myVerein

Server Specification

Student Research Project

for the

Bachelor of Science

at Course of Studies Applied Computer Science
at the Cooperative State University Stuttgart

by

Frank Steiler

September 2014

Time of Project	13 Weeks
Student ID, Course	8216767, TINF12A
Company	Hewlett-Packard GmbH, Böblingen
Internship Supervisor	Alfred Becker

Contents

Chapter 1

Document purpose

This document is extending the general Software Requirement Specification for the whole system, specifying details about the server application. This document therefore tracks the complete design of the backend of the system, like the admin panel, the database and every other component needed to run the application.

Chapter 2

Product purpose

The server application is intended to manage all required information of the system. These include the member management as well as the handling of push notification and chat messages.

A user friendly interface is needed to enable an intuitive usage of the system through a web interface.

Chapter 3

Product requirements

The following chapter is addressing the requirements specified in the Software Requirements Specification. The requirements listed within this document are only related to the server application. Some optional and additional requirements are added. These specification include some improvement that have been suggested by IT administrators of several clubs, that I have talked to.

3.1 Obligatory requirements

- The administrator needs to be able to modify the server according to the clubs name etc. during the initial setup
- The administrator needs to be able to modify the access rights of all members of the club
- The administrator needs to be able to schedule an event
- The system needs to ensure a fault tolerant, consistent operation.
- The system needs to be configurable
- The system needs to provide a secure user authentication
- The system needs to handle the login of multiple users at the same time
- The system needs to handle several messages at the same time
- The system should be configurable through a web interface, that enables the administrator to manage user

- The system needs to operate according to the data privacy act
- The system needs to be designed in a way that a user can only access a minimum amount of private data of the other users
- The system needs to be designed to be easily extensible
- The system needs to save and gather the data from a persistent data storage system
- The system needs to be developed in Java 8

3.2 Optional requirements

The following requirements are optional, but their implementation is nice to have.

- The application should have a sophisticated graphical user interface through a web interface
- The administrator should be able to create divisions
- Each user should be able to be part of one or more divisions, to only receive relevant information.
- The system should support private chats for each division
- Invitations to events should be send to divisions
- The user should be able to request access to a division, this access is granted by a higher level user
- The user should be able to share photos that are relevant to the club
- The application should use push notifications to effectively alert the user about incoming messages, news or upcoming events
- The system should be designed to ensure extensibility
- The user should be able to optional create a public profile containing contact information
- The system should be designed to allow a multi-lingual implementation

- The administrator should be able to manage the divisional arrangement via the web interface

3.3 Additional requirements

The following requirements would improve the overall user experience, but their implementation is not business critical.

- The application could have a central news feed, which contains the latest information provided by the administrator
- The administrator could be able to publish relevant information through the web interface
- Approved user besides the administrator could be able to manage user or publish news and events
- The system could send an Email newsletter for members that are not owning a smartphone running iOS, or allow the print out of the newsletter for a non-digital distribution
- The events could support assignment of supporting roles needed during the event
- The events could support voting buttons to find the ideal date for the meeting
- The system could support the download of shared pictures
- The system could support the use of shared pictures within the news of the club
- The system could allow the administrator to share news through the clubs homepage using plugins within Joomla, Wordpress, Facebook or other systems.

3.4 Non-requirements

The following requirements are not in the scope of this product.

- The system is not designed to create a homepage for the club

- The application is not designed to provide access to non-members of a club
- The system should not be used to collect statistics about the users

Chapter 4

License

Since the system is handling personal data and billing information it is very important to create a secure and transparent application, that is trusted by the user and administrators. Therefore an open source release is the best way to gain the trust of user and ensure a quick and effective exposure and fix of critical bugs.

On top of that the project is created within a research project, so the results of the application development should also be used to educate other people by publishing the research effort, the software planing phase as well as the final implementation of the system.

The chosen license for the source code is therefore a GNU General Public License version 2. That license is ensuring the development of free software and allows third parties to use the system, only if they acknowledge the initial author as well as license their product using the same license or a later version. This implies that if the software is used in future projects, these projects are going to be free software as well, increasing the amount of open sourced work.

The documentation of the the project is licensed using a Creative Commons - Attribution - Non Commercial - Share Alike - 4.0 International License, which is allowing everyone to use and quote the work, as long as they mark their changes, attribute the initial author, do not use the documents for commercial purpose and license the related work using the same license.

