



Authentication and Authorization for SAP Sailing Analytics

Bachelorthesis

for the

Bachelor of Science

at Course of Studies Applied Computer Science
at the Cooperative State University DHBW Karlsruhe

by

Benjamin Ebling

September 2014

Time of Project

Student ID, Course

Company

Supervisor in the Company

Reviewer

12 Weeks

4225565, TINF11B2

SAP SE, Walldorf

Dr. Axel Uhl

Renate Ebel

Restriction Notice

This report contains confidential information of

SAP SE

Dietmar-Hopp-Allee 16

69190 Walldorf

It may be used for examination purposes as a performance record of the department of Applied Computer Science at the University of Cooperative Education Karlsruhe. The content has to be treated confidentially. Duplication and publication of this report – as a whole or in extracts – is not allowed without the explicit printed authorization of SAP SE.

Author's declaration

Unless otherwise indicated in the text or references, or acknowledged above, this thesis is entirely the product of my own scholarly work. This thesis has not been submitted either in whole or part, for a degree at this or any other university or institution. This is to certify that the printed version is equivalent to the submitted electronic one.

Karlsruhe, September 2014

Benjamin Ebling

Abstract

The sailing sport has improved a lot in the last few years through application of new technologies. But not only the sailors themselves but also spectators, trainers and moderators experience a better understanding of the sailing races through the improvement of technologies. One of those technologies is the SAP Sailing Analytics application. It is a web application that collects GPS and wind data and processes these data for better understanding. The obtained data is presented on a web page in form of graphics and charts, which show information like the course which has been sailed, rankings and information about maneuvers. This thesis deals with the attempt to improve the application with user management in terms of authentication and authorization.

Zusammenfassung

Der Segelsport hat sich in den letzten Jahren deutlich durch neue Technologien verbessert. Aber nicht nur die Segler selber, sondern auch Zuschauer, Trainer und Moderatoren erfahren ein besseres Verständnis der Segelrennen durch den Fortschritt von Technologien. Eine dieser Technologien ist die SAP Sailing Analytics Anwendung. Sie ist eine Webanwendung welche GPS und Winddaten sammelt und zum besseren Verständnis verarbeitet. Die gewonnenen Daten werden auf einer Webseite in Form von Tabellen und Grafiken dargestellt. Diese zeigen Informationen wie den gesegelten Kurs, Ranglisten und Manöverinformationen an. Diese Arbeit behandelt das Bestreben die Anwendung durch eine Benutzerverwaltung im Sinne von Authentifizierung und Authorisierung zu verbessern.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Purpose of the Report	2
1.3	Structure	2
2	SAP Sailing Analytics	3
3	Requirements & Concept	7
3.1	Given Technologies	8
3.1.1	OSGi	8
3.1.2	Google Web Toolkit	10
3.2	Requirements	11
	Functional Requirements	11
	Non-functional Requirements	12
3.3	Technical Concepts	13
3.3.1	Application Environment	13
3.3.2	Security Frameworks	16
	Spring Security	16
	Apache Shiro	17
	Own solution	17
3.3.3	The OAuth 2.0 Authorization Framework	18
	Scribe	19
3.4	Evaluation of Technologies	20
4	Implementation	21
4.1	The Security Service	22
4.2	Integration of Apache Shiro	29
4.2.1	Configuration	29
	The main section	30
	Realms	32
4.2.2	Request Filter	32

4.3	The User Store	33
4.3.1	MongoDB Implementation	33
	Mongo Object Factory	33
	Domain Object Factory	34
4.4	The User Interface	36
4.4.1	Login Panel	36
	Login	37
	Register	38
	User details	40
4.4.2	User Management Page	41
4.4.3	Login and Registration and Pages	42
5	Conclusion & Outlook	43
	Figures	i
	Tables	ii
	Listings	iii
	Bibliography	iv
	Acronyms	v
	Appendix	vi
6	Appendix	vi
6.1	Listings	vi
6.2	Licenses	viii
6.2.1	MIT license	viii

1 Introduction

1.1 Motivation

The sun is burning on the skin, shouts echo across the water and the sailing boats are cutting through the waves. From the outside it is difficult to recognize what is going on. A sailing competition is taking place, but no one knows what is really happening. Therefore the SAP Sailing Analytics have been invented. They collect the gps data from the sailing boats, analyze it, calculate rankings and maneuvers and present those information over the Internet to anyone interested.

“Historically, it has been almost impossible for spectators to follow a sailing race. It is a costly challenge to get telling images of the water, and it is a challenge to make the spectators understand the dynamic nature of the game. But with the advances in GPS tracking technology, cutting edge analytics, and 3D animations, it is now possible to dive into the action of a sailing race, and better understand what makes a boat win.” - *Marcus Baur, June 2011*

The group of those interested people is very diverse. Just to name the common ones you have sailors, trainers, race committee, spectators with and without knowledge about sailing and a lot of other groups more or less involved. All of them want to have different information about the competition, which brings many different requirements to the application.

The first step to gain this fitted information is to determine the user who is currently using the application. The user provides identity information to the application and the application checks the validity. This process is called authentication.

When the user is authenticated the application can start collecting information for this user. With knowledge about the current user the application can check access rights to the different resources (authorization) and only deliver the required information to the user.

To offer this functionality the application needs a user management. A user management can provide authentication and authorization as well as many other related features like storing user comments, sending messages and further communication tools.

Thus the idea was to extend the SAP Sailing Analytics by authentication and authorization through a user management to provide the user with better information.

1.2 Purpose of the Report

The purpose of this report is to explain the development of the authentication and authorization for the SAP Sailing Analytics. How it started with the idea and the requirements and how the idea was developed and finally implemented.

1.3 Structure

The first part describes the SAP Sailing Analytics application. In the next chapter the concepts behind the project is explained. It starts with the technologies that were given through the SAP Sailing Analytics and goes on with the requirements. Then the technical concept will be shown and some required technologies are demonstrated. Finally the demonstrated technologies are evaluated.

The forth chapter covers the implementation of the project. First the Security Service will be explained, followed by the integration of the Apache Shiro framework. The next part covers the storing functionality and finally the user interface will be pictured.

The last chapter draws a conclusion and shows some possibilities to continue and improve the solution.

2 SAP Sailing Analytics



Figure 2.1: SAP Sailing Analytics Homepage

“By delivering real time analytics around race rankings, speeds, maneuvers, bearings etc., SAP is helping to simplify and demystify the complex sport of sailing.

SAP Sailing Analytics provide insights and transparency to the world of sailing by utilizing Cloud and In-Memory Technology, processing GPS and wind measurement data in real time and visualizing contents in various frontends accessible from anywhere.”2.1

This is a short description found on the SAP Sailing Analytics homepage bringing its purpose to the point.

The SAP Sailing Analytics are developed to support everyone involved in a sailing race. The homepage can be found at <http://www.sapsailing.com> and displays the latest sailing event which has been tracked and analyzed with the software.

