MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Web Services Sharing System

LICENSE THESIS

Graduate: **Florin Corde**

Supervisors: **Asis. Ing. Cosmina Ivan**

2016

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. Dr. Eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. Dr. Eng. Rodica POTOLEA

Graduate: **Florin Corde**

Web Services Sharing System

1. **Project proposal:** The purpose of this project is to define and construct a system which will be able to provide a Questions and Answers platform which will allow the users to provide as answers to the questions working implementations in the form of web services.

2. **Project contents:** Table of contents, Introductions, Project Objectives, Bibliographic Research, Analysis and Theoretical Foundation, Detailed Design and Implementation, Testing and Validation, User's manual, Conclusions, Bibliography and Glossary.

3. **Place of documentation**: Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants**: Asis. Ing. Cosmina Ivan

5. **Date of issue of the proposal:** November 1, 2015

6. **Date of delivery:** June 30th, 2016

Graduate: _____

Supervisor: _____

Table of Contents

Chapter 1. Introduction

The purpose of our project is to define and construct a system which will be able to provide an alternative way of publishing web services in order to supply a web service library for the users of the system. Also we will develop a web application which will allow the users of the system to work together for finding optimal solutions for various programming problems or exercises.

With the help of the Web Services Sharing System the users will be able to post questions which will be answered by the other users of the system. This system gives users the option to answer each other's question by posting a working implementation of the problem in the form of a web service which can be later used in other applications across multiple platforms. The users of the system will have the possibility of deciding which of the various implementations for a certain problem is optimal.

A common approach to Client-Server software development is to build the application from scratch, creating the resources and algorithms needed on the server and publishing those resources through web services. In this context, the community of software developers would need a system to provide a way of speeding up this process by putting at the software developer's disposal a library of web services which can be enlarged by the community, including themselves.

1.1. Project context

Software development has seen a massive growth in the last few years. These new tools allow the programmers to code faster, better and easier than ever before. The automatization of programming tasks like connection to databases, communications between clients and servers, concurrent access control to resources are now simple tasks because of frameworks developed in the last few years.

Also software development communities have seen a big growth in the last few years. A lot of question and answers sites have appeared lately. With the help of these web sites the users can ask for help for solving problems that they can't solve on their own. These question and answers websites have the goal of becoming a library of detailed answers for any programming question by having their users work together to achieve this goal.

Online IDEs are quickly gaining popularity also. The programmers have the possibility of accessing big computational resources from mediocre machines. Since using web services for a lot of computations in our applications, it's simple to see that we can do the same thing with the code writing process. The process of compiling, or syntax checking the code can be sustained by a powerful server instead of a personal computer. Another reason for their growth is that they are platform independent, being accessible from any smart device from any location with an internet connectivity. These IDEs offer the advantage of a workspace in a single and centralized environment in which all the users can build, edit and debug together. We can mention in this regard the following applications: Cloud9, Codeanywhere, Codenvy, Compilr etc.

1.2. Project requirements

In the context mentioned above, the purpose of this project is to offer a tool to the programmers' community that would be a question and answering system that will allow the users to swiftly publish web services as answers with the purpose of creating a web service library of optimal implementations for a various range of problems like frontend problems to backend problems to mathematical algorithms.

The main activities that will be implemented will be posting a web service, asking a question, responding to a question and testing the web services from the web service library. Posting a web service will require Java language knowledge and will offer the possibility of adding new maven dependencies, new imports and will give the programmer the full control over the code which will be posted in the web service.

There are also implicit functionalities to this system which are essential for achieving the implementation of the activities mentioned above like the user authentication system or the logging and testing system.

The Web Services Sharing System will be implemented in the Java programming language on a client-server architecture, with a separate module which will act as a container for the posted web services.

1.3. Project content

In the second chapter of this paper we will describe the main scenarios of the system: posting a web service, creating an issue by asking a question, commenting on a question by writing the solution in words, but also publishing a web service with the implementation described in the comment, and finally, testing a web service from the library. Also we will swiftly describe the non-functional requirements of the system, such as security and data integrity.

In the third chapter we will describe the domain/thematic of our project, where does our project stand in this domain and we will compare our system with the systems already existent on the market. We will take a close look at the existent web service publishing tools and their improvements in time, and compare our approach with the existent approaches. We will also take a close look in the domain of web IDE's, their advantages and disadvantages compared to classical desktop IDE's, and also explain how can our system improve this domain. We will also go into the concept of cloud computing and compare our questions and answers functionality with other question and answers systems and state the differences that separate our product from the other existent products.

In the fourth chapter we will take a close look on the theoretical foundation of the project. We will describe the technologies used for implementing the system along with argumentations about why we chose them, also we will describe the tools used for building the project and the tools needed to run the project. We will describe the architecture of the system, the way we integrate the pieces of the system together, and the protocol and design patterns used. We will give logic explanations for our chosen solutions and state the functional structure of the application. We will also fully describe the use cases of the project, the interaction between the actors of the system and the functional and non-functional requirements of the project.

In the fifth chapter we will present the detailed design and implementation of the Web Services Sharing system. The purpose of this chapter is to document the developed

application in such a way to be ready for future development. Here we will present the main modules of our application, the package structure and the description of the most important classes and files. We will show class diagrams along with database structure and we will highlight the pieces of the code that use the most relevant APIs for our application.

In the sixth chapter of this paper we will highlight the testing and validations techniques used for our project. We will describe the integration and unit testing tools along with performance figures for the various operations done using the Web Service Sharing System.

The seventh chapter represents the user's manual. Here we will describe the resources needed for installing and running the application and the most important elements of our user interface. We will highlight the configuration files that should be adjusted in the case of installing the project on another machine and also the file structure that has to be maintained in order for the project to work properly.

In the eighth chapter we will draw the conclusions after working on this project, the summary of our contribution and a short description of the possibilities of improvements and future development.

Chapter 2. Project Objectives

The objectives of the project can be placed into two categories: bibliographic study of general solutions which will give software developers support in posting web services, and the implementation of a system prototype which will offer a set of features in that regard.

The implementation objectives include creating a system with the following main scenarios:

- **Posting a web service.** This scenario allows the users of the system to post a web service through the user interface. For posting a web service the user has to have Java language knowledge, Maven framework knowledge and a basic understanding of RestFULL services.
- **Asking a question.** This scenario allows the users to ask programming questions through the user interface. These questions will be seen by the other users of the application and can be answered either by posting a comment, either by posting a proposed solution for the specific problem.
- **Answering a question.** This scenario allows the users to respond to each other's questions. The users can also contribute with liking each other's comments so that the best answer can be clearly distinguished.
- **Testing a web service from the library.** This scenario allows the user to test the web services posted by the users of the system. This objective has is important for the fact that these web services posted by the users can be used from all sorts of applications. Before using such a service the user needs to find out if the service which he intends to use into some other application works properly.
- **Sign up.** The users will be able to create and personalize a profile for using this application. They will authenticate in the application with a combination of username and password. In the future it can be developed a system of rewarding system in order to compensate the users which consistently respond correctly to other user's questions.
- **Authenticate.** The users which are not authenticated in the system don't have access to the system's functionality. Any user who has signed up for our application has access to all the web services in the web service library.

Next we will describe in more detail the four most important objectives of our system: publishing a web service, asking a question, answering a question and testing a web service from the library. The other scenarios are administrative and do not require further details.

2.1. Publishing a web service scenario

This scenario represents the main purpose of this application, to help programmers post web services easily. For implementing this operation we will need a basic web IDE in which the user will write the service's code.

In Chapter 3 of this paper we will look in more detail into the domain of web IDEs, cloud compilation and cloud computing.

2.2. Asking a question scenario

This scenario represents the Questions and Answers part of the system. We intend to create a Question and Answers module resembling systems already existent, like StackOverflow, with the extra feature of allowing the users to post actual working implementation along their proposed solution.

The users of the system will be able to ask questions which will be seen on a board by all the other users of the system.

2.3. Answering a question scenario

This scenario is strictly connected to the scenario described above. The users will be able to answer questions posted by other users by commenting on the question's page.

After posing a comment, at the click of a button the user has the option post a web service which will be associated to that comment. This web service will represent the implementation of his proposed solution for that specific question.

All the comments for a specific question will be in a hierarchy, so the most relevant comment will be at the top.

2.4. Testing a web service scenario

This scenario allows the users to test the web services which were published by the users in the community. If the web service has the expected behavior the user can then use that web service in other application, because posted web services will be available for use to anybody.

Chapter 3. Bibliographic Research

In this chapter we are going to describe our research on the subjects of this paper. We will take a look over the topics of Cloud Computing, Online IDEs and Questions and Answers systems.

We aim to build a library of web services which can be enhanced by the users of the system. As a result, users can split the functionality when building new applications and chose to publish algorithms within the Web Service Sharing System with the purpose of executing demanding computations on a remote server. That means that we would provide a service (the service of publishing a web service) over the Internet and we would provide the server's hardware to execute those services, which is the definition of cloud computing. Next we will take a closer look into this domain.

3.1. Cloud Computing

3.1.1. Definition of Cloud Computing

Cloud computing refers to both the application delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. [1] Cloud computing is a model for easy and convenient network access to configurable network resources such as servers, storage, applications and services that can be easily provided and released to the end users of the system.

3.1.2. Characteristics of Cloud Computing

As described in the NIST Definition of Cloud Computing [2], these are the essential characteristics of the cloud computing model:

- *On-demand self-service.* A client can get computing capabilities such as storage or processing resource automatically with no human interaction to serve him.
- *Broad network access.* The resources are accessible over a broad variety of thin or thick client platforms such as mobile phones, tablets, laptops or work stations.
- *Resource pooling.* The resources provided by the Cloud Computing system are pooled to serve multiple consumers dynamically. Each consumer may change its demands and the system should be able to reassign resources. The customer has no knowledge over the exact location of the provided resource but he may be able to specify the location at a higher level of abstraction (e.g., country, state or datacenter).
- *Rapid elasticity.* The resource can be scaled rapidly. To the consumer the resources available often appear to be unlimited and can be appropriated in any quantity at any time.
- *Measured service.* Cloud systems automatically control and optimize resource use. Resource usage can be controlled and reported providing transparency.

3.1.3. Service models of Cloud Computing

The services provided by this kind of systems are referred as Software as a Service (SaaS), but some vendors use terms such as IaaS and PaaS to describe their products. We will take a closer look over these concepts:

- *Software as a Service (SaaS).* The consumer has the option of using the provider's application running on a cloud infrastructure. These applications are accessible from various client devices such as tablets or web browsers.
- *Platform as a Service (PaaS).* This service model provides the capability to deploy onto the cloud infrastructure application created using programming languages, libraries, services and tools supported by the provider. The consumer does not manage the control of the infrastructure, but has control over the deployed application and configuration settings for the application-hosting environment.
- *Infrastructure as a Service (IaaS).* The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

In our case, for the Web Services Sharing System we put at the user disposal certain characteristics of Cloud Computing models such as network storage of programming routines, in the form of web services and the fact that the application server may be installed on a cloud infrastructure.

3.2. Online IDEs

The main purpose of this paper is to develop a tool that would make the process of posting a web service a simple task. For reaching this objective we will have to implement a web interface which will accommodate functionalities similar to those of an IDE. An IDE is an Integrated Development Environment and it provides the medium to execute programs for a certain programming language. An online IDE would achieve the same tasks, but the system would be a web application.

There are several online IDEs out there. In this section we will take a look over some of these systems and we will compare the end products with the objective we are trying to achieve in this paper.

3.2.1. Web Based IDE

This system has the objective of building a cloud compiler with the scope of diminishing costs in software production by moving to online IDE meanwhile accomplishing greater system control and increase of flexibility. Also multiple

programming languages are to be accommodated so that the users won't have to keep on their machine multiple compilers.

The system implements the following functionalities [3]:

- **Registration:** The system allows the clients to create a personalized account based on the user's personal data. The users will use this account to Log into the system.
- **Login:** A registered client has to log into the application based on its own username and secret password.
- **Make a new Project/File:** This module allows logged in users to create a new project in a workspace. Only the clients that created the workspace have full rights over it.
- **Open a Project/File:** This module permits users to open existing documents and tasks.
- **Delete Project/File:** This module allows clients to delete projects created by them or projects on which they have proper authority over.
- **Save:** This module allows the logged in clients to save their activities and records. These projects are stored in a Project DB and the documents are stored in Files.
- **Compile:** This module allows the clients to test their code by invoking a compiler. The result will be shown to the client.
- **Run:** This module allows the client to run their code. The outcome will be displayed on the interface.
- **Debug:** This module allows the users to implant breakpoints in the code when testing their program.
- **Share:** This module licenses logged in clients to share tasks to accomplish a certain goal.

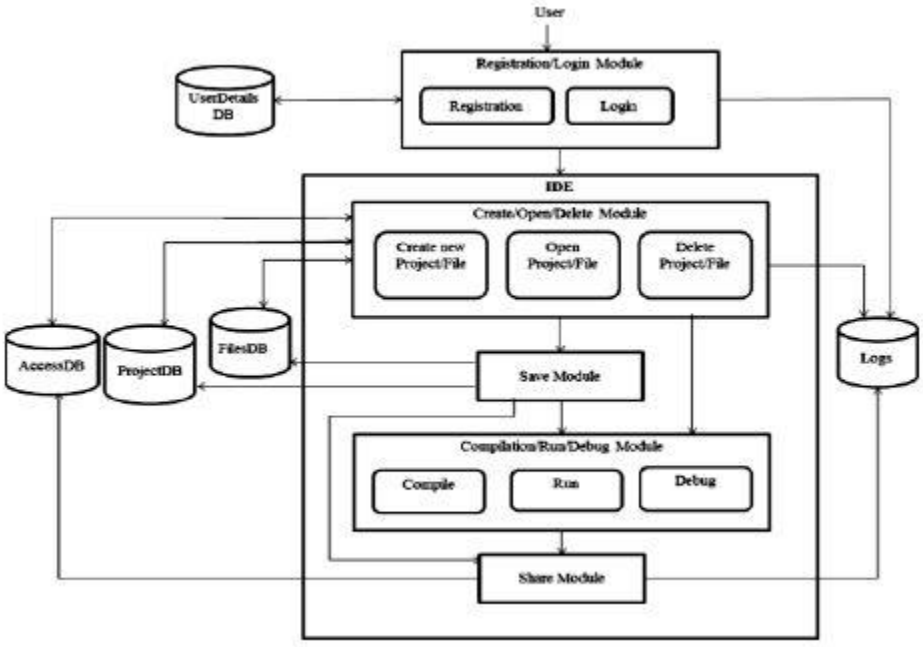

Figure 3-1 Web Based IDE architecture[3]

In term of functionalities, we are looking to develop a basic web IDE, and the compiler is our main goal. Out of the functionalities mentioned above our system will implement the following ones: registration, log in, save, compile, run and share. Within the Web Service Sharing system we look to implement an IDE for building a controller of a web service, so the system does not deal with a whole hierarchy of a project. As a consequence, the Web Service Sharing System won't show the users a structure of a project like the system described above, and won't implement the functionalities make a new project/file, open project/file and delete project/file.

However the Web service sharing system will bring into the discussion features not tackled by the system mentioned above. We are looking to implement a web IDE which will simplify the publishing of web services, so any class built with the help of our system would be available for use within other projects of any language. Also we are looking to build a questions and answers module with the objective of building a library of web services which are relevant for the difficulties that the programmers encounter day by day.

In the next section we will take a look on the Web Based IDE's implementation:

- **Web Services:** This project will use the Web Services Description Language (WSDL) to communicate between the frontend and the backend and will use SOAP messages with XML serialization.
- **Application Server:** Any Java application server will be usable.
- **Communication Protocol:** SOAP is used which depends on XML, HTTP and SMTP for message arrangement and transmission.

The web service sharing system will use a relatively similar approach of implementation with the system described above. We will use RESTFul web services between the frontend and the backend using the HTTP protocol of communication, and JSON for the data serialization. We will use the Tomcat application server to sustain our modules of the application.

Finally the Web based IDE and compilers were compared with the current windows based compilers to see the comparison in time and costs.

N Value	Turbo C		DEV C++		JAVA	
Type Of Compilers	Windows Based Compilers	Cloud Compilers	Windows Based Compilers	Cloud Compilers	Windows Based Compilers	Cloud Compilers
12345	39.29	25.50	34.56	8.353	15	4
5678	27.33	15.19	6.793	5.142	9	4
5630	18.70	9.19	12.07	6.785	15	7
6549	13.47	10.55	7.04	6.128	41	8
7395	11.45	9.15	11.23	6.273	9	5

Table 3.1 Run Time Analysis Table [3]

This table clearly state that cloud based compilers reduce the run time when contrasted to other ordinary compilers. This cloud compiler will eliminate out the need to introduce the compilers independently on local machines.