Chapter 5

Database

The server is going to use a database to store the persistent data, defined in the Software Requirement Specification. By choosing this common approach to store data, it is ensured that all information are stored consistent and persistent, even after a crash of the application.

The use of an industry standard noSQL database would ensure an efficient operation of the application. As a reliant and free database application, the open-source project *MongoDB* is chosen. This document orientated, state of the art, persistent data store will ensure an efficient a reliant storage of all user information.

5.1 Database model

Even though *MongoDB* is a database application that uses a dynamic schema, it is important to model the organisation of the data, to later ensure a reliant and consistent operation. The dynamic schema is very helpful within agile development.

5.1.1 User

The user information are stored within the *User* document store. The data is organised according to the *One-to-Many Relationships with Embedded Documents* model [Mon14, p. 141].

On the lowest level all required information about the user, like the user's email or his hashed password, are stored. Within the *PrivateInformation* array all given private information about the user are stored. These information can

only be viewed by the administrator and might include billing information. The *PublicInformation* array is holding all optional information stated by the user. These information can be accessed by every member of the club.

On top of that each user is part of one ore more divisions. The membership within the divisions is stored as nested arrays. These data structures hold the foreign key to the division as well as the date when the user joined the division.

5.1.2 Division

Each division is an entry within the division's document store. It is defined by its name and short description.

Divisions are organised hierarchical within a club, so this structure has to be represented within the database. This data structure is going to be represented within MongoDB using the *Model Tree Structures with an Array of Ancestors* [Mon14, p. 149]. This structure is storing the direct parent of the node, as well as an array of all ancestors, to easily query all ancestors. Within the use case of this application this behaviour is needed, since the system needs to subscribe the user to all chats of the user's division as well as all chats above his division.

Each division is administrated by a single user, who gains access to the administrator panel through this position. If there is no administrator specified, the super admin takes his role. This super user is defined outside of the database, to ensure access to the panel, even if the database connection is not working.

5.1.3 Messages

Each member of a division automatically joins a group chat between all members of this division. To ensure privacy for each chat, messages are only saved on the server as long as necessary. Therefore a *Message stack* document store is created. Every sent message is stored there until the recipient accesses it, or the system deletes the message after its expiration. The expiration of the message is handled by MongoDB through the functionality to set a Time-To-Life for collections [Mon14, p. 198].

Each entry contains the message to one member of the group chat. When the user syncs his application with the server, the server returns all messages

from the stack for the user. The rows are deleted as soon as the user receives the message.

5.1.4 Events

A core feature of the application is creation and management of events for each division. The events are managed within the *Event* data store. An event invites whole divisions, and every member can send a response to the invitation.

The invited divisions are stored as embedded documents according to the *One-to-Many Relationships with Embedded Documents* model [Mon14, p. 141], because it is unlikely that the user is going to add values to that field. If the amount of data added to these fields extends the reserved space for the document, the database has to reallocate the space, which leads to fragmentation and a slow write performance. Concluding the responses of the users is stored according to the *One-to-Many Relationships with Document References* model [Mon14, p. 143], since these fields are constantly extended.

The database is trying to keep the used storage low and therefore tries to delete all events that are

Besides that the event has several properties, e.g. a short description, a location, the date of the event and its last change.

5.1.5 Pictures

Since the user is able to upload pictures that are relevant to the club, a document store is created to manage these pictures. The picture's metadata is handled through that store, as well as the URL pointing to the file and an array with tags for the picture.

Each picture is uploaded by a specific user and the user can associate up to one division to the picture.

5.2 Conceptual schema: Entity Relationship Diagram

To create a durable database it is important to have a precise plan of the design. The first step –the conceptual schema of a database– can be expressed as an entity-relationship model (ER-model). Even though this schema is

originated from relational databases this project is trying to use it for a document based database, since relationships between the documents exist. The specific ER-diagram for *myVerein* can be found in appendix A on page v.

The diagram shows the standardised representation of all document stores described in the previous section.

5.3 Schema design

Since MongoDB is based on JSON-like objects (it actually stores the binary representation of serialised JSON files), the database schema in this section is modelled using JSON like syntax. This section is going to describe all relevant document stores needed for the server application.