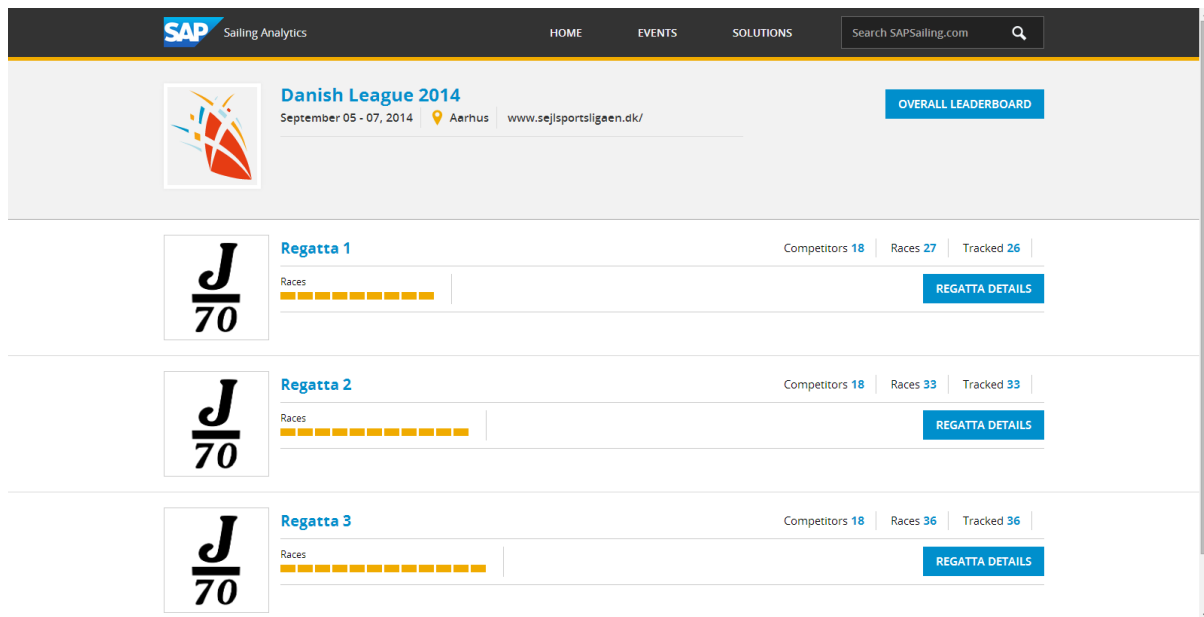


Figure 2.2: SAP Sailing Analytics event details

Following the “More info” button brings you to a page (figure 2.2) that displays a list of regattas at this event. It shows some quick information like the number of competitors, the number of total races in this regatta and how many of those races have been tracked.

The “Regatta details” button brings you to the page shown in figure 2.3. It displays all races that occurred in this regatta. Each race is provided with information like when it took place, who the winner was, if gps data and wind data has been collected and if a video or audio stream is available.

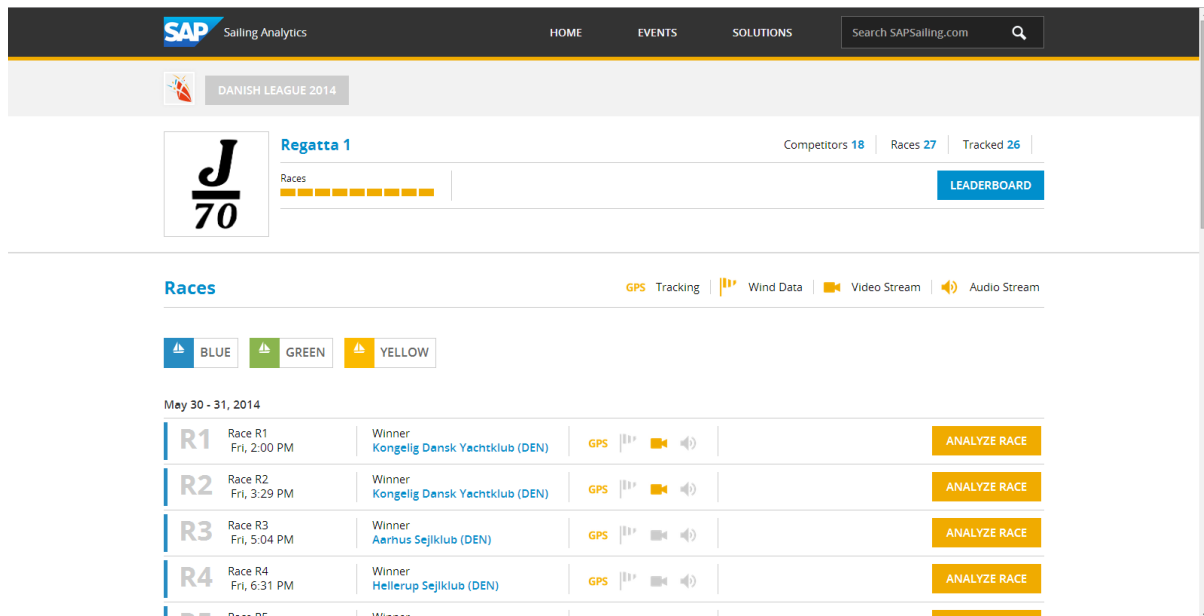


Figure 2.3: SAP Sailing Analytics regatta details

The “Analyze race” button now brings you to the page with the core functionality of the SAP Sailing Analytics, the “Raceboard”.

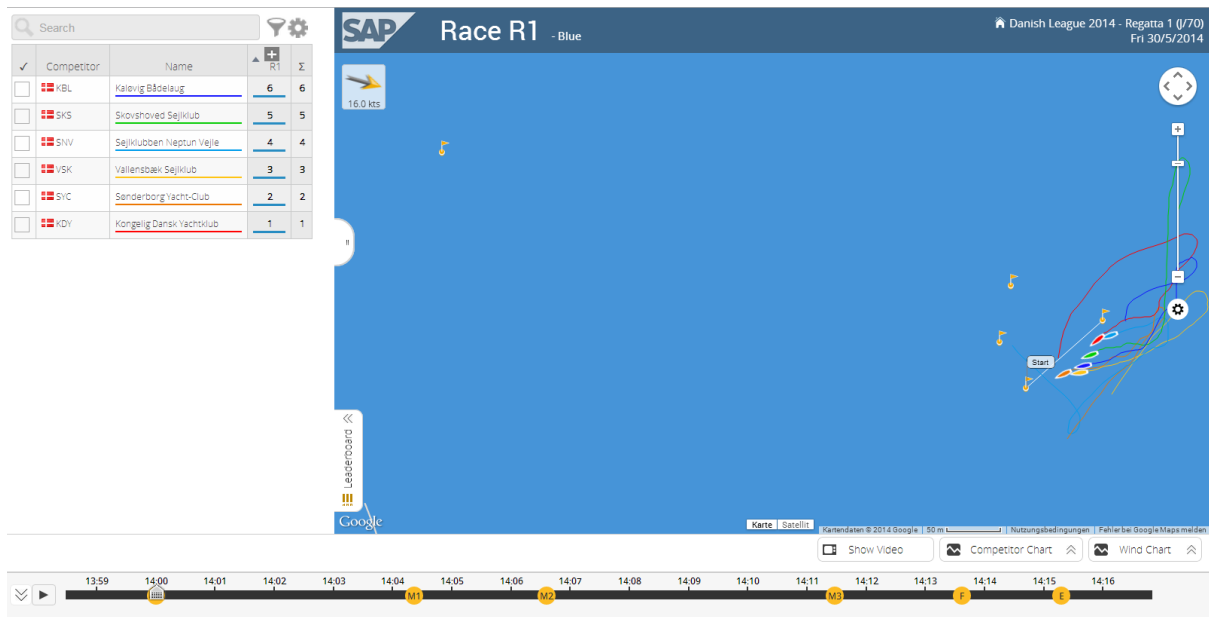


Figure 2.4: SAP Sailing Analytics raceboard

The Race board shows a race in detail. The race can be in replay mode (it happened in the past) or in life mode (the race is currently taking place on the water). The life mode can put certain restrictions on the controls (e.g. you cannot speed up a race in life mode). It consists of three parts:

- The **leaderboard** is the table at the left side of the page. It shows all competitors of the current race. This includes information about the total ranking, the names of the competitors and the position in the current race. The table can be expanded and configured to display information like speed, maneuvers, distance to leader and many more things.
- The **map** at the right side displays a google map with overlays of the boats with tracks, buoys and the wind.
- The **timeline** at the bottom of the page controls the progress of the displayed race. It can play and pause the race. Furthermore the play speed can be increased and by clicking on a certain point on the timeline the race will jump to this moment. The yellow marks indicate the mark rounding of the leading ship in the race.

3 Requirements & Concept

In this chapter I want to go into detail about the task of this project and the requirements and problems that came along with it.

First I want to go into detail about what security, authentication and authorization is:

Security: Security is a term that can relate to many different things like the protection of a person or the encryption of a password and generally describes “The state of being free from danger or threat”[4]. It becomes more and more precise the better the context is described. In the context of Java Scott Oaks [10, p. 2] assigns the following attributes to a possible secure Java program:

- Safe from malevolent programs
- Non-intrusive
- Authenticated
- Encrypted
- Audited
- Well-defined
- Verified
- Well-behaved
- ...

