

Table of Contents

[List of Abbreviations 4](#_Toc517120711)

[Introduction 5](#_Toc517120712)

[1. Problem analysis 6](#_Toc517120713)

[1.1. Overview of already existing solutions 6](#_Toc517120714)

[1.1.1. Email Templates AutoMailer 6](#_Toc517120715)

[1.1.2. Templates Email Marketing 9](#_Toc517120716)

[1.2. Conclusion 11](#_Toc517120717)

[1.3. Goal and tasks of the bachelor thesis 11](#_Toc517120718)

[2. Architecture and design 13](#_Toc517120719)

[2.1. Programing language, platforms, libraries, IDE 13](#_Toc517120720)

[2.1.1. Java Enterprise Edition 13](#_Toc517120721)

[2.1.2. TypeScript 13](#_Toc517120722)

[2.1.3. Spring Framework / SpringBoot 14](#_Toc517120723)

[2.1.4. Hibernate (spring-boot-starter-data-jpa) 17](#_Toc517120724)

[2.1.5. CSS3 20](#_Toc517120725)

[2.1.6. Bootstrap 3, PrimeNg, Semantic UI 20](#_Toc517120726)

[2.1.7. Angular 2/5 21](#_Toc517120727)

[2.1.9 Maven 23](#_Toc517120728)

[2.1.10 IDEs 24](#_Toc517120729)

[2.1.11 Git 25](#_Toc517120730)

[2.2. Solution to the problem 26](#_Toc517120731)

[2.3. Web Application Architecture 32](#_Toc517120732)

[2.3.1. MVC 32](#_Toc517120733)

[2.3.2. Architecture of the back-end 33](#_Toc517120734)

[2.4. Data organization 34](#_Toc517120735)

[2.4.1. ER model of the DB 35](#_Toc517120736)

[3. Practical solution 45](#_Toc517120737)

[3.1. Setting up the Workspace 45](#_Toc517120738)

[3.2. File Structure of the application 46](#_Toc517120739)

[3.3. Presentation Layer 47](#_Toc517120740)

[3.4. Model Layer 52](#_Toc517120741)

[3.5. Controller Layer 55](#_Toc517120742)

[3.6. Additional use of the email templates 58](#_Toc517120743)

[3.7. User guide for the module 60](#_Toc517120744)

[4. Conclusion 63](#_Toc517120745)

[4.1. Summary of the results 63](#_Toc517120746)

[4.2. Ideas for future research and development 63](#_Toc517120747)

[Literary sources 64](#_Toc517120748)

[Attachments 65](#_Toc517120749)

# List of Abbreviations

|  |  |
| --- | --- |
| Acronym | Meaning |
| IT | Information Technology |
| UI | User Interface |
| SDK | Software Development Kit |
| API | Application Programming Interface |
| GUI | Graphic User Interface |
| XML | Extensible Markup Language |
| HTML | Hypertext Markup Language |
| CSS | Cascading Style Sheet |
| IDE | Integrated Development Environment |
| JSP | JavaServer Pages |
| JS | JavaScript |
| DB | Database |
| WYSIWYG | What You See Is What You Get |
| JVM | Java Virtual Machine |
| MVC | Model View Controller |
| POJO | Plain Old Java Object |
| RMI | Remote Method Invocation |
| EJB | Enterprise Java Beans |
| URI | Uniform Resource Identifier |
| AJAX | Asynchronous JavaScript and XML |
| ORM | Object Relational Mapping |
| JPA | Java Persistence API |
| SQL | Structured Query Language |
| HQL | Hibernate Query Language |
| DOM | Document Object Model |
| SVN | Subversion (Repository) |
| ER | Entity Relationship |
| URL | Uniform Resource Locator |

# Introduction

All around the world people use emails as a mean to communicate with each other. What happens when you have a large network of contacts you wish to inform for some event? No matter how fast and good you are at typing emails, this task is time consuming and nerve wracking. The solution for this problem comes in the form of templates, previously recorded messages which you simply apply to your emails. With the help of templates people can send emails fast and efficiently, but there is one problem. When you send an email with a template you have to make the template “work” in the general cases and from that derive the problem: the people that receive such emails will not feel connection to you and probably ignore your message. To increase the chance not to be ignored you have to use context-sensitive content. With simple words that means that in the templates there are defined key words, which change their context according to the person to whom the email is send.

Considering all of the above, we have chosen as a main objective of this diploma thesis the design and implementation of a module for management of context-sensitive content for web-based information system. For the purpose of completing the objective of the thesis I have identified a few main processes. The first ordeal will be the research of the various ways for extracting data from the emails/DB. The Second will be to choose the most appropriate one and last but not least will be the implementation of the GUI for the management of the context-sensitive content.

In the next few chapters of the thesis a detailed overview of the template module will be given. First a comparison with other modules with the same functionality will be made. After that the technologies that have been used to create the module will be presented and a short explanation of them will be given. An explanation of the module architecture and how the front-end and back end work with each other will also will be given. At the end of the thesis user guide that show the complete form of the module will be presented.

# Problem analysis

## Overview of already existing solutions

The concept of personalized email templates is largely used and has many implementations. In this section we will try to examine and compare different solutions and ours.

### Email Templates AutoMailer

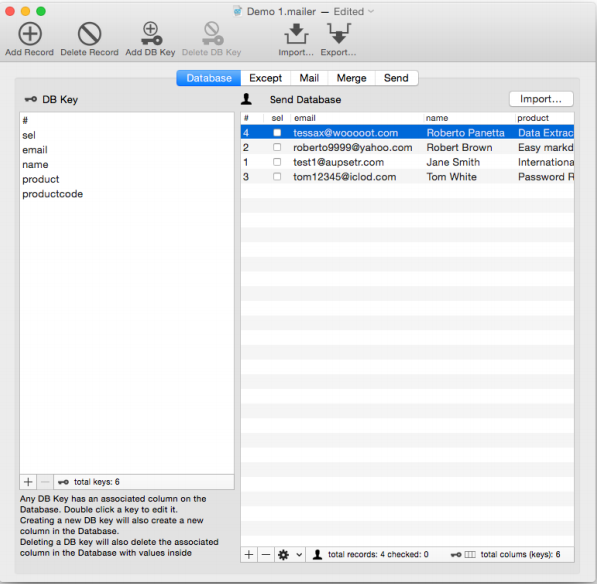
****

Fig 1: DB keywords creation in AutoMailer

AutoMailer is a flexible tool to manage your mailing list directly from your Mac. It offers a variety of functionalities one of which is personalized emails [1].

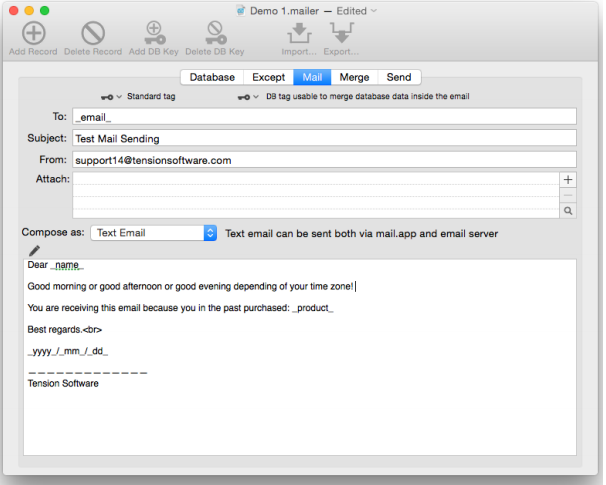
****

Fig 2: DB keywords usage in AutoMailer

**The email templates of AutoMailer offer the following functionality:**

* Document based, allows to save data and setting and have them ready for successive uses
* Possibility to manage an infinite number of data and letter template thanks to the document based approach with maximum flexibility of use.
* Very simple creation of email templates with keywords insertion to put database data inside the emails for any email recipient
* Automatically updated popup to insert data keywords inside the email template
* Live preview for any email to send with updated data (you see the email as the recipient will see it)
* There's no need to save, the application does it for you. Always.
* Big emphases on simplicity and intuitive way of usage.

The Application is simple, intuitive and with a clean UI. The main functionality is to allow the users to send personalized emails to multiple addresses, each merged with data from an internal database [1].

In the module for management of context-sensitive content, that is the objective of this thesis work, are incorporated some of the functionality from the email templates of AutoMailer application. A flow in AutoMailer is that it is not supported on windows and android devices. That is a major restriction on the windows/android users, which leads to profit loss on the company side. The fact that the user has to install the application on his local device can also be said as a shortcoming – if it was web based the users could have had access to the system from any device at their disposal. The following table (table 1) compares the emails templates of AutoMailer and the module that is object of this thesis.

|  |  |  |
| --- | --- | --- |
|  | **Email Templates of AutoMailer** | **Email Templates Module**  **Of Muffing Gamification** |
| Target Group | People who have large email list and want to send personalized emails. | Direct use in the web based information system Muffin Gamification. |
| Platform | OS X , iOS | It is web-based i.e. it can be used on any platform that supports internet |
| Dropdown menu in the composition of new emails with keywords. | No | Yes |
| Creating new keywords from the UI. | Yes | No |
| List of Templates. | No | Yes |
| Feature to create draft templates. | No | Yes |
| Feature to save templates for later usage. | No | Yes |
| Validation for user actions. | Yes | Yes |
| Live preview for emails. | Yes | No |
| Auto complete of keywords in mail content. | Yes | No |

Table 1: Comparison between Muffin Gamification Email Templates module and AutoMailer Email Templates module

In conclusion the email templates module of the Muffin Gamification web-based information system has bigger functionality than the email templates of AutoMailer.

### Templates Email Marketing

Email Marketing is a web based information system that targets various businesses [2].

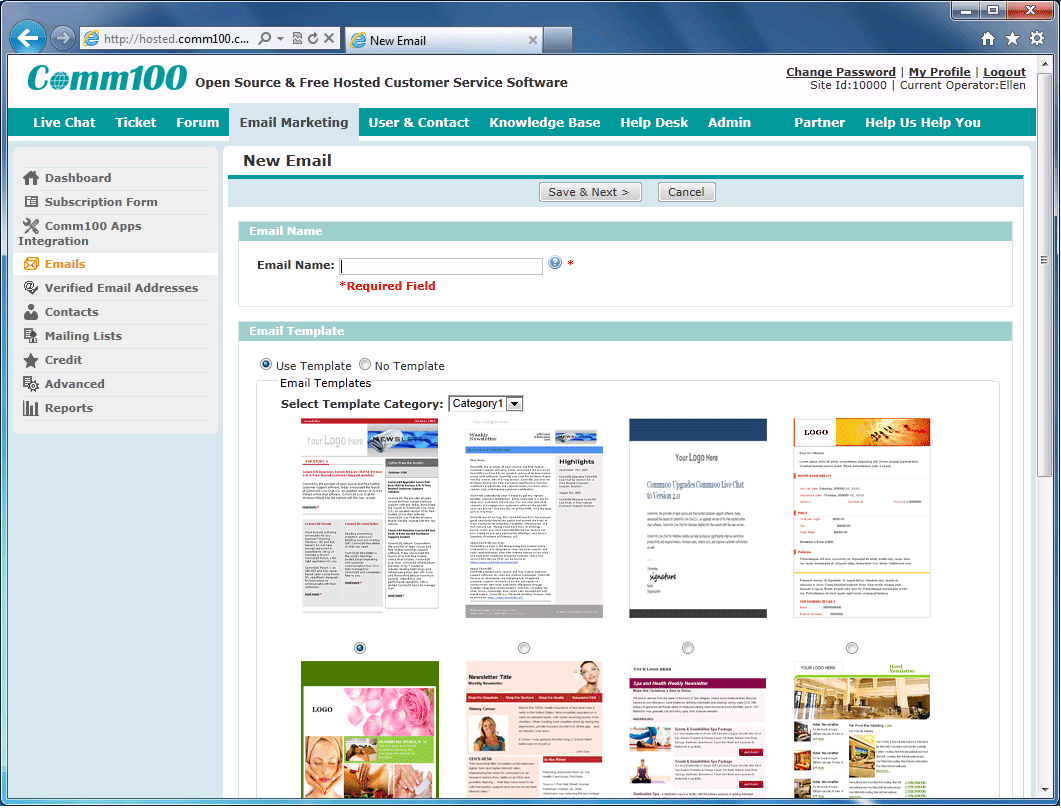
****

Fig 3: List of premade templates in Email Marketing

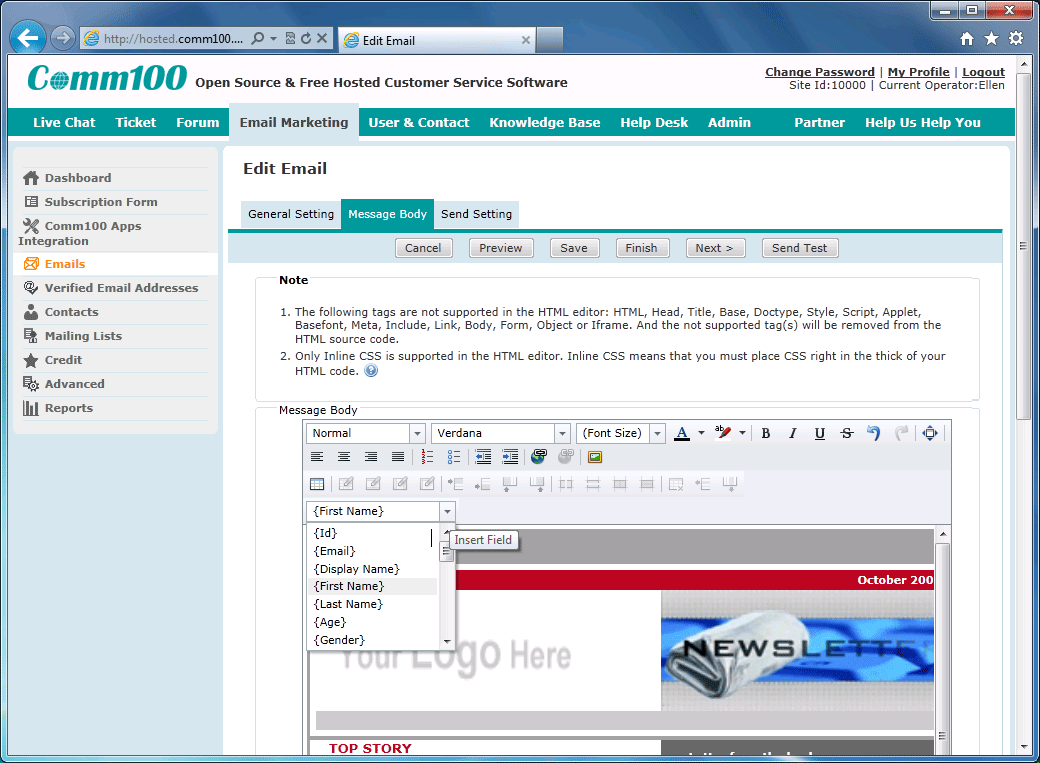
****

Fig 4: Keywords usage in Email Marketing

**The email templates of Email Marketing offer the following functionality:**

* List of Professional Email Templates
* Easy-to-Use WYSIWYG HTML Editor
* Email Personalization
* No Installation & Zero Maintenance

Email Marketing offers a large variety of premade templates, that the users can change to his or her liking. The users don’t require any technical expertise for personalization of emails with logos, images, color, etc. The application offers a large diversity of the most used keywords such as {First Name}, {Last Name}, {Age}, etc.