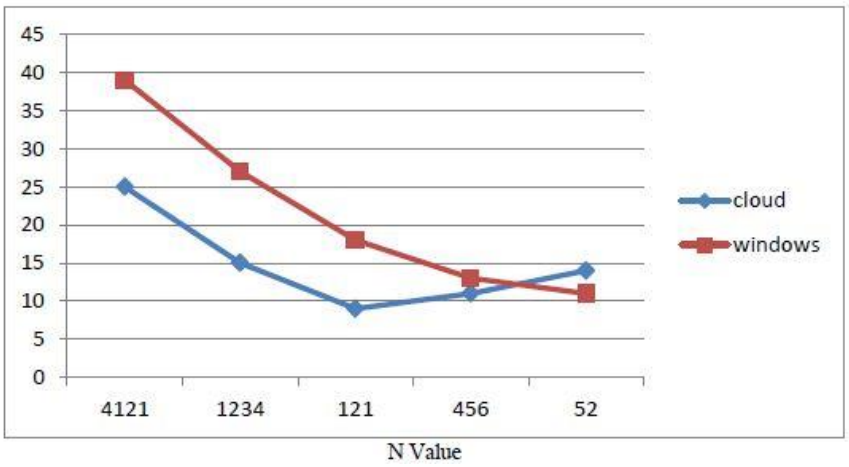

Figure 3-2 Cloud Vs Windows Compile Time [3]

3.2.2. Eclipse Che

Eclipse Che is a product developed by Eclipse and it provides a browser-based IDE, a RESTFul workspace server, Plug-ins for languages, framework and tools and an SDK for creating plug-ins and assemblies.

Che is a workspace server that runs on top of an application server like Tomcat. When the Che server is launched, the IDE is loaded as a Web application accessible via a browser. The browser downloads the IDE as a single page web app from the Che server. [1]

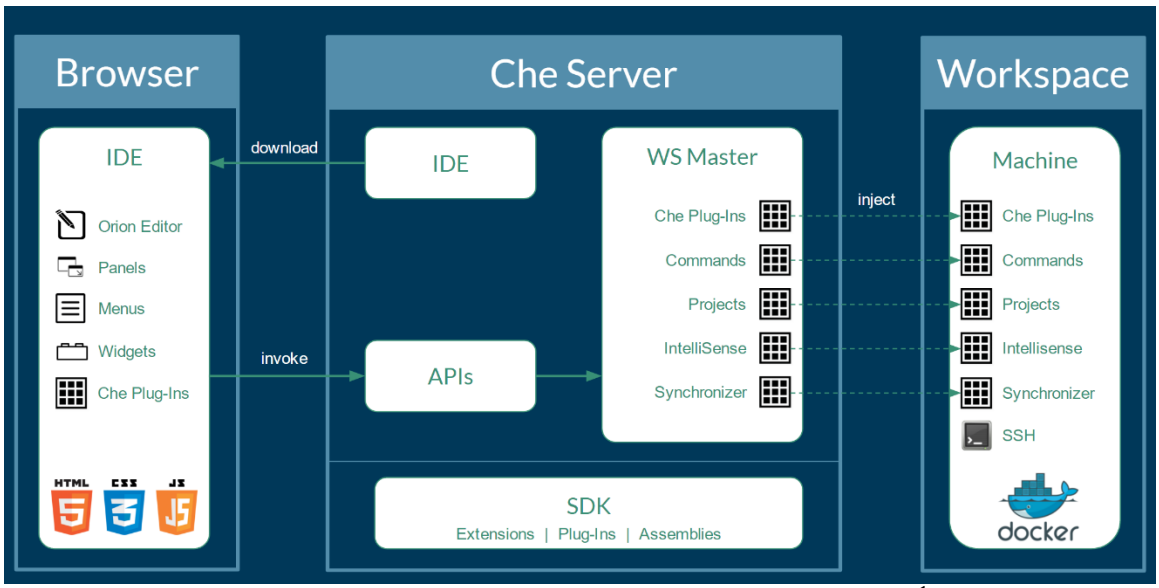

Figure 3-3 Overview of eclipse Che architecture.[1]

[1] https://eclipse-che.readme.io/docs/

A user interacts with the Web application, they will create workspaces, projects, environments, machines, and other artifacts necessary to code and debug a project. The IDE communicates with the Che over RESTFul APIs that manage and interact with a Workspace Master. A workspace can have many projects and it can be mapped to an external git or subversion repository.

Che provides an SDK for authoring new extensions, packaging extensions into plug-ins, and grouping plug-ins into an assembly. An assembly can either be executed stand alone as a new server, or, it can be installed onto desktops as an application using included installers.

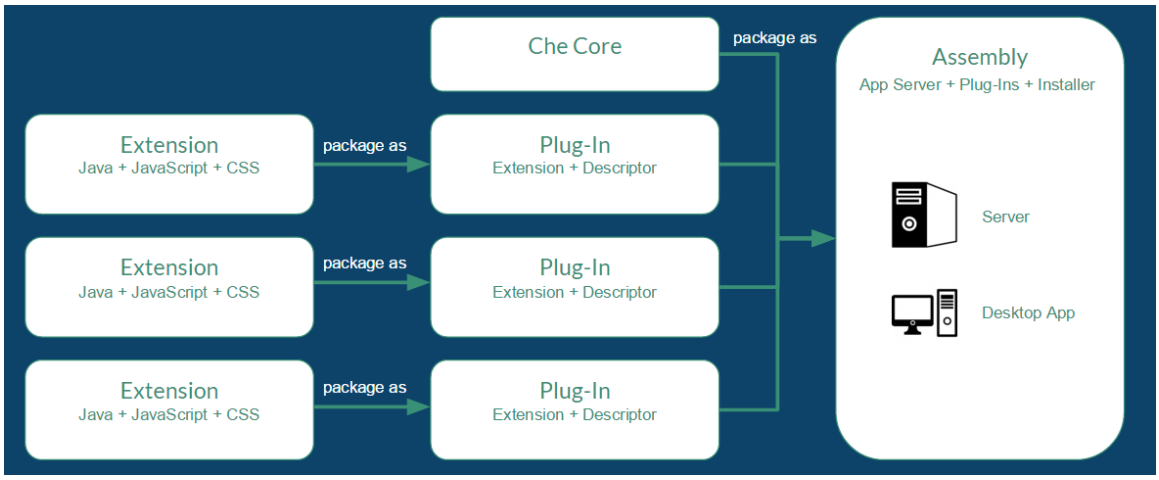

Figure 3-4 Eclipse Che extensibility - how extensions become plugins become assemblies.[1]

Eclipse Che functioning mechanism: When you develop with a desktop IDE, the workspace uses localhost as the execution environment for processes like build, run and debug. In a cloud IDE, localhost is not available, so the workspace server must generate the environments that it needs. These environments must be isolated from one another and scalable. We use Docker to generate containers that contain the software needed for each environment. Each workspace is given at least one environment, but users may create additional environments for each workspace if they want. Each container can have different software installed. Che installs different software into the machine based upon the project type. For example, a Java project will have the JDK, Git, and Maven installed. When a user is working within their Workspace, this container is booted by Che and the source code of the project is mounted within it. Developer actions like auto-complete and mvn clean install are processes that are executed within the container. Users can provide their own Dockerfiles that Che will build into images and extension developers can register Dockerfile templates associated with a project type. This allows Che to manage a potentially infinite number of environments while still giving users customization flexibility.[1]

The interface of Eclipse Che looks like Eclipse's desktop application interface. A snippet of it is shown in the figure below.

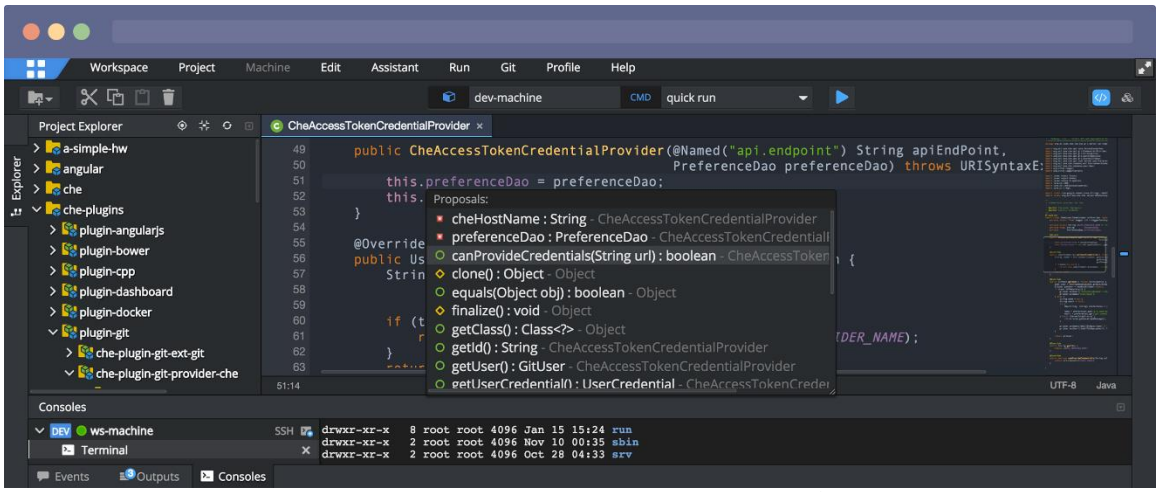

Figure 3-5 Che User Interface[1]

3.2.3. Browxy

Online Java compillers and runners make it easy to run Java code from anywhere even if the user does not have the installed compiller for that language on it's computer. This system, Browxy, brings a little more to the online IDEs world by letting it's users to publish their work, feature that is the closest that we could find of the Web Service Sharing System's feature of web service publishing. In the following section we will take a look over the features of browxy:

- **Authentication and Log in:** The users of the system have to create an account in order for them to save any work that they do. Un authenticated users however have access to compiling the project and running it, but can't save their work if they are not authenticated.
- **Open project:** The users of the system have the option of opening projects that they have previously worked on.
- **Download:** The users of the system can download the project that they have worked on in the from of an archive.
- **CRUD:** The users of the project have the option of creating, viewing, edditing and deleting their work. They have access to edit or remove all the applicaitons that they have chose to publish in the past.
- **Publish:** The users have the option to publish their code. This code will be part of a list with all the pieces of code published by all the users of the system. The user can comme back to the web site and test the functioning of he's previously published code, or test the code published by the other users of the system. Each piece of code can be accessed through an URL for fast access.
- **View Console:** A console text area is available for the users. Here it's outputted the build text and errors of compilation when trying to build the project. This console view works like a windows console.

- **Applet View:** If the user wants to build an application with a user interface he can test this application with the help of this applet view. When running the project the user has the option to display Jframes in that window.

Again, this system gives the option to build a whole project including packages and classes, feature that will not be implemented in the Web Service Sharing System. Our system will create a RESTFul controller class for each piece of code that the user publishes. However, there are a lot of similarityes between Browxy and the Web Service Sharing System in the fact that the users can publish their work for the other users to see and test. The main difference between the system described in this paper and Broxy in the publish respect is that the code published with the Web Service Sharing System will be calleble from other applications, while the code published with Browxy is just for testing and for viewing.

Another symilarity between Broxy and the Web Service Sharing System would be the possibility of creating, reading updating and deleting the work of the loggin user, the console view will be shown in the form of allert messages in the cases of errors, and the authentication and Log In will work based on the same principle with that of Browxy.

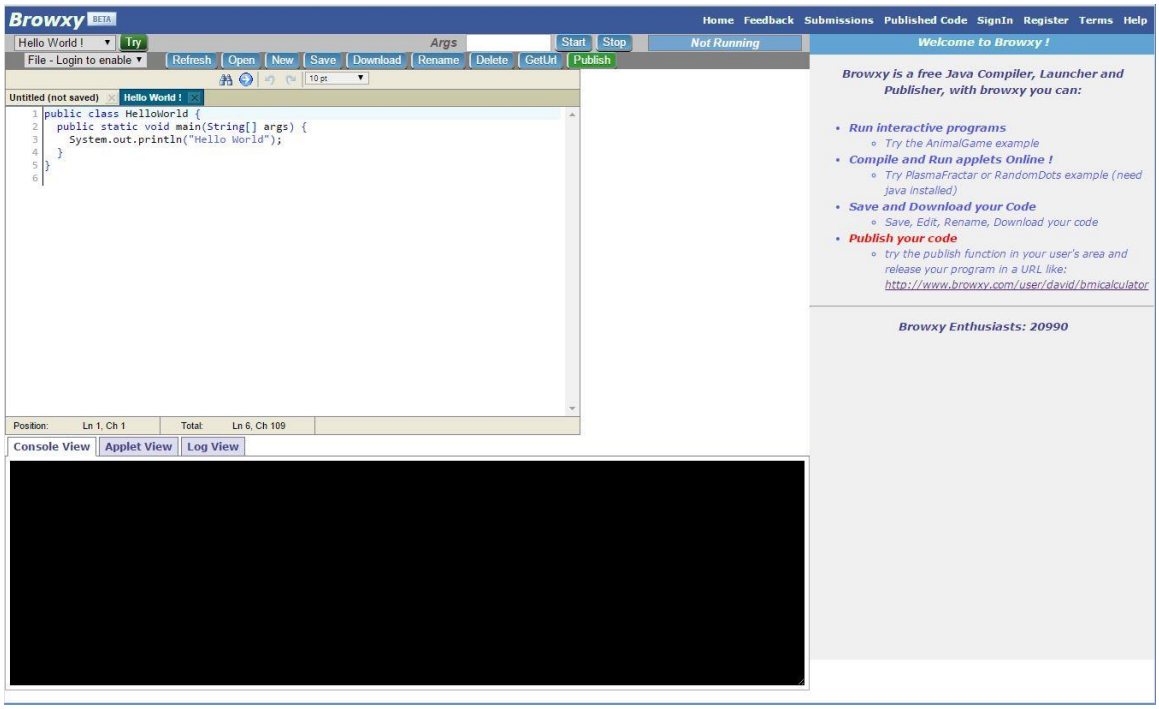

Figure 3-6 Browxy User Interface [4]

3.3. Questions and answers systems

Historically, mailing lists have been the preferred means for coordinating developmend and user support activities. Whit the emergence and popularity of social Q&A sites such as the StackExchange network (e.g. StackOverflow), this is beginning to change. Such sites offer different sociotechnical incentives to their participants than maililg

lists, e.g. tich web enviroments to store and manage content collaboratively, or a place to showcase their knowledge and expertise more vividly to peers or potential recrutiers. [5]

Question and answering websites are begining to become large repositories of valuable knowledge. While most Q&A sites were initially aimed at providing useful answers to the question asker, there has been a marked shift towards question answering as a community-driven knowledge creation process whose end product can be of enduring value to a broad audience. [6]

Over the last years we could see a trend in the migration from mailing lists to questions answers systems.

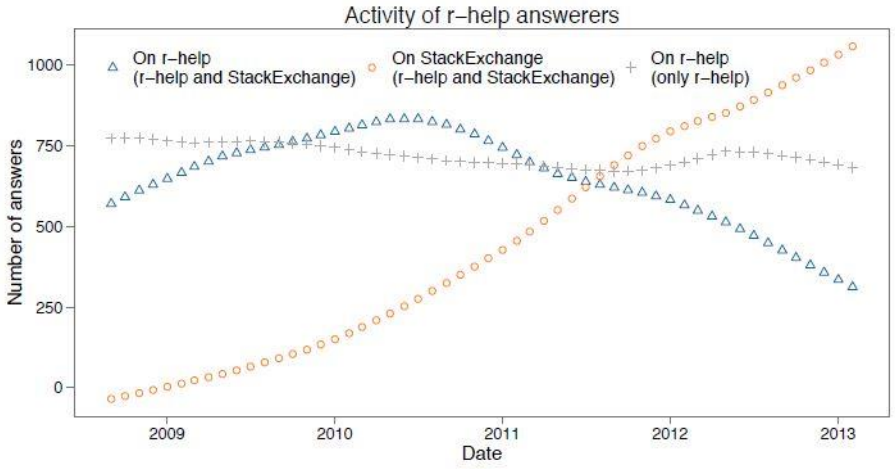

Figure 3-7 Number of questions answered on r-help and StackExchange [5]

We will investigate now whether there are significant differences between the speed with which r-help participants also active on StackExchange answer questions on r-help versus on StachExchange. If the incentives on StachExchange (e.g. gamification, more attractive user interface) do not influence behaviour, then we should not observe any meaningful differences. If instead users are engaging in the race for reputation and badges, then they might try to answer moree questions as well as answer questions faster on StackExchange than on the mailing list, where they are not rewarded for their answer.[5]

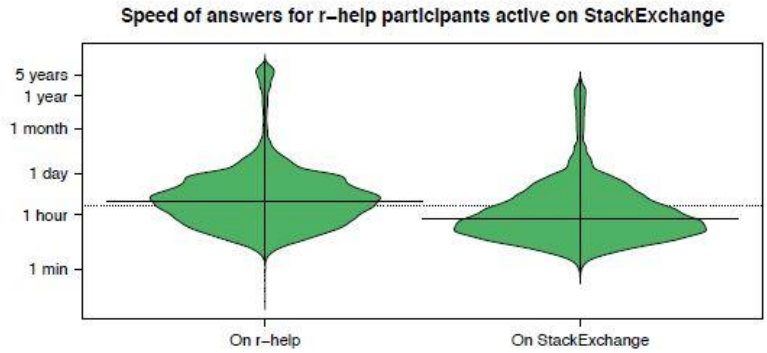

Figure 3-8 Answer speed for r-help users active on StackExchange [5]

This graph clearly suggests that the StackExchange answers are significantly faster with a median of 47 minutes versus 3 hours on r-help. This confirms that mailing lists prticipants behave differently when on StackExchange, where they are rewarded for their efforts more.

