

Pavlo Bazilinskyy

CUSTOMISABLE MULTITENANT WEB FORM WITH JSF AND MYSQL

Bachelor's Thesis
Information Technology

May 2012

DESCRIPTION

	Date of the bachelor's thesis	
	May 4, 2012	
Author(s)	Degree programme and option	
Pavlo Bazilinsky	Information Technology	
Name of the bachelor's thesis		
Customisable multitenant web form with JSF and MySQL		
Abstract		
<p>There is a tendency in Computer Science, nowadays, to move from single-user instances of application to web-based programs. With improvements in Information Technology and Computer Science fields of science it is possible nowadays to conduct business operations from within Internet. Thousands or in some cases millions of sheets of paper and man-hours of work can now be substituted by a single web form connected to a database on a certain website.</p> <p>In recent years a number of new technologies have been introduced to improve usability of Internet applications. It is now possible to create a multitenant piece of software that runs as one instance but serves different users. Nowadays, web forms, that are created for commercial purposes are normally not customisable and lack a possibility to adjust interface in order to suit needs of a particular client. Making multitenant web forms customisable is one of the most highly prioritised tasks for a number of companies that are working in the field of Internet.</p> <p>The aim of the study was to investigate means of creating a fully-functioning and customisable web form that is intended to be run on a server as a single instance. Through methods of user-specific configurations a test case was created that is able to serve a number of clients, giving each one a set of desired features. Before starting this work a following research question was raised: "How to develop the most optimised and the most versatile multitenant web form using JSF and MySQL?". Also, working on this study makes an attempt to answer this question by doing a theoretical research first and then developing a working product that could be used on a market.</p> <p>A part of the study that focuses on development of the test case application is present in the study. Difficulties and issues that are faced while working multitenant cloud-enabled applications are outlined. Listings of programming code are given as examples where they are essential for understanding of the technical aspects of the research. Additionally, different stages of testing are described to outline strengths and weaknesses of the final product.</p>		
Subject headings, (keywords)		
Mutlititenancy, SaaS, Software as a Service, cloud computing, Java EE, JSF, JSP, Java, MySQL, XHTML, HTML, Netbeans, IDE, CSS, Glassfish, web form, web field, tenant, tenant, client, multi		
Pages	Language	URN
99 pages + app. 34 pages	English	NBN:fi:amk-201205259840
Remarks, notes on appendices		
Tutor	Employer of the bachelor's thesis	
Matti Koivisto	Mikkeli University of Applied Sciences	

CONTENTS

1 INTRODUCTION	1
2 CLOUD COMPUTING AND MULTITENANCY	4
2.1 Cloud computing.....	4
2.2 Architecture of cloud computing.....	6
2.3 Software as a Service.....	8
2.4 Software as a Service business model	16
2.5 SWOT analysis of SaaS markets in Ukraine, Finland and the UK	19
2.5.1 Analysis of SaaS market in Ukraine	20
2.5.2 Analysis of SaaS market in Finland.....	22
2.5.3 Analysis of SaaS market in the UK	23
2.5.4 Results of the analysis	24
2.6 Infrastructure for SaaS.....	25
2.6.1 Cluster computer storage	25
2.6.2 Scaling storage for hosting large amounts of data.....	28
2.6.3 Relational databases and cloud computing.....	30
2.7 Multitenancy	31
2.7.1 Multitenancy at enterprise level.....	35
2.8 Development of multitenant applications.....	36
2.9 Agile development of SaaS.....	39
3 CUSTOMISABLE USER INTERFACE	41
3.1 User interface in Software Engineering.....	41
3.2 Problem of customisable user interfaces in modern Computer Science.....	44
3.3 Customisable user interfaces in web-based applications.....	46
4 CUSTOMISABLE WEB FORMS IN MULTITENANT APPLICATIONS.....	51
5 TECHNOLOGIES USED FOR THE TEST CASE.....	53
5.1 JSF 2.0	54
5.2 XHTML	56
5.3 CSS	57
5.4 MySQL	57
5.5 Netbeans IDE.....	58

6 THE TEST CASE APPLICATION.....	60
6.1 Description of views and user actions	63
6.1.1 Registration, login and logout	64
6.1.2 Front page	65
6.1.3 Creating new web forms.....	65
6.1.4 Viewing and filling web forms	68
6.1.5 Editing web forms	73
6.1.6 Managing account information.....	79
6.1.7 Changing tenant-specific configuration.....	80
6.2 Comments and Javadoc	81
6.3 Description of application code from the test case.....	83
6.3.1 Model classes.....	84
6.3.2 Logic classes.....	86
6.3.3 Database Access Object (DAO) classes	87
6.3.4 Database rowmapper classes	88
6.3.5 Views, CSS styling and UI classes	89
6.3.6 Techniques for localisation and session control utilised.....	91
6.4 Debugging in SaaS.....	93
6.5 The database and the application server	93
6.6 The database scheme used.....	94
7 CONCLUSION.....	98
8 BIBLIOGRAPHY	99

APPENDICES

1: SELECTED LISTINGS OF APPLICATION CODE

- 1.1. Method for editing web forms (WebFormView.java)
- 1.2. Method for creating new “child” web forms (WebFormView.java)
- 1.3. Method for parsing a web form (WebFormView.java)
- 1.4. Method for generating a list of user rights for a form (WebFormView.java)
- 1.5. Method for validating email address (EmailValidator.java)
- 1.6. Methods for fetching lists of users (AccountDao.java)
- 1.7. Method for editing web field privileges (WebFormView.java)
- 1.8. Login view and its backing bean (login.xhtml, Login.java)
- 1.9. Method that manages users logging in (Login.java)
- 1.10. Method for locating DataSource object (DataSourceLocator.java)

2: SQL STATEMENTS FOR CREATION OF THE DATABASE

3: FACES-CONIG.XML PROJECT CONFIGURATION FILE

4: DESCRIPTION OF ENTITIES OF THE DATABASE

1 INTRODUCTION

There is a tendency in Computer Science, nowadays, to move from single-user instances of application to web-based programs. With improvements in Information Technology and Computer Science fields of science it is possible nowadays to conduct business operations from within Internet. Thousands or in some cases millions of sheets of paper and man-hours of work can now be substituted by a single web form connected to a database on a certain commercial website.

In recent years a number of new technologies have been introduced to improve usability of Internet applications. It is now possible to create a multitenant piece of software that runs as one instance but serves different users. Nowadays, web forms, that are created for commercial purposes are normally not customisable and lack a possibility to adjust interface in order to suit needs of a particular client. Making multitenant web forms customisable is one of the most highly prioritised tasks for a number of companies that are working in the field of Internet.

The aim of the study is to investigate means of creating a fully-functioning and customisable web form that is intended to be run on a server as a single instance. Through methods of user-specific configurations a test case is created that is able to serve a number of clients, giving each one a set of desired features. Before starting this work a following research question was raised: “How to develop the most optimised and the most versatile multitenant web form using JSF and MySQL?”. Also, working on this study makes an attempt to answer this question by doing a theoretical research first and then developing a working product that could be used on a market by MHG Systems Oy.

A current description of MHG Systems Oy company, available at MHG Systems (2012), says that it is one of the world's leading suppliers of bioenergy ERP systems. The company utilises its partner network to produce customer-oriented IT and map service solutions designed for developing bioenergy and forest energy, and field work business operations. It is an international company with business advisers all over the world. MHG Systems offers its customer companies an MHG ERP and MHG Bioenergy ERP systems and also consultation and training services on bioenergy and forest energy business operations, and field work

management. In addition to the aforementioned, MHG Systems provides its customers with its “know-hows” on modern IT technologies, mobile technology and geographical data. It also offers a technology platform, which it has developed in-house. MHG Systems' services are targeted especially at companies operating in the following sectors: energy, biofuels, electricity and heating, harvesting, sawmilling, pellets, forest services and forestry industry.

MHG Systems works in a field of SaaS (Software as a Service) and an extensive layer of hardware that provides reliable and fault-tolerant connection for its customers and tenants backs up their current system. Solutions that MHG Systems offers for its clients are written in Java programming language. This research was commissioned to investigate and implement an improvement in customisation of their SaaS ERP system.

According to Spolsky (2001), there are three essential parts of success of an application:

- **Features**, referring to what the piece of software does for the user. The demands for the software.
- **Function**, referring to how well the software operates. Perfect program is without any malfunctioning in the logical part: without “bugs” it will function perfectly.
- **Face**, referring to how the application presents itself to the user, the program’s “user interface”, and the way application presents itself to the user.

Features, function and face can also be stated as questions:

- Does the application meet the user’s requirements? (Features)
- Does the application work as intended? (Function)
- Is the application easy to use? (Face)

This work’s goal is not only to answer the question put as a topic, but, additionally, help science advance with understanding of these three components of software in case of cloud computing multitenant applications. This study is organised as described below.

In the chapter Cloud Computing and Multitenancy research on the current situation of cloud computing is performed. Recent advancements in providing services to multiple tenants and cloud-based services are outlined. Also, theoretical foundations, such as used by Software as a Service architectures and implementations of infrastructure behind them, are given. Short

descriptions of methodologies used in development of such solutions as well as SWOT analysis of case studies are also presented.

Chapter Customisable User Interface is arranged in a way that gives a brief description of the concept of the user interface in general and advancements with web-based interfaces in particular. Additionally, recent findings of researchers in the field of creating customisable user interfaces are written.

Chapter Customisable Web Forms in Multitenant Applications is focused on means of research of making web-based input forms configurable and customisable. In other words, it describes investigation of ways of giving tenants of the application sets of tools that can be used to adjust the user interface of the web forms as well as web fields that are assigned to forms. Such modifications are needed to meet as larger amount of possible wishes about adjusting properties of forms that come from the client's side as possible.

In the Chapter Technologies Used for the Test Case a brief description of technology used in this study is outlined. Namely, JSF (JavaServer Faces), MySQL, XHTML and CSS (Cascading Style Sheets) are described. Furthermore, a programming environment that was used to conduct this research - Netbeans IDE - is mentioned.

Chapter The Test Case Application describes a technical part of the study, which deals with development of the test case application. This part of the study is focused on outlining a process of doing that work. Difficulties and issues that are faced while working on such projects are mentioned. Listings of programming code are given as examples where they are essential for understanding of the technical aspects of the research. Additionally, different stages of testing are described to outline strengths and weaknesses of the final product.

Conclusion summarises work concluded and collects achieved results along with comments and suggestions on how to improve the study. Also, in this chapter study gives a description of how findings from this research benefit to the world of Computer Science and improve existing knowledge on a problem of creation of multitenant web forms.

2 CLOUD COMPUTING AND MULTITENANCY

2.1 Cloud computing

"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry."

- John McCarthy, the father of cloud computing (speaking at the MIT Centennial in 1961)

Cloud computing is rapidly increasing its popularity. "It's become the phrase du jour," says Gartner senior analyst Ben Pring, describing needs and opinions of many of his clients and colleges. The term "cloud computing" is used so heavily nowadays that it is now difficult to give a solid definition of it. Yet, understanding a phenomena of "Utility Computing" or "Software-as-a-Service" or even "Application Service Provider" (which according to Aggarwal (2011) are synonyms to the term "cloud computing") is crucial for advancing in knowledge of multitenant architecture and its applications.

Cloud computing was defined by the National Institute of Science and Technology in 2011 as "... a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Mell & Grance, 2011). As Aggarwal (2011) points out there are at least two popular ways of interpreting a meaning of cloud computing present on the scientific scene today. Conservatives argue that cloud computing is merely an addition built on top of utility computing. Knorr & Gruman (2008) have drawn attention to the fact that the idea of utility computing is not new and by such companies as Amazon.com, Sun and IBM it is currently interpreted as offering server farms for their clients as virtual datacenters. Thus indicating that utility computing and cloud computing are not identical entities of Computer Science. On the other hand, some researches and vendors define this technology as broad as including everything that is located outside of a firewall used in the local network of the point of access to the global network.

Arguing about a correct definition of terms is not in a purpose of this research, yet it is valuable to realise how big a potential of this emerging movement is. As rightfully indicated

by Armbrust et al. (2009), cloud computing is making software more attractive than ever as a service and it is changing the way hardware is designed and manufactured. Hardware needs from the era of cloud computing demands powerful fast backbone servers and elements of ICT infrastructure and light clients with excellent network compatibilities. As Hardy (2012) perceptively states companies are buying thousands of servers in bulk, carrying less about brand. One example of hardware from the era of cloud computing is the browser-based Chromebook from Google where all functionality depends on cloud-based applications, it is essentially useless without them.

Reference to Hamdaqa et al. (2011) reveals that cloud duties are to provide compute, storage, communication and management capabilities for SaaS solutions. Tasks can be cloned into multiple virtual machines, and they are usually accessible through application programmable interfaces (API). Internet as a whole can be viewed as a giant cloud and even such its key component as DNS can be considered to be SaaS, following the work of Fox & Patterson (2012a). The growing number of enterprises that use Internet and emerging of broadband technologies for high connections opened the way for SaaS to make significant progress in the Software Engineering industry. (Blokdijk 2008)

Three implementation models exist for the cloud computing solutions, based on research made by Poelker (2011):

1. Private cloud: Created and run internally by an organisation or purchased and stored within the organisation and run by a third party.
2. Hybrid cloud: Outsources some but not all elements either internally or externally.
3. Public cloud: No physical infrastructure locally, all access to data and applications is external.

Naturally, each of advantages and features of cloud computing, that were described in this section, has within it a corresponding disadvantage or concern. First among these is security. Miller (2008, 28) Grossman (2009) claim that because cloud-based services are commonly hosted remotely (including hosted cloud services), their functionality and reliability can be threatened by the latency- and bandwidth-related problems. Furthermore, since hosted cloud services operate with large amounts of customers, various issues related to multiple customers sharing the same piece of hardware and one instance of application code can arise.

Smith (2009) correctly argues that plenty of enterprises are undecided about hosting their valuable data on a computer that is external to their own company and not located on the premises of their office locations. By now, there has been no client-to-client attacks of software or data hosted in the cloud. That may be due to necessary security provisions, or it may be because there has been no value in this kind of penetration in the past. This situation may change in favour of hackers in the future, where cloud computing is likely to become the main platform for software distribution. Additionally, another concern is location. Companies may be concerned about the physical location of the data that is being stored in the cloud. Having said that, it is not problematic to imagine a situation where a poorly designed system can be compromised by actions of tenants that produce unintended results. Finally, according to Miller (2008, 28) services that work from the cloud require high speed Internet connection from their users and might not be suitable for constant use. Cloud-based programs rarely require a lot of bandwidth to download, as do large documents.

In conclusion, let us consider one good example that helps with understanding of cloud computing, which was given in the article of Aggarwal (2011). In it the author compared cloud computing with mobile phones: both of these technologies freed people from using old-fashioned tools in favour of newly developed approaches and appliances. Nowadays, such companies as Oracle and SAP are spending billions in favour of research of cloud computing. And, indeed, thanks to IT companies choosing cloud computing in favour of more conservative solutions such, as not cross-platform desktop applications, web-based multitenant applications became a reality.

2.2 Architecture of cloud computing

As rightfully pointed out by Fox & Patterson (2012a), architecture is a way of organising components of a certain system and other architectures are possible. By choosing a specific architecture, we reject other ways of organisation. The web uses client-server architecture and since cloud computing is a part of Internet, it has clients and servers and the cloud in between. Of course, other architectures are possible and no one can tell for sure that people will still be using the same architecture in a hundred years from now. But architecture of multitenant programs is not a topic of this research, rather practical applications of the architecture that lies beneath it is.

As can be learnt from the online course about SaaS offered by the University of California at Fox & Patterson (2012b), cloud computing uses the most popular architecture in the world of network distributed systems: Client-Server architecture (Peer-to-Peer being the only likely alternative on the market today). Services, which are distributed using cloud computing typically consist of five layers: Server, Infrastructure (Infrastructure as a Service), Platform (Platform as a Service), Application (Software as a Service) and Client (see Figure 1).

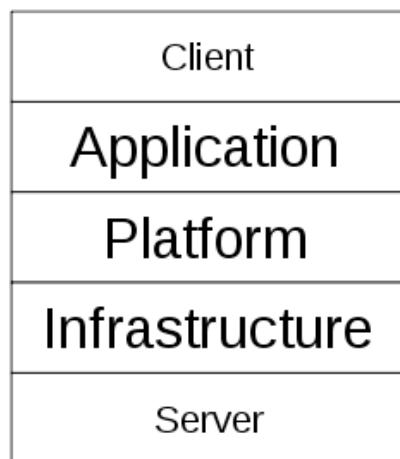


Figure 1. Layers of cloud computing. Wikipedia contributors (2012b)

Let us describe briefly each part of the cloud computing architecture paying special attention to the Application layer.

Clients are used by users to access cloud computing services. The essential requirement for these devices is network access features. Some of the clients rely on cloud based solutions so heavily that virtually all functionality depends on them. Cloud based applications can be accessed using special pieces of software such as email managers and RSS news readers, in other cases virtual machines are used. Majority of clients, however, do not require special programs for access to cloud computing and are capable of running programs using installed web browsers.

Application layer is the most interesting part of cloud computing architecture as for this research. A main purpose of cloud application service or Software as a Service is to publish software in Internet thus eliminating any need for installation of programs as singletenant

applications. According to Blokdijk (2008, 24) sometimes SaaS is also called hosted application, application service provider (ASP), hosted solution etc. This study focuses primarily on this layer of cloud computing.

Cloud **platform** services, also known as Platform as a Service (PaaS) solution stacks and/or computer platforms as a service. It often utilises offered services of the cloud infrastructure and supports cloud application layer. Possibly, the greatest innovation that was brought to Computer Science by introduction of cloud computing is migration of developers to cloud platforms. Such companies as Force.com and Heroku.com abstract the concept of servers entirely, companies of this type focus on core application development from the very first stage of project advancement. The application can be deployed with a single button click and there is no need for application developers to worry about multitenancy, availability, scalability etc.

Cloud **infrastructure** services, also known as Infrastructure as a Service (IaaS), deliver computer infrastructure as a service, extended with means of raw storage and networking. Clients may buy these services entirely outsourced instead of purchasing servers, network equipment, software and datacenters.

The **Server** layer contains software and hardware. Products that are specifically designed for the delivery of cloud services, including multi-core processors, cloud-specific operating systems are present on this layer.

2.3 Software as a Service

Traditional software is represented by binary code that is installed on one machine and runs wholly on the client device. Selling software as a subscription service over the Internet is another approach to the problem of distribution of IT assistance. Turning to research conducted in Blokdijk (2008, 16), one can find an interesting analogy: the authors compare SaaS to electricity, air travel and telephone services, clients pay for services to server providers in a similar manner as paying to electrical companies for using electricity and as buying tickets to use airplanes.

Many applications, which have been known as purely desktop solutions can now be reached in the cloud, for example: Microsoft Office 365, iWorks, TurboTax Online etc. Customers can now choose between paying for a license to use traditional software (and often separate licenses for each individual device using it) or use a product that can be accessed from the cloud. “Why pay more?” - this is a kind of question that can frequently be heard from supporters of SaaS. There are multiple advantages of SaaS over traditional software that make it a likely candidate for a dominant approach to Software Engineering in the near future. Fox & Patterson (2012a). Let us discuss six main advantages of SaaS:

1. No hardware or OS compatibility issues: compatible Internet browser program is enough.
2. Data loss is very unlikely: all data is stored on the remote site.
3. If data used is large and often changed it is simpler to store one copy of it on the remote site.
4. Ease of interactions with other users.
5. Compatibility of software more easily achievable for developers.
6. Simplified maintenance of updates for developers and easier process of updating for users.

Using Software as a Service brings a lot of positive innovations to, first of all, end users that use bought services every day. Indeed, one of the main advantages of cloud-based applications for developers is a high level of compatibility. Hence, there are common tasks that are identical for almost all web applications. When a user sends a request to the SaaS applications such processes occur:

1. “Mapping” of the URI (Uniform Resource Identifier) to a correct program and function.
2. Passing arguments.
3. Invoking the program on the server.
4. Handling storage and initial exchange of data.
5. Handling cookies.
6. Handling raised errors.
7. Output back to the user.

The description of deployment patterns from Microsoft (2012) states that aforementioned duties of SaaS can be categorised into three tiers:

1. **Presentation** (client). This is the top most layer. The main function of it is to translate tasks and processes to understandable to the end user language.
2. **Logic** (application). This layer coordinates the application, processes commands. It is responsible for decision-making and evaluation. It also moves data between other two layer. Separation from the Persistence layer is needed because amount of work done by this layer is greatly smaller compared to the Server tier.
3. **Persistence** (server). In this tier information is processed and stored either in the local storage or in the database. The ready information is passed back to the logic level and then eventually back to the user via presentation layer.

This kind of architecture is often called “Three-tier architecture” (see Figure 2). It is a variation of the N-tier architecture. In this architecture communication between layers with skipping the middle layer is impossible. With SaaS following the Three-tier architecture, developers can use benefits of that system, namely “Shared nothing” principle. What is means is that all three tiers of the architecture that lie behind SaaS application are independent and isolated from each other. It gives a benefit for scalability and development. Understanding of principles of its work is crucial for development of efficient and well-designed multitenant applications.

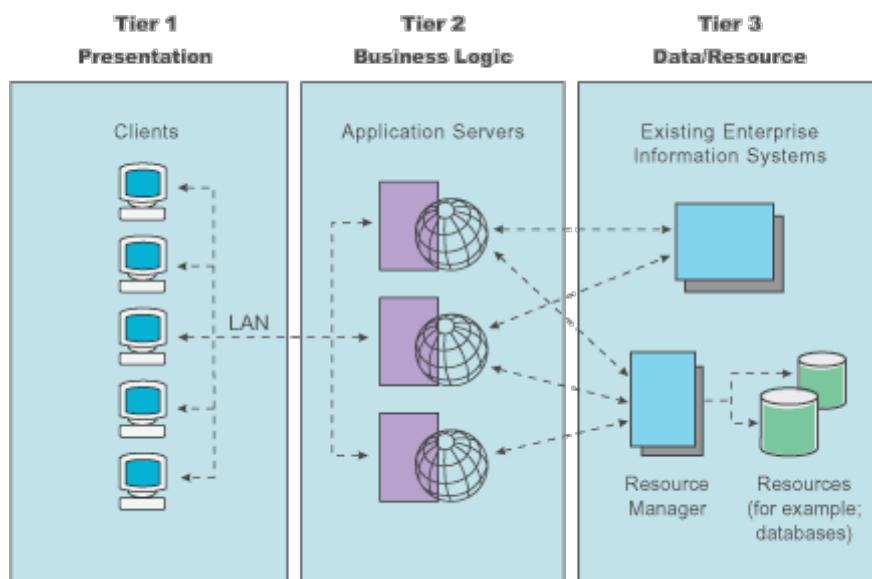


Figure 2. Example of Tree-tier architecture. IBM (2010)

One might confuse the three-tier architecture to model-view-controller (MVC) approach. It is a well-known architectural pattern. However, these two systems are different in their

fundamentals. Unlike three-tier architecture communication between the top layer and the lowest layer is possible without calling the middle layer. MVC is a programming design pattern where different portions of code are responsible for representing the Model, View, and controller in some application (see Figure 3).

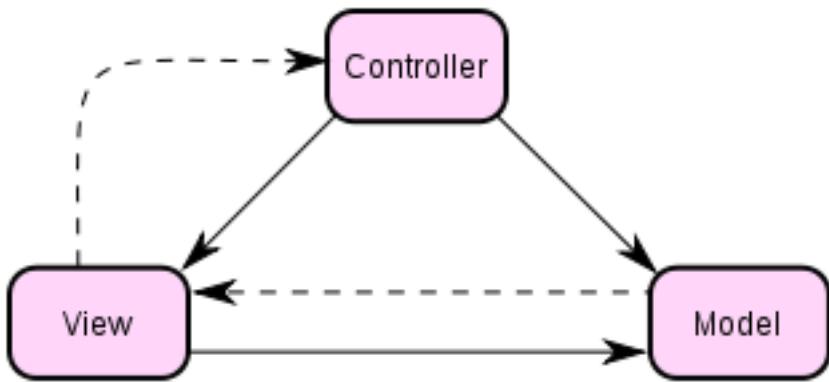


Figure 3. MVC concept. Wikipedia contributors (2012a)

These two concepts are related because, for instance, the Model layer may have an internal implementation that calls a database for storing and retrieving data. The controller may reside on the webserver and remotely call application servers to retrieve data. MVC abstracts away the details of how the architecture of an application is implemented. N-tier architecture just refers to the physical structure of an implementation. These two technologies are sometimes confused because an MVC design is often implemented using an N-tier architecture. Both concepts are used in development of multitenant SaaS applications. In the practical part of this work MVC approach is used for demonstration purposes.

Model-view-controller is the way of organising applications that an end-user is seeing. Presentation layer and UI are separated from the data layer and logical components of the program that are operating on it. With MVC whenever a user interacts with the application, the action is passed to a certain controller that has access to a certain model in the program. Essentially, everything in the program that follows MVC goes via controllers. Models in the MVC can communicate with each other. For example, in a hypothetical bioenergy ERP (Enterprise Resource Planning) system a model of web forms for chipping might be present. However, organising them without information about location of the sight and details about the task is difficult to implement. In this case communication with different models that store data about the location and details about the chipping task would be required.

It is hard to imagine SaaS applications without a term Service Oriented Architecture (SOA). It nearly disappeared from use because it is so unspecific and hard to define with absolute certainty. In this architecture all of the components are designed to be services. In it every component can be used by someone else, not just by its creator, every part of it is independent. For Software Engineering it means ability to create programs, which consist of components that are developed separately. With this architecture creation of a situation-specific pieces of software based on an existing solutions is easier. It is possible to take a ready-made program and change only a few components to make a new solution ready to be used in a new environment by new clients of the company. Mistakes in design are also easier to recover from. Based on the work of Fox & Patterson (2012a), the most distinguished difference between software implemented using SOA and programs that work as standard silo versions is that no service can access or name another service's data. Only requests for data through APIs are possible.

Continuing a topic of SOA architecture one can mention a curious situation that happened a few years ago. This interesting situation happened once in the world of online blogging. Steve Yegge, who has worked both in Amazon.com and Google posted a public blog post, which was intended to be viewed only by the staff of Google (his then current employer) and not outside of the company, copy of it available at Microsoft (2012). In that post he argued that despite all the weaknesses that Amazon.com has compared to Google (which is comprehensively a much larger and more successful company) they have a fully developed and functioning SOA and Google does not. "Start with a platform, and then use it for everything," Yegge demanded, referencing to Google+ as a main example of Google wrongly creating a mere product instead of a Facebook-style platform upon which products can be added and connected. As rightly pointed by the author, creation of SaaS based extendable platforms is the future of programming. Once it was discovered that the post was viewable in entire world, it was shortly deleted. However, it spread quickly over Internet and it was clear from it that even such giants as Google are incorporating the SOA into their products and how important that system for the modern world of software development is.

An absorbing email is worth mentioning in the scope of this research. It perspicuously shows how important and urgent moving to the Service Oriented Architecture is considered by the

key players in the world of IT. It can be backed up by Yegge's description of an email from Amazon.com's CEO Jeff Bezos that he received when Mr. Bezos was his employer:

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed: No direct linking, no direct reads of another team's data store, no shared-memory model, no back doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols -- doesn't matter. Bezos doesn't care.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.

It can be assumed that the last point made by Jeff Bezos is to ensure that all engineers without exception would follow these suggestions. Points 1-5 are more interesting, however, since they define steps required to take to make a SaaS, which emphasises all key aspects of SOA. This email was a beginning of a new strategy acquired by Amazon.com, which brought that company billions in earnings and is a part of one of many success stories of usage of cloud computing in eBusiness.

Turning to Amazon (2008), one can find that in less than two years after launch of the system, Amazon Web Services increased the number of different types of compute servers ("instances") from one to five, and in less than one year they added seven new infrastructure services and two new operational support options.

Cloud computing is becoming more and more popular with every month. Multiple analysis of growth of cloud computing have been made. One of the most notable of them is research conducted by the company called Forrester, which, based on research of Columbus (2011) and Kisker et al. (2011), investigated that SaaS will outgrow all other cloud services, achieving 50% adoption rate in companies (meaning that half of enterprises dealing with IT would have SaaS as parts of their strategies) in 2012, see Figure 4. In previous studies

Forrester has shown that SaaS is a major growth catalyst of ongoing investment in IaaS, PaaS and BPaaS (Business Process as a Service) in enterprises.

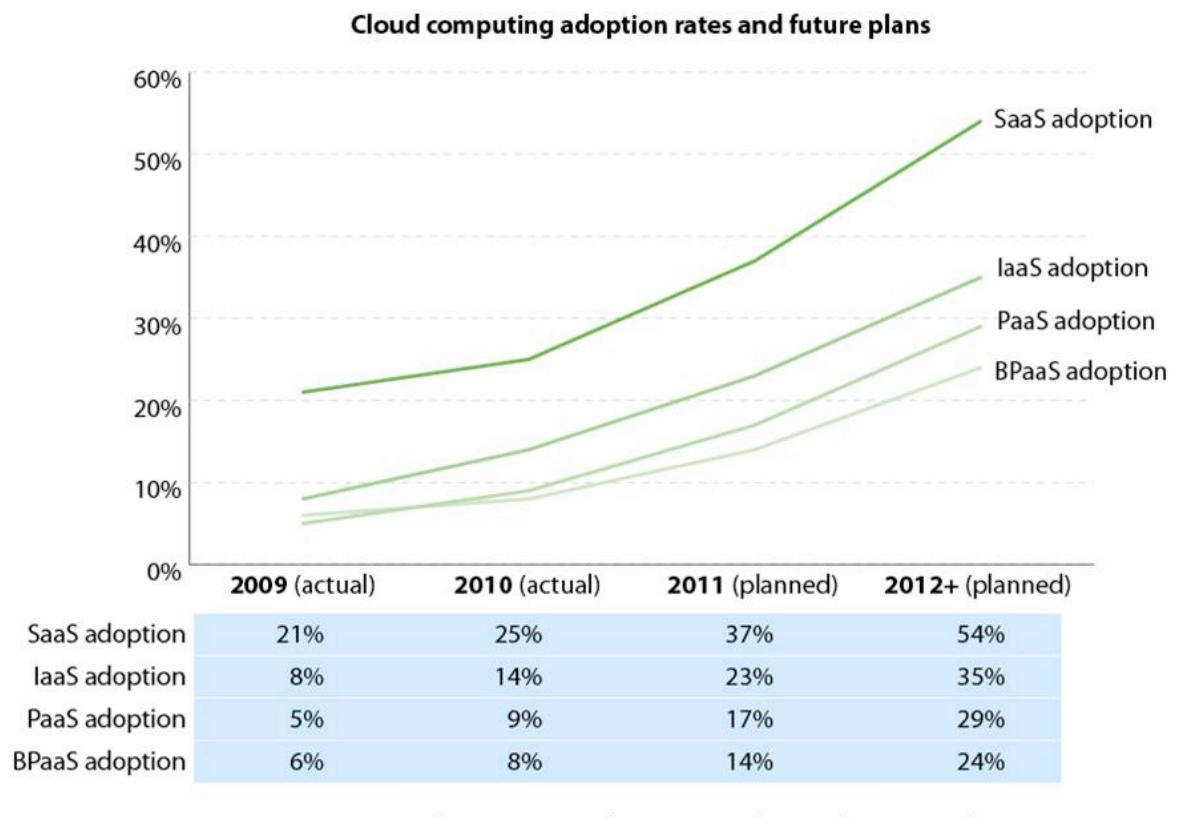


Figure 4. Cloud computing adoption in 2011-2012. Kisker et al. (2011)

Scientists that made their research on SaaS in Fox & Patterson (2012a) predict that virtually all software will be offered as a service by the end of this decade. Thousands of businesses around the world, large enterprises and freelance developers, share the same view on the situation of software development today. Arguably the most famous and known SaaS development company is Salesforce.com. The company's founders Marc Benioff and Parker Harris opened their business in 1999 in California, USA. According to Swartz (2007), this 3000-person company, which mainly focuses on tracking prospects of sales and share information, is "credited with turning the software industry on its head". It has contributed to the revolution that happened in the way software is designed and distributed. Based on the press release from the company at Salesforce.com (2002), after a decade the company has grown so much that they opened their own charity foundation, which in 2011 gave \$100,000 to projects conducted in Tibet. In 2008 Salesforce.com suppressed a \$1 billion mark in sales

and at the moment of writing this paragraph it was reported that the company raised \$632 in revenue in the last quarter of year 2012, as Rao (2012) has indicated. Moreover, analytics predict that earning of Salesforce.com will be growing by up to 30% each year.

According to Hardy (2012), a long-known IT giant Dell, on the other hand, reported a 7% fall in earnings, similar reports came from HP who claimed that a planned transformation in favour of SaaS of the company would take as many as five years. This example clearly shows that research and development in the area of cloud computing and Software as a Service can be beneficial for both entrepreneurs and the welfare of peoples of the world in general.

The evidence seems to indicate that a concept of Software as a Service is not an entirely positive technology, it has negative sides as well. Compared to more traditional ways of offering software, some of its disadvantages are a threat to its further development as a dominating way of offering IT services with help of Internet. Firstly, SaaS applications depend heavily on connection to the network. Secondly, it is a fresh technology and community of developers that are following it is still not as big as of those who create desktop solutions. Further, a range of tools that are offered by development community is in a process of maturing and improvement. Finally, Miller (2008, 29) states that not all tasks can be represented in the cloud, some services rely heavily on special hardware and software.

Blokdijk (2008) states erroneously that Software as a Service is now experiencing transformation into SaaS 2.0, which is arguable. What can you call as SaaS 2.0 if the term is so vastly large and far from a definite definition? No matter what iteration of SaaS is the world of programming in right now, one thing is clear: this architecture will be dominating in the near future.