The difference between the templates module of Email Marketing and the templates module that is the objective of the thesis is mainly in the WYSIWYG HTML editor of the templates. In the Muffin Gamification templates module the customization of the content is harder and not so refined. Table 2 shows in more detailed way the differences between the two modules.

|  |  |  |
| --- | --- | --- |
|  | **Email templates module of Email Marketing** | **Email Templates Module**  **Of Muffing Gamification** |
| Platform | All devices that have internet access | All devices that have internet access |
| Dropdown menu in the composition of new emails with keywords. | Yes | Yes |
| Creating new keywords from the UI. | No | No |
| List of Templates. | Yes | Yes |
| Feature to create draft templates. | No | Yes |
| Feature to save templates for later usage. | Yes | Yes |
| Validation for user actions. | Yes | Yes |

Table 2: Comparison between Muffin Gamification Email Templates module and Email Marketing Email Templates module.

## Conclusion

In conclusion we can say that there are a lot of similar modules in the application that offer mail services. Every application has added different functionality depending on the needs of the clients but their primary function is the fastest, easiest and the most efficient way of creating personalized emails for mass email sending.

## Goal and tasks of the bachelor thesis

The goal of the bachelor thesis is to create a module that has the main functionality that is defined in the previous section (1.2.). The point of the module is not to be overcomplicated but to be a simple, clean, and intuitive module that can be used by people that have no technical knowledge.

The tasks are:

* Design of the DB that will be used.
* Implementation of a page for visualization and management of email templates.
* Building the UI for the page.

# Architecture and design

## Programing language, platforms, libraries, IDE

This chapter of the thesis describes the technologies used in to implement the module that is the objective of the thesis.

### Java Enterprise Edition

“Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software. Java EE is developed using the [Java Community Process](http://www.jcp.org/), with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals”[5]. Even though I am not going to use a lot of these features Java EE includes a large variety of APIs such as RMI, e-mail, web services and XML. There are several unique specifications to Java EE: EJB, connectors, servlets, JSP and several web service technologies. If we keep in mind the fact that JDK is free to use and 2 of the best IDE that can be found use Java, Java is one of the main contributors to the open source community. With that been said I would like to point out that there are a lot of free to use libraries, some of them have event become an irreplaceable part of the programming community. Another advantage of Java is the large community that is always ready to help with algorithms, ideas and solutions to various problems.

### TypeScript

With the rapid development of JavaScript (from now called JS for short) emerged a lot of “deficits” that halt the developers to use the language for application-scale development. Features such as object orientation, compile time error handling and strong type checking are highly valued and sought after in today’s programming world. And with that comes TypeScript from now being called TS for short. As said above TS is an object oriented language that supports strong typing and compile time errors with the basic building blocks of JS. With other words TS is an enhanced version of JS (it is often said that TS is typed superset of JS that is compiled to JS).

Some of the important features of the language are exactly that the developers can easily switch from JavaScript to TS because of the similar syntaxes of both languages; TS can be consumed by any JS library and vice versa, meaning JS is TS and TS is JS. TS encompasses most of the basic features of ECMAScript5 (official specification for JS) and also incorporates features such as Modules, classes, generics and type annotations. So to sum up why I am going to use TS instead of JS for the implementation of the web based system:

* Compile time check – JS is interpreted language that needs to be ran to see if the code works, if everything is alright we see the result instantaneous but if there is error somewhere we have to spend a lot of time to investigate where and why our code doesn’t work correctly. TS with its compile time check don’t have such problems and we can save some investigation time.
* Strong Typing – JS doesn’t have types for the variables which inevitably leads to using variables that you are not sure exactly what they are hence increasing the possibility of making errors in your code. Once again TS has a solution for this problem – strong variable type.
* OOP – this is one of the most common and used principle in the programing. Like the back-end language that I am using for the project(Java) TS also supports concepts like classes, interfaces that will in its most insignificant part boost the readability of the code for outside personnel and that is never to be underestimated for future development.

### Spring Framework / SpringBoot

When you start a project one of the things you have to consider are the frameworks that you are going to use. The Spring Framework is one of the most popular and used frameworks for java enterprise applications. Spring is open source, lightweight, high performing framework with which you can develop any java application but it is mostly use for its extensions for building web application on top of J2EE platforms. Some of benefits of using Spring are but not only:

* The design of the framework is modular, meaning even thou there are a lot of packages, classes and interfaces the developer doesn’t need to manage all of them, he or she only needs to tend to the ones he/she is using.

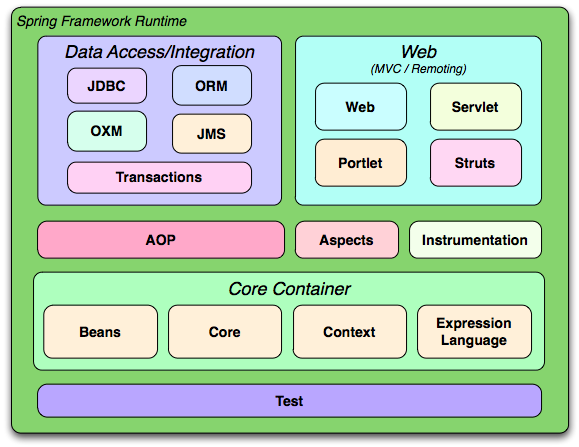


Figure #: Spring Modules (moje bi trqbva da q prepravq ili mahna nz )

* The framework incorporates in itself a lot of well know technologies such as ORM frameworks, logging frameworks and J2E.
* Transaction management interface – you can easily scale your transaction up and down when needed.
* Dependency Injection – one of the fundamental concepts in developing application is the ability to test and reuse your code, meaning you don’t want your classes to be tightly coupled with each other. Here comes the Dependency Injection into play – it helps us supply the dependency between two classes. A simple example of that is when you have an object car and you know that the car needs another object engine – the DI can give you the already made instance of the engine object so you don’t need to create the engine manually.
* POJOs – you can use plain old java objects for better testing or simply to get whole objects from forms (send from the client side).

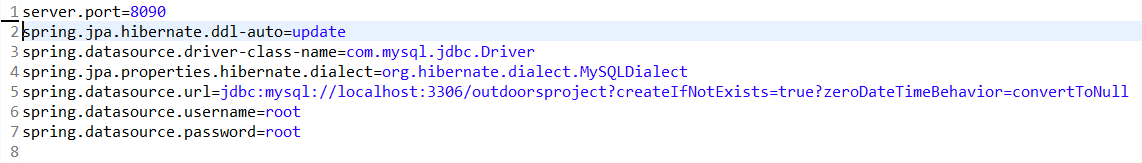
Spring also provides frameworks for security and data that can easily be used in your project to secure your application better or to simplify database access. Another one of the benefits of the framework is the flexibility to configure beans: currently there are 3 types for that – xml based approach, annotations and JavaConfig approach.



Example of XML approach

The flexibility here comes from the fact that you can use one or more meaning you can combine approaches) of the approaches to configure your beans depending on the situation at hand.

But one of the down sides of the Spring Framework is also the configuration part. Oftentimes when developers new to the framework try to build a project from the ground up, they have numerous problems with the setup. When you try to develop a typical spring/hibernate web application usually the processer of the configuration consists of the following: configure dependencies (using build tools such as maven or gradle), configure the various beans in spring framework (service/dao) – using the latest approach with JavaConfigs ypu can omit some of the lightly descriptions of configurations and using simple properties files (FIGURE)



Properties File Figure

And last but not least you have to configure front controller of this Spring Web application, responsible for handling all application requests i.e. the Spring DispatcherServlet which in itself can be configure in multiple ways. As you can already figure out configuring the framework is not very fast and easy if you have not done it multiple times:

* You are not always sure which libraries you need for your project to run, so you can miss some
* You have to spend some extra time while configuring the DB connections (EntityManager, TransactionManager)
* Configuring the various Spring Beans
* Configuring a web application server

All of these configurations are simplified and even in some cases made automatic by SpringBoot. For example the dependency/library management - in case of the most basic application instead of adding numerous dependencies, with SpringBoot you only need to add 2: spring-boot-starter-web and spring-boot-starter-data-jpa. These two dependencies are easily managed and pull every necessary library that you need such as spring-webmvc, spring-data-jpa, hibernate, tomcat and more. Springboot also creates default configurations for some of the most commonly used beans – front controllers and database connections. Spring-boot-starter-web very conveniently also pulls a servlet container (tomcat) on which serves as an embedded container where you can deploy your application without needing to install an external server/container.

<http://javainfinite.com/spring/life-cycle-bean-factory-application-context/>

nakraq flow-a na spring ? https://stackoverflow.com/questions/11359607/understanding-basic-spring-framework-and-total-flow

### Hibernate (spring-boot-starter-data-jpa)

As already mention above the spring-boot-starter-data-jpa is responsible for pulling a lot of useful libraries that help us with the development and one of those is Hibernate. Hibernate is an Object/Relational Mapping (ORM) framework. With that the framework helps the developers to write applications whose data persist after the applications process. In simpler words, that means that the objects that we use can be available for later usage. While Hibernate has its own API it also uses the Java Persistence API (JPA) which further simplifies and eases its use in any environment supporting JPA. Following the natural object-oriented idioms (inheritance, polymorphism, association, composition and java collections framework) Hibernate allows the developers to create persistent classes with ease. With no requirements for a special database tables and SQL generation at the system initialization time – Hibernate has very high performance. Other reasons for the high performance are the numerous fetching strategies and the usage of eager and lazy initialization. [11]

There is no need to fully understand SQL to use Hibernate, but the understanding of table joins is a must. As the core of relational database, the joins define the relationship between two (or more) tables (When we use the term tables, we also mean objects and visa versa. That is possible because Hibernate is ORM - we create objects with the information from the tables). The need to understand the relationships comes from the necessity to define the fetch type. Their role is simple but important. The fetch types decide whether to load all of the relationships of an object or not when it is initially fetched.

Whenever you specify a relationship (OneToOne, OneToMany, ManyToMany) you also have to specify the fetch type. By default the fetch type is set to “Lazy”. That means that whenever you load up something with Hibernate query, Hibernate will not load the information related to other tables (unless explicitly asked) and if you try to access it you will get a NullPointerException. If the “Lazy” fetch type doesn’t allow Hibernate to load the relationships of an object then the “Eager” fetch type does the opposite. It load all of the relationships related to the object. If the “Lazy” fetch type can (and possibly will) throw a NullPointerException why use it? The reason is simple – it’s faster. Imagine that you want access to an object but the object has 10 relationships to other objects. If you use “Eager” fetch method you have to load all of the objects and that’s time consuming. And what happens if every one of those 10 objects has other 10 more relationships to other objects? In conclusion the “Eager” fetch method is more convenient in some ways but can lead to massive delay in the application. With “Lazy” fetch type the developer has to write more code but in the end it’s faster and more efficient.

To ease the usage of Hibernate, it uses a HQL - special query language that’s close to SQL but is fully object-oriented and can understand the OOP principles like inheritance, polymorphism and association. HQL is also case-insensitive (exceptions are the java classes and properties and column names). That means that sElect is the same as SELECT. [14]

|  |  |
| --- | --- |
| SQL | HQL |
| SELECT \* FROM users | from Users |
| SELECT \*  FROM accounts a, users u  WHERE  a.owner\_id = u.user\_id  and u.user = '18' | from site.Account account  where  account.owner.id.age = ‘18’ |
| SELECT distinct CONT\_NAME  FROM CONTINENT  INNER JOIN COUNTRY on  CONTINENT.CONT\_ID = COUNTRY.CONT\_ID  WHERE AREA > 10000 | select distinct cont.name  from Continent cont joincont.countries ctry  where ctry.area > 10000 |

Table 3: Comparison between SQL and HQL

With all those features of Hibernate, there is no reason not to use it. And the people from the Spring community come one step further in simplifying its use. With the inclusion of the spring boot starter jpa you automatically get the jpa specification and hibernates implementation as a default, auto reading the data base configurations from the application.properties file and auto creation of entities as tables. Another big improvement is the Spring CrudRepository which builds up on top of the ordinary Hibernate and provides default implementations for the basic CRUD operations i.e. you don’t have (even thou you can) write hql/sql queries.

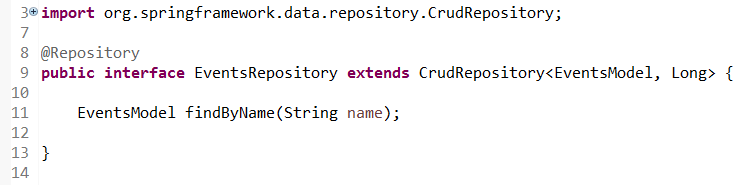


Figure ABOUT CRUD REPOSITORY

### CSS3

CSS3 is the latest standard for CSS. While it supports all of the previous versions of CSS it also adds a lot of new features such as rounded corners, animations, shadows and a few new layouts – multi-columns, grid layout. Due to holdbacks of secondary features CSS level 2 needed 9 years to reach the Recommendation status. Thus the CSS Working Group of the W3C took a different approach. They divided CSS in smaller components called modules. Each module is now independent part of CSS and it is developed at its own pace. Some of the most important CSS3 modules are: Selectors, Box Model, Text Effect, 2D/3D Transformations, Animations, Multiple Column Layout and User Interface.

Sadly a big part of the functionality of CSS3 is not supported by some of the older versions of the web browsers. This leads to the additional use of JS to recreate the missing functionality of the unsupported tags. [8]

### Bootstrap 3, PrimeNg, Semantic UI

Bootstrap (originally named as Twitter Blueprint) is free, open-source framework created for the purpose of easing the creation of websites and web applications. It contains HTML and CSS based design templates for various elements such as: typography, forms, tables, buttons, navigation and many other interface components. One of the main features of bootstrap is the responsive design. With its flexible grid structure the framework is perfect for web applications that look good on all of the different devices. Bootstrap defines 4 display resolutions categories:

* Xs – extra small (under 768px width, for phones)
* Sm – small (over 768px width, for tablets)
* Md – medium (over 992px width, for desktops)
* Lg – large (over 1200px width, for larger desktops)

The categories above can be combined to create more dynamic and flexible layouts. [9]

“Semantic is a development framework that helps create beautiful, responsive layouts using human-friendly HTML”. [10] https://semantic-ui.com/ . Semantic UI is a modern front-end framework that helps the developers to create good looking and responsive layouts. It is powered by LESS and jQuery and enables the users to use various already made styles for buttons, inputs, fields, containers and many more.

PrimeNg is one of the riches UI libraries for Angular2. It contains in itself vast collections of UI components that can be used together with Angular. Some of the most useful components include dropdown menus, Growl i.e. notification messages, calendars, accordions, confirm dialogs and data table grids that are easy to use and implement. And another big bonus is that it is also opensource which means it is always up to date with the latest’s trends and all of the bugs are fixed in a timely manner.

### Angular 2/5

As previously mention the choice of the frameworks that are being used is really important when you start a new project. For the back-end of the application there is Spring and as for the front-end there is Angular2. A few years back AngularJS was without a doubt the most popular and preferred choice for framework when dealing with front-end but with the coming of Angular2 and TypeScript that change. Currently Angular2 together with TS is the most desired combination for front-end development amongst the front-end developers. TypeScript is the recommended language for development of Angular2 application but as the language is a superset of JS you can just as easily use JavaScript (both ECMAScript 5 and 6) for your application. When we talk about web development one of the first architectures that come to mind is the MVC but in fact Angular2 is not a MVC framework. It is a component-based framework. What that mean is that the application is a tree of loosely coupled components that interact with each other. This is very useful when building a single page application (SPA) – you don’t have to reload the whole page just to update some information inside the page, you just should reload the component that contains the said information. Also, like Spring, Angular2 presents us with a convenient dependency injection so that developer can easily use shared service classes that contain the commonly shared business logic. In Spring, there are a few ways to use dependency injection but in Angular2 there is only one - objects can be injected only via constructors. When we talk about the architecture of the framework its worth to mention that it has 8 major blocks – see the figure below.

