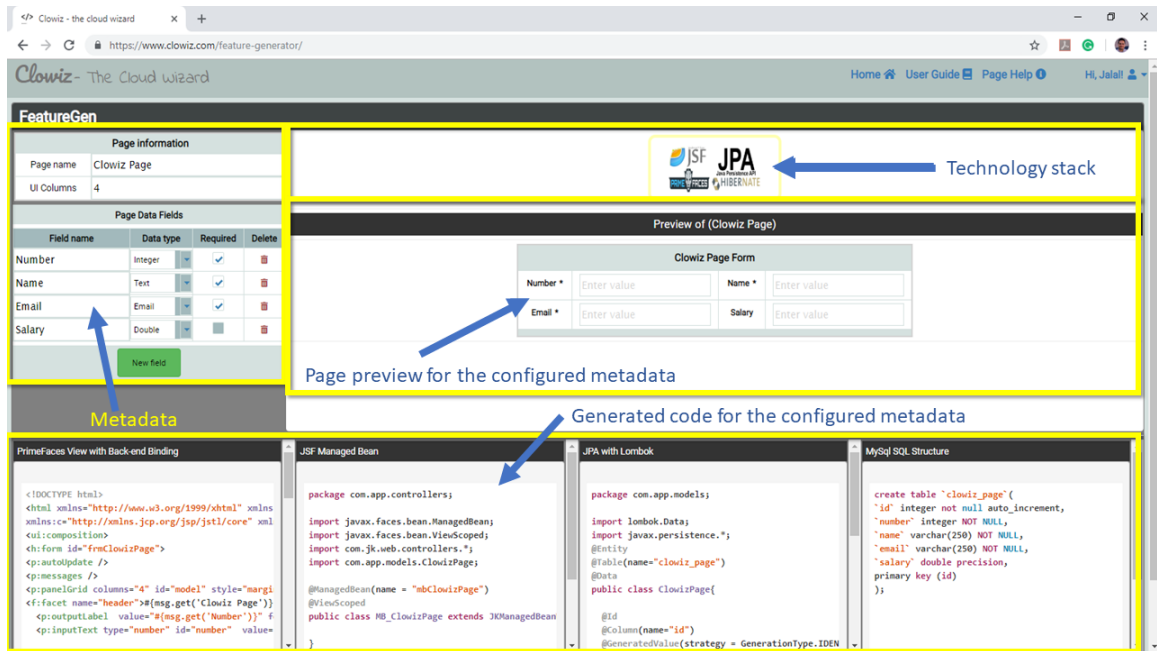


Figure A.23: FeatureGen Page

A.5.1 FeatureGen Metadata

As shown in Figure A.24, the metadata required to generate source code in FeatureGen is quite simple.

- **Page name:** is the name of the page to be generated.
- **UI Columns:** is the number of columns to render the user interface. This number should consider the labels as well. For example, if the page only has one field, and the desired output is to have the label next to the input field on the same row, then the number of columns should be 2, but if the desired output is to have the label above the field, then the value should be one.
- **Fields:** the input fields of the desired page.

Add Field

To add new fields to the metadata, click on the **New Field** button, this will automatically add a field to the list with the name automatically set on that field.

Page information			
Page name	Clowiz Page		
UI Columns	4		
Page Data Fields			
Field name	Data type	Required	Delete
Number	Integer <input type="text"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Name	Text <input type="text"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Email	Email <input type="text"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Salary	Double <input type="text"/>	<input type="checkbox"/>	<input type="checkbox"/>
<input type="button" value="New field"/>			

Figure A.24: FeatureGen Metadata

Modify Field

To change the field name, just change the value directly in the input box that contains the current field name. In addition, a user can change the field data type, and configures whether it is required or not.

Delete Field

To delete a field, click on the trash icon on the last column in the metadata table.

A.5.2 Fields Data Types

Every field should have a data type that will be used by the exporter to generate the code correctly. By default the data type of the fields is text.

A.5.3 FeatureGen Technology

The technology stack section in the FeatureGen is used to determine the exporters (generators) that will be used to generate the final artifacts.

The currently supported technology stacks in Clowiz platform are:

- **JavaEE Web:** JSF, PrimeFaces, JPA with Hibernate.
- **Spring Framework[In Progress]:** SpringBoot, SpringMVC, SpringData, and JPA/Hibernate.

A.5.4 Feature Page Preview

This section will enable the users to view the exact final pages prototype that will be generated by FeatureGen, with the default style provided by Clowiz.

The preview section is intended to be a prototype preview for the end users, and does not provide a functional user-interface.

A.5.5 FeatureGen Generated Code

This section shows the full artifacts generated for this page. Since the FeatureGen app follows the MVC design pattern, the generated code will include the view (page), model, controller, and the database script which is required for the relational database engine configured in the technology stack.

The artifacts created by FeatureGen is just for ad-hoc needs. To have the work saved and organized in a more modular way, users should consider using AppGen App.

A.6 AppGen App

The AppGen app of Clowiz platform is a new innovative way of building cloud-based software applications. It enables the model-driven approach, where users use graphical modeling tools to create and develop software applications. AppGen can be accessed at: <https://www.clowiz.com/app-manager>.

- **App Manager.**
- **App Designer**

- **Page Designer**

A.6.1 App Manager

The App Manager is the main entry of the AppGen. As shown in Figure A.25, the App Manager consists of the following:

- **Create New Software Applications:** this action is used to create a new application. The new application will be shown in the *User's Applications* area. The status of the application will be Development Status as discussed in AppGen Application Status. The application will be named *Application X* where X represents the number of current applications +1 of the user.
- **User's applications:** this represents the list of the current applications that were created by the user.

Note: If the user works in the guest mode, the list shows the applications that marked as **Allow Public Access** by other users. Any applications created by the user in this mode will be discarded when the user navigates to the other views.

The applications in the AppGen will have an assigned status that will be shown next to or under the application name in the AppGen home page.

- **Development Status:** indicates that the project is still under continuous development.
- **Deployment Status:** indicates that the applications are deployed on the cloud using AppDeployer App.
- **Archiving Status:** indicates that the project is archived and is not under active development.

The available functionalities for every application depend on the status. In general, users can *Design, Preview, Download Source Code, Deploy, Archive, UnArvice,* or *Delete projects.*

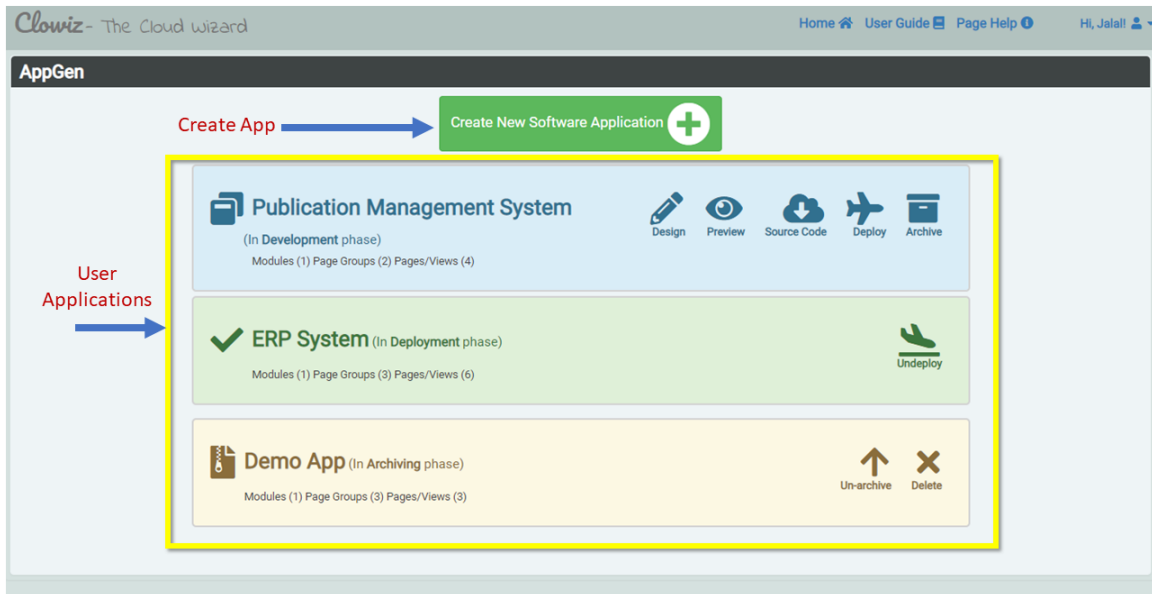


Figure A.25: Clowiz AppGen Home Page

Design Application

This action launches the App Designer for designing the high-level artifacts of the app.

Preview Application

This action launches a complete none-functional prototype of the application. The prototype includes the home page, the menu-bar, and all the pages design.

Download Source Code

This downloads a complete, end-to-end, web-based, software project source-code package. The package will be downloaded as a compressed package.

- **Main:** a launcher class that enables running the application without the need of web server.
- **Project Configuration:** a Maven POM file, which is a standard project configuration for the Java technology, so that it can be imported with any Java

IDE.

- **Pages:** a generated JSF view code for all the pages in the application.
- **Models:** a generated JPA entities code for all the pages.
- **Controllers:** a generated JSF managed bean classes which act as controllers for the views.
- **Theme:** a complete theme of the web applications based on PrimFaces, Bootstrap, Fonts-Awesome, and others.
- **Template:** a Facelet template, which is based on the standard template engine provided by the JSF framework. The views extensively use PrimeFaces widget libraries.
- **Logging Configurations:** configuration for SLF4J logger.
- **Configuration:** a configuration file for the database.

NOTE: The downloaded application does not require a separate database or a web server installation, embedded server and database engines are provided out of the box. **TIP:** The package utilizes some components from the Smart-Cloud framework, so compatibility with cloud providers is provided out of the box. Currently, the framework supports Pivotal Cloud Foundry (PCF) environment.

Deploy App

This action launches the application to the cloud using AppDeployer App.

Archive App

This action changes the status of the app to *Archiving* status.

UnArchive App

This action changes the status of the app to *Development* status.

Delete App

This action deletes the app permanently. This action is only available in the archive status.