In conclusion, let us give a quote of a futurist Jack Uldrich: "My prediction is that the term 'cloud' will have disappeared from the phrase 'cloud computing' by 2020, because the majority of computing will simply assumed to be done in the cloud. As a result of this transition, the market capitalization of major networking firms will be slashed to less than one-third of their 2012 levels, and 50% of all of today's IT vendors will be out of business by the decade's end."

2.4 Software as a Service business model

Reference to Fox & Patterson (2012a) Chong & Garraro (2006) reveals that developers who switch to working with off-premises software need to keep in mind aspects of three interrelated areas: business model, application architecture, and operational structure (see Figure 5).

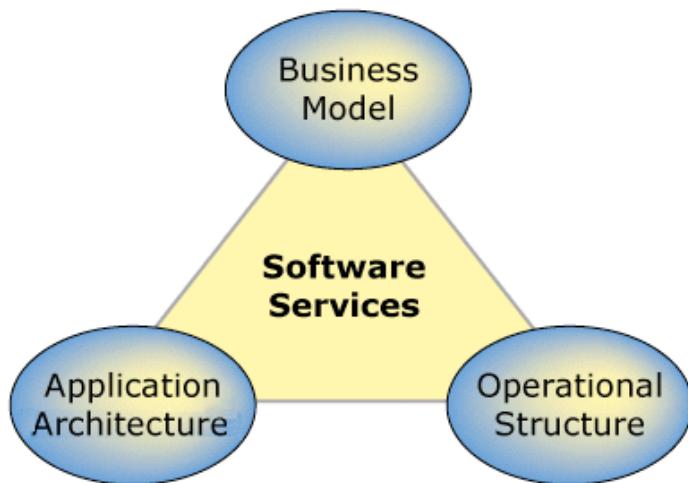


Figure 5. Areas for consideration for SaaS developers. Chong & Garraro (2006)

As application architecture and operational structure of cloud-based services are described extensively throughout this research, let us briefly discuss changes in a business model of an enterprise that need to be undertaken by software engineers before moving from offering on-premise software to offering Software as a Service.

The business model of SaaS has multiple differences compared to traditional software. A question of ownership of the program rises: ownership shifts from the customer to a service provider, responsibility for infrastructure (see section 2.6 Infrastructure for SaaS) and managements is given to a service provider, the cost of providing software services is reduced by means of specialisation and economical scaling, a new target can be chosen - "long tail" of smaller businesses by cutting prices for services.

In an average Software Engineering firm, a budget is divided into following categories, (Chong & Garraro 2006):

- *Software* - expenses on actual projects, all software and data related problems and costs.

- *Hardware* - desktop computers, networking components, servers and mobile devices that provide users with access to the software.
- *Professional services* - staff of the company that is responsible for maintenance of projects and ensuring reliability and customer support.

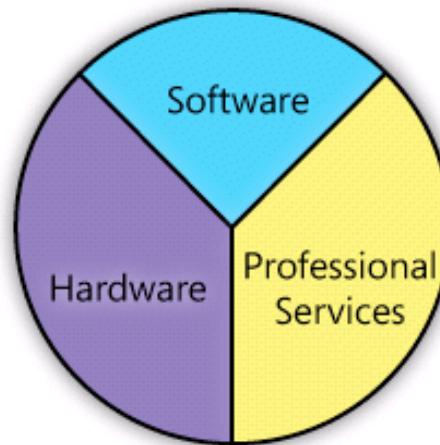


Figure 6. Typical budget for on-premises software environment. (Chong, Garraro 2006)

On Figure 6, one can see a typical division of budget for using projects that are implemented as traditional on-premises software. It is clear from the diagram that expenses on actual functionality of software are gradually lower than on hardware that supports it, and means of professional services. The hardware budget is spent in the direction of desktop and mobile computers for end users, servers that host data and programs, and components of networking. The professional services budget pays for supporting staff to deploy and support software and hardware. Also, consultants and development resources's salaries are paid from the “Professional Services” sector that helps design and build custom systems.

In a company that deals with off-premises cloud-based solutions, a division of money looks significantly different (refer to Figure 7).

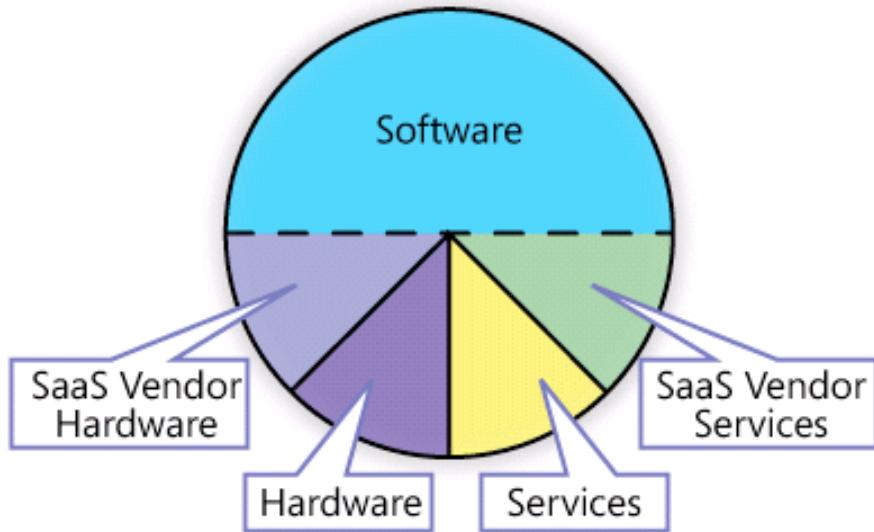


Figure 7. Typical budget for a Software as a Service business. (Chong & Garraro 2006)

Typical budget in an SaaS environment is represented by a much more complex model. In this model SaaS vendor hosts their applications and related data on centralised servers which are supported by dedicated staff. Moreover, applications delivered over Internet place significantly less demand on hardware, which enables the customer to extend the desktop technology lifecycle greatly. However, two new kinds of expenses can be seen in this model: fees for SaaS vendor hardware and fees for SaaS vendor services. On the other hand, it can be concluded that expenses on Software part of the service are largely higher than in the on-premises software model.

By lowering prices for services, Software as a Service solutions can now target a so called “long tail” of business, described in Anderson (2006). By removing a large amount of costs for maintenance, and using scalability in services to combine and centralise customers' hardware and services requirements, SaaS dealers can offer solutions at a much lower cost than traditional vendors. Also, it allows tenants to avoid using complex IT infrastructures. These features give SaaS access to customers for whom using traditional solutions has always been too expensive and unaffordable. Effectively targeting these clients can bring large profits.

Conclusively, the end result of the SaaS business model is that a much higher part of the IT budget is available to spend on actual software, typically in the form of subscription fees to SaaS providers. It can be stated that one of the main breakthroughs received with coming of Software as a Service is its business model that allows customers of such solutions to receive

much better functionality than from comparable on-premises traditional solutions.

Furthermore, even accounting for new costs exposed by SaaS vendors, tenants can still obtain significantly greater software functionality. Additionally, turning to Chong & Garraro (2006) Anderson (2006), one can say that SaaS is uniquely positioned to fill a gap in demand that has not been filled by traditional retailers known as “long tail”.

2.5 SWOT analysis of SaaS markets in Ukraine, Finland and the UK

The author of this research had an opportunity to reside in the UK, Finland and Ukraine during working on this study, see Figure 8. As a result a chance to make analysis of situations with Software as a Service markets in these very different countries was used.



Figure 8. Finland, Ukraine and the UK highlighted on the map of Europe.

These three countries are all from Europe. However, it can be stated that local situations with such high-end technologies as SaaS are rather different. Finland is from Scandinavia, the UK is from the West and Ukraine is an Eastern European country. Finland and the UK are members of the European Union and Ukraine is not. Demographic and political aspects of these countries are also extremely dissimilar from each other.

The UK has the biggest population of 62 million people (data for 2010, based on Google (2012c)), yet it has the smallest area among these countries: 243,610 sq. km (Nationmaster 2012). Ukraine, on the other hand, has a population of 45.8 million people (data for 2010, based on Google (2012a)), and it is the largest country in Europe with area of 603,628 sq. km. And Finland has the smallest population among these three countries, 5.3 million people (data for 2010, based on Google (2012b)), however, it is the 6th largest country in Europe.

Meaning, that these three test cases have very contrasting markets for such businesses as SaaS: Finland is a country with small density of population, when the UK is becoming overpopulated, especially in Southern England. Ukraine is in the middle with a large area and average (decreasing) population.

Finland and the UK are highly-developed industrial countries with very high standards of living and Ukraine is a developing nation still struggling to rebuild itself after the collapse of the USSR. Finland has a rapidly-developing market for new technologies and more and more spheres of life are converted into cloud-based enterprises. The UK, on the other hand, is a country with a bigger diversity of population: a gap between rich and poor is larger. Nevertheless, the government in London is increasing levels of interactions with computing in many areas of life in the country, sometimes turning whole departments of interactions with citizens of government agencies into, fundamentally, websites.

2.5.1 Analysis of SaaS market in Ukraine

Strengths:

Turning to WorldApp (2010), Ukraine has a big technological potential. Its software industry is a fast growing global market. It is a country with a large population and its location between Europe and Asia has been a source of success for international companies.

Weaknesses:

Ukrainian software market develops unsystematically and with a moderate pace. The market for Software as a Service is only emerging now. It can be explained by lack of interest to SaaS from companies of the IT “long tail”. Estimates claim that Ukrainian IT market is 5-6 years behind such developed countries as the USA and the UK. It is unlikely that business climate in Ukraine will be promising for SaaS vendors in the near future.

The country suffers from a vast corruption rate. Practically all levels of its society depend of corruption and almost nothing can be achieved without paying to the black market. In 1990s corruption was the main cause why Ukraine did not succeed in being accepted in the European Union and even now, after more than 20 years of independence, the country is struggling from it. Companies that consider stepping into Ukrainian market should discuss all its weaknesses and threats with other enterprises from the developed world that have experience in working in Ukraine.

Opportunities:

The country has a big potential for software development companies. Its educational system is very similar to the one that the USSR had. Demands on students are very high and to successfully graduate young scholars need to work hard. However, educators give very little freedom and students have to follow very strict rules. As a result, undergraduate and graduate students often take first places in programming competitions, according to ICPC (2011), and they graduate as already matured engineers.

Ukraine is an importer of software. Working for the national market is not profitable for most companies, since level of piracy in the country is exceedingly high and project budgets are often small. A lot of development is outsourced. It creates a perfect productional base for Western engineering enterprises.

Threats:

The government of Ukraine does not provide approving conditions for national Information Technology industry development. If it continues to be a case, a career of and IT specialist will become unattractive for young people of Ukraine and the country will lose all of its IT industry, as well as investments from outside.

Ukrainian piracy market is estimated to be as high as 83 % of the whole market (WorldApp 2010). If nothing will be done, Ukraine may be left without many important spheres of its market, which would collapse under pressure of fighting with piracy.

2.5.2 Analysis of SaaS market in Finland

Strengths:

Finland has a unique position, in the far east of the European Union. According to Finfacts (2011), Google has recently chosen this country as a location for its datacenter, due to cold climate beneficial for cooling of hardware and low energy prices. Moreover, Finland is one of world leading countries in terms of networking and Internet access, which makes it a perfect candidate for usage of Software as a Service. It was reported by BBC News (2010) that since 2009 this country has had broadband connection to Internet as one of fundamental rights of its citizens.

Finland is a country with a very developed and complex market. Turning to the work of Tiihonen (2003, 99), it cannot be said that Finland has no corruption at all, the country constantly occupies high rankings in world free-of-corruption lists. De Heide (2007) reports that in 1990s the Finnish government focussed its policy and instruments on improving R&D intensity, which resulted in the economic growth in the 1990s that outpaced most of its competitors.

Weaknesses:

A low population of Finland means a relatively small market. However, a high level of computer awareness of Finnish population makes their market still quite attractive for SaaS vendors. Additionally, although its location might be considered as a benefit, it can also be called a weakness. Finland is located in a remote part of Northern Europe and reaching Helsinki can be time-consuming.

Opportunities:

SaaS market is a fast-growing and developing market in Finland and in Scandinavia in general. Both local companies like MHG Systems Oy and global ones like Salesforce.com are investing money and resources into Nordic markets.

Because of low density of population in Finland even some traditional sectors of business like mail delivery are seeking ways of using Internet to save costs and improve their efficiency: as indicated by Helsingin Sanomat (2012), Itella (Finnish Post Service company) is now

considering opening physical letters and delivering them via means of Internet to reduce prices and improve their services and reliability.

Threats:

Like in most other developed countries, Finland has been suffering from the bank crisis lately. As a result its economy slowed down and companies like Nokia are closing their offices and moving to other countries. However, the rate of economical growth is still positive and people are expecting further improvements.

2.5.3 Analysis of SaaS market in the UK

Strengths:

The United Kingdom has a largest population among the three countries that were chosen as study cases. Having a population of 62 million people means that there are a lot of clients waiting to be connected to the cloud. It is also the most industrially-developed country with a GDP of \$2.253 trillion (data for 2011, based on International Monetary Fund (2011)). The UK has a leadership position in Cloud Computing adoption with even the government introducing systems that are on early stages of development.

Scaling is what makes cloud computing attractive in the UK. Insurance company Aviva, for example, moved all its enterprise content management and business intelligence tools to Microsoft's Sharepoint service. Also, logistics firm Pall-Ex grows fast and with reducing costs with every day thanks to moving their IT infrastructure to the UK hosting company Outsourcery. (Weber 2010)

The UK has a large software development base and many IT specialists from outside of the UK are attracted to high salaries in this country. It may be a good base for software development firms that seek expanding of their Research and Development departments.

Weaknesses:

The UK has been trying to be a relatively closed country. Even though it is a part of the European Union, it did not sign Schengen Agreement and it is the most isolated “old” member of the EU. This country is highly self-providing with low rates of import compared to its

neighbours. It may be a factor that slows down investments into a sector of SaaS from outside of Great Britain.

Opportunities:

The UK is one of world's leaders in scientific research in Computer Science. New technologies are welcomed in that country and SaaS vendors have big opportunities in Great Britain. Overall, computing is received positively in the country and SaaS vendors may seek profits from opening their services to the UK market.

Following the research of Hingley (2011), the total spending on ITC in the UK in 2011 were about £200 billion, of which around 16% were spent on cloud computing. By 2016 the total market is estimated to grow to £219 billion, of which cloud computing will account for 20% (£43 billion). Consumers and small businesses will spend most on Cloud Services, as researchers predict.

Threats:

Similar to Finland, the UK has been suffering from the bank crisis. As a result its economy slowed down and enterprises are closing their offices. Nevertheless, the rate of economical growth is positive.

2.5.4 Results of the analysis

A situation with SaaS seems to be the most interesting in Ukraine where it is undergoing a start of development and it analytics cannot predict with absolute certainty what will happen in next five years. Ukrainian market is difficult to forecast and, unlike Finland and the UK, the country has a large problem with corruption. Because, of this unpredictability, companies that are interested in investing into cloud computing in developing countries should be on alert and have ready-made strategies for rapid advancement to the market in case of Ukraine.

In cases of Finland and the United Kingdom, situation is quite stable with cloud computing and it promises to be developing further with a constant positive pace. Opening enterprises that are specialised in providing SaaS for customers in these countries have little risks and promising opportunities.

Surprisingly, it was difficult to find trustful information on the current states of SaaS markets in aforementioned countries. More extensive research needs to be conducted, providing how quickly cloud computing is advancing forward.

2.6 Infrastructure for SaaS

In the cloud computing users sign up for a service and in addition to the application itself, they hire the entire ICT infrastructure, which backs up the service. Only Internet connection and a piece of software, such as a modern browser, are required for efficient usage of the program. Furthermore, Software as a Service has specific requirements for the hardware that is used in its backbone and the hardware is essential for success of applications.

There are three piles that cloud computing stands on. SaaS needs to provide good means of communication with a service to its customers providing fine **reliability**: ability to send and receive data with sufficient data exchange ratios. Also, **scalability** is very important for servers that store data of the SaaS, storage used should be easily extendable. Additionally, without constant connection to the program that a user has signed up for, there is “no” service at all since he/she cannot use it under any other conditions other than exchanging information with infrastructure behind SaaS, fine level of connection is achieved by utilising **dependability**. If a user decides to switch to software in the cloud instead of using an application that is installed on the local machine, the SaaS must provide the same value for dependability as its desktop replacement.

2.6.1 Cluster computer storage

The last word in the science of choosing hardware for SaaS belongs to cluster computers. Using cluster computers can save money because ordering mainframe machines in bulk is much cheaper than buying traditional large servers. Furthermore, a group of workers required for maintenance of the farm of clusters usually consists of just a few persons. Clusters are also much more scalable, than large servers. Software hosting companies can now lower their costs of maintaining redundant services such as the upkeep of servers and software maintenance.

As discovered by Armbrust et al. (2009), software engineers with innovative ideas for services that can be launched from Internet no longer need to invest large capitals into hardware resources needed for deployment and human resources requested for maintenance. SaaS architecture is offering attractive possibilities for start-up companies. Starting your organisation in the world of eBusiness used to be difficult and required complex planning and risk analysis. Development of a start-up project often requires investments in multiple spheres of IT besides Software Engineering such as ordering servers for storing data, paying staff that takes care of maintenance of hardware, creating custom interfaces for data communication etc. Today, however, such services as Amazon.com EC2 offer attractive packages for renting hardware for SaaS oriented software development. Amazon.com's EC2 service is virtual and is built on top of the company's Elastic Computer Cloud. Turning to Shore & Warden (2007), one can learn that it is currently the 42nd fastest computer on Earth, which is unprecedented having in mind a price for renting the machine. Anyone can rent this service that has more than 30000 processor units for about €1000 an hour (other super computers with similar performance would require paying millions for completion of the task in the same amount of time). According to Fox & Patterson (2012b) flexible plans offered by companies like Amazon.com offering renting hardware for SaaS allow startup companies to start their businesses with less risk and better chances to succeed. Just five years ago ability to rent hardware for a price of a little more than a euro for an hour was hard to imagine.

With cloud computing developers need not think about overprovisioning for services that are not successful, which are wasting resources, and underprovisioning (see Figure 9) for the ones that meet their optimistic prognoses, thus missing potential customers and revenue. Additionally, enterprises with massive mass-oriented tasks can get updated outcome as quickly as their programs can scale, since using, as an example, 100 servers for one hour requires not more costs than using one server for 100 hours. This stretchiness of resources is extraordinary in the history of IT. Bezemer & Zaidman (2010) back up this idea by indicating that easier application development and better utilisation of infrastructure hardware reduce costs of development and this makes SaaS applications largely attractive for customers in the small and medium enterprise (SME) segment of the market, as those companies often have limited financial resources and do not demand the computational power of dedicated servers.

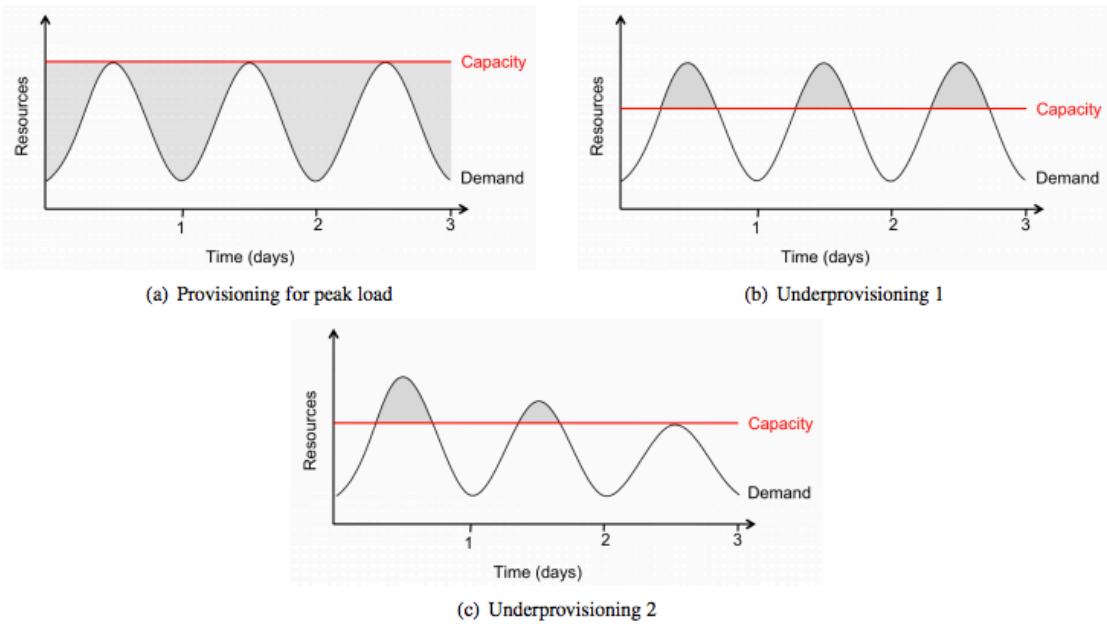


Figure 9. (a) Even if peak load can be correctly anticipated, without elasticity engineers waste resources (shaded area) during nonpeak times. (b) Underprovisioning case 1: potential revenue from users not served (shaded area) is sacrificed. (c) Underprovisioning case 2: certain users leave the site permanently after experiencing inaccessible service. (Armbrust et al. 2009)

Based on the research of Armbrust et al. (2009), the cloud computing's ability to scale quickly is the key observation when a decision about moving to the cloud is made. With elasticity offered by cluster farm services such as the Amazon.com' EC2 matching resources to workload in almost real time is now a reality. According to Rangan et al. (2008) utilisation of most datacenters is only 5 - 20 %, it is explained by the fact that workload during peak time might exceed average rate by factors of 2 to 10. Taking into account such elements of the system as seasonal bursts (e.g., e-commerce peaks during sales seasons and photo sharing sites peak after holidays) and unexpected demand bursts due to external events like news flashes in addition to simple diurnal patterns it can be said that benefits of elasticity can be even higher. With sufficient elasticity of clusters and dynamic allocation of resources utilisation can be raised to be close to 100 %.

Analysing reasons behind moving to cloud computing is important to consider before spending large resource on the change. Let us show a simple equation for estimating cost efficiency of cloud computing compared to a fixed-capacity datacenter:

$$y = UserHours_{cloud} \times (\text{revenue} - Cost_{cloud}) \geq UserHours_{datacenter} \times \left(\text{revenue} - \frac{Cost_{datacenter}}{\text{Utilization}} \right)$$

EQUATION 1. Estimating cost efficiency of cloud computing compared to a fixed-capacity datacenter. (Rangan et al. 2008)

In the left side of the equation an estimate of cost efficiency for cloud computing is outlined, where the net revenue per user-hour (revenue realised per user-hour minus cost of paying Cloud Computing per user-hour) is multiplied by a number of user-hours. In the right side an estimate for efficiency of a traditional fixed-capacity datacenter is given, where the same calculation is performed by factoring the average utilisation, including nonpeak workloads. The greater opportunity for profit is indicated by the side, which is greater in this equation. If Utilization was 1.0 the two sides of the equation would be equal. Authors of (Abramson et al. 2002) argue that in real world this value is typically in a range [0.6 , 0.8], where the equation states that cloud computing is more profitable. The equation explains that the crucial element in successful operation of cloud computing-based services is the ability to control the cost per user hour of operating the service.

The aforementioned model is simplified. The pricing of cloud services may be out of control of the outsourcing company. If the company that provides SaaS services holds a strong position compared to its competitors, it may “overcharge” for its service. And in this case this equation is not necessarily true.

Furthermore, cloud computing and utilisation of clusters can eliminate penalties, which are possible in case of scaling down of the system. The scaling down can happen due to improving software efficiency or business slowdown. For example, with 3-year depreciation, a €2,100 server removed after a year of operation generates a “penalty” of €1,400. With cloud computing it is not an issue. (Armbrust et al. 2009)

2.6.2 Scaling storage for hosting large amounts of data

One of the most discussed areas in research in distributed and cloud computing is a problem of scaling storage for hosting large amount of data. Scientist have invented two ways of doing that: sharding and replication.

According to Fox & Patterson (2012a) Hoff (2009), with sharding data is divided into small parts. One possible way of doing so can be seen from Figure 10 where data, which belongs to users of a certain application, is divided based on the first letter of surnames of users. Any application that utilises storage of this kind should be able to access data from any part of the system, for example, address of Mr. Wales should be as easily accessible as address of Mrs. Anderson. This sort of storage has a big factor of scalability. Additionally, high availability can be achieved: if one server goes down, other servers still work. Also based on Roy (2008), with no master database serialising “Write” queries one can write in parallel, which increases one’s write throughput, Hoff (2009) backs up this idea. The downside of this approach is big latencies of queries that work with populating databases with data.

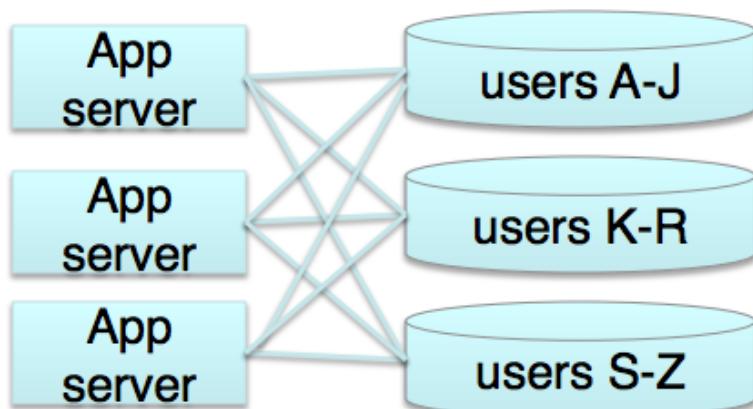


Figure 10. Data distribution using sharding. (Fox & Patterson 2012a)

Unlike sharding, replicated storage have normalised data: when it is needed, data is fetched from different servers and put together (Roy 2008). With replication data is propagated to all copies of the storage. It makes fast operations with writing data possible, but scaling becomes difficult. Whenever data is written to the storage, values of data become temporarily inconsistent. An example of replicated storage can be seen on Figure 11 below, red arrows represent propagation of data.

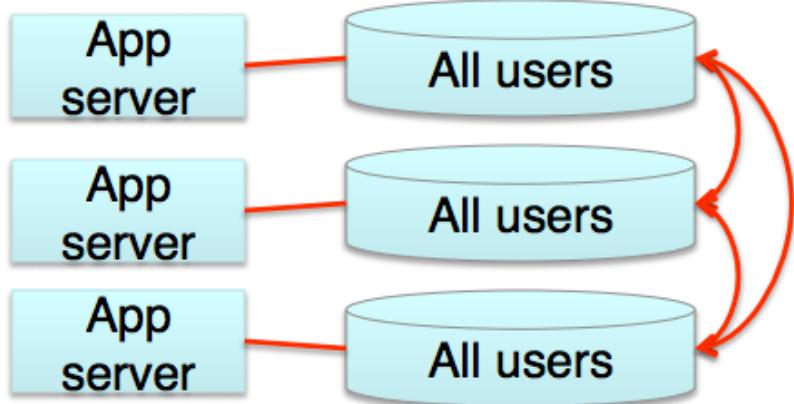


Figure 11. Data distribution using replication. (Fox & Patterson 2012a)

A well-known website Facebook.com has faced the issue with big latencies of queries in sharded databases. Their datacenters handle enormous amounts of data on 24/7 bases and whenever a user posts something on his or her “Wall” other users on the other side of the planet or even a large country would be able to see the post after a few seconds of delay, which is unacceptable in case of social networks. The company has decided to combine sharding with replication. Facebook decided to create one storage unit, which was responsible only for wall posts of users and removed issues connected with high latencies of “Write” operations with database.

The work of Roy (2008) assets that sharding solves problems with replication entirely. However & Roy (2008) does not support Fox & Patterson (2012b)'s argument that only sharding should be used and claims that in reality, most companies are using combinations of sharding and replication. Following material presented at Fox & Patterson (2012b) such combination allows achieving good scalability and fine performance with low latencies of writing data. Scalability of successful SaaS applications that work with large amounts of users and data is a very complicated field of research and it shall not be covered in details in the scope of this research, although understanding of the problem that developers need to face when they deploy their cloud computing projects to the cloud is essential for creation of multitenant SaaS programs.

2.6.3 Relational databases and cloud computing

According to research made by Fox & Patterson (2012b), models used in MVC usually store data in relational databases, which can be viewed using relational database management

systems (RDBMS) such as MySQL Workbench, which was utilised in the scope of this research. Applications and necessity of utilisation of this type of databases has been on agendas of almost all Computer Science related conferences lately. That is why mentioning a few words in the scope of this work can be considered beneficial for the topic.

Relational databases emerged in 1970s. The problem with relational databases is that they do not scale very well. Many solutions have been offered as possible substitutes to classical Relational databases, which are commonly called as “NoSQL”. They scale with a much better rate, which is crucial for development of multitenant SaaS applications.

Each model in the application that is created using MVC approach is associated with a Data Mapper that defines specific rules on how to work with a certain particular model. The negative side of using data mappers is that not all features that simplify complex relationships and queries of RDBMS can be used. Data mappers keep mapping independent of particular data storages and it provides better database system compatibility. Today multiple vendors and database systems are present on the market of DB management. Hence, having a sufficient layer of database compatibility is an important asset for any SaaS project. A popular example of extensive usage of data mappers is Google AppEngine.

The data mappers are used in the database, which was built to support a test case in this research. The alternative approach to data mappers is using Active Records, which might be beneficial in cases when requirement of writing complex data access queries is present. (Fox & Patterson 2012b)

2.7 Multitenancy

Turning to Goldszmidt & Poddar (2008), one can find that web-developed applications that follow a Software as a Service delivery model offer great business value for enterprises of any size. Programmers who develop new solutions following SaaS or transforming old projects into the cloud oriented computing face technical challenges. Among them is a multitenant approach to offering IT services. What is multitenancy and how is it different from the other ways of presenting SaaS applications in the cloud today?

According to Engelsen (2011) there are at least three different ways of implementing program-user relationships:

1. Users log into a single instance of codebase/database.
2. Users log into individually developed pairs of codebase/database where all information is isolated and not accessible by other users.
3. Users log into a single instance of the program, but such methods as configuration files, SQL tables or connection strings are used to locally isolate data and sensitive information.
So called “One-to-Many” approach.

Moreover, another issue is raised when development of applications that are offered as web-based solutions is considered: customisation of the program and ability to serve clients individually, taking their individual needs into account. In the first approach, where all tenants of the service use a single codebase, it is difficult because of issues of separation of data and respecting ownership rights. In the second approach a problem of data individuality does not rise any more but development of codebase/database packages for each client can be cost and time inefficient.

The third approach, however, attracts attention of programmers when they deal with issues of allowing different tenants to use their applications. With this way of implementing codebase for web applications software developers have ability to satisfy multiple sets of requirements asked by different tenants with creation of one single codebase and a logically separated database. It is called the Multitenant (Multi-Tenant) Architecture. On the pages of Goldszmidt & Poddar (2008), one can read that in this architecture a single solution, running on a service provider’s premises, is capable of serving multiple tenants/organisations.

As Jansen et al. (2010) has indicated, a multitenant web application is a program which enables usage of the same instance of a system for different customer organisations or individuals, without necessarily sharing data or functionality with other clients of the program. These tenants have one or more users that use this application to reach tenant-specific goals.

At this point it is important to outline a difference between multitenant and multiuser applications. The work of Bezemer & Zaidman (2010) shows that multiuser programs assume users utilising one application that has configurable elements, on the other hand multitenant

solutions offer much more diverse options for customisation. Multitenant applications are essentially SaaS solutions, which reside in the Application level of a typical system set up using cloud computing. Multitenancy is an organisational approach for SaaS applications as outlined by Bezemer & Zaidman (2010).

Multitenant architecture (example of which can be found on Figure 12) has gone a long way since it was first introduced in 1960s when companies were renting space and processing power of mainframe computing to reduce expenses spent on calculations. In 1990s Application Service Providers (ASPs) were hosting instances of their applications as separate processes or as instances on physically separated machines. Nowadays, such applications as Gmail, Hotmail and Google Calendar are designed as functionally single instances that are capable of servicing often up to hundreds of millions of clients. We are living in the era of multitenant applications and people in today's world are surrounded by them on various levels of their lives.

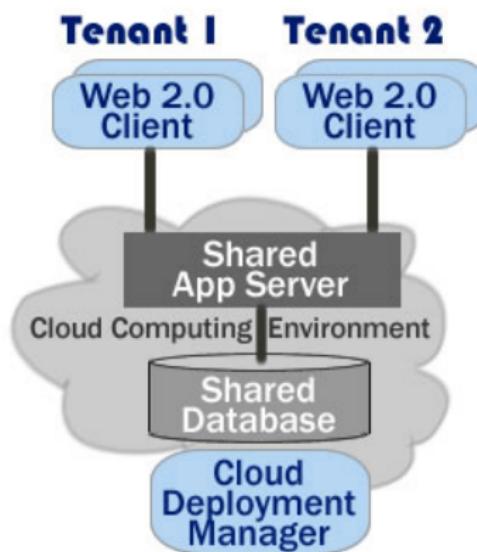


Figure 12. Multitenant architecture. (Keene et al. 2012)

A study by Weissman & Bobrowski (2009) shows that operating just one application that serves needs of multiple clients (in not rare cases millions of tenants of a certain company) provides great scale of economy for the provider. As with all Software as a Service solution, one set of hardware and a relatively small number of experienced workers can support the whole system, software developers need to create and maintain one codebase on a single

platform. The economics offered by multitenancy makes it possible for the application provider to offer the service at a lower cost to customers.