Authentication: Authentication is a process where the currently used application tries to identify the user. While the user is using an application running on his computer, requests are sent to the SAP Sailing Analytics application which is running on a server. Because the server wants to know who sent those requests and makes sure the requests are not corrupted in anyway it tries to authenticate the user.

“Its basic purpose is to prevent fraudulent connection requests”(S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, 1988)[2]

Authorization: The check, if the user of the application is allowed to access certain information or data source is called authorization. This is done by comparing the roles and permissions granted to the user with the roles and permissions required for the access of the resource.

Because the SAP Sailing Analytics come along with some technologies that bring some requirements with them they will be looked at firstly. Then the requirements will be described and the resultant concept will be explained.

3.1 Given Technologies

The Open Service Gateway Initiative (OSGi) specification and the Google Web Toolkit (GWT) are both technologies used in the SAP Sailing Analytics and bring both features and requirements to the scope of this project. Therefore they will be explained in detail in the following subsection.

3.1.1 OSGi

According to Richard Nicholson[3], CEO and Founder of Paremus, agile development methodologies are becoming more and more popular. He also states that many companies focus on the agile processes but do not pay attention to the structure of their applications. In his opinion a modular code base has a “fundamental importance”.

Therefore “[..] many agile initiatives fail to fully deliver the expected business benefits”.

The OSGi specification describes a modular system for Java and, amongst other things, the implementation, documentation and deployment of components. The application is divided in bundles. Each bundle can separately be installed, started, updated, stopped and uninstalled.

As the OSGi technology itself is only a specification there is a need for an implementation of it (also called OSGi runtime). In the scope of the SAP Sailing Analytics the Equinox technology was used. It provides server capabilities and follows the OSGi

specifications. This means that the this software is started on a computer that serves as a server and allows to install bundles. It also deploys the web capable bundles to a web application technology which makes the bundles available to the Internet.

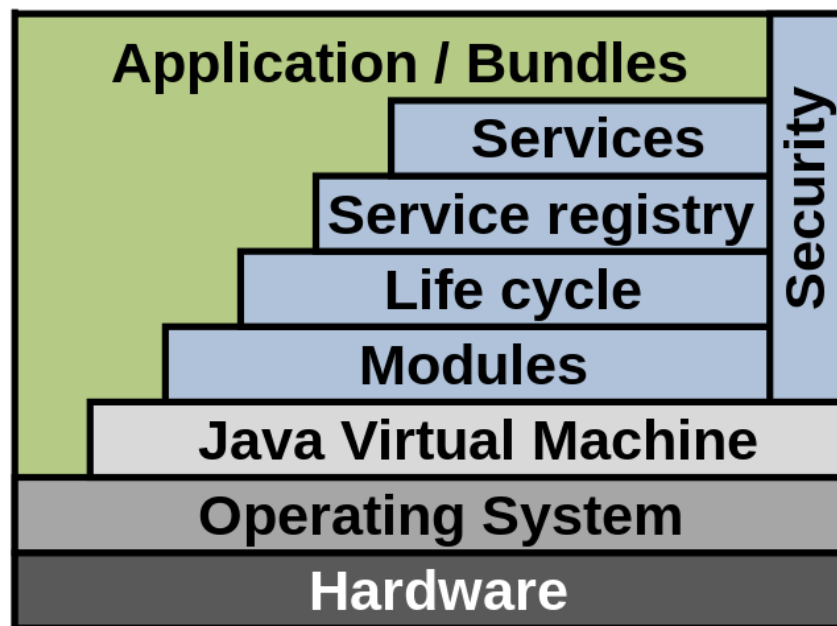


Figure 3.1: OSGi framework

Figure 3.1 shows a more detailed architecture of the OSGi framework. The green part is the application itself that uses the framework. The framework with its layers is colored blue and the grey part is the environment the OSGi runtime is running in.

- **Application / Bundles:** Bundles contain the application in form of Java Archive files. They can for example include java code, other Java Archives, property files and other ones like text files and images that are needed to run the application.
- **Services** Services can be registered and they make it possible to share information with each other in form of Plain Old Java Objects (POJOs).
- **Service Registry** The Service Registry manages the services.
- **Life cycle** The Interface for the management of the bundles. It provides features like installing, starting stopping, updating and uninstalling bundles.
- **Modules** The Modules describe how a bundle can import and export code from other bundles.
- **Security** This layer manages the security of the bundles and controls the access between the bundles.

3.1.2 Google Web Toolkit

To make an application available to the Internet it needs an User Interface (UI) that can be run in the browser. This browser application needs a possibility to communicate with the server. The Google Web Toolkit is taking care of these requirements. It is a toolkit that allows to write an application in java and then converts the client part of the application to JavaScript, so it can be run in a browser. It also defines the Application programming interface (API) between the java code on the server and the JavaScript code in the client, so they can communicate with each other. GWT also brings a couple of ready-to-use UI components like Buttons and, Labels and Layoutpanels.

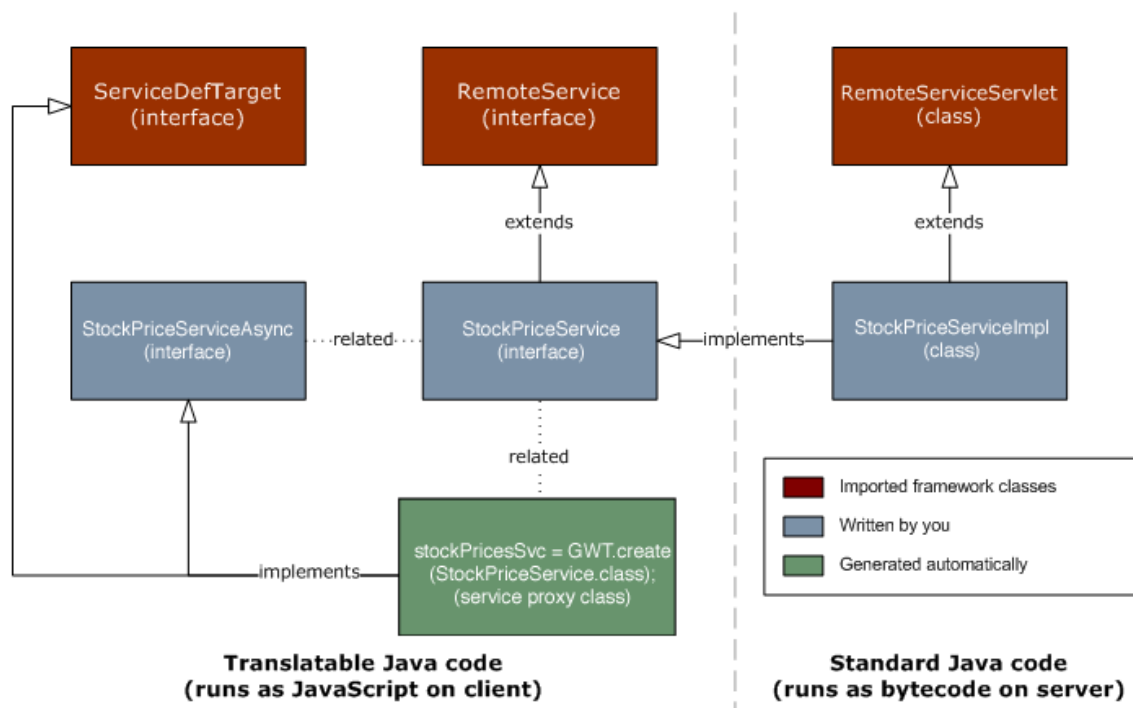


Figure 3.2: GWT Anatomy of Services

3.2 Requirements

The requirements split into 3 main points, the authentication, the authorization and additional features.

The authentication is the part where the application identifies the user. This can either happen with a simple login that requires username and password or with OAuth authentication over an OAuth provider like facebook or google. To identify a user the application first needs to know about this user and manage him.

The authorization controls the access to all resources. This includes the access to web resources which are identified through the url pattern as well as Java Objects in the software code. The access can be checked with roles (e.g. does the user have an “admin” role) or with permissions (e.g. is the user allowed to create another user).