A.6.2 App Designer

The App Designer is one of the pages of AppGen app of Clowiz platform; it enables users to configure the required high-level attributes of their applications using cloud-based design tool, as shown in in Figure A.26.

- **Application name:** which is an input text field that allows users to change the application name.
- **Actions:** includes the actions that users can do in this page, which includes: *Save Changes*, *Save Changes and Back*, or *Ignore Changes and Back*.
- **Advanced Mode:** a checkbox that enables the advanced features which includes:
 - **The Multi-Module support:** where users can divide the application into subsystems (modules).
 - **Advanced Attributes:** which includes the advanced attributes of the currently selected application.
- **Page-Groups:** this section includes the page groups (menus) of the application. Pages group will be rendered as the main menu in the final generated code.
- **Pages:** the page component represents the page that will be generated under this page group (menu). This page will be rendered as a menu item under the page group.

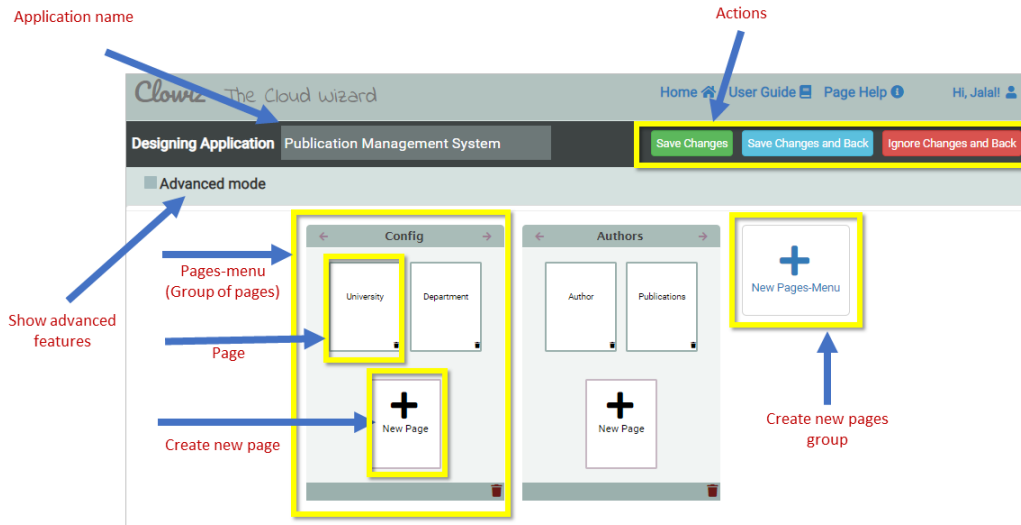


Figure A.26: AppGen App Designer

Create Page Group (Menu)

To create a page group, users need to click on the *New Pages-Group* button in the center of the page. This will create a new page group with a name automatically set. The group name can be changed by clicking on the header of the group and edit the name. This will make this page group appears as menu-bar inside the application menu bar.

Create new Page

To create a new page, users should click on a *New Page* button inside any page group; this will create a new page inside that group. This will add a menu item under the page-group menu in the final application menu bar.

Clicking on any page will open the Page Designer view for that page, which is discussed in the following section.

A.6.3 Page Designer

The page designer view, which is shown in Figure A.27, is the view that enables users to design the contents of any page without writing code. To design a page can be opened by clicking on any page in the App Designer.

- **Metadata:** this section includes the metadata required for designing the page.
- **Page Preview:** which shows the realtime preview for the page. The preview includes two parts, *Form Preview* and *Data-table Preview*.
- **Generated Code:** which is a checkbox the enables user to preview the code that will be generated for this page.

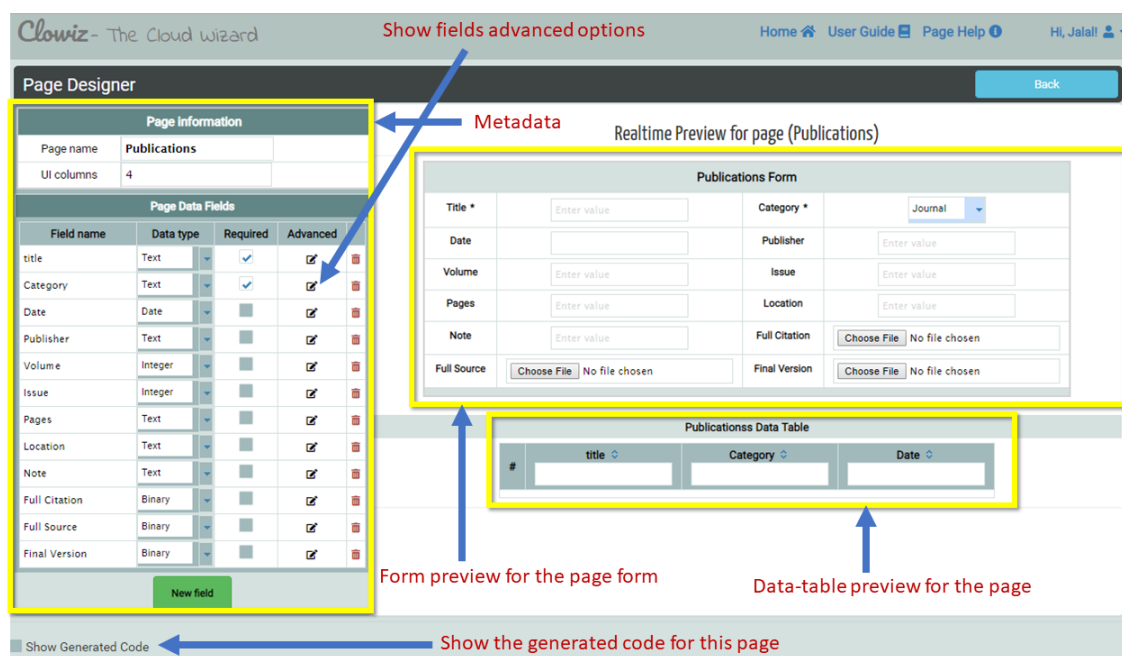


Figure A.27: AppGen Page Designer

The metadata section is the part which determines how the code will be generated for this page. The metadata includes page-level metadata, and the fields metadata.

- **Page name:** which will be used in the menu item, in the page title, and all the other artifacts names.

- **UI Columns:** which determines the number of user interface columns for the generated form.

The fields metadata include the configurations of every field individually.

For every field, there are core properties that are shown directly on the front of the Page Designer, and there are more advanced metadata that can be shown by clicking on the advanced edit icon in the field row in the fields metadata table.

Table A.1 shows the list of the field attributes with their description.

When the user is done with modifying the page, he/she can click on the *Back* button to get back to the App Designer.

Table A.1: Fields metadata attributes

	Property Name	Description
1	Name	Field name
2	Data Type	Field datatype
3	Required	Required or optional
4	Enabled	Enabled or disabled in the user interface
5	Updatable	Updatable
6	Main field	Main field of the entity generated for this page (e.g., in Java, this will be part of the toString() method). This can be helpful if this page is used as a reference in other pages
7	List of values	Comma separated values that will be rendered as a multi-choice select box, for example, "A,B,C"
8	Default value	The default value for this field
9	Visible	Determines whether to show this field on the user interface or not
10	Show in Datatable	Determines whether to show in the generated datatable or not
11	Max length	The maximum length for this field
12	Width	The width of this field in the user interface
13	Height	The height of this field in the user interface
14	Background color	The background color of this field. Any valid CSS color value is accepted
15	Color	The font color of this field. Any valid CSS color value is accepted
16	Linked with Page	Determines whether this field is linked with other pages in a relation (e.g., foreign key relationship)
17	Reference Page	If the <i>Linked with Page</i> checkbox is checked, this will determine the page that this field is related with
18	Reference Field	If the <i>Linked with Page</i> checkbox is checked, this will determine the field in the selected page, that this field is related with

Appendix B

Excerpts from Source Code

This appendix include excerpts from the source code of both, the framework, and PMS case study. Table B.1 shows the summary of the included source code.

Table B.1: Source code excerpts summary

File	Description
JavaClassExporter	The Java class that exports EntityMetadata objects into Java Class [Listing B.1].
HtmlExporter	This Java class exports an EntityMetadata object into HTML page [Listing B.2].
JKWebApplicationExporter	This Java class exports an ApplicatioMetadata and its internal metadata data structure into full end-to-end applications [Listing B.3].
Main	This class is generate by the framework, and it is used to launch PMS system [Listing B.4].
Author	Generated model code for the Author metadata [Listing B.5].
MB_Author	Generated controller code for the Author metadata [Listing B.6].
index.xhtml	Generated view code for the Author page [Listing B.7].

```

package com.jk.exporters.common.java;

import com.jk.exporters.core.AbstractExporter;
import com.jk.exporters.core.Exporter;
import com.jk.exporters.core.Exporter.Layer;
import com.jk.exporters.core.Exporter.Technology;
import com.jk.metadata.core.EntityMetadata;
import com.jk.metadata.core.FieldMetadata;
import com.jk.util.JK;
import com.jk.util.java.JKCompileUtil;

@Exporter(name = "Java Class",
    layer = Layer.MODEL,
    technology = Technology.JAVA,
    description = "Normal Java Class (POJO)",
    language = "java",
    unitName="Class")
public class JavaClassExporter extends AbstractExporter {

    @Override
    public String export(EntityMetadata entity) {
        reset();
        // package
        add("package %s;", getPackageName().concat(".models"));
        line();
        // imports
        addImportSection(entity);

        // annotations
        // class line
        add("public class %s{", entity.getCamelCaseWithCapFirst());
        line();
        // instance variables
        for (FieldMetadata field : entity.getAllFields()) {
            add("    private %s %s;", field.getJavaType().getSimpleName(), field.
                getCamelCaseWithSmallFirst());
        }

        // setters and getters
        for (FieldMetadata field : entity.getAllFields()) {
            line();
            String fieldWithSmallFirst = field.getCamelCaseWithSmallFirst();
            String fieldWithCapsFist = field.getCamelCaseWithCapFirst();
            String javaType = field.getJavaType().getSimpleName();
            add("    public void set%s(%s %s){", fieldWithCapsFist, javaType, fieldWithSmallFirst);
            add("        this.%s=%s;", fieldWithSmallFirst, fieldWithSmallFirst);
            add("    }");
            line();
            add("    public %s get%s(){", javaType, fieldWithCapsFist);
            add("        return this.%s;", fieldWithSmallFirst);
            add("    }");
        }
        add("}");
        return getResults();
    }

    protected void addImportSection(EntityMetadata entity) {
        for (FieldMetadata field : entity.getAllFields()) {
            if (!field.getJavaType().getName().startsWith("java.lang") && !field.getJavaType().getName().
                startsWith("[B"]){
                add("import %s;", field.getJavaType().getName());
            }
        }
    }

    @Override
    public String exportAndTest(EntityMetadata entity) {
        String export = export(entity);
        if (JKCompileUtil.compileJavaClass(export)) {
            return export;
        }
        JK.error("Failed to export entity %s with code %s", entity,export);
        return null;
    }
}