Benefits of multitenancy include, but are not limited to, improved user satisfaction, quality and customer control. Dissimilar to singletenant applications, which are isolated solutions used outside the reach of the application provider, a multitenant application is one large instance that is hosted by the provider itself. This design lets the service provider collect analytical information from the collective user population and make persistent, progressive improvements to the service that benefit the entire user community in an instance.

Furthermore, collaboration and integration can also be improved when multitenancy is used.

Unlike more traditional pieces of software, one of the most important aspects of development and maintenance of multitenant applications, namely web forms which can often be parts of bigger projects, is utilisation of an appropriate system for determining what sets of features and requirements must be shown for certain users and what must be turned off. It is crucial for providing security and giving tenants exactly what they are asking for. Turning to Bezemer & Zaidman (2010), one finds that in contrast to the multiuser applications, multitenancy requires customisation of a single instance of the application for meeting multi-faced requirements of tenants.

Moreover, potential issues can arise in multitenant applications, based on Goldszmidt & Poddar (2008):

1. **Isolation:** since tenant share the same instance of software and hardware, availability of one tenant can possibly be affected by actions of other users.
2. **Security:** if the shared program does not have adequate protection, users of one tenant might be capable of accessing data from the other tenant. However, turning to Weissman & Bobrowski (2009), one can find that thanks to all users running all applications in one virtual space, letting any user of any application assorted access to specific sets of data is easily reachable, if required.
3. **Customisability:** appropriate level of customisation of multitenant software is difficult to achieve, since all customers use the same instance of the solution. This is a topic of interest for many researchers.

4. **Application upgrades can cause problems for tenants:** simultaneously upgrading shared software may not be desirable for all tenants.
5. **Recovery:** tenants use the same database, which makes it difficult to back up and restore data for each individual client.

Software developers must be aware of these problems of multitenancy and avoid them in their products.

2.7.1 Multitenancy at enterprise level

A great number of enterprises are looking into ways of transforming their singlenetant applications into multitenant ones. Yet, two obstacles of multitenant architecture are slowing this process down, namely:

- Enterprises differ in terms of initial budgets for reengineering their singlenetant applications into multitenant programs (Tsai et al. 2007).
- Maintenance personnel is anxious that multitenancy may bring additional problems into maintenance based on the fact that these solutions are highly customisable and they require effective ways of customisation, in the process eliminating the advantage that multitenancy offers through the fact that updates are developed and applied only once. (Bezem & Zaidman 2010)

Mutlitenancy is used in various projects. Especially big value for using this approach can be found in a field of development of enterprise resource planning (or ERP) systems which, according to Blokdijk (2008), are sets of services attempting unification of all data information and processes of a particular company into one cohesive system. Multitenant ERP systems that serve multiple clients need to be capable of providing well designed tools for customisation on all levels of the system. Most companies would prefer the ERP system to be as customised as possible and fit to their particular needs.

Turning to pages of Leon (2008), one can find that initially ERP systems were targeted only at the manufacturing enterprises and included common for such businesses functions such as sales management, accounting, cost management, etc. However, in recent years ERP systems have been adapted by a great number of companies from other areas and field of business. Enterprise resource planning systems have reached a truly global level. It can be said that in

today's world the enterprise resource planning system is a very important asset for practically any organisation.

2.8 Development of multitenant applications

Before development of any service it is important to assess all sides of upcoming development process. One of the major phases of development of software is determination of the service value. Developers should ask questions like "Why does the customer need this service?" and "Why should the customer give our company this project?" To answer these questions two factors need to be examined: Service Warranty and Service Utility, which in combination produce Service Value (see Figure 13). (Hatch 2008, 17 - 18)

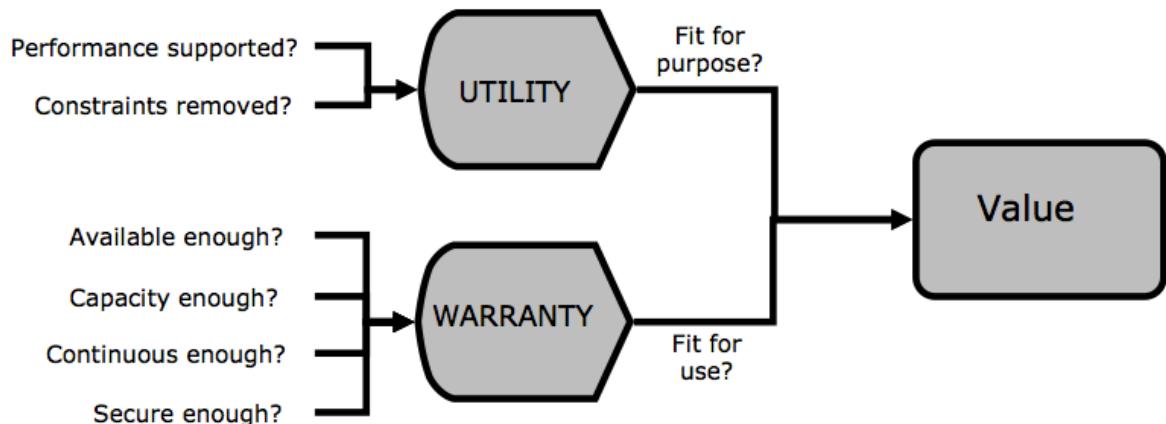


Figure 13. Creating Service Value. (Hatch 2008) p.17

Service Utility defines the functionality side of the service from the user's point of view (i.e. what the service does).

Service Warranty represents level of reassurance and guarantee for meeting discussed requirements for the project.

Together they represent the value of the service, which can be determined with help of Equation 2:

$$\text{Service Warranty} + \text{Service Utility} = \text{Service Value}$$

EQUATION 2. Service value. (Hatch 2008, 17)

If after assessment of risks, which needs to be taken during the work on the project, results prove that development will be successful and profitable, the SaaS application can be developed and released. Let us take a look at the brief history of development of RIA (Rich Internet Applications) and aspects of working on projects of this type.

In the elder days Internet was a collection of websites that consisted of HTML pages, which sometimes had CSS styling put on top of them. Eventually, web developers faced a problem of the necessity of dynamic content for the web. At first, by such means of programming as PHP and Perl languages of programming, websites started to have pieces of code put inside of HTML tags that generated something unique for specific situations (Fox & Patterson 2012a). Later, scientists and engineers came up with a concept of using templates with snippets of code that were capable of generating unique output for individual clients. This was a time of first e-commerce and advanced Web 1.0 web-based projects.

According to work done in Goldszmidt & Poddar (2008), there are multiple technical challenges connected to development of multitenant SaaS solutions:

1. **Access control:** a problem of separating tenants from each other. Application resources, such as virtual portals, database tables, workflows are to be shared by users of different tenants. The technical difficulty is in providing means of controlling areas of access for users of different tenants of the service. It faces isolation and security of multitenant software.
2. **Customisability:** customisation of different elements of multitenant solutions can be tedious because each time a user from a certain tenant logs into the application a configuration set for exactly that client, and not for any other, must be loaded and used.
3. **Tenant provisioning:** an issue of configuring the system to allow addition of new clients with as few manual steps as possible.
4. **Usage-based metering:** monitoring usage of the service by tenant can be challenging. It is required, though, for an intelligent system of charging, where tenants pay money only for that time when they actually use the service.

Because of these challenges developers of multitenant applications need to pay special attention to selecting a multitenant server framework. In an article by Keene et al. (2012) one

can find main points that need to be considered before starting development of multitenant software, as seen by IBM:

1. An open, standards-based server framework needs to be chosen. Multiple Platform as a Service options use proprietary languages and hosting provider. This should be avoided and open, possibly Java-based framework, should be chosen instead.
2. Per tenant extensions need to be enabled within shared schema. Tenants sometimes require adding tenant-specific data extensions without affecting the overall state of the system. Customisation is needed.
3. Per tenant data backup and recovery is needed. Using SaaS means that customers need to entrust sensitive data to service providers. Ensuring per-tenant backup and recovery as a part of the multitenant server framework improves trustworthiness of the service.
4. Integration of role-based user security with client-side user interface. A disconnect between client-side and server-side security frameworks is often a place where malicious code penetrates the system and deals damage. The server-side role based access control framework should provide a sufficient level of management of client access to UI widgets, services and data.
5. Integrated deployment to cloud hosting environments. The multitenant server framework should allow with ease connections to leading cloud providers such as Amazon EC2, which have their own APIs and requirements.
6. Transparent failure management. In case of failures of application or database servers transitions to backup servers should be seamless and painless to end-users.

One may argue that this list made by IBM is biased, since it promotes products developed by that company. However, the main principles exposed in it will hold true if used with other proprietary products from other companies, such as Microsoft .NET Framework 4. Keeping these six aspects of deploying multitenant applications in mind is important and can help with creation of a successful and promising SaaS application that uses multitenancy. Agile development can increase quality of software projects and help create multitenant SaaS solutions.

2.9 Agile development of SaaS

There are two main approaches to development of software projects today: “agile” and “waterfall” development processes. With development that is conducted based on the waterfall approach a lot of planning is done in the initial stages of the project, which is typically divided into the following sections, see Figure 11:

1. Requirement analysis and specification.
2. Architectural design.
3. Implementation and integration.
4. Verification.
5. Operation and maintenance.

This model interlaces a lot with a similar approach that is used for development of hardware. Why? Because with the waterfall model catching malfunctioning and removing bad flaws of design can be done during all phases of development. Nevertheless, the final outcome cannot be tested until the very last stage of development. The key principal of this method is outlining the “big” design of the project upfront, before any actual development has started. Having malfunctioning in the final version of the hardware product can cause millions and it can put an end to the project, if competitor teams have succeeded in a similar design. That is why different teams of specialised professionals using extensive documentation as medium of cooperation often do these five stages. Another issue with using this technique for development of software and hardware is little interactions with a client who ordered creation of the program. If a customer is unsatisfied, the project often needs to be started over from the starting stage.

The waterfall model works really well for enterprises where specifications do not change fast, for example: Operating System for a drone sent to Mars or aircraft control. In development of most modern day software, however, a radically different approach is used because programs typically need to meet new requirements with dramatic speed. Creating SaaS products utilises iterative lifecycle heavily. Projects are developed in small iterations (see Figure 14) with earning money and releasing versions of the software while the project is advancing further.

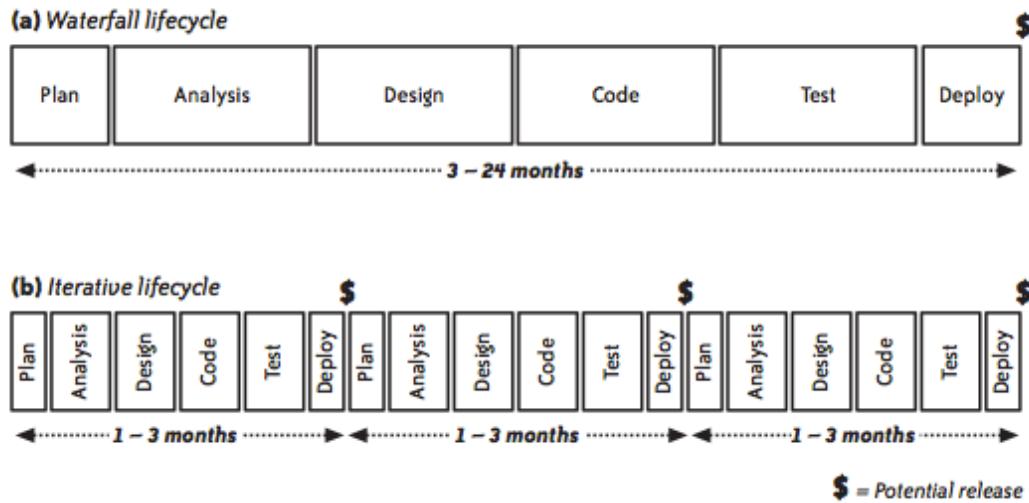


Figure 14. Waterfall and agile development lifecycles. (Shore & Warden 2007, 16)

Interestingly, agile software development started as a movement of programmers in 2001. A website called “Manifesto for Agile Software Development” was created where they outlined main principles of what in their opinion was a foundation for a better way of creating software. And, more than a decade later, the following principles are considered as core values in organising software development teams (Beck et al. 2001):

- *Individuals and interactions* over processes and tools.
- *Working software* over comprehensive documentation.
- *Customer collaboration* over contract negotiation.
- *Responding to change* over following a plan.

Values on the left in the list above (highlighted in italics) are core principles of agile software development, while statements on the right are valued as well, but with smaller respect. Software development that follows these four lines of manifesto created in 2001 is known to be the best technique available today for development of customer-oriented software and, namely, SaaS and it will be put into test in the scope of this research. (Shore & Warden 2007, 9 - 11)

Agile software methodologies are a radical departure from the traditional, document-heavy waterfall activities that are still in extensive use today. These methodologies share a set of common techniques. They all try to find a useful compromise between informal development processes and formalised, traditional ones. (Larman 2003)

3 CUSTOMISABLE USER INTERFACE

3.1 User interface in Software Engineering

Information Technology has gone a long way from command line-controlled computers to modern cross-platform and/or cross-browser applications with polished and user-friendly interfaces. Myers & Rosson (1992b) has drawn attention to the fact that for the last 40 years assembling applications from components has been a focus of extensive research and it led to improvements of knowledge in such areas of Computer Science as Software Engineering using component-based approach, middleware development and service composition.

However, little amount of studying has been done to the layer of presentation, which includes user interface (UI) design and applications. User interface is the top-most level of application and it is the only layer visible to most end-users without knowledge in IT, hence, developers should be paying substantial attention to this field of human-machine interaction.



Figure 15. Workstation in 1960s. One of the first Graphical user interfaces (GUI). (Foremski 2005)

User interface is a way humans interact with their devices and if usage of programs on a computer becomes a problem, usually the UI is to be blamed. User interfaces can be rightfully

called “a central ingredient of computer user satisfaction”. Research on the UI started in 1945 when the batch interface was invented. The history of progression of UI can be broken into three parts: batch (1945-1968), command-line (1969-1983) and graphical (1984 till today). An example of an average computer from 1960s can be seen on Figure 15. All started, according to multiple sources, with the invention of the digital computer. The opening years on the latter two eras are the days when new interface technologies were invented and began to transform users' expectations about interfaces in a serious way, those technologies were interactive timesharing and the graphical user interface. (Raymond & Landley 2004)

User interface has gone a long way since 1950s and 1960s where a computer with a circular screen and a one-button mouse presented on the picture above was the top of the range. Today, UI, almost in its all entirety, used in a form of GUI. Nowadays, according to Bowman et al. (2004), research is conducted even on development of 3D user interfaces. However, one aspect of user interfaces has not changed in last 60 years and is unlikely to change in the future: UI plays a central role in usability of the system. And, taking care about it is an important task for all programmers, engineers, designers and anyone working with the UI.

One integral part of good usability (according to the dictionary: “The ease of use and learnability of a human-made object”) is a high degree of user-friendliness. With today’s level of technological advancements, users interact with a massive amount of variations of UI on daily basics. And, the less time it takes for a user to get accustomed with the interface of a certain application, the bigger chances are that the user in question will not choose a competitor’s solution.

Turning to Spolsky (2001, 7 - 8), one can find a good example about a diversity of the UI in modern world: a story about a power user, whose area of responsibility covers Microsoft Windows OS trying to fix a problem in a Mac OS - powered laptop. Because Mac OS is so different compared to Windows OS, he was faced with a serious of challenges that were required to be overcome to fix a small issue and a headache for the rest of the day. The moral of this story is that UI needs to be created in such a way that using it is entirely logical and effortless. Satisfied user is hard to achieve.

The following points have been reported by many sources as possible reasons for user frustrations with user interfaces:

1. When a program does not work properly or exits with errors.
2. When a system does not do what the user wants it to do.
3. When a user's requirements are not met.
4. When a system does not provide sufficient information to enable the user to know what to do.
5. When indistinct, annoying or condemning error messages appear.
6. When appearance of an interface is over-bright, glamorous or patronising.
7. When a system requires users to carry out too many steps to perform a task.
8. Troublesome interference.

Satisfaction of users can be met by creating easy to use software. However, as interfaces become easier to use, they become harder to create (Myers 1994). Following the work of Spolsky (2001), designing a good user interface is a challenging process. To succeed, a designer/developer needs to constantly evaluate his or her work. After designing the UI, one has to evaluate done job, make necessary adjustments, if any. This process needs to be repeated until requirements are met or the team ran out of time/money. While design is important, the most crucial aspect of creating a good user interface is in evaluation techniques. Evidently, a designer/developer should be able to use his or her own user interface. If that person cannot use it, how can anyone else?

Daniel et al. (2006) makes clear that one of possible ways of improving a level of quality of modern user interfaces is integrations of graphical UIs. Integrations of components of UI can be achieved by combining presentation front-ends. The idea behind it is achieving a composite application that utilises individual characteristics of components that are put into it. The need for such applications is manifest and it is already extensively used in modern software development, for example: real estate information overlay for Google Maps, personal web-based homepages, etc. Unlike data and application integration, UI integration composes programs by using their own user interfaces and the presentation layer of the composed application is composed on its own. This approach can be used in cases where data of application integration is not practical.

An interesting survey can be found that was done by Myers & Rosson (992a). It is an old research, but results of it are still valid today. Authors of this work tested 74 companies that

work in software development and representing a variety of countries and types of organisations. They asked them what is a percentage of UI devoted programming in the whole project. Average value was 47.6%, as can be seen from Figure 16. User interfaces are extensively more complicated today and thus the number can be expected to be slightly higher. However, even the original number reported by the authors indicates large importance of UI programming for a successful project.

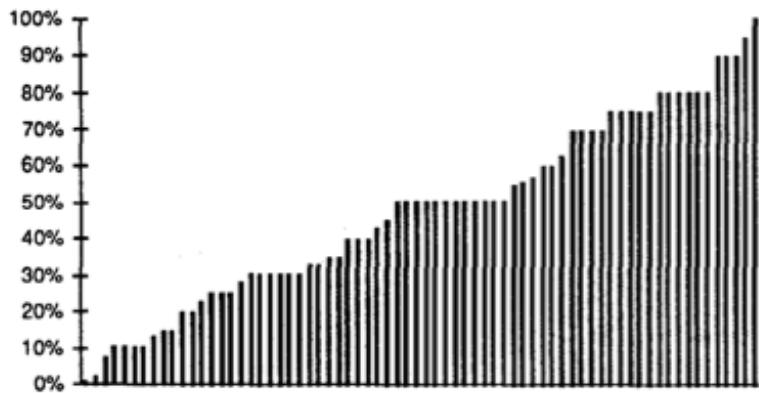


Figure 16: The percentage of code devoted to the UI programming, based on 71 test cases.

(Myers & Rosson 1992a)

Unfortunately, there is still a lot of research and using “trial and error” method to be done in the future in finding “The perfect UI”. Meanwhile, small advancements are being made in the field and knowing them is important for creating a successful cloud computing application.

3.2 Problem of customisable user interfaces in modern Computer Science

Software developers implementing multitenant web-based systems have all at some point wondered what the available Customisation Realisation Techniques (CRTs) are. This may lead to a research question: What are the possible ways of achieving a required level of customisation and configurability in multitenant web applications? The following sections discuss this issue from a multitenant application developer’s point of view.

Patent on “Customisable user interfaces” states that: “The goal of these customizing applications is to provide a more user-friendly interface to potential customers or clients in addition to attempting to provide a sense of personal service to individuals accessing a

company's web site." (Halabieh 2003). This patent has been filled in 2003. Since then, customisation of user interface has been a well-discussed by researchers topic and development of techniques for improving existing tools of customisation has been a point of attentions of thousands of companies around the world.

Referencing Myers (2003), customisation of user interface can be achieved by providing possibilities for users to change or extend applications using languages of programming, for example, AutoCAD provides Lisp for customisation, and many Microsoft programs use Microsoft's on language Visual Basic. More effective mechanisms for users to customise existing applications and create new ones are required.

It is important to have in mind that the terms customisation and personalisation are often used distinctively and sometimes interchangeably. Occasionally, personalisation is used to describe presenting content to individual users based on knowledge of who they are. Another example might be presentation of a logged in user with information about that particular user of the application, making it easy for that person to access often-used information. In other cases personalisation refers to giving users ability to define what parts of the application they need to use. Customisation is mostly. Updated 12.12.2010. Referred as setting certain preferences of the program that affect how it behaves. For the purposes of this research these terms are considered interchangeable and a word "customisation" is used in all cases.

Any software engineer who has background in development of graphical applications would admit that a stage of working on the user interface component of the program is often the most time- and effort-consuming stage of the development. Reusing of components can be beneficial for the UI development. Many application frameworks, such as JavaServer Faces 2.0, offer ready-made components like buttons, menus and bars (Daniel et al. 2006). Having this in mind, it is worth mentioning that development of custom customisable components that add a concept of reuse to the UI can save time for software engineers and enhance user experience.

As Miller (2003) perceptively states, unfortunately, often such aspects of software as scripting and customisation support get suspended in favour to more vital problems like feature set, performance, reliability, and usability. However, a problem of improving current methods of adding customisation to software is a critical for consideration topic. According to Spolsky

(2001), if a targeted group of consumers has different user communities (or the same user with different jobs), one may need different user interfaces, customisable user interfaces or both.

Customisation may mean compromise, since it is hard to predict what exactly tenants of a service require to be customisable. Let us turn to Salesforce.com once again. As indicated by the staff of the company at Salesforce.com (2012), “More power to customise” often is a top-ranked wish of business application users. The challenge for Information Technology organisations is that such power usually comes with big expenses, including increased project costs and ongoing risks. Faced with customisations that are costly, resource intensive, and difficult to upgrade, IT organisations often must make painful compromises or persuade users to accept “plain vanilla” programs. However, making users go for compromises may mean decreased attractiveness of the application and better ways of customisation should be researched.

Furthermore, the most important topic of research in the field of the presentation layer of web-based application recently has been a problem of customisation of user interface in web applications, which is described in the next section.

3.3 Customisable user interfaces in web-based applications

“We’re at a tipping point, where mission-critical applications are moving into the cloud” says an analyst with Nomura Securities Rick Sherlund (Hardy 2012). And, since, as one might state that UI is the most important asset in a successful commercial piece of software, development of customisable UI in SaaS has become a point of interest of a great number of researchers around the globe.

In situations when desktop application do not provide APIs users that wish to customise their programs need to resort to automating the UI, often called screen scraping. Cross-application solutions that record macros allow users to record mouse movements and keystrokes. Once the action is required again, the macro action can be executed. The downside of this approach is inability of macro records to know a state of the application they are working with based on an application’s display. Solution like Triggers Potter (1993) and VisMap Zettlemoyer, St.

Amant (1999) deal with this problem by interpreting the display contents at a pixel level, but this technology is challenging and resource-consuming.

Interpretation of desktop application output is difficult. However, web application provide their output as easily-readable HTML code, making screen scrapping more accessible. Using automation is being practised in web development today. Nevertheless, it does not provide a full level of customisation required for most projects, which needs to be investigated.

Present-day web applications, unlike dedicated desktop solutions, do not employ a concept of haystack. Turning to Quan et al. (2003), one finds that with the haystack, continuations and other supporting abstractions could be used to help users store their commands in operation and continue in convenient time. This technology is difficult to imagine in terms of web-based applications, since storing such data of user operations on the server side is difficult to imagine and local storage compatibilities of web browsers are still not perfect and saved information can be vulnerable.

Web-based user interfaces or web user interfaces (WUI) are a subclass of GUIs which accept input of users and generate output in a form of either an updated version of the page where input was entered or a new web page. They are usually viewed with help of a web browsing program.

Often tenants of a certain company require web forms that have unique for their needs sets of functions. Customisation of the way the form looks like and feels to end-users is needed. According to research of Hadlock (2011) it is a common requirement in the world of Internet today. Popularity of personalised homepages and dashboards, such as MyAOL, MyYahoo! and iGoogle, has been rising and possibilities for changing user interface of web pages offered to customers has been in great demand.

People did not have an urgent need to use customisable interfaces in Internet-based applications straight away. Based on Fox & Patterson (2012b), there was a problem with classic websites that were done in the early days of Internet: HTTP protocol is stateless and adding such features as a shopping cart in a eBusiness website or a checkout page was not possible. In the mid 1990s the problem of adding such features rose and a new technology was introduced: cookies. One way of providing client-specific interfaces that can supply a

satisfactory level of personalisation is enabling cookies for the website. With this technology users can configure elements of a certain page, for example widgets in a personalised homepage like My Yahoo and BBC.co.uk (see Figure 17).

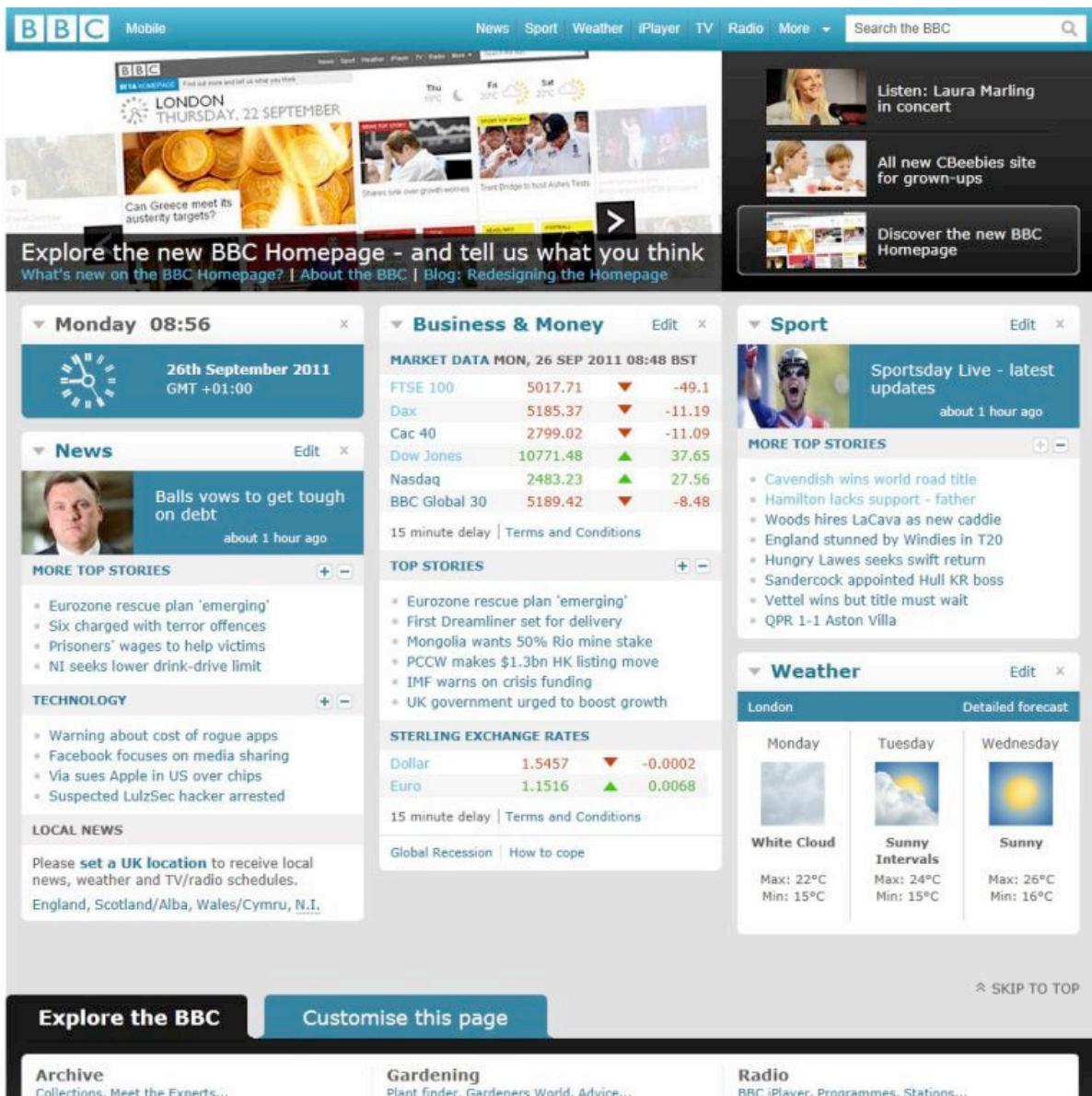


Figure 17. Personalised using cookies homepage of BBC (<http://bbc.co.uk>).

It is important, however, to always check if a client is legit in SaaS. A client could be a “bot” (AI controlled program that pretends to be a human) or it can be a person with an outdated browser that does not support required features and have critical flaws. In development of SaaS a response received from the client must always be checked to avoid crucial mistakes. Using cookies for customisation of the website is a reasonable choice for a developer, while it removes workload from the servers by storing configuration information

locally. Nevertheless, there are better ways of implementing customisable UI in web applications.

Dynamic web pages can currently be achieved with both server-side and client-side languages of programming. It is worth mentioning such server-side technologies as ASP, ColdFusion, PHP, Perl and JSP. Also, such client-side scripting languages like JavaScript or ActionScript, Flash are often used to manipulate media elements of the presentation layer of the web page.

Based on the article by Selvitelle (2010), one great example of a modern-looking and properly working interfaces is Twitter (see Figure 18). The current version of it was done almost entirely using JavaScript open-source libraries (e.g. jQuery, LABjs). In this website users and even companies can adjust many settings of the visual appeal of the program to make personalised profiles that could be used for promoting and business. This website is an excellent example of a level of customisation that is achievable using primarily client-side technologies like JavaScript.



Figure 18. A new look of twitter.com, offering great tools for customisation.

Unlike client-side languages of programming, a much greater level of customisation can be accessed using server-side languages. This level of customisation is usually used in enterprise

programming and systems that are used commercially, e.g. ERP systems for bioenergy management (see Figure 19). In this kind of programs settings that define levels of customisation are often stored in a remote database, not locally. And they can be fetched from it using server-side scripting with Java or any other server-side language of programming.

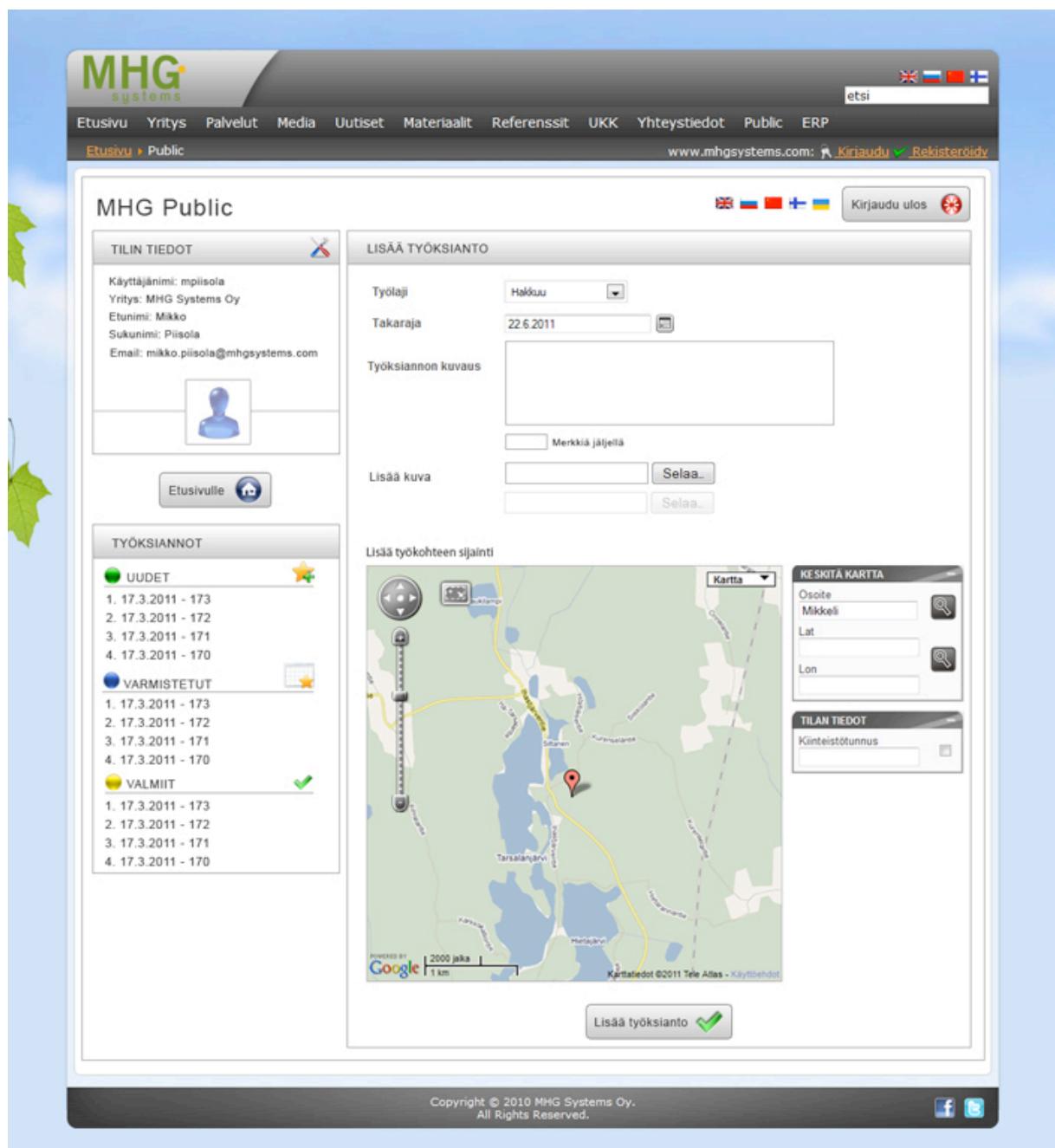


Figure 19. MHG Public web interface for bioenergy ERP system. Customisation is provided by JSF 2.0 technology. Such elements of the design as a logo, icons, fonts, colours are customisable. Reordering of the elements of the page can also be implemented, if required by the tenant.