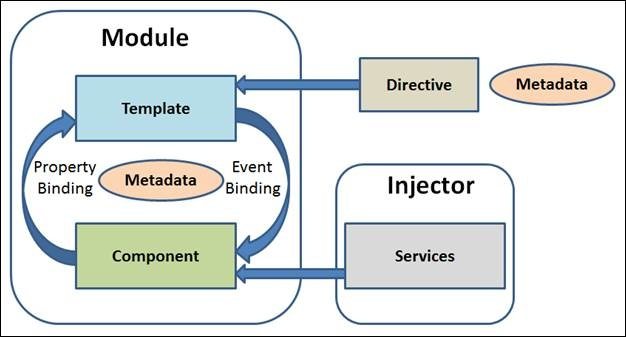


Figure: 8 major blocks of Angular2 https://qph.ec.quoracdn.net/main-qimg-feec1638f923cdce67e5935f3d626f18-c

* Module – the modules can be seen as classes, which perform a single, particular task. To define a class, you must use not only the key word class but also the key word export.
* Component – View of the application and logic on the page. It must be annotated with @Component so it is register as component and there can be only one component per DOM element.
* Template – This is the actual view of the component or with other words said it is the HTML.

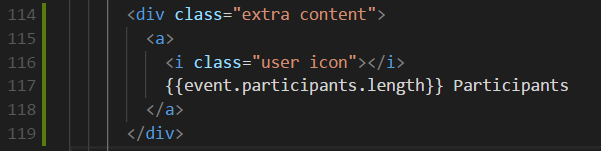


Figure # : Part of the template

* Metadata – it is used to extend the functionality of the class. An example is the @Component decorator (annotation).
* Data Biding – this is the feature that connects the view and the model. There are 4 ways supported in Anuglar2 for binding data: Property Binding, Event Binding, Interpolation and Two-Way Binding. Later in the thesis the four way will be further discussed and showed how will be used in the context of the web application that is the object of the thesis.
* Directive – Directives are used to add behavior to the DOM elements. There are three types: Directive-with-a-template, Structural, Attribute. They will also be discussed further in later chapters.
* Service – class with containing any function, feature with specific purpose. Often used with functionality that can be shared between multiple modules in the application.
* Dependency Injection - It allows injecting a dependency as a service throughout the web application. It also increases the maintainability, reusability and testability.

### Maven

Maven is a software management tool that simplifies and makes the building of the projects faster. With Maven, you can easily manage the builds, the documentation, the releases and the dependencies of every application. Maven project structure and contents are declared Project Object Model (POM) file in the form of an xml – pom.xml. Maven uses convention over configuration what that means is that the developers doesn’t need to create the build process themselves – the tools automatically creates a sensible default structure. Some of the features of maven include the following, but also many more:

* Very simple project setup.
* Dependency management.
* A large repository of libraries.
* Release management and distribution publication – can be integrated with version control systems.
* Backward Compatibility.

### IDEs

Today there is a large variety of IDEs and the right choice of IDE can influence the development of the project throughout the whole project lifecycle. The web based application that is the focus of this thesis can be split in two – front-end part and back-end part. Taking in to account the technologies and languages that are going to be used the best IDEs for the job is without a doubt IntelliJ IDEA that gives us the chance to write both the front-end and back-end in the same place, but sadly it has one major flaw and that it is commercial and you have to pay for it. A good alternative is to use STS for the back-end and Visual Studio Code for the front-end.

The Spring Tool Suite can be said that is built on top of is an Eclipse and that is customized for developing Spring applications. It provides a ready-to-use environment to implement, debug, run, and deploy your Spring applications, including integrations for other technologies that are going to be used in the project such as Git and Maven and a build-in tomcat that is optimized for Spring.

Some of the key features include:

* Recognition and understanding of Spring Projects - parses the configuration files and displays detailed information about the beans.
* Comprehensive Validations for your Spring Configuration – with combination of the previous feature the STS apples automatically validation in the configuration that show errors directly in the IDE.
* Extensive Refactoring Support – the IDE supports not only the usual well know Java refactoring ways, but also new Spring elements refactoring.
* Code Assists – no matter where you are in the project, the STS provides a meaningful content assist and quick fixes for common problems.

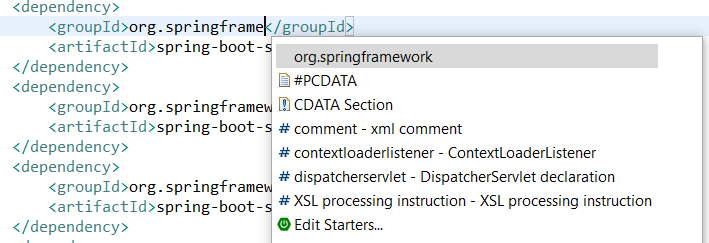


Figure #: Code Assist in an XML file

* Graphical Views – The STS provides various relevant to the projects views.
* And many, many more.

Developed by Microsoft, Visual Studio Code is a source code editor with powerful developer tooling such as code completion and debugging. Some of the key features include, but are not limited to:

* Easy editing, building, and debugging – increases productivity by syntax highlighting, bracket-matching, auto-indentation, box-selection. Includes built-in support for IntelliSense code completion for more serious programing logics, interactive debugger and supports Git for fast working with source control.
* Highly Customizable – the IDE is very customizable; you can add any third-party extension that you want with almost no extra configuration. Being an open source project, you can also join the community and contribute to the improvement of the code base.
* Build for web – the IDE comes with enriched built-in support for Node.js development with JavaScript and TypeScript.

### Git

Git is a widely-used version control system which tracks changes in computer files and coordinating work on those files among multiple people. One of the main selling points of Git is its architecture distributed architecture. Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes. Other advantages of Git are the enhanced performance, security and flexibility.

## Solution to the problem

To complete the objective of the bachelor thesis we have decided to use the technologies, libraries and programing languages that are described in the previous section of the thesis. To achieve a nice, clean and intuitive module for templates, everything has to be kept as simpler as it can be while keeping as much of the functionality.

When the site is loaded in front of the user will be display a login form that prompts the user to log in the site using his/her profile, if the user is new to the site he/or she can also create a new account. It’s important to mention that the content of the site should not be visible without being logged in the site. To make the user experience as best as it can be the registration form should not include all of the fields that can be written by the user, but only the essentials. Also there should be basic validation for the fields in the form. After every field is correctly written for the registration and the user clicks the register button, a new record should be written in the data base containing the said information. If record is successfully saved the user should be redirected back to the logging page where he or she can immediately, without needing to confirm the registration through email or needing to wait for administrator’s approval, log in the site and begin using its functionalities. The log in form should be simple username/password login form that checks the database for existing user. If the username or the password is incorrect the logging in should not be successful. If they are correct the use should be redirected to the content of the site.

After successful login the user should be greeted by a nice looking but in the same time simple interface to all of the main functionalities of the site:

* Home page
* Event page
* Destination page
* Chat page
* Messages page
* Search functionality
* Drop down menu to control user

The home page should display the information of the user. This information includes personal information, profile picture (if there is a set on), interests, a status input, list of all the events that the user has been to or is going to be, and the next event that the user is going to go to.

The event page should contain lists with the past events and upcoming events. These events should have some kind of information about them displayed such as date, picture, number of participants etc. and functionality to redirect to the said event when clicked on it. There should also be a functionality that enables the user to create a new event. When the button is pressed, the user should be redirected to a new page where he or she would be prompt to type the necessary information for the creation of a new event such as title, description, date, meeting point and destination. Basic validation of the input should also be included here. When a user clicks on one of the events, he or she should be redirected to the information page of the said event. Here should be displayed the most important information about the event such as where and when will be held the event and who is going to the event. The user should have the ability to join the said event and/or write comments for the event.

In the Destination page the user should be able to see all of the available destination as well he or she should be able to create new destinations. When creating new destinations the user should be able to write the necessary information for the creation of a new destination such as title, description, location and a picture. Similar to the events page, when the user clicks on a destination he or she should be redirected to the destination information page, where all of the information for the destination can be found. In the destination information page the user should be able to review the information as well as write comments and add more pictures of the destination.

Chat page – this should be a global chat where all of the users can come together and chat with each other.

The Messages page should enable the user to write new massages to other user and receive/ read messages sent from other users.

The search functionality should enable the user to search other users, events and destinations and redirect the user to them.

The Drop down menu should be a standard dropdown that shows two additional functionalities – logging out of the site and editing the personal data of the user. The editing personal data functionality should be done in a new page where the user should be able to add or change his photo or personal information.

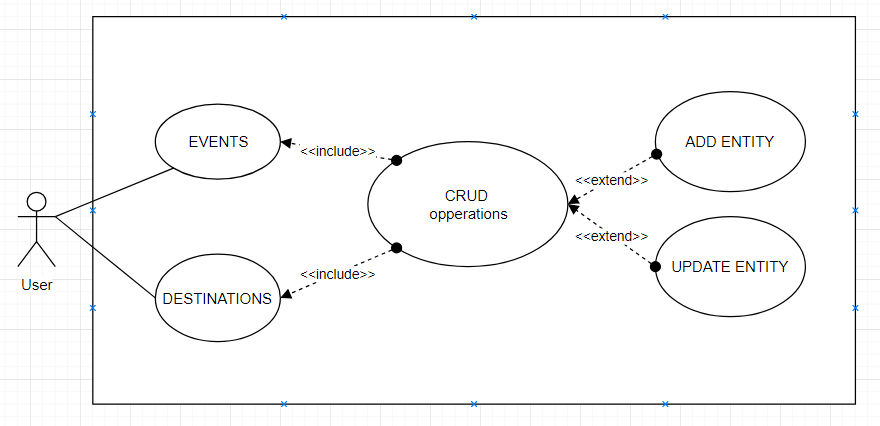


Figure #: UML Use Case diagram for the main

The presented Use Case Diagram shows a basic representation of the needed functionality for creating and editing the destinations and the events. The functionalities are supported by the operations that they use.

As for the functionality of the site, the following flow char diagrams show some of the algorithms that help to achieve the functionalities described at the beginning of the chapter.

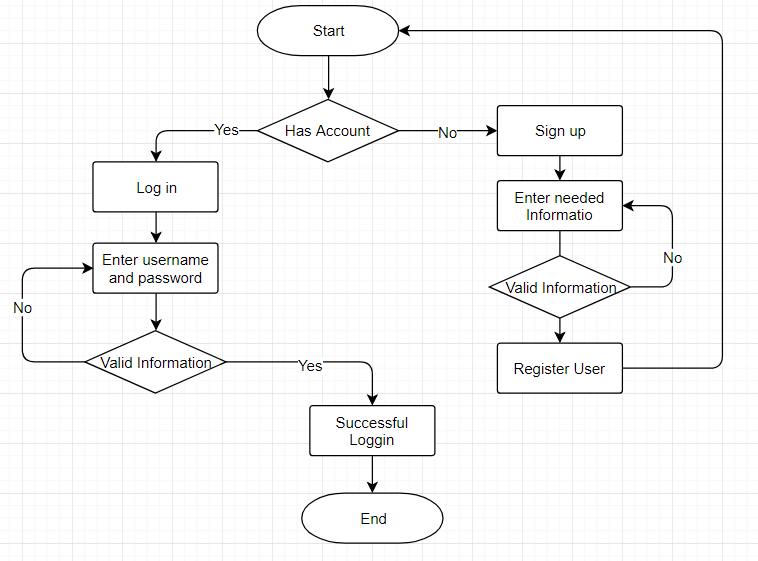


Figure #: Flow Char diagram of the algorithm for sign up and sign in the site.

The main goal of the algorithm above will be to check if a user has an account. If the user doesn’t have one, one must be created to enter the site. To do that the correct information should be verified and stored in the database at which point if the information is successfully recorded the user will be ask to write the username and password for the account. If the username and the password are correct the user will proceed in entering the site. It is worth mentioning that there should be no way possible for unregistered user to enter the site.

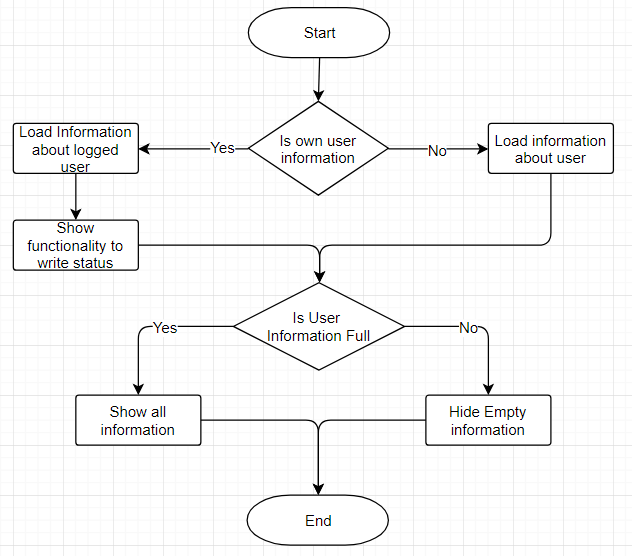


Figure #: Flow Char diagram of the algorithm for loading the user information page.

The main goal of the algorithm above will be to load the correct information in the user information page. The first check that it should make is if the logged user is trying to view his/hers own profile information or the information of another user. If the logged user is trying to look at the information page of his account, the correct info should be loaded and an additional functionality for writing statuses should be displayed. In both cases, when the user is trying to see his own information or when he/she is trying to see another users information if the information in the DB is not full i.e. the said user hasn’t written all his information some of the fields in the web page will be omitted and not displayed.

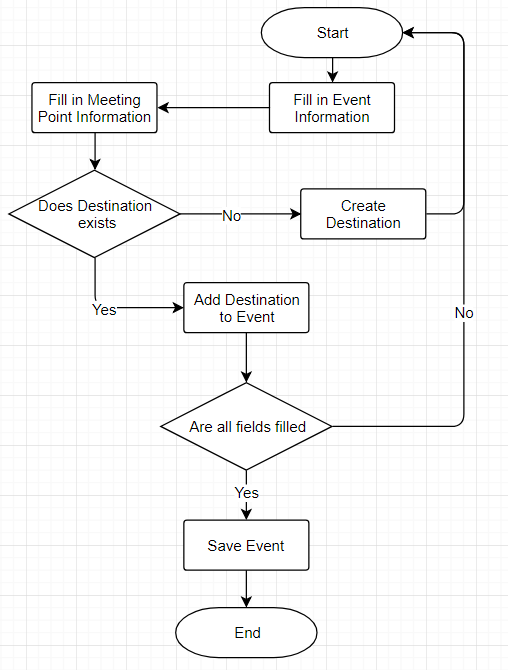


Figure #: Flow Char diagram of the algorithm for creating new Events.

The main goal of the algorithm above will be to allow the user to create a new Event. First all the fields should be filled, that includes the event information, meeting point information and destination. If the user wants to create an event that has a destination that has yet to be added to the data base of the application, the said destination should be added to the database. Before the saving of the event, verification must be made that all of the fields are not empty; if any are empty the event should not be saved in the database.