In conclusion, the Web Service Sharing System will implement some of the features described in this section about question and answering web sites but we are trying to bring something extra to this dommain by allowing the users that answer questions to publish working solutions together with their answers in the form of web services.

Chapter 4. Analysis and Theoretical Foundation

4.1. Technologies used

4.1.1. The Spring Framework

The Spring Framework[2] is a Java platform that provides an infrastructure for developing Java applications. This framework can be used for easily achieving benefits such as executing a Java method in a database transaction without the need of using transaction APIs, transforming a local method to a procedure that can be called remotely without the need of using remote APIs, handle messages without the need of using JMS or RabbitMQ APIs.

In the last few years Spring has released a Spring Boot module. The purpose of this module is to make it easy to create stand-alone, production-grade based Applications that you can "just run". The goals of Spring Boot: [2]

- Provide a radically faster and widely accessible getting started experience for all Spring development.
- Be opinionated out of the box, but get out of the way quickly as requirements start to diverge from the defaults.
- Provide a range of non-functional features that are common to large classes of projects (e.g. embedded servers, security, metrics, health checks, externalized configuration)
- Absolutely no code generation and no requirement for XML configuration.

Spring's compatibility with REST services is another reason we chose to use this framework in the implementation of the Web Service Sharing system. Spring Data REST builds on top of Spring Data repositories and automatically exports those as REST resources. It leverages hypermedia to allow clients to find functionality exposed by the repositories and integrates these resources into resources into related hypermedia based functionality automatically.

Spring Data REST is itself a Spring MVC application and is designed in such a way that it should integrate with the existing Spring MVC application with little effort. An existing layer of services can run alongside Spring Data REST with only minor considerations. This functionality is very important to the system we will develop because of the fact that we will use Spring configured REST controllers in the web services module. Also Spring REST will be used in the main module as a web service to expose the data base resources and to give the web application access to the various services that will be implemented in the backend.

The Spring Framework is made of several modules including Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging and Test.

[2] http://docs.spring.io/

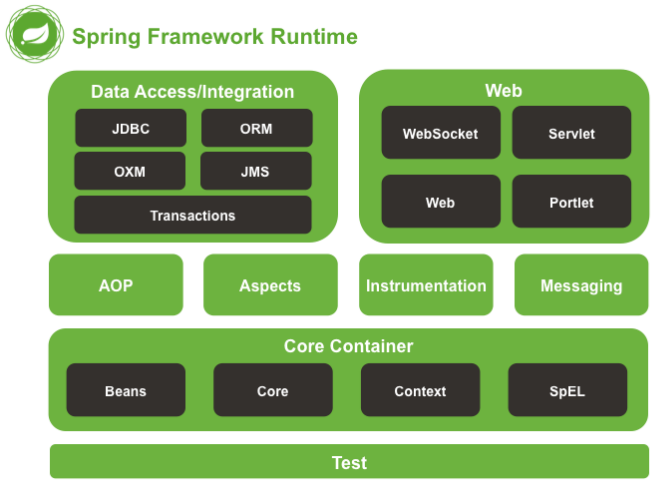

Figure 4-1 Spring Framework Overview[2]

The following modules were used for the implementation of the Web Service Sharing System:[2]

- **Core Container**

We used in application the spring's core container for its Dependency Injection features between the RESTFull controllers on the server side and the Services layer. We also used spring-context from the same module for managing the beans on the server side of the application. This Context module supports Java EE features such as EJB.

We chose to use Spring over EJB for the framework's much wider range of features.

- **Data Access/Integration**

We chose to use Spring also for database transactions since the framework offers this type of features. The Data Access/Integration module of the Spring Framework consists of JDBC, ORM, OXM, JMS and Transaction modules. Form this layer we user the JDBC module for mapping our Plain Old Java Objects to the database and for all database transactions.

- **Web**

From the web layer we use the spring-mvc module (Web-Servlet module) and the REST Web services implementation. In the Web Services Sharing System we will use .jsp pages for the front end, and AngularJS to make calls to the REST services which will provide the data from the database and from other backend services and algorithms.

The test module supports unit tests and integration tests using Junit or TestNG. We will use this module for testing the backend of our proof of concept.

- **Test**

The reason we chose spring is for wiring different frameworks together. Since Spring Boot was released in 2014 configuring all these different frameworks becomes easier, since no configuration xml is needed any more. Spring incorporates the four functionalities described earlier into one easy to configure framework.

Dependency Injection should make the code less dependent on the container than it would be with traditional Java EE development. The POJOs that make up your application should be testable in Junit or TestNG tests with objects simply instantiated using the new operator, without Spring or any other container. We can use mock objects to test code in isolation. Spring Framework provides mock objects and testing support classes.

The Web Service Sharing System will be tested using Spring configured JUnit tests.

4.1.2. Maven

The Maven word means accumulator of knowledge. This tool is a building and management tool for Java projects. This framework allows a developer to make the build process easy and to keep track of the external JARs used in the project.

This framework uses an .xml file called project object model (POM) in which the programmer can import external JARs in the form of dependencies, specify the build package of the project, specify relations between the modules of the application and also this framework allows the user to use plugins such as a web server plugin. When a mvn clean install command is executed the framework dynamically download Java libraries and Maven plug-ins form one or more repositories, and stores them locally. Also at the same command the following goals are executed:

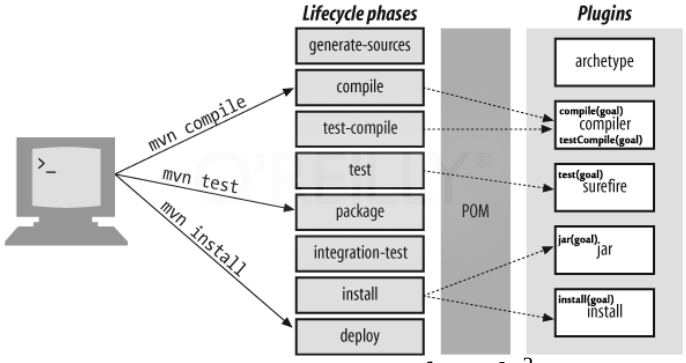

Figure 4-2 Maven Life cycle[3]

This is called the build default lifecycle and is a list of named phases that is used to build the project. These goals can be executed separately.

The framework comparable to Maven is Ant. The fundamental difference between Maven and Ant is that Maven's design regards all projects as having a certain structure and a set of supported task work-flows. While most software projects in effect support these operations and actually do have a well-defined structure, Maven requires that this structure and the operation implementation details be defined in the POM file. Thus, Maven relies on a convention on how to define projects and on the list of work-flows that are generally

[3] http://stackoverflow.com/

supported in all projects. In Ant, projects do not really exist from the tool's technical perspective. Ant works with XML build scripts defined in one or more files. It processes targets from these files and each target executes tasks. Each task performs a technical operation such as running a compiler or copying files around. Targets are executed primarily in the order given by their defined dependency on other targets. Thus, Ant is a tool that chains together targets and executes them based on inter-dependencies and other Boolean conditions.[4]

We use this framework in the Web Services Sharing System because it's a big improvement over adding dependencies in the project class path manually, and also for the easiness is provides in the build management part of the project.

4.1.3. Hibernate

Hibernate is an object-relational mapping framework for Java. The primary function of this framework is to map Java classes to database tables and mapping Java data types to SQL data types. This framework also provides the functionality of data queries generating SQL commands from Java code.

Hibernates allows the creation of queries with the help of a Hibernate Query Language, which allows creation of queries against Hibernate's data objects. Criteria query is also supported. This framework can be integrated with the help of the Spring framework.

We use this framework because of its query building functionalities. We don't have to create simple native SQL queries because Hibernate is capable of generating those by itself.

The license proof of concept will use a MySQL database server for its scalability and flexibility, high performance, high availability, strong data protection and ease of management.

4.1.4. RestFull web services

In computing, representational state transfer (REST) is an architectural style consisting of a coordinated set of components, connectors, and data elements within a distributed hypermedia system, where the focus is on component roles and a specific set of interactions between data elements rather than implementation details. Its purpose is to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability. REST is the software architectural style of the World Wide Web.[5]

RESTFul system usually communicate over Hypertext Transfer Protocol using GET, POST, PUT, DELETE. We would use this type of services for communicating between the frontend of our application to the backend with JSON. We would also use REST for the web service library dynamically created by the users of the application described in the current document.

The advantages of application built with RESTful is that they follow the following principles:

- **Resource identification through URI:** a RESTFul web service publishes a certain resource that can be identified by an URI.

[4] https://en.wikipedia.org/wiki/Apache_Maven
[5] https://en.wikipedia.org/wiki/Representational_state_transfer

- **Uniform interface:** the services are called using four basic calls. These are GET, for retrieving information from the service, PUT for creating a new resource which can be deleted using DELETE. POST is for sending data to the server.
- **Self-descriptive messages:** Information is serialized into an intermediate text format such as HTML, XML, PDF, JPEG, JSON or others.
- **Stateful interactions through hyperlinks:** Interactions with resources posted with RESTFull is stateless. The request messages do not depend on anything else than themselves. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.[6]

4.1.5. JShell

The JShell API will be used for its interactive evaluations of statements and expressions that the user will input into the web application in order to post a RESTFull web service. This API provides a rapid investigation of code, statements and expressions.

JShell is useful for this license because of its completion API which can determine when an input is incomplete or would be complete if a semicolon were added, in which case the wool will append the semicolon. The tool has a set of commands for saving and restoring work and for configurations.

This API will be used for validating the user code inputs.

4.1.6. HTML5 and AngularJS

AngularJS is a web application Javascript framework maintained by Google and by a community of developers. This framework works by reading the HTML page and searching for its custom tag attributes. These attributes are interpreted as directives to bind input or output parts of the page to a Javascript file.

Angular provides a $scope handler which binds data to a model with the help of a simple ng-model directive. Two way data binding and saving to the server now takes a small number of lines in Angular, but in jQuery would require creating your own object, and several different click and event handlers.

4.1.6.1. Angular Core Module

This module is provided by default and contains the core components of AngularJS.[7]

- **Directives:** This is the core collection of directives you would use in your template code to build an AngularJS application. Some examples include: ngClick, ngInclude, ngRepeat, etc.
- **Services/Factories:** This is the core collection of services which are used within the DI of your application. Some examples include: $compile, $http, $location, etc.

[6] http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html
[7] https://docs.angularjs.org/api

- **Filters:** The core filters available in the ng module are used to transform template data befor it is rendered within directives and expressions. Some examples include: filter, date, currency, lowercase, uppercase, etc.
- **Global APIs:** The core global API functions are attached to the angular object. These core functions are useful for low lever JavaScript operations within your application. Somme examples include: angular.copy(), angular.equals(), angular.element(), etc.

4.1.6.2. ngRoute

ngRoute is used to enable URL routing to your application. The ngRoute module supports URL management via both hashbang and HTML5 pushState. The following services are used for route management:

- $routeParams is used to access the querystring values present in the URL.
- $route is used to access the details of the route that is currently being accessed.
- $routeProvider is used to register routes for the application

4.1.6.3. ngAnimate

ngAnimate is used to enable animation features within the application. Various core ng directives will provide animation hooks into the application when ngAnimate is included. Animations are defined by using CSS transitions/animations or JavaScript callbacks.

$animate is used to trigger animation operations within directives code. Connection with CSS is done by naming structures references. Once defined, the animation can be triggered by referencind the CSS class within the HTML template code.

Using module.animation() a JavaScript animation is registered. Once registered, the animation can be triggered by referencing the CSS class within the HTML template code.

4.1.6.4. ngResource and ngCookies

ngResource module is used when querying and posing data to a REST API. The $resource service is used to define the RESTful objects which communicate with a REST API.

ngCookies module is used to handle cookie management. The following services are used for cookie management:

- $cookie service is a convenient wrapper to store simple data within browser cookies.
- $cookieStore is used to store more complex data using serialization.

4.1.6.5. AngularJS Directives

At a high level, directives are markers on a DOM elements (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler ($compile) to attach a specified behavior to that DOM element (e.g. via event listeners), or even to transformthe DOM element and its children.

Angular comes with a set of these directives built-in, like ngBind, ngModel, and ngClass. Much like you create controllers and services, you can create your own directives for Angular to use. When Angular bootstraps your application, the HTML compiler traverses the DOM matchind directives against the DOM elements.

We will use the modules described above in our system's interface. We will implement the interface using HTML5 and AngularJS, and communication with the backend will be done with the help of RESTFul web services. We have described only a few of the most important modules of AngularJS, but alot more were used.

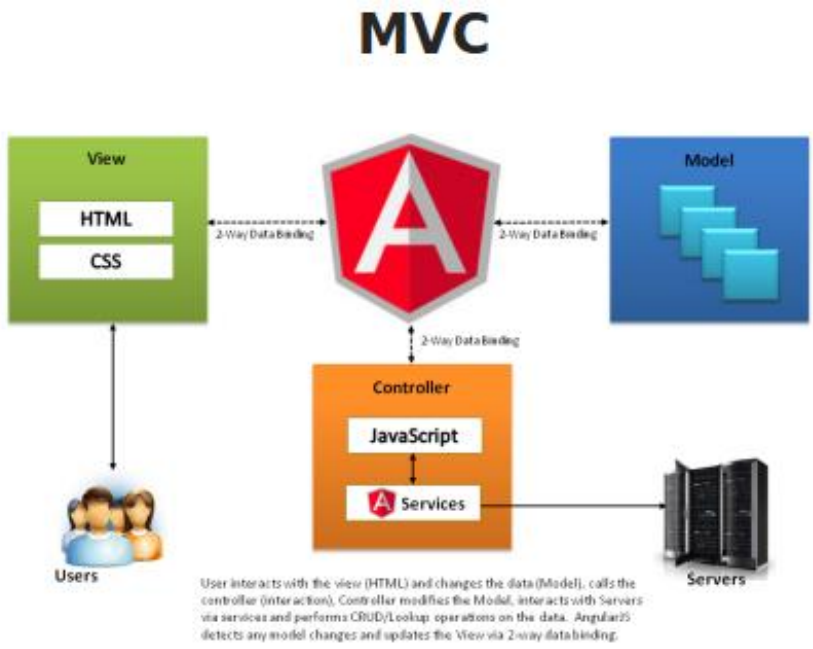

Figure 4-3 AngularJS MVC Architecture[8]

4.1.7. Apache Tomcat

Apache Tomcat[8] is an open-source web server developed by the Apache Software Foundation. Tomcat implements several JavaEE specifications including Java Servlet, JavaServer Pages, WebSocket, and provides a Java HTTP web server environment in which Java code can run.

4.1.7.1. Catalina

Catalina is Tomcat's servlet container. Catalina implements Sun Microsystems' specifications for servlet and JavaServer Pages (JSP). In Tomcat, a Realm element represents a „database" of usernames, passwords, and roles assigned to those users.

[8] https://en.wikipedia.org/wiki/Apache_Tomcat

Different implementations of Realm allow Catalina to be integrated into environments where such authentication information is already being created and maintained, and then use that information to implement Container Managed Security as described in Servlet Specification.

4.1.7.2. Jasper

Jasper is Tomcat's JSP engine. Jasper parses JSP files to compile them into Java code as servlets (that can be handled by Catalina). At runtime, Jasper detects changes to JSP files and recompiles them. Jasper 2 has the following important features:

- **JSP Tag library pooling:** Each tag markup in JSP file is handled by a tag handler class. Tag handler class objects can be pooled and reused in the whole JSP servlet.
- **Background JSP compilation:** While recompiling modified JSP Java code, the older version is still available for server requests. The older JSP servlet is deleted once the new JSP servlet has finished being recompiled.
- **Recompile JSP when included page changes:** Pages can be inserted and included into a JSP at runtime. The JSP will ont only be recompiled with JSP file changes but also with included page changes.
- **JDT Java compiler:** asper 2 can use the Eclipse JDT (Java Development Tools) Java compiler instead of And and javac

4.1.7.3. Coyote

Coyote is a Connector component for Tomcat that supports HTTP 1.1 protocol as a web server. This allows Catalina, nominally a Java Servlet or JSP container, to also act as a plain web server that servers local files as HTTP documents.

Coyote listens for incoming connections to the server on a specific TCP port and forwards the request to the Tomcat Engine to process the request and send back a response to the requesting client.

4.1.7.4. Cluster

This component has been added to manage large applications. It is used for load balancing that can be achieved through many techniques. Clustering support currently requires the JDK version 1.5 or later.