On top of that user preferences and other user related data needs to be stored. The term user management is used for the combination of those features.

Functional Requirements

- Authentication
 - User management (create, read update and delete user)
 - User login and logout.
- Authorization
 - Access Control
 - Restriction of resources.
 - Role and permission management for users.
- Additional features
 - Settings management
 - Storage of user related data (e.g. comments)
 - Storage of user preferences

Non-functional Requirements

In addition the project brings non-functional requirements. The noteworthy requirements are:

- **Accessibility:** It should be made safe that everywhere in the application the user management can be accessed.
- **Scalability:** The user management should work for 10 users as well as for 1000 users or more,
- **Safety:** All user and application information should be protected against fraudulent attacks.
- **Documentation & Maintainability:** The user management should consist of clean code that is documented, so that the work can easily be continued by another developer.

3.3 Technical Concepts

This section looks at the concept behind the authorization and authentication, the requirements and explains the technologies that were used. Some of those technologies like OSGi and GWT were already given by the SAP Sailing Analytics.

3.3.1 Application Environment

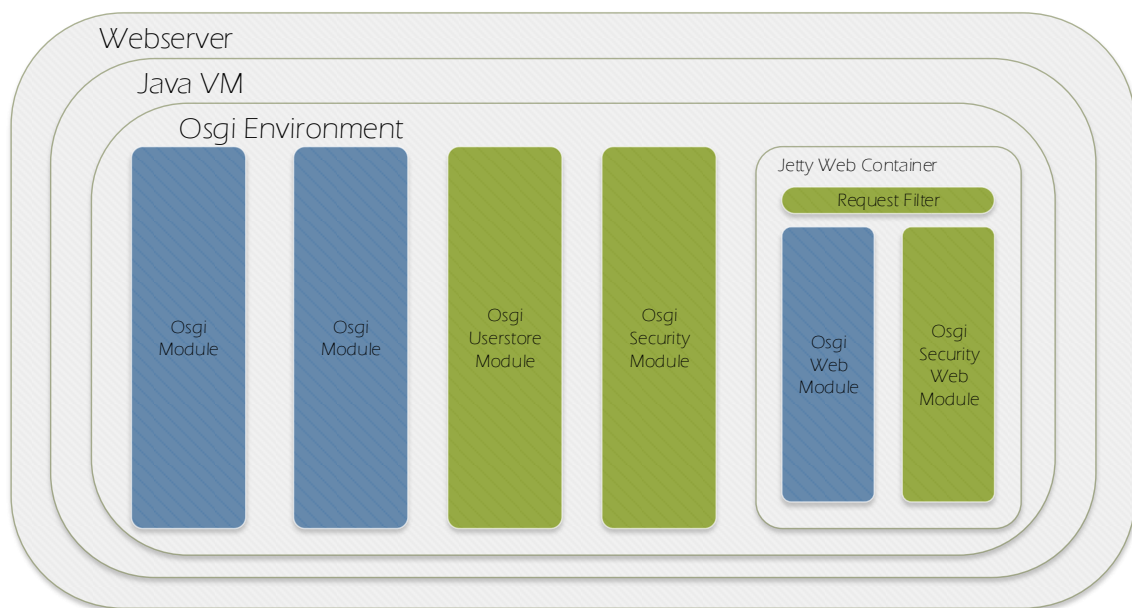


Figure 3.3: Application Environment

Figure 3.3 shows the simplified architecture of the SAP Sailing Analytics (blue) and the planned architecture to integrate the user management (green) which was developed in the context of this bachelor thesis.

The project is running on a server in the web. Because the project is written in Java it needs a Java Virtual Machine (JVM) to run on a computer. In the JVM an OSGi environment is started to manage the different modules of the project.

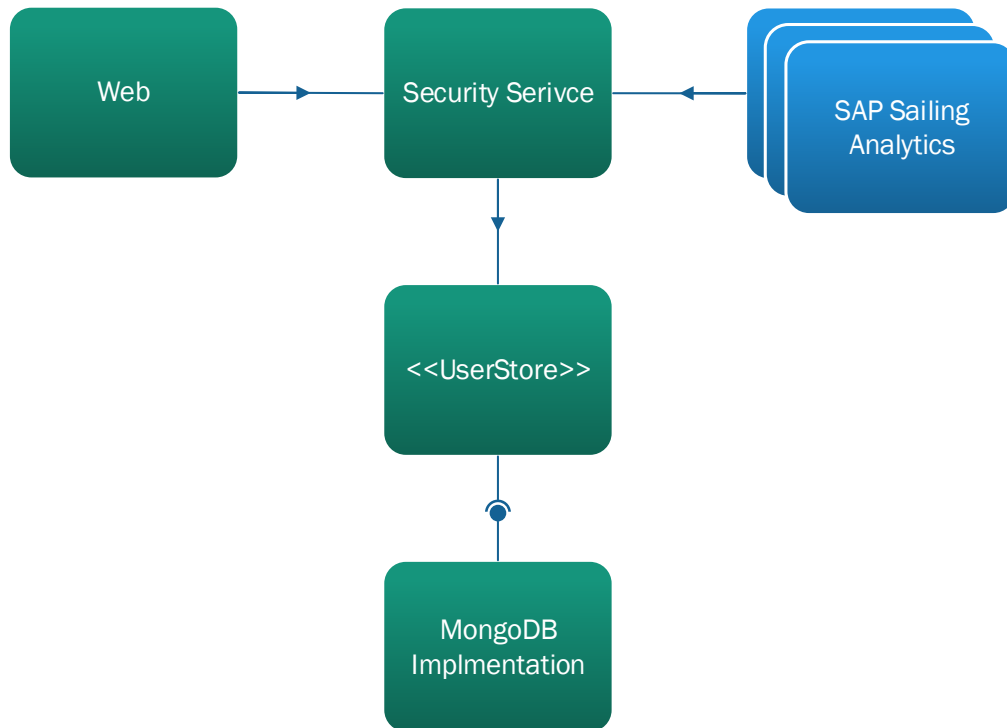


Figure 3.4: Module Architecture

The user management will integrate into the SAP Sailing Analytics in form of additional OSGi modules:

- **OSGI Security Module** This module is the core of the user management and provides the project with an OSGi service called security service enabling the application to call all the relevant features.
- **OSGI UserStore Module** The UserStore module is called from the security module and provides an API for all kinds of data storing. This includes a list of registered users, the application settings, the current user, sessions and similar data. Because it only provides the interface an implementation is required to use the User Store.
- **OSGI MongoDB¹ Implementation Module** This is a specific implementation of the User Store API. The data will be stored in the application cache and will be written to the Mongo database used by the SAP Sailing Analytics for backup purposes.

¹ The Mongo database will be explained in chapter 4.3

- **OSGi Web Module** The web module is responsible for the user interface in form of web pages and a web service interface to handle security functionality with gadgets that cannot use the user interface.

3.3.2 Security Frameworks

There are two different possibilities to provide authentication and authorization: use an existing framework and fit it to the needs or develop an own solution.

There are many security frameworks available and little information about which what should be preferred one or the other way. Spring Security and Apache Shiro (Shiro) are two of those frameworks and were looked at in the context of this project.

Spring Security

The Spring Security framework's main purpose is to secure web applications that are based on the Java Platform Enterprise Edition[5]. It is a powerful and highly customizable authentication and access-control framework[6].

On the web page[6] it claims to have the following features

- Comprehensive and extensible support for both Authentication and Authorization
- Protection against attacks like session fixation, clickjacking, cross site request forgery, etc
- Servlet API integration
- Optional integration with Spring Web MVC

The first point exactly relates to the needs of this project. Coming with improved safety through protection against different kinds of attack is also a welcoming point. The Servlet Api integration is not needed in the frame of the project but may come in handy in the future. The last point does not bring any advantages as the SAP Sailing Analytics are not developed with Spring Web MVC.

Apache Shiro

Apache Shiro (Shiro) is a security framework for and written in Java. It was build to provide a broad set of features without the help of other libraries.