```

Listing B.1: Source code of JavaClassExporter.java

```

package com.jk.exporters.common.html;

import java.sql.Types;
import java.util.List;

import com.jk.exporters.core.AbstractExporter;
import com.jk.exporters.core.Exporter;
import com.jk.exporters.core.Exporter.Layer;
import com.jk.exporters.core.Exporter.Technology;
import com.jk.metadata.core.EntityMetadata;
import com.jk.metadata.core.FieldMetadata;
import com.jk.util.JKIOUtil;
import com.jk.util.datatypes.JKType;

@EntityExporter(
    name = "HTML Only",
    layer = Layer.FONTEND,
    technology = Technology.WEB,
    description = "HTML 5",
    language = "html",
    unitName = "Page")
public class HtmlExporter extends AbstractExporter {

    @Override
    public String export(EntityMetadata entity) {
        reset();
        add("<form method='post' action='' id='frm%' />", entity.getName());
        add("<fieldset>");
        add("<legend>%s Form</legend>", entity.getDisplayName());
        {
            add("<input type='hidden' name='%s' />", entity.getIdField().
                getLowerCaseNameWithUnderScores());
            List<FieldMetadata> fieldList = entity.getFields();
            for (FieldMetadata fieldMetadata : fieldList) {
                String htmlName = fieldMetadata.getCamelCaseWithSmallFirst();
                add("<span>");
                add("<label for='%s' >%s</label>", htmlName, fieldMetadata.getDisplayName());
                add("<input type='%s' name='%s' id='%s' />", getFieldTypes(fieldMetadata), htmlName,
                    htmlName);
                add("</span>");
            }
            add("</fieldset>");
        }
        {
            add("<div>");
            add("<input type='submit' value='submit' />");
            add("</div>");
        }
        add("</div>");
        add("</form>");
        if (isIncludeTemplate()) {
            return compileTemplate(getResults());
        } else {
            return getResults();
        }
    }

    protected String getFieldTypes(FieldMetadata fieldMetadata) {
        int type = fieldMetadata.getType().getCode();
        switch (type) {
            case JKType.EMAIL:
                return "email";
            case JKType.URL:
                return "url";
            case JKType.MONTH:
                return "month";
            case JKType.TELEPHONE:
                return "tel";
            case Types.BIT:
            case Types.BOOLEAN:
                return "checkbox";
            case Types.TINYINT:
            case Types.INTEGER:
            case Types.NUMERIC:
            case Types.DOUBLE:
            case Types.FLOAT:
            case Types.DECIMAL:
                return "number";
            case JKType.PASSWORD:
                return "password";
            case Types.BINARY:
            case Types.BLOB:
                return "file";
            case Types.DATE:
                return "date";
            case Types.TIME:
                return "time";
            case JKType.JAVA_CLASS:
            case JKType.PROPERTIES:

```

```
    case Types.VARCHAR:
    case Types.CLOB:
    case Types.NCLOB:
    case Types.LONGVARCHAR:
    case Types.CHAR:
    case Types.VARBINARY:
    case Types.LONGVARBINARY:
    default:
        return "text";
    }
}

private String compileTemplate(String sourceCode) {
    String file = JKIOUtil.readFile("/exporters/templates/bootstrap-html-template.html");
    return file.replace("${pageContents}", sourceCode);
}

public static void main(String[] args) {
    HtmlExporter e=new HtmlExporter();
    System.out.println(e.getName());
}

@Override
public String exportAndTest(EntityMetadata entity) {
    return export(entity);
}
}
```

Listing B.2: Source code of HtmlExporter.java


```

package com.jk.exporters.jkweb;

import java.io.File;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;

import com.jk.exporters.commons.java.JpaEntityExporter;
import com.jk.exporters.commons.java.JpaEntityWithLombokExporter;
import com.jk.exporters.commons.sql.MySqlExporter;
import com.jk.exporters.commons.sql.SqlExporter;
import com.jk.exporters.core.ApplicationExporter;
import com.jk.exporters.core.ExportManager;
import com.jk.exporters.core.MetadataExporter;
import com.jk.exporters.core.TechnologyStack;
import com.jk.metadata.core.ApplicationMetadata;
import com.jk.metadata.core.EntityMetadata;
import com.jk.metadata.core.ModuleMetadata;
import com.jk.metadata.core.ViewGroupMetadata;
import com.jk.metadata.core.ViewMetadata;
import com.jk.metadata.util.NameConvertor;
import com.jk.util.JK;
import com.jk.util.JKIOUtil;
import com.jk.util.JKStringBuffer;
import com.jk.util.logging.JKLogger;
import com.jk.util.logging.JKLoggerFactory;
import com.jk.util.zip.JKZipUtility;

@TechnologyStack(
    name = "JavaStack" ,
    exporters = {
        ViewExporter.class ,
        ControllerExporter.class ,
        JpaEntityWithLombokExporter.class ,
        MySqlExporter.class }
)

public class JKWebApplicationExporter implements ApplicationExporter {
    private static final String PROJECTNAME = "${jk.project.name}";

    static JKLogger logger = JKLoggerFactory.getLogger(JKWebApplicationExporter.class);

    private Path rootFolder;
    private ApplicationMetadata app;
    private File webAppFolder;
    private File javaFolder;
    private File resourceFolder;

    private File templatesFolder;

    private String urlFormat;

    private File incFolder;

    ////////////////////////////////////////
    @Override
    public File export(ApplicationMetadata app) {
        logger.debug("Exporting applications ({})" , app.getName());
        this.app = app;
        try {
            InputStream input = JK.getInputStream("/jk-web-template.zip");
            rootFolder = JKZipUtility.unzip(input);
            webAppFolder = new File(rootFolder.toFile().getAbsolutePath() + "/src/main/webapp");
            javaFolder = new File(rootFolder.toFile().getAbsolutePath() + "/src/main/java");
            resourceFolder = new File(rootFolder.toFile().getAbsolutePath() + "/src/main/resources");
            templatesFolder = new File(rootFolder.toFile().getAbsolutePath() + "/src/main/webapp/WEB-INF/
/templates");
            incFolder = new File(rootFolder.toFile().getAbsolutePath() + "/src/main/webapp/WEB-INF/
templates/inc");

            List<ModuleMetadata> modules = app.getModules();
            logger.debug("Processing application artifacts...");
            processApplicationArtifacts(app);
            for (ModuleMetadata moduleMetadata : modules) {
                logger.debug("Processing module ({})" , moduleMetadata.getName());
                processModuleArtifacts(moduleMetadata);
                List<ViewGroupMetadata> viewGroups = moduleMetadata.getViewGroups();
                for (ViewGroupMetadata group : viewGroups) {
                    logger.debug("Process group metadata ({})" , group.getName());
                    processViewGroupArtifacts(group);
                    List<ViewMetadata> items = group.getViews();
                    for (ViewMetadata view : items) {
                        logger.debug("Process view artifacts ({})" , view.getName());
                        processViewArtifacts(view);
                    }
                }
            }
        }
        File finalZipFile = new File(JKIOUtil.createTempDirectory(), getFinalAppName(app) + ".zip");
        logger.debug("Final file name ({})" , finalZipFile.getAbsolutePath());
    }
}

```

```

        JKZipUtility.zipDirectory(rootFolder.toFile(), finalZipFile);
        finalZipFile.deleteOnExit();
        return finalZipFile;
    } catch (Exception e) {
        JK.throww(e);
        return null;
    }
}

////////////////////////////////////
@Override
public void processAppliactionArtifacts(ApplicationMetadata metadata) {
    // pom file
    {
        File file = new File(rootFolder.toFile(), "pom.xml");
        String pomFile = JKIOUtil.readFile(file);
        pomFile = pomFile.replace("jk-web-template", getFinalAppName(metadata));
        JKIOUtil.writeBytesToFile(pomFile.getBytes(), file.getAbsolutePath());
    }
    {
        File file = new File(webAppFolder, "index.xhtml");
        String configFile = JKIOUtil.readFile(file);
        configFile = configFile.replace("Your Application", getAppTitle(metadata));
        JKIOUtil.writeBytesToFile(configFile.getBytes(), file.getAbsolutePath());
    }
    {
        File file = new File(incFolder, "html-head.xhtml");
        String headerFile = JKIOUtil.readFile(file);
        headerFile = headerFile.replace("Your Project Title", getAppTitle(metadata));
        JKIOUtil.writeBytesToFile(headerFile.getBytes(), file.getAbsolutePath());
    }
    // menu file
    {
        String indexFile = createMenuFileContents(metadata);
        JKIOUtil.writeDataToFile(indexFile.getBytes(), new File(incFolder, "menu.xhtml"));
    }
}

////////////////////////////////////
@Override
public void processModuleArtifacts(ModuleMetadata moduleMetadata) {
}

////////////////////////////////////
@Override
public void processViewGroupArtifacts(ViewGroupMetadata group) {
}

////////////////////////////////////
@Override
public void processViewArtifacts(ViewMetadata view) {
    processViewWebArtifacts(view);
    processViewJavaAtifacts(view);
}

////////////////////////////////////
protected void processViewJavaAtifacts(ViewMetadata view) {
    logger.debug("processViewJavaAtifacts ({}", view.getName());
    if (view.getMainEntityMetadata() == null) {
        return;
    }
    {
        ControllerExporter exporter = new ControllerExporter();
        String packageName = "com.app.controllers";
        exporter.setPackageName(packageName);
        File packageDir = getPackageDir(javaFolder, packageName);
        String export = exporter.export(view.getMainEntityMetadata());
        File javaFile = new File(packageDir, "MB_" + view.getMainEntityMetadata().
getCamelCaseWithCapFirst() + ".java");
        JKIOUtil.writeDataToFile(export, javaFile);
    }
    {
        JpaEntityExporter exporter = new JpaEntityExporter();
        String packageName = "com.app.models";
        exporter.setPackageName(packageName);
        File packageDir = getPackageDir(javaFolder, packageName);
        String export = exporter.export(view.getMainEntityMetadata());
        File javaFile = new File(packageDir, view.getMainEntityMetadata().getCamelCaseWithCapFirst()
+ ".java");
        JKIOUtil.writeDataToFile(export, javaFile);
    }
}

/**
 * @param root
 * @param packageName
 * @return