Turning to the work of Miller et al. (1997), web application users, unlike customers of traditional desktop solutions, can be rather intolerant of unadaptable interface since they usually have alternatives in form of other websites offering same services.

4 CUSTOMISABLE WEB FORMS IN MULTITENANT APPLICATIONS

Forms play a role of “bridges” between complex record of data and users, average humans. This role is a very important one, since most users still find computers intimidating, let alone databases. The easier the interactions between users and the application can be made, the more successful that application is likely to be. Forms “humanise” the persistence layer of the application.

Web forms are a type of forms. Web forms allow data that is entered by a user to be sent to a server that processes it. Similar, to forms used in traditional desktop solutions, web forms consist of such elements as radio buttons, checkboxes and text fields. For example, web forms could be used to gather survey data, enter credit card information, or register account on a website.

Turning to Whitehorn, Marklyn (2006, 14 - 15), forms are devices which allow one to look at and edit the data stored in the database. One can usually alter the table directly and perform both editing and viewing, but forms are preferable, since they tend to be more attractive and easier in use for users. Forms can be thought of as filters between the tables of data in, usually, relational database and the users of the database. Humans usually prefer to be able to see each record of the table of data individually and not as rows neighbouring other, often unnecessary for the specific user in the specific time, data.

Web fields are what web forms consist of. Fields are one of the elements of web forms. Input fields can contain such elements:

- Text — a simple text box that allows input of a single line of text.
- Checkbox — a check box.
- Radio — a radio button.
- File — a file select control for uploading a file.

- Reset — a reset button that, when activated, tells the browser to restore the values to their initial values.
- Submit — a button that tells the browser to take action on the form.

With combinations of these elements web forms can allow input or almost all types of data that can be imagined relevant in web-based application solutions. In an ideal Software as a Service application all of these aforementioned elements of web forms should be fully customisable.

A problem with web fields in multitenant applications is a strong connection of them to specific fields in a database. A situation where a piece of information desired to be entered in a field by one tenant is different from a wish of other tenants concerning data entry in the same field in a web form can easily be imagined. In situations like that tools allowing overloading of web fields should make possible configuration of web forms by each tenant individually.

Today's UI tools mostly help with the generation of the code of the interface, and presume that the fundamental user interface design is complete. Tools to help with the generation, specification, and analysis of the design of the interface are also needed. Creation of customisable web fields in web-based applications require much more advanced and, often coded from the scratch, tools.

Additionally, data entry should also be controlled in web form fields. Turning to Whitehorn & Marklyn (2006), one can find that it is important to specify what kind of information may be entered in each field of the form. Data control can also be implemented on the database level, which is more important, because multiple forms can rely on the same table in the database. Moreover, if the project is developed in a team, database is easier to manage and keep organised than multiple forms. Theoretically, data control can entirely be implemented on the form level, but it is a sign of bad design patterns. With JSF 2.0 framework data control handling can be managed easily.

Several technologies available today can be considered as suitable techniques for creating customisable web forms in multitenant applications. JavaServer Pages technology was chosen to produce a test application to support this research. This server-side language of

programming reuses CGI (Common Gateway Interface) concepts in their APIs but dispatch all web requests into a shared virtual machine.

Development of extensively-customisable web-based input fields may seem like an unimportant and too vague for consideration topic, but it is an important part of Internet of the future, fully customisable and offering exactly what a user requires to see.

Problems of multitenant architecture, customisable user interfaces and tenant-custom functionality in web applications are among the most discussed and research topics of Computer Science today. Thus study on a particular way of implementing multitenant forms inside of applications written using JSF 2.0 and MySQL technologies was chosen as a topic of this research to contribute to these topics and to the world of Information Technology and Computer Science in general.

5 TECHNOLOGIES USED FOR THE TEST CASE

From a developer's point of view, choosing the suitable architecture and tools for developing a web application is a vital decision, which assumes thinking about the following dimensions of the expected program: 1) the size of the database, 2) having dynamic components, 3) existence of customisation in the website, 4) overall expectation from the design.

A problem of balancing between these four aspects of development of SaaS application is a difficult one. Achieving high levels of customisation and design can be considerably easily attained by hard-coding appropriate web pages, but developers using this technique must sacrifice scalability and responsiveness to updates. On the other hand, automatic HTML generators, which are capable of returning web pages based on data stored in a database, can ensure a good level of scalability and responsiveness to updates. Compensating between these two, it may seem like, opposite approaches is challenging. Detailed consideration of tools to be used needs to be performed before starting development.

The multitenant SaaS application with customisable web forms, which is developed as a practical prove of this study tries to be an example of a harmonically working application utilising main principles of cloud computing and Software Engineering in general. It uses technologies, which are shortly described in this section.

5.1 JSF 2.0

According to Anderson (2006), with time when projects become more complicated, the code becomes the “tail that wagged the dog”. It steps out of the Web server and most tasks are handled by frameworks for web development of Web 2.0 websites, which are available in plenty. Providing the pages, filtering input provided in form fields, and delivering new pages as a result, must be implemented using some sort of server-side scripting technology as the back-end of the web page, or by a framework. These tools are normally rather different from the tools used to work with the client-side pages that the user sees.

The sample program developed as a part of this work uses Java language of programming. One of the most noticeable aspects of this language is its openness and the large amount of companies, tools, and technologies use the language. Additionally, a large variety of hardware depends on Java, starting from handheld devices like mobile phones and ending with large enterprise systems.

Special significance in Java development plays prior to development phase of selection of tools. There are thousands of tools, both commercial and open-source, available on the market today. The selection of the ‘best’ set of tools for a Java project can prove to be a hard task. Some tools can be changed later in favour to other technologies with acceptable cost (such as, for example, switching to another issue tracking system, as long as the old one provides some data export facility), but others cannot be changed without altering most of the done work.

In the scope of this research JavaServer Faces (JSF) 2.0 Java-based Web application framework is used. This framework is intended for simplification of web-based user interface development.

JSF is included in the Java EE platform, so engineers can create applications that use JSF 2.0 technology without adding any extra libraries to their Java-based projects. JSF is capable of using such bean containers as Spring and it works with almost equal performance and output as a standalone web framework. Developers of various skill levels can build web applications with ease by utilising such aspects of the technology as assembly of reusable UI components

into web pages, connecting these components to an application data source, and wiring client-generated events to server-side event handlers.

JSF is a request-driven MVC web framework. According to JavaServer Faces (2011), it has two main functions. The first one is generation of user interfaces, usually using HTML language. As mentioned before, this UI is represented on the page as a tree of components and elements in the UI. The actual interface is generated when the component tree is rendered. The separation between user interface and component tree allows JavaServer Faces to support such markup languages as XHTML.

The second main function of this framework is responding to user-generated events by calling server-side listeners. This process is usually followed by generation of another web page with another UI or an update to already showing UI. JSF can be called an event-driver framework.

For summarisation, a list of advantages of JSF 2.0 compared to other frameworks used in development of cloud computing can be outlined using the work of Khan (2010):

1. JSF provides a substantial API with associated tags for creating HTML forms with complex interfaces.
2. Large community of developers and number of external libraries.
3. Event handling.
4. Managed beans. Meaning that Java beans can be automatically populated based on request parameters. With JSF's utilities parameter processing is significantly simplified compared to other MVC frameworks.
5. Form field conversion and validation.
6. Centralised file-based configuration. Rather than hard-coding information into Java programs, many JSF values are represented in XML or property files.
7. Consistent approach
8. Support for Ajax, jQuery, Dojo and other interface libraries.

Further, interpreted languages like PHP are in almost all cases slower than compiled languages like JSF. The downside of using JSF technology is that usually files created by this technology get compiled and complex, so once the server is up and running and doesn't get

changed anymore, the performance will be better than a PHP script that gets interpreted every time a request comes in.

5.2 XHTML

Implementing user interfaces for Internet generally uses different tools than building GUIs for desktop usage. Additionally, the technology and tools are changing quite rapidly. Therefore, this is a brief overview of one of them - XHTML.

One reason for the wide spread acceptance of the World Wide Web was the concept of a universal client - the web browser, based on the use of a key content language, namely Hypertext Mark-up Language (HTML), based on W3C-XHTML (1997). Simple sites of Web 1.0 era were collected from static text and images with embedded links, and these can be created by directly typing the underlying HTML code. As an alternative, the designer/developer may also use more interactive tools, for example Microsoft FrontPage, which therefore works as an Interface Builder. Pages that are required to be dynamic can also be authored by using scripting language embedded in the html code, for example: Javascript or VBscript (Visual Basic Script). On the other hand, a specialised animation language can be used, such as Adobe Flash.

According to W3C-XHTML (2004), the first version of XHTML recommended by the W3C - XHTML 1.0 came in 3 various, namely strict, transitional, and frameset. The main purpose of XHTML 1.0 was redefining HTML as an XML program, and different variations of it provided a transaction stage for smooth transformations. The idea of separately defining content and presentation was not new. The main goal of XHTML 2.0 is to provide a cleaner and more structural mark-up for describing the content only of a hypertext page. Therefore, allowing proper marking up of content in a practical way, and a clear defined content, style and behaviour separation.

In the scope of this research XHTML is used, while it is a common technology used as a way of representing web pages generated by JSF 2.0 framework. Since, JavaServer Faces is an MVC - driven application framework, it can be said that XHTML is utilised to output views of an application.

5.3 CSS

Cascading Style Sheets (CSS) is used as a tool for altering the way view pages generated with help of JSF framework are outputted. It was created in 1996, but it is still a de-facto standard for styling web pages. Using CSS brings another layer of separation into application framework. With this technology document content (generated with XHTML) can be separated from document presentation.

Based on Andrew (2007, 1 - 2), styles can also be defined using standard HTML code. However, using pure HTML code is insufficient when adding modern-looking elements to the style of the page. As an example, using `...` tag from HTML syntax describes font that is to be used for text between an opening and a closing tags. On average, hundreds of just this tag would be used to describe font of an average-looking and with average load on content page in a modern website. If changing colours, ways of outputting images, positions of elements on the page is required, a number of HTML tags used for describing style can easily reach thousands. It is extremely inefficient. Cascading Style Sheets makes designing pages and web forms easier and faster.

5.4 MySQL

MySQL is chosen as an RDBMS for a test case in this study. It is currently the world's most used database management system. MySQL comes with MySQL Workbench which is a tool developed to be used specifically with MySQL, see Figure 20. This application significantly simplifies designing a scheme for a database and maintaining the database. This program provides data modelling, SQL development, and extensive administration tools for server management, user administration and other tasks.

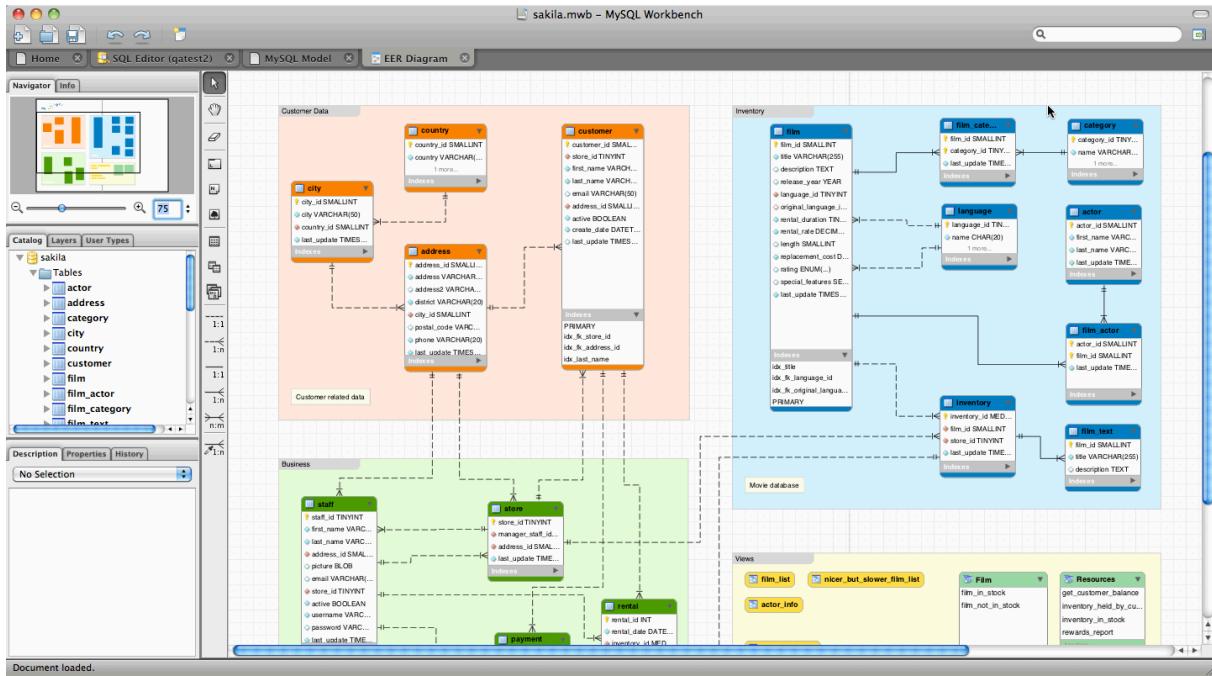


Figure 20. MySQL Workbench 5.2 running on Mac OS.

5.5 Netbeans IDE

The NetBeans IDE (Integrated Development Environment) was the first free and open source tool providing support for building J2EE web tier applications in the beginning of 2000s.

With the 4.1 release, the NetBeans IDE (see Figure 21) was developed even further and it include full support for building complete J2EE 1.4 programs, as well as supporting the key capability of J2EE 1.4 web services. At the moment of doing this study Netbeans IDE was already in its 7th iteration. It is considered to be one of the best tools for Java development. (Keegan et al. 2006)

This IDE has full support for Java EE development and that is a main reason why it is chosen for this research. All key features of development SaaS with Java EE are fully integrated with this tool, which provides a complete environment for creating and debugging J2EE applications. It has integrated support for Glassfish 2.x application server that is used. With a single click the NetBeans IDE can start the application server, deploy the program, and run the application in a mode ready for real-time debugging.

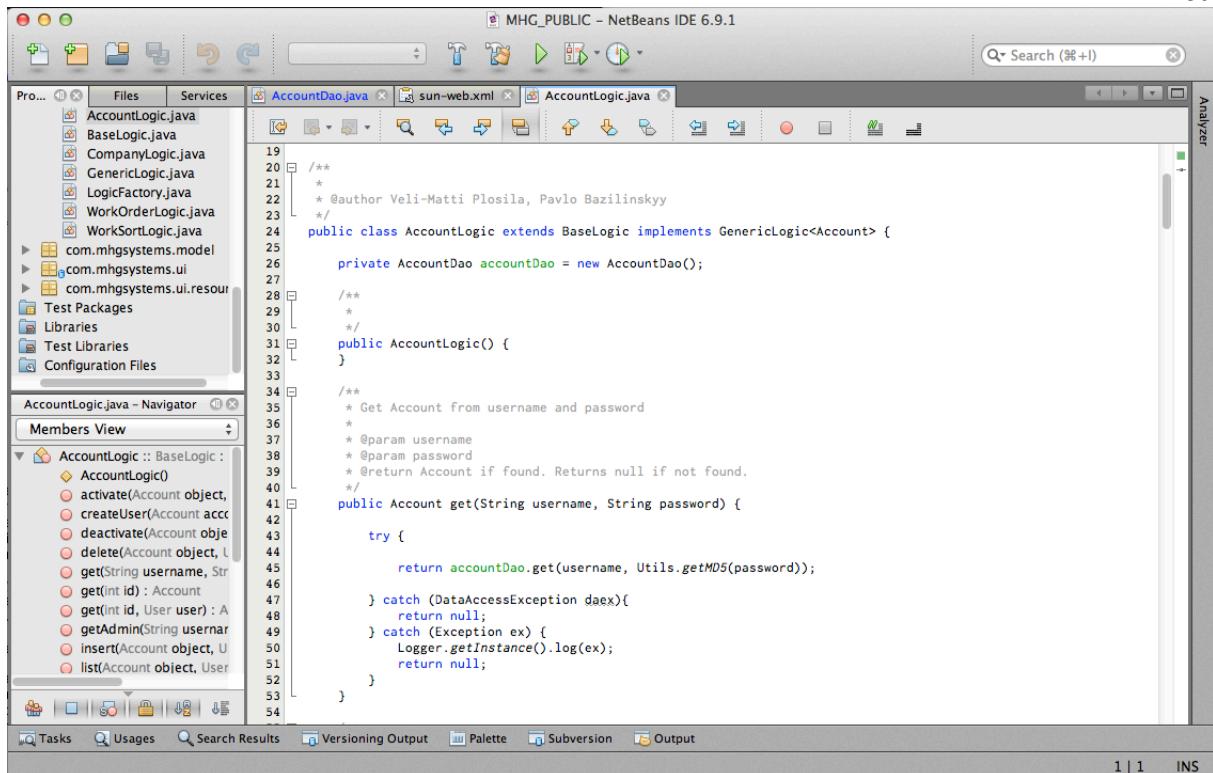


Figure 21. Netbeans IDE 6.9.1 running on Mac OS.

In the middle of development of the test case for this research support for Glassfish 2.x server in Netbeans 6.9.1 broke down. Netbeans 7.1.1 with Glassfish 3.x and ICEFaces 3.0 was used for further development. All programming code and a database design described below were developed having these technologies in use. Most of it should be backward compatible with Glassfish 2.x and ICEFaces 2.x running on top of JSF 2.0 framework.



Figure 22. Problem with an occupied port in Netbeans IDE.

On the other hand, Netbeans IDE as any software has its flaws. One issue that rises during development of JSF 2.0 applications for GlassFish server on Mac OS X is a problem with

occupied ports. More precisely, if an application is deployed with runtime errors, the second time that the application is deployed, GlassFish cannot start because a port (in case of Mac OS X port number 1527) is occupied. An error message that is returned by Netbeans IDE can be seen on Figure 22.

6 THE TEST CASE APPLICATION

“Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.” (Hunt & Thomas 1999). It is one of the core principles in modern software architecture Don’t Repeat Yourself (DRY).

A current world of Information Technology has a gap in terms of applications that provide means of working with multitenant web forms using MySQL and JSF technologies. To support this study and answer to the research question “How to develop the most optimised and the most versatile multitenant web form using JSF and MySQL?” a test case application was developed. This application serves as a framework for management of web forms that can be used by multiple tenants. In this chapter a workflow of organisation and management of forms is described. Additionally, technical aspect of the program such as models, views and controllers are described. Listings of programming code and various pieces of the application are given in places where they can contribute to better understanding of logic of the application. Additionally, references to appendices are given. In the appendices a number of functions from the application and a MySQL scheme are described in greater details.

The central asset of the test case application is a web form. Web forms can be created, managed and viewed/filled in. Furthermore, web forms can be inherited from other forms and used by different tenants, which brings an aspect of multitenancy to the program. Users of tenants can be granted permissions for working with certain web forms. When a web form is created all users of the tenant are given rights to manage the form. A process of rendering web forms can be summarised in a flow chart outlined in Figure 23. Working with web forms is described in details in this chapter.

Moreover, a concept of “predefined web forms” is described in this study. Predefined fields can be created and maintained by users of the tenant. These objects can describe commonly used web fields, such as a list of countries, a group of radio buttons for choosing one’s gender,

etc. Later, these fields can be used when new web forms are created; it saves time and improves usability and performance.

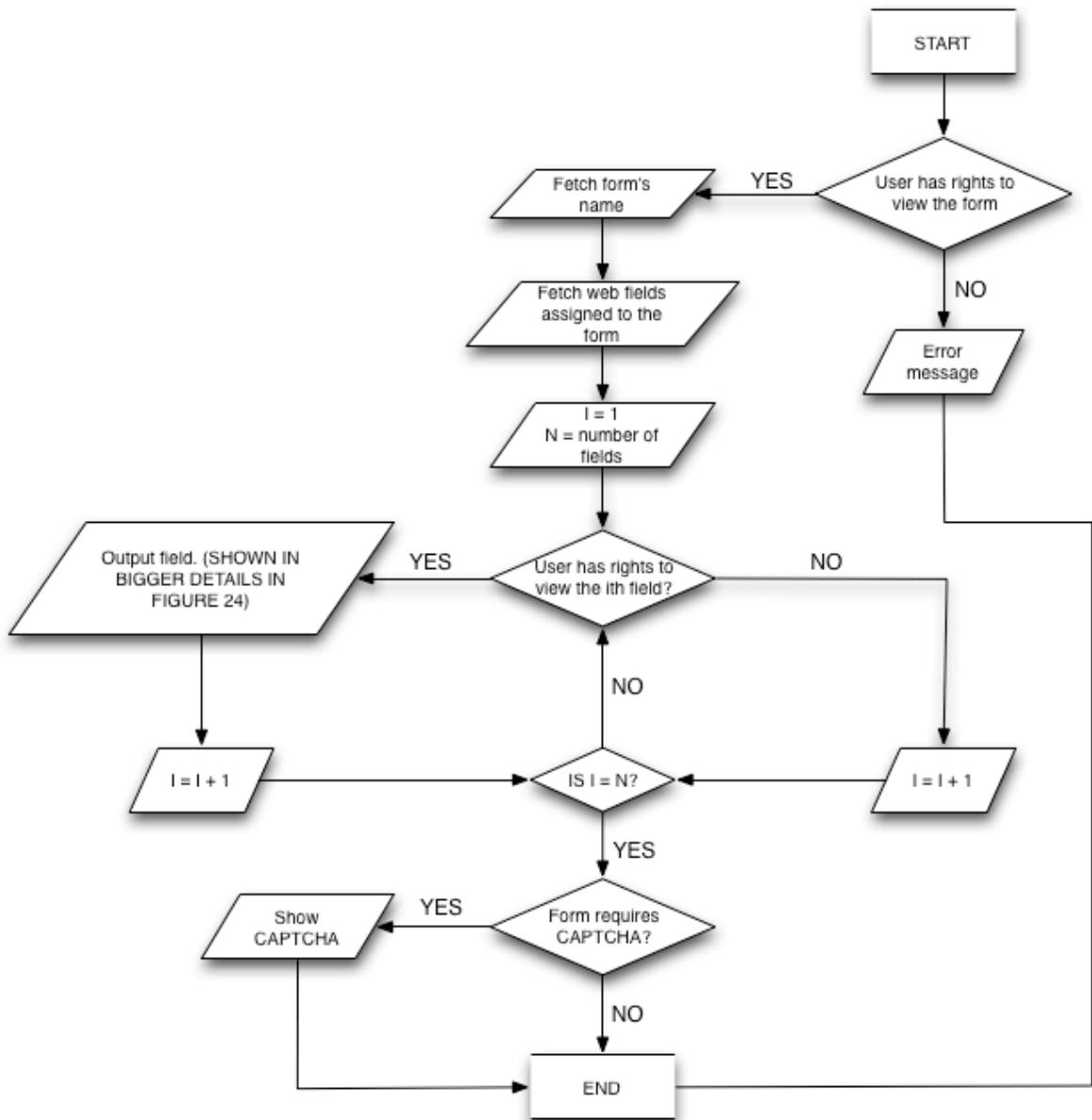


Figure 23. Flow chart describing a process of rendering web forms.

All web forms are compiled from web fields. Web fields are fully-customisable and it allows tenants configure the test case program for their liking. When web fields are rendered a number of properties such as label text, type of the field, label colour, etc. are fetched from the database. Users of tenants can be granted rights for working with certain web fields. When a web form is created all users of the tenant are given permissions to work with fields of the form. A process of rendering a web field for a given form can be summarised in a flow char on Figure 24. Working with web fields is described in details in this chapter.

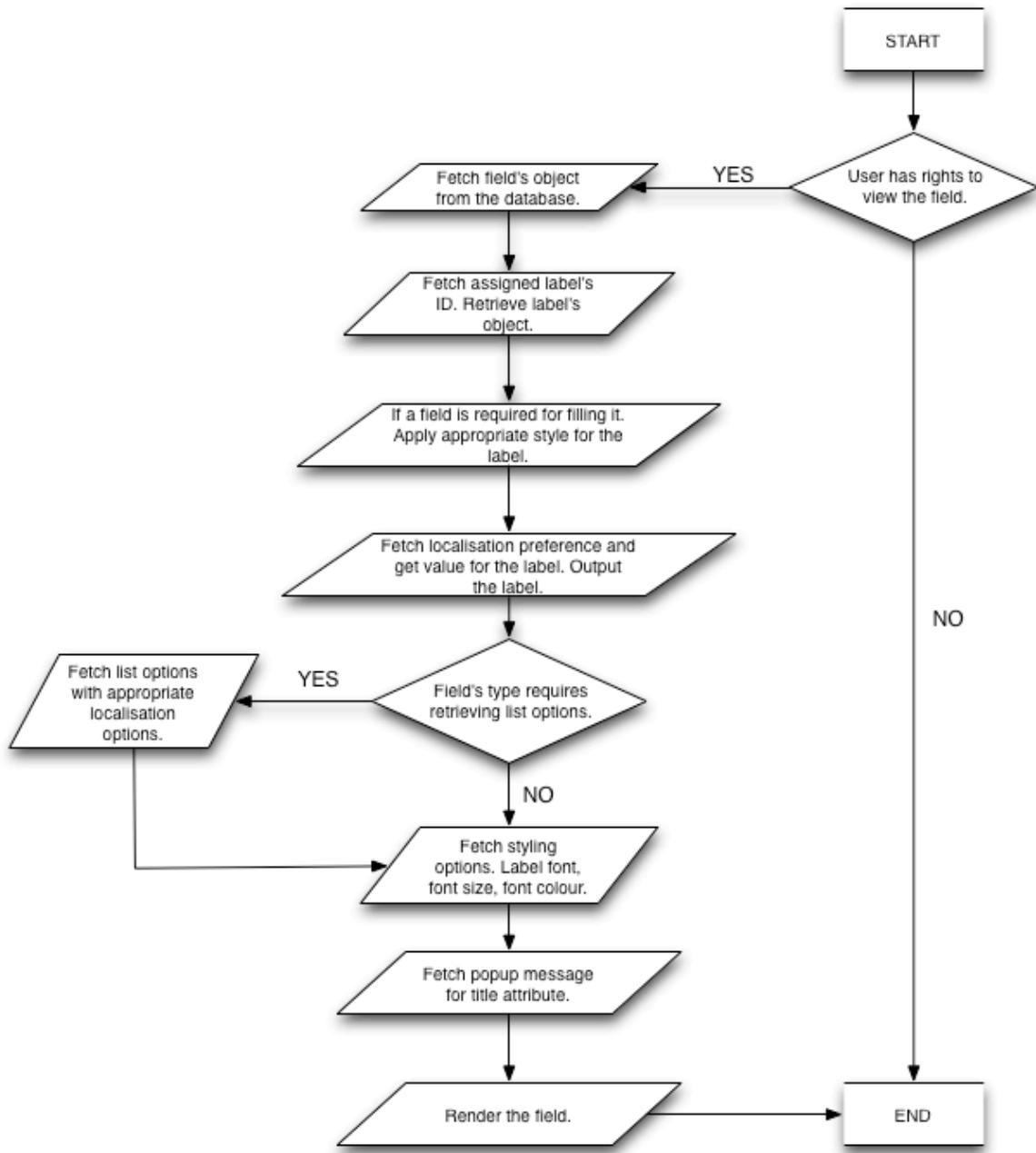


Figure 24. Flow chart describing a process of rendering web fields.

An important asset in the test case application is a tenant. Multitenant applications cannot be imagined without tenants using them. Additionally, tenants can have users. Users are actual people that work for tenants and use the application. Different users can have different rights within the application.

The code in the test case of this research was done with an intention to implement “RESTful” (Representational State Transfer) programming, which was first described by Roy Thomas Fielding in his doctoral dissertation in Fielding (2000). Creators and supporters of REST defined its main goals to be: Scalability of component interactions, Generality of

interfaces, Independent deployment of components and Intermediary components reducing latency, enforcing security and encapsulating legacy systems. This technique has proven to be incredibly effective in cases of creation of SaaS. The main focus in this research is given to generalisation of interfaces and building of appropriate easily-extendable APIs.

A common technique of MVC applications - CRUD (Create, Read, Update, Delete) - is used in the thesis. Fox & Patterson (2012b) state that a matter of good practice of writing applications using MVC is avoiding adding any logical components into views: having all “real code” outside of boundaries of view components. Indeed, this principle helps with organisation of code inside of the application, and it is utilised heavily in the test case. One exception to this rule is usage of JavaScript for enhancements of the UI.

In section of this chapter below a close look into application code of the test case is taken. The project consists of Java classes, XHTML views, and CSS files that can be divided into a number of categories: models, views, logic, DAO etc. These categories are described in details in the sections below. Additionally, examples with comments and explanations are given to provide better understanding of processes that take place once the test case is deployed to the application server. All examples are given with English language set as a localisation preference in the program. Furthermore, one may find detailed listings of application code in appendix 1 and a description of the scheme in appendix 2.

The test case application may be found on GitHub.com website where it is stored as a public repository. A link to the project is https://github.com/Hollgam/multitenant_webforms.

6.1 Description of views and user actions

This section gives short descriptions of all web pages that are present in the test case. It may be used for reference and better understanding of workflow of managing and using multitenant web forms. The test case program’s section created for end users consists of a main part with content and a sidebar on the left (see Figure 26). In the sidebar a user’s avatar picture and links for a quick access to such sections as the front page (it can be accessed by clicking on the avatar picture) and management of individual web forms. Web forms that are listed there are marked with their names and IDs. Also, flags of available translations are shown in a horizontal menu in the upper right part of the page.

All functions described in this sections can be accessed only if a user is logged in (except for registration).

6.1.1 Registration, login and logout

To register one should follow a “Register” link in a login prompt shown on any other page providing that no user is logged in from a browser used, see Figure 25. Also, **register.jsp** can be accessed directly. To successfully register as a new user, one should fill information for all fields marked as required. Other fields may be filled as well, but it is optional (it can be changed later by editing account information of a new account).

Multitenant webforms

REGISTRATION

Please fill fields below. Fields labeled with a * are compulsory.

*Username
Username must be 5 - 15 characters long.
No special characters are allowed.
 Username is free!

*Tenant

*First name

*Surname

*Password
Password must 6 - 15 characters long.

*Confirm Password

*Email address

Please enter text in the image

VSGXYI

Figure 25. Web page used for registration.

In this view entered value for email address is checked using regular expression ”.+@.+\\\\.[a-z]+”, which is processed by **EmailValidator.java** validation bean (see a detailed listing in appendix 1.5). At the bottom of the page a button to the front page is shown for quick access along with a button for clearing entered data. Also, a button “Check Availability” can be used to check if a desired username is not occupied by other users. After successful registration a user is redirected to the front page with a message about successful registration shown. Users can log in after successful registration.

To login any page that exists in the program may be opened or **login.jsf** file may be accessed directly. “Remember me” option means that on a new session from the same browser no login

will be required for a current user. To logout a link “logout” in the upper right corner (next to translation flags) should be pressed.

6.1.2 Front page

A front page is essentially a window for managing web forms that are accessible to end users. When a user logs in he/she sees a list of web forms that can be viewed/edited. Additionally, a request for generating a new form as well as inheriting an existing one may be accessed from this page. This is a main page in the program, which serves as a dashboard with links to common tasks that users can do, while logged in to the system, see Figure 26.

The screenshot shows the 'Multitenant webforms' application interface. At the top, there's a 'Logout' button and some language selection icons. On the left, there's a sidebar titled 'ACCOUNT INFORMATION' containing user details: Username: user1, Tenant: MAMK, First name: Pavlo, Surname: Bazilinskyy, Email address: pavlo.bazilinskyy@gmail.com, and a small profile picture of a person with glasses. Below this is a section titled 'WEB FORMS' with a status indicator 'TO BE FILLED' and a green checkmark. A list of web forms is displayed on the right, each with an ID, name, and number of web fields. Each entry has 'View form' and 'Edit form' buttons. The forms listed are:

ID	Name	Web fields	Action	Action
ID: 1	Name: Registration	4	View form	Edit form
ID: 3	Name: Registration	1	View form	Edit form
ID: 4	Name: Webform test	16	View form	Edit form
ID: 26	Name: NEW USER	9	View form	Edit form
ID: 27	Name: Company assignment	8	View form	Edit form
ID: 50	Name: Work order	5	View form	Edit form
ID: 51	Name: Registration	5	View form	Edit form
ID: 53	Name: FORM 1	0	View form	Edit form
ID: 54	Name: Form 2	0	View form	Edit form
ID: 55	Name: Test case	0	View form	Edit form

At the bottom, there's a 'Next page' button.

Figure 26. Front page.

6.1.3 Creating new web forms

To create a new web form a link “New work form” may be clicked on the home page or `newWebForm.jsf` page may be accessed directly, see Figure 27. A person that is using the test-case program may add new web fields to a new web form. When new web fields are added to the form, such values of web fields are inputted and processed: label text, type, whether the field is required for filling in or not and a position in the form. Furthermore, an

existing label can be chosen instead of creating a new one. This action can be performed by user clicking on a green icon next to an input field for a label value. When that icon is pressed a JavaScript function is invoked. This function makes a list of all available labels appear below a location for inputting value for a new label. When an existing label is chosen, a “label_id” property of **webfield** entity in the database is updated with a value corresponding to the label that was chosen. Hence, when that label is changed, it is rendered differently in all web field that use it. And not only for the field that is was initially generated for.

Figure 27. Web page where creating new web forms is handled.

