**Strawhacks**

Rock Chalk Rendezvous

# Software Architecture Document

**Version 1.0**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 04/04/2024 | 1.0 | | Shayna Weinstein, Ben Phillips, Delroy Wright, Dylan Kneidel, Andrew Reyes |
| | | | |
| | | | |

# Table of Contents

# 1. Introduction

## 1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

## 1.2 Scope

The Software Architecture Document outlines the high-level structure of a software system, influencing developers, project managers, quality assurance teams, system analysts, operations teams, maintenance teams, and stakeholders. It serves as a guide for design, development, testing, deployment, and maintenance, ensuring alignment with project goals, scalability, and system integrity throughout the software lifecycle.

## 1.3 Definitions, Acronyms, and Abbreviations

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

## 1.4 References

| Title | Date | Publishing Organization | Source |
|---|---|---|---|
| Architectural Diagrams | 4/11/24 | Strawhacks | RockChalkRendezvous Documents/Diagrams |
| Glossary of Terms | 4/11/24 | Strawhacks | RockChalkRendezvous Documents (gloss1) |

## 1.5 Overview

Overall Description — Includes information about all the functional and non-functional requirements for a given piece of software. The Software Architecture Document serves to capture and convey the significant architectural decisions which have been made on the system.

Architectural Representation — Describes what software architecture is for the current system, and how it is represented. It enumerates the views that are necessary, and for each view, explains what types of model elements it contains

Architectural Goals and Constraints — Describes the software requirements and objectives that have some significant impact on the architecture

Logical View — This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages.

Interface Description — A description of the major entity interfaces, including screen formats, valid inputs, and resulting outputs.

Quality — A description of how the software architecture contributes to all capabilities (other than functionality) of the system

# 2. Architectural Goals and Constraints

This software is built to follow a request-based, client-server architecture for an internet application. Most functions, or processes as defined in the Architectural Diagrams are represented internally as specific requests made by clients to the server.

Because calendars are shared between users, we must provide a safe and accessible method of storing user data. Thus, all shared data like Groups, Calendars, and User Data remains on the server, to be requested by the user.

Because data exists on the server, it must be requested by the user. Therefore, we must define behavior for, process, validate, and allow clients to create these requests. This has defined our software architecture, defining functions as operations on data stored by the server, and these functions being accessed through HTTP requests to our server, originating from a client.

The HTTP protocol being used for internet communication requires arbitrarily-formatted data to be expressed as plain text. Therefore, the data contained in any data structure that is intended to be communicated between the client and the server must be able to be translated (encoded) both to and from plain text that uses only printable characters. Thus, all of said data structures have functions or methods that can encode their state to a string object and can attempt to decode their state from a stream object.

# 3. Architectural Representation

## 3.1 Functional View

The functional view, or the control flow of the program as a whole, is primarily divided between the client and server applications in order to allow centralized access and data security. The client application consists of the user interface, structuring requests to send to the server, buffering response data from the server, and performing the actual time analysis and calendar synthesis on the data received.

The server application is focused on storing user data, responding to client requests, and determining whether or not the requesting client is allowed to access the requested data. User data is stored as local files that are accessed per-request, and group data is stored in memory until the server is stopped, when it is saved in its own file.

Requests and responses are sent over the internet using HTTP Post requests. The server implements a REST API so that it does not need to track the states of connections, so the client must send all necessary authentication and contextual information with each request.

## 3.2 Structural View

The structural view is the organization and hierarchy of the data structures that are used by both the client and server. At the highest level this includes groups, users, and calendars, and at the lowest level entails enum types, type definitions, and TimeAndDate, the most basic representation of an instant in time. The user structure is mostly relevant to the server for storage purposes because it contains information that should only ever be available to the corresponding user, like their password and complete group membership list.