```

```

*/
public static File getPackageDir(File root, String packageName) {
    if (packageName == null || packageName.trim().length() == 0) {
        return root;
    }
    packageName = NameConvertor.convertToPackageName(packageName);
    String packageDirName = packageName.replace(".", JK.FILE_SEPARATOR);
    File file = new File(root, packageDirName);
    if (!file.exists()) {
        file.mkdirs();
    }
    return file;
}

////////////////////////////////////
protected void processViewWebArtifacts(ViewMetadata view) {
    String path = getPagePath(view);
    logger.trace("Processing page ({} ) at path ({})", view.getName(), path);

    File pageFolder = new File(webAppFolder, path);
    try {
        Path dir = Files.createDirectories(pageFolder.toPath());
        File xhtml = new File(dir.toFile(), "index.xhtml");
        File css = new File(dir.toFile(), "page.css");
        File js = new File(dir.toFile(), "page.js");

        logger.trace("View artifacts are: ({}), xhtml ({}), css ({}), js ({})", path, xhtml.getPath(),
            css.getPath(), js.getPath());

        String pageFullPath = view.getFullQualifiedPath();
        logger.trace("Page file title ({})", pageFullPath);
        String xhtmlContents = createViewContents(view, true);

        logger.debug("Xhtml contents ({})", xhtmlContents);
        JKIOUtil.writeDataToFile(xhtmlContents, xhtml);
        JKIOUtil.writeDataToFile("/* CSS file for this index page*/", css);
        JKIOUtil.writeDataToFile("//Java script file for this index page", js);
    } catch (Exception e) {
        JK.throw(e);
    }
}

////////////////////////////////////
public String createViewContents(ViewMetadata view, boolean includeTemplate) {
    EntityMetadata mainEntityMetadata = view.getMainEntityMetadata();
    if (mainEntityMetadata != null) {
        ViewExporter exporter = new ViewExporter();
        exporter.setIncludeTemplate(includeTemplate);
        return exporter.export(mainEntityMetadata);
    }
    return createFacetletPage("View contents goes here " + view.getName(), view.getName(),
        includeTemplate);
}

////////////////////////////////////
public String createMenuFileContents(ApplicationMetadata app) {
    String menuContents = createTabMenu(app); // createMegaMenu(app);
    return createFacetletPage(menuContents, null, false);
}

////////////////////////////////////
protected String createFacetletPage(String contents, String title, boolean includeTemplate) {
    JKStringBuffer buf = new JKStringBuffer();
    buf.add("<!DOCTYPE html>");
    buf.add("<html xmlns='http://www.w3.org/1999/xhtml' xmlns:h='http://java.sun.com/jsf/html'
        xmlns:p='http://primefaces.org/ui' xmlns:ui='http://xmlns.jcp.org/jsf/facelets'>");
    buf.add("<xmlns:c='http://xmlns.jcp.org/jsp/jstl/core' xmlns:f='http://xmlns.jcp.org/jsf/core'>");
    buf.add("<ui:composition %s>", includeTemplate ? "template='/WEB-INF/templates/default.xhtml'"
        : "");
    if (includeTemplate) {
        if (title != null) {
            buf.add("<ui:define name='page-title'>%s</ui:define>", title);
        }
        buf.add("<ui:define name='contents'>");
    }
    buf.append(contents);
    if (includeTemplate) {
        buf.add("</ui:define>");
    }
    buf.add("</ui:composition>");
    buf.add("</html>");
    return buf.getResults();
}

////////////////////////////////////
private String createTabMenu(ApplicationMetadata app) {
    JKStringBuffer buf = new JKStringBuffer();
    List<ModuleMetadata> modules = app.getModules();

```

```

boolean multiModule = modules.size() > 1;

buf.add("<h:form >");
buf.add("<p:autoUpdate/>");
if (multiModule)
    buf.add("<p:tabView style='margin:auto'>");

for (ModuleMetadata m : modules) {
    if (multiModule)
        buf.add("<p:tab title='%s' icon='fa %s'>", m.getName(), m.getIcon() == null ? "jk-icon"
        : m.getIcon());

    List<ViewGroupMetadata> groups = m.getViewGroups();
    buf.add("<p:menubar>");
    buf.add("<p:menuitem value='Home' url='/' icon='fa fa-home' />");
    for (ViewGroupMetadata group : groups) {
        buf.add("<p:submenu label='%s' icon='fa %s'>", group.getName(), group.getIcon() == null ?
        "jk-icon" : group.getIcon());
        List<ViewMetadata> views = group.getViews();
        for (ViewMetadata view : views) {
            String url = getPageAction(view);
            buf.add("<p:menuitem value='%s' url='%s' />", view.getName(), url);
        }
        buf.add("</p:submenu>");
    }
    buf.add("</p:menubar>");
    if (multiModule)
        buf.add("</p:tab>");
}
if (multiModule)
    buf.add("</p:tabView>");
buf.add("</h:form>");
return buf.getResults();
}

////////////////////////////////////
private String createMegaMenu(ApplicationMetadata app) {
    JKStringBuilder buf = new JKStringBuilder();
    buf.add("<p:megaMenu style='margin-top:20px'>");

    List<ModuleMetadata> modules = app.getModules();
    for (ModuleMetadata m : modules) {
        buf.add("<p:submenu label='%s' icon='fa %s'>", m.getName(), m.getIcon());
        List<ViewGroupMetadata> groups = m.getViewGroups();
        for (ViewGroupMetadata group : groups) {
            buf.add("<p:column>");
            buf.add("<p:submenu label='%s' icon='fa %s'>", group.getName(), m.getIcon());
            List<ViewMetadata> views = group.getViews();
            for (ViewMetadata view : views) {
                String url = getPageAction(view);
                buf.add("<p:menuitem value='%s' url='%s' ajax='false' />", view.getName(), url,
                view.getIcon() == null ? "jk-icon" : view.getIcon());
            }
            buf.add("</p:submenu>");
            buf.add("</p:column>");
        }
        buf.add("</p:submenu>");
    }
    buf.add("</p:megaMenu>");
    return buf.getResults();
}

////////////////////////////////////
protected String getAppTitle(ApplicationMetadata metadata) {
    return NameConvortor.convertToTitle(metadata.getName());
}

////////////////////////////////////
protected String getFinalAppName(ApplicationMetadata metadata) {
    return NameConvortor.convertToLowerWithDashed(metadata.getName());
}

////////////////////////////////////
protected String getPageAction(ViewMetadata view) {
    if (urlFormat == null) {
        return "#{request.contextPath}".concat(getPagePath(view));
    }
    return String.format(urlFormat, view.getFullPath());
}

////////////////////////////////////
protected String getPagePath(ViewMetadata view) {
    StringBuffer b = new StringBuffer();
    String moduleName = view.getParentViewGroup().getParentModule().getName();
    String menuName = view.getParentViewGroup().getName();

    b.append("/pages/");
    if (app.getModulesCount() > 1)
        b.append(NameConvortor.convertToLowerWithDashed(moduleName)).append("/");
}

```

```

        b.append(NameConvertor.convertToLowerWithDashed(menuName)).append("/");
        b.append(NameConvertor.convertToLowerWithDashed(view.getName()));
    }
    return b.toString();
}

////////////////////////////////////
public void setUrlFormat(String urlFormat) {
    this.urlFormat = urlFormat;
}

////////////////////////////////////
public static void main(String[] args) {
    List<Class<MetaDataExporter>> exporters2 = ExportManager.getInstance().getExporters();
    for (Class<MetaDataExporter> class1 : exporters2) {
        JK.print(class1);
    }

    List<Class<ApplicationExporter>> technologyStacks = ExportManager.getInstance().
        getTechnologyStacks();
    for (Class<ApplicationExporter> class1 : technologyStacks) {
        JK.print(class1);
    }
}

////////////////////////////////////
public Path getRootFolder() {
    return rootFolder;
}
}

```

Listing B.3: Source code of JKWebApplicationExporter.java

```

package com.app;

import com.jk.web.embedded.JKWebApplication;

public class Main {
    public static void main(String[] args) {
        JKWebApplication.run(9632);
    }
}

```

Listing B.4: Source code of PMS Main.java

```

package com.app.models;

import javax.persistence.*;
@Entity
@Table(name="author")
public class Author{

    @Id
    @Column(name="id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer id;

    @Column(name="first_name" , nullable=false)
    String firstName;

    @Column(name="last_name" , nullable=false)
    String lastName;

    @Column(name="email" , nullable=false)
    String email;

    @OneToOne
    @JoinColumn(name="university" , nullable=false)
    University university;

    @OneToOne
    @JoinColumn(name="department" , nullable=false)
    Department department;

    @Column(name="phone" )
    String phone;

    @Column(name="mobile" )
    String mobile;

    @Column(name="profile" )
    String profile;

    public void setId(Integer id){
        this.id=id;
    }

    public Integer getId(){
        return this.id;
    }

    public void setFirstName(String firstName){
        this.firstName=firstName;
    }

    public String getFirstName(){
        return this.firstName;
    }

    public void setLastName(String lastName){
        this.lastName=lastName;
    }

    public String getLastName(){
        return this.lastName;
    }

    public void setEmail(String email){
        this.email=email;
    }

    public String getEmail(){
        return this.email;
    }

    public void setUniversity(University university){
        this.university=university;
    }

    public University getUniversity(){
        return this.university;
    }

    public void setDepartment(Department department){
        this.department=department;
    }

    public Department getDepartment(){
        return this.department;
    }

    public void setPhone(String phone){
        this.phone=phone;
    }
}