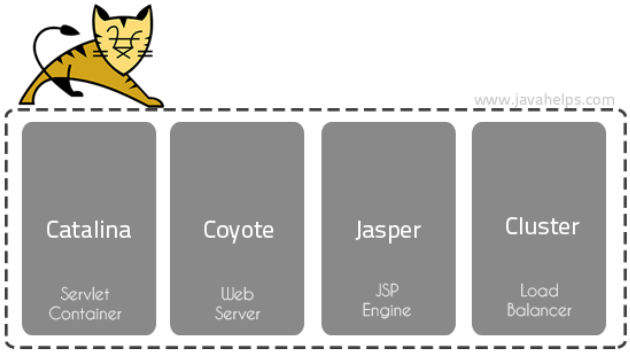

Figure 4-4 Tomcat[8] Components

4.2. System Requirements

4.2.1. Functional Requirements

The functional requirements of a system represent the functions achieved by the system or by one of it's componenets. These are described by a set of inputs, behaviour and outputs. The main activity of the Web Service Sharing System is to offer the possibility of publishing Java code in the form of a web service. Also this system will implement a questions and answers module.

FR-1	**User authentication**
FR-1.1	Any user can create an account.
FR-1.2	The users log in using a combination of username and password.
FR-1.3	The users can edit or delete their account.
FR-2	**Posting a web service**
FR-2.1	Logged in users can post a web service
FR-2.2	Logged in users can edit or delete a web service posted by them
FR-3	**Asking and answering to questions**
FR-3.1	Logged in users have the possibility of asking questions
FR-3.2	Logged in users have the possibility of answering to questions
FR-3.3	Logged in users have the possibility of attaching a web service to their answer to a question.
FR-3.4	Logged in users have the possibility of sorting and searching in a list of questions.
FR-3.5	Logged in users have the possibility of edditing questions they have asked.
FR-3.6	Logged in users have the possibility of edditing answers they had given
FR-4	**Testing a web service**
FR-3.1	Logged in users have the possibility of testing a web service
FR-3.2	Logged in users have the possibility of managing their posted services.

Table 4.1 *Funcctional Requirements*

4.2.2. Non-functional Requirements

The users have expectations about how well the system will perform. These characteristics include the usability of the system, it's speed, reliability and how does the system behave when something unexpected happens. These are the non-functional requirements:

NFR-1	**Extensibility**
NFR-1.1	Each functionality must be implemented as a web service for better modularity.
NFR-1.2	The system must obey a layered architecture
NFR-2	**Speed and performance**
NFR-2.1	The operations must be implemented using loading screens. One web service publish must take less than 30 seconds.

NFR-3	**Robustness**
NFR-3.1	The application must warn the users in case of a failure through suggestive messages.
NFR-4	**Security**
NFR-4.1	The passwords stored in the database must be hashed.
NFR-4.2	Communication between the client and the server must be crypted.
NFR-5	**Reusability**
NFR-5.1	The services posted with the help of this tool must be usable within other applications.

Table 4.2 *Non-functional requirements*

4.3. Use cases

In the following chapter we are going to present the use cases of the application describing the primary actor of the use case, the description, stakeholders and interests, basic flow, alternative flows, preconditions, post conditions and extension points for each of them.

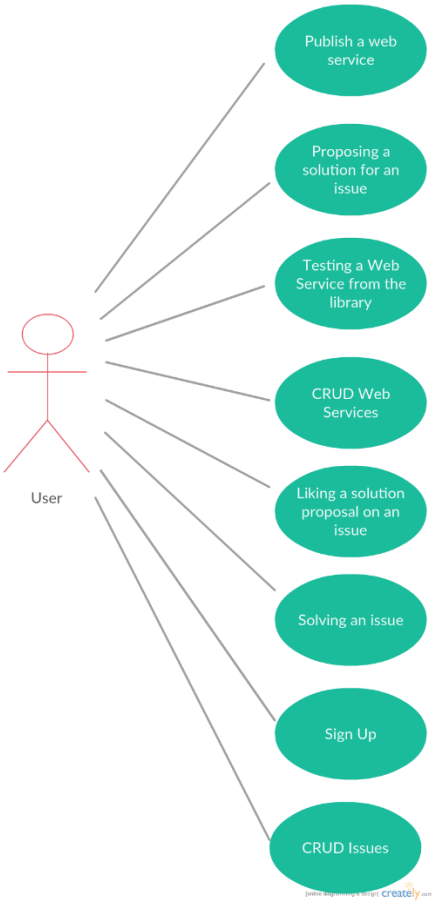

Figure 4.5 Use case Diagram

4.3.1. Publishing a web service

Primary Actor: Normal User
Description: The purpose of this application is to create and maintain a web services library. This use case will describe the flow for publishing a new web service. The user has to have knowledge of the Java language in order to carry this operation.
Stakeholders and interests: Java Developer – interested in having a user friendly page for the process of publishing a web service in order to share and use them from other applications.
Basic Flow:

1. The user clicks on the "Publish!" button.
2. The user inputs the Maven dependencies required to run the code.
3. The user inputs the imports required for compiling the service's code.
4. The user inputs other service details like the name, the return type, the number of parameters, their type and name.
5. The user fills the web service body.
6. The user pushes the button "Publish!"

6.1. The input data is not correct. The user goes back to step 2.

7. The system returns a success message.

Alternative Flows:

1. Cancel publish operation
This flow can occur in each of the steps 1, 2, 3, 4 and 5.

1.1 The actor selects either the "Home" page or the "Library" page or clicks the button "Log out".

Special requirements

1. The system has to show clearly the errors in the case in which the web service is not successfully deployed.

Preconditions:

1. The actor has a web browser installed on his machine.
2. The actor has an account for using this application.

Postconditions:

1. The web service published by the user must be online and available for use.

Extension Points:

1. Search if there's a resemblance between the service posted by the user and the other services in the library.

4.3.2. Testing a web service from the library

Primary Actor: Normal User
Description: The purpose of this application is to create and maintain a web services library. This use case will describe the flow testing the functioning of a deployed service from the library.
Stakeholders and interests:
Java Developer – interested in having a user friendly page for testing a web service before using it in a java application.

Basic Flow:

1. The user clicks on the “Library” button.
2. The user selects the desired web service from the list.
3. The user inputs the required parameters.
4. The user hits the button “Test”.
5. The system returns a result by calling the published web service.

Alternative Flows:

-

Special requirements:

1. The system has to generate front end elements so that the user can input the desired values to test the web service.

Preconditions:

2. The actor has a web browser installed on his machine.

3. The actor has an account for using this application.

Postconditions:

1. The web interface will show the response of the web service or an error message in the form of an alert.

Extension Points

-

4.3.3. Creating an issue

Primary Actor

Normal User

Description

The application has an issue tracking system. The user who needs to use a certain web service (algorithm, procedure, etc.) creates an issue. The issues will be visible on the home page after the users log in and they can propose and publish solutions by commenting and publishing their solution to that certain problem.

Stakeholders and interests

Java Developer – interested in having a user friendly page for posting a problem that he has to be solved with the help of the other users which use this application.

Basic Flow

1. The user clicks on the “Issues” button.
2. The user clicks on the button “New”.
3. The user fills the issue input fields.
4. The user hits the button “Create Issue”.

Alternative Flows

1 Cancel issue creation operation

This flow can occur in each of the steps 1, 2 and 3.

1.1 The actor selects either the “Home” page or the “Library” page or clicks the button “Log out”.

Special requirements

-

Preconditions

1. The actor has a web browser installed on his machine.
2. The actor has an account for using this application.

Postconditions

1. The issue has to be in the issues list in the tab “Issues”.

Extension Points

1. The application can be modified in such a way that the user would be able to modify and edit the fonts and the styling of the answer. This inputs can be stored in the database as a HTML code and posted on the interface accordingly.

4.3.4. Solving an issue.

Primary Actor

Normal User

Description

The application has an issue tracking system. The user who created a new issue, if he finds that another user has posted a web service that would solve his problem can close the issue.

Stakeholders and interests

Java Developer – interested in solving the issue that he has posted.

Basic Flow

1. The user clicks on the “Issues” button.
2. The user clicks on the button “My issues”.
3. The user clicks on one of the issues in the list.
4. The user presses on the “Close issue!” button

Alternative Flows

-

Special requirements

-

Preconditions

3. The actor has a web browser installed on his machine.
4. The actor has an account for using this application.

Postconditions

1. The issue has to be in the issues list in the tab “Issues”.

Extension Points

-

4.3.5. Proposing a solution for an issue.

Primary Actor

Normal User

Description

The application has an issue tracking system. When logging into the application, on the main page the user will see a list of all posted issues. He can click on one of them and post a comment (proposed solution).

Stakeholders and interests

Java Developer – interested in answering a question posted by other users.

Basic Flow

1. The user clicks on the “Issues” button.
2. The user selects one of the issues in the list.
3. The user writes a comment in the “Comment” text area.
4. The user presses on the Post!

Alternative Flows

-

Special requirements

-

Preconditions

5. The actor has a web browser installed on his machine.
6. The actor has an account for using this application.

Postconditions

1. The comment that he posted has to be in the list of comments on that particular issue.

Extension Points

-

4.3.6. Liking a solution proposal on an issue.

Primary Actor
Normal User

Description
The application has an issue tracking system. The user can view an issue and the answers posted by the other users on that issue. He has the option of liking or disliking one of the answers.

Stakeholders and interests
Java Developer – interested in helping the users in the community to find the best answer in the comment section.

Basic Flow
1. The user clicks on the "Issues" button.
2. The user selects one of the issues in the list.
3. The user reads the posted comments on that certain issue.
4. The user presses like on the answer he views as being good.

Alternative Flows
-

Special requirements
-

Preconditions
7. The actor has a web browser installed on his machine.
8. The actor has an account for using this application.

Postconditions
1. The comment number of likes is incremented by 1.

Extension Points
-

4.3.7. Sign up

Primary Actor
Normal User

Description

For a user to be able use the application he needs a valid account. This use case will describe the process of signing up.

Stakeholders and interests
Java Developer – interested in creating a new account.

Basic Flow
1. The user clicks on the “Sign up” button in the corner of the screen.
2. The user fills his data in the form.
3. The user clicks on the “Sign up” button.

Alternative Flows
1. User or password input error
   This flow can occur in step 3.

   1.1 The actor is redirected to step 2.

Special requirements
-

Preconditions
1 The actor has a web browser installed on his machine.
2 The actor has an account for using this application.

Postconditions
1. The user can log in with the username and password that he has provided

Extension Points
-

4.3.8. Edit web service

Primary Actor
Normal User

Description
The users can edit the web services posted by them.

Stakeholders and interests
Java Developer – interested in improving a web service that he posted.

Basic Flow
1. The user clicks on the “Library” button.

2. The user selects the web service he wants to edit.
3. The user makes the modifications to that web service.
4. The user clicks on the button “Post”.

Alternative Flows

1 Cancel publish operation
This flow can occur in each of the steps 1, 2, 3, 4 and 5.

1.1 The actor selects either the “Home” page or the “Library” page or clicks the button “Log out”.

Special requirements

-

Preconditions

2 The actor has a web browser installed on his machine.
3 The actor has an account for using this application.

Postconditions

1. The web service data is changed. The web service has the expected behaviour

Extension Points

-

4.3.9. Edit comment

Primary Actor

Normal User

Description

The users can edit the comments posted by them.

Stakeholders and interests

Java Developer – interested in improving the response that he posted.

Basic Flow

1. The user clicks on the “Issues” button.
2. The user selects one of the issues in the list.
3. The user clicks on the “Edit” button of one of the comments.
4. The user changes the text of the comment.
5. The user pushes the button “Post”.

Alternative Flows

-

Special requirements

-

Preconditions

4 The actor has a web browser installed on his machine.
5 The actor has an account for using this application.

Postconditions

1. The comment posted by the user is changed.

Extension Points

-

4.3.10. Edit question

Primary Actor

Normal User

Description

The users can edit the questions posted by them.

Stakeholders and interests

Java Developer – interested in improving the question that he posted.

Basic Flow

1. The user clicks on the “Issues” button.
2. The user selects one of the issues in the list.
3. The user clicks on the “Edit” button of the question.
4. The user changes the text or title of the question.
5. The user pushes the button “Ask!”

Alternative Flows

-

Special requirements

-

Preconditions

1 The actor has a web browser installed on his machine.
2 The actor has an account for using this application.

Postconditions

1. The question posted by the user is changed.

Extension Points

-

4.3.11. Delete question, comment or web service

Primary Actor

Normal User

Description

The users can delete the questions, comments or web services posted by them.

Stakeholders and interests

Java Developer – interested deleting a question, comment or web service that he posted.

Basic Flow

1. The user goes on to the Issues page or into the Library of web services.
2. The user selects one of the items in the list.
3. The user clicks a Delete button for the comment, web service or question.

Alternative Flows

-

Special requirements

-

Preconditions

1 The actor has a web browser installed on his machine.
2 The actor has an account for using this application.

Postconditions

1. The question, comment or web service is deleted.

Extension Points

-

These are all the use cases to be implemented by the Web Service Sharing System. Next we will take a look on the implemented system, we are going to study the architecture of the system, the design of the database and we are going to look over the code implementation.

Chapter 5. Detailed Design and Implementation

In this chapter we will document the developed system with its architecture, and code implementation. We will look into the database model and design and we will explain in detail the modules of the system.

5.1. System Architecture

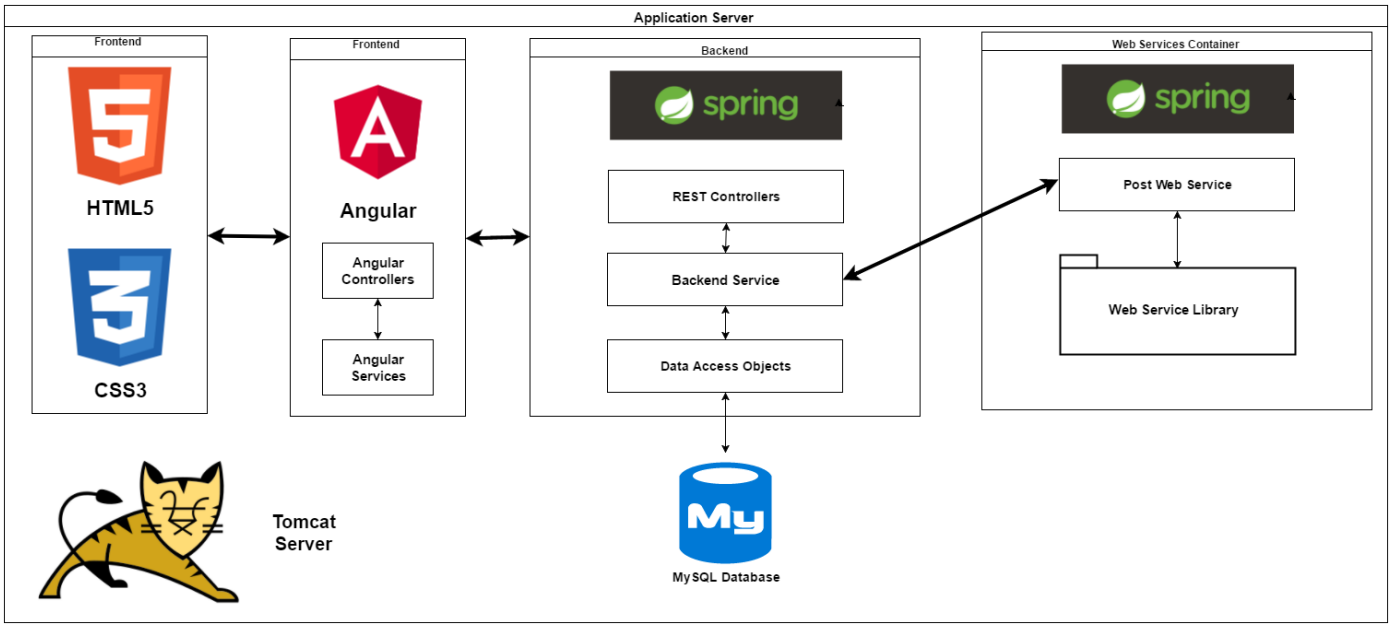

Figure 5-1 Web Service Sharing System Architecture

The Web Service Sharing System is designed based on the principle of Layers Architecture. We implemented the system using HTML5, CSS3 and AngularJS framework in the frontend. From the AngularJS service Javascript files we make calls to the REST controllers in the backend. The components in the backend are Spring managed components.

From the Rest controllers calls are made to the Backend Services Layer which contain backend services and also expose the database resources. Also from the backend service layer we have access to the Web Services Container module which holds the published web services. This module is deployed on the Tomcat server as another war, and when we wish to update the repository with new web services, then the war is rebuilt and redeployed.

5.2. Files structure and packages

The Web Service sharing system is organized into two maven modules, licenta-application and licenta-web-services, each packaged in different web archives when

building the application. These two modules have as parent the module named licenta-suite used for easy import of the project.