All of these structures possess encoding and decoding functions in order to translate them to plain text and back. The plain text representation is necessary for transmitting data structures over the internet via HTTP requests and storing the data in files on the server side. The encoding and decoding functions have a standardized signature so that they can simply call each other pseudo-recursively in order to translate an entire tree of diverse data types in a modular, concise, and readable fashion. In other words, they use the single responsibility model where each structure only needs to know its own encoding scheme.

# 4. Logical View

## 4.1 Overview

Architecturally significant parts of the design model include the Client and Server. These contain code for connecting our Calendar data to the server, and allowing the client to interact with said data. Other architecturally significant parts include the Calendar struct, its used attributes (TimeAndDates, RepeatTypes), the Group struct, and the User struct. Each major data structure exists in its own package and header file alongside minor related structures and the functionality that is relevant to those structures.

## 4.2 Architecturally Significant Design Modules or Packages

This section of the software architecture document focuses on Architecturally Significant Design Modules or Packages. Each shared package is outlined in section 4.2.1 with a subsection containing its name and a brief description. Similarly, section 4.2.2 references client packages and section 4.2.3 references server packages. In section 4.3, illustrative diagrams showcase all significant classes and nested packages within, in the form of functional and structural diagrams.

### *4.2.1 Shared Packages*

**Core Utilities** - core_utils.hpp
- Basic utility type definitions, structs, and functions used throughout the codebase
- Encoding and decoding functions for the necessary standard library types

**Time and Date** - timeanddate.hpp
- TimeAndDate structure
- All functionality related to translating between different representations of time

**Calendar** - calendar.hpp
- Calendar and TimeBlock structures, and RepeatType enum
- All functionality related to TimeBlocks and their role in Calendars

**Group** - group.hpp
- Group structure
- GroupID type definition
- Encoding and decoding functions for Group and GroupID types

**User** - user.hpp
- User structure
- User encoding and decoding methods
- Functionality for checking the validity of a potential username

**Networking** - networking.hpp
- Enumerates the sets of valid client requests and server responses

- String formatting of server response codes

#### *4.2.2 Client Packages*

**Client Main** - client/main.cpp

- User interface
- Formatting data for submitting requests to the server
- Interpreting server responses