```

```

}

public String getPhone(){
return this.phone;
}

public void setMobile(String mobile){
this.mobile=mobile;
}

public String getMobile(){
return this.mobile;
}

public void setProfile(String profile){
this.profile=profile;
}

public String getProfile(){
return this.profile;
}
@Override
public String toString(){
StringBuffer buf=new StringBuffer();
buf.append(this.firstName).append(" ");
buf.append(this.lastName).append(" ");
return buf.toString();
}
@Override
public boolean equals(Object obj) {
if (obj == null) {
return false;
}
return this.getId() == ((Author) obj).getId();
}
}

```

Listing B.5: Source code of PMS Author.java

```

package com.app.controllers;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import com.jk.web.controllers.*;
import com.app.models.Author;

@ManagedBean(name = "mbAuthor")
@ViewScoped
public class MB_Author extends JKManagedBeanWithOrmSupport<Author> {

}

```

Listing B.6: Source code of PMS MBAuthor.java

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html" xmlns:p="http://
primefaces.org/ui" xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
xmlns:c="http://xmlns.jcp.org/jsf/core" xmlns:f="http://xmlns.jcp.org/jsf/core">
<ui:composition template="/WEB-INF/templates/default.xhtml">
<ui:define name="page-title">Author Management</ui:define>
<ui:define name="contents">
<h:form id="frmAuthor">
<p:messages />
<p:panelGrid columns="4" id="model" style="margin:auto">
<p:autoUpdate />
<f:facet name="header">#{msg.get('Author')} Form</f:facet>
<p:outputLabel value="#{msg.get('first name')}" for="firstName" />
<p:inputText type="text" id="firstName" value="#{mbAuthor.model.firstName}" required="true"
disabled="#{mbAuthor.readOnlyMode}" readOnly="#{mbAuthor.readOnlyMode}" placeholder="Enter
value" />

<p:outputLabel value="#{msg.get('last name')}" for="lastName" />
<p:inputText type="text" id="lastName" value="#{mbAuthor.model.lastName}" required="true"
disabled="#{mbAuthor.readOnlyMode}" readOnly="#{mbAuthor.readOnlyMode}" placeholder="Enter
value" />

<p:outputLabel value="#{msg.get('email')}" for="email" />
<p:inputText type="email" id="email" value="#{mbAuthor.model.email}" required="true" disabled=
#{mbAuthor.readOnlyMode}" readOnly="#{mbAuthor.readOnlyMode}" placeholder="Enter value" />

<p:outputLabel value="#{msg.get('University')}" for="university" />
<p:selectOneMenu value="#{mbAuthor.model.university}" id="university" required="true" disabled="#{
mbAuthor.readOnlyMode}" converter="omnifaces.SelectItemsConverter">
<f:selectItem itemLabel=" " itemValue="#{null}" />
<f:selectItems value="#{mbUniversity.modelList}" var="ref" itemLabel="#{ref.name}" itemValue="#{
ref}" />
</p:selectOneMenu>
<p:outputLabel value="#{msg.get('Department')}" for="department" />
<p:selectOneMenu value="#{mbAuthor.model.department}" id="department" required="true" disabled="#{
mbAuthor.readOnlyMode}" converter="omnifaces.SelectItemsConverter">
<f:selectItem itemLabel=" " itemValue="#{null}" />
<f:selectItems value="#{mbDepartment.modelList}" var="ref" itemLabel="#{ref.name}" itemValue="#{
ref}" />
</p:selectOneMenu>
<p:outputLabel value="#{msg.get('Phone')}" for="phone" />
<p:inputText type="text" id="phone" value="#{mbAuthor.model.phone}" disabled="#{mbAuthor.
readOnlyMode}" readOnly="#{mbAuthor.readOnlyMode}" placeholder="Enter value" />

<p:outputLabel value="#{msg.get('Mobile')}" for="mobile" />
<p:inputText type="text" id="mobile" value="#{mbAuthor.model.mobile}" disabled="#{mbAuthor.
readOnlyMode}" readOnly="#{mbAuthor.readOnlyMode}" placeholder="Enter value" />

<p:outputLabel value="#{msg.get('Profile')}" for="profile" />
<p:inputText type="text" id="profile" value="#{mbAuthor.model.profile}" disabled="#{mbAuthor.
readOnlyMode}" readOnly="#{mbAuthor.readOnlyMode}" placeholder="Enter value" />

<f:facet name="footer">
<div align="center">
<p:commandButton value="Add" action="#{mbAuthor.add}" rendered="#{mbAuthor.allowAdd}" process=
"model" />
<p:commandButton value="Edit" action="#{mbAuthor.edit}" rendered="#{mbAuthor.allowEdit}"
process="@this" />
<p:commandButton value="Save" action="#{mbAuthor.save}" rendered="#{mbAuthor.allowSave}" process
="model" />
<p:commandButton value="Delete" action="#{mbAuthor.delete}" rendered="#{mbAuthor.allowDelete}"
process="@this" />
<p:commandButton value="Reset" action="#{mbAuthor.reset}" rendered="#{mbAuthor.allowReset}"
process="@this" />
<p:commandButton value="Fill" action="#{mbAuthor.fill}" rendered="#{mbAuthor.allowFill}"
process="@this" />
<p:commandButton value="Cancel Edit" action="#{mbAuthor.cancelEdit()}" rendered="#{mbAuthor.
editMode}" process="@this" />
</div>
</f:facet>
</p:panelGrid>
<br />
<p:dataTable value="#{mbAuthor.modelList}" var="model" rowKey="#{model.id}"
paginator="true" paginatorAlwaysVisible="false" paginatorPosition="bottom" selectionMode="single"
filteredValue="#{mbAuthor.filterList}"
selection="#{mbAuthor.model}" emptyMessage=" " rowIndexVar="row">
<p:ajax event="rowSelect" update="@form:model" />
<p:autoUpdate />
<f:facet name="header">Authors Data Table</f:facet>
<p:column headerText="#">#{row+1}</p:column>
<p:column headerText="#{msg.get('first name')}" sortBy="#{model.firstName}" filterBy="#{model.
firstName}" filterMatchMode="contains">
<h:outputText value="#{model.firstName==null?'-':model.firstName}" />
</p:column>
<p:column headerText="#{msg.get('last name')}" sortBy="#{model.lastName}" filterBy="#{model.
lastName}" filterMatchMode="contains">
<h:outputText value="#{model.lastName==null?'-':model.lastName}" />
</p:column>
<p:column headerText="#{msg.get('email')}" sortBy="#{model.email}" filterBy="#{model.email}"

```



