KTH ICT II1302 Studentproject

Group 2

Displaimer

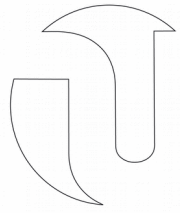
Web-application Component Specification

Abstract

This document describes the thoughts behind and the process of creating the web-application that is used in our product Displaimer. The document will briefly describe how the system is built, required behavior and interfaces of the web-application and how one could continue working with this system to further improve and evolve it.

Version History

Date	Version	Author	Description
22/05/2017	1.0	Sebastian Heimlén	First draft.



IVAR JACOBSON CONSULTING

An Essential Unified Process Document

Table of Contents

1 Introduction

1.1 About this document

The purpose of this document is to give the reader a better insight into the general structure of the web-application of the product Displaimer, so that the reader then can take this application in its current state and use it as a basis for creating there own web-services with its own purposes and means. This documentation will point out things that we thought could have been done better, things that other developers could improve on and further develop, if they choose to pick this system up and continue development.

To get the most out of this document you should be interested in the process of building both the back-end and the front-end for a web-application, you should have some knowledge in programming, specifically in Scala or Java. You should also have an interest in HTML and building websites and front-end, since this document also will discuss how the website could be improved. Last but not least you should be prepared and ready to read more documentation, since this application is built on large frameworks that have much more functionality than what is described in this document.

1.2 Background information about the component

The web-application is a large component and a central part of any IT-system, this because the web-application often has several features that are detrimental to the system as a whole. Examples of things that are usually included in the web-application is database connectivity and access, user interface and the entire infrastructure required to securely send data from the user interface to the database without compromising any data. This huge responsibility and list of requirements of the component often makes it feel obstructively large and complex, and it often is rather complex to build, therefore this component can be used either as inspiration or as a basis for other similar projects.

1.3 Document Purpose

The purpose of this document is to:

- Briefly describe the requirements and interfaces that is needed for this component.
- Establish the dependencies that exist with other components.
- Support the independent development of this component.
- Provide a basis for testing that the component operates as required.

1.4 Document Scope

The scope of this document is limited to consideration of:

- The specification of the required behaviors for this component.
- The specification of the interfaces that this component must implement.

- Listing the required interfaces that this component depends upon.
- Optionally specifying the internal structure and design of the component.

This scope of this document does not include consideration of:

- The unit tests that required to verify that this component conforms and performs to its specification.
- The specification or implementation of any other components that depend upon this component or that this component depends upon.
- The detailed explanation how everything in the system works.

1.5 Document Overview

This document contains the following sections:

- **Required Behavior** – behavior that is required from the component in terms of the responsibilities that it must satisfy and related requirements
- **Implementation Constraints** – constraints like implementation environment, implementation language and related patterns, guidelines and standards
- **Provided Interfaces** – the interfaces that the component must provide specified in terms of their signatures and constraints
- **Required Interfaces** – lists the interfaces that the component depends upon
- **Internal Structure** – optionally specifies the internal design structure of the component in terms of its constituent components and their relationships
- **Internal Element Designs** – optionally specifies the internal design of the constituent elements the component.
- **References** – provides full reference details for all documents, white papers and books referenced by this document.

2 Required Behavior

This section specifies the behavior that is required from the component in terms of:

1. The functions or services it must provide for its clients
2. The non-functional requirements that relate to these functions
3. The general functional requirements that relate to these functions

The big difference between a web-application and for example the message scheduler that is a smaller component in our system is that in a web-application there is a lot going on behind the scenes that one try to hide from the user, which results in rather few functions for the client, but a lot more non-functional requirements to make this work. The scheduler on the other hand provides a lot of functionality for the user, but has a lot fewer non-functional requirements, since it runs on the web-application that has already sorted out these requirements. Always keep in mind in what extent the system wants to provide functionality for the user, or if the system rather hide it to keep “safe” from the users.

1) The web-application should provide the following services for its clients:

- An user interface in the form of a website that the client can use to work with the web-application.

When the word web-application comes up, a lot of people immediately associate the word with some sort of website, since that is what a web-application is to most people, no matter how it works behind the scenes the clients must be able to interact with the web-application.

- The ability to log in to this website, to keep the interaction private for the user.

This is not really a function that the user per se demands, but rather a requirement on the web-application to guarantee the users security and privacy while using the application, as well as having history and being able to trace what every user has done. I choose to put this as a function for the user, because I see it as a functionality that the user would not be happy to be without.

- Give the user the ability to input data that is then handled by the web-application.