Figure 3.5 shows the features and the focus of Shiro:

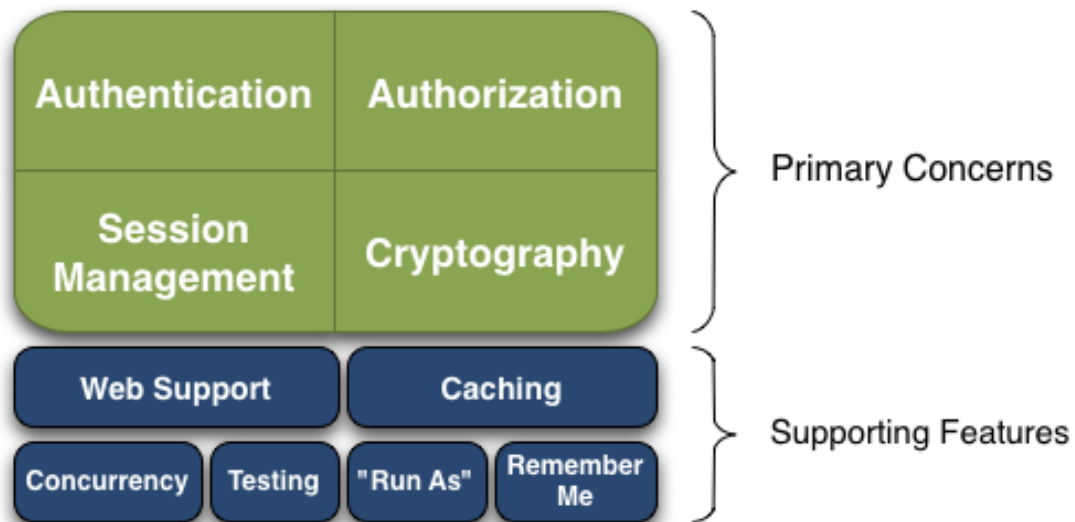


Figure 3.5: Apache Shiro Features

The main features support our requirements for authentication and authorization. Also it provides session management for web application and crypthography for the protection of the data. It comes a long with many more small features, which makes the use of the framework much more comfortable.

Own solution

Furthermore an own solution of a security framework was considered, but some research and discussion discovered pretty fast that this would include far more work than was available in the frame of this project and given that there are good frameworks available the work would be unnecessary work.

3.3.3 The OAuth 2.0 Authorization Framework

“The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.”[9]

The OAuth 2.0 Authorization Framework (OAuth) enables the SAP Sailing Analytics to authenticate the current user by asking an OAuth provider like facebook and google+ for information about the current user.

The process of an OAuth authentication.

1. The client tells the SAP Sailing Analytics that he wants to login with an OAuth provider (e.g. facebook).
2. The server answers with an authentication url(e.g. the facebookpage), its OAuth id and a callback url, when the authentication with the OAuth provider is done.
3. The client opens the authentication url, logs in with the OAuth provider and tells the provider that the user wants to allow the SAP Sailing Analytics to use the account.
4. The OAuth provider grants access and tells the client to go back to the SAP Sailing Analytics.
5. The client tells the SAP Sailing Analytics thatthe user allowed the SAP Sailing Analytics to use the Oauthprovider and gives it the authentication code.
6. The SAP Sailing Analytics tells the OAuth provider itsid, its secret (similar to a password) and the authentication code.
7. The OAuth provider grants access forthe SAP Sailing Analytics and gives it a token. With this token the SAP Sailing Analytics can now make further requests to ask for any desired information.

Scribe

Scribe¹ is a library written in Java that supports the process of OAuth authentication. Is released under the MIT license² which allows us to use it in this project.

The library collects the required information and builds an http request with it. Every OAuth provider requires a different content for the http request. Therefore Scribe builds the request depending on the provider.

¹ The website can be found here <https://github.com/fernandezpablo85/scribe-java>

² The license can be found in the Appendix

3.4 Evaluation of Technologies

I picked those two security frameworks, because they are both developed by big, famous software companies / foundations and, as I could not find any literature on java security frameworks, I searched the web and many people in one of the biggest java communities (“stackoverflow”) recommend using either one of those. The text on the slide shows what they promise to do on their websites. There are many more (mostly smaller, less known) security frameworks and there is few information available about which one should be preferred. Taking a closer look at those frameworks I developed a demo project with each of them. With those demo projects I got a good impression of what security features they provide and how they integrate with the Sailing Analytics. To following chart shows the information I could win from those demo projects.

	Apache Shiro	Spring Security	Own Solution
Size	Lightweight	Big	Minimum
Features	All needed	Many features	All needed
Oauth	With Scribe or similar	Supported	With Scribe or similar
Access Control	Code / Annotation	Code / Annotation	Code (Annotation)
Documentation	Yes	Yes+community	No
Complexity / Impl. Effort	Moderate	High	Moderate-High
success rate	Medium	Slow	Fast

Figure 3.6: Evaluation of security frameworks

4 Implementation

The implementation can be divided in three main parts:

- The **Security Service**: This is the “core” of the implementation. It provides the whole application with usermanagement and security features. It perform the access checks, manages the users and provides all kind of user data and settings.
- The **User Store**: The user store is the part that is responsible for storing all the data either in the cache or persisting it to the disk. In most cases the cache is the main store and the persisted data is only for backup purposes.
- The **User Interface (UI)**: The UI is what the end user can see in the browser. In this case it contains pages for user login, registration, a login panel that can be embedded in other pages and a user management page. The first three of those are available to everyone and provide the features for a typical authentication flow in an application.

4.1 The Security Service

The Security Service is the the main part providing all the relevant logic. It comes in form of the OSGi module called “com.sap.sse.security”. The Security Service itself is an OSGi service and created and registered on bundle startup (Listing4.1).

```
securityService = new SecurityServiceImpl();
registration = context.registerService(SecurityService.class.getName(),
securityService, null);
```

Listing 4.1: Creation of the Security Service

When the Security Service is created other bundles can receive the service through the OSGi service registry (Listing4.2).

```
ServiceReference<?> serviceReference = context.getServiceReference(
    SecurityService.class.getName());
securityService = (SecurityService) context.getService(serviceReference);
```

Listing 4.2: Receive the the Security Service

The “shiro.ini” file, which contains the basic Shiro configuration, is loaded in the constructor of the service as shown in Listing4.4.

```
shiroConfiguration.load(SecurityServiceImpl.class.getResourceAsStream("/
shiro.ini"));
```

Listing 4.3: Loading the Shiro configuration file

When the loading is done further information from the User Store is loaded and inserted into the configuration. Finally a factory for the Security Manager of the Shiro framework is created. With the factory a new Security Manager is created and assigned to the class variable. The Security Manager is the core class of the Apache Shiro framework.

```
Factory<SecurityManager> factory = new WebIniSecurityManagerFactory(
    shiroConfiguration);
SecurityManager securityManager = factory.getInstance();
this.securityManager = securityManager;
```

Listing 4.4: Loading the Shiro configuration file

The Security Service is the core of the user management and provides the following methods to the software:

Method	Configuration	Description
getSecurityManager	Parameter	-
	Returns	The Security Manager
	Comment	<p>Every osgi module should use this method to make the security service features available. Example:</p> <pre>SecurityUtils. setSecurityManager(securityService. getSecurityManager());</pre>
getUserList	Parameter	-
	Returns	A Collection of all registered users.
	Comment	This method returns all registered users.
getUserByName	Parameter	String name
	Returns	A User.
	Comment	Searches for a user with the specified name. The method returns null if no user was found.

getCurrentUser	Parameter	-
	Returns	A User.
	Comment	The user, who send the request which is currently processed by the system, is returned.
login	Parameter	String username String password boolean rememberMe
	Returns	A URL.
	Comment	The URL that the user has visited before he tried to log in so he can be redirected back to that page.
getAuthenticationUrl	Parameter	Credential credential
	Returns	A URL.
	Comment	This method provides the authentication url for the selected OAuth provider. It throws a UserManagerException if something went wrong (e.g. not provider was selected).
verifySocialUser	Parameter	Credential credential boolean rememberMe
	Returns	A User.
	Comment	Tries to authenticate the user by contacting the OAuth provider. It throws a UserManagerException if the provided information are wrong or the web service of the provider could not be reached.