```
        filterMatchMode="contains">
      <h:outputText value="#{model.email==null?'-':model.email}" />
    </p:column>
  </p:dataTable>
</h:form>
</ui:define>
</ui:composition>
</html>
```

Listing B.7: Source code of PMS Author index.xhtml

Appendix C

Clowiz Full Questionnaire

This appendix includes the full questionnaire conducted as an evaluation for the Smart-Cloud framework.

Clowiz (Cloud-Wizard) Questionnaire

This questionnaire is part of "Smart-Cloud: A Framework for Cloud Native Applications Development" dissertation work of Jalal Al Kiswani, a Ph.D. candidate at the University of Nevada, Reno.

We appreciate the time that you will take to complete this form. Please, be sure to go over the tutorial sent with the email invitation before completing this questionnaire.

CONSENT

Your participation in this study indicates that you have read the information provided (or the information was read to you) on this link:

https://drive.google.com/file/d/1nLvNt1w6X_vZj8Wiujfnn2SC74I0fM/view?usp=sharing

The consent form indicates that you are not waiving any of your legal rights as a research participant, your personal information will be confidential and will not be shared with any third party, and you can withdraw from participating at any time.

* Required

1. Email address *

2. Your Name (optional)

3. Can we contact you to get more feedback if required? *

Mark only one oval.

Yes

No

Participant Information

This section includes information about the participant working and professional experience

4. What is your experience level in the software development field? *

Mark only one oval.

Expert level

Professional level

Intermediate level

Entry level

Other: _____

5. What is your highest academic degree? *

Mark only one oval.

Doctoral

Masters

Bachelor

Other: _____

6. What is your current job title?

7. How many years in total do you have with practice, management, and/or research in the software development field? *

8. What is your current day to day work in software development projects? (please check all that apply) *

Check all that apply.

- Research
- Project management
- Business analysis
- Software architecture
- Software development
- Software implementation
- Testing
- Team management
- Technical support
- Teaching or training
- Other: _____

Cloud Applications

This section include questions about modern trends in software applications development, especially those related to cloud applications.

9. For new software applications, choosing a cloud-based approach can be more beneficial than choosing traditional approaches. *

Mark only one oval.

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree
- Other: _____

10. There is a shortage of experienced software engineers who can develop high-quality cloud applications *

Mark only one oval.

- Strongly agree
- Agree
- Neutral
- Disagree
- Strongly disagree
- Other: _____

11. **The cost of software development of cloud based applications is higher than cost of developing traditional software . ***

Mark only one oval.

- Strongly agree
 Agree
 Neutral
 Disagree
 Strongly disagree
 Other: _____

12. **The maintainability cost of cloud based applications is higher than that of traditional software applications ***

Mark only one oval.

- Strongly agree
 Agree
 Neutral
 Disagree
 Strongly disagree
 Other: _____

13. **The operational cost of cloud based applications is higher than that of traditional software applications ***

Mark only one oval.

- Strongly agree
 Agree
 Neutral
 Disagree
 Strongly disagree
 Other: _____

14. **Migrating applications developed using traditional approaches to be cloud-based is an expensive and risky process. ***

Mark only one oval.

- Strongly agree
 Agree
 Neutral
 Disagree
 Strongly disagree
 Other: _____

Clowiz (Cloud-Wizard)

This section includes information about the experience of participants working on Clowiz platform.

15. I find it easy to learn and work on Clowiz *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

16. I understand the process of developing cloud applications using Clowiz. *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

17. Clowiz platform can reduce the development cost of creating cloud-based applications. *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

18. The code generated by Clowiz is high-quality and maintainable *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

19. I would use Clowiz features if it could generate the code in my day-to-day working programming language and technology. *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

20. Clowiz can generate good quality end-to-end cloud-based code, features, and applications. *

Mark only one oval.

	1	2	3	4	5	6	7	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

21. Which feature of Clowiz do you think is more important? *

Mark only one oval.

- CodeGen: Code generation for Software Engineers
- FeatureGen: End-to-end feature code generation
- AppGen: End-to-end application generation
- Other: _____

Suggestions and Recommendations

This section includes suggestions and feedback from users who participated in the experiment.

22. Do you think that any of the current features of Clowiz could be done better? if yes, how?

23. Do you think that there is a functionality that should be available in Clowiz that is not currently available? If yes, please describe.

24. Do you have any other comment?

Bibliography

- [1] Afuah, Allan and Tucci, Christopher L. *Internet business models and strategies*. ACM, <https://dl.acm.org/citation.cfm?id=557033>, 2001.
- [2] Afuah, Allan and Tucci Christopher, <https://dl.acm.org/citation.cfm?id=579515>. *Internet business models and strategies*. McGraw-Hill New York, 2001.
- [3] Amazon. *Amazon Large Datasets*. (Date last accessed April 15, 2018). URL: <https://aws.amazon.com/public-datasets/>.
- [4] Apache. *JUnit*. (Date last accessed Jan 6, 2019). URL: <https://junit.org/junit4/>.
- [5] Apache. *Maven*. (Date last accessed Jan 6, 2019). URL: <https://maven.apache.org/>.
- [6] Armbrust, Michael, Fox, Armando, Griffith, Rean, Joseph, Anthony D, Katz, Randy, Konwinski, Andy, Lee, Gunho, Patterson, David, Rabkin, Ariel, Stoica, Ion, and Zaharia, Matei. “A view of cloud computing”. In: *Communications of the ACM* 53.4 (2010). <https://dl.acm.org/citation.cfm?id=1721672>, pp. 50–58.
- [7] Armbrust, Michael, Fox, Armando, Griffith, Rean, Joseph, Anthony D, Katz, Randy H, Konwinski, Andrew, Lee, Gunho, Patterson, David A, Rabkin, Ariel, Stoica, Ion, and Zaharia, Mate. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. UCB/EECS-2009-28. EECS Department, University of California, Berkeley, <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf>, 2009.
- [8] Avison, David and Fitzgerald, Guy. *Information systems development: methodologies, techniques and tools*. McGraw Hill, <https://www.amazon.co.uk/Information-Systems-Development-Methodologies-Techniques/dp/0077114175>, 2003.
- [9] Azevedo, Leonardo G, Tizzei, Leonardo P, Bayser, Maximilien de, and Cerqueira, Renato. “Installation service: Supporting deployment of scientific software as a service”. In: *Communications (LATINCOM), 2015 7th IEEE Latin-American Conference on*. IEEE, <https://ieeexplore.ieee.org/document/7430148/>. 2015, pp. 1–6.

- [10] Balalaie, Armin, Heydarnoori, Abbas, and Jamshidi, Pooyan. “Microservices architecture enables DevOps: migration to a cloud-native architecture”. In: *IEEE Software* 33.3 (2016), pp. 42–52.
- [11] Banerjee, Prith, Friedrich, Richard, Bash, Cullen, Goldsack, Patrick, Huberman, Bernardo, Manley, John, Patel, Chandrakant, Ranganathan, Parthasarathy, and Veitch, Alistair. “Everything as a service: Powering the new information economy”. In: *Computer* 44.3 (2011), pp. 36–43.
- [12] Bang, Soon K, Chung, Sam, Choh, Young, and Dupuis, Marc. “A grounded theory analysis of modern web applications: knowledge, skills, and abilities for DevOps”. In: *Proceedings of the 2nd annual conference on Research in information technology*. ACM, <https://dl.acm.org/citation.cfm?id=2512229>. 2013, pp. 61–62.
- [13] Bang, Soon K, Chung, Sam, Choh, Young, and Dupuis, Marc. “A grounded theory analysis of modern web applications: knowledge, skills, and abilities for DevOps”. In: *Proceedings of the 2nd annual conference on Research in information technology*. ACM, <https://dl.acm.org/citation.cfm?id=2512229>. 2013, pp. 61–62.
- [14] Barham, Paul, Dragovic, Boris, Fraser, Keir, Hand, Steven, Harris, Tim, Ho, Alex, Neugebauer, Rolf, Pratt, Ian, and Warfield, Andrew. “Xen and the art of virtualization”. In: *ACM SIGOPS operating systems review*. Vol. 37. 5. ACM, <https://cse.buffalo.edu/~stevko/courses/cse704/fall10/papers/2003-xensosp.pdf>. 2003, pp. 164–177.
- [15] Bass, Len, Clements, Paul, and Kazman, Rick. *Software Architecture in Practice (3rd Edition)*. <https://www.amazon.com/Software-Architecture-Practice-3rd-Engineering/dp/0321815734>. Addison-Wesley Professional, 2012.
- [16] Bass, Len, Weber, Ingo, and Zhu, Liming. *DevOps: A Software Architect’s Perspective*. Addison-Wesley Professional, <https://www.oreilly.com/library/view/devops-a-software/9780134049885/>, 2015.
- [17] Beck, Kent, Fowler, Martin, and Beck, Grandma. “Bad smells in code”. In: *Refactoring: Improving the design of existing code* (1999), pp. 75–88.
- [18] Bezemer, Cor-Paul and Zaidman, Andy. “Challenges of reengineering into multi-tenant SaaS applications”. In: *Delft University of Technology, Tech. Rep. TUD-SERG-2010-012*, <https://pdfs.semanticscholar.org/fe9/c3d013c88663670b1a0195d28c79fb5df62b.pdf> (2010).
- [19] Bezemer, Cor-Paul and Zaidman, Andy. “Multi-tenant SaaS applications: maintenance dream or nightmare?” In: *Proceedings of the joint ercim workshop on software evolution (evol) and international workshop on principles of software evolution (iwipse)*. ACM, <https://dl.acm.org/citation.cfm?id=1862393>. 2010, pp. 88–92.

- [20] Buyya, Rajkumar and Bubendorfer, Kris. *Market-oriented grid and utility computing*. Vol. 75. John Wiley & Sons. <https://leseprobe.buch.de/images-adb/26/9f/269f6e68-2f32-46fa-a53b-bf5ad901b0d2.pdf>, 2009.