Secondly, new web fields can be added to the form. It can be done by clicking on an empty field below a list of existing web fields. Once a user clicks on the field, another line for adding next new field is rendered lower. It is done using jQuery library and a JavaScript function, a snippet of which can be viewed below.

```

$(document).ready(function() {
    for (var i = 1, i < 40, i++) {
        var id = "editForm:editList:" + i + ":editPanel",
            $('[id=' + id + ']').hide(),
    }
}

var id = "editForm:editList:" + "0" + ":webFieldLabel",
$('[id=' + id + ']').click(function(){

```

```

var id = "editForm:editList:" + "1" + ":editPanel",
$(['id=' + id + '']).show(500),
})

... ( code omitted) ...

}

```

A list of 40 web fields is generated in a backing bean, after that they are passed to the page and get hidden by a jQuery function. A number of members of an array of new web fields can be extended, if required. Later, all fields, except for a field with zeroth index are hidden before the rest of the view is rendered. The same approach is used in views that help tenants edit existing web forms.

A user has another option for creating new web forms. A new form can be generated based on another web form, the “mother” form. In this case a newly generated form becomes a “child” web form inheriting its mother’s properties. This process can be performed from a **newChildWebForm.jsf** page, or by clicking on the link “New CHILD web field” in the upper part of the home page. Once a logged in user is redirected to that page he or she needs to pick a web form that is to be used as a “mother”. It must be noted that only those web forms that have a positive value of “can_be_mother” property can be used to generate new web forms.

When a base web form is chosen a view is updated and a user sees a list of all web fields that the “mother” web form consists of, as can be seen on Figure 28. A user has a choice of having web fields in the new form inheriting properties of web fields in the “mother” form, it can be done by triggering “Child” checkbox next to web fields. If such action is performed, web fields become bound to their “mother” fields and all updates that are performed on mother fields get copied to the “child” web fields (described in details in section 6.1.5 Editing web forms).

The screenshot shows a web-based application for managing web forms. On the left, there's a sidebar titled 'ACCOUNT INFORMATION' with user details: Username: user1, Tenant: MAMK, First name: Pavlo, Surname: Bazilinskyy, Email address: pavlo.bazilinskyy@gmail.com. Below this is a small profile picture of a person with glasses. To the right, the main area is titled 'CREATE NEW WEB FORM BASED ON ANOTHER'. It shows a 'Name of mother form' dropdown set to 'Registration' with an 'Update' button. A section for 'Name of the form' has 'Inherited from Registration'. Below this are several fields: 'Email' (Email field, Required, 1, Child checkbox), 'Location' (Embedded map, Not required, 2, Child checked), 'Birthday' (Birthday field, Required, 3, Child checked), 'Administrator' (Checkbox, Required, 4, Child checkbox). At the bottom are 'Submit', 'Reset', and 'Back' buttons.

Figure 28. A new web form generated based on a “mother” form.

Such properties of web fields as “type” and “required/not required” cannot be changed in a process of creating a new web form based on a “mother” form. It is done in this way to achieve a level of security for an action of inheriting web form’s properties. However, these properties can be modified for “non child” web fields on later stages by going to **editWebForm.jsf** view. Once a “Submit” button is clicked a method `public String newChildWebForm()` is invoked. A listing of this method can be found in appendix 1.2.

6.1.4 Viewing and filling web forms

To view an existing web form a link to that form should be clicked. Before being rendered a web form is parsed in a backing bean class, one may find a method that is responsible for a process of parsing the form in appendix 1.3. All web forms can be viewed. Users may be given rights to fill web forms with relevant information. If a user was granted permissions to do so, a web form may be accessed through **fillWebForm.jsf** page or by clicking on the link, see Figures 29 - 31. After information has been put into the form, data leaves the system for routing to appropriate pieces of the database by external frameworks and modules. A web form may be filled in by a user once.

Once a user accesses a web page where a web form is shown an ID of the form is recorded in **SessionBean.java** and a list of web fields that are paired with the form is generated. Then,

web fields are outputted with `<ui:repeat>...</ui:repeat>` tags from JSF 2.0 framework.

Depending on a type of the field an appropriate set of settings is fetched and used for rendering objects the web page. In the test case web fields can be of these types (listed in a form: value of type - description):

- 1 - input field. Rendered with `h:inputText`.
- 2 - email address input field. External validation is applied. Rendered with `h:inputText` and `f:validator`.
- 3 - text area. Rendered with `h:inputTextarea`.
- 4 - birth date picker. External validation is applied. Rendered with `ice:selectInputDate` and `f:validator`.
- 5 - date picker. External validation is applied. Rendered with `ice:selectInputDate` and `f:validator`.
- 6 - drop-down menu. One value may be selected. Rendered with `h:selectOneMenu`.
- 7 - select one radio button group. Rendered with `h:selectOneRadio`.
- 8 - select one radio button group. Rendered with `h:selectManyCheckbox`.
- 9 - checkbox. Rendered with `h:selectBooleanCheckbox`.
- 10 - embedded map. Google Maps API is used.

Different settings for web fields are processed in runtime. Namely, web fields are marked as required or not required for filling in. This property of the field is fetched from a column “required” in the table **webfield** in the database. An example of this assignment can be seen from this piece of code: `<h:outputLabel styleClass="required#{webField.required}" ... />`. Fields are made compulsory by assigning class attribute to ”required1”, which is described in a CSS style sheet as follows (note: in case of the field set to be not compulsory for filling in the class is set to ”required0” and processed differently by CSS rules):

```
.required1 {
    background:url(./css-images/required.gif),
    background-position: 0% 50%,
    background-repeat:no-repeat,
    margin-left:auto,
    margin-right:auto,
    padding-left:7px !important,
    padding-right:2px !important,
    padding-bottom:1px,
```

}

Further, labels of fields can have such attributes as text colour, font size and family, etc. changed to meet needs of tenants. Such personalisation is achieved by utilising inline CSS styling as can be seen in this fragment of XHTML code: `<h:outputLabel styleClass = "required#{webField.required}" style = "color: ##{webField.colour}, font-family: #{webField.labelFont}, font-size: #{webField.labelFontSize}px," for = "webfield#{webField.type}" value="#{webField.label.en}" />`. Values for CSS attributes are fetched from a `webField` object, as was described above. These settings are tenant-specific, they cannot be modified to meet wishes of specific users. However, such adjustment is possible with little modifications to program's logic.

Moreover, a CAPTCHA image can be shown on the view. This image may be added to the form to achieve a layer of protection from hackers and bot programs. Rendering of it can be disabled for web forms, if required.

On Figures 29 and 30 one may see a web form ready to be filled in by users of a certain tenant (let us call it “tenant1” and its users “user1” and “user2”). In the sidebar on the left side of the version of the program viewable by these two users one may see “Tenant: MAMK” indicating that they are indeed assigned to the same tenant. Web fields in this form have its labels and other aspects configured to represent possibilities for configuration available in the test case application.

Views that are responsible for adjustments of properties of that web form are shown on Figures 32 - 34. By comparing these Figures one may get a better understanding of logic of working with web forms implemented in the test case for this study. In section 6.1.5 Editing web forms a process of configuring multitenant forms is described having a web form outlined on Figure 28 as an example.

Multitenant webforms


[Logout](#)
✖

ACCOUNT INFORMATION



Username: user1
Tenant: MAMK
First name: Pavlo
Surname: Bazilinskyy
Email address: pavlo.bazilinskyy@gmail.com

FILL WEB FORM — ID: 1

Please fill fields below. Fields labeled with a '*' are compulsory.

***Email**

Location



+/-
Map
Satellite

*Lat
*Lon

***Birthday** [Calendar icon]

***Administrator**

Please enter text in the image
BVKVFLU

Figure 29. Registration web form ready to be filled in as seen by user1 from tenant1.

User2's version of the form viewable on Figure 29 is inherited from the instance of user1's form. One may notice that user2's version has a field "Location" overwritten, where its type is different from user1's version. Additionally, field "More information" was added. Moreover, styling rules for labels and ordering of web field are different. Further, no CAPTCHA ("Completely Automated Public Turing test to tell Computers and Humans Apart") image is required to be filled in by users.

The screenshot shows a web-based form titled "FILL WEB FORM -- ID: 51". The form is titled "Please fill fields below. Fields labeled with a ** are compulsory." It contains several input fields:

- Location:** 61,003; 27,232
- *Email:** user2@email.com
- *Administrator:**
- *Birthday:** (A red error icon is present next to this field)
- More information:** (A text area with placeholder text: "You may enter up to 0 characters.")

At the bottom of the form are three buttons: "Submit" (with a green arrow), "Reset" (with a green arrow), and "Back" (with a green arrow).

ACCOUNT INFORMATION (Left sidebar):

- Username: user2
- Tenant: MAMK
- First name: User
- Surname: Userovich
- Email address: user@userland.eu

 Below this is a cartoon illustration of a woman with purple hair and a blue shirt.

WEB FORMS (Left sidebar):

- TO BE FILLED** (indicated by a green circle with a dot)
- 1. Registration - 51** (indicated by a green checkmark)

Figure 30. Registration web form ready to be filled in as seen by user2 from tenant1.

On Figure 30 one may see another iteration of the same web form. This time it is a form that is owned by another tenant (let us call it “tenant2”), it is not only viewed by users of tenant2. Any web form can be created based on an instance made by another tenant, providing that appropriate privileges are given to the tenant. In this web form new fields “Gender”, “Country” and “Address” have been appended. Further, fields “Administrator”, “Birthday” and “Email” (with edited style options) are inherited from the “Registration” form created by users in tenant1. One may notice a small red icon on Figure 30, which indicates a validation error raised by the fact that a required field “Birthday” had not been filled in before a “Submit” button was pressed.

The screenshot shows a web-based form titled "FILL WEB FORM — ID: 52". The form includes fields for Email (mandatory), Address, Country, Gender, and Birthday. The "Address" field contains "Planet Express New New York 4003 Planet Earth". The "Country" field has "Finland" selected. The "Gender" field has "Male" selected. The "Birthday" field has a calendar icon. On the left sidebar, under "ACCOUNT INFORMATION", it lists: Username: user3, Tenant: MHG Systems, First name: Philip J., Surname: Fry, Email address: deliveryguy@pizzany2000.us. Below this is a cartoon illustration of Fry. Under "WEB FORMS", there is a list: "TO BE FILLED" (with a green checkmark) and "1. dsdsdsdcccc - 31" and "2. Register to Tenant 2 - 52". At the top right are links for "Logout" and "Help".

Figure 31. Registration web form ready to be filled in as seen by user3 from tenant2.

Once the web form is populated with data a user can submit it. Before information is processed and passed away from the web form module to other frameworks, data is validated. Validation is performed using Java-style validators. If errors are found an icon is shown next to web fields, which generated errors. Additionally, if required for filling in fields have not been entered and a web form is submitted with them being empty, the same error icon is rendered, but with a different error message. If a web form that one tries to view or fill in does not exist, an error message is shown indicating that a logged in user have no permissions for working with the form of a given ID.

6.1.5 Editing web forms

To edit an existing web form a link to that web form should be clicked. Alternatively, if a user was granted permissions to do so, a web form may be accessed through **editWebForm.jsf** page. Editing web forms is similar to creating new ones. All fields may be changed. Lists of web forms that are visible to a particular user are present at the home page. On the front page one may see a list of web forms that can be viewed or edited. A sidebar on the left side of views also has a shorten list of the most recently created web forms that should be given higher priority for processing. By clicking on “Edit form” button in a list of web forms, a user

is sent to a page where management of the web forms is possible, see Figure 32. On this view a logged in person can change such aspects of the form:

- Web form's name.
- User privileges for working with the form.
- Whether or not the form requires its users to fill in CAPTCHA image. Triggered by checking “Captcha required” checkFbox.
- Possibility of the form to provide inheritance for other web forms. Triggered by checking “Can be mother” checkbox.
- All web fields that are present in the form and their properties.

The screenshot shows the 'Multitenant webforms' application. On the left, there's a sidebar with 'ACCOUNT INFORMATION' containing details like Username: user1, Tenant: MAMK, First name: Pavlo, Surname: Bazilinskyy, Email address: pavlo.bazilinskyy@gmail.com, and a profile picture of a man with glasses. Below this is a 'WEB FORMS' section with two tabs: 'TO BE FILLED' (containing 1. Registration - 1, 2. Service Survey - 3, 3. Webform test - 4, 4. NEW USER - 26, 5. Company assignment - 27) and 'CONFIRMED' (containing 1. Registration - 1, 2. Service Survey - 3, 3. Webform test - 4, 4. NEW USER - 26, 5. Company assignment - 27). The main area is titled 'EDIT EXISTING WEB FORM — ID: 1'. It has a form with fields for 'Name of the form' (set to 'Registration'), 'User rights for the form' (a list box showing '1: Pavlo Bazilinskyy', '4: Optimus Prime', '5: Harry Fischer', '6: Someone Fishy'), and checkboxes for 'Captcha required' (checked) and 'Can be mother' (checked). At the bottom are 'Submit', 'Reset', and 'Back' buttons.

Figure 32. View for editing web forms.

If management of users that have rights for working with the form is required a list labeled with “User rights for the form” can be used. In this list one can see a list of all users of the tenant. By checking names of users privileges are granted, by unchecking them rights are revoked. One may find a method that generates this list in Appendix 1.4.

A web form can also be requested to be deleted. When a user checks a checkbox next to “Delete Form” label a warning message appears. The message is generated using a JavaScript command attached to “onclick” property of HTML tag rendering a checkbox:

```
onclick="return confirm('#{ui.doYouWantToDeleteForm}'),".
```

This message is used to guarantee that no forms are deleted without intention by checking a “Delete Form” box.

The biggest part of the view responsible for editing web forms occupies a list of web fields that are assigned to the web form in question. These web fields are fully customisable with such elements ready to change:

- Label’s text that is assigned to the field.
- Type of the field.
- Whether or not the field is required to be filled in by users.
- Position in the form.

Moreover, further changes to web fields can be made by following links that are attached to web fields in a list. Different approaches can be utilised to outline elements of web fields that can be edited. In the test case the most important aspects of web fields such as type, label value, etc. are represented in “Edit Web Form” view. Less important and less frequently changed values such as label font colour, user privileges for the field, etc. are rendered in separate popup windows. These windows are opened by clicking on links that call a JavaScript function described in a header of **editWebForm.jsf**:

```
function doPopup(source) {
    popup = window.open(source, "popup", "height=460,width=540"),
    popup.focus(),
}
```

However, it is worth mentioning, that other ways are possible for achieving same results. With utilising such dynamic UI technologies as Ajax less cluttered and more usable web pages can be created. Nevertheless, with pure JSF 2.0 separating logically-connected elements in different views is a reasonable approach.

One of the popup windows that can be accessed for web fields is used for management of styling properties of web fields. This window can be accessed by following “Style” links on

the page responsible for editing web forms, see Figure 33. On that page `rendered` structure is also utilised, where appropriate properties of different types of web fields can be changed. An alternative approach is possible for achieving the same result: `<c:if>...</c:if>`. Using this tag gives similar output.

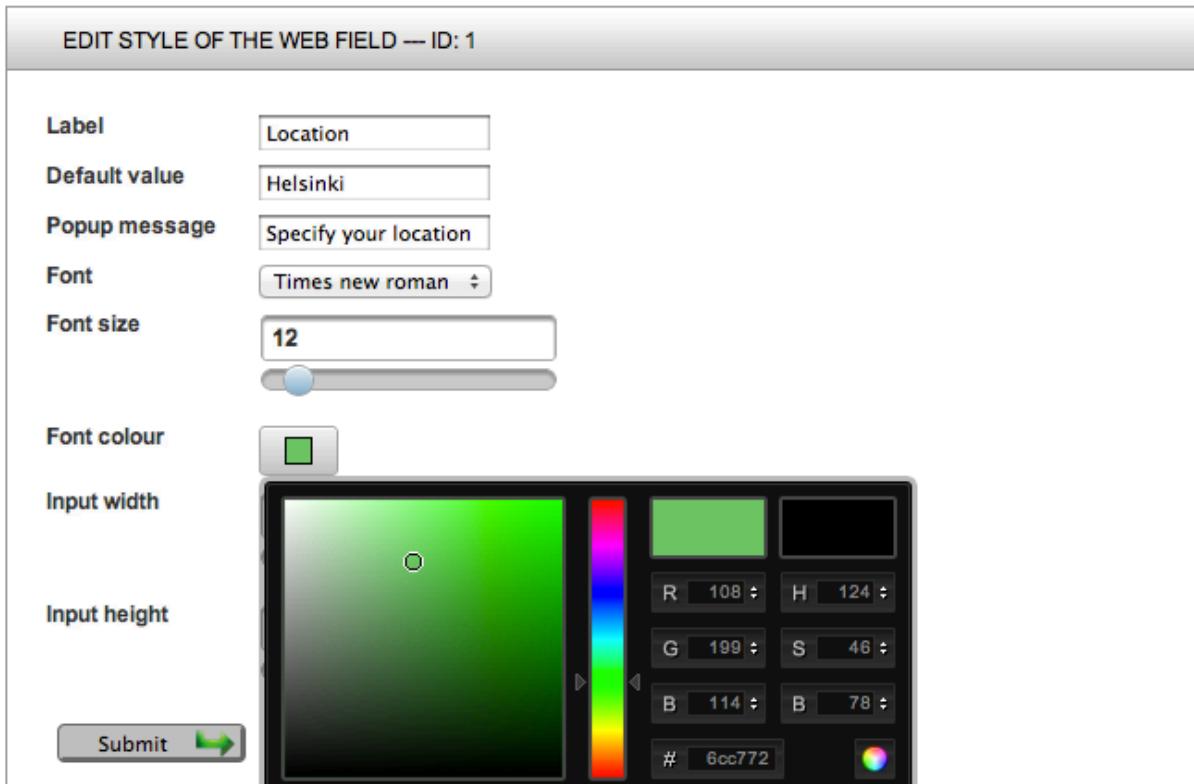


Figure 33. View for editing web field style.

Such properties of appearance of web fields as label text, font size and family, font colour etc. are editable in the test case program. Integer input sliders and a colour picker utilise elements of PrimeFaces library. A list of fonts is hard-coded and it can be populated with additional fonts. Another option for achieving similar user experience is creating a separate view for managements of tenant-specific values for the list of fonts.

The second link for additional adjustments of web field's properties is called "Rights". When a user clicks on the link an XHTML passed to the browser program, which shows a list of users of the tenant that is operating the web field. If user's rights for the web field in question are revoked, whenever a web form that contains that web field is rendered for the user, he/she does not see that field and no information is required to be filled in for the web field. The way users are granted rights for viewing web fields is similar to a list that manages rights of users

for web forms in **editWebForm.jsf** view. When a button “Submit” is pressed a method that edits user privileges is invoked. One may find a listing of this method in Appendix 1.7.

To edit values that are rendered in web fields that require multiple options, one may click a link “List”. This action opens a view where users can manage list options (see Figure 34). In this view a value that is given to the field by selecting an option in question and text to be rendered in the list that is assigned to the option can be adjusted. Additionally, a checkbox next to each new label is rendered: be setting it to TRUE, an option in the field is selected to be a default value when the field is outputted. It should be noted that if a user tries to manage list values for a field that does not require multiple options, nothing is shown on the view for management of list options.

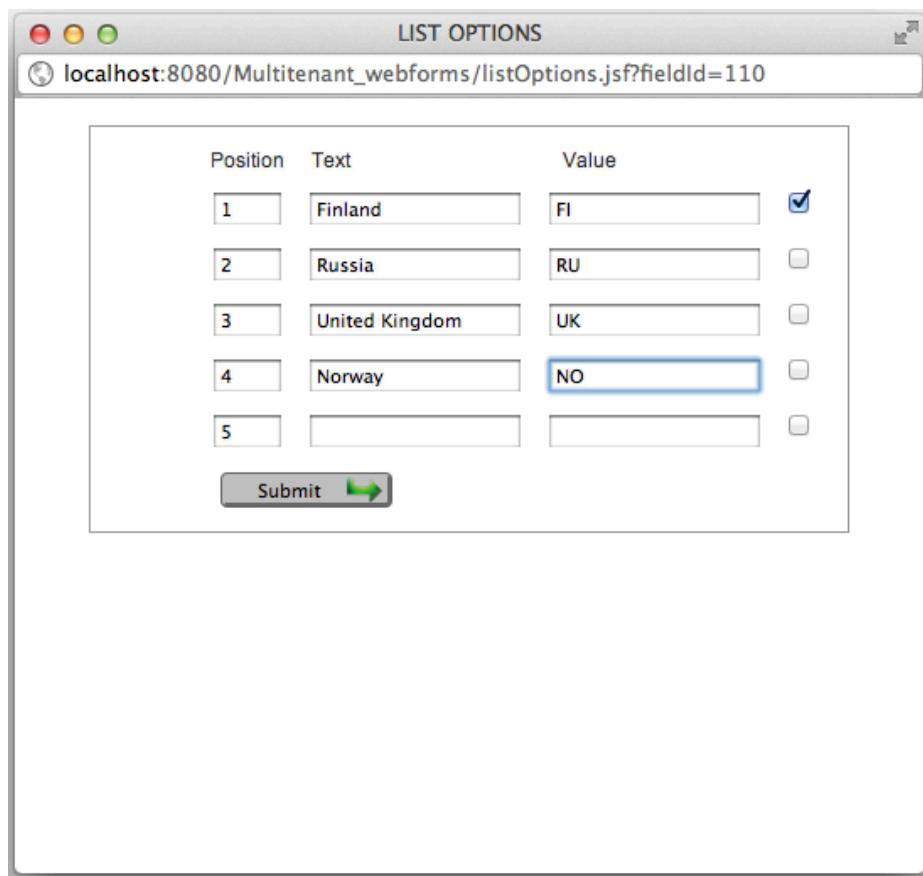


Figure 34. View for changing list options for fields that require selecting values from multiple elements.

After list options are edited for the web field, they can be rendered got end users. If the web field is a list of options (type is equal to “6”), values for options are gathered from **list_value** table. Moreover, same rules are applied for web fields of types “select one radio group” (type

property is equal to “7”) and “select one checkbox group” (type property is equal to “8”). The following method is responsible for fetching options for the list and adding them to the field:

```
/**
 * Get values for a list (type = 6)
 *
 * @return the value of listOptions
 */
public List getListOptions() {
    if (listOptions == null) {
        listOptions = new ArrayList(),
        UITextHandler uiTextHandler = new UITextHandler(),

        ListValueLogic listValueLogic = new ListValueLogic(),
        ListValue listValue = new ListValue(),
        listValue.setWebfieldId(id),
        List<ListValue> list = listValueLogic.list(listValue, null),

        for (ListValue valueList : list) {
            listOptions.add(new SelectItem(valueList.value,
uiTextHandler.getText(valueList.text))),
        }
    }
    return listOptions,
}
```

In this list a variable `ListOptions` of type `List` is returned. Members of this list are fetched using `list` method of `ListValueLogic`. This method can be extended further by dynamically locating values, which are common for a particular localisation used in the tenant’s instance of the program. In the aforementioned version of this method values are picked using `UITextHandler uiTextHandler`.

The last link that is assigned to web fields in a view where web forms are managed is called “Preset”. It is responsible for selecting a preset field. By following that link a user can select a previously created field, such as a list of countries or a radio button group for selecting gender. These fields are created by administrators in a separate application module.

Additionally, web fields that have a red letter “M” rendered next to them are “mother” fields. It indicates that changes made to those fields will also be applied to “child” web fields. Further, web fields that are inherited from other “mother” fields have such properties as

“type” and “required” locked. Mother-child relationships between web fields is described in greater details in 6.1.3 Creating new web forms.

New web fields can be added to the web form. The process of adding new web fields is also described in section 6.1.3 Creating new web forms. There is one difference to adding new web fields to a new web form, however. Instead of an array of 40 new web fields, an array of 20 is generated to improve performance. This value can also be adjusted.

Once a “Submit” button is clicked a method editWebForm is invoked. A listing of this method can be found in Appendix 1.1.

6.1.6 Managing account information

To manage user’s account information a link “Account” in the front page can be clicked. Also, users can go directly to **account.jsf** page (see Figure 35). At that page all information previously entered may be edited and new pieces of information may be added. A model **Account.java** was designed in a way that allows further extension that allows appending new columns for adding more information. A test case program described in this study has a limited set of parameters ready to be filled in by users for their account settings.

Multitenant webforms

ACCOUNT INFORMATION

Username: user1
Tenant: MAMK
First name: Pavlo
Surname: Bazilinskyy
Email address: pavlo.bazilinskyy@gmail.com

MANAGING ACCOUNT

Please fill fields below. Fields labeled with a '*' are compulsory.

Username	user1
Email address	pavlo.bazilinskyy@gmail.com
Tenant	MAMK
*First name	Pavlo
*Surname	Bazilinskyy
Old Password	
New Password	
New Password Again	

WEB FORMS

TO BE FILLED

1. Webform 1 - 1
2. Webform 2 - 2
3. Webform 3 - 3
4. Webform 4 - 4
5. Webform 5 - 5

CONFIRMED

1. Webform 1 - 1
2. Webform 2 - 2
3. Webform 3 - 3
4. Webform 4 - 4
5. Webform 5 - 5

Figure 35. Account management page.

If a password needs to be changed a previous one must be entered and a new password must be inputted twice for success. Once a new set of account settings was recorded and inserted into the database, a user is redirected to a page that shows a message indicating successful changes to the program, see Figure 36.

6.1.7 Changing tenant-specific configuration

Configuration of a tenant can be changed following “Configuration” link at the top of the page or by directly accessing **configuration.jsf** page. Configuration serves a purpose of changing different aspects of the program to meet specific needs of tenants. For example, limitations on numbers of shown objects in lists can be altered in this section. A list of settings that can be configured in the test case application can be seen below:

- *Number of web forms in sidebar* - web forms that are shown to end users in the sidebar on the left.
- *Number of web forms per page* - amount of web forms shown per page.

- *Instance URL* - an URL that users can access a tenant-specific version of the program. It can be used by external modules.
- *API Key* - a key that is used by the tenant to access modules of the test-case application.

These settings are individual for every tenant that uses the test case program. It must be noted that if changes to the tenant's configurations of the test case application are made, users of the tenant must log out and log back in to see changes. Moreover, this are only an example of what settings can be attached to a multitenant application like the test-case program.

Extensions to a list of configurable components of the program are possible.

Once a new set of configuration was recorded and inserted into the database, a user is redirected to a page that shows a message indicating successful changes to the application, see Figure 36.

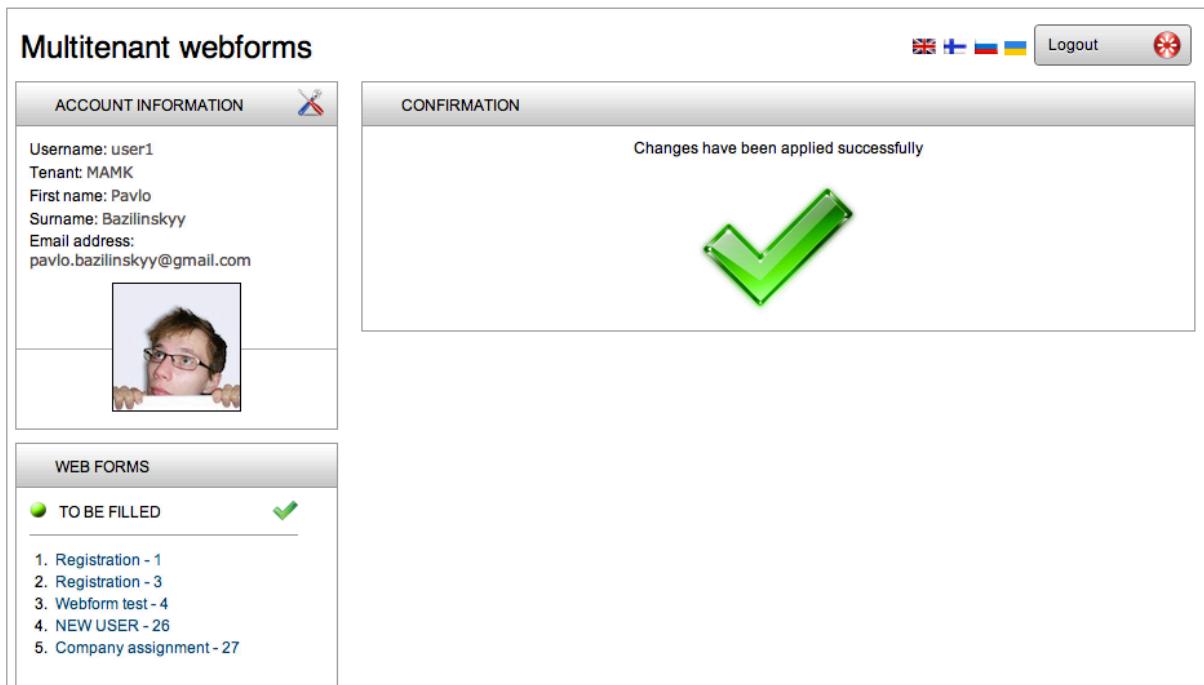


Figure 36. A view with confirmation message.

6.2 Comments and Javadoc

For comments and descriptions of code in the test case application, Javadoc framework is used. Turning to Kramer (1999), one may find that "doc comments" format used by Javadoc is the de facto industry guideline for documenting classes and methods written with Java. An example of a function with a Javadoc description from the DAO object can be seen below:

```

/**
 * Get a list of accounts from database according to searchAccount.
 * When user is not known.
 *
 * @param searchAccount
 * @param tenant
 * @param options
 * @param listOptions
 * @return
 * @throws DataAccessException
 */
public List<Account> list(Account searchAccount, Tenant tenant, Map options,
ListOptions listOptions) throws DataAccessException {
    //Getting list of users
    this.searchOptions = new ArrayList<SearchOption>(),
    if (searchAccount.getVerified() != -1)
        searchOptions.add(new SearchOption("verified", new
Integer(searchAccount.getVerified()), SearchOption.EQUAL)),
    String sql = "SELECT * FROM account WHERE tenant_id=:tenant_id "
        + SQLFactory.generateSQL(searchOptions, true)
        + " ORDER BY user_id ASC",
    MapSqlParameterSource parameters = new MapSqlParameterSource(),
    parameters.addValue("tenant_id", tenant.getId()),
    return jdbcTemplate.query(sql, parameters, new AccountRowMapper()),
}

```

All methods and all classes in the test case have respective Javadoc entries, which were entered manually and maintained by Netbeans IDE. Pre-defined tags are used for describing programming logic, specifically:

1. **@author** [author name] - identifies author(s) of a class or interface.
2. **@version** [version] - version info of a class or interface.
3. **@param** [argument name] [argument description] - describes an argument of method or constructor.
4. **@return** [description of return] - describes data returned by method (unnecessary for constructors and void methods).
5. **@ throws** [exception thrown] [exception description] - describes exception thrown by method.

Moreover, Java-style inline comments are put into code, where it helps keep track of logic in the program. As an addition, remarks on further improvements of the test-case project are placed into comments in code with a keyword TODO from Netbeans IDE.

6.3 Description of application code from the test case

As stated earlier, the test case application is developed using Netbeans Integrated Development Environment. In this program JSF projects have well-established groups of files. XHTML views of the program are stored in **Web Pages** folder, all Java code along with resource files are located in **Source Packages**, test packages are stored in **Test Packages**, libraries in **Libraries** folder, test libraries in **Test Libraries** folder and project configuration files in **Configuration Files** location (refer to Figure 37 with the test case program exposed as a project in Netbeans IDE). This section is organised in a manner that corresponds to the way files are organised in the Netbeans project.

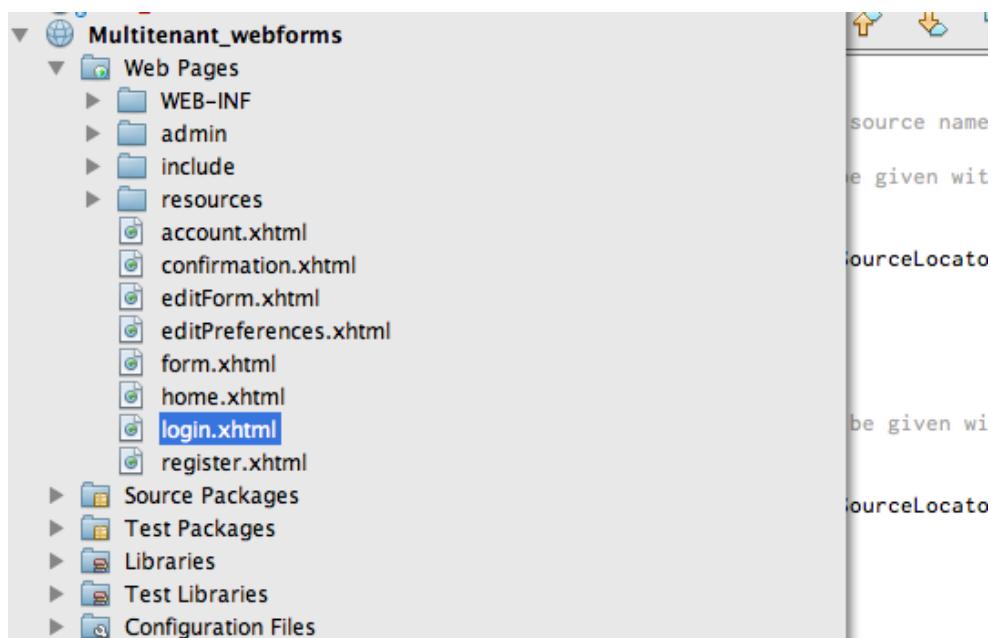


Figure 37. Project organisation in Netbeans IDE.

Netbeans IDE Java EE projects store settings in XML file format. Files with settings are stored in a folder WEB-INF. The one may find the main portion of project-specific preferences in **faces-config.xml** file. This XML document is crucial for proper functionality of JSF 2.0-powered applications. The file consists of XML tags, which describe such properties of a Java EE application as managed beans, resource bundles, localisation files, etc.