logout	Parameter	
	Returns	-
	Comment	Logs out the current user. He will now be recognized as anonymous by the application.
createSimpleUser	Parameter	String name String email String password
	Returns	A user.
	Comment	Creates an account with the user-name, email and the password. The password will be encrypted for security reasons. It throws a UserManagementException if the user could not be created.
createSocialUser	Parameter	String name SocialUserAccount socialUserAccount
	Returns	A user.
	Comment	Creates an account by contacting the OAuth provider. throws UserManagementException
deleteUser	Parameter	String username
	Returns	-
	Comment	Deletes the user with the specified username. Does nothing if the user could not be found.
getRolesFromUser	Parameter	String username

	Returns	A Set of Strings conaining the roles.
	Comment	If the user with the specified name can be found, the method returns all roles for this user.
addRoleForUser	Parameter	String username String role
	Returns	-
	Comment	Adds the specified role to the user.
removeRoleFromUser	Parameter	String username String role
	Returns	-
	Comment	Removes the specified role to the user.
addSetting	Parameter	String key Class<?> clazz
	Returns	-
	Comment	Adds a setting option with the type of the specified class to the application settings.
setSetting	Parameter	String key Object setting
	Returns	-
	Comment	Sets the value for the specified setting.
getSetting	Parameter	String key Class<T> clazz
	Returns	An object of the class "clazz".

	Comment	Returns the stored value for the setting.
getAllSettings	Parameter	
	Returns	A Map with the setting keys and the corresponding values.
	Comment	Returns the application settings.
getAllSettingTypes	Parameter	
	Returns	A Map with the setting keys and the corresponding classes.
	Comment	Returns the classes of the application settings.
storeUserSetting	Parameter	String username String key String value
	Returns	-
	Comment	Stores the setting for the specified user.
storeUserSetting	Parameter	String username String key
	Returns	A String value of the setting.
	Comment	Reads the value for the setting, which was stored for this user.
removeUserSetting	Parameter	String username String key
	Returns	-
	Comment	Removes the setting of the stored settings for this user.

getUserSettings	Parameter	String username
	Returns	A Map with the setting String key and a corresponding String value.
	Comment	This method returns all settings for the specified user.
sendVerificationEmail	Parameter	String username
	Returns	-
	Comment	Sends an email to the specified user which contains a link. By clicking on the link the user can mark the account as verified.
activateAccount	Parameter	String username String verificationCode
	Returns	-
	Comment	This method is usually triggered through the click on the link generated by the <i>sendVerificationEmail</i> method. It marks the account as verified.

Table 4.1: Security Service interface

4.2 Integration of Apache Shiro

This section addresses the the integration and configuration of the Apache Shiro framework (short: Shiro).

4.2.1 Configuration

Most of the configuration for the Shiro framework happens through a file called “shiro.ini”. The file is separated in four sections which are marked in the file with surrounding square brackets.

- **main:** The main section configures all the Java classes that are responsible for different aspects. They will be explained further below.
- **users:** In this section the developer can create static users for the application. In our case all the user are created dynamically. Therefore this section is empty.
- **roles:** The same goes for the roles.
- **urls:** In this section the access rights for url patterns can be configured. Most of them will be configured dynamically. We only want to make sure the user can always access the login page. Because of that we assign the anon filter (anon stands for anonymous) to the login page, which will grant everyone acces.

```
[urls]
/Login.html = anon
```

Listing 4.5: Url configuration

As mentioned above the file is located in the module “com.sap.sse.security” and is loaded on the module startup.

The main section

The first part of the main section is the configuration of the credentials matcher. The credentials matcher is called when a user tries to authenticate with a username and a password (not when he tries to authenticate with oauth!). In this case the `Sha256CredentialMatcher` is set. It uses the Sha256 hashing algorithm for the password. This is a security measure in case someone (illegally) reads the database they cannot read the password. The flag “`storedCredentialsHexEncoded`” determines whether the credentials are stored in Hex encoding or in this case Base64 encoding (`false`). Furthermore the hashing operation is repeated 1024 times to make it this many times harder for someone to hack the password (but this will also slightly slow down the authentication process).

```
credentialsMatcher =  
org.apache.shiro.authc.credential.Sha256CredentialsMatcher  
    credentialsMatcher.storedCredentialsHexEncoded = false  
credentialsMatcher.hashIterations = 1024
```

Listing 4.6: Crednetials Matcher Configuration

The realms are configured next. Realms are Java classes that provide Shiro with authentication and authorization information. For each possibility an extra realm has to be configured and assigned to the security manager. In this case we need a realm for the username/password authentication and a realm for the oauth authentication. We also assign the before created credentials matcher to the username/password realm.

```
upRealm = com.sap.sse.security.UsernamePasswordRealm  
upRealm.credentialsMatcher = $credentialsMatcher  
oauthRealm = com.sap.sse.security.OAuthRealm  
securityManager.realms = $upRealm, $oauthRealm
```

Listing 4.7: Realm configuration

Next a session manager is configured. Because the user connect with this application over the internet this is a web session manager. It handles http sessions and configures the cookies for the client.

```
sessionManager = com.sap.sse.security.SecurityWebSessionManager  
securityManager.sessionManager = $sessionManager
```

Listing 4.8: Web Session Manager

Finally a request filter is configured. The request filter processes all request from any user. It checks the authentication status and if an authentication is required it sends the user to the specified login URL. If the login was successful and no redirection URL is specified the user is send to the success URL afterwards.

```
authc = com.sap.sse.security.CustomFilter  
authc.loginUrl = /security/Login.html  
authc.successUrl = /security/UserManagement.html
```

Listing 4.9: Request Filter

Realms

Realms are Java objects that provide the Apache Shiro framework with information about the current user.

There are two methods: **doGetAuthorizationInfo** provides the authorization information and **doGetAuthenticationInfo** provides the authentication information. For each authentication method exists one realm. Because this project should provide a username / password authentication and an OAuth authentication, one realm for each is available.

An example of the implementation of the *doGetAuthenticationInfo* method can be found in the appendix.

4.2.2 Request Filter

The request filter checks all requests from all users for the required authentication status and redirects them to the login page if necessary. The request filter is configured in the web.xml file. This file contains the basic configuration for a Java web project. Through the entry in this file the authentication check can be done before anything else happens with the request.

4.3 The User Store

The User Store is an OSGi module and a service that is responsible for all kinds of storing and persisting of data. It should only be accessed by the Security Service as any authorization checks are bypassed in the methods. The module also contains the classes for users and accounts. The API contains methods for all sort of storing and reading user management related data.

4.3.1 MongoDB Implementation

As the User Store itself is only an interface describing the needed functionality an implementation of the User Store must be provided. The Sailing Analytics itself store all data in the application cache and only writes the data to the disk for backup purpose. The database that is used to store the data on the disk is called MongoDB.

The User Store implementation follows this pattern and is therefore called MongoDB Implementation. This implementation offers a in-memory cache as well as persistence through the MongoDB. The MongoDB is a database that is freely available on the Internet.

Mongo Object Factory

The Mongo Object Factory is a class that converts all relevant Java Object to Mongo Objects that fit the format of the Database.

The following Listing shows an example for the storage of an user object:

```
DBCollection usersCollection = db.getCollection(CollectionNames.USERS.
    name());
usersCollection.ensureIndex(FieldNames.User.NAME.name());
DBObject dbUser = new BasicDBObject();
DBObject query = new BasicDBObject(FieldNames.User.NAME.name(), user.
    getName());
dbUser.put(FieldNames.User.NAME.name(), user.getName());
dbUser.put(FieldNames.User.EMAIL.name(), user.getEmail());
dbUser.put(FieldNames.User.VERIFIED.name(), user.isVerified());
dbUser.put(FieldNames.User.ACCOUNTS.name(), createAccountMapObject(user
    .getAllAccounts()));
dbUser.put(FieldNames.User.ROLES.name(), user.getRoles());
```



```

dbUser.put(FieldNames.User.SETTINGS.name(), new BasicDBObject(user.
    getSettings()));

usersCollection.update(query, dbUser, /* upsert */ true, /* multi */
    false, WriteConcern.SAFE);

```

Listing 4.10: Mongo Object Transformation

First the database collection, which is responsible for storing all objects of the same kind, is fetched. And the *ensureIndex* method makes sure that no users with the same name can exist. Then a database object is created and filled with the required data. Finally an update with the created object is called on the collection.