- [21] Cervantes, Humberto and Kazman, Rick. *Designing Software Architectures: A Practical Approach (SEI Series in Software Engineering)*. Addison-Wesley Professional; 1st edition, <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=454919>, 2016.
- [22] Chen, Peter Pin-Shan. “The entity-relationship model—toward a unified view of data”. In: *Readings in artificial intelligence and databases*. Elsevier, <https://dl.acm.org/citation.cfm?id=320440>, 1988, pp. 98–111.
- [23] Choudhary, Vidyanand. “Software as a service: Implications for investment in software development”. In: *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/4076800>. 2007, 209a–209a.
- [24] Chouhan, Pushpinder Kaur, Yao, Feng, and Sezer, Sakir. “Software as a service: Understanding security issues”. In: *Science and Information Conference (SAI), 2015*. IEEE, <http://ieeexplore.ieee.org/document/7237140/>. 2015, pp. 162–170.
- [25] *Clowiz-website*. from <http://clowiz.com>. (Date last accessed Jan 15, 2019).
- [26] Crockford, Douglas. *JSON (JavaScript Object Notation)*. (Date last accessed April 15, 2018). URL: <https://www.json.org/>.
- [27] Dey, Akon, Chinchwadkar, Gajanan, Fekete, Alan, and Ramachandran, Krishna. “Metadata-as-a-service”. In: *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/7129536>. 2015, pp. 6–9.
- [28] Dixon, Tim and Hargitay, Stephen. *Software Selection for Surveyors*. Springer, <https://link.springer.com/book/10.1007/978-1-349-21696-3>, 1989.
- [29] Dropwizard. *Dropwizard Production-ready, out of the box*. (Date last accessed April 15, 2018). URL: <http://www.dropwizard.io>.
- [30] Ebert, Christof, Kuhrmann, Marco, and Prikladnicki, Rafael. “Global software engineering: Evolution and trends”. In: *Global Software Engineering (ICGSE), 2016 IEEE 11th International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/7577432>. 2016, pp. 144–153.
- [31] Eisele, Markus. *Modern Java EE Design Patterns: Building Scalable Architecture for Sustainable Enterprise Development*. O’Reilly Media, <https://www.oreilly.com/library/view/modern-java-ee/9781492042266/>, 2016.
- [32] Elmasri, Ramez and Navathe, Shamkant. *Fundamentals of database systems*. Addison-Wesley Publishing Company, <https://www.pearson.com/us/higher-education/program/Elmasri-Fundamentals-of-Database-Systems-7th-Edition/PGM189052.html>, 2017.

- [33] Espadas, Javier, Molina, Arturo, Jiménez, Guillermo, Molina, Martín, Ramírez, Raúl, and Concha, David. “A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures”. In: *Future Generation Computer Systems* 29.1 (2013), pp. 273–286.
- [34] Fano, Robert M and Corbató, Fernando J. “Time-sharing on computers”. In: *Scientific American*, <https://www.jstor.org/stable/24931051> 215.3 (1966), pp. 128–143.
- [35] Fehling, Christoph, Leymann, Frank, Retter, Ralph, Schupeck, Walter, and Arbitter, Peter. *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer, <https://www.springer.com/us/book/9783709115671>, 2015.
- [36] Feng, Xinyang, Shen, Jianjing, and Fan, Ying. “REST: An alternative to RPC for Web services architecture”. In: *Future Information Networks, 2009. ICFIN 2009. First International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/5339611>. 2009, pp. 7–10.
- [37] Fortune.com. *Here’s Why Amazon’s Cloud Suffered a Meltdown This Week*. (Date last accessed April 15, 2018). 2017. URL: <http://fortune.com/2017/03/02/amazon-cloud-outage/>.
- [38] Foster, Ian, Zhao, Yong, Raicu, Ioan, and Lu, Shiyong. “Cloud computing and grid computing 360-degree compared”. In: *Grid Computing Environments Workshop, 2008. GCE’08, IEEE*. <https://ieeexplore.ieee.org/document/4738445>. 2008, pp. 1–10.
- [39] Fowler, Martin. *Microservices: a definition of this new architectural term*. (Date last accessed Dec 17, 2018). 2015. URL: <https://martinfowler.com/bliki/MonolithFirst.html>.
- [40] Fowler, Martin and Lewis, James. *Microservices: a definition of this new architectural term*. (Date last accessed Dec 17, 2018). 2014. URL: <https://martinfowler.com/microservices/>.
- [41] Fowler, Martin and Lewis, James. *Microservices: a definition of this new architectural term*. (Date last accessed April 15, 2018). 2014. URL: <https://martinfowler.com/articles/microservices.html>.
- [42] Fromholz, Julia M. “The European Union data privacy directive”. In: *Berk. Tech. LJ*, <https://scholarship.law.berkeley.edu/cgi/viewcontent.cgi?article=1281&context=btlj> 15 (2000), p. 461.
- [43] FSF. *Free Software Foundation*. (Date last accessed April 15, 2018). URL: <https://www.fsf.org/>.
- [44] Gao, Jerry, Pattabhiraman, Pushkala, Bai, Xiaoying, and Tsai, Wei-Tek. “SaaS performance and scalability evaluation in clouds”. In: *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*. IEEE, <https://ieeexplore.ieee.org/document/6139093>. 2011, pp. 61–71.

- [45] Garfinkel, Simson L. *An evaluation of amazon's grid computing services: EC2, S3, and SQS*. Technical Report TR-08-07. Harvard Computer Science Group, <https://dash.harvard.edu/handle/1/24829568>, 2007.
- [46] Garlan, David. "Software architecture: a travelogue". In: *Proceedings of the on Future of Software Engineering*. <https://dl.acm.org/citation.cfm?id=2593886>. ACM. 2014, pp. 29–39.
- [47] Garlan, David. "Software Engineering: Reflections on an Evolving Discipline". In: *International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC) 1.1* (2015). <https://dl.acm.org/citation.cfm?id=2025116&dl=ACM&coll=DL>, pp. 70–77.
- [48] Ge, Yizhe, He, Shan, Xiong, Jingyue, and Brown, Donald E. "Customer Churn Analysis for a Software-as-a-service Company". In: *Systems and Information Engineering Design Symposium (SIEDS), 2017*. IEEE, <https://ieeexplore.ieee.org/document/7937698/>. 2017, pp. 106–111.
- [49] GNU. *GNU Operating System*. (Date last accessed April 15, 2018). URL: <https://www.gnu.org>.
- [50] Google. *Angular*. <http://angular.io>. (Date last accessed Dec 17, 2018).
- [51] Google Inc. *Google App Engine*. (Date last accessed April 15, 2018). URL: <https://cloud.google.com/appengine/>.
- [52] Gorelik, Eugene. "Cloud computing models". MA thesis. Massachusetts Institute of Technology, <https://dspace.mit.edu/handle/1721.1/79811>, 2013.
- [53] Gray, Jim. "Distributed computing economics". In: *Queue* (2003).
- [54] Guo, Chang Jie, Sun, Wei, Huang, Ying, Wang, Zhi Hu, and Gao, Bo. "A framework for native multi-tenancy application development and management". In: *e-commerce Technology and the 4th IEEE International Conference on Enterprise Computing, e-commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*. IEEE, <https://www.computer.org/csdl/proceedings/cec-eee/2007/2913/00/29130551-abs.html>. 2007, pp. 551–558.
- [55] Guradian, The. *Cloud computing is a trap, warns GNU founder Richard Stallman*. (Date last accessed April 15, 2018). 2008. URL: <https://www.theguardian.com/technology/2008/sep/29/cloud.computing.richard.stallman>.
- [56] Hamilton, James. *Internet-scale service efficiency*. (Date last accessed April 15, 2018). 2008. URL: http://www.cs.cornell.edu/projects/ladis2008/materials/JamesRH_Ladis2008.pdf.
- [57] Harrison, Neil B and Avgeriou, Paris. "How do architecture patterns and tactics interact? A model and annotation". In: *Journal of Systems and Software* 83.10 (2010). <https://www.rug.nl/research/portal/files/2617153/2010JSystSoftwHarrison.pdf>, pp. 1735–1758.

- [58] Holt, Adam, Flannery, Simon, Devgan, Sanjay, Malik, Atif, Rozof, Nathan, Wood, CFA1 Adam, Standaert, Patrick, Meunier, Francois, Lu, Jasmine, Chen, Grace, Lu, Bill, Han, Keon, Khare, Vipin, and Miyachi, Masaharu. “Cloud Computing takes off”. In: *Morgan Stanley Blue Paper* (2011). http://www.dabcc.com/resources/cloud_computing.pdf.
- [59] Httermann, Michael. *DevOps for developers*. Apress, <https://www.apress.com/us/book/9781430245698>, 2012.
- [60] Humble, Jez and Farley, David. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. <https://www.amazon.com/Continuous-Delivery-Deployment-Automation-Addison-Wesley/dp/0321601912>. Pearson Education, 2010.
- [61] Humble, Jez and Molesky, Joanne. “Why enterprises must adopt devops to enable continuous delivery”. In: *Cutter IT Journal* 24.8 (2011). <https://www.cutter.com/article/why-enterprises-must-adopt-devops-enable-continuous-delivery-416516>, p. 6.
- [62] Inc., Docker. *Docker*. (Date last accessed April 15, 2018). URL: <https://www.docker.com>.
- [63] JBoss. *Hibernate Community Documentation*. (Date last accessed April 15, 2018). URL: <https://docs.jboss.org/hibernate/orm/3.3/reference/en-US/html/events.html>.
- [64] *jBPM*. <https://www.jbpm.org>. (Date last accessed Dec 17, 2018).
- [65] *Jenkins Community*. (Date last accessed Jan 6, 2019). URL: <https://jenkins.io/>.
- [66] John F. Gantz, Pam Miller. *The Salesforce Economy: Enabling 1.9 Million New Jobs and \$389 Billion in New Revenue Over the Next Five Years*. Tech. rep. IDC, <https://www.salesforce.com/blog/2017/10/salesforce-economy-idc-study-2022>, 2016.
- [67] Kaufman, Lori M. “Data security in the world of cloud computing”. In: *IEEE Security & Privacy* 7.4 (2009).
- [68] Killalea, Tom. “The hidden dividends of microservices”. In: *Communications of the ACM* 59.8 (2016), pp. 42–45.
- [69] Kiswani, Jalal, Muhanna, Muhanna, and Qusef, Abdullah. “Using metadata in optimizing the design and development of enterprise information systems”. In: *Information and Communication Systems (ICICS), 2017 8th International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/7921969/>. 2017, pp. 188–193.

- [70] Kiswani, Jalal, Muhanna, Muhanna, Dascalu, Sergiu, and Harris, Frederick. “Software Infrastructure to Reduce the Cost and Time of Building Enterprise Software Applications: Practices and Case Studies”. In: *Proceedings of ISCA 26th International Conference on Software Engineering and Data Engineering (SEDE 2017)*. ISCA, <https://www.searchdl.org/Resources/Public/Conf/2017/SEDE/1021.pdf>, 2017.
- [71] Käkölä, Timo and Dueñas, Juan Carlos. *Software product lines*. Springer, <https://link.springer.com/book/10.1007/978-3-540-71437-8>, 2006.
- [72] Krebs, Rouven, Momm, Christof, and Kounev, Samuel. “Architectural Concerns in Multi-tenant SaaS Applications.” In: *Closer* 12 (2012). <http://www.scitepress.org/Papers/2012/39576/39576.pdf>, pp. 426–431.
