4DV608

Project Development

- *Statistics Calendar*

Author: Sarpreet Singh Buttar
Email: sb223ce@student.lnu.se

1. Introduction

The purpose of this report is to highlight the design decisions of Statistics Calendar system, that aim to allow a user to see how s/he spent time on his/her Google calendar. Following are the requirements of Statistics Calendar system:

1. The system should allow the user to log in
2. The user should be able to select one of his/her Google calendar
3. The user should be able to add keyword to a calendar event
4. The user should be able to add budget to a calendar event
5. The user should be able to see the following statistics:
    a. Percentage of time spent by him/her in calendar events for a particular time period
    b. Used and remaining budget for each calendar event
6. The system should expose API of its requirements

2. Design Outline

In this section, we present the overview of the design. We begin with the system architecture. Then, we describe the technologies used to build the system. In addition, we also explain the alternatives that we considered during the design phase.

2.1 System Architecture

We aim to split the Statistics Calendar system in to two standalone parts: client side (user interface) and server side (API/logic). This separation enables us to build an independent REST API that can work with multiplatform clients such as mobile, desktop and web browser. To achieve this separation, we considered two architectures named as Model-View-Controller (MVC) and Client-Server. MVC is not selected because it restricts the API to work with one specific user interface, i.e., client. On the other side, Client-Server architecture is selected because it allows one or more clients to interact with API via HTTP, see figure 1.

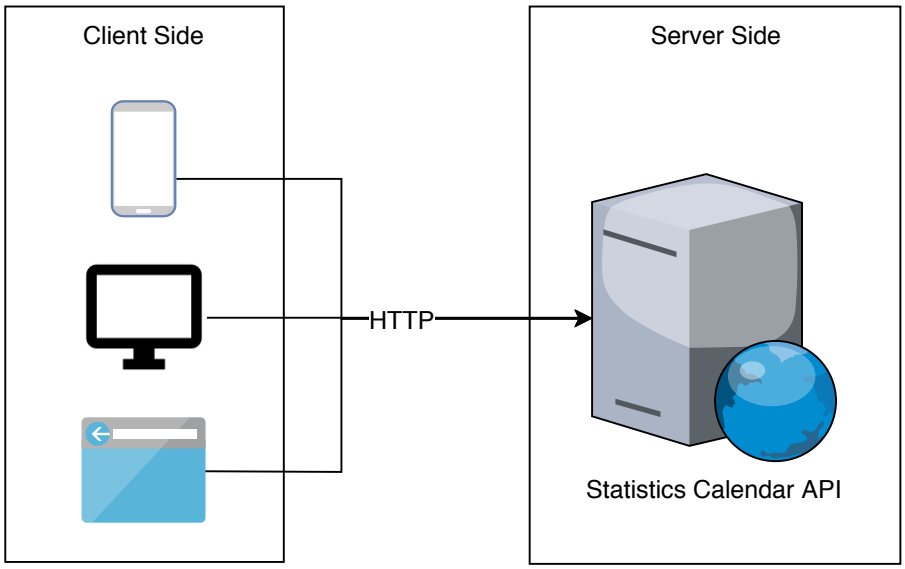

Figure 1: Architecture of Statistics Calendar System

2.2 Technologies

On server side, Java, NodeJS, and Python are very famous programming languages. One of the requirements of Statistics Calendar API is to show two different types of statistics. Therefore, to select the programming language we priorities this requirement as remaining requirements can be easily implemented in any of these languages. We found that Java has JavaFX library to create statistics images. However, we reject Java because JavaFX is very complex compare to the libraries provided by NodeJS and Python. NodeJS provides Plotly library which is very lightweight and easy to use. However, it has one downside. It requires client to add some dependencies to show the images. Therefore, we also reject it as we aim to create a standalone API. On the other hand, Python is widely used for statistics. Mataplotlib is Python's most used library for creating various types of graphs, charts, etc. It is a standalone library and does not demand any dependency on the client side. As a result, we selected Python language with Mataplotlib library. Python provides Django and Flask frameworks for implementing API. We used Flask because it is light weight, easy to use, and faster compare to Django.

On client side, we have options to build a desktop app, web app, or mobile app because our API is able to support all these clients. However, due to ease of development we selected web app. Moreover, if needed web app can be easily converted into hybrid apps that can run on mobile as well as desktop. ReactJS and Angular are the most used frameworks for building web apps. Angular is a heavy enterprise framework compare to ReactJS. Whereas, ReactJS is light weight and faster than Angular. Moreover, it is widely used in small scale apps. Therefore, we used ReactJS for creating our web app.

3. Design Details

In this section, we describe the design details. We start with software architecture. Then, we show the use cases that we derived from the given requirements. Furthermore, we present the components of the system. Lastly, we show the deployment view of the system.