Domain Object Factory

The Domain Object Factory reads the required data from the database and transforms it back into Java Objects.

The following Listing shows an example for the reading of a database object:

```

String name = (String) userDBObject.get(FieldNames.User.NAME.name());
if (name == null || name.length() == 0){
    return null;
}
String email = (String) userDBObject.get(FieldNames.User.EMAIL.name());
Boolean verified = (Boolean) userDBObject.get(FieldNames.User.VERIFIED.
    name());
Set<String> roles = new HashSet<String>();
BasicDBList rolesO = (BasicDBList) userDBObject.get(FieldNames.User.
    ROLES.name());
if (rolesO != null){
    for (Object o : rolesO){
        roles.add((String) o);
    }
}
DBObject settings = (DBObject) userDBObject.get(FieldNames.User.
    SETTINGS.name());
DBObject accountsMap = (DBObject) userDBObject.get(FieldNames.User.
    ACCOUNTS.name());
Map<AccountType, Account> accounts = createAccountMapFromDBObject(
    accountsMap);
User result = new User(name, email, settings == null ? new HashMap<
    String, String>() : settings.toMap(), accounts.values());

```

```
result.setVerified(verified == null ? false : verified);  
result.setRoles(roles);  
return result;
```

Listing 4.11: Domain Object Transformation

The Domain Object Factory does pretty much the opposite of the Mongo Object Factory. It reads the database object from the collection and extracts the values from it. With those values a Java object is build and returned.

4.4 The User Interface

The User Interface consist of several standalone web pages and a login panel that can be embedded in other pages. Most user will never set eyes on the pages and will only use the login panel, as it provides all functionality an average user needs. This includes a login with username and password or an oauth provider, a registration form and, in case the user is logged in, a log out button. The login and the registration pages are only for fallback purpose. The user management page is made for the administrators of the current server and can only be accessed of the few users that have the admin role.

4.4.1 Login Panel

The Login Panel is the main UI component and is a component that can be embedded in all pages that wish to provide user management functionality. Its appearance is determined by two attributes:

- **Toggle status:**
Possible states: expanded, collapsed
- **Content type:**
Possible states: login form, user details, registration

Login

Figure 4.1 shows the default Login Panel that appears, if not changed by the developer, in the right, top corner of the page.



Figure 4.1: Login Panel [collapsed, login form]

By clicking on the “Login” label the toggle status will be changed from collapsed to expanded and reveal a login form4.2.

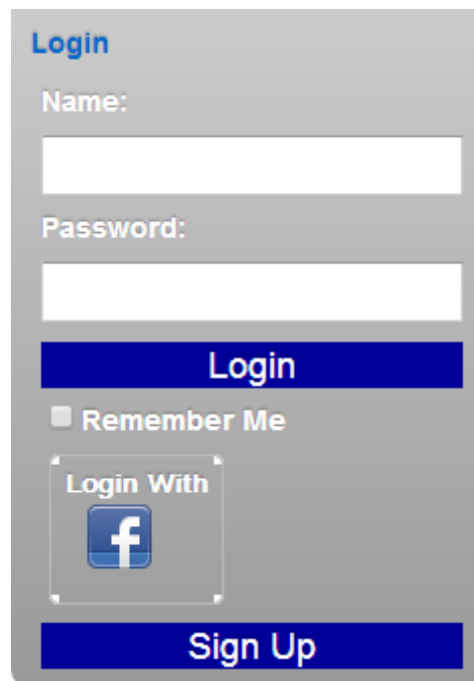
A vertical grey panel with rounded corners. At the top, the word "Login" is in blue. Below it is the label "Name:" followed by a white text input field. Then the label "Password:" followed by another white text input field. Below the password field is a blue button with the word "Login" in white. Underneath the button is a checkbox labeled "Remember Me". Below the checkbox is a button labeled "Login With" with a Facebook 'f' logo. At the bottom is a blue button with the words "Sign Up" in white.

Figure 4.2: Login Panel [expanded, login form]

The content displays a login form where the user can log in with username and password. A checkbox is following the form which determines if the user should be remembered if he leaves the page and comes back later. Below the checkbox is a button for the login with facebook using oauth.

If the user has no account yet he can sign up by clicking on the “Sign Up” button on the bottom of the panel.

Register

The image shows a web form titled 'Login' in blue text. Below the title is a section header 'Create a new account' in bold black text. The form contains four text input fields stacked vertically, each with a light gray border and a light gray placeholder text: 'Name', 'Email', 'Password', and 'Confirm password'. Below the input fields are two blue buttons with white text. The top button is labeled 'Sign Up' and the bottom button is labeled 'Log in'.

Figure 4.3: Login Panel [expanded, register form]

The register form provides text fields for name, email-address and two fields for password (one for confirmation). If the user enters no or invalid data red message boxes will appear to inform the user about the incorrect data (figure4.4). By pressing the “Sign Up” a new account is created for the user (if the name is not already taken) and the user is immediately logged in.

The image shows a login panel with a grey background. At the top left, the word "Login" is written in blue. Below it, the text "Create a new account" is centered in black. The form contains four input fields: "Name", "Email", "Password", and "Confirm password". Below these fields are two blue buttons: "Sign Up" and "Log in". To the left of the form, there are three red rectangular boxes containing the following text: "Please enter a name", "Please enter an email", and "Please enter a password".

Figure 4.4: Login Panel [expanded, register form] with info pop-ups

The messages will disappear if the cursor leaves the registration form or when the user clicks the "Sign Up" button again and the data is correct.

Pressing the "Log In" button will bring the user back to the login form.

User details

The user details show up when the user successfully logs into the application. The login panel will automatically register the change and display the message “Welcome, [name]!” with [name] being the name of the user.



Figure 4.5: Login Panel [expanded, user details]

If the user clicks on the label it will expand, also showing the “Logout” button. Pressing this button will bring the user back to the login form.

4.4.2 User Management Page

The user management page is, as the name already suggests, the user interface to manage the users. Furthermore it allows to configure the security settings. Therefore it can only be accessed by an administrator (a user with the role “admin”).

Create User Settings

Filter users...

5-8 of 9

Daniel

Facebook*Benjamin Ebling

Franz

Hans

User details

Name: Facebook*Benjamin Ebling
Email: benjamin.ebling@gmx.de
Verified: false

Send verification email

Facebook-Account

name	Benjamin Ebling
firstname	Benjamin
gender	male
email	benjamin.ebling@gmx.de
provider	Facebook
lastname	Ebling

Roles:

Add role... Add role

admin

Settings

KEY	VALUE
VERIFICATION	8f2ca08b-5953-4749-a

Add setting

Delete user

Figure 4.6: User Management Page

On the left side a list with all registered users can be found. The list can be filter by entering a name or a part of a name in the text field above. When a user is selected the user details are displayed on the right side of the page. The user details also provide button to send a verification mail to the user, add or remove roles and manage user specific settings.

On the top of the page two more buttons can be found. The first one opens a form for creating a user, the second opens a page for security settings of the application. The settings include identification codes for the Oauth providers and access right for different pages and urls of this application.

4.4.3 Login and Registration and Pages

The login and registration pages are a backup for whatever reason the login panel is not used. As they only serve for backup purpose they haven't undergone a visual makeup yet.

The login page offers two textfields for username and password with a login button to do so. Depending on the current login status the page also suggests links to other pages of the application(e.g. the registration page, if the user is not logged in or the user management page if an administrator is logged in).

The registration page has the same content like the registration form in the login panel, but is a page on its own.