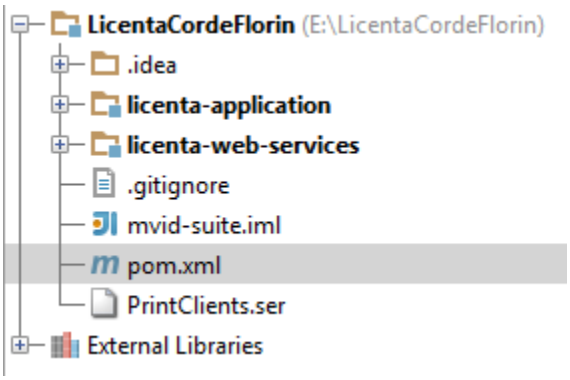

Figure 5-2 Web Service Sharing System Maven modules

The licenta-suite module specifies in its pom.xml file its subordinate modules and other configuration details:

```
<groupId>org.cf.card</groupId>
<artifactId>licenta-suite</artifactId>
<version>1.0-SNAPSHOT</version>
<name>Licenta Suite</name>
<url>http://maven.apache.org</url>
<packaging>pom</packaging>

<modules>
    <module>licenta-web-services</module>
    <module>licenta-application</module>
</modules>
```

5.2.1. Module licenta-application

In this module we have the main web application with all of its features of Q&A functionality and with the web service publishing functionality and interface.

In the directory src/main we have three main directories: Java, resources and webapp.

In the **java** directory we have stored our Java sources for the backend of the Web Service Sharing system. This directory contains the following packages: configuration, controller, dao, exceptions, model and service.

In the configuration package we placed the configuration classes for Spring. In these classes we can find configurations in regard to viewResolvers, component scans for spring, resource handler directory, session factory configurations, data source configurations, hibernate configurations and entity manager configurations.

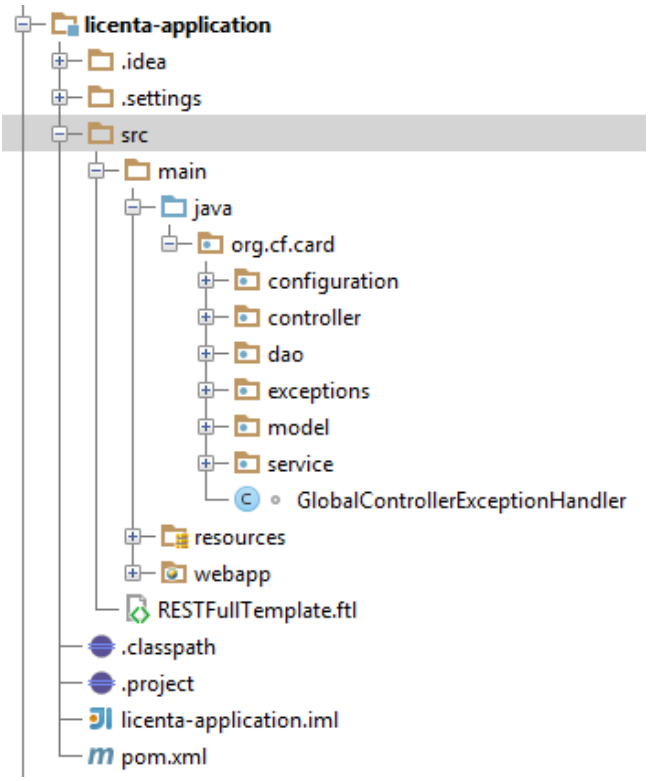

Figure 5-3 The Java directory package structure

In the package controller one can find the REST controllers for the main web application. In the package dao one can find the data access objects, the package exceptions is destined to store the custom exceptions that we have created, package model holds the POJOs of the application, and the package service represents the Service layer.

In the **resources** directory we have stored the properties file for configuring the data base connection like url, username and password, hibernate dialect and hibernate database creation parameter.

In the **webapp** we can find two important folders, **static** and **WEB-INF**.

The **static** directory contains the assets folder which corresponds to the template used for the interface of this application, the css folder with the main css files used, the images directory where we store all the immages used in the interface and a js folder.

In the js folder we have placed the AngularJS controllers and service filles. The controller files deals with tying the dom to the js files and handling button and navigation events. In the service folder we can find the service files designed only to provide the services of calling the REST apis posted by the Java backend.

The WEB-INF folder contains the jsp views of the application.

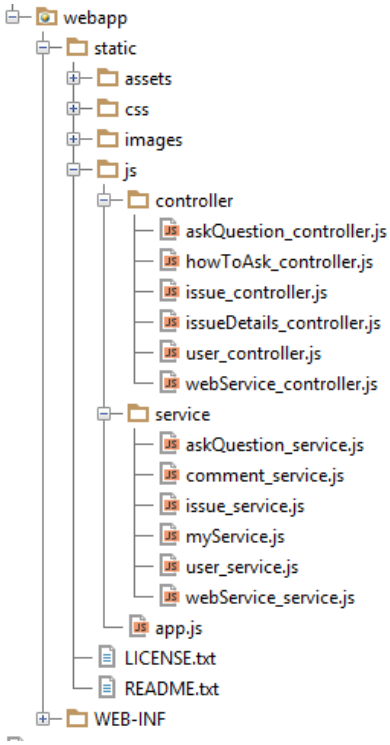

Figure 5-4 Webapp directory structure

5.2.2. Module licenta-web-services

This module is used for storing the web services posted by the users. It is configured also with Spring, but Spring Boot is not used here, so the configuration files for Spring are placed in the resources directory. This module is simpler and it contains only the controller package. These Java classes in these package are REST controllers and are filled with data from the user interface using the RESTFullTemplate.ftl file. We will explain the behaviour of this scenario in the following subchapters.

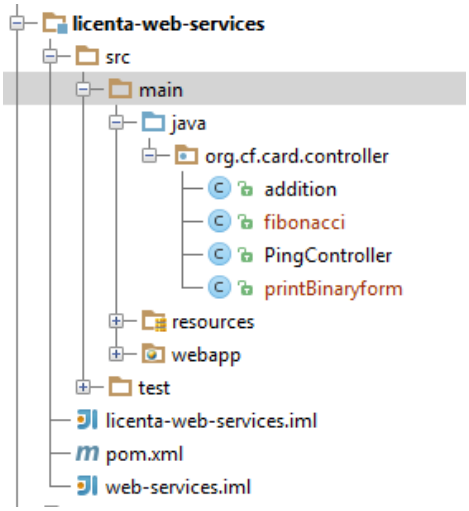

Figure 5-5 licenta-web-services module file structure

5.3. Publishing a web service scenario implementation

In the Web Service Sharing System publishing a web service can be done in two ways: as an answer to a question posted by a user, even by himself, or as a standalone web service with no connection to the questions and answers part of the system.

The web service publishing page will give the user the option of adding a Maven dependency, and to add extra imports. The mandatory fields on this page will be the web service name textfield, the number of arguments, the name and type of the arguments, the service's return type and the web service's code.

These components will be marked with the attribute "ng-model" which binds the input fields to a property on the scope. These fields then can be used in AngularJS controllers. In this case, we are dealing with the page RestServiceForm.jsp in the views directory, webService_controller.js from the controllers directory and webService_service.js from the services directory.

When the "Post!" button is pushed on the Web Service Publish page the action is handled by the createWebService function from webService_controller.js:

```
self.createWebService = function(webService){
        $scope.loading = true;
        WebServiceService.createWebService(webService)
            .success(function(data, status, headers, config){
                $scope.loading = false;
            })
            .error(function(data, status, headers, config){
                $scope.loading = false;
                alert("Publish fail! Recheck your code!")
            });
    };
```

Here a $scope.loading variable is used for displaying the loading screen. The functions success and error are called after the response from the REST web service has arrived.

The function makes a call to the createWebService method from webService_service.js file. This function just returns the response from a HTTP POST.

```
createWebService: function(webService){
            return $http.post('http://localhost:8092/licenta-
application/webService/', webService)
        }
```

This HTTP POST calls the createWebService method from the WebServicesController located in the controller package in the backend. The Rest controllers, the backend services and the data access objects are Spring managed components, so we don't have to instantiate new objects like WebServiceService, because of Spring's container.

```java
@RestController
public class WebServicesController {

    @Autowired
    public WebServiceService webServiceService;  //Service which will do all data retrieval/manipulation work

    @RequestMapping(value = "/webService/", method = RequestMethod.POST, consumes = APPLICATION_JSON_VALUE)
    public @ResponseBody ResponseEntity<Void> createWebService(@RequestBody WebService service) throws Exception {
        System.out.println("Creating a web service :" + service.getCode() + ", " + service);

        if (webServiceService.isWebServiceExistent(service)) {
            System.out.println("A Web Service with the name " + service.getCode() + " already exists");
            return new ResponseEntity<Void>(HttpStatus.CONFLICT);
        }
        webServiceService.processWebService(service);
        return new ResponseEntity<Void>(HttpStatus.OK);
    }
}
```

Figure 5-6 WebServiceController class

In the Web Service Sharing System we work with ResponseEntity to send the status of the operation of the REST service. The communication between the server and the client is done through JSON.

From this point the application calls the processWebService forom the WebServiceService.java class. This is the most important class in the entire project because it handles creating a REST controller class from a template and manages building the war for the licenta-web-service module and publishes it. We will describe the flow of this operation next.

First the processWebService method looks like this:

```java
public void processWebService(WebService webService) throws ServicePublishFailedException {
    Configuration cfg = new Configuration();
    try {
        Template template = cfg.getTemplate(SERVICE_TEMPLATE);
        Map<String, Object> webServiceBody = new HashMap<~>();
        buildWebService(webService, webServiceBody);
        writeToFile(template, webServiceBody, webService.getWebServiceName());
        executeMavenCommands(webService);
    } catch (IOException e) {
        e.printStackTrace();
    } catch (TemplateException e) {
        e.printStackTrace();
    } catch (MavenInvocationException e) {
        e.printStackTrace();
    }
}
```

Figure 5-7 processWebService method from WebServiceService.java

First the template configuration is instantiated. For generating the web services we use the freemarker api. This allows the user to create templates with the extension .ftl, and to fill them with the help of jsf like sintax: “${variable}”. Our created template is under /src and it’s called RESTFullTemplate.ftl.

```
package org.cf.card.controller;
${imports}
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import static org.springframework.web.bind.annotation.RequestMethod.GET;
import static org.springframework.http.MediaType.APPLICATION_JSON_VALUE;

@RestController
@RequestMapping(value = "/${url}")
public class ${className} {

    @RequestMapping(method = GET, produces = APPLICATION_JSON_VALUE)
    public @ResponseBody int ${methodName}(${arguments}) {
        ${methodBody}
    }
}
```

Figure 5-8 Template RESTFullTemplate.ftl

After loading the template, the web service file is built with the help of the method buildWebService from the WebServiceService.java class.

```
private void buildWebService(WebService webService, Map<String, Object> webServiceBody) {
    webServiceBody.put("imports", webService.getImports());
    webServiceBody.put("url", webService.getWebServiceName());
    webServiceBody.put("className", webService.getWebServiceName());
    webServiceBody.put("methodName", webService.getWebServiceName());
    String arguments = "";
    for (int i = 0; i < Integer.parseInt(webService.getNumberOfArguments()); i++)
        if (i == 0)
            arguments = arguments + "@RequestParam " + webService.getArguments().get(i).getType() +
        else
            arguments = arguments + ", @RequestParam " + webService.getArguments().get(i).getType()
    webServiceBody.put("arguments", arguments);
    webServiceBody.put("methodBody", webService.getCode());
}
```

Figure 5-9 buildWebService method from WebServiceService.java

At this point in time the web service body is built and we just need to copy the body in a java file, and then copy the java file into the licenta-web-services module in the controller package. This operation is done by the writeToFile method in the class WebServiceService.java. At this point in time the details of the web service inserted by the user in the user interface are written in a Java class in the appropriate places, and that Java class is copied in the sources of the licenta-web-services module. Now we have to build the war file of the licenta-web-services module and deploy it on the Tomcat server.

For managin this operations the Maven framework was used through a maven invoker API. This maven invoker API is used to „clean” the project, which means deleting

the target directory from the project's tree, and the „install" command generates the new web archive which will have the newly created web service.

```
private void executeMavenCommands(WebService webService) throws MavenInvocationException, ServicePublishFailedExcept
    InvocationRequest request = new DefaultInvocationRequest();
    request.setPomFile(new File(POM_WEB_SERVICES_MODULE));
    request.setGoals(Arrays.asList(MAVEN_COMPILE));

    Invoker invoker = new DefaultInvoker();
    InvocationResult result = invoker.execute(request);

    if (result.getExitCode() != 0) {
        String filepath = CONTROLLER_PATH + webService.getWebServiceName() + ".java";
        Path path = FileSystems.getDefault().getPath(filepath);
        try {
            Files.delete(path);
        } catch (NoSuchFileException x) {
            System.err.format("%s: no such" + " file or directory%n", path);
        } catch (DirectoryNotEmptyException x) {
            System.err.format("%s not empty%n", path);
        } catch (IOException x) {
            System.err.println(x);
        }
        throw new ServicePublishFailedException("Publish failed!");
    } else {

        request.setGoals(Arrays.asList("clean", "install"));
        invoker.execute(request);

        String command = TOMCAT_PATH +"bin\\shutdown.bat";
        try {
            Process child = Runtime.getRuntime().exec(command);
        } catch (IOException e) {
            e.printStackTrace();
        }
        File file = new File(TARGET_FILE_PATH +"licenta-web-services.war");
        File destinationDir = new File(TOMCAT_PATH+"webapps");

        try {
            FileUtils.copyFileToDirectory(file, destinationDir);
        } catch (IOException e) {
            e.printStackTrace();
        }
        command = TOMCAT_PATH+"bin\\startup.bat";
        try {
            Process child = Runtime.getRuntime().exec(command);
        } catch (IOException e) {
            e.printStackTrace();
        }
        WebService createdWebService = createWebService(webService);
        for (int i = 0; i < Integer.parseInt(webService.getNumberOfArguments()); i++) {
            webService.getArguments().get(i).setWebService(createdWebService);
            webServiceArgument.createArgument(webService.getArguments().get(i));
        }
    }
```

Figure 5-10 executeMavenCommands method from WebServiceService.java

First we have to compile the project that we are trying to build. For this we use also Mave, with it's „compile" goal. We execute the compilation and we check the response. If the response is different from 0 then there's a compilation error in the source code that the user has imputted. In this case we have to delete his created service from the sources of

the module licenta-web-services. After doing that we throw an exception which will be displayed on the interface for the user to recheck his code and to try publishing the web service again.

In the case of success, we use again the maven invoker with the path of the pom.xml from the module licenta-web-services and execute the „clean" and „install" goals. These goals should execute without many problems because the code has allready been compiled. After the execution of these two goals we have to redeploy the war on the Tomcat server. Here a shutdown command is executed at the path of the server, then the web service created by the user is coppyed at the proper path on the Tomcat server, and after this operation the server is started again. This last few steps can be avoided if the implementation would have used ClassLoader. Also, the same server can be used for the two modules, but we chose this approach because of the modularity that it provides.

If everything goes to plan we persist the data of the web service in the database. For this operation we use the method createWebService from the class WebServiceService.java along with the bean WebServiceDAO.

```
@Inject
private WebServiceDAO webServiceDAO;
@Autowired
public WebServiceArgumentService webServiceArgument;

public WebService createWebService(WebService webService) {
    return webServiceDAO.save(webService);
}
```

Figure 5-11 createWebService method from the class WebServiceSeervice.java

In the data access objects we used the extension of CrudRepository. This allows us to avoid building querries of any form.

```
@Transactional
@Repository
public interface WebServiceDAO extends CrudRepository<WebService, Long> {

    public WebService findById(Long id);

    public WebService findByCodeLike(String code);

    public WebService findByWebServiceNameLike(String Name);

}
```

Figure 5-12 WebServiceDAO.java

This approach allows the programmer to specify method names like the ones above and the Spring framework will generate the SQL for him. Also all the CRUD operations are predefined.

At this stage the web services should be up and running and the details about the web service persisted in the database. We will now show a sequence diagram for the scenario presented above.