5.3.1 User

Listing 1: *User*-document store

```
User: {
    _id: int,
    FirstName: String,
    LastName: String,
    Email: String,
    Password: String,
    Salt: String,
    PrivateInformation: {
        IBAN: String,
        ...
    },
    PublicInformation: {
        Mobile: String,
        ...
    },
    Divisions: [
        div1: {
            div_id: <division_id>,
            joined: Date
        },
```

```
21          ...
22      ]
23  }
```

5.3.2 Division

Listing 2: *Division*-document store

```
1  Division: {
2      _id: int,
3      DivisionName: String,
4      DivisionDescription: String,
5      Parent: <division_is>,
6      Ancestors: [
7          <division_id>,
8          ...
9      ]
10     Administrator: <user_id>
11 }
```

5.3.3 Messages

Listing 3: *Message*-document store

```
1  Message: {
2      _id: int,
3      Content: String,
4      Timestamp: Date,
5      Sender: <user_id>,
6      Receiver: <user_id>,
7      Group: <division_id>
8  }
```

5.3.4 Events

Listing 4: *Event*-document store

```
Event: {
    _id: int,
    Name: String,
    Description: String,
    Location: String,
    LastChange: Date,
    EventTime: Date,
    InvitedDivisions:[
        <division_id>,
        ...
    ]
}
```

Listing 5: *InvitationAnswers*-document store

```
InvitationAnswers: {
    _id: int,
    Event: <event_id>,
    User: <user_is>,
    answer: int
}
```

5.3.5 Picture

Listing 6: *Picture*-document store

```
Picture: {
    _id: int,
    Name: String,
    URL: String,
    Description: String,
    Tags: [
        String,
        ...
    ],
    Uploader: <user_id>,
    Division: <division_id>
```

```
12 }
```

Chapter 6

Initialisation

After the initial installation, the application needs to perform some initial configuration. These steps should be performed through the web interface. The following list includes all steps that need to be performed.

1. Super admin account creation
2. MongoDB database connection
3. General club settings including:
    - Name
    - Logo
4. Initial definition of club organisation
5. Creation/Import of member information

Chapter 7

Back end design

The main work of the server needs to be done within the back end. This part is managing all requests, gathering the data and handling over the appropriate information to the view. Concluding the system is going to be based on a model-view-controller design pattern.

The frameworks, that are going to be used within this project in the backend and the design principles they are based on, are going to be discussed in this chapter.

7.1 Model-view controller (MVC)

A MVC separates business logic from the data model and graphical representation (view). On top of that it enables the developer to generate an effective access control. It is part of software engineering best practices, defined as an architectural design pattern.

The server application needs to be designed according to this pattern, since it is a de-facto standard requirement for enterprise grade applications.

7.2 Aspect-oriented programming (AOP)

The development paradigm describes a way that enhances current development by increasing the modularity of applications. AOP is not replacing, but extending object-orientated programming. Since the modularity of AOP is on source-code level, it describes an engineering discipline.

An example where Aspect-oriented programming is a perfect fit, is logging. This aspect should be able to hook into every written function, to enable a useful logging mechanism. On top of that it is important to meet different requirements by enabling the developer to easily exchange the logging framework.

The server application is intended to use Aspect-oriented programming, to create a state of the art application.

7.3 Back end framework: Java Spring

To easily implement the above described techniques the Java Spring framework is going to be used within this project. This open source project is a powerful tool to develop enterprise grade applications. On top of that it tries to enforce good practice within these application.

The framework itself includes several sub-frameworks, that allow the developer to easily implement a Model-view controller or use Aspect-oriented programming among several other modules.

7.4 Back end framework: Pushy

Since push notification are an essential part of the application, the system needs to support Apple Push Notification Service (APNS). A well documented and maintained Java library, implementing this service is Pushy.

Unfortunately recent research revealed that it seems that there can be only one Server per application that implements the APNS. Concluding a work around or extension for the notification needs to be found.

Chapter 8

Front end design

In the case of the server application, the front end design describes the layout of the administrator panel within the web interface. The used framework, as well as an initial design is presented in this chapter.

8.1 Front end framework: Twitter Bootstrap

The Bootstrap framework is one of the most used projects within Github and evolved into being the standard when it comes down to effective, responsive and simple web design. Concluding this fully customisable front end framework is a perfect fit for this project.

8.2 Wireframes

Within this section an initial design for the web interface is presented.

Listings

Bibliography

[Mon14] Inc. MongoDB. *MongoDB Documentation*. Release 2.6.4. MongoDB, Inc. Sept. 2014.

Acronyms

AOP Aspect-oriented programming.

APNS Apple Push Notification Service.

MVC Model-view controller.

Appendices

Entity Relationship Diagram

myVerein

Author: Frank Steiler

Version: 1.1