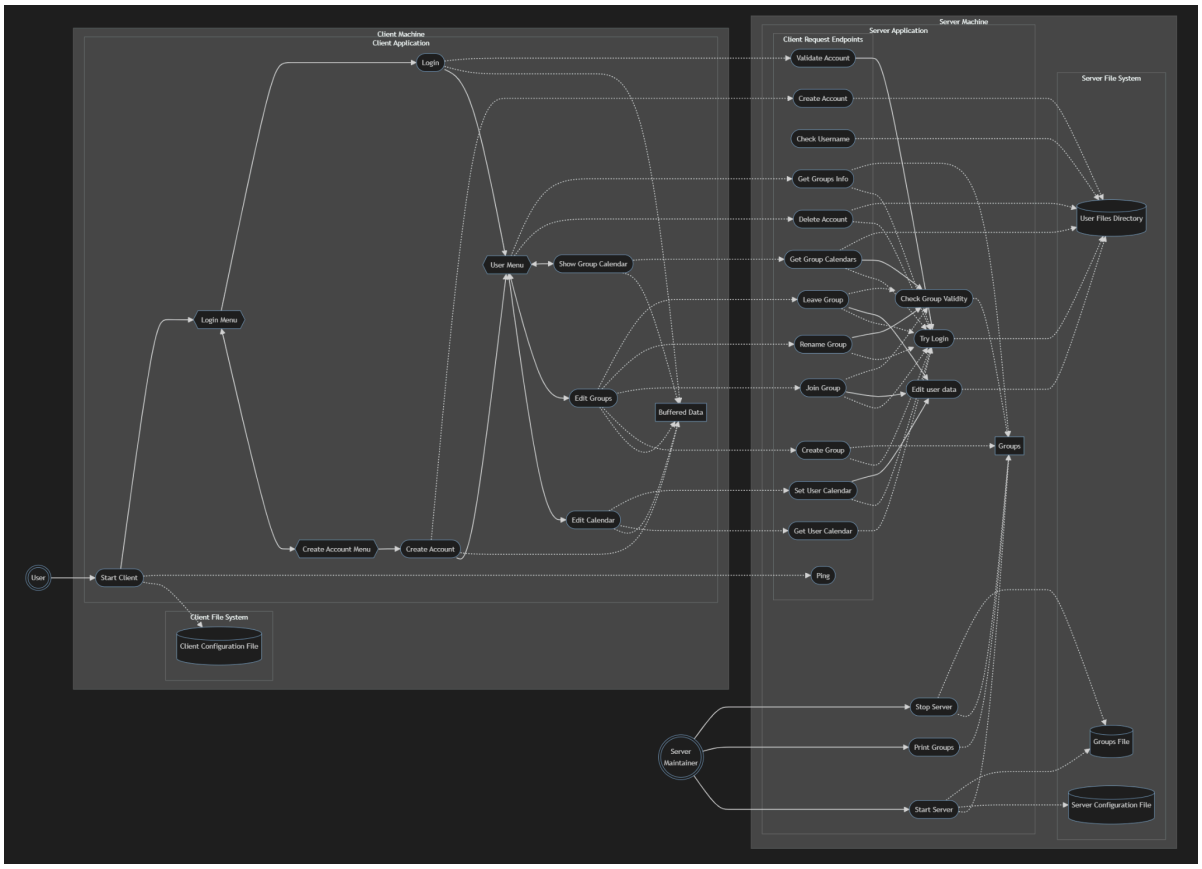
#### *4.2.3 Server Packages*

**Server Main** - server/main.cpp

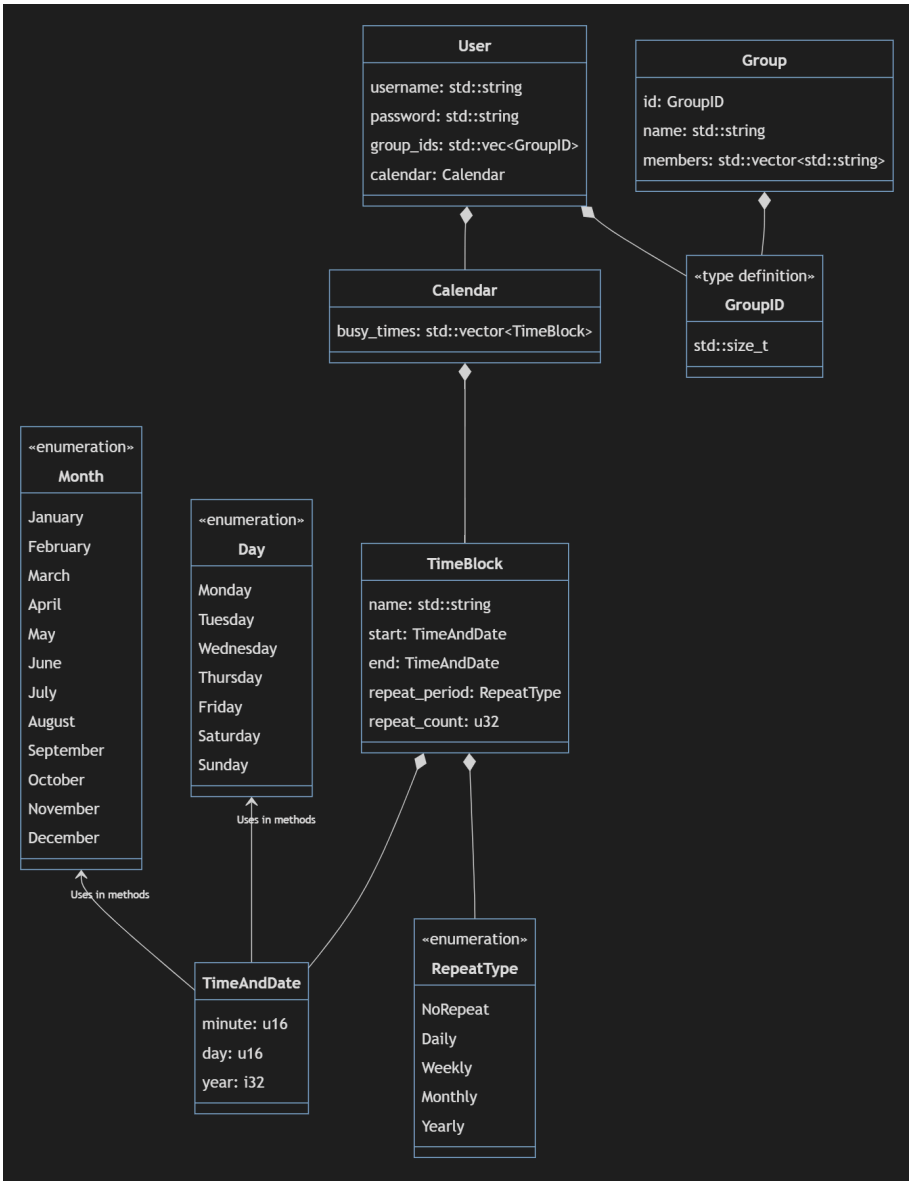
- HTTP endpoint functions
- All functionality that interacts with files on the server machine
- User authentication
- Maintaining groups list
- Resolving references between users and groups