## Web Application Architecture

### MVC

“Software Architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.”[ANSI/IEEE Std 1471‐2000]. Being a vital part of the web application a poorly selected architecture leads to low performance and availability, insufficient maintainability and expandability. On the other hand, if the right architecture is being used it makes the system more understandable and flexible.

One of the most commonly used architectures (design pattern) is MVC. It is a 3-layer architecture with data layer, presentation layer and business layer. The data layer (Model) represents the core functionality of the system. It is also responsible to the DB manipulation. The presentation layer (View) renders the results of the requests in the desired format. The business layer (Controller) is responsible to connect the Model and the View. It’s also responsible for the actions of the users.

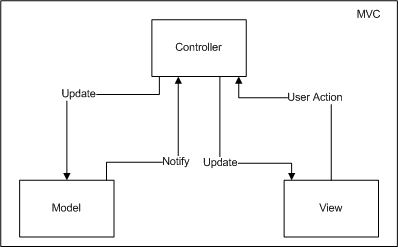


Figure 12: Simple representation of MVC

The MVC design pattern has 3 major benefits which make it one of the most popular and used design patterns in the web programming. The first one is the separation of concerns. The separation in different parts allow the developers to reuse the business logic and on a different units of the program without affection others. The second one is the specialization and focus of the developers. With clearly defined front-end and back-end the developers can focus solely on their tasks of the project i.e. UI developers work on the UI without caring how the back and works and visa versa – the back-end developers don’t need to know how the front-end works. The third benefit is a combination of the first two – parallel development. Without being bound the presentation logic and the business logic can be developed and updated separately from each other and without slowing each other down. With that being said, MVC is simple but very effective design pattern. [15][16]

### Architecture of the back-end

As shown above MVC fits perfectly for the needs of the back-end of the site that is the objective of the master thesis because of its separation of concerns.

The Model layer is composed of the following components:

* MySql Database - object-relational database management system that stores the information
* Hibernate – ORM that uses JPA to manage the communication between the module and the DB
* Entities – Java classes that show the exact structure of the DB and store the information that is pulled from the DB.

The Presentation layer is not the typical presentation of most MVC architectures that use technologies such as:

* JSP – Technology that allows the developers to create dynamic web pages.
* Servlets – Java Classes that have doGet and doPost methods that respond to the clients requests.
* Struts2 Actions – Separate methods and classes that transfer date from the request through to the view and determine what result should be rendered in the view.

In the architecture that will be used for the application that is the object of this thesis, the presentation layer will consist of JSON objects that are consumed by another project that will be written in Angular2.

The Controller layer is composed of the following components:

* Spring Beans – Java classes that encapsulate a set of objects in one object.
* RestController – this is a specialized version of the controller in the usual MVC, this controller simply populates and returns an object and the object data will be written directly to the HTTP response as JSON.

Figure 13 shows graphically, how all of the above mention components are positioned in the MVC design pattern.

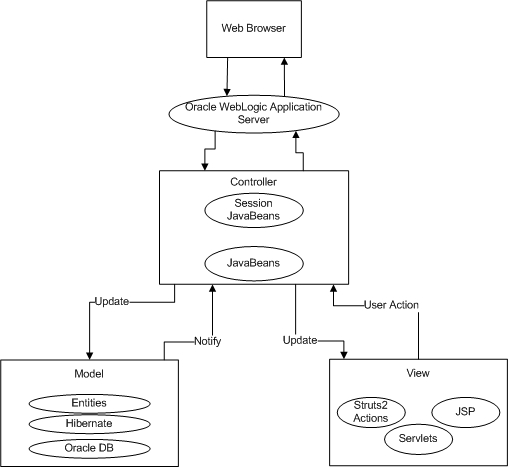


Figure 13: MVC for back-end (TODO: da se preprevi)

### 2.3.3 Architecture of the front-end

As previously mentioned the front end will be written in angular2 and Typescript.

## Data organization

To meet the requirements of the object of the master thesis the web based system needs a place to store the information so a database is need. . For that purpose, is used a local database MySQL. This database is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons like:

* MySQL is released under an open-source license. So it is free for everyone who want to use it.
* MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
* MySQL uses a standard form of the well-known SQL data language.
* MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
* MySQL works very quickly and works well even with large data sets.
* MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but can be increased (if the operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
* MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments. [18]

### ER model of the DB

The presented relationships between the different DB tables show the database structure for the module. For clarity of the ER model the types of the attributes are not included.

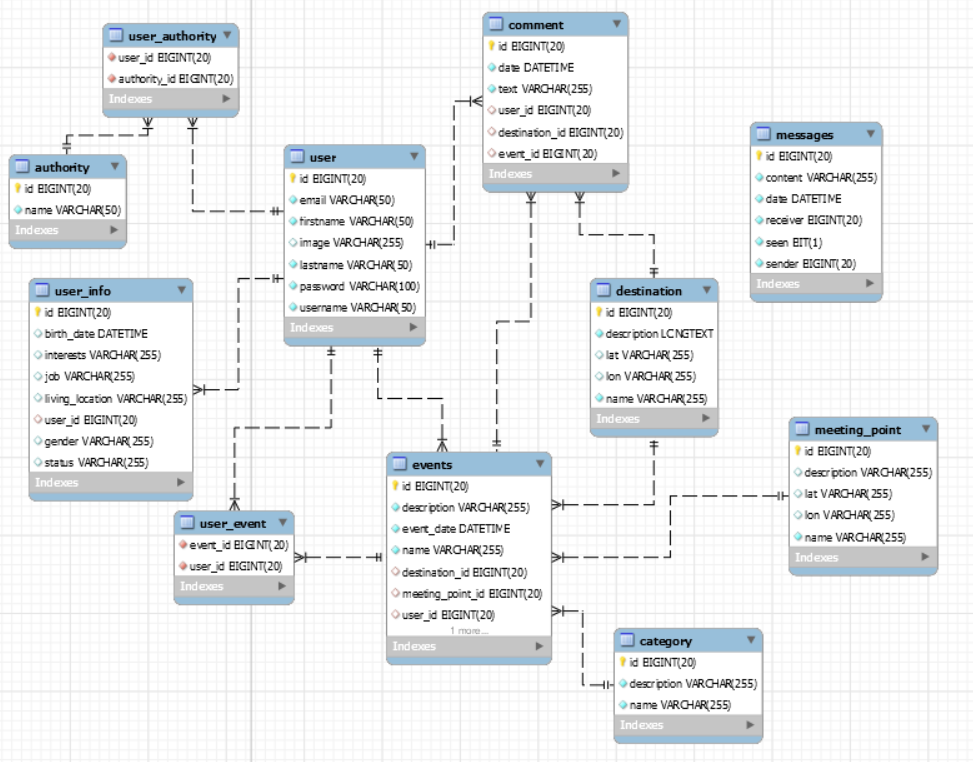


Figure $: ER model of the DB

On figure 5 can be seen the logical structure of the database of the application. This diagram is visualizing how the various data in the application is related to each other. The name of database of the application is “outdoorsproject”. It contains 11 different tables with data about the users, the events and the destinations and everything necessary for the needed functionalities to work correctly.

A relationship between tables is an important aspect of a good relational database. Well defined relationship means that developer will not facing problems when he try to insert, update, or delete records in related tables. This chapter discusses in detail the tables, with which the database is structured, as well the relationships between them.

In the followed tables are represented all database table structures. Each row of these tables represented each column of the database table. Each table has five columns. The first column only shows the number order of the properties. The Second column contains information about names of the table columns. The third column shows the type of each table column. From the fourth column can be understood if the table column can have null data or not. And the last column shows if the column have some extra configurations.

Table 1 shows the structure of the database table – events. The primary key is “id” (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. What that means is that when a new record is created in the table, the column will auto increment with constant number set beforehand for the database. Also the column is marked as Not Null, meaning that this column can never be null. The columns “Description”, “Event\_date” and “Name” contain the information concerning the event directly. “Description” and “Name” are both self-explanatory and are of type varchar (255) meaning that they will contain string information. The “event\_date” column will contain the date of the event and is of type datetime. All of these columns are marked as non-null because they are necessary for the event.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | Bigint(20) | No | Auto Increment |
| 2 | Description | Varchar(225) | No | - |
| 3 | Event\_date | datetime | No | - |
| 4 | Name | Varchar(255) | No | - |
| 5 | Destination\_id | Bitint(20) | No | - |
| 6 | Meeting\_point\_id | Bigint(20) | No | - |
| 7 | User\_id | Bitint(20) | No | - |
| 8 | Category\_id | Bigint(20) | No | - |

Table 1: Events

This table can be considered as one of the main tables in the application that is the object of this thesis. It contains all of the necessary information (both directly and “indirectly”) needed to visualize the events. The “indirect” information that was mention previously is the last three columns of the “events” table. They represent relationships with other tables and also cannot be null.

“Destination\_id” is a relationship with the “Destination” table, it is a foreign key. The relationship here is many-to-one. A many-to-one association is one of the most common kinds of association where an Object can be associated with multiple objects. Many-to-one mapping means that one row in a table is mapped to multiple rows in another table. In this case the event has one Destination, but this same Destination can be also associated with another Event as well.

“Meeting\_point\_id” marks the relationship with the “Meeting\_point” table, it is a foreign key. The relationship here is one-to-one. A one-to-one association is similar to many-to-one association with a difference that the column will be set as unique. In this case an event record (object) can be associated with a single meeting point\_ record (object).

“User\_id” marks the relationship with the “User” table. The relationship here is many-to-one and represents the “owner” of an event. So in this case the Event can have one owner but this same owner (user) can be associated with another Events. The column is also part of a many-to-many relationship again with the User table to define the participants that join the event. In this case many events can have many participants or with simpler words any user can join any event and any event can be joined by any user. To achieve the desired affect the two tables – “User” and “Events” are joined with a third table called “User\_events”.

“Category\_id” is a relationship with the “Category” table, it is a foreign key. The relationship here is many-to-one. So in this case the Event can have one category but this same category can be associated with another event.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | bigint(20) | No | - |
| 2 | Description | Varchar(225) | No | - |
| 3 | Name | Varchar(225) | No | - |

Table 2: Category

Table 2 shows the structure of the database table – Category. The primary key is “id” (also marked in the table below). The column is not marked as auto increment because there are a limited number of categories that are inserted in the database and the user will not be able to change that, making the auto increment unnecessary. Also the column is marked as Not Null, meaning that this column can never be null. The columns “Description” and “Name” contain the information concerning the category directly and are self-explanatory. Both columns are of type varchar (they will contain strings) and are also marked as not null.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | bigint(20) | No | Auto Increment |
| 2 | text | Varchar(225) | No | - |
| 3 | date | datetime | No | - |
| 5 | Destination\_id | Bitint(20) | Yes | - |
| 6 | Event\_id | Bigint(20) | Yes | - |
| 7 | User\_id | Bitint(20) | No | - |

Table 3: Comment

Table 3 shows the structure of the database table – Comment. The primary key is “id” (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. What that means is that when a new record is created in the table, the column will auto increment with constant number set beforehand for the database. Also the column is marked as Not Null, meaning that this column can never be null. The columns “text” and “date” contain the information concerning the event directly. “Text” is the column that contains the content of every comment a user makes in the site. It is of type varchar and it cannot be null. The column “date” represents the time at which point the comment is made by the users. It is of type datetime and also cannot be null.

This table can also be considered as one of the main tables in the application that is the object of this thesis. It contains all of the necessary information (both directly and “indirectly”) needed for all of the comments in the site. The “indirect” information that was mention previously is the last three columns of the “Comment” table. They represent relationships with other tables and also cannot be null.

“Destination\_id” is a relationship with the “Destination” table, it is a foreign key. The relationship here is many-to-one. As previously mentioned a many-to-one association is one of the most common kinds of association where an Object can be associated with multiple objects. In this case here the comment is associated with one destination, but this same destination can be also associated with other comments as well. The field is not marked as not null, because one comment can be either associated with a destination or an event – it cannot be associated with both.

“Event\_id” is a relationship with the “Events” table, it is a foreign key. The relationship here is many-to-one. In this case here the comment is associated with one event, but this same event can be also associated with other comments as well. The field is not marked as not null, because one comment can be either associated with a destination or an event – it cannot be associated with both.

“User\_id” marks the relationship with the “User” table. The relationship here is many-to-one and represents the creator of the comment. So in this case the comment can have one creator (user) but this same creator (user) can be associated with other comments.

.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | bigint(20) | No | Auto Increment |
| 2 | Description | Varchar(225) | No | - |
| 4 | Lat | Varchar(255) | No | - |
| 5 | Lon | Varchar(255) | No | - |
| 6 | name | Varchar(255) | No | - |

Table 4: Destination

Table 4 shows the structure of the database table – Destination. The primary key is “id” (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. What that means is that when a new record is created in the table, the column will auto increment with constant number set beforehand for the database. Also the column is marked as Not Null, meaning that this column can never be null. The columns “Description”, “Lat”, “Lon” and “name” contain the information concerning the destination. All of these columns are marked as not null and are of type varchar. “Lon” and “Lat” represent the exact coordination of the destination. This table can also be considering as main table for the application because of all the relationships that it has with other tables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | bigint(20) | No | Auto Increment |
| 2 | Description | Varchar(225) | No | - |
| 4 | Lat | Varchar(255) | No | - |
| 5 | Lon | Varchar(255) | No | - |
| 6 | name | Varchar(255) | No | - |

Table 5: Meeting\_Point

Table 5 shows the structure of the database table – Meeting\_Point. The primary key is “id” (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. Also the column is marked as Not Null, meaning that this column can never be null. Similar to the columns in the Destination table, the columns here are also “Description”, “Lat”, “Lon” and “name”. They contain the information concerning the meeting point of every event. All of these columns are marked as not null and are of type varchar. “Lon” and “Lat” represent the exact coordination of the destination. It is important to mention that because of the one-to-one relationship with the “Events” table, every time a new event is being made a new record will be inserted in the table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | bigint(20) | No | Auto Increment |
| 2 | Content | Varchar(225) | No | - |
| 3 | date | datetime | No | - |
| 4 | receiver | Varchar(255) | No | - |
| 5 | sender | Varchar(255) | No | - |
| 6 | seen | Bit(1) | No | Default value 0 |

Table 6: Messages

Table 6 shows the structure of the database table – Messages. The primary key is “id” (also marked in the table below). It has an extra configuration that show that the column is marked as auto increment. Also the column is marked as Not Null, meaning that this column can never be null. The other columns in the table are “Content”, “date”, ‘receiver”, “sender” and “seen”. The “Content” column is self-explanatory – it contains the content of the messages. It cannot be null and it is of type varchar. “Date” represents the date and time at which the massage is being send. It’s of type datetime and also cannot be null. The “seen” column is of type bit and it also cannot be null. The column represents if a message has been seen by a user. It has a default value of 0, meaning that it has not been read yet. The columns “receiver” and “sender” represent the user that have send the massage and receive it. To limit the unnecessary relationships from and to the Users table these two columns are not foreign keys but instead are the usernames of users.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | bigint(20) | No | Auto Increment |
| 2 | email | Varchar(225) | No | - |
| 3 | firstname | Varchar(225) | No | - |
| 4 | Lastname | Varchar(225) | No | - |
| 5 | image | Varchar(225) | Yes | - |
| 6 | Password | Varchar(225) | No | - |
| 7 | username | Varchar(225) | No | Unique |

Table 7: User

Table 7 shows the structure of the database table – User. The primary key is “id” (also m arked in the table below). It has an extra configuration that show that the column is marked as auto increment – meaning with every record that is inserted it will increment. Also the column is marked as Not Null, meaning that this column can never be null. The other columns in the table are “email”, “firstname”, ‘lastname”, “image” and “password” and “username”. All of the columns are of type varchar and all of them except “image” cannot be null. The image column represents a string that is the name of the profile picture of the user. The “username” column has an extra configuration and that is that it should be unique – meaning that there can be only unique usernames in the site. This table can be considered as main table, because it contains the login information for each user. When user log in to the site, the data entered by user will be compare with the “username” and “password” columns from this table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | User\_id | bigint(20) | No | - |
| 2 | Authority\_id | bigint(20) | No | - |

Table 8: User\_authority

Table 8 shows the structure of the database table – User\_authority. The idea behind this table is that it helps with the many-to-many relationship between the “User” table and the “Authority” table. It contains two columns – “user\_id” and “Authority\_id”. They are both of type bigint and also cannot be null.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | bigint(20) | No | Auto Increment |
| 2 | name | Varchar(50) | No | - |

Table 9: Authority

Table 9 shows the structure of the database table – Authority. The primary key is “id” (also marked in the table above). It has an extra configuration that show that the column is marked as auto increment – meaning with every record that is inserted it will increment. Also the column is marked as Not Null, meaning that this column can never be null. The other column in the table is “name” – also marked that it cannot be null. The table is used to set security roles for the application. It’s worth mentioning that currently there is only one active role in the application and that’s for normal users, but if the need arises an admin role can be easily added.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | User\_id | bigint(20) | No | - |
| 2 | Event\_id | bigint(20) | No | - |

Table 10: User\_event

Table 10 shows the structure of the database table – User\_event. The idea behind this table is that it helps with the many-to-many relationship between the “User” table and the “Event” table. It contains two columns – “user\_id” and “event\_id”. They are both of type bigint and also cannot be null.