As an example, one may see below a description of a managed bean responsible for a registration page taken from the **faces-config.xml** file:

```
<managed-bean>
    <managed-bean-name>register</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.Register</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

In this piece of code a name of the managed bean, a class responsible for it and a scope of the bean are given. Similar to a listing described above other elements of Java EE programs are given attributes, which are processed on the stage of compiling. For more details, one may look at the listing of **faces-config.xml** used in the test case project in Appendix 3.

Classes in the test case are designed having further extension in mind. Namely, some of methods that are described in classes have been defined but not implemented. Instead, they raise an exception, as can be seen from the example below:

```
/**
 * Activation of tenants. May be used later
 *
 * @param object
 * @param user
 * @return
 * @throws DataAccessException
 */
public int activate(Tenant object, User user) throws DataAccessException {
    throw new UnsupportedOperationException("Not supported yet."),
}
```

Once, a compiler gets to a line that raises the exception, a program execution continues (providing no critical errors are returned). And, the exception may be seen in log files and appropriate changes to the code may be made.

6.3.1 Model classes

A model is an essential part of any MVC application. Model serves a role of a “foundation” that describes “stones” that the solution consists of. Practically any action described in the Logic layer in a SaaS application deals with a model/models. In JSF 2.0 framework

persistences are used. By adding such tags as `@Table` and `@Column` programmers can put bookmarks for an application server to link variables that are fetched from the database to specific types of data. A simple implementation of a model that corresponds to a **webform** entity in the database is described next:

```

@Table(name = "webform")
public class WebForm implements Serializable {
    protected UITextHandler uiTextHandler,
    public WebForm() {
        this.uiTextHandler = new UITextHandler(),
    }

@Id
@Column(name = "webform_id")
private int id,
public int getId() {
    return id,
}
public void setId(int id) {
    this.id = id,
}

@Column(name = "name")
private String name,
public String getName() {
    return name,
}
public void setName(String name) {
    this.name = name,
}
}

```

In the example above each column in a **webform** entity (described in section 6.6 The database scheme used) is linked to a Java variable. By using `@Table(name = "webform")` one indicates what table (or entity) in the database the model needs to be connected to. `@Id` corresponds to a primary key that is used in the database. Then, `@Column(name = "webform_id")` shows columns in the table that need to be processed.

It is a really simple model that is easy to comprehend and understand. In production-ready applications, however, a single model can often occupy thousands of lines of code and maintenance of them can be a time-consuming process. It is important to keep values given to

the tags described higher up to date because IDEs currently have no means of checking them and once the database is altered all values must be corrected by hard-coding.

In Appendix 1.6 one may find examples of fetching objects of models for rendering. Two methods in the Appendix together are good examples of one of the most important paradigms in Object-Oriented Programming (OOP) - abstraction. Additionally, it outlines inheritance and usage of interfaces in Java.

Additionally, all models that are present in the test case have corresponding DAO classes. Furthermore, all other CRUD functions are described in Logic classes.

6.3.2 Logic classes

Classes from this group contribute to the Controller layer of the application. Classes that are stored in the Logic category serve as “bridges” between View Java classes and DAO objects. Often a method from a Logic file that is called from the View class invokes a method in a DAO class that has an identical name and a list of parameters. Usage of Logic layer is important in cloud-based solutions because it creates an extra layer of security between users and sensitive information accessed by Data Access Objects. For example, here is a listing of a `list` method in **AccountLogic.java**:

```
public List<Account> list(Account object, Tenant tenant) {
    try {

        return accountDao.list(object, tenant,null,null),
    } catch (DataAccessException daex){
        return null,
    } catch (Exception ex) {
        Logger.getInstance().log(ex),
        return null,
    }
}
```

Where `accountDao` is described as a variable of the same class: `private AccountDao accountDao = new AccountDao();`

This method calls `public List<Account> list(Account searchAccount, User user, Map options, ListOptions listOptions)`, which is described in section 6.3.3 Database Access Object (DAO) classes. It passes an object of type **Account** and tenant of type **Tenant** and returns a list of accounts associated with tenant. Additionally, all other CRUD functions are described in Logic classes.

6.3.3 Database Access Object (DAO) classes

All Database Access Object (DAO) classes inherit an interface **GenericDao<T>**, where **T** is a model that is connected to a database entity that the DAO class works with. In this interface a number of methods are described that must be overridden once inheritance is established, namely:

- `public T get(int id, User user) throws DataAccessException`
- `public List<T> list(T object, User user, Map options, ListOptions listOptions) throws DataAccessException`
- `public int insert(T object, User user) throws DataAccessException`
- `public int update(T object, User user) throws DataAccessException`
- `public int activate(T object, User user) throws DataAccessException`
- `public int deactivate(T object, User user) throws DataAccessException`
- `public List findByNameQuery(Object object, int namedQuery) throws DataAccessException`
- `public int delete(T object, User user) throws DataAccessException`

These methods are self-explanatory based on their names and return value types. Here **DataAccessException** is an exception that is raised when errors with working with the database rise. For example, when wrong settings are set in the **DataSourceLocator.java** class. Naturally, other methods can be described and existing ones may be overloaded in classes that inherit **GenericDao<T>** interface. Also, in DAO object in the test case, sensitive information such as user passwords is converted into MD5 hashes before it is sent to the database.

As an example, let us consider an implementation of `public T get(int id, User user) throws DataAccessException` method in **WebFieldDao.java** class that deals with **WebField.java** model described in section 6.3.1 Model classes. From this example one may see that a signature of a method corresponds to a method that is described in the interface. In

Java in order for a class to successfully inherit an interface all methods must be overridden and fully implemented:

```
public WebField get(int id, int tenantId, User user) throws DataAccessException {
    String sql = "SELECT * FROM webfield WHERE webfield_id=:webfield_id AND
tenant_id=:tenant_id",
    MapSqlParameterSource parameters = new MapSqlParameterSource(),
    parameters.addValue("webfield_id", id),
    parameters.addValue("tenant_id", tenantId),

    return jdbcTemplate.queryForObject(sql, parameters, new
WebFieldRowMapper()),
}
```

As another example of implementation of DAO classes in the test case, one may consider two implementations of a method `public list` in **AccountDao.java** model described in Appendix 1.6.

6.3.4 Database rowmapper classes

Database **RowMapper** classes inherit from **RowMapper.java** that is a part of Spring library that is used in the project. This interface has only one method that needs to be overloaded:

`public T mapRow(ResultSet rs, int i) throws SQLException.` This method helps to translate a table in the database and turn it into values that can be linked to a model. A challenge with writing classes of type **RowMapper** is linking SQL types such as INT(10) and VARCHAR(100) into standard Java types of data.

An example may be considered. Let us take an implementation of `public Tenant mapRow(ResultSet rs, int i) throws SQLException` in **TenantRowMapper** class into consideration. It deals with **Tenant.java** model described in section 6.3.1 Model classes:

```
public Tenant mapRow(ResultSet rs, int i) throws SQLException {
    Tenant tenant = new Tenant(),

    //Main parameters
    tenant.setId(rs.getInt("tenant_id")),
    tenant.setName(rs.getString("name")),

    //Settings
    tenant.setNumberWebFormsPerPage(rs.getInt("number_webforms_per_page")),
```

```

tenant.setNumberUsersPerPage(rs.getInt("number_users_per_page")),
tenant.setNumberWebFormsVerified(rs.getInt("number_webforms_verified")),
tenant.setNumberWebFormsUnverified(rs.getInt("number_webforms_unverified")),
tenant.setInstanceUrl(rs.getString("instance_url")),
tenant.setApiKey(rs.getString("api_key")),
tenant.setSecurityKey(rs.getString("security_key")),

return tenant,
}

```

This method demonstrates how columns in a MySQL table are linked with variables of a Java class. `ResultSet rs` parameter is an object returned by a call to a method in `jdbcTemplate` in a DAO class. `mapRow` receives an object of type `NamedParameterJdbcTemplate` and returns an object of a model that it is linked to.

6.3.5 Views, CSS styling and UI classes

Program logic that is described in this section can be used for references for better understanding of processes that are described in 6.1 Description of views and user actions. This group of files is, perhaps, the most distributed and essential part of the program that is outlined on the pages of this study. In order for views to work and look how intended, .XHTML view file, .Java class and a .CSS style sheet must cooperate and use the same namespace and rules. Each XHTL view must have a Java class linked to it, CSS styling is not compulsory. Moreover, CSS rules might be described inside of XHTML files, with no creation of separate CSS style sheets.

First, a view is described using XHTML language of markup. On this stage User Interface designers can be separated from logic programmers since no application logic is stored on View level. View pages are described using standard HTML/XHTML tags. An example of a **Login** view described using XHTML can be seen in Appendix 1.8.

Properties are used in backing bean classes for views. Property is a variable that has a getter and a setter methods (in the case of a class described in Appendix 1.8: `public String getPassword()` and `public void setPassword(String password)`). Getters and setters allow greater level of control over operations performed on the variable. Once a user clicks on submit button a value put into `h:inputSecret` is sent to the **Login.java** class and a variable gets `String password` initialised.

Additionally, styling rules are applied to the view. In the mentioned example styles are linked to elements of XHTML code by describing objects by their tags. For instance, a submit button is described using the following CSS rule that is bid to all elements that have `id = "buttonSubmit"`:

```
.buttonSubmit
{
    display:inline-block,
    -webkit-box-shadow:0 0 0 1px rgba(0,0,0,.6),
    -webkit-border-radius:3px,
    -moz-box-shadow:0 0 0 1px rgba(0,0,0,.6),
    -moz-border-radius:3px,
    box-shadow:0 0 0 1px rgba(0,0,0,.6),
    border-radius:3px,
    background-image:url('img/buttonEdit.png'),
    background-repeat:no-repeat,
    background-position:center right,
    padding-left:10px,
    padding-right:30px,
    margin-right:3px,
    margin-left:10px,
    text-align:center,
    line-height:10px,
    position:relative,
    min-width:100px,
    cursor:pointer,
}
```

It results in the submit button looking differently compared to a corresponding element in all web browsers (see Figure 26). It should be noted that, unfortunately, CSS rules are translated in separate ways by web browsers. Receiving the same output from the same CSS file applied to the same file is a matter of complex styling rules, utilisation of frameworks and style-reset files, this study does not focus on this problem. In this thesis styling of web pages is done to be compatible with Google Chrome (version 18.0) browser running on Operating System Mac OS X Lion.

“Remember me” option is present in the view to enable saving of session parameters such as a username and a password. Then, when a submit button is clicked by a user, `loginAction()` method is invoked, which can be seen in `action="#{login.loginAction}"` action description in **login.xhtml**. The method is outlined in Appendix 1.9.

Further, in such views as **fillWebForm.xhtml** and **register.xhtml** validation beans are used. In Java such classes extend a class **Validator.java**. In these files a value is received as input and a decision is made if it passes tests for validation. Each validation bean must implement a method called **validate**. This method can be implemented with logical checks (as can be seen in a code listing below) or with validation using regular expressions. As an example, a **validate** methods that validates an inputted data against checks that determine if the date can be accepted as a birth date:

```
public void validate(FacesContext context, UIComponent toValidate, Object value)
throws ValidatorException {
    Date myDate = (Date)value,
    Date today = new Date(),
    if (myDate.after(today)) {
        ((UIInput) toValidate).setValid(false),
        FacesMessage message = new FacesMessage(ERROR_MESSAGE.replaceAll("\\{0\\}",
        this.uiTextHandler.getText("dateIsFuture"))),
        context.addMessage(toValidate.getClientId(context), message),
    } else {
        ((UIInput) toValidate).setValid(true),
    }
}
```

This validation bean is used when web fields that are used for inputting birth dates are added to web forms. Further, in Appendix 1.5 one may find a listing of a validation bean that validates inputted email addresses.

6.3.6 Techniques for localisation and session control utilised

All session-specific variables are stored in a Java class **SessionBean.java**. An object of this class is created each time a user opens the test case application. A default constructor gets called for each user, which makes creation of user- and session-specific variables possible. A listing of the constructor is presented below:

```
public SessionBean() {
    this.messageHandler = new MessageHandler(),
    locales = new HashMap<String, Locale>(2),
    locales.put("english", new Locale("en", "UK")),
    locales.put("finnish", new Locale("fi", "FI")),
```

```

locales.put("russian", new Locale("ru", "RU")),
locales.put("ukrainian", new Locale("uk", "UA")),
setLanguage("en"),

setSearchWebForm(new WebForm()),
setSearchAccount(new Account()),
getSearchAccount().setVerified(0),
}

```

In this listing of code one can see an initialisation of localities used in the projects. Such languages as English, Finnish, Russian and Ukrainian are put into a pool of localisation options. Localisation is presented as a group of flag icons in the upper part of each view (see Figure 26). Localisation files are made using Java resource files. There are multiple possible techniques that can be used to achieve storing values for localisation strings, e.g. JSON files. However, Java resource files have an advantage of having high support from Netbeans IDE. In files of Java resource type values are separated by “=” and a list of values has the following look:

```

main=main
contact=contact
login=login
logout=Logout
register=Register
fillFieldsBelow=Please fill fields below. Fields labeled with a '*' are compulsory.
username=Username
checkAvailability=Check Availability

```

Additionally, after localisation has been loaded and configured, objects used as a default web form and a default account that a logged in user later gets assigned to are created in the constructor: `setSearchWebForm(new WebForm())` and `setSearchAccount(new Account())`.

Further, an important part of any Software as a Service solution is error message handling. A technique used in the test case not only works with logical errors that might rise during program execution, but also it is capable of handling messages that serve as notifications to end-users of the program. A separate file called **message.xhtml** serves as a web page that can be included into any view in the application. The file utilises a Java EE tag `ui:composition` and the main part of it is represented in this way:

```

<ui:composition>
    <ice:panelGroup id="alertPanelGroup"
visible="#{sessionBean.messageHandler.visible}"
styleClass="#{sessionBean.messageHandler.message.messageType}">
        <h:outputText id="messageInfo"
value="#{sessionBean.messageHandler.message.messageText}" />
    </ice:panelGroup>
</ui:composition>

```

A message is passed to the file handling error messages and notifications. A variable `sessionBean.messageHandler.message.messageText` is fetched from **SessionBean.java** bean. Later, it is passed to the language handler, where an appropriate value in a language that is used as a localisation preference is used. Additionally, styling rules are applied from **style.css** CSS file.

6.4 Debugging in SaaS

“Program testing can be used to show the presence of bugs, but never show their absence!”

Edsger W. Dijkstra

As stated by Fox & Patterson (2012a), debugging of Software as a Service applications can be difficult. Such methods of showing malfunctioning in programs as outputting error messages into the Terminal console are difficult to implement in cloud-based solutions. Because actions in a SaaS application usually path a long list of steps before output is rendered (example from Ruby on Rails framework: URI - route - controller - model - view - render) a specific piece of logic that generates the error is problematic to determine. For example, a wrongly rendered view might be caused by a line of code that is stored in a controller class.

When a project is in the development stage, any action (namely, printing error messages to the terminal, logging using external libraries, interactive debugging etc.) can be used. However, once a project is transferred into production, only logging should be utilised.

6.5 The database and the application server

The database used in the test case application is using replication, which is briefly described in Chapter 2. Usage of sharding is an option as well, but the test case described in this

research is a relatively small application highlighting benefits of using multitenant architecture along with customisable web forms. Hence, replication can be utilised with better results. Additionally, Tocker (2009) found that turning to sharding is not profitable in small cases and if performance could be enhanced by optimisation of the codebase.

Furthermore, ensuring secure tenant access to shared data is important in projects dealing with cloud computing. A robust SaaS application requires secure data access to ensure each user sees only data that belongs to their tenant.

GlassFish Server 3.1 application server from Oracle is utilised for the test case described in this thesis. For connecting it to the database a JDBC Connection Pool and a JDBC Resource are set up. The JDBC resource is connected to the pool of type **javax.sql.DataSource** and a classname set to **com.mysql.jdbc.jdbc2.optional.MysqlDataSource** (which is a class used for connecting to MySQL database that is used for storing data). Otherwise, all settings that are used in the project are default values given by Oracle Corporation.

When the application is deployed to the server, a class **DataSourceLocator.java** from a package **com.mhgsystems.db** (which is a part of MHG Systems framework) locates an object of type **DataSource** that is required for establishing connection to the database. A function that performs this search is listed in Appendix 1.10. After an object of type **DataSource** is found, GlassFish server can locate the database that is stored in MySQL database server.

6.6 The database scheme used

MySQL Workbench object notation and Crow's Foot (IE) relationship notation are used to describe Entity Relationship Diagrams. Providing M:N (Many-to-Many) relationships is achieved by decomposing them into two 1:M relationships.

As a base for designing a scheme for the database a model developed by Veli-Matti Plosila for MHG Systems is used (see Figure 38). In this relatively simple model web forms can be overwritten for populating them with work order-related data such as deadlines, latitude and longitude of working sites, etc.

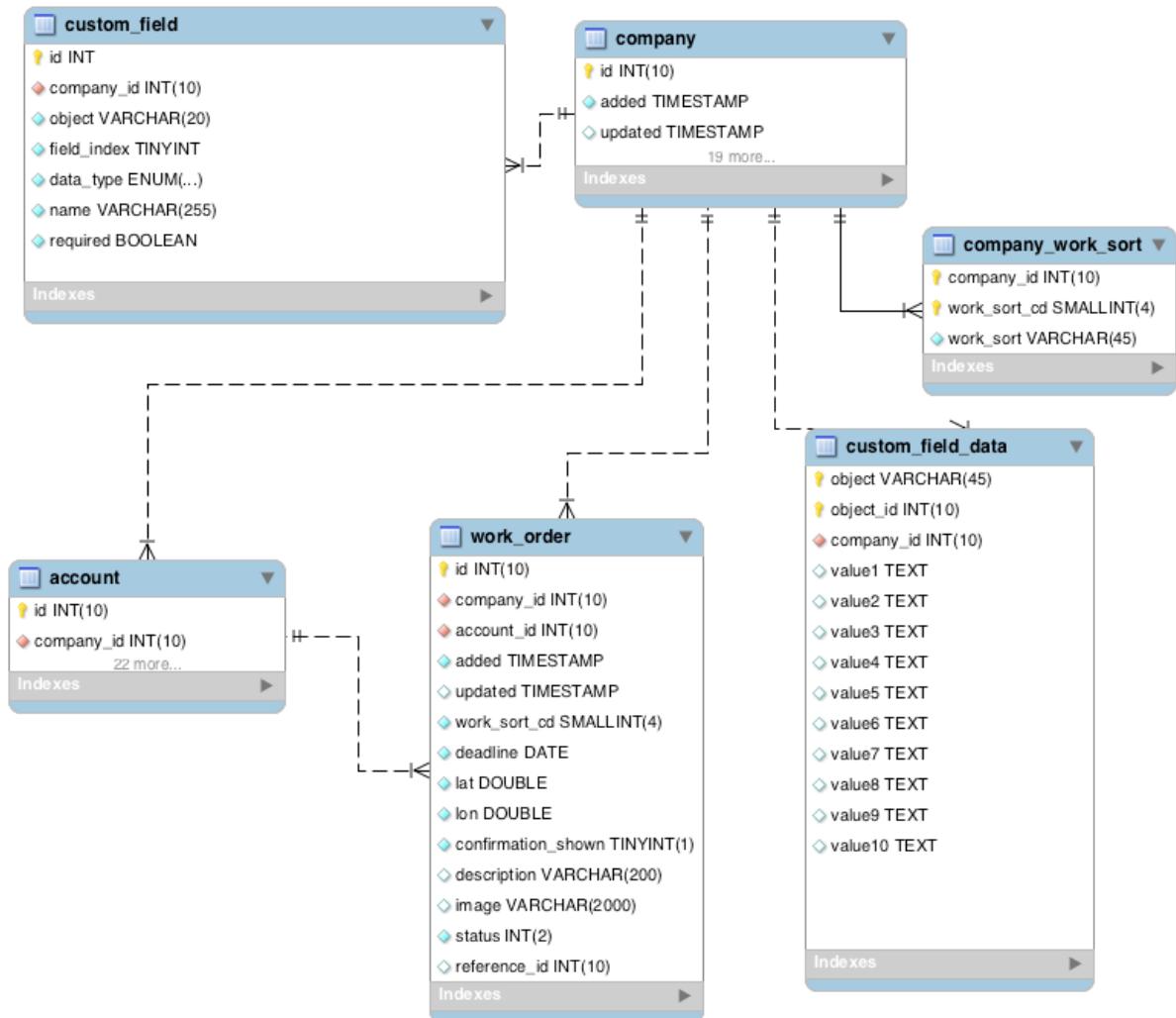


Figure 38. EER diagram of the database utilised in MHG Systems.

After research on possible implementation of a scheme for the test case a more complex model was designed. The model described on Figure 39 is used in the test case application. As can be seen from the Figure web forms are represented as a separated entity, unlike in the previous scheme described. Web forms consist of objects of type **webfield**. Additionally, support for several not-trivial featured is outlined:

1. Mother-child relationship for fields where users have ability to use a field that has already been used in other web forms. It copies data over and in case of changing of mother field child web fields also receive changes. This functionality is described with **mother_child_webfield** table.
2. Users can be granted or not granted privileges of viewing/filling web forms. More precisely, users within one company may have different lists of accessible web forms. This functionality is described with **user_webform**.

3. Users can be granted or not granted privileges of seeing particular fields in a form. As an example, a certain user1 sees and is required to fill three fields in a certain form and user2 sees five fields. This functionality is described with **user_webfield**.
4. A separate table **label** is present to add reuse of labels and support for localisation.
5. By separating properties connected with information stored in list-boxes, a table **list_value** is created that adds an ability to have list-boxes with unlimited numbers of values to pick from (e.g. more than 200 countries in predefined list).

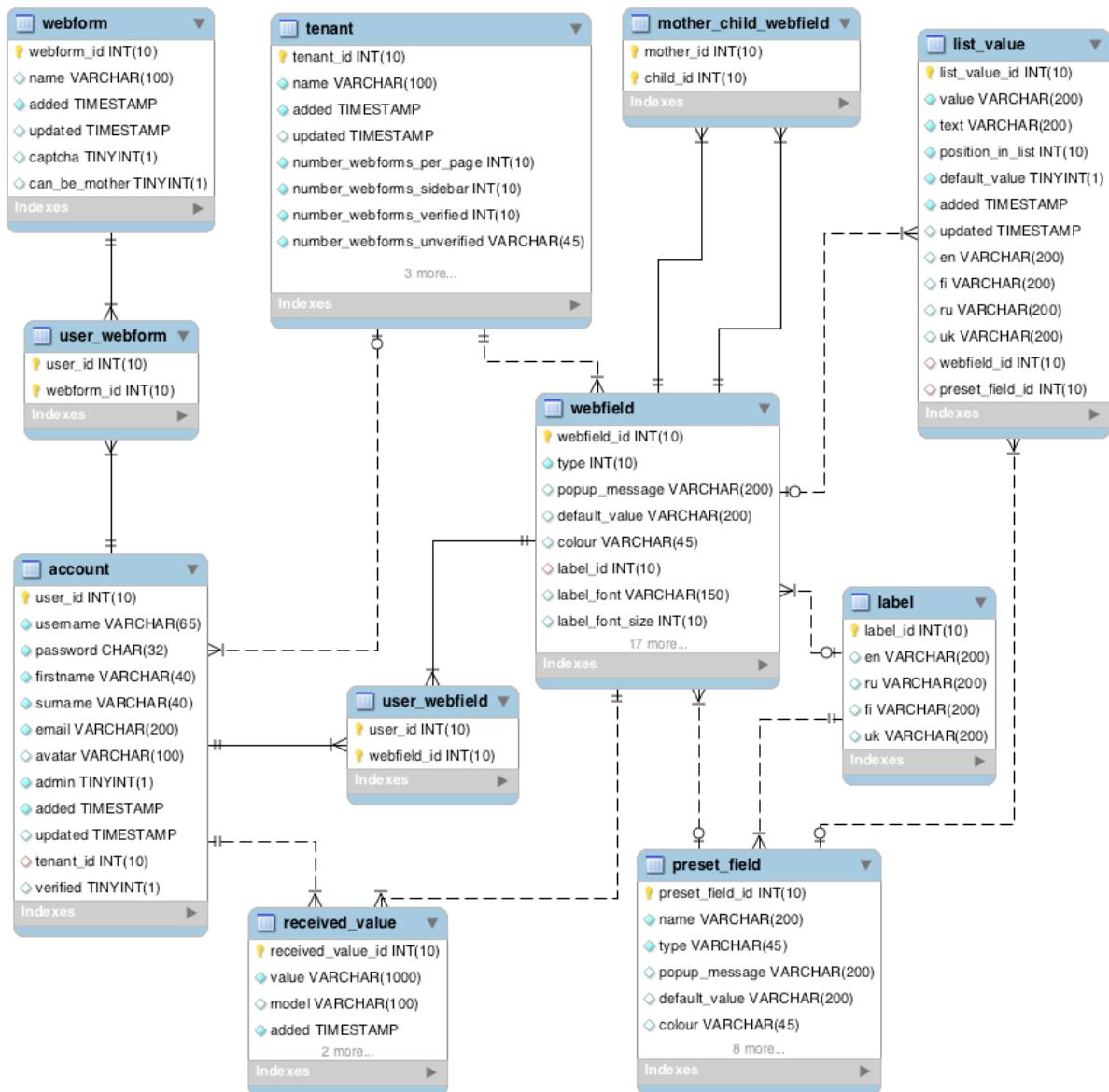


Figure 39. EER diagram of the database for the test case.

For synchronising and storing backups of the database MySQL Workbench MWB models are used. These files allow forward and backward engineering of databases. As can be seen from

Figure 40 during synchronisation process both updates of the database and of models are possible.

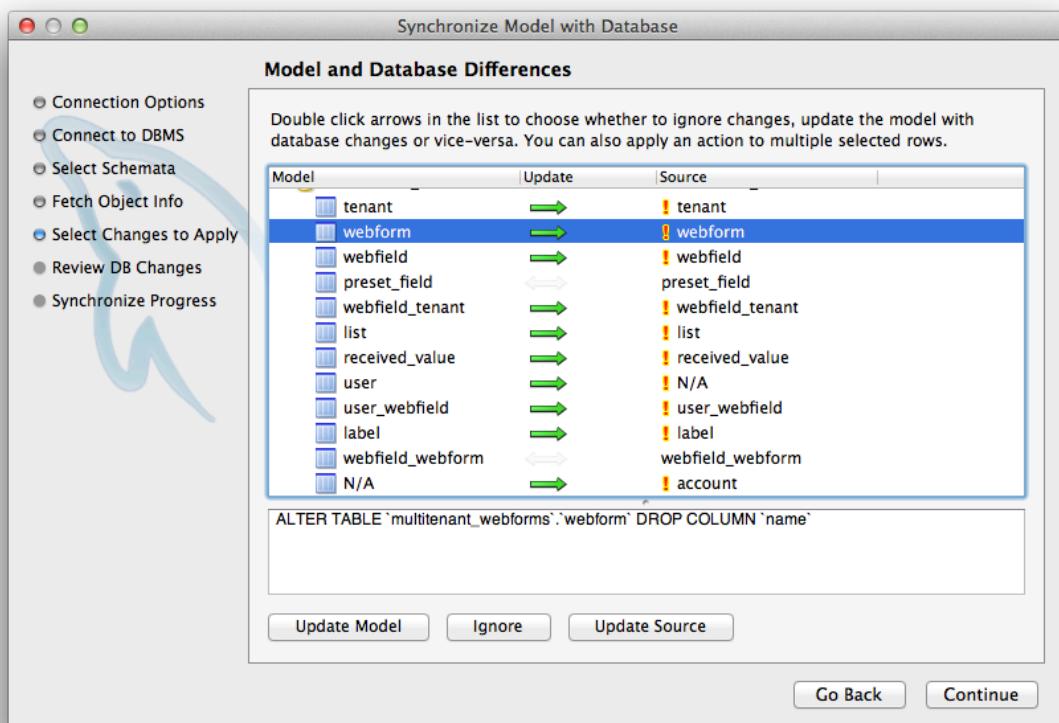


Figure 40. Synchronisation of the database with a MySQL Workbench MWB model.

One may find a detailed list of entities in Appendix 4. The aforementioned design of the database for a project concentrating on multitenant web forms is not the only variation of the scheme. One may wish to extend the design. For example, separate entities for each type of the web field (radio button, checkbox, list etc.) may be described and connected to the main entity **webfield**. This approach will reduce dimensions of **webfield** table and it may improve performance. However, the design described in this study is not a production-ready model, it is a research case.

It is assumed that properties added and updated of type TIMESTAMP may be used on later stages of advancement in the project for statistical purposes.

One may find a listing of SQL commands that are required for reproducing the database that is utilised for the test case application in Appendix 2.

7 CONCLUSION

Scalable multitenancy is an important asset for development of cloud-based Software as a Service solutions. Nevertheless, building multitenant solutions requires facing a few technical challenges. Achieving a comprehensible level of multitenantness in pieces of software applies complicated demands that are, sometimes, hard to meet by software engineers and difficult to develop into finished products. Furthermore, if the system is well-balanced and stable achieved results are likely to exceed all expectations. Additionally, a larger amount of financial saves can be achieved.

Software as a Service proved to be the best platform for development of multitenant web form enabled applications currently available on the market. SaaS is capable of providing relatively inexpensive computing solutions and acceptable flexibility on software use, which is important for modern businesses and enterprises. Software as a Service is practically in all phases of lives of people nowadays. Even for writing of this work multiple SaaS solutions were used: cloud reference management system, cloud writing and citation tools, cloud code compilers etc. Our world is becoming more and more dependant on cloud-based software solutions with every day, yet little research has been done on various aspects of cloud computing. This is the main reason why such study is an important asset to the world of science.

Results and findings from this research attempt to claim that creation of multitenant web forms by using MySQL and JSF 2.0 is, indeed, possible. Moreover, the simple test case application that is described on pages of this research can be used as a base for development of a commercial application. The test case application tries to benefit to the research question about development of optimised and versatile multitenant web forms using JSF and MySQL.

The main focus of the test case application is creation of a module that can be attached to ERP systems. Such resource management systems as MHG Bioenergy are very important for a successful company. In his work Blokdijk (2008) claims that without a well-designed and programmed ERP system, a company would “suffocate” with a number of software applications that do not synchronise with each other. Such process may lead to a failure in

effective interface. This research makes an attempt to contribute into the world of SaaS and its “Multitenant ERP systems” branch especially.

On pages of this study an idea that creation of multitenant web forms is an achievable task can be found. Moreover, an author of this research thinks that such element can be an important asset to most modern cloud-based software solutions. Meeting requirements of all clients of a company can require a great deal of endurance and tight deadlines from company's employees but it is hard to imagine that in a hundred years from now people will need to compensate on their needs while filling out web-based forms.

8 BIBLIOGRAPHY

Abramson, David, Buyya, Rajkumar & Giddy, Jonathan 2002. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. Future Gener.Comput.Syst., vol. 18, no. 8. 1061 - 1074.

Aggarwal, Nitin 2011. The Rise of Cloud Computing. WWW-document. <http://w3c-compliant.com/2011/03/the-rise-of-cloud-computing>. Updated 12.12.2010. Referred 9.2.2012.

Amazon 2008. Public Data Sets on AWS2008. WWW-document. <http://aws.amazon.com>. Updated 10.2.2012. Referred 28.2.2012.

Anderson, Chris 2006. The Long Tail: Why the Future of Business Is Selling Less of More. New York: Hyperion.

Andrew, Rachel 2007. The CSS anthology. Melbourne: Sitepoint.

Armbrust, Michael, Fox, Armando, Griffith, Rean, Joseph, Anthony D., Katz, Randy H., Konwinski, Andrew, Lee, Gunho, Patterson, David A., Rabkin, Ariel, Stoica, Ion, Zaharia & Matei 2009. Above the Clouds: A Berkeley View of Cloud Computing. Berkley: University of California, EECS Department.

BBC News 2010. Finland makes broadband a 'legal right'. WWW-document. <http://www.bbc.co.uk/news/10461048>. Updated 1.7.2010. Referred 15.3.2012.

Beck, Kent , Grenning, James, Martin, Robert C., Beedle, Mike & Highsmith, Jim 2001. Manifesto for Agile Software Development. WWW-document. <http://agilemanifesto.org>. Updated 1.1.2001. Referred 27.2.2012.

Bezemer, Cor-Paul & Zaidman, Andy 2010. Multi-tenant SaaS applications: maintenance dream or nightmare?. Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE). New York: ACM. 88.

Blokdijk, Gerard 2008. SaaS 100 Success Secrets - How companies successfully buy, manage, host and deliver software as a service. London: Emereo Pty.

Bowman, Doug A., Kruijff, Ernst, LaViola, Joseph J. & Poupyrev, Ivan 2004. 3D User Interfaces: Theory and Practice. Redwood City: Addison Wesley Longman Publishing Co.

Chong, Frederick & Garraro, Gianpaolo 2006. Architecture strategies for catching the long tail, Microsoft white paper. WWW-document. <http://msdn.microsoft.com/en-us/library/aa479069.aspx>. Updated 1.4.2006. Referred 14.3.2012.

Columbus, Louis 2011. Predicting Cloud Computing Adoption Rates . WWW-document. <http://softwarestrategiesblog.com/2011/07/24/predicting-cloud-computing-adoption-rates/3/5>. Updated 25.7.2011. Referred 14.3.2012.

Daniel, Florian, Yu, Jin, Benatallah, Boualem, Casati, Fabio, Matera, Maristella & Saint-paul, Regis 2006. Understanding UI integration: A survey of problems, technologies, and opportunities. IEEE Internet Computing May 2006.

de Heide, Marcel 2007. Country Review Finland. Technopolis: United Nations University.

Engelsen, Nathaniel 2011. Multi Tenant Architecture via Dependency Injection: Part 1. WWW-document. <http://blog.tallan.com/2010/07/11/multi-tenant-architecture-via-dependency-injection-part-1>. Updated 11.7.2011. Referred 8.2.2012.

Fielding, Roy Thomas 2000. Architectural styles and the design of network-based software architectures. Irvine: University of California.

Finfacts 2011. Google chooses cool Finland for latest data centre, Facebook said to be considering Swedish location. WWW-document. http://www.finfacts.ie/irishfinancenews/article_1023105.shtml. Updated 9.11.2011. Referred 15.3.2012.

Foremski, Tom 2005. A tribute to one of Silicon Valley's most influential and forgotten researchers at Xerox Parc event. WWW-document. http://www.siliconvalleywatcher.com/mt/archives/2005/06/a_tribute_to_on.php. Updated 9.6.2005. Referred 9.3.2012.

Fox, Armando & Patterson, David A. 2012a. Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing. Los Angeles: Strawberry Canyon LLC.

Fox, Armando & Patterson, David A. 2012b. Video lectures for Software as a Service online class. WWW-document. <https://www.coursera.org/saas/lecture/index>. Updated 20.2.2012. Referred 21.2.2012. Video.

Goldszmidt, German & Poddar, Indrajit 2008. Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware. WWW-document. <http://www.ibm.com/developerworks/webservices/library/ws-middleware/index.html>. Updated 24.4.2008. Referred 6.3.2012.

Google 2012a. World Development Indicators and Global Development Finance - Google Public Data Explorer. WWW-document. <http://www.google.com/publicdata/explore?>

ds=d5bncppjof8f9_&met_y=sp_pop_totl&idim=country:UKR&dl=en&hl=en&q=population+ukraine. Referred 15.3.2012.

Google 2012b. World Development Indicators and Global Development Finance - Google Public Data Explorer. WWW-document. http://www.google.com/publicdata/explore?ds=d5bncppjof8f9_&met_y=sp_pop_totl&idim=country:FIN&dl=en&hl=en&q=population+finland. Referred 15.3.2012.

Google 2012c. World Development Indicators and Global Development Finance - Google Public Data Explorer. WWW-document. http://www.google.com/publicdata/explore?ds=d5bncppjof8f9_&met_y=sp_pop_totl&idim=country:GBR&dl=en&hl=en&q=population+uk. Referred 15.3.2012.

Grossman, Robert L. 2009. The Case for Cloud Computing. *IT Professional*, vol. 11, no. 2. 23 - 27.

Hadlock, Kris 2011. Create customizable web interfaces with the jQuery UI and Ajax. WWW-document. <http://www.ibm.com/developerworks/web/library/wa-jqueryui/index.html?ca=drs->. Updated 8.3.2011. Referred 8.2.2012.

Halabieh, Abdul 2003, Customizable User Interfaces, 702/181 edn, G06F 19/00, USA.

Hamdaqa, Mohammed, Livogiannis, Tassos & Tahvildari, Laden 2011. A Reference Model for Developing Cloud Applications. CLOSER 2011. SciTePress. 98.

Hardy, Quenty 2012. The Week the Cloud Won. WWW-document. <http://bits.blogs.nytimes.com/2012/02/24/the-week-the-cloud-won>. Updated 24.2.2012. Referred 25.2.2012.

Hatch, Ralph 2008. SaaS Architecture, Adoption and Monetization of SaaS Projects using Best Practice Service Strategy, Service Design, Service Transition, Service Operation and Continual Service Improvement Processes. London: Emereo Pty Ltd.

Helsingin Sanomat 2010. Itella to begin opening letters and delivering them via email. WWW-document. <http://www.hs.fi/english/article/Itella+to+begin+opening+letters+and+delivering+them+via+email/1135255790584>. Updated 30.3.2010. Referred 15.3.2012.

Hingley, Martin 2011. UK Cloud Computing Forecast - Recession-Busting Growth. WWW-document. <http://itcandor.net/2011/09/21/cc-uk-q311>. Updated 21.9.2011. Referred 15.3.2012.

Hoff, Todd 2009. High Scalability - An Unorthodox Approach to Database Design : The Coming of the Shard. WWW-document. <http://highscalability.com/unorthodox-approach-database-design-coming-shard>. Updated 6.8.2009. Referred 5.3.2012.

Hunt, Andrew & Thomas, David 1999. *The pragmatic programmer: from journeyman to master*. Boston: Addison-Wesley Longman Publishing.

IBM 2010. Three-tier architectures. WWW-document. http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=%2Fcom.ibm.websphere.base.doc%2Finfo%2Faes%2Fae%2Fcovr_3-tier.html. Updated 20.9.2010. Referred 21.2.2012.

ICPC 2011. ICPC - Results World Finals 2011. WWW-document. <http://cm.baylor.edu/ICPCWiki/Wiki.jsp?page=Results%20World%20Finals%202011>. Updated 7.11.2011. Referred 15.3.2012.

International Monetary Fund 2011. United Kingdom. WWW-document. <http://www.imf.org/external/pubs/ft/weo/2011/02>. Updated 1.4.2011. Referred 15.3.2012.

Jansen, Slinger, Houben, Geert-Jan & Brinkkemper, Sjaak 2010. Customization realization in multi-tenant web applications: case studies from the library sector. Proceedings of the 10th international conference on Web engineering. Berlin: Springer-Verlag. 445.

JavaServer Faces 2011. JSF in a nutshell. WWW-document. <http://www.javaserverfaces.org>. Updated 23.6.2011. Referred 13.3.2012.

Keegan, Patrick, Champenois, Ludovic, Crawley, Gregory, Hunt, Charlie & Webster, Christopher 2006. NetBeans(TM) IDE Field Guide: Developing Desktop, Web, Enterprise, and Mobile Applications, 2nd edn, Upper Saddle River: Prentice Hall PTR.

Keene, Chris, Poddar, Indrajit, Nicke, Joe & Budnik, Uri 2012. Cloud Quick Start: A Roadmap For Adopting Cloud Computing. WWW-document. <http://www.wavemaker.com/ibm-quickstart.pdf>. Referred 5.3.2012.

Khan, Qasim 2010. JSF 2.0: Introduction and Overview. WWW-document. <http://developerarticles.com/jsf-2-0-introduction-and-overview>. Updated 30.5.2010. Referred 13.3.2012.

Kisker, Holger, Matzke, Pascal, Ried, Stefan, Lisserman, Miroslaw 2011. Forrsights: The Software Market In Transformation, 2011 And Beyond. Cambridge: Forrester.

Knorr, Eric & Gruman, Galen 2008. What cloud computing really means. WWW-document. <http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031>. Updated 9.9.2008. Referred 9.2.2012.

Kramer, Douglas 1999. API documentation from source code comments: a case study of Javadoc. Proceedings of the 17th annual international conference on Computer documentation. New York: ACM. 147.

Larman, Craig 2003. Agile and Iterative Development: A Manager's Guide. Glasgow: Pearson Education.

Leon, Alexis 2008. Enterprise Resource Planning. New Delhi: Tata McGraw-Hill Education.

Mell, Peter & Grance, Timothy 2011. The NIST Definition of Cloud Computing. Gaithersburg: National Institute of Standards and Technology.

MHG Systems 2012. MHG Systems Oy Ltd in a nutshell. WWW-document. <http://www.mhgsystems.com/en/company>. Updated 17.12.2010. Referred 7.3.2012.

Microsoft 2012. Deployment Patterns. WWW-document. <http://msdn.microsoft.com/en-us/library/ms998478.aspx>. Referred 24.2.2012.

Miller, Michael 2008. Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online. New York: Que Publishing.

Miller, Renée J., Miller, Ren'ee J., Tsatalos, Odysseas G., Williams, John H. 1997. DataWeb: Customizable Database Publishing for the Web. IEEE Multimedia, vol. 4. 14-21.

Miller, Robert C. 2003. End User Programming for Web Users. IST PROGRAMME. Proceedings Workshop on End-User Development held in conjunction with the ACM CHI 2003 Conference. 61.

Myers, Brad 1994. Challenges of HCI design and implementation. Interactions, vol. 1, no. 1. 73-83.

Myers, Brad 2003. Graphical User Interface Programming. Pittsburgh: Carnegie Mellon University.

Myers, Brad & Rosson, Mary B. 1992. Survey on User Interface Programming. Pittsburgh: Carnegie Mellon University.

Nationmaster 2012. Europe area statistics - countries compared. WWW-document. http://www.nationmaster.com/graph/geo_eur_are-geography-europe-area. Updated 6.1.2012. Referred 15.3.2012.

Poelker, Chris 2011. The three layers of cloud computing. WWW-document. http://blogs.computerworld.com/18338/the_three_layers_of_cloud_computing. Updated 24.5.2012. Referred 15.3.2012.

Potter, Richard 1993. Watch what I do: Programming by Demonstration. Cambridge: MIT Press. 361 - 380.

Quan, Dennis, Huynh, David, Karger, David R., Miller, Robert 2003. User interface continuations. Proceedings of the 16th annual ACM symposium on User interface software and technology. New York: ACM. 145.

Rangan, Kash, Cooke, Alan, Post, Justin & Schindler, Nat 2008. The Cloud Wars : 100 + billion at stake. The Analyst. May 2008. 1 - 90.

Rao, Leena 2012. Salesforce Beats, Q4 Revenue Up 38 Percent to \$632 Million, Raises Guidance. WWW-document. <http://techcrunch.com/2012/02/23/salesforce-beats-q4-revenue-up-38-percent-to-632-million-raises-guidance>. Updated 23.2.2012. Referred 24.2.2012.

Raymond, Eric, Steven & Landley, Rob 2004. The Art of Unix Usability. WWW-document. <http://catb.org/~esr/writings/taoou/html>. Referred 24.2.2012.

Roy, Rahul 2008. Shard – A Database Design. WWW-document. <http://technoroy.blogspot.com/2008/07/shard-database-design.html>03/05. Updated 28.7.2008. Referred 24.2.2012.

Salesforce.com 2002. Salesforce.com and salesforce.com/foundation sponsor 12th annual tibet house benefit concert held at new york city's famed carne. WWW-document. <http://www.salesforcefoundation.org/node/77>. Updated 22.2.2002. Referred 24.2.2012.

Salesforce.com 2012. Customization. WWW-document. <http://www.salesforce.com/platform/customization/>. Updated 10.3.2012. Referred 15.3.2012.

Selvitelle, Britt 2010. The Tech Behind the New Twitter.com. WWW-document. <http://engineering.twitter.com/2010/09/tech-behind-new-twittercom.html>. Updated 20.9.2010. Referred 9.3.2012.

Shore, James & Warden, Shane 2007. The art of agile development. New York: O'Reilly.

Smith, Roger 2009. Computing in the cloud. Research Technology Management. September 2009. 1 - 6.

Spolsky, Joel 2001. User Interface Design for Programmers. New York: Apress.

Swartz, Jon 2007. Salesforce CEO leads charge against software. WWW-document. http://www.usatoday.com/money/companies/management/2007-07-22-benioff_N.htm. Updated 24.7.2010. Referred 24.2.2012.

Tiihonen, Seppo 2003. The history of corruption in central government. Amsterdam: IOS Press.

Tocker, Morgan 2009. Why you don't want to shard - MySQL Performance Blog. WWW-document. <http://www.mysqlperformanceblog.com/2009/08/06/why-you-dont-want-to-shard/>. Updated 6.8.2009. Referred 5.3.2012.

Tsai, Chang-Hao, Ruan, Yaoping, Sahu, Sambit, Shaikh, Anees, Shin, Kang G. 2007. Virtualization-based techniques for enabling multi-tenant management tools. Proceedings of the Distributed systems: operations and management 18th IFIP/IEEE international conference on Managing virtualization of networks and services. Berling: Springer-Verlag. 171.

W3C-XHTML 2004. Extensible HyperText Markup Language XHTML 2.0. WWW-document. <http://www.w3.org>. Referred 11.3.2012.

W3C-XHTML 1997. HyperText Markup Language (HTML4.01). WWW-document. <http://www.w3.org>. Referred 11.3.2012.

Weber, Tin 2010. Cloud computing for business goes mainstream. WWW-document. <http://www.bbc.co.uk/news/10097450>. Updated 5.5.2010. Referred 15.3.2012.

Weissman, Craig D. & Bobrowski, Steve 2009. The design of the force.com multitenant internet application development platform. Proceedings of the 35th SIGMOD international conference on Management of data. New York: ACM. 889.

Whitehorn, Mark & Marklyn, Bill 2006. Inside Relational Databases with Examples in Access. Secaucus: Springer-Verlag New York.

Wikipedia contributors 2012a. Model–view–controller. WWW-document. <http://en.wikipedia.org/wiki/Model-view-controller>. Updated 1.5.2010. Referred 24.2.2012.

Wikipedia contributors 2012b. Cloud computing. WWW-document. http://en.wikipedia.org/wiki/Cloud_computing. Updated 12.2.2010. Referred 24.2.2012.

WorldApp 2010. SaaS (Software-as-a-Service) in Ukraine. WWW-document. <http://www.slideshare.net/Rinky25/saas-softwareasaservice-in-ukraine>. Updated 21.5.2010. Referred 15.3.2012.

Zettlemoyer, Luke S. & St. Amant, Robert 1999. A visual medium for programmatic control of interactive applications. Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit. New York : ACM. 199.

APPENDICES

1: SELECTED LISTINGS OF APPLICATION CODE

The test case program has about 19200 lines of code. Because of a relatively large size of the project only selected pieces of code are given in this Appendix as examples. Next to descriptions of listings of code names of classes where they are stored are given in brackets.

1.1. Method for editing web forms (**WebFormView.java**)