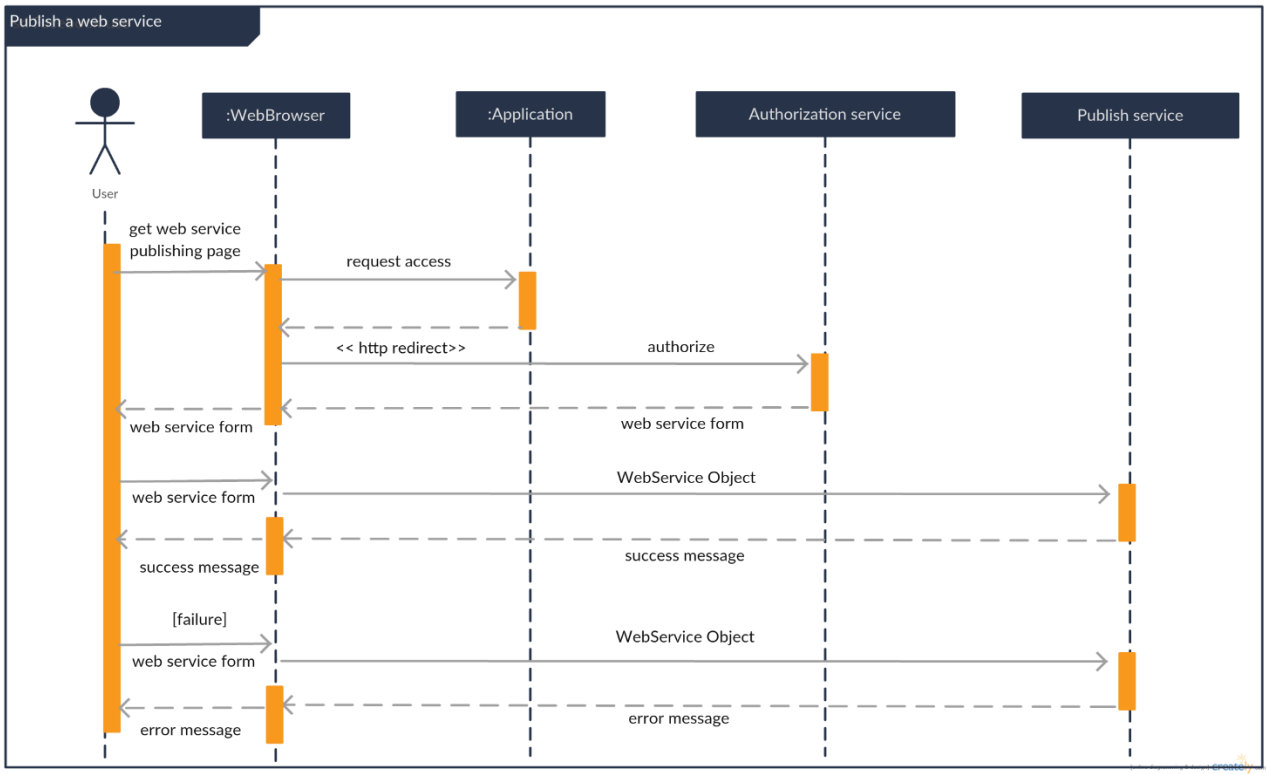

Figure 5-13 Publish a web service sequence diagram

The APIs and frameworks used in this scenario are imported using Maven dependencies. For example, for the maven invoker and the freemaker template api the following dependencies were declared in the pom.xml file of the licenta-application module:

```
<dependency>
    <groupId>org.freemarker</groupId>
    <artifactId>freemarker</artifactId>
    <version>2.3.20</version>
</dependency>

<dependency>
    <groupId>org.apache.maven.shared</groupId>
    <artifactId>maven-invoker</artifactId>
    <version>2.2</version>
</dependency>
```

Figure 5-14 Maven Dependencies syntax

Also for the licenta-appplication module we use two plugins, a tomcat plugin and a maven war plugin. This is the easyest way to build the web archives from the sources of the project and also to deploy the web archive. The plugins were specified in the following way:

```
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.4</version>
                <configuration>
                    <warSourceDirectory>src/main/webapp</warSourceDirectory>
                    <warName>licenta-application</warName>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>

    <finalName>licenta-application</finalName>

    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.2</version>
            <configuration>
                <port>8092</port>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Figure 5-15 Maven tomcat and .war plugin

5.4. Asking a question scenario implementation

This scenario is part of the questions and answers part of this system. When logged in, the user will have the option of asking a question by clicking on a button in the header of the page. This button will redirect the user to a page called the “How to ask” page.

In this “How to ask” page the user will have to read some useful advice about how should he organize the content of his question, what to look for in the responses that he gets, and how should he ask the question in such a way that the other people in the community can benefit from the question posted by him. This page will have a validation

procedure, so that the user can go to the next page only if he checks a checkbox which will ask for confirmation that he read that page.

At this point the user will be redirected to the page in which he will have to fill the content of his question. The question will have a title and a body. The title is meant for the actual question, and the body is for further explications, the research that he has done on that subject, the code that does not work, or other details. After the user has finished specifying these details he can proceed by clicking a confirmation button. This leads him to the question page which will contain his question along with some details about when he has asked that question, how many views has he got, etc., and the list of comments which is, in this case empty, because he has just posted the question.

First when submitting the question the AngularJS controller is called and the submit function is executed.

```
'use strict';

App.controller('AskQuestionController', ['$scope', 'AskQuestionService', 'myService',
    var self = this;
    self.issue = {id: null, title: '', description: ''};

    self.submit = function () {
        AskQuestionService.createIssue(self.issue)
            .success(function (data, status, headers, config) {
                window.localStorage['issueId'] = angular.toJson(data.id);
                document.location.href = "Issue";
            })
            .error(function (data, status, headers, config) {
                alert("Fail!")
            });
    };
}]);
```

Figure 5-16 AskQuestionController.js

Here in the case of success we redirect the page to the question answering page with the question we have just posted. From here the AskQuestionService file is called and a http POST message is sent to the backend through the following function:

```
'use strict';

App.factory('AskQuestionService', ['$http', '$q', function($http, $q){

    return {

        createIssue: function(issue){
            return $http.post('http://localhost:8092/licenta-application/issue/', issue)
        }
    };
}]);
```

Figure 5-17 AskQuestionService.js

From here the IssueController is called. And from here on the flow is like the flow described in the previous section, the part where the data is persisted in the database.

```
@Autowired
public IssueService issueService;

@RequestMapping(value = "/issue/", method = RequestMethod.POST, consumes = APPLICATION_JSON_VALUE)
public @ResponseBody
ResponseEntity<Issue> createIssue(@RequestBody Issue issue) throws Exception {
    Issue createdIssue = issueService.createIssue(issue);
    return new ResponseEntity<Issue>(createdIssue, HttpStatus.OK);
}
```

Figure 5-18 IssueController.java method createIssue

In this class we also have implemented a GET mapping for the redirect we are doing on question's page in case of success.

```
@RequestMapping(value = "/issue/{id}", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<Issue> getIssue(@PathVariable("id") long id) {
    Issue issue = issueService.findById(id);
    if (issue == null) {
        return new ResponseEntity<Issue>(HttpStatus.NOT_FOUND);
    }
    return new ResponseEntity<Issue>(issue, HttpStatus.OK);
}
```

Figure 5-19 IssueController.java method getIssue

Again the data is persisted in the database with the help of querries created by Spring. The other scenarios use the same architecture and the same layers so there's no need of presenting them in great detail.

5.5. Database design and model

For the database we used a MySQL server. The database is composed out of 5 tables: comment, issue, user, webservice and webserviceargument.

The webservice table is used to store information about the posted web services from the library. It has as a primary key the field webservice_id, and fields code, imports, mavenDependencies, numberOfArguments, returnType, webServiceName and a user_id foreign key.

The webserviceargument table stores information about the arguments of the posted web services. It has a webserviceargument_id primary key, and the fields name and type, and a webservice_id foreign key.

The table comment is used for storring comments on questions from the Questions and Answers module of the system. It has as a primary key the fields comment_id, and the fields commentText, numberOfLikes, postTime, and the foreign keys issue_id, user_id and webService_webservice_id.

The table issue is used for storing the questions of the users from the Questions and Answers module of the system. It has as a primary key the field issue_id and the fields description, postTime, solved, title and viewsNumber and the foreign key user_id.

And finally the table user stores information about the users of the system. It contains a primary key user_id, and the fields address, email, firstName, lastName, password, type and username.

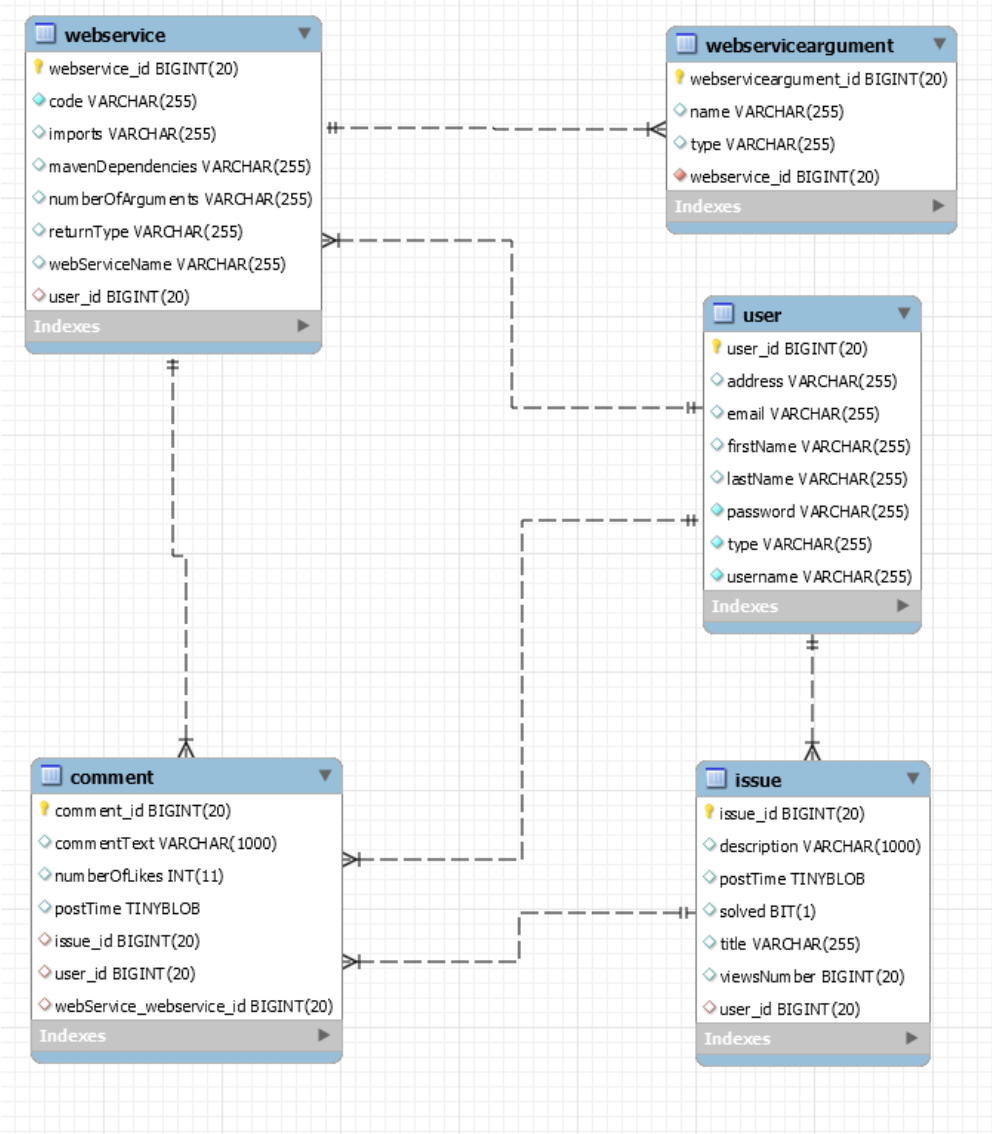

Figure 5-20 Database schema

Chapter 6. Testing and Validation

The Web Services Sharing system is tested both by manual tests and by automated testing. We will describe in the next section both the implemented automated testing and the manual testing that has been done to assure the functioning correctness of this system.

6.1. Automated Testing

For testing the web services which expose the database resources we had created automated unit tests using the Junit framework. This framework is configured with the help of the Spring framework.

```
public class WebServiceTests extends BaseTest{

    @Autowired
    private WebServiceService webServiceService;

    private WebService constructWebService() {

        WebService webService = new WebService();
        webService.setCode("return 0;");
        webService.setImports("");
        webService.setMavenDependencies("");
        webService.setNumberOfArguments(String.valueOf(0));
        webService.setReturnType("int");
        webService.setWebServiceName("testName");
        return webService;

    }
    @Test
    public void addWebServiceTest() {
        WebService expected = constructWebService();
        WebService received = webServiceService.createWebService(expected);
        assertThat(received, is(expected));
    }
}
```

Figure 6-1 addWebServiceTest case

We used test cases like the one above. This test case test the insertion mechanism in the database with the help of assertThat Junit method to check the value transmitted back from the database. All the test cases are executed with the help of Maven after the compilation is done when trying to build the web archives from the sources of the project.

All the test cases work as expected.

6.2. Manual Testing

TC1.
Title: Publishing a web service
Description: The user has to be able to post a web service.
Scenario 1:
Step 1: The user navigates to the page "Publish a web service".
Step 2: The user inputs the data for the web service, but the imputed code has an error.
Step 3: The user pushes the "Post!" button.
Expected result: The system displays a compilation error message.

Scenario 2:
Step 1: The user navigates to the page "Publish a web service".
Step 2: The user inputs the data for the web service, but the imputed title is already existent in the database.
Step 3: The user pushes the "Post!" button.
Expected result: The system displays a web service already existent error message.

Scenario 3:
Step 1: The user navigates to the page "Publish a web service".
Step 2: The user inputs the data for the web service correctly.
Step 3: The user pushes the "Post!" button.
Expected result: The system displays a web success message.

TC2.
Title: Asking a question
Description: The user has to be able to post a question.
Scenario 1:
Step 1: The user navigates to the page "Ask a question".
Step 2: The user inputs only the body of the question.
Step 3: The user pushes the "Ask!" button.
Expected result: The system displays a warning message for the title text field.

Scenario 2:
Step 1: The user navigates to the page "Ask a question".
Step 2: The user inputs only the title and the body of the question.
Step 3: The user pushes the "Ask!" button.
Expected result: The system redirects the user to the "Question Details" page with the question that he asked on the top of the page.

TC3.
Title: Posting a comment
Description: The user has to be able to post a comment on any of the posted questions.
Scenario 1:
Step 1: The user navigates to the page "Question Details".

Step 2: The user navigates on the bottom of the page and pushes the "Post!" button
Expected result: The system displays a warning message for the empty comment text field.

Scenario 2:
Step 1: The user navigates to the page "Question Details".
Step 2: The user writes the comment he wants to post in the comment textbox.
Step 3: The user pushes the "Ask!" button.
Expected result: The system refreshes the comments section of the page and makes the user's comment visible.

TC4.
Title: Posting a web service attached to a comment
Description: The user has to be able to post a web service as an attachment to a comment.
Scenario 1:
Step 1: The user navigates to the page "Question Details".
Step 2: The user navigates on the bottom of the page and pushes the "Publish!" button on one of the comments that he posted.
Step 3: The user executes the successful scenario from TC1.
Expected result: The system attaches a web service to the comment. This can be seen by all the users of the system.

Scenario 2:
Step 1: The user navigates to the page "Question Details".
Step 2: The user navigates on the bottom of the page and pushes the "Publish!" button on one of the comments that he posted.
Step 3: The user executes one of the unsuccessful scenarios from TC1.
Expected result: The system does not attach a web service to the comment.

TC5.
Title: Testing a web service from the library
Description: The user has to be able to test any web service published by any user.
Scenario 1:
Step 1: The user navigates to the page "Library".
Step 2: The user selects one of the web services from the list.
Step 3: The user clicks on the "Test" button.
Step 4: The user inputs the wrong type of inputs.
Expected result: The system displays an error message with incorrect inputs.

Scenario 2:
Step 1: The user navigates to the page "Library".
Step 2: The user selects one of the web services from the list.
Step 3: The user clicks on the "Test" button.
Step 4: The user inputs correct data.
Expected result: The system displays the response of the web service.

TC6.
Title: Liking a comment
Description: The user has to be able to give likes to comments.
Scenario 1:

Step 1: The user navigates to the page "Question Details".
Step 2: The user clicks on the thumbs up or thumbs down on any posted comment.
Expected result: The system increases or decreases the number of likes by one.

Chapter 7. User manual

In this section I will describe details about the project installation, the hardware ans software resources needed for installing and running the application, and a step by step description of how this application can be deployed.

Next, in the user manual, we will describe how to use the application with no inside technical information.

7.1. Project Installation

Before deploying the project a few parameters have to be adjusted first in the file persistence-mysql.properties. In this file the administrator has to specify the jdbc driver class name, the url of the database, the username and the password. Also the user has to specify in the same file the path of the server that he will use to deploy the application.