The last table shows the structure of the database table – User\_info. This table can also be considered as one of the main tables in the application because it contains all of the extra information about the users. The primary key is “id” (also marked in the table above). It has an extra configuration that show that the column is marked as auto increment – meaning with every record that is inserted it will increment. Also the column is marked as Not Null, meaning that this column can never be null. The columns “job”, “intrests”, “living\_location”, “gender”, “status” are all of type varchar and can be null as they are not obligatory for the site to function correctly. The column birth\_date is used to store the users birth day, it’s of type datetime and also have null value.

“User\_id” marks the relationship with the “User” table. The relationship here is one-to-one because one user can have only one specific user information about itself and vice versa.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | Name | Type | Null | Extra |
| 1 | id (primary key) | bigint(20) | No | Auto Increment |
| 2 | job | Varchar(225) | Yes | - |
| 3 | birth\_date | datetime | Yes | - |
| 4 | interests | Varchar(255) | Yes | - |
| 5 | Living\_location | Varchar(255) | Yes | - |
| 6 | gender | Varchar(255) | Yes | - |
| 7 | User\_id | Bitint(20) | No | - |
| 8 | Status | Varchar(255) | Yes | - |

Table 1: User\_Info

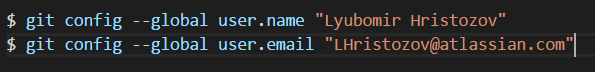
It is also worth mentioning that there are some auto-generated tables that help with the security of the site, but I’ll not go into more details about them because all of the information in them is not managed by the code directly.

# Practical solution

As mention earlier in the thesis the web application consists of two separate projects – one for the back-end i.e. the server side and one for the front-end i.e. the client side. All the technologies, libraries and programing languages listed in the architecture and design part of the thesis are being used for the implementation of both projects. In this chapter the implementation will be thoroughly reviewed and explained. Because of the unusual architecture – having two separate projects for the front-end and the back-end, a special attention will be given to the setup of the environment. Also, each of the layers in both projects will be reviewed and it will be explained how they interact with one another.

## Setting up the Workspace

Since the web site is being split in two separate projects the first step in the development is the correct configuration and installation of all of the necessary software that was mention above and of course all of the necessary helper software for those technologies. In this part of the thesis, a short walkthrough will be given on the necessary steps to set up the workspace. Since we are planning to use Java as a main back-end technology first we need to have the Java Development Kit (JDK) installed on the computer that we will be using to develop the web based system. The installation of JDK is free and anybody can download it from the Oracle official site and install it. The next step is the installation of a software versioning and revision control system. We will be using Git (Github) because it offers an easy to use repository of current and historical versions of the projects files. The installation itself is very straightforward and easy. For windows user there is a convenient to use installer – for other OS there is a step by step installation guides. The only configuration that you have to do is to configure your username and email – see figure ###



Figure##: Bash commands to configure username and email

After this is done you can use it as such and only use it as a local repository, but that is not very effective because if you go to another computer you cannot download and use your code very easy. So to avoid this you have to make yourself a repository. For this particular project I have selected Github. In the official site everything is described step by step how to start so I’ll not be going in details. The next step in the setting up of the workstation is the installation of an IDEs. As said in the previous section we will be using STS for the server side project (Spring project) and Visual Studio Code for the client side (Angular2/5 project). As both of these IDEs are free to use, you can download them from their official sites and use the installation guides there to install them. The next step in the setup is the installation of Angular. To install Angular we need to install the Angular CLI. But to do this we actually need to install something else first and that is npm – this is a package manager for JavaScript and the world’s largest software registry. Luckily this is very easy because npm is distributed with Node.js - which means that when you download Node.js, you automatically get npm installed on your computer. So we simply have to download node.js and install it. After all of this is done the installation of angular itself is very easy with a singular command:



Figure #: Command to install Angular CLI

The next important thing that is needed for the project is the software management tool. As mention previously the project will use Maven. As a free to use Maven can be directly download from the official Apache site. The only configuration that has to be made is to create 2 new variables (M2\_HOME and MAVEN\_HOME) in the Windows environment and point them to the installation folder of Maven and of course them to the PATH variable. If you already have maven, but you are not sure what version you are using or if you doubt, that the installation is done - you can simply run “mvn –version” in the command prompt. If everything is right you will see information about your maven version, the maven directory, the version of the Java you are using and the location of the java.

So now we have all the necessary software installed so we can proceed with the setup of the project themselves. First is the Spring project. Luckily the Spring Tool Suite (STS) is design to help its user with such task so we can simply create a Spring Boot Maven project directly from it - File > New > Spring Starter Project. This will open the creation wizard where you will be prompt to select the standard configurations of the project such as Group ID, Artifact ID, Root package, build tool – in our case Maven and etc. On the next page you will be asked to add dependencies that you will be using. It is not necessary to select all any dependencies – they can be added at any time from the Pom.xml file. With this a new project will be created and if everything is correctly set, this is all for the setting up of the Spring Boot project. In the next chapter the file stricter and all of the dependencies will be reviewed.

The setting up of the Angular 2/5 project is fairly simpler. Since we are using the Angular CLI we only need to input a command in the command line and everything will be done for us automatically (See figure #). The next chapter will also include the structure and dependencies of the front-end project.



Figure #: Setting up of Angular project

## File Structure and dependencies of the application

In this section a short explanation of the file structure of both projects will be given. First will be the back-end. Being a Spring Boot project the files are organized in the standard application assembly (packaging) way. The packages can be divided in 2 types- module and full-fledged applications. A Spring module is a collection of components of the same type such as service or repository or etc. The module also has to have the necessary type of deployment descriptors. If there are no deployment descriptors, the module can be deployed as a stand-alone module. So in the project there are 6 packages:

* Config – as the name implies in this package will be stored all of the configuration files of the application such as the SecurityConfig, CorsConfig and AuthorizationServerConfig files.
* Controllers – again self-exploratory name, this package will contain all of the controllers of the application for example the EventsController, the DestinationConroller and CommentController.
* Models – this package contains the Entities of the application. Examples are the EventsModel, the DestinationModel and the CommentModel.
* Repositories – as we will be using hibernate and spring data we need repositories for the various Entities. Examples for repositories are the EventsRepository, the DestinationRepository and the CommmentRepository.
* Resources – the resource package contains plain old java objects (POJOs). We need them for easier manipulation of data. Examples are the EventsResource, the DestinationResource and the CommentResource.
* Services – as the name implies the package contains the service classes of the application – exmaples are the EventsService, the DestinationService, and the CommentService.

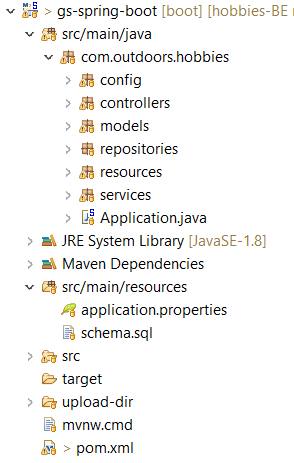


Figure ##: Structure of the Spring Project --- moje da mahna (nadolu e spomenata)

All of the file in these packages share a common good naming practice – the first word of the name is the name of the entity and the last word is the name of the type i.e. service/repository etc.

First it is important to mention that when the creation wizard was creating files it made one file that defines the project as a spring boot project – Application.java. This file can be viewed as the starting point of the application – it contains the main method and also there is the @SpringBootApplication annotation that marks the class and the project. All of the dependencies in the project are defined in the pom.xml file. Here are auto-defined from the creation wizard the spring-boot-starter-web and spring-boot-starter-data-jpa dependencies. Another important dependency that must be added is the dependency for the database that the application is going to interact with - mysql-connector-java. Of course there are other dependencies that are added but they will be explained and reviewed later in the thesis. Another equally important file is the application.properties file. As the name suggests this file contains the properties of the project such as the server port that the application will be using, the database dialect, driver, url, username and password. Here can also be defined some extra properties such as “spring.jpa.show-sql = true“, that enables the sql queries to be showed in the server console. All of these files can be seen in Figure ##

The structure of the Angular 2/5 project….

## Presentation Layer

The presentation layer of the email template module is realized thought the use of JSPs, JS and CSS. The email template module uses a singular JSP which contains all of the necessary elements i.e. the list of existing template, the form for the content and the buttons to save/delete/discard. Because Struts2 framework is being used, the JSP contains Struts2 specific tags. The list of already created templates is displayed through such tags. It uses the <s: iterator> and <s: property> tags to fill a container with the template names. Every template name is being placed in separate div element (the code is shown on figure 16).

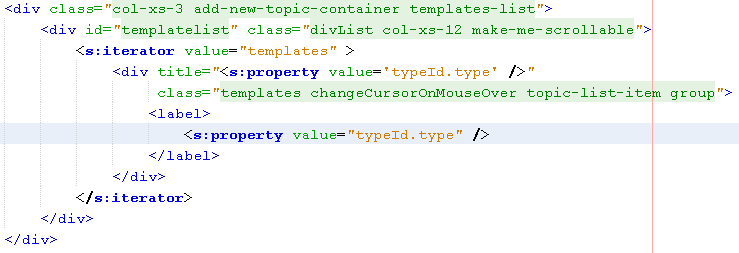


Figure 16: Code Snippet of the template list container.

To achieve the wanted visual outcome of the module (to have the form that is responsible for the content of the templates next to the list of templates) the grid system of bootstrap is being used. By using the bootstrap provided classes we can divide the screen in columns. Each container is divided in 12 columns. For the list of templates “col-xs-3” has been used. That leaves 9 columns for the container that holds the form for manipulating the email template’s content. Inside this container the bootstrap grid system has been used one more time – to arrange the buttons at the bottom of the form. Each one of the four buttons occupies an area of 3 columns of the container.

All of the functionalities in the presentation layers are achieved through the use of JavaScript. The following text and graphics are a detailed explanation of the methods and functions that have been used.

* loadAndRememberText (template) – this function loads the content of templates in to the form design to hold the content. It also remembers the template name.
* areThereAnyChanges – this function checks if there are any changes made in the description (content) and title of the template and returns true or false depending on the result. It is used for validation when the administrator wants to change template or has unsaved information.
* loadTemplateInfoRichBox – this function checks if the form which shows the content of the templates is initialized. If it’s not, this function initializes it with the necessary field. It is called when the page is loaded.
* confirmYourAction (text) – this function opens up a pop-up screen witch asks for the confirmation of the administrator. It offers several text options such as: delete, save, discard, save as draft, start new template, load template. This function is being used every time when the action of the administrator will lead to irreversible consequences regarding the templates.
* confirmYourActionAndSave (text) - in contrast to the previous function, this function allows the administrator to save the changes he or she has made before executing their action. Both functions have to be synchronized with the actions from which they are called. They need to return an answer to a question and depending on the answer, the actions perform differently.
* processTemplate (operation) – this function gets as a parameter a String which represents different kind of operations. The operation can be: save template, save as draft, delete, discard or update. This function is called mainly from the buttons that are located bellow the description (content) of the template.
* deactiveTopicSelectionHighlighting () – this function deactivates the highlighting of the selected template in the list of templates.
* highlightNewSelectedTopic() – this function highlights the clicked template name from the list of templates
* refreshTemplateGroups () – this function refreshes the template list if there are any changes made to the list.

There are several elements in the module that can trigger events:

* Delete Button – the delete button triggers an event leading to the removal of a template from the database:

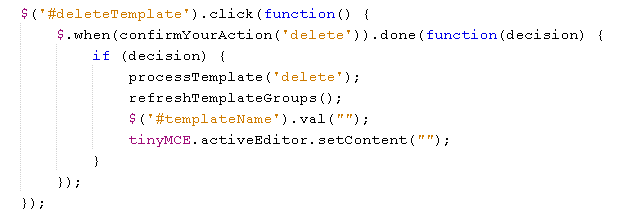


Figure 17: Code Snippet of the logic behind the delete button.

The code from figure 17 shows the logic behind the delete button. When the administrator has selected a template from the template list and clicks the delete button, he or she will be asked if he/she wants to proceed with the action. Here is the place where the “confirmYourAction” function comes in play. If the answer is positive (true) then the JavaScript continues with deleting of the template. After that it refreshes the template list and sets the fields for name and description (content) to an empty string. If the answer to the decision is negative (false) then the click event does nothing.

* Save As Draft Button – this button saves a template as a draft:

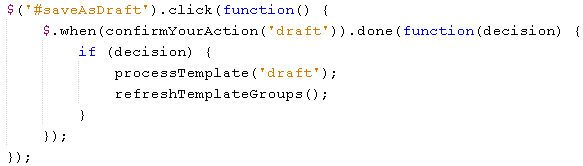


Figure 18: Code Snippet of the logic behind the save as draft button

The logic behind the “save as draft” button is very similar to the logic of the delete button. When clicked the administrator is asked if he/she is sure they want to proceed. If yes the template is being processed (saved in the DB) as a draft template and the list of templates is being refreshed with the new information.

* Discard Button – discards the content of the template



Figure 19: Code Snippet of the logic behind the discard button

When the discard button is being clicked a warning pop up will displayed to warn the administrator that he or she will lose the content (description) and name (title) of the template that is selected. The removal of the content is not permanent if it’s not saved.

* Create Template – prepares the field for a new template.

This button is similar to the discard button because it also clears all of the fields. The only difference is that it also unmarks the chosen email template (if any is selected).

* Save Template Button – saves the templates

This button saves the newly created templates and transforms the drafts to regular templates. The JavaScript here first checks if there are any changes made to the content (description) and the name (title) of the template. If there are changes a pop-up screen to note that is being shown. If the administrator chooses to proceed with the action, the script checks if the template is a draft or not and does the necessary operations to remove the visual draft status from the template.