This is an optional requirement, in our case with Displaimer it was essential to give the user the ability to enter a message that could be shown on the display, depending on what the purpose of your application is, this may not be needed. Most websites have a need for some kind of data input, for example if you are creating a forum it is essential, but if you are creating a wiki-website where all information is to be provided by you as the owner, then there is no need for the user to be able to input data or log on to the site.

2) For these requirements to be fulfilled there are a few non-functional requirements that need to exist.

- Reliability (Performance).

The web-application must be reliable and work. The web-application should always be up and accessible for the user, only for brief moments once in a while should there be need for some kind of maintenance. The web-application must also be responsive and fast, and the user should get some kind of feedback in the UI within at most one second so that the user is informed that the system is actually doing something and not just sitting there unresponsively. All parts of the system must be fast, this includes the interaction with sub-systems such as the database and the scheduler.

- Data integrity.

The application must ensure that all data input on the website is safe and that the integrity of the user and the data entered is kept, the data must be secure on the way through the system to the database, and after that it is the database that has to make sure the data is safe.

3) To ensure these requirements there exists a few functional requirements:

- A Database to store input data and a way to interact with this database.

There needs to exist a database where the data entered by the user as well as user details are stored. The web-application must also be able to write and read from the database.

- A web-server that runs the web-application.

To run a web-application there needs to exist a web-server that the application is actually ran on.

3 Implementation Constraints

This section specifies the constraints acting upon the implementation of the component, such as:

1. The deployment environments within which it must operate
2. What language(s) it must be written in
3. The component frameworks it should operate within
4. Design patterns it should conform to
5. Environment constraints such as memory footprint and other resource usage constraints

1) The web-application can be exported as a JAR-file that can be deployed on the Java 8-JVM. The application requires Sodium 1.0.12 [1] that is used to hash the passwords.

2) The application is written in Scala, the reason for this is that Scala is compatible with Java and runs on JVM, Scala is a scalable language, which lets the developers mold the product to their liking, we had also already had our eyes on the Play framework that the application is built on, and therefore Scala was an obvious choice.

The actual website is also constructed in Scala, within the same framework as the rest of the application, however the website is of course written in HTML and CSS, like most websites, and also utilize JavaScript to an extent, however the website works perfectly fine without JavaScript as well.

3) The web-application is built on a framework called Play. Play is a web framework with the intention to build web-applications therein, which for us was perfect. Play is “lightweight, stateless and web-friendly”. [2] Play is built on Akka [3], which is a toolkit to build concurrent and distributed applications on the JVM, which is exactly what we want when we built this web-application.

The website uses, in addition to the Play framework which it is built upon, the bootstrap framework which is a HTTP, CSS and JS framework that is utilized for its convenience as well as its ability to create responsive websites easily. [4] The responsiveness is very important to us, since we feel like our particular product will be used a lot on smart phones and tablets, and we have opted not to create a special smart-phone application but instead have made the website responsive and all around.

4) Play framework uses a MVC design pattern, and therefore the web-application must conform to this pattern. Play in turn is built on Akka, which is a toolkit that provides the actor design pattern, this gives us the advantage that we also utilize actors in our application to take advantage of the multi-core processor that is in the Raspberry Pi. By utilizing both MVC as well as Actors we can really maximize the performance of the system and use several different techniques to accomplish the goal of creating a stable and fast web-application.

5) In our special case environment constraints exist since this application will run on a raspberry Pi, thereby there is quite a large constraint on the resource usage. However this web-application would be able to run on any machine that can run the Java 8-JVM, and therefore there is not an environment constraint on the application per se, this constraint will not necessarily be a problem for other developers if they choose to use this web-application.

4 Provided Interfaces

The web-application must provide an interface that defines what services the website should provide. I have chosen to call this interface, or trait as they are called in Scala, “UserServices” and it defines what services that needs to be provided to the user of the website.

The application must also provide an interface that defines how a user is authenticated when logging in, and defines how a new session is created, this trait is called “AuthSupport”

Required Interfaces

There is not really any interfaces that the web-application requires to provide its clients with the stated functionality, however one could argue that the interfaces that are provided by the scheduler as well as the database connectivity is “required” to make the product as a whole functional, but we will not discuss those interfaces in any more detail in this specification.

Internal Structure

This system as a whole is made up of 4 components, the web-application, the database, the website and the scheduler. The internal structure is as follows:

- The web-application is the center piece, it is in the web-application that the rest of the components interact with each other and they are built in the web-application on the frameworks that the application provides.
- The website is used by the clients to interact with the system, in our case it is used for writing messages to the display.
- The web-application handles these messages, takes the data from them and writes it to the database, and then sends a refresh message to the scheduler.
- The scheduler fetches the message from the database and takes care of the delivery to the display.