```
# jdbc.X
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/licence_database?createDatabaseIfNotExist=true
jdbc.user=root
jdbc.pass=

# hibernate.X
hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
hibernate.show_sql=false
hibernate.hbm2ddl.auto=create
tomcat.path="C://tomcat"
```

Figure 7.1 persistence-mysql.properties file

After these configurations are made the administrator has to run the database.sql on a MySQL server. After the database was created he has to copy the two wars: licenta-application.war and licenta-web-services.war to the web applications directory on a Java application server.

At this point the project will be available under http://domain:port/licenta-application.

7.2. Utilization instructions

After the configurations are set, and the application is deployed on a Java application server the application will be available under http://domain:port/licenta-application. When accessing this link, a presentation page will be shown like in the Figure 36. This page gives the option of Signing up or, by scrolling down, we would be able to see the log in form.

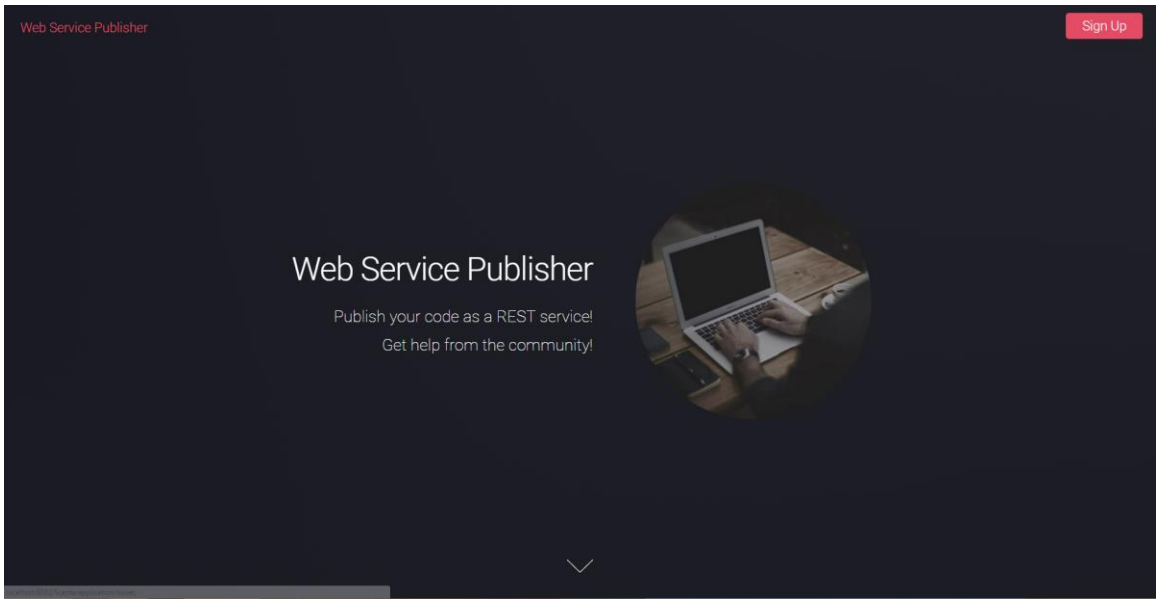

Figure 7-1 Web Service Publisher Intro page

The login form has the standard layout, a username field, a password field and a Sign In button.

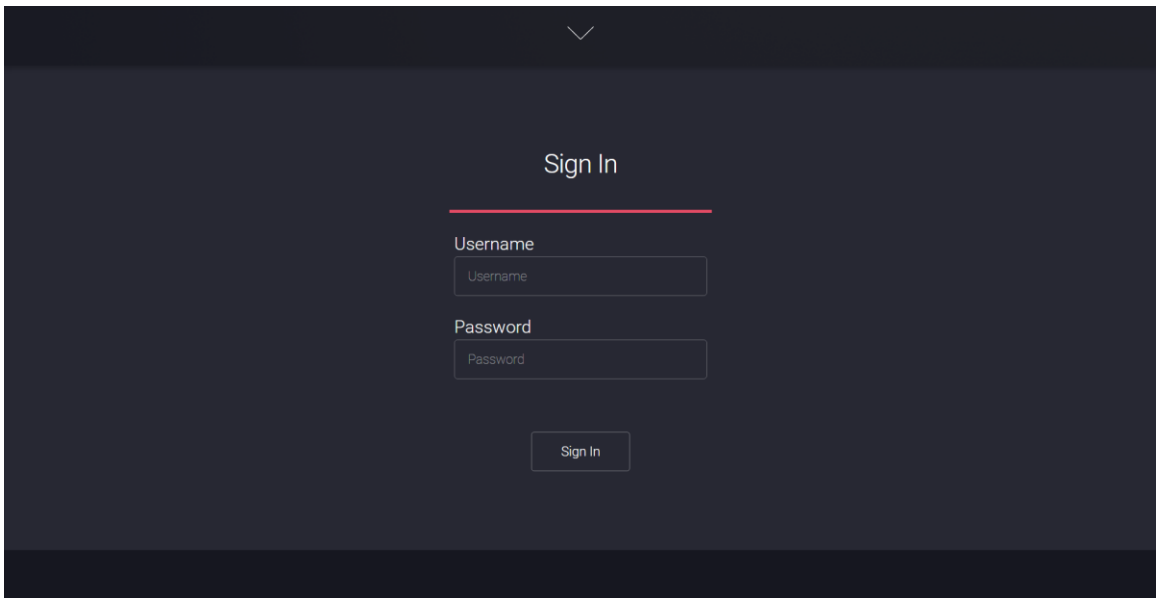

Figure 7-2 Sign in form

After Signing in the user is redirected to the list of questions from the questions and answers module. This page was reffered accross this paper as the „List of Questions” page.

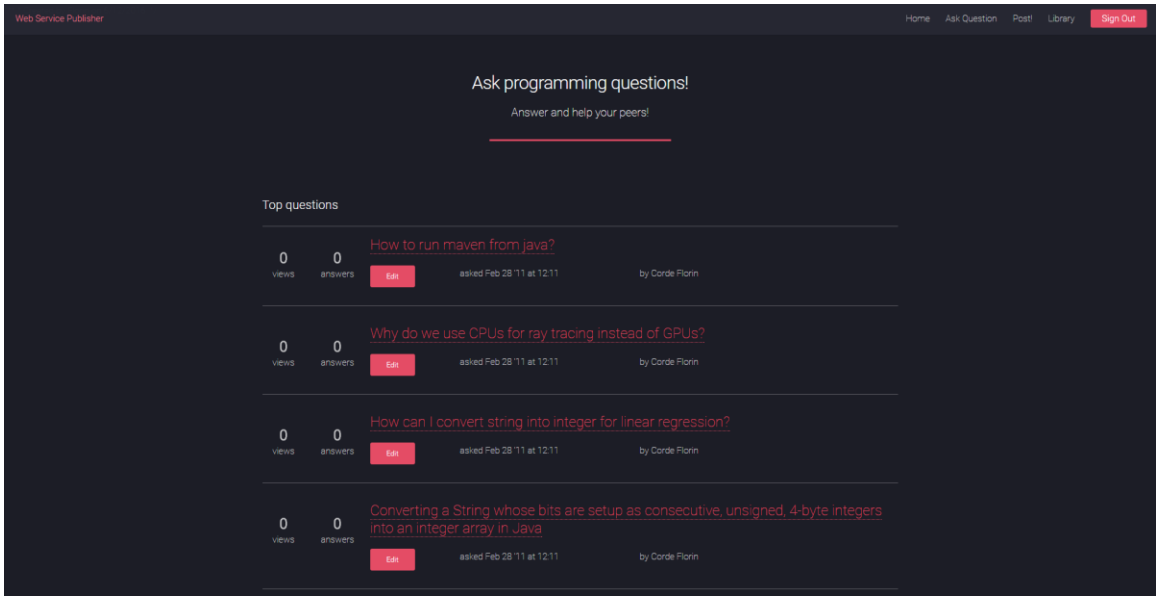

Figure 7-3 „List of Questions" Page

At this moment in time the user is logged in the application. He can access all the features of the application by using the menu bar on the top right part of the page. The menu includes the items Home, which will redirect the user to the „List of Questions Page", Ask Question, which will allow the user to post questions which will appear in the list shown in Figure 38, „Post!" which allows the user to publish standalone web services with no connection to comments or issues, and Library for vizualizing all the web services published with the Web Service Sharing System.

Next the user can opt to select a question from the list of published questions and answer it by posting a comment.

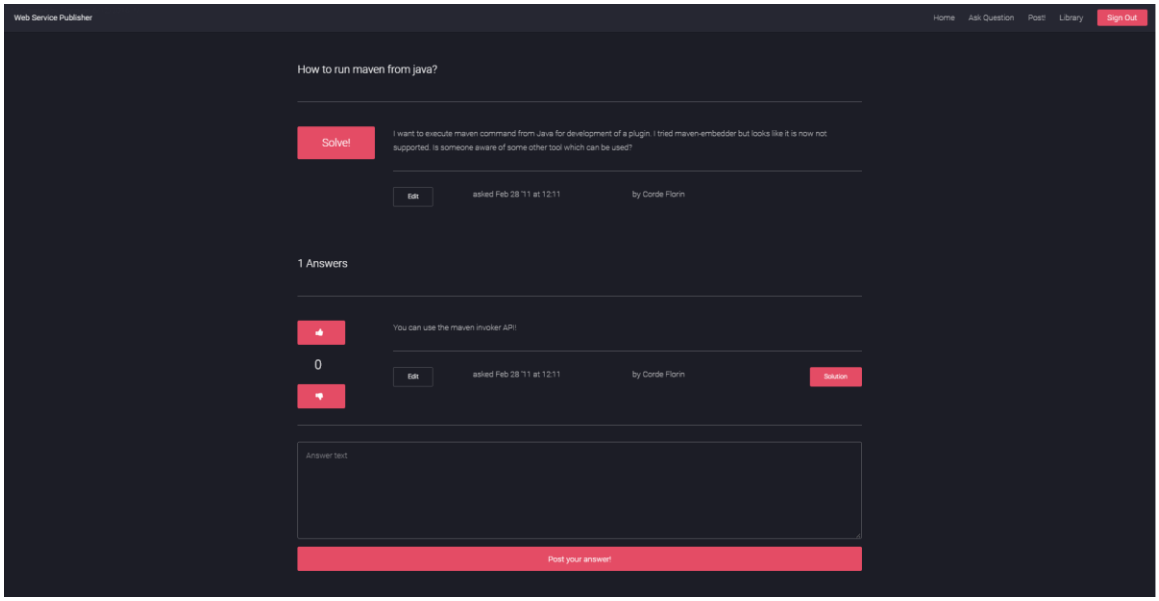

Figure 7-4 „Question Details" Page

After posting a comment the user has the option of attaching a web service to the comment by pushing on the button „Solution”. This function is available only for the comments posted by the same user that is logged in the application. The user that has asked the question can push the button „Solve!” to change the marker of the question as solved. The user that has posted the question can also edit the question that he asked, and also the users that have posted comments on questions can eddit their comments.

The user would post a comment by typing a comment in the text box at the bottom of the screen an by pushing the button „Post your answer!”. The users can also like each other’s comments by using the thumbs up and thumbs down buttons on each of the comments.

In the next section we will present how to post a question. The user can make this operation by clicking the „Ask question” on the top of the page. This will lead to the „How to ask” page which shows some recommandations about how the format of the question should be, what should be the content and how to make the question relevant for the other users.

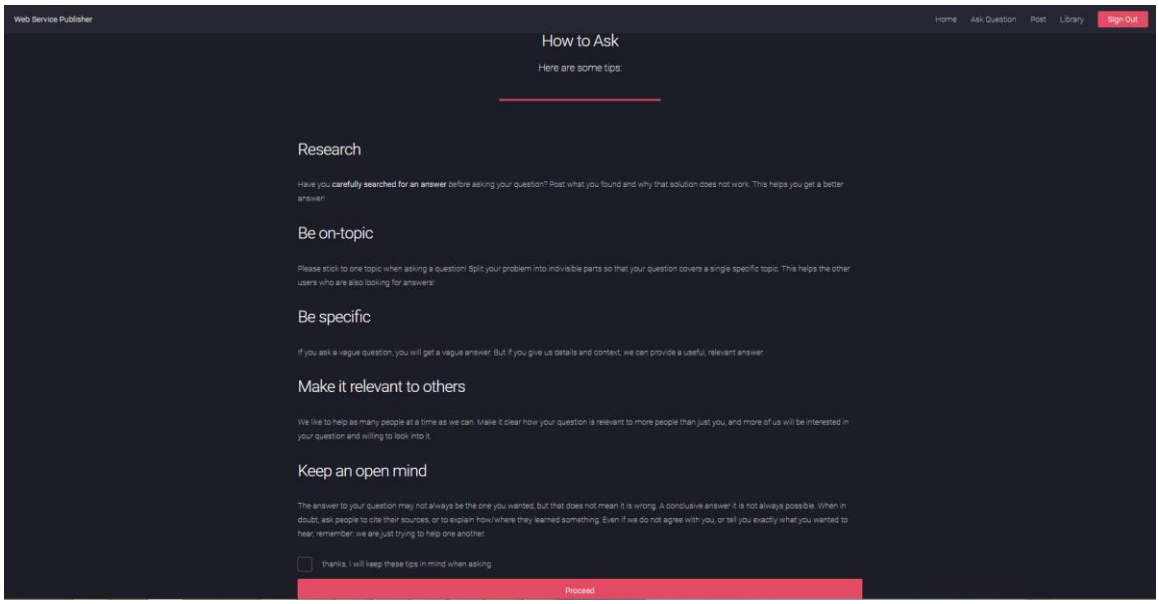

Figure 7-5 „How to ask” page

The user needs to ckeck the checkbox to confirm that the user has read the page. He can go to the next step by clicking on the button „Proceed”. This action redirects the user to the „Ask Question” page in which the user can fill the text boxes for the question’s body and the question’s title. Some validations are done on this page, for example, the user can not submit the page if one of the fields is empty. When submitting this form the user is redirected to the Question Details page.

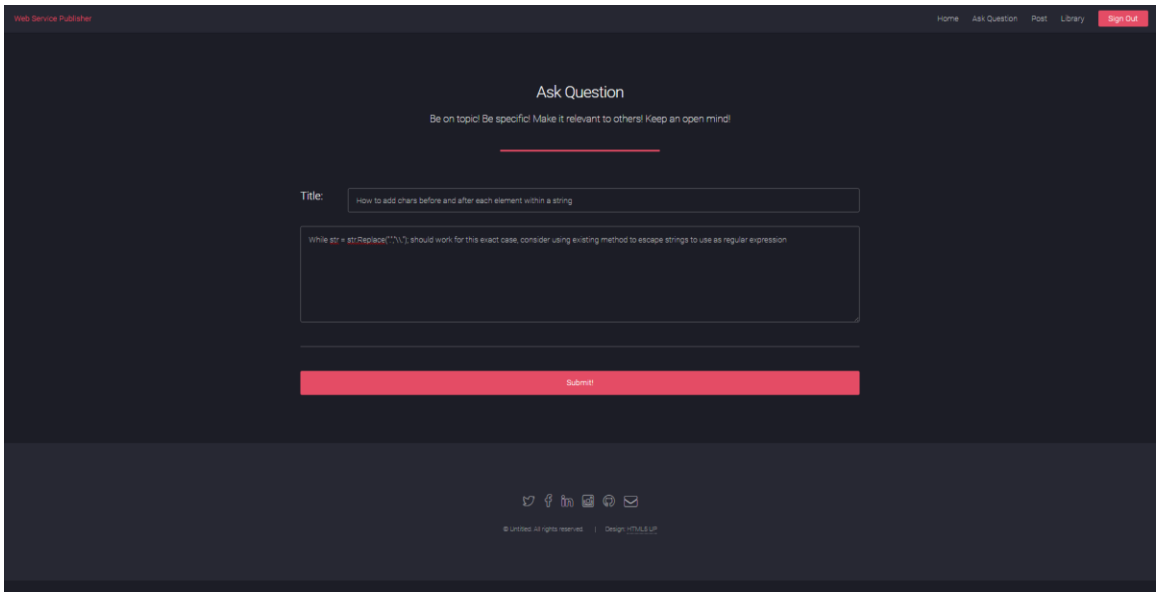

Figure 7-6 „Ask question" page

In the next section we will describe the „Post your code" page. This page cand be accessed by cliking on the „Post!" button from the header, or by clicking the button „Solution" on a comment, or by goring into the „Library" page and selecting one of the web services from the list, case in which the page will have all of it's fields in read-only mode.

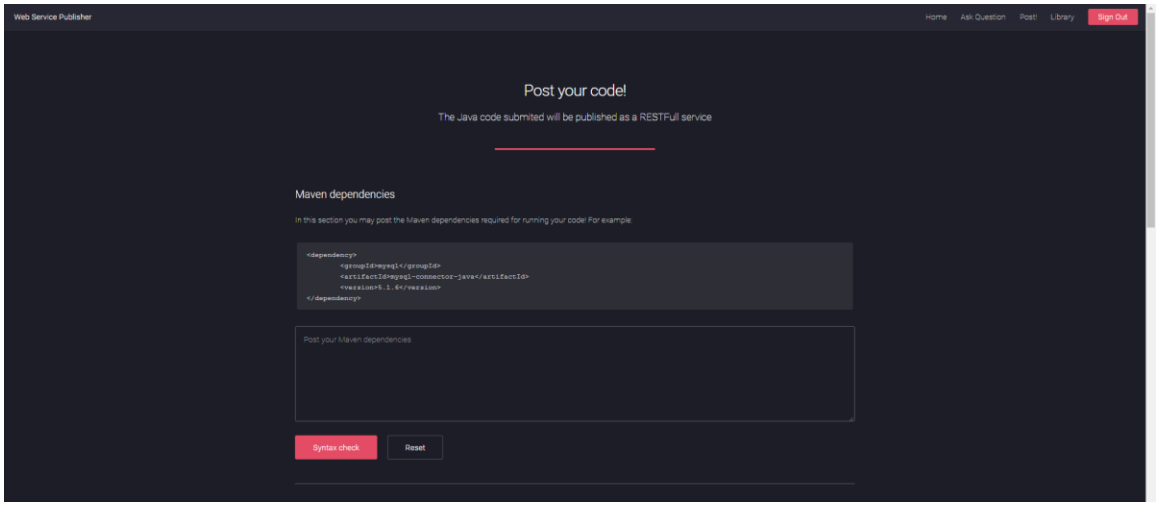

Figure 7-7 "Post your code" page Maven dependencies and header

The Figure 42 presents the top part of the page „Post your code!" which is the Maven dependencies text field. Above each text field we have examples of code for the user to see and take notice of. The Maven section is dedicated to the users that want to use in their code external APIs which are not in the standar Java library. The dependencies

posted here will be added to the project's pom.xml file and imported into the project as jars.

The next section of the page is shown in the Figure 43, which is the inports section. This section is to be used as the first part of a Java class is used to import different libraries.

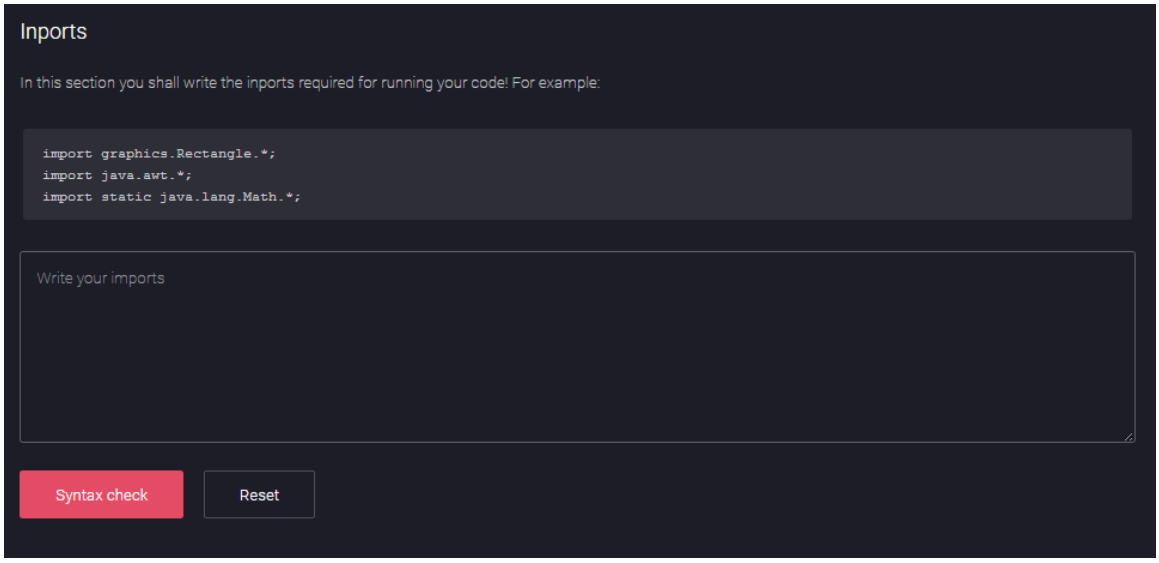

Figure 7-8 "Post your code" page Inports section

After this section we have created a „Service Details" section in which the user will have to imput details about the web service he will post such as the web service name, the number of parameters the web service will have, the return type of the web service, and details about the web service's arguments: their name and their type.

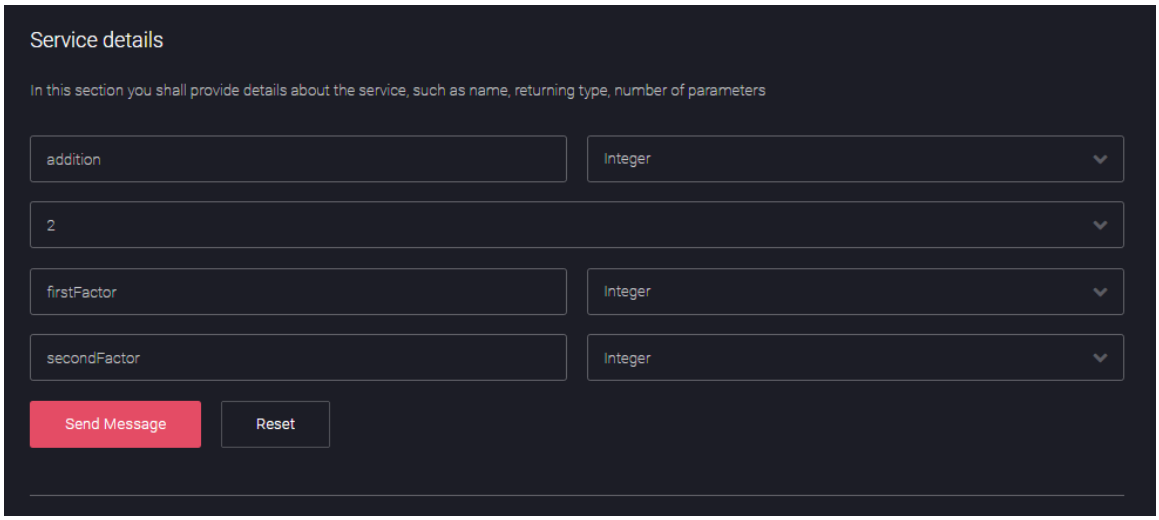

Figure 7-9 "Post your code" page Service details section

Finally there's a section which will allow the user to publish his code. Here the user has to write the code as it is in a java method's body returning a value which is of the same type as the value specifyed in the Service details section of the page.

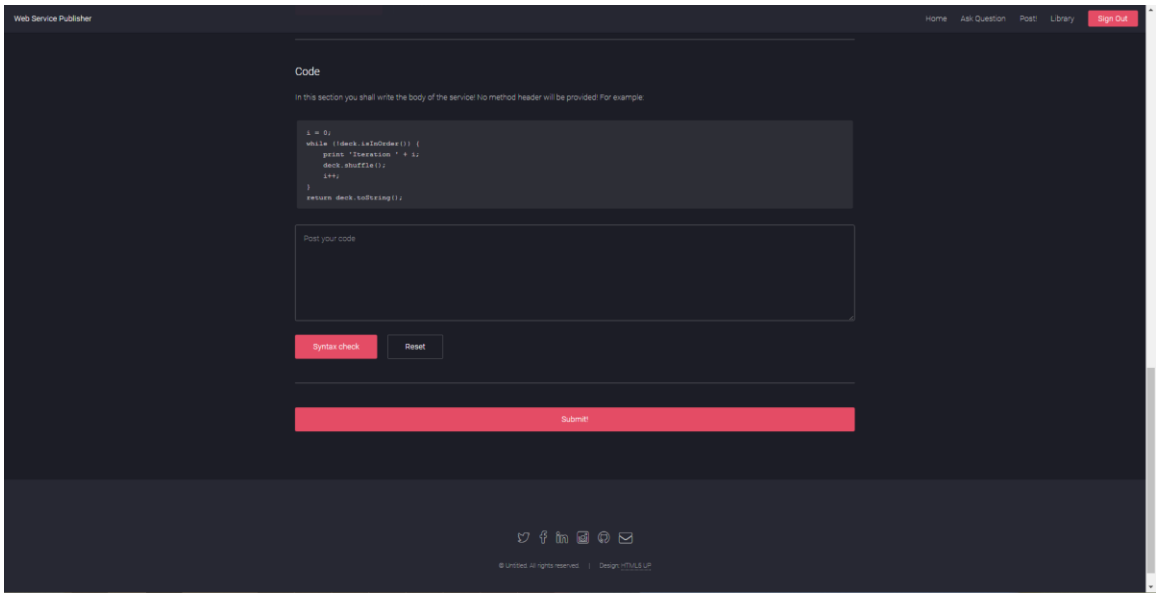

Figure 7-10 "Post your code" page, Code section

When the user finishes writing the details of the web service he wants to publish he can push the button „Submit!”. This will leed him to a loading screen which will be displayed until the web service has published or has encountered an error, in which case he will have the chance to recheck his code and his other imputs.

In the next section we will describe the „Library” page. Here the user will have the chance to view all the web services posted in the application.

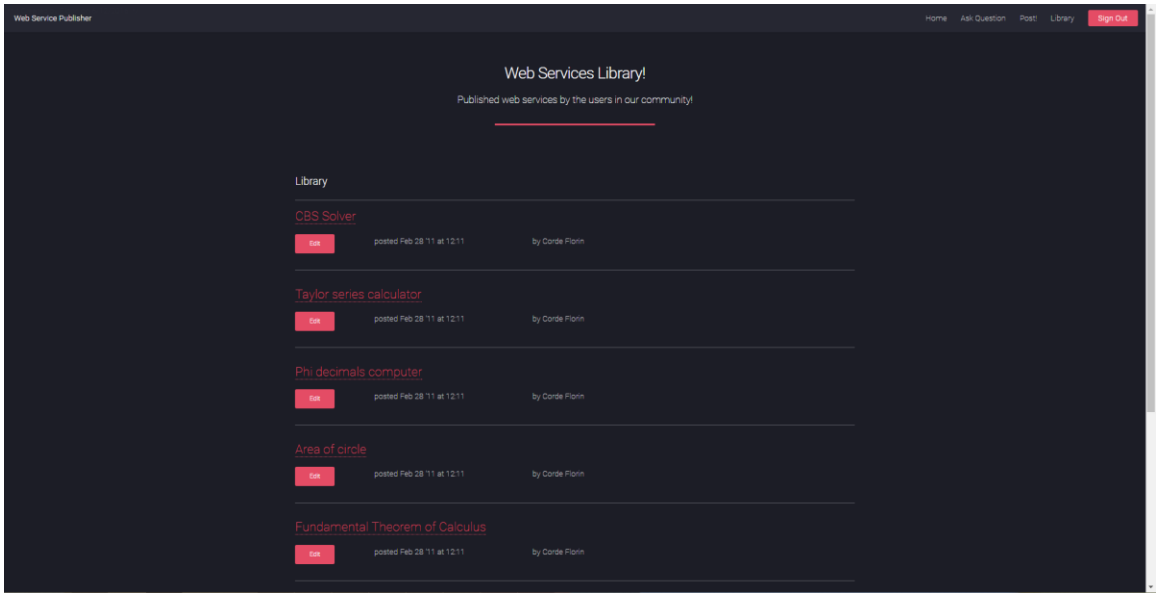

Figure 7-11 "Library" page

Here the user can see the name of the web services published, and by clicking on them he will be redirected to the „Post your code!” page wich will have the published service details.

7.3. Requirements

For running the application, the user has to have installed a web browser which supports HTML5 and CSS3.

For the machine that will hold the application, it needs to have installed an operating system that whould be compatible with a Tomcat server and a MySQL server, operating systems like MacOS, Windows or Linux. The system holding the servers needs to have installed java 8. The memory and processing resources needed to run this system for one user are small but requirements grows with the number of users which post and use web services from the Web Service Sharing System.

Chapter 8. Conclusions

8.1. Achievements

We have managed to create a question and answers system that allows users to post implementation as answers to those questions. The system is built on an architecture that allows adding extensions easy.

We have managed to implement all the project objectives that have been proposed at the start of this paper, which are posting a web service, asking question, responding to question, testing the posted web services, and sign up and authentication of the users.

8.2. Results