* The highlighting of the templates in the template list

To prevent the administrator to accidently deleting the progress he or she has made in template that is being worked on, the template list except the highlighting of the chosen template, has the functionality to detect changes in the content (description) and title (name) of the templates. The following text explains the algorithm:

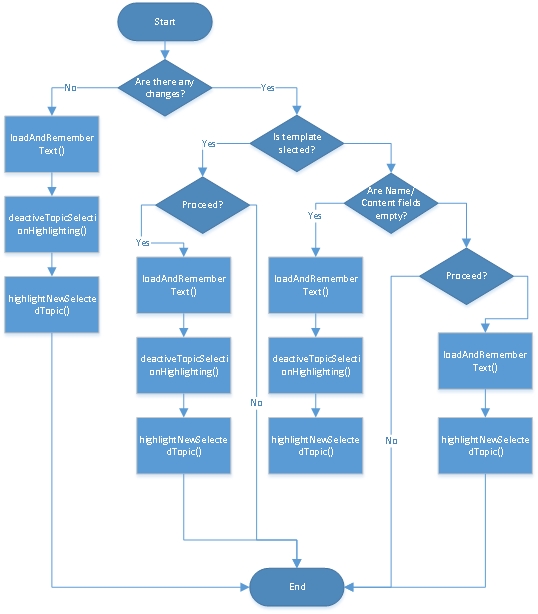


Figure 20: Flow chart diagram of the algorithm for highlighting chosen templates

First the algorithm checks if there are any changes made to the template (if one is loaded). If there are no changes, the script loads the selected template and highlights the name of the template in the template list. If there are changes, the script checks if any template in the template list has been chosen (highlighted). If it finds such a template the algorithm shows a pop-up for confirmation of the action (change of template). When a positive answer is given the algorithm proceeds to load the new template, deactivate the highlighting of the previous template and activates the highlighting of the new one. If the algorithm doesn’t find a chosen (highlighted) template it checks if the content (description) or the name (title) fields are empty the algorithm highlights the desired template and loads it. If the content and name fields aren’t empty the algorithm shows a pop-up for conformation of the action and base on the answer loads or not the content of the template. For better understanding of the algorithm figure 20 can be used.

## Model Layer

In the MVC design pattern the Model has only one purpose and that is to represent the data. The model is independent from the other 2 layers (the view layer and the controller layer). The model layer is composed of Entity classes (Hibernate persistent entities) that represent the database tables. The Entity classes are plain POJOs and as such they have to follow some simple rules:

* Persisted classes need a default constructor
* Persisted attributes should be encapsulated and should have the appropriate getters and setters
* All classes should contain ID for easy identification of objects within Hibernate and the DB.

Because the Entity classes follow the same rules they all have similar form and an explanation only to one of those classes will be given. The class that will be view is called Conferences and as the name suggests it represents the table Conferences from the Database. As a POJO that follows rules first it has to have a default constructor:

C:\Users\lyubomir.hristozov\Desktop\Diploma\Snimki za diplomna\Code\defaultConstructor.PNG

Figure 21: Code Snippet of the default constructor for the Conference class

To instantiate a persistent class Hibernate must have no-argument constructor because it uses Constructor.newInstance(). If there are other constructors in the entity class, the default constructor is not needed - it is provided by the JVM.

Every column in the table should be represented by a field with the same name as the name of the column. The access level of the fields must be set to private. As private fields they must also have the appropriate getters and setters methods.

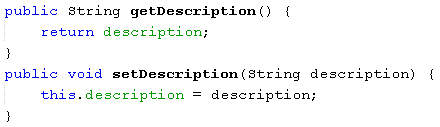
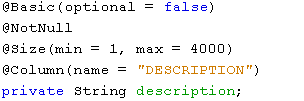


Figure 22: Code Snippet for declaration of a field and its setter and getter methods

There are two ways to transfer information from the entity classes to the database tables. The first one is with the use of XML mapping file and the second is with the use of Hibernate Annotations. We have decided to use the later. The Hibernate Annotations provide a better understanding of the table structure and entity classes simultaneously during the development. There are 2 things that are necessary to use the Hibernate Annotations. The first being the need for JDK version 5.0 or higher and second is the Hibernate 3.x annotations distribution package.

To connect the entity class to the table the annotation @Table is being used above the class. This annotation provides some additional attributes such as:

* Override the name of the table
* Override the catalogue of the table
* Override the schema of the table
* Enforce unique constraints on columns

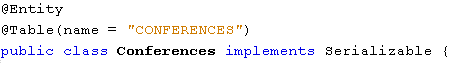


Figure 23: Code Snippet for connecting the entity class to the database table.

The @Entity annotation (see figure 23) marks the class as a persistence class and must have a no-argument constructor, fields that represent the columns of the database table and the appropriate getters and setters.

After connecting the name of the class to the name of the table we need to connect every field in the class to the desired column from the table. This is achieved through the use of the @Column annotation (see figure 22). This annotation is used to specify details about the column. Some of the most commonly used attributes are:

* Name – explicitly specify the name of the column.
* Length – explicitly specify the size of a column value (used mostly for strings).
* Nullable – allows a column to be marked as NOT NULL.
* Unique – allows a column to be marked as containing only unique values.

Hibernate offers a large variety of annotations. The most useful ones are the annotation that offers validation:

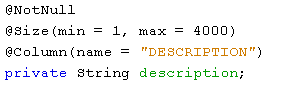


Figure 24: Code Snippet for annotations that offer validation.

On Figure 24 can be seen two validations. These validations are on the application level – they are faster than the database validations. The first two annotations in figure 24 validate the field before a query to the database is made. The first validation checks if the field has a value of null (i.e. its empty). The second validation checks if the size of the field is between one and four thousand.

As every table has a primary key (it can also have more than one), that primary key also should be noted as such in the entity class. The annotation that represents that a field (column) is a primary key is @Id.

To create a relationship between the persistent classes hibernate uses 3 different degrees of relationship annotations:

* @ManyToOne / @OneToMany - one of the most commonly used relationships. We use one of those depending on the perspective we are looking at the relationship. An Object can be associated with multiple objects.
* @OneToOne – similar to @ManyToOne but the column will be set as a unique.
* @ManyToMany – usually implemented with a Set Collection.

The annotation @JoinTable indicates that there will be an interaction with another table. To further develop the relationship between the tables the @JoinColumns annotations can be used. The name attribute holds the foreign key identifier for the relationship – specifies the column name in the join table. The referenceColumnName attribute holds the name of the primary key identifier for this relationship [13] [14].

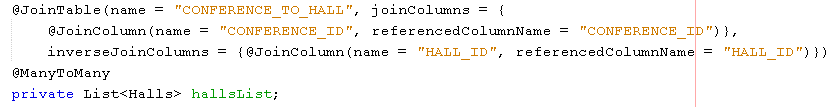
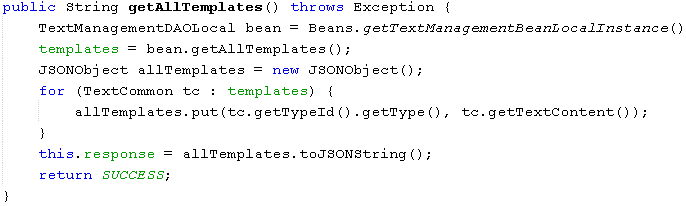


Figure 25: Example for joining tables with Hibernate

## Controller Layer

The Controller is the connection between the model layer and the view layer. Its job is to handle the requests from the users. The controller is responsible for fetching or processing data from the model and sending it to the view layer to be rendered. All of that is being done by the following methods:

* Show – this method is being called by the action “templatesBuilder”. It gets all of the templates in the database. If the return value is success the JSP page corresponding to the management of the templates is being opened. The method uses the help of another method – getAllTemplates.
* saveAsTemplate – this method is being called by the "templatesBuilderDo" action which is connected to the save as template button in the view layer. As the name suggests the method saves the current template. If the current template is a draft template, the method removes the draft status from the information in the database for the template and save it as a ready to be used template. If the current template is a newly written template – the method saves the template in the database. This method also uses other methods: checkIsTemplateExisting, deleteMailTemplate, insertMailTemplate and updateMailTemplate.
* saveAsDraft – this method is being called by the “templatesSaveAsDraft” action. This action is being used by the “save as draft” button in the view layer. It saves the current template as a draft template. If the current template is also a draft template – the method updates the current draft template. This method uses two other methods – insertMailTemplate and updateMailTemplate.
* Delete – this method is called by the “templatesBuilderDelete” action. This action is connected to the “delete” button in the view layer. The method as the name suggests deletes the current template with the help of the deleteMailTemplate method.
* getAllTemplates – this method is being called with the “getAllTemplates” action. If the return value is success, the action returns a string result with all of the templates in the database. One of the places that this action is being used is to populate the list of templates in the view layer. This method uses also the method getAllTemplates.

 Figure 26: Code Snippet of the method that gets all of the templates in the DB and returns them as a string.

* insertMailTemplate – this method is directly communication with the database and inserting the new templates in the database. It takes 2 inputs – the content of the template and the name of the template. In the method there are validations if the input parameters are correct.
* deleteMailTemplate – as the name suggests this method communicates with the database and deletes the template with the same content and name as the input parameter of the method.
* updateMailTemplate – This method updates the content of the templates.
* getAllTemplates – this method communicates with the database and returns a list with all of the templates in the database. The method returns a list of the templates.
* checkIsTemplateExisting – this method searches through the database if a template with the same name already exists. The input parameter is a string and the return type is either true or false.

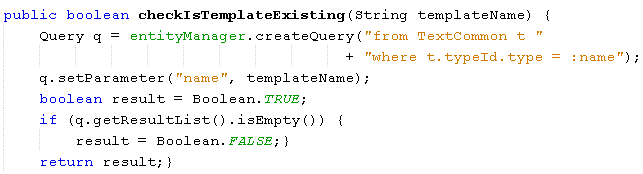


Figure 27: Code Snippet of the method that checks the DB if a template already exists.

* getTemplateText – this method gets the content of the template. As an input parameter it takes the id of the text. The result of the method is a string.

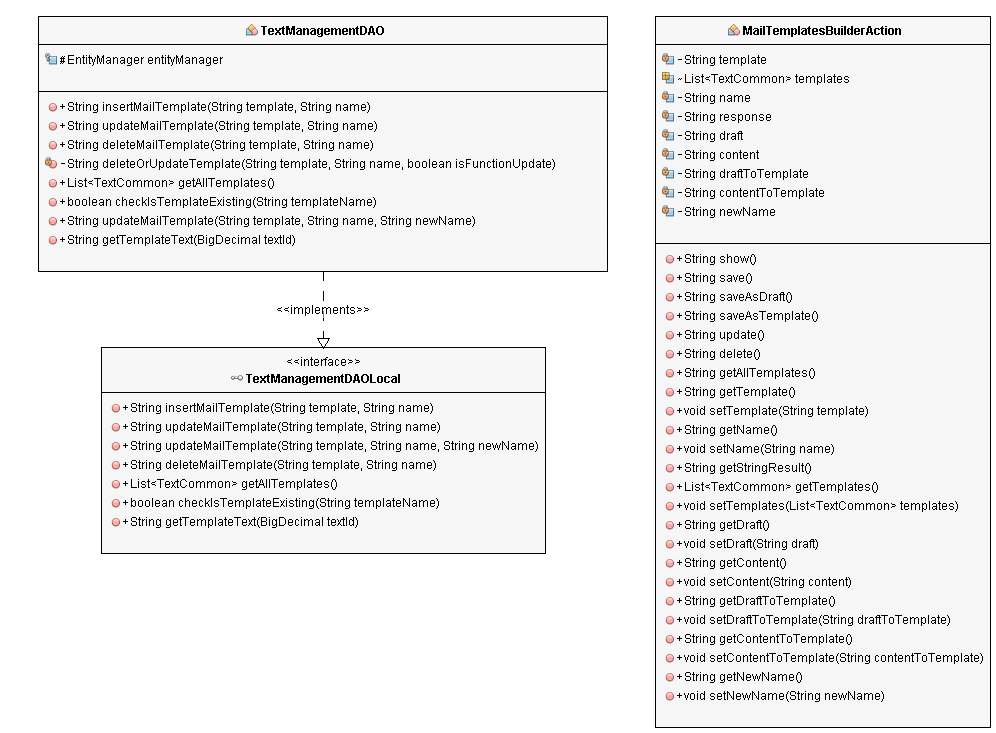


Figure 28: Class Diagram

## Additional use of the email templates

Except for creating a personalized content for emails the email templates module allows the administrator to also manage the content of the auto generated emails that the system sends in the event of registration to the site, approval or rejection to an Event, forgotten password of the user account. The algorithm for all of these auto generated messages is basically the same (with small changes) for all of the messages, so a detailed explanation of only one will be given. The following flow chart diagram is the representation of the general algorithm for content changes.

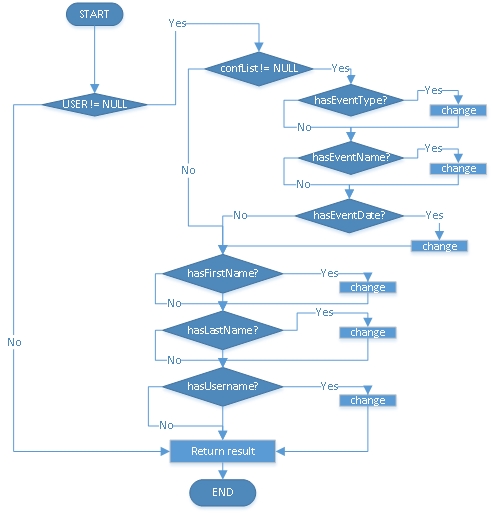


Figure 28: Flow Chart Diagram representing the algorithm for changing the content of auto generated emails.

The algorithm shown in figure 28 represents the flow of the processes when a message is being handled and the keywords in the content are being replaced with real data. In general the algorithm makes various verifications in the content of the messages and based on the result of the verifications, the algorithm makes changes to the content. As the messages have to be personalized to every user to whom they are being sent to, the first thing that has to be checked is if the user to whom the email is being send to is corresponding to an entity in the database table that is representing the users. If there is no such entity the algorithm ends cannot continue to the inability to perform real data replacement of the keywords. If there is such entity the algorithm can extract the necessary data to make changes in the content of the messages. The next verification that the algorithm needs to make is to check if the user is registered for any of the events. If he/she is registered for an event the algorithm continues with the changes concerning the information about the event. The content of the message is being scanned for any of the keywords concerning the events and if such a key word is found it is replaced with the correct information from the database (see figure 29).

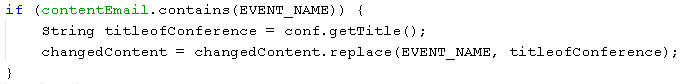


Figure 29: Code Snippet for replacing content in a message.

If the user isn’t registered for any events the algorithm can’t replace information for the event, so it skips to the part where the information about the user is being replaced. The replacement here happens in the same way as the information about the events. If a keyword concerning the user information is being found, it is replaced with the correct information stored in the database.