## 4.3 Architectural Diagrams

### *4.3.1 Functional Diagram*


Client Machine
Client Application
Login
Login Menu
User Menu
Show Group Calendar
Edit Groups
Buffered Data
Edit Calendar
Create Account Menu
Create Account
User
Start Client
Client File System
Client Configuration File
Server Machine
Server Application
Client Request Endpoints
Validate Account
Create Account
Check Username
Get Groups Info
Delete Account
Get Group Calendars
Leave Group
Rename Group
Join Group
Create Group
Set User Calendar
Get User Calendar
Ping
Check Group Validity
Try Login
Edit user data
Groups
Server File System
User Files Directory
Server Maintainer
Stop Server
Print Groups
Start Server
Groups File
Server Configuration File

### *4.3.2 Structural Diagram (Class Diagram)*



# 5. Interface Description

## 5.1 Client Desktop Application Interface:

### *5.1.1 Screen Formats*

The client desktop application will feature a user-friendly TUI terminal interface with commands for scheduling meetings, viewing group members' schedules, and managing meeting invitations.

### *5.1.2 Valid Inputs*

Users will be able to input meeting details such as date, time, location, and participants' availability. Additionally, users can select group members to invite to meetings and specify meeting preferences.

### *5.1.3 Resulting Outputs*

Upon inputting meeting details and preferences, the client application will display available meeting times based on the composite schedule of group members. Users will receive notifications for meeting invitations and updates on meeting statuses.

## 5.2 Backend Server Application Interface

### *5.2.1 Screen Formats*

The backend server application does not have a graphical user interface accessible to end-users.

### *5.2.2 Valid Inputs*

The server application will receive data from client desktop applications, including meeting details, group member schedules, and user preferences.

### *5.2.3 Resulting Outputs*

Based on the received data, the server application will process and analyze schedule information to generate composite schedules for group meetings. It will then send responses to client applications, indicating available meeting times and handling meeting scheduling logic.

# 6. Quality

The RESTful nature of the architecture adds both reliability and portability to this app. Because the app uses HTTP as its communication, the client can be designed in almost any way as long as it facilitates the interfacing between user and HTTP request and response. Moreover, the stateless nature of the application will make implementation simpler and more reliable. As C++ has been used for the language of choice, the reliability of a strongly typed compiler has facilitated a robust and reliable design around how data is represented in this app, and thus improves the reliability and extensibility of the app.