```
/*
 * Function called when web form needs to be edited.
 *
 * @return String
 * @throws DataAccessException
 */
public String editWebForm() throws DataAccessException {
    WebFormLogic webFormLogic = (WebFormLogic)
        LogicFactory.getNewGenericLogic(WebForm.class),
    LabelLogic labelLogic = (LabelLogic) LogicFactory.getNewGenericLogic(Label.class),
    WebFieldLogic webFieldLogic = (WebFieldLogic)
        LogicFactory.getNewGenericLogic(WebField.class),
    UserWebFieldLogic userWebFieldLogic = (UserWebFieldLogic)
        LogicFactory.getNewGenericLogic(UserWebField.class),
    UserWebFormLogic userWebFormLogic = (UserWebFormLogic)
        LogicFactory.getNewGenericLogic(UserWebForm.class),
    MotherChildWebFieldLogic motherChildWebFieldLogic = new MotherChildWebFieldLogic(),
    LogicResponse logicResponse = null,
    boolean success = true,

    if (getSessionBean().isDeleteForm()) {
        // Delete web fields of the form that is to be deleted.
        WebField webField = new WebField(),
        webField.setWebformId(getSessionBean().getWebForm().getId()),
        List<WebField> fieldsDelete = webFieldLogic.list(webField, getUser()),
        for (WebField field : fieldsDelete) {
            // Delete user rights for web field.
            UserWebField userWebField = new UserWebField(),
            userWebField.setWebFieldId(field.getId()),
            logicResponse = userWebFieldLogic.deleteAllUsersForForm(userWebField, null),
            if (!logicResponse.isSucceeded()) {
                success = false,
                break,
            }
        }
    }
}
```

```

// Delete mother child assignments for web field.
MotherChildWebField motherChild = new MotherChildWebField(),
motherChild.setChildId(field.getId()),
logicResponse = motherChildWebFieldLogic.deleteAllForChild(motherChild, null),
if (!logicResponse.isSucceeded()) {
    success = false,
    break,
}
motherChild.setMotherId(field.getId()),
logicResponse = motherChildWebFieldLogic.deleteAllForMother(motherChild, null),
if (!logicResponse.isSucceeded()) {
    success = false,
    break,
}
// Delete web field
logicResponse = webFieldLogic.delete(field, null),
if (!logicResponse.isSucceeded()) {
    success = false,
    break,
}
}
// Delete user rights for web form.
UserWebForm userWebForm = new UserWebForm(),
userWebForm.setWebFormId(getSessionBean().getWebForm().getId()),
logicResponse = userWebFormLogic.deleteAllUsersForForm(userWebForm, null),
if (!logicResponse.isSucceeded()) {
    success = false,
}
// Delete form if delete was checked.
logicResponse = webFormLogic.delete(getSessionBean().getWebForm(), null),
if (!logicResponse.isSucceeded()) {
    success = false,
}
// If no errors return to the view. Otherwise, show error message.
if (success) {
    getSessionBean().setConfirmation("formDeleted"),
    return "confirmation",
} else {
    getSessionBean().getMessageHandler().createMessage("errorEditWebForm"),
    return null,
}
} else {
// Update form.
// Toggle captcha
if (getSessionBean().getWebForm().isCaptchaRequired()) {
    getSessionBean().getWebForm().setCaptcha(1),
} else {
    getSessionBean().getWebForm().setCaptcha(0),
}
// Toggle can be mother

```

```

if (getSessionBean().getWebForm().isCanBeMotherBool()) {
    getSessionBean().getWebForm().setCanBeMother(1),
} else {
    getSessionBean().getWebForm().setCanBeMother(0),
}
logicResponse = webFormLogic.update(getSessionBean().getWebForm(), null),
if (!logicResponse.isSucceeded()) {
    success = false,
}
// Add new web fields.
for (WebField newField : getSessionBean().newWebFieldsInForm) {
    if (!newField.getLabel().getEn().isEmpty()) {
        logicResponse = labelLogic.insert(newField.getLabel(), null),
        if (!logicResponse.isSucceeded()) {
            success = false,
            break,
        }
        int newLabelId = labelLogic.getMaxLabelId(),
        newField.setLabelId(newLabelId),
        newField.setWebformId(getSessionBean().getWebForm().getId()),
        newField.setTenantId(getSessionBean().getUser().getTenantId()),
        logicResponse = webFieldLogic.insert(newField, null),
        if (!logicResponse.isSucceeded()) {
            success = false,
            break,
        }
        int newWebFieldId = webFieldLogic.getMaxWebFieldId(),
        newField.setId(newWebFieldId),
        // Allow users of the tenant use the web field.
        logicResponse = userWebFieldLogic.insertForAllTenantUsers(newField,
getSessionBean().getTenant()),
        if (!logicResponse.isSucceeded()) {
            success = false,
        }
    }
}
// Update old fields.
for (WebField field : getSessionBean().webFieldsInFormForUser) {
    if (field.isToDelete()) {
        // Delete field if required.
        // Delete user rights for web field.
        UserWebField userWebField = new UserWebField(),
        userWebField.setWebFieldId(field.getId()),
        logicResponse = userWebFieldLogic.deleteAllUsersForForm(userWebField,
null),
        if (!logicResponse.isSucceeded()) {
            success = false,
            break,
        }
    }
}
// Delete mother child assignments for web field.

```

```

MotherChildWebField motherChild = new MotherChildWebField(),
motherChild.setChildId(field.getId()),
logicResponse = motherChildWebFieldLogic.deleteAllForChild(motherChild,
null),
    if (!logicResponse.isSucceeded()) {
        success = false,
        break,
    }
    motherChild.setMotherId(field.getId()),
    logicResponse = motherChildWebFieldLogic.deleteAllForMother(motherChild,
null),
    if (!logicResponse.isSucceeded()) {
        success = false,
        break,
    }
// Delete web field
logicResponse = webFieldLogic.delete(field, null),
if (!logicResponse.isSucceeded()) {
    success = false,
    break,
}
} else {
// Update field.
logicResponse = webFieldLogic.update(field, null),
if (!logicResponse.isSucceeded()) {
    success = false,
    break,
}
logicResponse = labelLogic.update(field.getLabel(), null),
if (!logicResponse.isSucceeded()) {
    success = false,
    break,
}
// Update child field. Possibly recursion can be used here.
// Check if field is a mother from another field.
MotherChildWebField tempMotherChildWebField = new MotherChildWebField(),
MotherChildWebField motherChildWebField = new MotherChildWebField(),
motherChildWebField.setMotherId(field.getId()),
tempMotherChildWebField = null, // Set to null after check fo child above.
tempMotherChildWebField =
motherChildWebFieldLogic.checkIfFieldIsMother(motherChildWebField, getUser()),
if (tempMotherChildWebField != null) { // Web field is a mother
    // Get list of all fields that are children to the field.
    List<MotherChildWebField> listChildren =
motherChildWebFieldLogic.listChildredToMother(motherChildWebField,
getSessionBean().getUser()),
    // Loop through list of fields and update values.
    for (MotherChildWebField child : listChildren) {

```

```

        WebField childField = webFieldLogic.get(child.getChildId(),
getSessionBean().getUser()),
        // Set new values
        childField.setInputWidth(field.getInputWidth()),
        childField.setInputHeight(field.getInputHeight()),
        childField.setInputSize(field.getInputSize()),
        childField.setTextareaCol(field.getTextareaCol()),
        childField.setTextareaRow(field.getTextareaRow()),
        childField.setType(field.getType()),
        // Update child field
        logicResponse = webFieldLogic.update(childField, null),
        if (!logicResponse.isSucceeded()) {
            success = false,
            break,
        }
    }
}
}

// Update user rights for the form
UserWebForm userWebForm = new UserWebForm(),
userWebForm.setWebFormId(getSessionBean().getWebForm().getId()),
// Delete all user rights for the form.
logicResponse = userWebFormLogic.deleteAllUsersForForm(userWebForm,
getSessionBean().getUser()),
if (!logicResponse.isSucceeded()) {
    success = false,
}
// Add updated rights.
userWebForm.setWebFormId(getSessionBean().getWebForm().getId()),
for (String userRight : userRightsSelected) {
    userWebForm.setUserId(Integer.parseInt(userRight)),
    logicResponse = userWebFormLogic.insert(userWebForm,
getSessionBean().getAccount()),
    if (!logicResponse.isSucceeded()) {
        success = false,
        break,
    }
}
// If no errors return to the view. Otherwise, show error message.
if (success) {
    getSessionBean().setConfirmation("formEdited"),
    return "confirmation",
} else {
    getSessionBean().getMessageHandler().createMessage("errorEditWebForm"),
    return null,
}
}
}
}

```

1.2. Method for creating new “child” web forms (WebFormView.java)

```

/**
 * Create new Web Form based on another web form.
 *
 * @return String
 * @throws DataAccessException
 */
public String newChildWebForm() throws DataAccessException {
    // Logic class declarations.
    WebFormLogic webFormLogic = (WebFormLogic)
        LogicFactory.getNewGenericLogic(WebForm.class),
        LabelLogic labelLogic = (LabelLogic)
        LogicFactory.getNewGenericLogic(Label.class),
        WebFieldLogic webFieldLogic = (WebFieldLogic)
        LogicFactory.getNewGenericLogic(WebField.class),
        UserWebFormLogic userWebFormLogic = (UserWebFormLogic)
        LogicFactory.getNewGenericLogic(UserWebForm.class),
        UserWebFieldLogic userWebFieldLogic = (UserWebFieldLogic)
        LogicFactory.getNewGenericLogic(UserWebField.class),
        MotherChildWebFieldLogic motherChildWebFieldLogic = (MotherChildWebFieldLogic)
        LogicFactory.getNewGenericLogic(MotherChildWebField.class),
        boolean success = true, // Trigger of errors.

    // Insert new web form.
    LogicResponse logicResponse =
        webFormLogic.insert(getSessionBean().getWebForm(), null),
        if (!logicResponse.isSucceeded()) {
            success = false,
        }
        int newWebFormId = webFormLogic.getMaxWebFormId(),
        getSessionBean().getWebForm().setId(newWebFormId),

    // Allow users of the tenant use the web form.
    logicResponse =
        userWebFormLogic.insertForAllTenantUsers(getSessionBean().getWebForm(),
        getSessionBean().getTenant()),
        if (!logicResponse.isSucceeded()) {
            success = false,
        }
        // Add web fields from the mother web form.
        for (WebField newField : getSessionBean().webFieldsInFormMotherForm) {
            if (!newField.getLabel().getEn().isEmpty()) {
                logicResponse = labelLogic.insert(newField.getLabel(), null),
                if (!logicResponse.isSucceeded()) {
                    success = false,
                    break,
                }
                int newLabelId = labelLogic.getMaxLabelId(),

```

```

newField.setLabelId(newLabelId),
newField.setWebformId(newWebFormId),
newField.setTenantId(getSessionBean().getUser().getTenantId()),
logicResponse = webFieldLogic.insert(newField, null),
if (!logicResponse.isSucceeded()) {
    success = false,
    break,
}
int newWebFieldId = webFieldLogic.getMaxWebFieldId(),
newField.setIdInMotherForm(newField.getId()),
newField.setId(newWebFieldId),
// Allow users of the tenant use the web field.
logicResponse = userWebFieldLogic.insertForAllTenantUsers(newField,
getSessionBean().getTenant()),
if (!logicResponse.isSucceeded()) {
    success = false,
}
// Determine if a web filed is generated based on another field (child
set to TRUE).
if (newField.isChild()) {
    MotherChildWebField motherChildWebField = new
MotherChildWebField(),
    motherChildWebField.setMotherId(newField.getIdInMotherForm()),
    motherChildWebField.setChildId(newField.getId()),

    // Insert mother-child pair. No logic response is retrieved because
breaks in logic it generates at this place. TODO fix it.
    motherChildWebFieldLogic.insert(motherChildWebField,
getSessionBean().getUser()),
}
}

}

// Add new web fields.
for (WebField newField : getSessionBean().newWebFieldsInForm) {
    if (!newField.getLabel().getEn().isEmpty()) {
        logicResponse = labelLogic.insert(newField.getLabel(), null),
        if (!logicResponse.isSucceeded()) {
            success = false,
            break,
        }
        int newLabelId = labelLogic.getMaxLabelId(),
        newField.setLabelId(newLabelId),
        newField.setWebformId(newWebFormId),
        newField.setTenantId(getSessionBean().getUser().getTenantId()),
        logicResponse = webFieldLogic.insert(newField, null),
        if (!logicResponse.isSucceeded()) {
            success = false,
            break,
        }
        int newWebFieldId = webFieldLogic.getMaxWebFieldId(),
    }
}

```

```

        newField.setId(newWebFieldId),
        // Allow users of the tenant use the web field.
        logicResponse = userWebFieldLogic.insertForAllTenantUsers(newField,
getSessionBean().getTenant()),
        if (!logicResponse.isSucceeded()) {
            success = false,
        }
    }
}

// If there were no errors, go back to edit web form view.
if (success) {

getSessionBean().getMessageHandler().createMessage("editWebFormSuccessful"),
    return "editWebForm",
} else {
    getSessionBean().getMessageHandler().createMessage("errorEditWebForm"),
    return null,
}
}
}

```

1.3. Method for parsing a web form (WebFormView.java)

```

/**
* Parse web form via f:param. e.g. <f:param name="idWork"...
*
* @return String
*/
public String parseFillWebForm() {
    FacesContext context = FacesContext.getCurrentInstance(),
    Map requestMap = context.getExternalContext().getRequestParameterMap(),
    String idWork = (String) requestMap.get("idWork"),
    int id = Integer.parseInt(idWork),
    WebFormLogic WebFormLogic = (WebFormLogic)
    LogicFactory.getNewGenericLogic(WebForm.class),
    getSessionBean().setWebForm(WebFormLogic.get(id, getUser())),
    if (getSessionBean().getWebForm().getNumberWebfields() < 1) {
        getSessionBean().getMessageHandler().createMessage("noWebFields"),
        return "fillWebForm",
    } else {
        return "fillWebForm",
    }
}

```

1.4. Method for generating a list of user rights for a form (WebFormView.java)

```

/**
 * Get the value of userRightsOptions
 *
 * @return the value of userRightsOptions
 */
public List getUserRightsOptions() {
    if (userRightsOptions == null) {
        userRightsOptions = new ArrayList(),
        AccountLogic accountLogic = (AccountLogic)
        LogicFactory.getNewGenericLogic(Account.class),
        Account tempUser = new Account(),
        tempUser.setTenantId(getSessionBean().getTenant().getId()),
        tempUser.setVerified(-1),
        // Get list of all users for a given form.
        List<Account> listUsers = accountLogic.list(tempUser,
        getSessionBean().getTenant()),
        Logger.getInstance().log("tenant " + tempUser.getTenantId()),
        Logger.getInstance().log("size " + listUsers.size()),
        for (Account user : listUsers) {
            // Filter users from other tenants.
            if (user.getTenantId() != getSessionBean().getTenant().getId()) {
                continue,
            }
            // Build name of the user: ID-FIRST-SECOND.
            String nameUser = Integer.toString(user.getId()) + ":" +
            + user.getFirstname() + " " + user.getSurname(), // Add id first second.
            userRightsOptions.add(new SelectItem(user.getId(), nameUser)), // TODO: add
            support for localisation.
        }
    }
    return userRightsOptions,
}

```

1.5. Method for validating email address (EmailValidator.java)

```

public void validate(FacesContext context,
    UIComponent toValidate,
    Object value) {
try {
    String enteredEmail = (String) value,
    //Set the email pattern string
    Pattern p = Pattern.compile(".+@.+\\\\.\\[a-z]+"),
    //Match the given string with the pattern
    Matcher m = p.matcher(enteredEmail),
    //Check whether match is not found
}

```

```

ValidationBean validationBean = new ValidationBean(),
if (!m.matches()) {
    throw new Exception("regex"),
} else if (!validationBean.checkFreeEmail(enteredEmail)) {
    //Check if email is available
    throw new Exception("taken"),
}
((UIInput) toValidate).setValid(true),
} catch (Exception ex) {
    if (ex.getMessage().equals("taken")) {
        //Email is taken
        ((UIInput) toValidate).setValid(false),
        FacesMessage message = new FacesMessage(ERROR_MESSAGE.replaceAll("\\{0\\}", this.uiTextHandler.getText("emailIsTaken"))),
        context.addMessage(toValidate.getClientId(context), message),
    } else if (ex.getMessage().equals("regex")) {
        //Regex validation failed
        ((UIInput) toValidate).setValid(false),
        FacesMessage message = new FacesMessage(ERROR_MESSAGE.replaceAll("\\{0\\}", this.uiTextHandler.getText("notValidEmail"))),
        context.addMessage(toValidate.getClientId(context), message),
    }
}
}
}

```

1.6. Methods for fetching lists of users (AccountDao.java)

```

/**
* Get a list of accounts from database according to searchAccount.
* When user is not known.
*
* @param searchAccount
* @param tenant
* @param options
* @param listOptions
* @throws DataAccessException
*/
public List<Account> list(Account searchAccount, Tenant tenant, Map options,
ListOptions listOptions) throws DataAccessException {
    //Getting list of users
    this.searchOptions = new ArrayList<SearchOption>(),
    if (searchAccount.getVerified() != -1)
        searchOptions.add(new SearchOption("verified", new
Integer(searchAccount.getVerified()), SearchOption.EQUAL)),
    String sql = "SELECT * FROM account WHERE tenant_id=:tenant_id "
        + SQLFactory.generateSQL(searchOptions, true)
        + " ORDER BY user_id ASC",
    MapSqlParameterSource parameters = new MapSqlParameterSource(),
    parameters.addValue("tenant_id", tenant.getId()),

```

```

    return jdbcTemplate.query(sql, parameters, new AccountRowMapper()),
}

/**
 * Get list of accounts from database according to searchAccount with limitations on
 * the size of returned list.
 *
 * @param limitStart
 * @param limitLength
 * @param searchAccount
 * @param tenant
 * @param options
 * @param listOptions
 * @throws DataAccessException
 */
public List<Account> list(int limitStart, int limitLength, Account searchAccount,
Tenant tenant, Map options, ListOptions listOptions) throws DataAccessException {
    //Getting list of users
    this.searchOptions = new ArrayList<SearchOption>(),
    if (searchAccount.getVerified() != -1)
        searchOptions.add(new SearchOption("verified", new
Integer(searchAccount.getVerified()), SearchOption.EQUAL)),
    String sql = "SELECT * FROM account WHERE tenant_id=:tenant_id "
        + SQLFactory.generateSQL(searchOptions, true)
        + " ORDER BY user_id ASC LIMIT "
        + limitStart
        + ", "
        + limitLength,
    MapSqlParameterSource parameters = new MapSqlParameterSource(),
    parameters.addValue("tenant_id", tenant.getId()),
    return jdbcTemplate.query(sql, parameters, new AccountRowMapper()),
}

```

Where, `String sql` is an SQL statements that is executed using `jdbcTemplate.query(sql, parameters, new AccountRowMapper())` method, which returns a list of objects of type `Account` that is afterwards returned from the `list` method.

1.7. Method for editing web field privileges (WebFormView.java)

```

/**
 * Edit user privileges for getSessionBean().getWebField().
 *
 * @return
 */
public String editWebFieldRights() {
    UserWebFieldLogic userWebFieldLogic = new UserWebFieldLogic(),

```

```

// Update user rights for the form
UserWebField userWebField = new UserWebField(),
userWebField.setWebFieldId(getSessionBean().getWebField().getId()),
// Delete all user rights for the form.
boolean success = true,
LogicResponse logicResponse = userWebFieldLogic.deleteAllUsersForForm(userWebField,
getSessionBean().getUser()),
if (!logicResponse.isSucceeded()) {
    success = false,
}
// Add updated rights.
userWebField.setWebFieldId(getSessionBean().getWebField().getId()),
for (String userRight : userRightsWebFieldSelected) {
    userWebField.setUserId(Integer.parseInt(userRight)),
    logicResponse = userWebFieldLogic.insert(userWebField,
getSessionBean().getAccount()),
    if (!logicResponse.isSucceeded()) {
        success = false,
        break,
    }
}
if (success) {
    return "closePopup",
} else {
    getSessionBean().getMessageHandler().createMessage("errorEditWebField"),
    return null,
}
}
}

```

1.8. Login view and its backing bean (login.xhtml, Login.java)