The conceptual architecture is shown below:

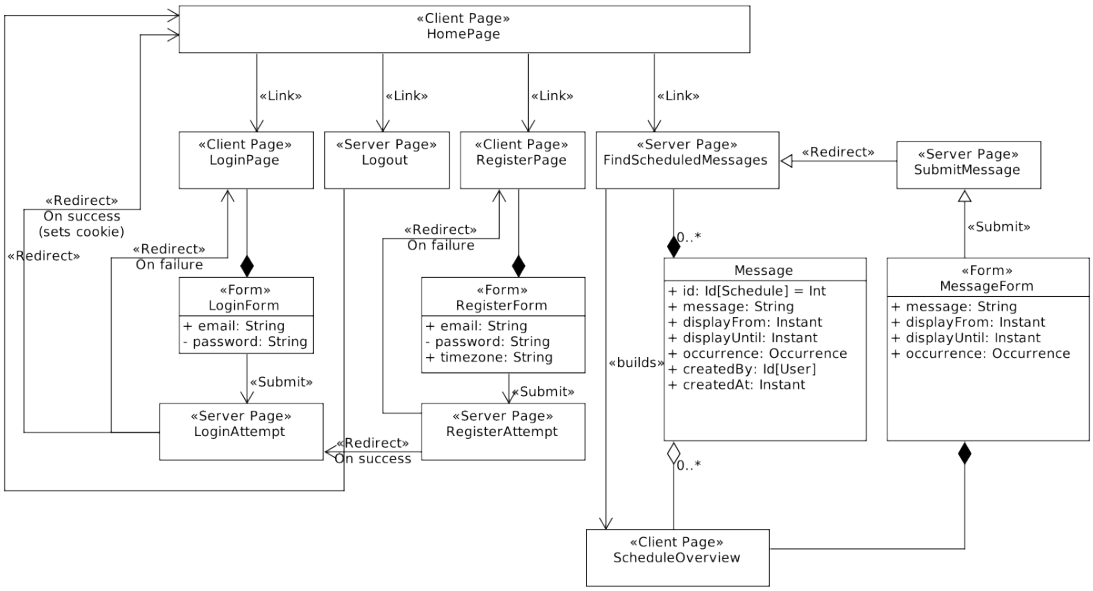
5 Testing the component

The component is tested using ScalaTests, the official Scala testing suite. I have written both unit tests that test small parts of the component as well as a system test which tests the entire web-application and website from start to finish. By utilizing both types of test we can target specific parts of the application and do tests on that part, as well as testing the entire chain of events together, this grants us the ability to make radical changes and try different methods and very easily detect if something breaks whilst doing these changes. Utilizing test driven development has really granted us the ability to try different approaches without fear of not knowing if a change will break the system, since we would detect that immediately thanks to the tests.

6 Known issues and improvements

As mentioned earlier the web-application works as it was planned and how it is supposed to, therefore there are no known issues with the actual back-end, however the websites interface is not that pretty and could be a lot more stylish, the design of the website was unfortunately something that had to be put on hold because of other parts of the project that took a lot longer and was higher prioritized. This is thus something that other developers that took on the development of this application or utilized out application as a basis could work on.

The website has a clear architecture which is shown below, and all the functionality in the architecture is implemented, so improving the design or creating a website that needs the same functionality but has a different purpose would be very easy, extending the architecture below to include even more functionality is doable and encouraged.



7 Using the Component

As previously mentioned, you will need Java 8 (or newer) JVM and Sodium 1.0.12 installed. To compile the JAR-file you will also need a JDK, SBT and Node.js installed. All other dependencies will be handled by SBT.

To register an account and use the message-scheduler you will also have to create a PostgreSQL-database with the name “wilcd” and this has to be reachable by the current user. If you want to rename the database you can update the config file located at “wilcd-ui/application.conf”.

To start the web-application in developer-mode, run the following command in the terminal while being in the directory “wilcd-ui”:

$ sbt run

To create a debian package, do:

$ sbt debian:packageBin

Note that to compile all the sub-systems as the scheduler and the database-connection etc. you have to open your web browser and go to “http://localhost:9000/”. This is not needed for the Debian-package.

Appendix A - References

Use this section to give full reference details for all documents, white papers and books that are referenced by this document.

[1] Sodium URL: https://download.libsodium.org/doc/ Website checked 22/05/2017

[2] Play framework URL: https://www.playframework.com/ Website checked 22/05/2017

[3] Akka toolkit URL: http://akka.io/ Website checked 22/05/2017

[4] Bootstrap URL: http://getbootstrap.com/ Website checked 22/05/2017