5 Conclusion & Outlook

The project started with the analysis of a suitable security framework for the SAP Sailing Analytics. Research and development of small demo projects showed, that the Apache Shiro framework was best fitting for the requirements. With the support of an appropriate framework a solid user management solution could be developed. The SAP Sailing Analytics now supports login with username and password as well as login with facebook (oauth). This is a major requirement for the individual usermanagement. This improvement to the application offers a whole new set of possible other improvements. Despite the fact that there is still much left to be done, the improvement exhibits a big step. Users of the application can now be identified and attributed to create a user profile, which enables the software to let the user “feel home” because he is supplied with better and appropriate information. Other developers can now easily find out about the current state of the user and react to it. All together I would call the project a success because the users can now log in with their very popular facebook accounts and developers can find out about the current users of the application and their actual needs.

These advantages of the user management are currently roughly visible to the end user, but the introduction of the user management opens the way for many new features for the users. Some of those possibilities are:

- **Comments** may be used to leave a permanent opinion on a race or share some interesting information.
- A **life chat** that allows users who are currently spectating a race to communicate with each other.
- A **friend-finding algorithm** may find users that are currently sailing and inform friends spectating the race about it.

This solution enhances the communication between sailors, trainers, moderators and spectators and thereby improves the experience of the sailing sport and extends the sailing knowledge.

List of Figures

2.1	SAP Sailing Analytics Homepage	3
2.2	SAP Sailing Analytics event details	4
2.3	SAP Sailing Analytics regatta details	5
2.4	SAP Sailing Analytics raceboard	6
3.1	OSGi framework	9
3.2	GWT Anatomy of Services	10
3.3	Application Environment	13
3.4	Module Architecture	14
3.5	Apache Shiro Features	17
3.6	Evaluation of security frameworks	20
4.1	Login Panel [collapsed, login form]	37
4.2	Login Panel [expanded, login form]	37
4.3	Login Panel [expanded, register form]	38
4.4	Login Panel [expanded, register form] with info pop-ups	39
4.5	Login Panel [expanded, user details]	40
4.6	User Management Page	41

List of Tables

4.1	Security Service interface	28
-----	--------------------------------------	----

Listings

4.1	Creation of the Security Service	22
4.2	Receive the the Security Service	22
4.3	Loading the Shiro configuration file	22
4.4	Loading the Shiro configuration file	22
4.5	Url configuration	29
4.6	Crednetials Matcher Configuration	30
4.7	Realm configuration	30
4.8	Web Session Manager	30
4.9	Request Filter	31
4.10	Mongo Object Transformation	33
4.11	Domain Object Transformation	34
6.1	Complete method doGetAuthenticationInfo for the OAuth realm	vi

Bibliography

- [1] SAP SE *SAP Sailing Analytics*, website found at <http://www.sapsailing.com/>
- [2] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer 1988 *Kerberos Authentication and Authorization System*, available at <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.7727&rep=rep1&type=pdf>
- [3] Richard Nicholson 2014 *Agility and Modularity: Two sides of the same coin*, available at <http://www.osgi.org/wiki/uploads/Resources/AgilityandModularity2014v2.pdf>
- [4] *Oxford Dictionaries*, webpage available at <http://www.oxforddictionaries.com/definition/english/security>
- [5] Aleksander Dikanski, Roland Steinegger, Sebastian Abeck 2012 *Identification and Implementation of Authentication and Authorization Patterns in the Spring Security Framework*
- [6] Pivotal Software, Inc *Spring Security*, website found at <http://projects.spring.io/spring-security/>
- [7] Nathan A. Good 2010 *Introducing Apache Shiro*, available at <http://www.ibm.com/developerworks/library/wa-apacheshiro/wa-apacheshiro-pdf.pdf>
- [8] Apache Software Foundation *Apache Shiro*, available at <http://shiro.apache.org/index.html>
- [9] D. Hardt 2012, *The OAuth 2.0 Authorization Framework*, available at <http://tools.ietf.org/html/rfc6749>
- [10] Scott Oaks 1998 *Java Security*

Acronyms

API Application programming interface.

GPS Global Positioning System.

GWT Google Web Toolkit.

JVM Java Virtual Machine.

OAuth The OAuth 2.0 Authorization Framework.

OSGi Open Service Gateway Initiative.

POJO Plain Old Java Object.

Shiro Apache Shiro.

UI User Interface.

6 Appendix

6.1 Listings

```
protected AuthenticationInfo doGetAuthenticationInfo(
    AuthenticationToken token) throws AuthenticationException {
    OAuthToken otoken = (OAuthToken) token;
    Credential credential = (Credential) otoken.getCredentials();
    System.out.println("Checking authentication!");
    int authProvider = credential.getAuthProvider();
    logger.info("authProvider: " + authProvider);
    String authProviderName = ClientUtils.getAuthProviderName(
        authProvider);
    logger.info("Verifying social usr from " + authProviderName);
    Token requestToken = null;
    String yahooGuid = null;
    String protectedResourceUrl = ClientUtils.getProctedResourceUrl(
        authProvider);

    requestToken = SessionUtils.getRequestTokenFromSession();
    OAuthService service = null;
    Verifier verifier = null;
    Token accessToken = null;

    if (authProvider != ClientUtils.DEFAULT) {
        try {
            service = getOAuthService(authProvider);
        } catch (Exception e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        verifier = new Verifier(credential.getVerifier());
        logger.info("Requesting access token with requestToken: " +
            requestToken);
        logger.info("verifier=" + verifier);
    }
```

```

        try {
            accessToken = service.getAccessToken(requestToken, verifier
                );
        } catch (Exception e) {
            System.out.println("Error receiving request token:");
            e.printStackTrace();
        }
        if (accessToken == null) {
            logger.error("Could not get Access Token for " +
                authProviderName);
            throw new AuthenticationException("Could not get Access
                Token");
        }
        logger.info("Got the access token: " + accessToken);
        logger.info(" Token: " + accessToken.getToken());
        logger.info(" Secret: " + accessToken.getSecret());
        logger.info(" Raw: " + accessToken.getRawResponse());
    }
    //[...]

    String sessionId = makeRandomString();
    SessionUtils.saveSessionIdToSession(sessionId);
    SessionUtils.saveAuthProviderToSession(authProvider);

    SocialUserAccount socialUser = null;
    SessionUtils.saveAccessTokenToSession(accessToken);
    SessionUtils.saveProtectedResourceUrlToSession(protectedResourceUrl
        );

    logger.info("Getting protected resource");
    logger.info("Protected resource url: " + protectedResourceUrl);
    try {
        OAuthRequest request = new OAuthRequest(Verb.GET,
            protectedResourceUrl);
        service.signRequest(accessToken, request);

        Response response = request.send();
        logger.info("Status code: " + response.getStatusCode());
        logger.info("Body: " + response.getBody());

        String json = response.getBody();
        socialUser = getSocialUserFromJson(json, authProvider);
    } catch (Exception e) {

```

```

        logger.error("Could not retrieve protected resource: " + e);
        throw new RuntimeException("Could not retrieve protected
            resource: " + e);
    }

    if (socialUser.getProperty("NAME") == null){
        throw new AuthenticationException("Username cannot be null!");
    }

    socialUser.setSessionId(sessionId);
    socialUser.setProperty(Social.PROVIDER.name(), authProviderName);

    String socialname = authProviderName + "*" + socialUser.getProperty(
        Social.NAME.name());

    User user = store.getUserByName(socialname);
    if (user == null){
        try {
            user = store.createUser(socialname, socialUser.getProperty(
                Social.EMAIL.name()), socialUser);
        } catch (UserManagementException e) {
            throw new AuthenticationException(e.getMessage());
        }
    }
    SessionUtils.saveUsername(user.getName());

    SimpleAuthenticationInfo sai = new SimpleAuthenticationInfo();
    SimplePrincipalCollection spc = new SimplePrincipalCollection();
    spc.add(otoken.getPrincipal(), otoken.getPrincipal().toString());
    sai.setCredentials(otoken.getCredentials());
    sai.setPrincipals(spc);
    return sai;
}

```

Listing 6.1: Complete method doGetAuthenticationInfo for the OAuth realm

6.2 Licenses

6.2.1 MIT license

The MIT License (MIT)

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.