## User guide for the module

The purpose of this section is to show how the email templates module functions. Being a module of another system, the email templates module has been integrated in the administration panel of the site. The following figure represents the module when opened through the administration panel:

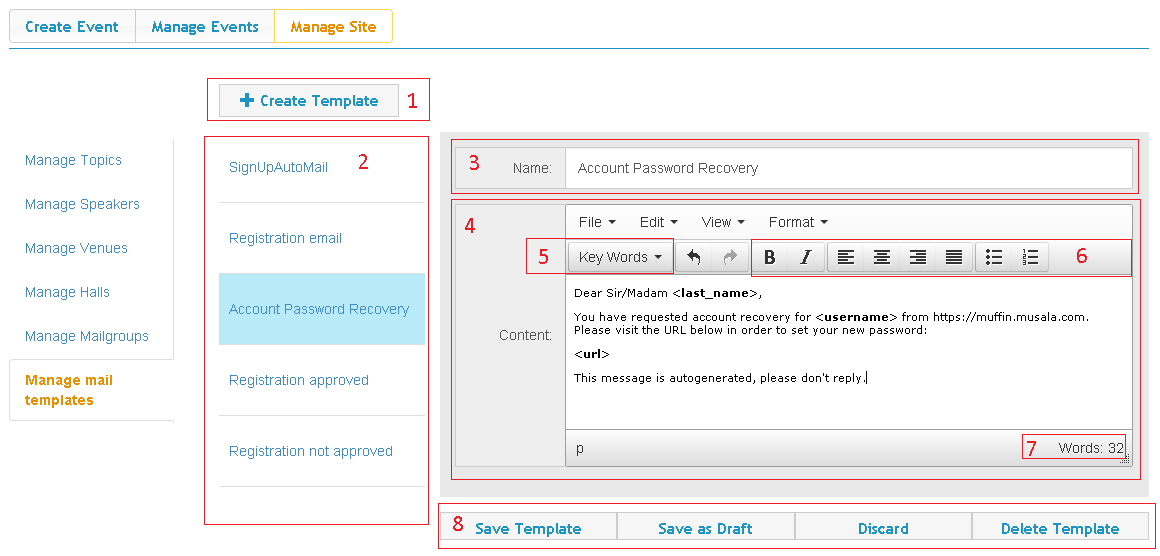


Figure 30: Email Templates Module opened through the administration panel

On the figure, all of the important functionalities are marked with a number:

1. Create template button
2. List of existing template
3. Field for the name of the template
4. Field containing the content of the template
5. A drop-down menu for key words
6. Button-like functionalities for formatting the content
7. Words counter
8. Buttons for saving as template / draft, discarding the content and deleting the template.

On figure 31 can be seen the two variations of the confirmation of admin actions:

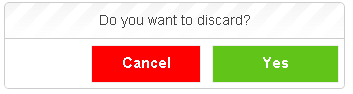


Figure 31: Confirmation boxes

The two figures above show the confirmation boxes when there are changes on the current template (the one to the left) and there is a need to be able to save the changes from the confirmation box. The one to the right shows when there is no need for a save functionality in the confirmation button.

There are two more kinds of pop-ups. They display if there is an error while trying to complete an action and if the action is completed:

C:\Users\lyubomir.hristozov\Desktop\Diploma\Snimki za diplomna\userGuide\5.PNGC:\Users\lyubomir.hristozov\Desktop\Diploma\Snimki za diplomna\userGuide\6.PNG

Figure 32: Pop-ups with error or success messages

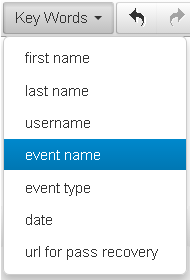


Figure 33: Drop-down menu with key words

When the administrator clicks on the drop-down menu (see № 5 from figure 30) it expands and shows all of the key words that are available. When a key word is selected it is appended to the text where the cursor of the content form is located.

# Conclusion

This section has the goal to analyze the development process and draw conclusions about the results:

## Summary of the results

The goal of the bachelor thesis is the creation of an easy to use module that helps the administrators to create personalized email templates. This goal has been achieved through the completion of three main tasks:

* To achieve the creation of the module, a place to store the information about every template is needed. That is done by a database. Several new tables has been created and added to the already existing ones of the system in which the email template module is integrated.
* The module interacts directly to the DB, allowing dynamically to change the content of already existing templates or to create new templates with various key words.
* The user interface of the email template module has been kept as cleaner and simpler as it can be so the use of the module to be easy and intuitive.

## Ideas for future research and development

The module can be further developed to help the administrators of the site to create better email templates. One of the new functionalities that can be added is the ability the administrator to create his own key words that correspond to some meaning in the data base. Without such functionality the administrator has a limited number of keywords that may prove insufficient. Another functionality that can be useful is the autocomplete of key words inside the content i.e. the administrator doesn’t need to choose a key word from the drop-down menu but simply to start typing the key word and it will auto complete. The last idea is the creation of a live preview of the template – when the administrator creates the template, he or she should be able to see how the email with the template will look like.

# Literary sources

1. <http://www.tensionsoftware.com/osx/automailer/>
2. <http://emailmarketing.comm100.com/features.aspx>
3. Mandala, R., JSP, <http://java-brain.blogspot.com/2011/10/jsp.html> May 7, 2006
4. JavaServer Pages Overview, <http://www.oracle.com/technetwork/java/overview-138580.html>
5. Java Platform, Enterprise Edition (Java EE), <http://www.oracle.com/technetwork/java/javaee/overview/index.html>
6. Struts 2 Architecture and Flow, [http://www.javatpoint.com/struts-2-architecture-and-flow#](http://www.javatpoint.com/struts-2-architecture-and-flow)
7. Struts 2 Architecture, <http://www.tutorialspoint.com/struts_2/struts_architecture.htm>
8. CSS3, <https://developer.mozilla.org/en/docs/Web/CSS/CSS3>
9. Bootstrap Get Started, <http://www.w3schools.com/bootstrap/bootstrap_get_started.asp>
10. JavaScript, <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
11. Hibernate ORM, <http://hibernate.org/orm/>
12. Application Server 8 Developer’s Guide, Chapter 3. Assembling and Deploying J2EE Applications, <http://docs.oracle.com/cd/E19253-01/817-6087/dgdeploy.html>
13. Hibernate - Persistent Class, <http://www.tutorialspoint.com/hibernate/hibernate_persistent_classes.htm>
14. Bernard, E., Chapter 2. Mapping Entities, <https://docs.jboss.org/hibernate/annotations/3.5/reference/en/html/entity.html>, September 15, 2010
15. Dalling, T., Model View Controller Explained, <http://www.tomdalling.com/blog/software-design/model-view-controller-explained/> May 31, 2009
16. Design Patterns MVC Pattern, <http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm>
17. jQuery – Overview, <http://www.tutorialspoint.com/jquery/jquery-overview.htm>
18. MySQL, https://www.tutorialspoint.com/mysql/mysql-introduction.htm

# Attachments

MailTemplatesBuilderAction.java

package com.muffin.admin.actions;

import com.muffin.entities.TextCommon;

import com.muffin.globals.Beans;

import com.muffin.local.text.management.TextManagementDAOLocal;

import com.muffin.results.StringResultInterface;

import static com.opensymphony.xwork2.Action.INPUT;

import static com.opensymphony.xwork2.Action.SUCCESS;

import com.opensymphony.xwork2.ActionSupport;

import com.opensymphony.xwork2.validator.annotations.RequiredStringValidator;

import java.util.List;

import org.apache.struts2.convention.annotation.Action;

import org.apache.struts2.convention.annotation.InterceptorRef;

import org.apache.struts2.convention.annotation.Namespace;

import org.apache.struts2.convention.annotation.Result;

import org.apache.struts2.interceptor.validation.SkipValidation;

import org.json.simple.JSONObject;

@Namespace("/")

@InterceptorRef("adminAuthenticationStack")

public class MailTemplatesBuilderAction extends ActionSupport implements StringResultInterface {

private String template;

List<TextCommon> templates;

private String name;

private String response;

private String draft;

private String content;

private String draftToTemplate;

private String contentToTemplate;

private String newName;

@Action(value = "templatesBuilder", results = {

@Result(name = SUCCESS, location = "adminPages/manageTemplate.jsp")

})

@SkipValidation

public String show() throws Exception {

TextManagementDAOLocal bean = Beans.getTextManagementBeanLocalInstance();

templates = bean.getAllTemplates();

return SUCCESS;

}

@Action(value = "templatesBuilderDo", results = {

@Result(name = SUCCESS, type = "stringResult"),

@Result(name = INPUT, location = "adminPages/mailTemplatesBuilder.jsp")

})

public String save() throws Exception {

TextManagementDAOLocal bean = Beans.getTextManagementBeanLocalInstance();

this.response = bean.insertMailTemplate(template, name);

return SUCCESS;

}

// save as draft

@Action(value = "templatesSaveAsDraft", results = {

@Result(name = SUCCESS, type = "stringResult"),

@Result(name = INPUT, location = "adminPages/mailTemplatesBuilder.jsp")

})

public String saveAsDraft() throws Exception {

TextManagementDAOLocal bean = Beans.getTextManagementBeanLocalInstance();

if (name.contains("draft")) {

draft = name.replaceAll("- draft", "");

draft = draft + "- draft";

content = template;

} else {

draft = name + "- draft";

content = template;

}

if (!bean.checkIsTemplateExisting(draft)) {

this.response = bean.insertMailTemplate(content, draft);

} else {

this.response = bean.updateMailTemplate(content, draft);

}

return SUCCESS;

}

// save as template

@Action(value = "saveAsTemplate", results = {

@Result(name = SUCCESS, type = "stringResult"),

@Result(name = INPUT, location = "adminPages/mailTemplatesBuilder.jsp")

})

public String saveAsTemplate() throws Exception {

TextManagementDAOLocal bean = Beans.getTextManagementBeanLocalInstance();

draft = name;

if (template.equals("")) {

content = " ";

} else {

content = template;

}

if (bean.checkIsTemplateExisting(draft)) {

if (name.contains("draft")) {

draftToTemplate = name.replaceAll("- draft", "");

if (template.equals("")) {

contentToTemplate = " ";

} else {

contentToTemplate = template;

}

this.response = bean.deleteMailTemplate(content, draftToTemplate);

this.response = bean.insertMailTemplate(contentToTemplate, draftToTemplate);

this.response = bean.deleteMailTemplate(content, draft);

} else {

this.response = bean.updateMailTemplate(content, draft);

}

} else {

this.response = bean.insertMailTemplate(content, draft);

}

return SUCCESS;

}

@Action(value = "templatesBuilderUpdate", results = {

@Result(name = SUCCESS, type = "stringResult"),

@Result(name = INPUT, location = "adminPages/mailTemplatesBuilder.jsp")

})

public String update() throws Exception {

TextManagementDAOLocal bean = Beans.getTextManagementBeanLocalInstance();

this.response = bean.updateMailTemplate(template, name, newName);

return SUCCESS;

}

@Action(value = "templatesBuilderDelete", results = {

@Result(name = SUCCESS, type = "stringResult"),

@Result(name = INPUT, location = "adminPages/manageTemplate.jsp")

})

public String delete() throws Exception {

TextManagementDAOLocal bean = Beans.getTextManagementBeanLocalInstance();

this.response = bean.deleteMailTemplate(template, name);

return SUCCESS;

}

@Action(value = "getAllTemplates", results = {

@Result(name = SUCCESS, type = "stringResult")

})

@SkipValidation

public String getAllTemplates() throws Exception {

TextManagementDAOLocal bean = Beans.getTextManagementBeanLocalInstance();

templates = bean.getAllTemplates();

JSONObject allTemplates = new JSONObject();

for (TextCommon tc : templates) {

allTemplates.put(tc.getTypeId().getType(), tc.getTextContent());

}

this.response = allTemplates.toJSONString();

return SUCCESS;

}

public String getTemplate() {

return template;

}

@RequiredStringValidator(key = "error.template.content", message = "")

public void setTemplate(String template) {

this.template = template;

}

public String getName() {

return name;

}

@RequiredStringValidator(key = "error.template.name", message = "")

public void setName(String name) {

this.name = name;

}

@Override

public String getStringResult() {

return this.response;

}

public List<TextCommon> getTemplates() {

return templates;

}

public void setTemplates(List<TextCommon> templates) {

this.templates = templates;

}

public String getDraft() {

return draft;

}

public void setDraft(String draft) {

this.draft = draft;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

public String getDraftToTemplate() {

return draftToTemplate;

}

public void setDraftToTemplate(String draftToTemplate) {

this.draftToTemplate = draftToTemplate;

}

public String getContentToTemplate() {

return contentToTemplate;

}

public void setContentToTemplate(String contentToTemplate) {

this.contentToTemplate = contentToTemplate;

}

public String getNewName() {

return newName;

}

public void setNewName(String newName) {

this.newName = newName;

}

}

**TextManagementDAO.java**

@Stateless(name = "TextManagementDAO", mappedName = "TextManagementDAO")

public class TextManagementDAO implements TextManagementDAOLocal {

@PersistenceContext(unitName = "test-ejbPU")

protected EntityManager entityManager;

@Override

public String insertMailTemplate(String template, String name) {

String result = "success";

TextDomain td = getTextDomainBy(GlobalEnums.Domains.EMAIL\_TEMPLATE.toString());

if (td == null) {

td = new TextDomain();

td.setName(GlobalEnums.Domains.EMAIL\_TEMPLATE.toString());

td = entityManager.merge(td);

}

Query q = entityManager.createQuery("from TextType t where t.type = :type and t.domainId = :domain");

q.setParameter("type", name);

q.setParameter("domain", td);

if (q.getResultList().isEmpty()) {

TextType templateName = new TextType();

templateName.setDomainId(td);

templateName.setType(name);

templateName = entityManager.merge(templateName);

TextCommon tc = new TextCommon();

tc.setDomainId(td);

tc.setTypeId(templateName);

tc.setTextContent(template);

entityManager.merge(tc);

} else {

result = "duplicate";

}

return result;

}

@Override

public String updateMailTemplate(String template, String name) {

return deleteOrUpdateTemplate(template, name, true);

}

@Override

public String deleteMailTemplate(String template, String name) {

return deleteOrUpdateTemplate(template, name, false);

}

private String deleteOrUpdateTemplate(String template, String name, boolean isFunctionUpdate) {

String result = "success";

TextDomain td = getTextDomainBy(GlobalEnums.Domains.EMAIL\_TEMPLATE.toString());

Query q = entityManager.createQuery("from TextCommon t where t.typeId.type = :type and t.domainId = :domain");

q.setParameter("type", name);

q.setParameter("domain", td);

List<TextCommon> templatesList = q.getResultList();

if (templatesList.isEmpty()) {

result = "invalid";

} else {

TextCommon templ = templatesList.get(0);

TextType type = templ.getTypeId();

if (isFunctionUpdate) {

templ.setTextContent(template);

type.setType(name);

entityManager.merge(templ);

} else {

entityManager.remove(templ);

entityManager.remove(type);

}

}

return result;

}

@Override

public List<TextCommon> getAllTemplates() {

Query q = entityManager.createQuery("from TextCommon t where t.domainId.name = :name");

q.setParameter("name", GlobalEnums.Domains.EMAIL\_TEMPLATE.toString());

List<TextCommon> result = q.getResultList();

return result;

}

@Override

public boolean checkIsTemplateExisting(String templateName) {

Query q = entityManager.createQuery("from TextCommon t "

+ "where t.typeId.type = :name");

q.setParameter("name", templateName);

boolean result = Boolean.TRUE;

if (q.getResultList().isEmpty()) {

result = Boolean.FALSE;}

return result;}

@Override

public String updateMailTemplate(String template, String name, String newName) {

String result = "success";

