Strawhacks

Rock Chalk Rendezvous

Software Requirements Specifications

Version 1.1

Revision History

Date	Version	Description	Author
03/04/2024	1.0	Preliminary document with assigning roles	Shayna Weinstein, Ben Phillips, Dylan Kneidel, Delroy Wright, Andrew Reyes
3/24/2024	1.1	Preliminary draft including desired functionality	Shayna Weinstein, Ben Phillips, Dylan Kneidel, Delroy Wright, Andrew Reyes

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The SRS fully describes the external behavior of the application or subsystem identified. It also describes nonfunctional requirements, design constraints, and other factors necessary to provide a complete and comprehensive description of the requirements for the software.

1.2 Scope

This application aims to implement the essential functional requirements described in section 3 of this document following the schedule described in section 4.2.4 of *Project Plan*. The end goal is a product in line with what is described in *Project Vision*. General use cases for this are described in section 2.2 of this document.

1.3 Definitions, Acronyms, and Abbreviations

See *Glossary of Terms* Document (gloss1) for relevant definitions, acronyms, and abbreviations.

1.4 References

[1] Project Vision, version 1.0 from 2/19/2024. Team Strawhacks, accessible via team folder.

[2] Glossary of Terms, version 0.1 from 2/19/2024. Team Strawhacks, accessible via team folder.

[3] Project Plan, version 1.1 from 3/15/2024. Team Strawhacks, accessible via team folder.

1.5 Overview

Overall Description — Includes information about all the functional and non-functional requirements for a given piece of software. The **SRS** serves as the main point of reference for the software development team who'll build the software product, as well as for all other involved stakeholders.

Specific Requirements — This section of the **SRS** contains all software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements

Classification of Functional Requirements — Contains an extensive list of the functional requirements pertaining to this project in addition to their types (Essential, Desirable, Optional)

Appendices — Includes an organized list of all relevant documents and materials pertaining to the reader of the **Software Development Plan** as well as any project technical standards and plans which apply to this project.

1.6 Product perspective

1.6.1 System Interfaces

- File storage and access

1.6.2 User Interfaces

- User authorization
- Calendar functions

1.6.3 Hardware Interfaces

- Graphics library for user interface
- Keyboard and mouse input

1.6.4 Software Interfaces

- C++
- HTML

1.6.5 Communication Interfaces

- Internet communication between client and server

1.6.6 Memory Constraints

- 100 MB maximum for client
- 800 MB maximum for server (max for small computers like Raspberry PI)

1.7 Product Functions

- User authentication
- Per-user calendar creation
- Sharing option
- Calendar comparison

1.8 User characteristics

- Availability time frame (ex: 9AM-5PM)
- Calendar availability (schedule-dependent, excludes availability time frame)
- Existing groups

1.9 Constraints

- Software must be completed before the specified end date.
- Software must implement all Essential requirements.
- Software should meet the scope defined in the Project Plan Document.
- Software must be accessible through an online interface or GitHub.

1.10 Assumptions and dependencies

Assumptions:

- **User Engagement** - It is assumed that users will actively engage with the application, regularly updating their availability to ensure the calendar remains accurate and useful.
- **Internet Connectivity** - The application assumes that users have consistent access to the internet to enable real-time updates and synchronization across ussr calendars.
- **Compatibility** - The applications assumes user's devices are compatible with the software requirements outlined in section 2.1.4 (Software Interfaces)
- **Data Accuracy** - The software assumes that data inputted by users, such as availability times and dates, is accurate.
- **Software Environment Stability** - Assumes that the programming environment and associated libraries used for development are stable and will not undergo significant changes, impacting software functionality.

Dependencies:

- **Third-Party Services** - The software's performance and functionality depend on third-party services (ical format, google calendar, outlook calendar), and internet service providers.
- **Browser Compatibility** - The web-based version of the application depends on compatibility with major web browsers.
- **Server Infrastructure** - The software depends on the reliability and availability of the server infrastructure to handle requests, store data, and manage synchronization efficiently.
- **Web Service Dependencies**- The software depends on an ability to manage and

handle web communications, data processing, and interfacing with external services using C++.