- [73] Kshetri, Nir. “Cloud computing in developing economies”. In: *Computer* 43.10 (2010), pp. 47–55.
- [74] Kumara, Indika, Han, Jun, Colman, Alan, and Kapuruge, Malinda. “Software-Defined Service Networking: Performance Differentiation in Shared Multi-Tenant Cloud Applications”. In: *IEEE Transactions on Services Computing* 10.1 (2017). <https://ieeexplore.ieee.org/document/7522643>, pp. 9–22.
- [75] Laplante, Phillip A, Zhang, Jia, and Voas, Jeffrey. “What’s in a Name? Distinguishing between SaaS and SOA”. In: *IT Professional* 10.3 (2008).
- [76] Linda M. Northrop Paul C. Clements Contributor Reed Little, John McGregor Liam O’Brien Felix Bachmann John K. Bergey Gary Chastek Sholom G. Cohen Patrick Donohoe Lawrence G. Jones Robert W. Krut Jr. *A Framework for Software Product Line Practice, Version 5.0*. (Date last accessed Dec 17, 2018). 2018. URL: <https://www.sei.cmu.edu/productlines>.
- [77] Link, Björn and Back, Andrea. “Classifying systemic differences between software as a service-and on-premise-enterprise resource planning”. In: *Journal of Enterprise Information Management* 28.6 (2015), pp. 808–837.
- [78] Liu, Feng, Guo, Weiping, Zhao, Zhi Qiang, and Chou, Wu. “SaaS integration for software cloud”. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/5557968>. 2010, pp. 402–409.
- [79] Ma, Dan and Kauffman, Robert J. “Competition between software-as-a-service vendors”. In: *IEEE Transactions on Engineering Management* 61.4 (2014). <https://ieeexplore.ieee.org/document/6857369>, pp. 717–729.
- [80] Mather, Tim, Kumaraswamy, Subra, and Latif, Shahed. *Cloud security and privacy: an enterprise perspective on risks and compliance*. <https://www.amazon.com/Cloud-Security-Privacy-Enterprise-Perspective/dp/0596802765>. O’Reilly Media, Inc., 2009.
- [81] Mell, Peter and Grance, Tim. *The NIST definition of cloud computing*. <http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>. 2011.

- [82] Microsoft Live. *Live Email*. (Date last accessed April 15, 2018). 1996. URL: <https://outlook.live.com/>.
- [83] Mietzner, Ralph, Metzger, Andreas, Leymann, Frank, and Pohl, Klaus. “Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications”. In: *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*. IEEE, <https://ieeexplore.ieee.org/document/5068815>. 2009, pp. 18–25.
- [84] Moens, Hendrik, Dhoedt, Bart, and De Turck, Filip. “Management of customizable Software-as-a-Service in cloud and network environments”. In: *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, <https://ieeexplore.ieee.org/document/7502932>. 2016, pp. 955–960.
- [85] Nam, Taewoo and Yeom, Keunhyuk. “Ontology model to support multi-tenancy in software as a service environment”. In: *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/6984188>. 2014, pp. 146–151.
- [86] Netflix. *AWS Service registry for resilient mid-tier load balancing and failover*. (Date last accessed April 15, 2018). URL: <https://github.com/Netflix/eureka>.
- [87] Northrop, Linda. *Trends and new directions in software architecture*. Tech. rep. Software Engineering Institute - Carnegie Mellon University, https://resources.sei.cmu.edu/asset_files/Webinar/2015_018_101_438680.pdf, 2014.
- [88] NSF - National Science Foundation. (Date last accessed April 15, 2018). URL: <https://www.nsf.org>.
- [89] Nurmi, Daniel, Wolski, Rich, Grzegorzczuk, Chris, Obertelli, Graziano, Soman, Sunil, Youseff, Lamia, and Zagorodnov, Dmitrii. “Eucalyptus: A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems UCSB Computer Science Technical Report Number 2008-10”. In: *Computer Science Department* (2008). <http://www.cs.yale.edu/homes/you-minlan/teach/csci599-fall12/papers/eucalyptus-tr08.pdf>.
- [90] Ojala, Arto. “Software-as-a-service revenue models”. In: *IT Professional* 15.3 (2013), pp. 54–59.
- [91] O’Leary, Daniel E. *Enterprise resource planning systems: systems, life cycle, electronic commerce, and risk*. Cambridge university press, <https://www.cambridge.org/core/books/enterprise-resource-planning-systems/FBE044FD5D602059092F5F8A33FF29DD>, 2000.
- [92] Oracle. *Java*. <https://go.java/index.html?intcmp=gojava-banner-java-com>. (Date last accessed Dec 17, 2018).

- [93] Oracle Inc. *Oracle Cloud Platform*. (Date last accessed April 15, 2018). URL: <https://www.oracle.com/cloud/platform.html>.
- [94] Oracle NetSuite. *Cloud ERP*. (Date last accessed April 15, 2018). 1998. URL: <http://www.netsuite.com/portal/home.shtml>.
- [95] Palmer, Michael and Walters, Michael. *Guide to operating systems*. Cengage Learning, <https://www.amazon.com/Guide-Operating-Systems-Michael-Palmer/dp/1111306362>, 2012.
- [96] Parkhill Douglas, <https://www.amazon.com/Challenge-Computer-Utility-Douglas-Parkhill/dp/0201057204>. *The Challenge of the Computer Utility*. Addison-Wesley Educational Publishers Inc. US, 1966.
- [97] Pastor, Oscar, España, Sergio, Panach, José Ignacio, and Aquino, Nathalie. “Model-driven development”. In: *Informatik-Spektrum* 31.5 (2008), pp. 394–407.
- [98] PBXL. *Cloud computing services*. (Date last accessed April 15, 2018). URL: <http://pbxl.co.jp/en/saas-paas-iaas/>.
- [99] Petcu, Dana, Macariu, Georgiana, Panica, Silviu, and Crăciun, Ciprian. “Portable cloud applications?from theory to practice”. In: *Future Generation Computer Systems* 29.6 (2013), pp. 1417–1430.
- [100] Pettey, C. “Gartner Says Worldwide Public Cloud Services Market to Grow 18 Percent in 2017.” In: *Gartner, Press Release* (2017). <https://www.gartner.com/en/newsroom/press-releases/2017-02-22-gartner-says-worldwide-public-cloud-services-market-to-grow-18-percent-in-2017>.
- [101] Pivotal. *Pivotal Cloud Foundry*. (Date last accessed April 15, 2018). URL: <https://cloud.spring.io/spring-cloud-cloudfoundry/>.
- [102] Pivotal. *Spring Boot*. (Date last accessed April 15, 2018). URL: <https://projects.spring.io/spring-boot/>.
- [103] Pivotal. *Spring Cloud-Native*. (Date last accessed April 15, 2018). URL: <https://pivotal.io/cloud-native>.
- [104] Pivotal. *Spring cloud native applications*. <https://pivotal.io/spring-app-framework>. (Date last accessed Dec 17, 2018).
- [105] Redhat. *Hot vs cold deployment*. <https://developer.jboss.org/wiki/HotVsColdDeployment>. (Date last accessed Dec 17, 2018).
- [106] RightScale. *State of the Cloud Report*. Technical Report. RightScale, <https://www.rightscale.com/press-releases/rightscale-2017-state-of-the-cloud-report-uncovers-cloud-adoption-trends>, 2017.
- [107] RightScale. *State of the Cloud Report, Date to navigate your multi cloud strategy*. Technical Report. RightScale, <https://www.rightscale.com/lp/state-of-the-cloud>, 2018.

- [108] Rimal, Bhaskar Prasad, Choi, Eunmi, and Lumb, Ian. “A taxonomy and survey of cloud computing systems”. In: *INC, IMS and IDC, 2009. NCM’09. Fifth International Joint Conference on*. IEEE, <https://ieeexplore.ieee.org/document/5331755>. 2009, pp. 44–51.
- [109] Rumbaugh James, Ivar Jacobson Grady Booch. *Unified modeling language reference manual*. Pearson Higher Education, <https://www.amazon.com/Unified-Modeling-Language-Reference-paperback/dp/032171895X>, 2004.
- [110] *SalesForce CRM*. (Date last accessed April 15, 2018). 1998. URL: <https://www.salesforce.com/crm/>.
- [111] Schlossnagle, Theo. *Scalable internet architectures*. <https://www.oreilly.com/library/view/scalable-internet-architectures/0768666767/>. Sams, 2006.
- [112] Siegele, Ludwig. *Let it rise: A special report on corporate IT*. Special report. The Economist, <https://www.economist.com/special-report/2008/10/23/let-it-rise>, 2008.
- [113] Sommerville. *Software Engineering (10th Edition)*. AddisonWesley, <https://www.pearson.com/us/higher-education/program/Sommerville-Software-Engineering-10th-Edition/PGM35255.html>, 2015.
- [114] *Spring Cloud Foundry*. (Date last accessed April 15, 2018). URL: <https://cloud.spring.io/spring-cloud-cloudfoundry/>.
- [115] Sullivan, Arthur and Sheffrin, Steven M. *Economics: Principles in action*. Pearson Prentice Hall, <https://www.amazon.com/Economics-Principles-Action-Arthur-Sullivan/dp/0131334832>, 2003.
- [116] Sun, Wei, Zhang, Xin, Guo, Chang Jie, Sun, Pei, and Su, Hui. “Software as a service: Configuration and customization perspectives”. In: *Congress on Services Part II, 2008. SERVICES-2. IEEE*. IEEE, <https://ieeexplore.ieee.org/document/4700495>. 2008, pp. 18–25.
- [117] Tsai, Wei-Tek, Huang, Yu, and Shao, Qihong. “Testing the scalability of SaaS applications”. In: *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/6166245/>. 2011, pp. 1–4.
- [118] Tsai, WeiTek, Bai, XiaoYing, and Huang, Yu. “Software-as-a-service (SaaS): perspectives and challenges”. In: *Science China Information Sciences* 57.5 (2014), pp. 1–15.
- [119] Turilli, Matteo, Vaccaro, Antonino, and Taddeo, Mariarosaria. “Internet neutrality: Ethical issues in the internet environment”. In: *Philosophy & Technology* 25.2 (2012), pp. 133–151.
- [120] Turner, Mark, Budgen, David, and Brereton, Pearl. “Turning software into a service”. In: *Computer* 36.10 (2003), pp. 38–44.

- [121] Umble, Elisabeth J, Haft, Ronald R, and Umble, M Michael. “Enterprise resource planning: Implementation procedures and critical success factors”. In: *European journal of operational research* 146.2 (2003), pp. 241–257.
- [122] Utterback, James. “The dynamics of innovation”. In: *Harvard Business School Press, Boston* (1994). <https://www.amazon.com/Mastering-Dynamics-Innovation-James-Utterback/dp/0875847404>.
- [123] Vaquero, Luis M, Rodero-Merino, Luis, and Buyya, Rajkumar. “Dynamically scaling applications in the cloud”. In: *ACM SIGCOMM Computer Communication Review* 41.1 (2011), pp. 45–52.
- [124] Varia, Jinesh. “Architecting for the cloud: Best practices”. In: *Amazon Web Services*, https://d1.awsstatic.com/whitepapers/AWS_Cloud_Best_Practices.pdf ().
- [125] Wu, Shiliang, Wortmann, Hans, and Tan, Chee-wee. “A pricing framework for software-as-a-service”. In: *Innovative Computing Technology (INTECH), 2014 Fourth International Conference on*. IEEE, <https://ieeexplore.ieee.org/document/6927738/>. 2014, pp. 152–157.
- [126] Yahoo Inc. *Yahoo Mail*. (Date last accessed April 15, 2018). 1996. URL: <https://mail.yahoo.com>.
- [127] *Zoho*. <http://zoho.com>. (Date last accessed Dec 17, 2018).