BestPlaces
Software Requirements Specification
For <Subsystem or Feature>

Version <1.0>

[Note: The following template is provided for use with the Rational Unified Process. Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields in Microsoft Word (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.]

Revision History

Date	Version	Description	Author
19/10/2016	1.0	Initial fill up	Marco Kolb, Franziska Neumann

Table of Contents

1. Introduction 4
   1.1 Purpose 4
   1.2 Scope 4
   1.3 Definitions, Acronyms, and Abbreviations 4
   1.4 References 4
   1.5 Overview 4

2. Overall Description 4
   2.1 Product perspective 5
   2.2 Product Functions 5
   2.3 User characteristics 6
   2.4 Constraints 6
   2.5 Assumptions and dependencies 6

3. Specific Requirements 6
   3.1 Functionality 6
      3.1.1 User-Management 6
      3.1.2 Search 6
      3.1.3 Timeline (Tracking places) 6
      3.1.4 Favorites 6
      3.1.5 Statistics 6
      3.1.6 Interaction 6
      3.1.7 Recommendations 6
   3.2 Usability 6
      3.2.1 User-Interface 6
   3.3 Reliability 7
      3.3.1 Server status 7
      3.3.2 Web-Client 7
   3.4 Performance 7
      3.4.1 Speed 7
      3.4.2 Capacity 7
      3.4.3 Response Time 7
   3.5 Supportability 7
      3.5.1 Compatibility for server software 7
      3.5.2 Clients 7
   3.6 Design Constraints 7
      3.6.1 Application architecture 7
      3.6.2 MVC-model 7
   3.7 On-line User Documentation and Help System Requirements 7
   3.8 Purchased Components 7
   3.9 Interfaces 8
      3.9.1 User Interfaces 8
      3.9.2 Hardware Interfaces 8
      3.9.3 Software Interfaces 8
      3.9.4 Communications Interfaces 8
   3.10 Licensing Requirements 8
   3.11 Legal, Copyright, and Other Notices 8
   3.12 Applicable Standards 8

4. Supporting Information 8

Software Requirements Specification

1. Introduction

1.1 Purpose

This SRS will further define our Project and the requirements.

1.2 Scope

BestPlaces will be a service, which allows you to store information about places you have visited and you may want to visit in the future. In the first place our application is useful on mobile, but also on Pc to plan trips. Therefore, we will focus on a mobile-first and responsive website, which will support all platforms.

1.3 Definitions, Acronyms, and Abbreviations

Vaadin	GUI-Framework based on Google Web Toolkit, which enables Java to make websites.
ODOD	On-Demand-Own-Data
RESTful API	Web-Service to gather data from a server.
MVC	Model-View-Controller. Division of data, processing and view.