1.11 Requirements subsets

2. Specific Requirements

2.1 Functionality

2.1.1 User Authentication

User data must be identifiable so that clients can access and update their data using a set of login information consisting of a password and either a username or email address. User data must not be accessible to anyone who does not provide the appropriate authentication information, and an authenticated user must only have access to the user data that belongs to them.

2.1.2 Sharing With Other Users (Link, Code, etc)

Group calendars must be identifiable by a token that can be shared between users in order to give access to it.

2.1.3 Creating a personal calendar

Users may create a calendar that reflects their personal availability schedule for use in all group calendars that they are a member of.

2.1.4 Comparing individual calendar data

Creates a group calendar with all individual users' calendar data merged together in order to give a detailed overview of group member availability over time.

2.1.5 Adding others to a group calendar

The system must be able to dynamically change the roster of users that are a member of each group calendar. Users must be able to join and leave existing groups.

2.1.6 Setting limits on meeting time

Users have the option to have set intervals for meeting times (e.g. 30 mins, 1 hour, etc.).

2.1.7 Graphical user interface

Containing user input or output with a visual interface.

2.1.8 Multiple time zone support

All times and dates will be displayed according to the user's current time zone.

2.1.9 Meeting notification reminder

At a set time before the meeting the user will receive a reminder notification.

2.1.10 Linking personal calendar using iCal file

When setting up a personal calendar, users can link an iCal file of their schedule to speed up the schedule creation process.

2.1.11 Different location and meeting methods

Users can specify how and where they are able to meet at a given time. For instance, a user may be able to meet online but not in person, or only meet in person in a particular place.

2.1.12 Automatic time meeting suggestion (Function of meeting length/ all users)

The application will look at all schedules of participating users in a group calendar and how long the desired meeting is and will automatically find spots without overlap and present them as meeting time options to the user.

2.1.13 Anonymous users

Allow users to access core functionality of the application without requiring them to have a password-protected account.

2.2 Use-Case Specifications

See the use-case diagram ‘use-cases.pdf’ in the project documentation folder.

2.3 Supplementary Requirements

2.3.1 Responsive networking under 2s

Basic client requests must receive a response from the server within 2 seconds.

2.3.2 Must be written in C++

The project’s final binary executable artifacts must be created by compilation of source code using the g++, clang, or other equivalent C++ compiler application.

2.3.3 Easily-shareable identifiers

Clients must be able to easily share group identifier tokens. They should have a readable form such as a short string of familiar characters.

2.3.4 High server uptime

Server must be available for the main demographic of users at most hours, only going offline for maintenance.

2.3.5 Concurrent access

Allows many users to use the application and communicate with the server at the same time. The server must be able to handle an arbitrarily large number of clients concurrently.

2.3.6 Not vulnerable to security exploits

Robust security system to protect user data. Implementation must avoid vulnerabilities by any means necessary.

2.3.7 Intuitive user interface

Makes it easy for users to understand and use the application without needing complicated user instructions. Simplifies user interactions by displaying calendars and controls in a logical, user-friendly manner, minimizing the need for extensive guidance.

2.3.8 Cross-platform support

The client-side application must function on multiple different operating systems including both Windows, MacOS, and basic Linux platforms. The major elements of the user interface must be consistent between platforms.

3. Classification of Functional Requirements

Functionality	Type
User authentication	Essential
Sharing with other users (Link, Code, etc)	Essential
Creating a personal calendar	Essential
Comparing individual calendar data	Essential
Adding other users to a group calendar	Essential
Setting meeting time limits	Desirable
Graphical user interface	Desirable
Multiple time zone support	Optional
Meeting notification reminder	Optional
Linking personal calendar using iCal file	Optional
Different locations and meeting methods	Optional
Automatic meeting time suggestion (Function of meeting length / all users)	Optional
Anonymous users	Optional

4. Appendices

The project will follow the UPEDU process.

Other applicable process plans are listed in the references section, including Programming Guidelines.