A modular and easy to use system has been created. The interface is intuitive and implemented in HTML5 and CSS3 style. The frontend is controlled by the AngularJs framework which was new for us. This lead to extended periods of time of getting to know the framework, following tutorials and examples for getting to a level of knowledge of the framework that will allow us to build the functionalities proposed for this project.

The previous experience of working with Spring helped me in the implementation of the backend modules and has been a big part in the modularization and architecture of the project.

In the end we have created a product that has similar functions to a Q&A web site which has the extra feature of publishing web services as answers. This leads to a lot of benefits like being able to implement solutions in the Web Services Sharing System and eliminate the need to implement them once again in the projects that users work on their own.

8.3. Future Developments

The architecture of the Web Services Sharing System allows for adding future developments in the easiest way possible. In the backend all you have to do is create new REST controllers and go from there to the backend service layer, and in the frontend to create new controllers for new pages implemented and services to communicate with the REST controller that he has just created.

The main future development that can be done is a syntax checker for the code that the user is posting. The user finds out that there's a syntax error in his code only after the web service is build and compiled. Here a real live syntax checker can be used to highlight the code that is wrongly typed in the text fields of the page "Post your code". This can be done through a lot of different existing services like Java 9's JShell.

Another possible future development would be saving the user's inputs both in the Questions and Answers module and in the Publish a web service module, in a HTML format. Giving the user options to format their code and saving in the database details about this format, like Fonts or indentation details, so that when the web service is displayed it looks as the user inserted it. Right now the code is saved in a String format which disregards details about the user's input indentation and format.

Another possible future development would be using a Class Loader to insert the REST controllers that the user publishes in the licenta-web-services module without interrupting the server's function when doing that. This would mean that the web services published by the users would be fully reliable to use in other projects. Right now, the way that the system is implemented, every time the user publishes a new web service the application with web services is restarted, and for a short period of time the web services are not available.

Another possible future development would be rewarding users that consistently post answers which have a high number of likes, and creating a medals or badges feature that would distinguish the users between themselves. This kind of approach is implemented in all the existing Q&A sites like StackOverflow.

Another future development can be an autocomplete feature of the code inserted by users. This is common in all the existent IDE's and it would be a nice addition to the system.

One last future development worth mentioning would be a benchmark system of some kind that would test the different implementations of the question to establish the most efficient implementation out of all proposed implementations. This benchmark system would take in consideration execution time and memory used and would create a hierarchy of implementations for the same problem. The implementation with the best parameters will be the only one posted into the library. This way we can make improvements in regard to space used on the server.

Bibliography

[1] Michael Armbust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia, "A View of Cloud Computing", In *Communications of the ACM* 53.4 (2010).

[2] Peter Mell, Timothy Grance, "The NIST Definition of Cloud Computing", In *NIST Special Publication 800-145*, September 2011.

[3] Surya Chandra, Durga Charan, Sudha Rani, "Online C, C++ & Java Compilers using cloud computing", *In IJCSMC, Vol. 4, Issue. 8,* August 2015.

[4] Browxy Documentation. Available at: http://browxy.com/ [Accessed 19 Jun 2016]

[5] B. Vasilescu, A. Serebrenik, P. T. Devanbu, and V. Filkov. How social Q&A sites are changing knowledge sharing in open source software communities. In CSCW, pages 342–354, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2531720

[6] Anderson, A., Huttenlocher, D. P., Kleinberg, J. M., and Leskovec, J. "Discovering value from community activity on focused question answering sites: a case study of Stack Overflow", *In Proc. KDD*, ACM (2012). [Online]. Available: http://dl.acm.org/citation.cfm?id=2339665

[7] Sajid Abdulla, Srinivasan Iyer, Sanjay Kutty, "Cloud Based Compiler", In *International Journal of Students Research in Technology & Management, Vol 1(3),* May 2013.

[8] Bruno Costa, Paulo F. Pires, Flavia C. Delicato, "Evaluating a Representational State Transfer (REST) Architecture", In *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on* (pp. 105-114). IEEE.

[9] Sravanthi Emani, N.B. Pokale, Arti Chetwani, Archana Patwari, "Web Based C IDE: Approach", In *International Journal on Computer Science and Engineering* 4.3 (2012).

[10] Tejashri Gaikwad, Poonam Dhavale, Karuna Gaware, Mr. Nitin Shivale, "Web Based IDE", In *International Journal of Research in Information Technology (IJRIT) Volume 2, Issue 4,* April 2014, Pg:616-620.

[11] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns, "What Makes a Good Code Example? A Study of Programming Q&A in StackOverflow", In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012.

Appendix 1. Glossary

Term	Definition
Web Service	Standard Web Services use The SOAP protocol which defines the communication and structure of messages, and XML is the data format. Web services are designed to allow applications built using different technologies to communicate with each other without issues.
RESTFull	A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.
Maven	Maven is a build automation tool used primarily for Java projects. The word maven means "accumulator of knowledge" in Yiddish.
Spring	The Spring Framework is a Java platform that provides an infrastructure for developing Java applications.[9] This framework can be used for easily achieving benefits such as executing a Java method in a database transaction without the need of using transaction APIs, transforming a local method to a procedure that can be called remotely without the need of using remote APIs, handle messages without the need of using JMS or RabbitMQ APIs.
Back-end	The back side of the software, which is unseen by the user.
Front-end	The part of the system which sits between the user and the Back-end.
Framework	In computer programming, a software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software.
API	An API is set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.
Session	In computer science, in particular networking, a session is a semi-permanent interactive information interchange, also known as a dialogue, a conversation or a meeting, between two or more communicating devices, or between a computer and user.
Factory	In object-oriented programming (OOP), a factory is an object for creating other objects – formally a factory is a function or method that returns objects of a varying prototype or class from some method call, which is assumed to be "new".
Benchmark	A standard or point of reference against which things may be compared.
REST	REST stands for Representational State Transfer. (It is sometimes spelled "ReST".) It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used. REST is an architecture style for designing networked applications.

[9] http://docs.spring.io/

Appendix 2. Figures list