1.4 References

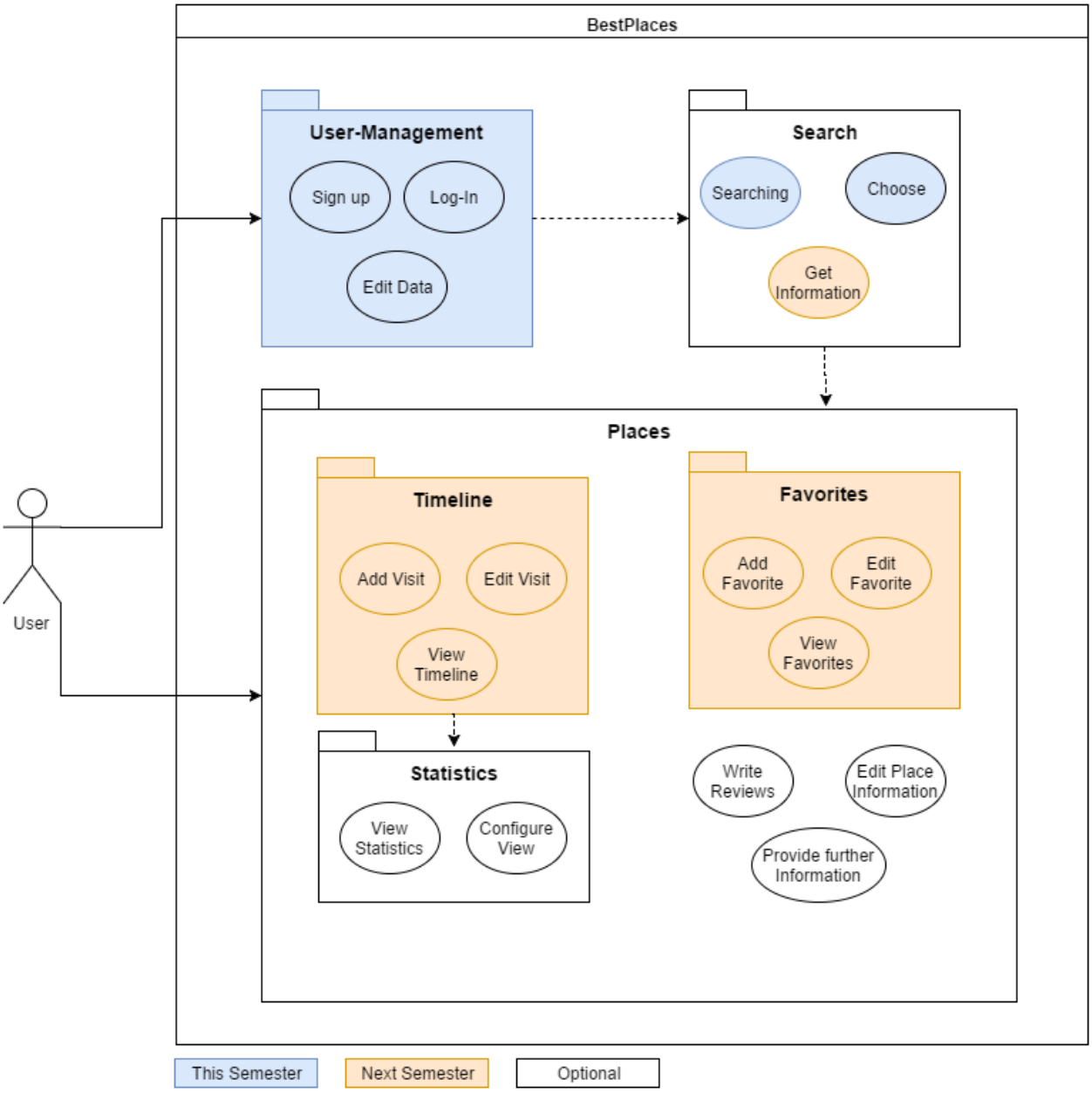
Not applicable.

1.5 Overview

BestPlaces is all about your favorite places. You can track where you were and it will tell you what you might like to visit.
The user should be able to add visited places by also providing additional information to the place. For example, if he wants to go there again or how expensive it was. By providing this information we're able to give personalized recommendations for places the user might want to visit. Additionally, this data can be used to help other users to get information about the place.

2. Overall Description

On the following page we provided our Overall User Case Diagram, which briefly depicts the features of out application and when there will be implemented.



2.1 Product perspective

BestPlaces will make it easier to track your favorite places. Our target is defined by total comfort for the user. Every function and any use should be as easy as it could. The user should have fun to use the application instead of get bored or angry about it. So our claim is it to build an application which is easily to use, appealing in style, multisided in use and never gets disliked by the user.

2.2 Product Functions

The user should be able to track his favorite places as well as those, where he might want to go. A ‘Place’ includes information like GPS coordinates, costs, opening hours, own experience, the possibility to perceive which food the user wants to eat in the future and of course the option to get opinions about this place from other users. Furthermore, the user has the option to chart his places by for example the number of visits, the money he spent, the food he ate and so on.

2.3 User characteristics

Our application is suitable for everyone who likes to gather information about the own lifestyle. Due to the analysis of the tracked data one can find out more about the behavior of oneself.

2.4 Constraints

Our project is only by time, effort and missing work equipment restricted. There are many good ideas for our application but in the given time it is not possible to implement all of them.

2.5 Assumptions and dependencies

- IDE: IntelliJ and PyCharm
- Version control: GitHub
- Scrum: JIRA
- Programming language: Python, Java
- Database: MySQL
- Test: Junit,…

3. Specific Requirements

3.1 Functionality

3.1.1 User-Management

The user-management includes everything concerning the management of the users. These should be able to sign-up and log-in to the service and their data should be stored in the database. Therefore, it should be possible for the user to have exact data on all devices.