TextDomain td = getTextDomainBy(GlobalEnums.Domains.EMAIL\_TEMPLATE.toString());

Query q = entityManager.createQuery("from TextCommon t where t.typeId.type = :type and t.domainId = :domain");

q.setParameter("type", name);

q.setParameter("domain", td);

List<TextCommon> templatesList = q.getResultList();

if (templatesList.isEmpty()) {

result = "invalid";

} else {

TextCommon templ = templatesList.get(0);

TextType type = templ.getTypeId();

templ.setTextContent(template);

type.setType(newName);

entityManager.merge(templ);

entityManager.merge(type);

}

return result;

}

@Override

public String getTemplateText(BigDecimal textId) {

Query q = entityManager.createQuery("From TextCommon t where t.textId = :textId ");

q.setParameter("textId", textId);

List<TextCommon> result = q.getResultList();

TextCommon txt = result.get(0);

return result.get(0).getTextContent();

}

}

**ManageTemplate.jsp**

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%@taglib prefix="s" uri="/struts-tags" %>

<script type="text/javascript" src="<%=request.getContextPath()%>/tinymce/js/tinymce/tinymce.min.js"></script>

<script language="javascript" src="<%=request.getContextPath()%>/noty-2.3.2/js/noty/packaged/jquery.noty.packaged.min.js"></script>

<script language="javascript" src="<%=request.getContextPath()%>/JavaScript/rich-text-init.js"></script>

<script language="javascript" src="<%=request.getContextPath()%>/JavaScript/notyUtils.js"></script>

<script language="javascript" src="<%=request.getContextPath()%>/JavaScript/templatesSelectRefresh.js"></script>

<script language="javascript" src="<%=request.getContextPath()%>/JavaScript/mailTemplatesBuilder.js"></script>

<script language="javascript" src="<%=request.getContextPath()%>/JavaScript/manageTemplate.js"></script>

<!DOCTYPE html>

<div id="test" class=" col-xs-12 main-content-form-style">

<div class="row col-xs-12 add-new-topic-container">

<div class="pull-left">

<button style="width:180px" title="This button will prepare the forms for a new template." style="margin-left: -5px" id="createTemplate" class="btn-create-event btn-add-formater">

<span class="glyphicon glyphicon-plus"></span>

<label>Create Template</label>

</button>

</div>

</div>

<div class="row">

<div class="col-xs-3 add-new-topic-container templates-list">

<div id="templatelist" class="divList col-xs-12 make-me-scrollable">

<s:iterator value="templates" >

<div title="<s:property value='typeId.type' />"

class="templates changeCursorOnMouseOver topic-list-item group">

<label>

<s:property value="typeId.type" />

</label>

</div>

</s:iterator>

</div>

</div>

<div class="col-xs-9 add-new-topic-container">

<div class="panel-body panel-wide ">

<s:hidden id="templateId" name="templateId" />

<s:hidden name="templates" />

<div class="input-group input-group-login">

<span class="input-group-addon addon-admin-short"><s:text name="label.template.name" /></span>

<s:textfield id="templateName" name="name" cssClass="form-control" cssErrorClass="field-error" />

</div>

<div class="input-group input-group-login " >

<span class="input-group-addon addon-admin-short"><s:text name="label.content.email" /></span>

<s:textarea id="contentTemplate" name="contentTemplate" style="width:100%" rows="7" cssClass="form-control" cssErrorClass="field-error" />

</div>

</div>

<div>

<div >

<button id="saveAsTemplate" title="This button will save the changes and make a draft into fully functional template."style ="margin-left: 0px" class="btn-create-event btn-add-formater col-xs-3"

<label>Save Template</label>

</button>

<button id="saveAsDraft" title="This button will save the template as a draft." style ="margin-left: 0px" class="btn-create-event btn-add-formater col-xs-3"

<label>Save as Draft</label>

</button>

<button id="discard" title="This button will discard the information." style ="margin-left: 0px" class="btn-create-event btn-add-formater col-xs-3"

<label>Discard</label>

</button>

<button id="deleteTemplate" title="This button will delete the current template." style ="margin-left: 0px" class="btn-create-event btn-add-formater col-xs-3"

<label>Delete Template</label>

</button>

</div>

</div>

</div>

</div>

</div>

**templatesSelectRefresh.js**

function refreshTemplateGroups() {

$.ajax({

type: 'post',

url: 'getAllTemplates',

dataType: 'html',

success: function(data) {

console.log(data);

window.templatesList = JSON.parse(data);

var $field = $('#templatelist');

$field.empty();

$.each(window.templatesList, function(k, v) {

$div = $(document.createElement('div'));

$div.attr('class', 'templates changeCursorOnMouseOver topic-list-item group');

$div.attr('title', k);

$label = $(document.createElement('label')).val(k).text(k);

$div.append($label);

$field.append($div)

listOfTemplates = JSON.parse(data);

});

}

});

}

function refreshTemplateGroupsSelect() {

$.ajax({

type: 'post',

url: 'getAllTemplates',

dataType: 'html',

success: function(data) {

var properties = [];

window.templatesList = JSON.parse(data);

//console.log(window.templatesList);

var iterator = 1;

$.each(window.templatesList, function(k, v) {

properties.push({label: k, value: iterator});

iterator++;

});

properties.sort(function(a, b) {

if (a.label > b.label) {

return 1;

}

if (a.label < b.label) {

return -1;

}

return 0;

});

$('#mail-templates-select').multiselect('dataprovider', properties);

}

});

}

**manageTemplate.js**

var listOfTemplates;

var oldName;

var currentlySelectedTemplate;

function getTemplateById(listOfTemplates, desiredTemplateId) {

for (var i = 0; i < listOfTemplates.length; i++) {

if (listOfTemplates[i]['templateId'] === desiredTemplateId) {

return listOfTemplates[i];

}

}

}

function areThereAnyChanges() {

// check if:

// -> description was modified;

// -> title was modified;

var m = currentlySelectedTemplate.description;

if (currentlySelectedTemplate.description !== tinymce.get(0).getContent()

|| currentlySelectedTemplate.title !== $("#templateName").val()) {

return true;

}

return false;

}

function selectTemplate(template)

{

template.title = $("#templateName").val();

template.description = tinyMCE.get('contentTemplate').getContent();

}

function loadAndRememberText(template) {

// fills the richbox

$('#templateName').val($(template).attr('title'));

tinymce.get(0).setContent(listOfTemplates[$(template).prop('title')]);

oldName = $('#templateName').val();

selectTemplate(currentlySelectedTemplate);

}

function loadTemplateInfoRichBox() {

if (tinymce.editors.length > 0) {

tinyMCE.execCommand('mceAddEditor', false, "contentTemplate");

tinyMCE.execCommand();

} else {

/\* init rich text \*/

tinymce.init({

selector: "#contentTemplate",

plugins: "wordcount",

toolbar: "mybutton | undo redo | bold italic | alignleft aligncenter alignright alignjustify | bullist numlist | link image ",

setup: function(editor) {

editor.addButton('mybutton', {

type: 'menubutton',

text: 'Key Words',

icon: false,

menu: [

{text: 'first name', onclick: function() {

editor.insertContent("&lt;first\_name&gt;");

}},

{text: 'last name', onclick: function() {

editor.insertContent("&lt;last\_name&gt;");

}},

{text: 'username', onclick: function() {

editor.insertContent("&lt;username&gt;");

}},

{text: 'event name', onclick: function() {

editor.insertContent("&lt;event\_name&gt;");

}},

{text: 'event type', onclick: function() {

editor.insertContent("&lt;event\_type&gt;");

}},

{text: 'date', onclick: function() {

editor.insertContent('&lt;date&gt;');

}},

{text: 'url for pass recovery', onclick: function() {

editor.insertContent("&lt;url&gt;");

}}

]

});

}

});

}

}

// noty pop-up for confirming action

function confirmYourAction(text) {

var dfd = $.Deferred();

var msg;

if (text === "delete") {

msg = "Do you want to delete?";

} else if (text === 'save') {

msg = "Do you want to save changes?";

} else if (text === 'discard') {

msg = "Do you want to discard?";

} else if (text === 'draft') {

msg = "Do you want to save as draft?";

} else if (text === 'createNew') {

msg = "Do you want to start new template?";

} else if (text === 'changeTemplate') {

msg = "Do you want to load template?";

}

var n = noty({

text: msg,

type: 'confirm',

dismissQueue: false,

layout: 'center',

theme: 'defaultTheme',

buttons: [

{addClass: 'btn-red', text: 'Cancel', onClick: function($noty) {

$noty.close();

dfd.resolve(false);

}

},

{addClass: 'btn-green', text: 'Yes', onClick: function($noty) {

$noty.close();

dfd.resolve(true);

}

}

]

});

return dfd.promise();

}

function confirmYourActionAndSave(text) {

var dfd = $.Deferred();

var msg;

if (text === "delete") {

msg = "Do you want to delete?";

} else if (text === 'save') {

msg = "Do you want to save changes?";

} else if (text === 'discard') {

msg = "Do you want to discard?";

} else if (text === 'draft') {

msg = "Do you want to save as draft?";

} else if (text === 'createNew') {

msg = "Do you want to start new template?";

} else if (text === 'changeTemplate') {

msg = "Do you want to load template?";

}

var m = noty({

text: msg,

type: 'confirm',

dismissQueue: false,

layout: 'center',

theme: 'defaultTheme',

buttons: [

{addClass: 'btn-red', text: 'Cancel', onClick: function($noty) {

$noty.close();

dfd.resolve(false);

}

},

{addClass: 'btn-green-bland', text: 'Yes', onClick: function($noty) {

$noty.close();

dfd.resolve(true);

}

},

{addClass: 'btn-green ', text: 'Save Changes', onClick: function($noty) {

$noty.close();

var str1 = $('#templateName').val();

var str2 = "draft";

if (str1.indexOf(str2) != -1) {

processTemplate('draft');

refreshTemplateGroups();

dfd.resolve(true);

} else {

processTemplate('asTemplate');

refreshTemplateGroups();

dfd.resolve(true);

}

}

}

]

});

return dfd.promise();

}

function processTemplate(operation) {

var typeOfOperation = "";

var params = new Object();

if (operation === 'save') {

action = 'templatesBuilderDo';

typeOfOperation = "saved";

params['name'] = $("#templateName").val();

} else if (operation === 'update') {

action = 'templatesBuilderUpdate';

typeOfOperation = "updated";

params['newName'] = $('#templateName').val();

params['name'] = oldName;

} else if (operation === 'delete') {

action = 'templatesBuilderDelete';

typeOfOperation = "deleted";

params['name'] = $('#templateName').val();

} else if (operation === 'draft') {

action = 'templatesSaveAsDraft';

typeOfOperation = "saved as draft";

params['name'] = $("#templateName").val();

} else if (operation === 'asTemplate') {

action = 'saveAsTemplate';

typeOfOperation = "saved as template";

params['name'] = $("#templateName").val();

} else if (operation === 'discard') {

action = 'discardTemplateContent';

typeOfOperation = "discard template content";

params['name'] = $("#templateName").val();

}

params['template'] = tinyMCE.activeEditor.getContent();

if (params['name'] !== "") {

notificate("Processing..", "warning", 100000);

$.ajax({

type: 'post',

url: action,

data: params,

dataType: 'html',

success: function(data) {

if (data === 'success') {

notificate("Template " + params['name'] + " " + typeOfOperation + "!", "success");

refreshTemplateGroups();

} else if (data === 'invalid') {

notificate("Template does not exist!", "error");

} else if (data === 'duplicate') {

notificate("Template " + params['name'] + " already exist!", "error");

} else {

notificate("There is something wrong!", "error");

}

setTimeout(function() {

$.noty.closeAll();

}, 5000);

}

});

} else {

notificate("Template name not set!", "error", 3000);

}

}

function deactiveTopicSelectionHighlighting() {

$('.active-topic').removeClass("active-topic");

}

function highlightNewSelectedTopic(htmlSectionElement) {

$(htmlSectionElement).addClass("active-topic");

}

$(document).ready(function() {

window.templatesList = {};

refreshTemplateGroups();

currentlySelectedTemplate = {

title: "",

description: "",

id: ""

};

/\* delete button \*/

$('#deleteTemplate').click(function() {

$.when(confirmYourAction('delete')).done(function(decision) {

if (decision) {

processTemplate('delete');

refreshTemplateGroups();

$('#templateName').val("");

tinyMCE.activeEditor.setContent("");

}

});

});

// create button

$('#createTemplate').click(function() {

$.when(confirmYourActionAndSave('createNew')).done(function(decision) {

if (decision) {

var text = "<p></p>";

tinyMCE.get('contentTemplate').setContent(text);

$('#templateName').val("");

deactiveTopicSelectionHighlighting();

}

});

});

// create as draft button

$('#saveAsDraft').click(function() {

$.when(confirmYourAction('draft')).done(function(decision) {

if (decision) {

processTemplate('draft');

refreshTemplateGroups();

}

});

});

// Save as template button

$('#saveAsTemplate').click(function() {

if (areThereAnyChanges()) {

$.when(confirmYourAction('save')).done(function(decision) {

if (decision) {

var str1 = $('#templateName').val();

var str2 = "- draft";

processTemplate('asTemplate');

if (str1.indexOf(str2) != -1) {

var newTitle = str1.replace(str2, "");

$('#templateName').val(newTitle);

}

selectTemplate(currentlySelectedTemplate);

refreshTemplateGroups();

}

});

} else {

$.when(confirmYourAction('save')).done(function(decision) {

if (decision) {

var str1 = $('#templateName').val();

var str2 = "- draft";

processTemplate('asTemplate');

if (str1.indexOf(str2) != -1) {

var newTitle = str1.replace(str2, "");

$('#templateName').val(newTitle);

}

selectTemplate(currentlySelectedTemplate);

refreshTemplateGroups();

}

});

}

});

// discard button

$('#discard').click(function() {

$.when(confirmYourAction('discard')).done(function(decision) {

if (decision) {

var text = "<p></p>";

tinyMCE.get('contentTemplate').setContent(text);

$('#templateName').val("");

}

});

});

refreshTemplateGroups();

loadTemplateInfoRichBox();

$("#tabs").tabs().addClass("ui-tabs-vertical ui-helper-clearfix");

$("#tabs li").removeClass("ui-corner-top").addClass("ui-corner-left");

$.ajax({

type: 'post',

url: 'getAllTemplates',

dataType: 'html',

success: function(data) {

listOfTemplates = JSON.parse(data);

}

});

// highlights when a template name is clicked

$('#templatelist').on('click', '.templates', function() {

var t = $(this);

if (areThereAnyChanges()) {

if ($('.divList').find('div.active-topic').length !== 0) {

var ar = $("#templateName").val();

$.when(confirmYourActionAndSave('changeTemplate')).done(function(decision) {

if (decision) {

loadAndRememberText(t);

deactiveTopicSelectionHighlighting();

highlightNewSelectedTopic(t);

}

});

}

else {

if ($("#templateName").val() !== "" || tinyMCE.get('contentTemplate').getContent() !== "") {

$.when(confirmYourActionAndSave('changeTemplate')).done(function(decision) {

if (decision) {

loadAndRememberText(t);

highlightNewSelectedTopic(t);

}

});

} else {

loadAndRememberText(t);

deactiveTopicSelectionHighlighting();

highlightNewSelectedTopic(t);

}

}

} else {

loadAndRememberText(t);

deactiveTopicSelectionHighlighting();

highlightNewSelectedTopic(t);

}

});

});