```

<h:body>
<div id="content">
<ui:include src="include/hmenu.xhtml"/>
<span id="contentMain" class="adminContentMain">
    <h:form onkeypress="if (event.keyCode == 13) this.submit(),">
        <div id="box" class="loginBox">
            <div id="boxHeader"><h:outputText value="#{ui.loginToTheSystem}" /></div>
        <div id="loginBoxContent">
            <h:panelGrid columns="2" styleClass="loginTable"
columnClasses="loginCol1,loginCol2">
                <f:facet name="header">
                    <ui:include src="include/message.xhtml"/>
                </f:facet>
                <h:outputLabel id="passwordLbl" value="#{ui.password}" />
                <h:inputSecret id="password" value="#{login.password}" />
                <h:outputLabel id="rememberMe" value="#{ui.rememberMe}" />

```

```

        <h:selectBooleanCheckbox title="Remember Me"
value="#{login.rememberMe}" />
        <h:outputLabel value=" " />
        <h:commandButton styleClass="buttonSubmit"
value="#{ui.loginBUTTON}" action="#{login.loginAction}" id="buttonSubmit" />
    </h:panelGrid>
    <div class="alignedCenter">
        <br />
        <br />
        <h:commandLink value="#{ui.register}"
action="#{login.registerAction}" />
    </div>
</div>
</div>
</h:form>
</span>
</div>
</h:body>

```

Here `f:facet` is a tag used by JSF framework, this framework uses `f:` and `ui:` to describe its custom elements and `h:` for describing common HTML tags. Further, PrimeFaces framework is utilised to render custom complex elements of GUI, such as sliders and colour pickers. In this example lines outline web field that receive values from a user and then pass them to a corresponding Java class in UI package. For example, `<h:inputSecret id="password" value="#{login.password}" />` describes an input field used for inputting sensitive information such as user passwords (it hides entered symbols by showing asterisks '*' instead). `#{login.password}` part of the line indicates that a value is passed to a variable `password`. This variable is described as a property in **Login.java**:

```

private String password,
/***
 * Get the value of password
 *
 * @return the value of password
 */
public String getPassword() {
    return password,
}

/***
 * Set the value of password
 *
 * @param password new value of password

```

```
 */
public void setPassword(String password) {
    this.password = password,
}
```

1.9. Method that manages users logging in (Login.java)

```
/**
 * Call AccountLogic to login to the system
 *
 * @return String view to navigate
 */
public String loginAction() {

    AccountLogic accountLogic =
(AccountLogic)LogicFactory.getNewGenericLogic(Account.class),
    Account account = accountLogic.get(username, password),

    if (account != null) {
        //Account found
        getSessionBean().setAccount(account),
        getSessionBean().setUser(accountLogic.createUser(account)),
        TenantLogic tenantLogic =
(TenantLogic)LogicFactory.getNewGenericLogic(Tenant.class),
        Tenant tenant = tenantLogic.get(account.getTenantId()),
        Logger.getInstance().log("tenant name: " + tenant.getId()),
        getSessionBean().setTenant(tenant),
        getSessionBean().setLoggedIn(true),
        // Save the userid and password in a cookie
        FacesContext facesContext = FacesContext.getCurrentInstance(),
        Cookie btuser = new Cookie("btuser", username),
        Cookie btpasswd = new Cookie("btpasswd", password),
        if (rememberMe == false) {
            rememberMe1 = "false",
        } else {
            rememberMe1 = "true",
        }
        Cookie btremember = new Cookie("btremember", rememberMe1),
        btuser.setMaxAge(3600),
        btpasswd.setMaxAge(3600),
        ((HttpServletResponse)

facesContext.getExternalContext().getResponse()).addCookie(btuser),
        ((HttpServletResponse)
facesContext.getExternalContext().getResponse()).addCookie(btpasswd),
        ((HttpServletResponse)
facesContext.getExternalContext().getResponse()).addCookie(btremember),
        return "home",
    } else {
        getSessionBean().getMessageHandler().createMessage("wrongUsername"),
    }
}
```

```

        return null,
    }
}

```

In this piece of code two possibilities are possible: either a user provided correct credentials (username, password) or they were wrong. If the first option holds true, than the username and the password are remembered as browser cookies (providing the user clicked on “Remember me” option), as described by lines `Cookie btuser = new Cookie("btuser", username)` and `Cookie btpasswd = new Cookie("btpasswd", password)`. If either the username or the password (or both) are incorrect, the user is redirected back to the login prompt, where an error message is shown. Also, an instance of **Account.java** class is used in the described example. The line `Account account = accountLogic.get(username, password)` invokes a get method in the logic class that in its turn calls a get method in the DAO class, which passes back either an object of type Account or raises an exception.

1.10. Method for locating DataSource object (DataSourceLocator.java)

```

/**
 * Get DataSource object with given name
 * @param name of the data source
 * @return DataSource object
 */
public DataSource getDataSource(String name) {
    DataSource dataSource = null,
    try {
        //Try to find DataSource from cache
        dataSource = cache.get(name),

        //DataSource not found from cache. Let's try to add it to the cache
        if (dataSource == null) {
            InitialContext initialContext = new InitialContext(),
            dataSource = (DataSource) initialContext.lookup(JNDI_PREFIX + name),
            if (dataSource == null) {
                throw new Exception("DataSource not found with name " + name),
            }

            if (!cache.containsKey(name)) {
                Logger.getInstance().log("DataSource " + name + " added to the
cache"),
                cache.put(name, dataSource),
            }
        }
        return dataSource,
    } catch (Exception ex) {

```

```
    Logger.getInstance().log(ex),  
    return null,  
}  
}
```

2: SQL STATEMENTS FOR CREATION OF THE DATABASE

```

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL';
CREATE SCHEMA IF NOT EXISTS `multitenant_webforms` DEFAULT CHARACTER SET latin1 ;
USE `multitenant_webforms` ;

-- -----
-- Table `multitenant_webforms`.`tenant`
-- -----
DROP TABLE IF EXISTS `multitenant_webforms`.`tenant` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`tenant` (
  `tenant_id` INT(10) NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR(100) NOT NULL ,
  `added` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
  `updated` TIMESTAMP NULL DEFAULT NULL ,
  `number_webforms_per_page` INT(10) NOT NULL DEFAULT '10' ,
  `number_webforms_sidebar` INT(10) NOT NULL DEFAULT '5' ,
  `number_webforms_verified` INT(10) NOT NULL DEFAULT '20' ,
  `number_webforms_unverified` VARCHAR(45) NOT NULL DEFAULT '3' ,
  `instance_url` CHAR(200) NULL DEFAULT NULL ,
  `api_key` CHAR(36) NULL DEFAULT NULL ,
  `security_key` CHAR(36) NULL DEFAULT NULL ,
  PRIMARY KEY (`tenant_id`)
)
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8;

-- -----
-- Table `multitenant_webforms`.`account`
-- -----
DROP TABLE IF EXISTS `multitenant_webforms`.`account` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`account` (
  `user_id` INT(10) NOT NULL AUTO_INCREMENT ,
  `username` VARCHAR(65) NOT NULL ,
  `password` CHAR(32) NOT NULL ,
  `firstname` VARCHAR(40) NOT NULL ,
  `surname` VARCHAR(40) NOT NULL ,
  `email` VARCHAR(200) NOT NULL ,
  `avatar` VARCHAR(100) NULL DEFAULT NULL ,
  `admin` TINYINT(1) NOT NULL DEFAULT '0' ,
  `added` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP ,
  `updated` TIMESTAMP NULL DEFAULT NULL ,

```

```

`tenant_id` INT(10) NULL DEFAULT NULL ,
`verified` TINYINT(1) NULL DEFAULT NULL ,
PRIMARY KEY (`user_id`) ,
UNIQUE INDEX `username_UNIQUE` (`username` ASC) ,
INDEX `user_tenant` (`tenant_id` ASC) ,
CONSTRAINT `user_tenant`
    FOREIGN KEY (`tenant_id`)
    REFERENCES `multitenant_webforms`.`tenant` (`tenant_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 8
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `multitenant_webforms`.`label`

DROP TABLE IF EXISTS `multitenant_webforms`.`label` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`label` (
`label_id` INT(10) NOT NULL AUTO_INCREMENT ,
`en` VARCHAR(200) NULL DEFAULT NULL ,
`ru` VARCHAR(200) NULL DEFAULT NULL ,
`fi` VARCHAR(200) NULL DEFAULT NULL ,
`uk` VARCHAR(200) NULL DEFAULT NULL ,
PRIMARY KEY (`label_id`) ,
INDEX `webfield_table11` (`ru` ASC) )
ENGINE = InnoDB
AUTO_INCREMENT = 162
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `multitenant_webforms`.`preset_field`

DROP TABLE IF EXISTS `multitenant_webforms`.`preset_field` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`preset_field` (
`preset_field_id` INT(10) NOT NULL AUTO_INCREMENT ,
`name` VARCHAR(200) NOT NULL ,
`type` VARCHAR(45) NOT NULL DEFAULT 'text_field' ,
`popup_message` VARCHAR(200) NULL DEFAULT NULL ,
`default_value` VARCHAR(200) NULL DEFAULT NULL ,
`colour` VARCHAR(45) NULL DEFAULT NULL ,
`label_font` VARCHAR(45) NULL DEFAULT NULL ,
`label_font_size` INT(10) NULL DEFAULT NULL ,
`input_width` INT(10) NULL DEFAULT NULL ,
`input_height` INT(10) NULL DEFAULT NULL ,
`input_size` INT(10) NULL DEFAULT NULL ,
`label_id` INT(10) NOT NULL ,
`textarea_col` INT(10) NULL DEFAULT NULL ,

```

```

`textarea_row` INT(10) NULL DEFAULT NULL ,
PRIMARY KEY (`preset_field_id`) ,
INDEX `preset_field_label` (`label_id` ASC) ,
CONSTRAINT `preset_field_label`
    FOREIGN KEY (`label_id` )
        REFERENCES `multitenant_webforms`.`label` (`label_id` )
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `multitenant_webforms`.`webfield`

-- -----
DROP TABLE IF EXISTS `multitenant_webforms`.`webfield` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`webfield` (
`webfield_id` INT(10) NOT NULL AUTO_INCREMENT ,
`type` INT(10) NOT NULL DEFAULT '1' ,
`popup_message` VARCHAR(200) NULL DEFAULT NULL ,
`default_value` VARCHAR(200) NULL DEFAULT NULL ,
`colour` VARCHAR(45) NULL DEFAULT NULL ,
`label_id` INT(10) NULL DEFAULT NULL ,
`label_font` VARCHAR(150) NULL DEFAULT 'Cambria,''Times New Roman'', ''Nimbus
Roman No9 L'', ''Freeserif'', Times, serif' ,
`label_font_size` INT(10) NULL DEFAULT '12' ,
`input_width` INT(10) NULL DEFAULT '20' ,
`input_height` INT(10) NULL DEFAULT NULL ,
`input_size` INT(10) NULL DEFAULT '50' ,
`required` TINYINT(1) NOT NULL ,
`added` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP ,
`updated` TIMESTAMP NULL DEFAULT NULL ,
`position_in_webform` INT(10) NOT NULL ,
`preset_field_id` INT(10) NULL DEFAULT NULL ,
`webform_id` INT(10) NOT NULL ,
`tenant_id` INT(10) NOT NULL ,
`textarea_col` INT(10) NULL DEFAULT '40' ,
`textarea_row` VARCHAR(45) NULL DEFAULT '5' ,
`default_value1` VARCHAR(200) NULL DEFAULT NULL ,
`default_value2` VARCHAR(200) NULL DEFAULT NULL ,
`default_value3` VARCHAR(200) NULL DEFAULT NULL ,
`default_value4` VARCHAR(200) NULL DEFAULT NULL ,
`default_value5` VARCHAR(200) NULL DEFAULT NULL ,
PRIMARY KEY (`webfield_id`) ,
INDEX `webfield_preset_field` (`preset_field_id` ASC) ,
INDEX `webfield_webform` (`webform_id` ASC) ,
INDEX `webfield_tenant` (`tenant_id` ASC) ,
INDEX `webfield_label` (`label_id` ASC) ,

```

```

INDEX `webfidl_label` (`label_id` ASC) ,
CONSTRAINT `webfield_label`
  FOREIGN KEY (`label_id` )
    REFERENCES `multitenant_webforms`.`label` (`label_id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
CONSTRAINT `webfield_preset_field`
  FOREIGN KEY (`preset_field_id` )
    REFERENCES `multitenant_webforms`.`preset_field` (`preset_field_id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
CONSTRAINT `webfield_tenant`
  FOREIGN KEY (`tenant_id` )
    REFERENCES `multitenant_webforms`.`tenant` (`tenant_id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 164
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `multitenant_webforms`.`list_value`

DROP TABLE IF EXISTS `multitenant_webforms`.`list_value` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`list_value` (
  `list_value_id` INT(10) NOT NULL AUTO_INCREMENT ,
  `value` VARCHAR(200) NOT NULL ,
  `text` VARCHAR(200) NOT NULL ,
  `position_in_list` INT(10) NOT NULL ,
  `default_value` TINYINT(1) NOT NULL DEFAULT '0' ,
  `added` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP ,
  `updated` TIMESTAMP NULL DEFAULT NULL ,
  `en` VARCHAR(200) NULL DEFAULT NULL ,
  `fi` VARCHAR(200) NULL DEFAULT NULL ,
  `ru` VARCHAR(200) NULL DEFAULT NULL ,
  `uk` VARCHAR(200) NULL DEFAULT NULL ,
  `webfield_id` INT(10) NULL DEFAULT NULL ,
  `preset_field_id` INT(10) NULL DEFAULT NULL ,
  PRIMARY KEY (`list_value_id`) ,
  INDEX `list_value_webfield` (`webfield_id` ASC) ,
  INDEX `list_value_preset_field` (`preset_field_id` ASC) ,
  CONSTRAINT `list_value_preset_field`
    FOREIGN KEY (`preset_field_id` )
      REFERENCES `multitenant_webforms`.`preset_field` (`preset_field_id` )
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `list_value_webfield`
    FOREIGN KEY (`webfield_id` )

```

```

REFERENCES `multitenant_webforms`.`webfield` (`webfield_id` )
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 52
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `multitenant_webforms`.`mother_child_webfield`
-- -----
DROP TABLE IF EXISTS `multitenant_webforms`.`mother_child_webfield` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`mother_child_webfield` (
`mother_id` INT(10) NOT NULL ,
`child_id` INT(10) NOT NULL ,
PRIMARY KEY (`mother_id`, `child_id`) ,
INDEX `mother` (`mother_id` ASC) ,
INDEX `child` (`child_id` ASC) ,
CONSTRAINT `child`
FOREIGN KEY (`child_id` )
REFERENCES `multitenant_webforms`.`webfield` (`webfield_id` )
ON DELETE NO ACTION
ON UPDATE NO ACTION,
CONSTRAINT `mother`
FOREIGN KEY (`mother_id` )
REFERENCES `multitenant_webforms`.`webfield` (`webfield_id` )
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `multitenant_webforms`.`received_value`
-- -----
DROP TABLE IF EXISTS `multitenant_webforms`.`received_value` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`received_value` (
`received_value_id` INT(10) NOT NULL AUTO_INCREMENT ,
`value` VARCHAR(1000) NOT NULL ,
`model` VARCHAR(100) NULL DEFAULT NULL COMMENT 'Model that this data is assigned
to.' ,
`added` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP ,
`webfield_id` INT(10) NOT NULL ,
`user_id` INT(10) NOT NULL ,
PRIMARY KEY (`received_value_id`) ,
INDEX `received_value_webfield_tenant` (`webfield_id` ASC) ,
INDEX `received_value_user` (`user_id` ASC) ,
CONSTRAINT `received_value_user`
FOREIGN KEY (`user_id` )

```

```

REFERENCES `multitenant_webforms`.`account` (`user_id` )
ON DELETE NO ACTION
ON UPDATE NO ACTION,
CONSTRAINT `received_value_webfield`
FOREIGN KEY (`webfield_id` )
REFERENCES `multitenant_webforms`.`webfield` (`webfield_id` )
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `multitenant_webforms`.`user_webfield`
-- -----
DROP TABLE IF EXISTS `multitenant_webforms`.`user_webfield` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`user_webfield` (
`user_id` INT(10) NOT NULL ,
`webfield_id` INT(10) NOT NULL ,
PRIMARY KEY (`user_id`, `webfield_id`) ,
INDEX `user_webfield_user` (`user_id` ASC) ,
INDEX `user_webfield_webfield` (`webfield_id` ASC) ,
CONSTRAINT `user_webfield_user`
FOREIGN KEY (`user_id` )
REFERENCES `multitenant_webforms`.`account` (`user_id` )
ON DELETE NO ACTION
ON UPDATE NO ACTION,
CONSTRAINT `user_webfield_webfield`
FOREIGN KEY (`webfield_id` )
REFERENCES `multitenant_webforms`.`webfield` (`webfield_id` )
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----
-- Table `multitenant_webforms`.`webform`
-- -----
DROP TABLE IF EXISTS `multitenant_webforms`.`webform` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`webform` (
`webform_id` INT(10) NOT NULL AUTO_INCREMENT ,
`name` VARCHAR(100) NULL DEFAULT NULL ,
`added` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ,
`updated` TIMESTAMP NULL DEFAULT NULL ,
`captcha` TINYINT(1) NULL DEFAULT '0' ,
`can_be_mother` TINYINT(1) NULL DEFAULT '1' ,
PRIMARY KEY (`webform_id` )
ENGINE = InnoDB
AUTO_INCREMENT = 57

```

```
DEFAULT CHARACTER SET = utf8;
```

```
-- -----
-- Table `multitenant_webforms`.`user_webform`
-- -----
DROP TABLE IF EXISTS `multitenant_webforms`.`user_webform` ;

CREATE TABLE IF NOT EXISTS `multitenant_webforms`.`user_webform` (
  `user_id` INT(10) NOT NULL ,
  `webform_id` INT(10) NOT NULL ,
  PRIMARY KEY (`user_id`, `webform_id`) ,
  INDEX `webform_tenant_webform` (`webform_id` ASC) ,
  INDEX `webform_tenant_user` (`user_id` ASC) ,
  CONSTRAINT `webform_tenant_user`
    FOREIGN KEY (`user_id` )
    REFERENCES `multitenant_webforms`.`account` (`user_id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `webform_tenant_webform`
    FOREIGN KEY (`webform_id` )
    REFERENCES `multitenant_webforms`.`webform` (`webform_id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

3: FACES-CONIG.XML PROJECT CONFIGURATION FILE

```

<?xml version='1.0' encoding='UTF-8'?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd" version="2.0">
    <application>
        <locale-config>
            <default-locale>en</default-locale>
            <supported-locale>en</supported-locale>
            <supported-locale>fi</supported-locale>
            <supported-locale>ru</supported-locale>
            <supported-locale>uk</supported-locale>
            <supported-locale>zh</supported-locale>
        </locale-config>
        <resource-bundle>
            <base-name>com.mhgsystems.ui.resources.UIResources</base-name>
            <var>ui</var>
        </resource-bundle>
        <message-bundle>com.mhgsystems.ui.resources.JSFResources</message-bundle>
        <render-kit>
            <renderer>
                <component-family>javax.faces.Message</component-family>
                <renderer-type>javax.faces.Message</renderer-type>
                <renderer-class>com.mhgsystems.commons.jsf.MessageRendererImpl</
render-class>
            </renderer>
        </render-kit>
    </application>
    <!--Managed beans-->
    <managed-bean>
        <managed-bean-name>login</managed-bean-name>
        <managed-bean-class>com.mhgsystems.ui.Login</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>register</managed-bean-name>
        <managed-bean-class>com.mhgsystems.ui.Register</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>home</managed-bean-name>
        <managed-bean-class>com.mhgsystems.ui.Home</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    <managed-bean>

```

```

<managed-bean-name>webFormView</managed-bean-name>
<managed-bean-class>com.mhgsystems.ui.WebFormView</managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>editWebForm</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.EditWebForm</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>newWebForm</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.NewWebForm</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>viewWebForm</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.ViewWebForm</managed-bean-class>

    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>account</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.AccountView</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>vmenu</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.VerticalMenu</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>editWebFieldStyle</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.EditWebFieldStyle</managed-bean-
class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>editWebFieldRights</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.EditWebFieldRights</managed-bean-
class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>listOptions</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.ListOptions</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>presetFieldView</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.PresetFieldView</managed-bean-class>

```

```

<managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>chooseLabel</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.ChooseLabel</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>hmenu</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.HorizontalMenu</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>confirmation</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.Confirmation</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<!--Validation managed beans-->
<managed-bean>
    <managed-bean-name>validationBean</managed-bean-name>
    <managed-bean-class>com.mhgsystems.commons.jsf.ValidationBean</managed-
bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<!--Validation-->
<validator>
    <validator-id>birthdayValidator</validator-id>
    <validator-class>com.mhgsystems.commons.jsf.BirthdayValidator</validator-
class>
</validator>
<validator>
    <validator-id>deadlineValidator</validator-id>
    <validator-class>com.mhgsystems.commons.jsf.DeadlineValidator</validator-
class>
</validator>
<validator>
    <validator-id>emailValidator</validator-id>
    <validator-class>com.mhgsystems.commons.jsf.EmailValidator</validator-
class>
</validator>
<!--Administration-->
<managed-bean>
    <managed-bean-name>configuration</managed-bean-name>
    <managed-bean-class>com.mhgsystems.ui.Configuration</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<!-- Sessions -->
<managed-bean>
    <managed-bean-name>sessionBean</managed-bean-name>

```

```
<managed-bean-class>com.mhgsystems.ui.SessionBean</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<!--Navigation-->
<navigation-rule>
    <from-view-id>*</from-view-id>
    <navigation-case>
        <from-outcome>register</from-outcome>
        <to-view-id>/register.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
    <navigation-case>
        <from-outcome>login</from-outcome>
        <to-view-id>/login.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
    <navigation-case>
        <from-outcome>home</from-outcome>
        <to-view-id>/home.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
    <navigation-case>
        <from-outcome>newWebForm</from-outcome>
        <to-view-id>/newWebForm.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
    <navigation-case>
        <from-outcome>newChildWebForm</from-outcome>
        <to-view-id>/newChildWebForm.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
    <navigation-case>
        <from-outcome>editWebFieldStyle</from-outcome>
        <to-view-id>/editWebFieldStyle.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
    <navigation-case>
        <from-outcome>editWebFieldRights</from-outcome>
        <to-view-id>/editWebFieldRights.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
    <navigation-case>
        <from-outcome>listOptions</from-outcome>
        <to-view-id>/listOptions.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
    <navigation-case>
        <from-outcome>presetField</from-outcome>
        <to-view-id>/presetField.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
```

```
</navigation-case>
<navigation-case>
    <from-outcome>chooseLabel</from-outcome>
    <to-view-id>/chooseLabel.xhtml</to-view-id>
    <redirect/>
</navigation-case>
<navigation-case>
    <from-outcome>closePopup</from-outcome>
    <to-view-id>/closePopup</to-view-id>
    <redirect/>
</navigation-case>
<navigation-case>
    <from-outcome>editWebForm</from-outcome>
    <to-view-id>/editWebForm.xhtml</to-view-id>
    <redirect/>
</navigation-case>
<navigation-case>
    <from-outcome>viewWebForm</from-outcome>
    <to-view-id>/viewWebForm.xhtml</to-view-id>
    <redirect/>
</navigation-case>
<navigation-case>
    <from-outcome>fillWebForm</from-outcome>
    <to-view-id>/fillWebForm.xhtml</to-view-id>
    <redirect/>
</navigation-case>
<navigation-case>
    <from-outcome>account</from-outcome>
    <to-view-id>/account.xhtml</to-view-id>
    <redirect/>
</navigation-case>
<navigation-case>
    <from-outcome>confirmation</from-outcome>
    <to-view-id>/confirmation.xhtml</to-view-id>
    <redirect/>
</navigation-case>
<!--Administration-->
<navigation-case>
    <from-outcome>admin</from-outcome>
    <to-view-id>/admin/home.xhtml</to-view-id>
    <redirect/>
</navigation-case>
<navigation-case>
    <from-outcome>configuration</from-outcome>
    <to-view-id>/configuration.xhtml</to-view-id>
    <redirect/>
</navigation-case>
</navigation-rule>
</faces-config>
```

4: DESCRIPTION OF ENTITIES OF THE DATABASE

Let us discuss briefly each entity in the database. All entities (or tables) are listed with their attributes described in details. Each attribute is indicated by its name and its type. Attributes that are underlined serve as primary keys.

- **webform**: this entity represents a web form that can be created by users and later be used by different tenants of the program.

Attributes:

- webform_id `INT(10)` - Primary key, automatically incremented, not NULL.
- `name` `VARCHAR(100)` - Name of the form.
- `added` `TIMESTAMP` - Timestamp indicating time of creation.
- `updated` `TIMESTAMP` - Timestamp indicating time of last update (if any).
- `captcha` `TINYINT(1)` - Boolean variable indicating if the form requires CAPTCHA image.
- `can_be_mother` `TINYINT(1)` - Boolean variable indicating if other web forms can inherit from the form in question.

- **webfield**: this entity represents a web field that is a part of a web form. One web field may be used by different users. It is the biggest and possibly the most important table in the database. It describes web fields that are paired with a tenant that uses it, web form that the field is a part of and a label that is rendered next to the field. It holds information about customisation that is applied to the field, such as colour of the label, width of text inputs, label font, etc. It is designed in a way that makes further improvements and extension possible. If one wanted to add new ways of customising web forms and web fields in them, adding a new column to this table should be enough.

Attributes:

- webfield_id `INT(10)` - Primary key, automatically incremented, not NULL.
- `type` `INT(10)` - Defines type of the web field. The same web field returning one defined piece of data may be presented in different ways. For example, location can be received by inputting text into an input field, picking point on embed map or by manually inputting

latitude and longitude in two input fields, which can be presented by one object of type **webfield**. Type is represented by a code of type Integer.

- `popup_message VARCHAR(200)` - Optional popup message that appears when users hover on the web field.
- `default_value VARCHAR(200)` - Value that is given by default. Not compulsory.
- `colour VARCHAR(45)` - It allows setting colour of the label, if applicable.
- `label_id INT(10)` - ID of the label that is used with the field.
- `label_font VARCHAR(150)` - Font of the label assigned to the field.
- `label_font_size INT(10)` - Size of text, if text can be entered into the web field.
- `input_width INT(10)` - Width of the element. If applicable.
- `input_height INT(10)` - Height of the element. If applicable.
- `input_size INT(10)` - Amount of characters that users may input.
- `required TINYINT(10)` - Shows if the field is required for filling in by users.
- `added TIMESTAMP` - Timestamp indicating time of creation.
- `updated TIMESTAMP` - Timestamp indicating time of last update (if any).
- `position_in_webform INT(10)` - Position relative to other elements of the web form.
- `preset_field_id INT(10)` - Foreign key to **preset_field**'s `preset_field_id INT(10)`.
- `webform_id INT(10)` - Foreign key to **webform**'s `webform_id INT(10)`.
- `tenant_id INT(10)` - Foreign key to **tenant**'s `tenant_id INT(10)`.
- `textarea_col INT(10)` - Number of columns in text areas. If applicable.
- `textarea_row INT(10)` - Number of rows in text areas. If applicable.
- `default_value1 VARCHAR(200)` - Value that is given by default to the second element in the field. Not compulsory.
- `default_value2 VARCHAR(200)` - Value that is given by default to the third element in the field. Not compulsory.
- `default_value3 VARCHAR(200)` - Value that is given by default to the forth element in the field. Not compulsory.
- `default_value4 VARCHAR(200)` - Value that is given by default to the fifth element in the field. Not compulsory.
- `default_value5 VARCHAR(200)` - Value that is given by default to the sixth element in the field. Not compulsory.

- **tenant**: this entity describes companies that use the application as its tenants. Each tenant has users (represented by an entity **account**) that use web forms. Tenants, in this case, may

be viewed as groups of users. Tenants may exist without any users, although no users may work without a hosting tenant. A more simplified case may be considered, where tenants act as users. Although adding this extra layer of control of those who use the application gives bigger flexibility and a more extended set of features.

Attributes:

- tenant_id `INT(10)` - Primary key, automatically incremented, not NULL.
- name `VARCHAR(100)` - Name of the tenant (company).
- added `TIMESTAMP` - Timestamp indicating time of creation.
- updated `TIMESTAMP` - Timestamp indicating time of last update (if any).
- number_webforms_per_page `INT(10)` - A number of web forms shown per page for users of the tenant.
- number_webforms_sidebar `INT(10)` - A number of web forms shown in a sidebar for users of the tenant.
- number_webforms_verified `INT(10)` - A number of verified web forms shown per page for users of the tenant. May be used in further enhancements of the test case program.
- number_webforms_unverified `INT(10)` - A number of unverified web forms shown per page for users of the tenant. May be used in further enhancements of the test case program.
- instance_url `CHAR(200)` - An URL address by which users of the tenant can access the test case program.
- api_key `CHAR(36)` - API key given to the tenant.
- security_key `CHAR(200)` - Security key given for enhanced security.

- **account:** this entity describes users that use the application as members of enterprises represented by an entity **tenant**. Many users can be members of one company. Separating users in this way allows giving privileges and usage of different web forms within one company, not just on a firm level.

Attributes:

- user_id `INT(10)` - Primary key, automatically incremented, not NULL.
- username `VARCHAR(65)` - Username used for authenticating.
- password `CHAR(32)` - Password encrypted using MD5 checksum. Used for authenticating.
- firstname `VARCHAR(40)` - User's first name.
- surname `VARCHAR(40)` - User's surname.

- `email` `VARCHAR(200)` - User's email address.
- `avatar` `VARCHAR(100)` - Address by which a user's avatar image can be retrieved.
- `admin` `TINYINT(1)` - Boolean variable that indicates that the user has administrative privileges. May be used in further enhancements of the test case program.
- `added` `TIMESTAMP` - Timestamp indicating time of creation.
- `updated` `TIMESTAMP` - Timestamp indicating time of last update (if any).
- `tenant_id` `INT(10)` - Foreign key to `tenant`'s `tenant_id` `INT(10)`.
- `verified` `TINYINT(1)` - Boolean variable that indicates that the user has been verified by administrators. May be used in further enhancements of the test case program.

- **user_webfield**: it serves as a link for M:N relationship that users and web fields share. It has a combined primary key. By using this table privileges are granted to users concerning access control to web fields that are members of web forms. Additionally, it serves as a layer of security separating data from other users and tenants.

Attributes:

- `user_id` `INT(10)` - Foreign key to `account`'s `user_id` `INT(10)`.
- `webfield_id` `INT(10)` - Foreign key to `webfield`'s `webfield_id` `INT(10)`.

- **user_webform**: it serves as a link for M:N relationship that users and web forms share. It has a combined primary key. By using this table privileges are granted to users concerning access control to web forms. Additionally, it serves as a layer of security separating data from other users and tenants.

Attributes:

- `user_id` `INT(10)` - Foreign key to `account`'s `user_id` `INT(10)`.
- `webform_id` `INT(10)` - Foreign key to `webform`'s `webform_id` `INT(10)`.

- **mother_child_webfield**: it served as a table that list all web fields that are inherited from “mother” fields.

Attributes:

- `mother_id` `INT(10)` - Foreign key to `webfield`'s `webfield_id` `INT(10)`.
- `child_id` `INT(10)` - Foreign key to `webfield`'s `webfield_id` `INT(10)`.

- **preset_field**: it holds data about preset web fields that users may choose instead of creating custom ones. When used a value of `preset_field_id` foreign key in **webfield** is set to an ID of the field.

Attributes:

- `preset_field_id INT(10)` - Primary key, automatically incremented, not NULL.
- `name VARCHAR(200)` - Name of the field that appears in a list that users see.
- `type VARCHAR(45)` - Type of the list. May be a set of radio buttons, a list of values, etc.

Note: a custom domain may be used, though it would limit further extension and customisation of preset web fields.

- **label**: it holds labels that are used for web fields. Current implementation of the entity supports four localisation. This table may be extended to provide support for a bigger number of languages.

Attributes:

- `label_id INT(10)` - Primary key, automatically incremented, not NULL.
- `en VARCHAR(200)` - Text of the label in English.
- `ru VARCHAR(200)` - Text of the label in Russian.
- `fi VARCHAR(200)` - Text of the label in Finnish.
- `uk VARCHAR(200)` - Text of the label in Ukrainian.

- **list_value**: it holds values for elements of lists that are used for web fields. They may or may not be used. One element of the list may be used for different fields. Additionally, preset fields may utilise this entity.

Attributes:

- `list_value_id INT(10)` - Primary key, automatically incremented, not NULL.
- `value VARCHAR(200)` - Value that is assigned to the element.
- `text VARCHAR(200)` - Text in default language of a label assigned to the list option.
- `position_in_list INT(10)` - Position in the list.
- `default_value TINYINT(1)` - Boolean variable. It set to “1” if the list option is a default one in the list.

- added `TIMESTAMP` - Timestamp indicating time of creation.
- updated `TIMESTAMP` - Timestamp indicating time of last update (if any).
- `webfield_id INT(10)` - Foreign key to `webfield`'s `webfield_id INT(10)`.
- `preset_field_id INT(10)` - Foreign key to `preset_field`'s `preset_field_id INT(10)`.

- **received_value**: this is a table that is used as a bridge between a customisable web field and a piece of database that is meant to be used for storing received from the field data.

Attributes:

- `received_value_id INT(10)` - Primary key, automatically incremented, not NULL.
- `value VARCHAR(200)` - Value that is assigned to the element.
- `model VARCHAR(100)` - name of a model that represents received data. Not compulsory in this test case.
- added `TIMESTAMP` - Timestamp indicating time of creation.
- `webfield_id INT(10)` - Foreign key to `webfield`'s `webfield_id INT(10)`.
- `user_id INT(10)` - Foreign key to `account`'s `user_id INT(10)`.