3.1.2 Search

To make it easier for the user to find and track places without providing all information by themselves, there should be a small search engine for these places. By using existing sources like Google, we prevent high costs for databases.

3.1.3 Timeline (Tracking places)

The “Timeline” displays all places, the user visited and the user may plans to visit (with fixed date) in the future. It will be displayed after logging in. To track places, the user should have the ability to add and edit his places.

3.1.4 Favorites

The user should be able to save their favorite places and places they may want to visit in the future, but without defining any date.

3.1.5 Statistics

The application should present statistics about the places the user visited. This include costs, how often the place was visited and many more.

3.1.6 Interaction

To enable users to add their own places and provide additional information, we use our “ODOD”-Model (“On-Demand-Own-Data”), which means that if the user adds places or information about it, we will make a data record in our databases for that place which will further be used for that place.

3.1.7 Recommendations

Based on user data we may be able to give recommendations on places, the User might be interested in.

3.2 Usability

3.2.1 User-Interface

The application will be easy to use, due to a clean and structured user-interface. To provide full usability on most possible devices, the web-client should be responsive.

3.3 Reliability

3.3.1 Server status

The server should always be running and online to provide full functionality. Therefore, it should be developed carefully and without minor bugs. Even if there is a bug, the server should be able to stay running and ignoring the bug.

3.3.2 Web-Client

The web-client should always be running, because it's the main interface of the application. This will be accomplished by using exception-handling.

3.4 Performance

3.4.1 Speed

Because of the server-client architecture we are able to provide high speed on client devices. The speed of our webclient is depending on the amount of users and general server load.

3.4.2 Capacity

Our server is running on a hosted Linux-Server located in Frankfurt. Because our financial resources are limited this server has a rather low speed in data processing and not that much disk space. Therefore, we aren't able to handle thousands of people using our applications. Nevertheless, we could expand rather quickly by using a bigger Linux-server.

3.4.3 Response Time

Our Server is located in Frankfurt in a big data center with high-speed internet connection. Therefore, we should be able to provide quick response times concerning internet connection. Depending on the amount of users the response times could be worse, because it's not a high-performance server.

3.5 Supportability

3.5.1 Compatibility for server software

To develop out server, we will use python and MySQL, which nearly runs on every system. Therefore, we're able to easily switch between systems.
The server has to handle different database types and switching between database management systems should be as easy as possible.

3.5.2 Clients

Our web-client is based on Vaadin, which uses JavaScript to display the website. Therefore, the web-client should run in every modern browser, which supports JavaScript. To provide full usability we will may implement a native Android-app.

3.6 Design Constraints

3.6.1 Application architecture

Our application will be based on a server-client-structure. The Server will be responsible for data warehouse and data processing. To easily communicate with clients, we will provide a RESTful API, which enables interaction with the server on nearly every client.

3.6.2 MVC-model

Our application should strictly follow the MVC-model to lower the effort for implementing new clients and to write clean and understandable code.

3.7 On-line User Documentation and Help System Requirements

Not applicable.

3.8 Purchased Components

Not applicable.

3.9 Interfaces

3.9.1 User Interfaces

On the web the Interface is written in Java using the Vaadin GUI-framework. As described before the main view should be a timeline, which presents the user places he visited and is planning to visit. Moreover, there should be a search bar on the top, which enables the user to search places.

3.9.2 Hardware Interfaces

No defined hardware.

3.9.3 Software Interfaces

To get as much information as possible about places, we will communicate with the Google Place API, which provides all information on places from Google. Additionally, we will reuse other API's, like TripAdvisor to provide even more information.

3.9.4 Communications Interfaces

We will use a server-client-model, which divides data warehouse and data processing from the client, which the user will interact with. This makes it possible to support all kind of devices, because we do not need to implement our logic on every device. Moreover, it speeds up the client application.
The server, which the client has to interact with, will be based on Python which provides web services by a RESTful API. Information about the interfaces will follow.

3.10 Licensing Requirements

Not applicable.

3.11 Legal, Copyright, and Other Notices

Not applicable

3.12 Applicable Standards

Not applicable.

4. Supporting Information