3.1 Software Architecture

As we mentioned earlier, that the system uses Client-Server architecture. Now, we present the architecture of server-side. We aim to have low coupling, high cohesion, and good reusability so that the system can be easily scalable. To achieve these properties, we use a combination of Service-Oriented architecture (SOA) and Multilayer architecture. SOA is used to communicate with remote services, in our case Google Calendar API. Multilayer architecture is used to communicate with library, i.e., Mataplotlib, added as dependency in server -side.

3.2 Use Cases

From given requirements, we developed 7 use cases, see figure 2:

1. *Log In*: enables the client (web app) to allow its user to log in with his/her Google account.

2. *Log Out*: allows the client to delete its user's token from Statistics Calendar API.

3. *Get Calendars:* allows the client to fetch user's calendars.

4. *Get Calendar Events*: allows the client to fetch events of a specific calendar.

5. *Add Keyword/Budget:* allows the client to add keyword/budget on an event.

6. *Get User Info*: allows the client to fetch extra user information such as email, name, profile link, etc. This is an extra use case and can be helpful to display the name of logged in user.

7. *Get Statistics:* allows the client to get statistics of given calendar.

Use case 1, 3, 4, 5 and 6 are dependent on external service, i.e., Google Calendar API. Whereas use case 7 is dependent on Matplotlib library.

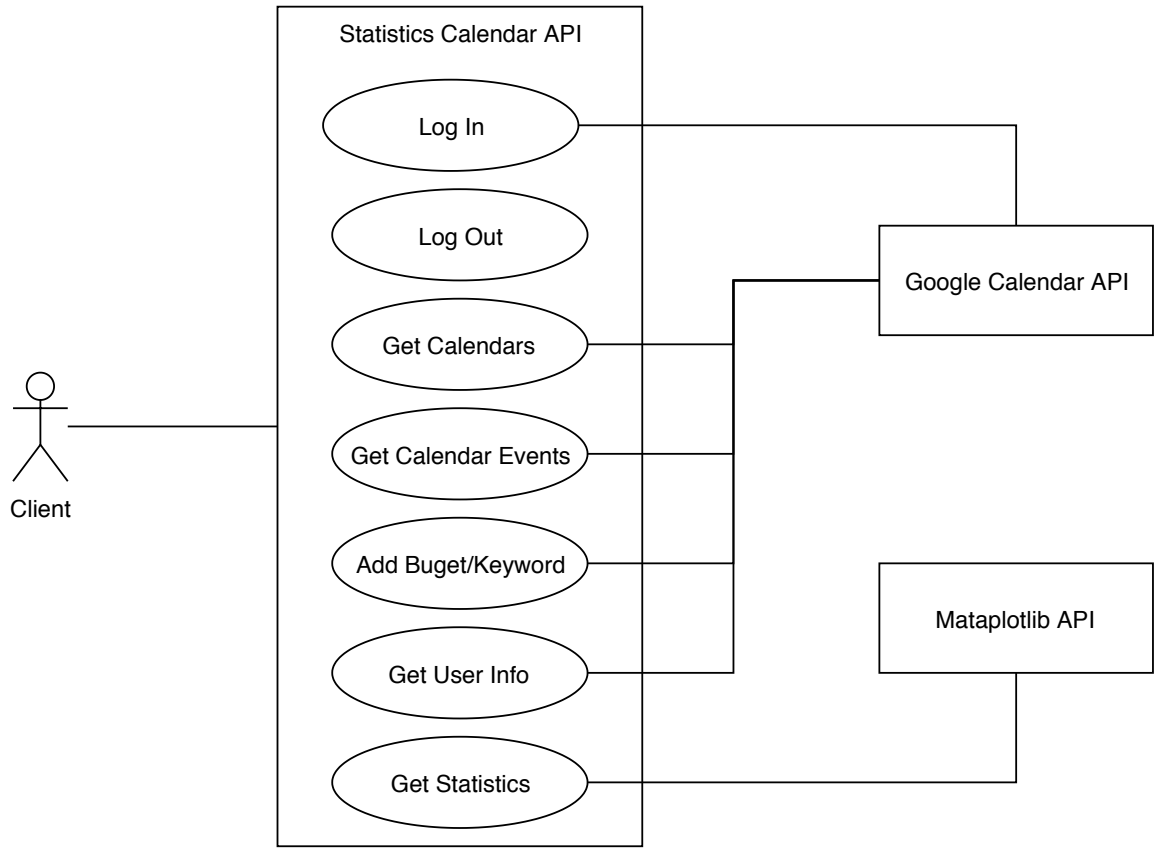

Figure 2: Use Cases of Statistics Calendar System

3.3 Components

We developed two components named as Web App and Statistics Calendar, see figure 3.

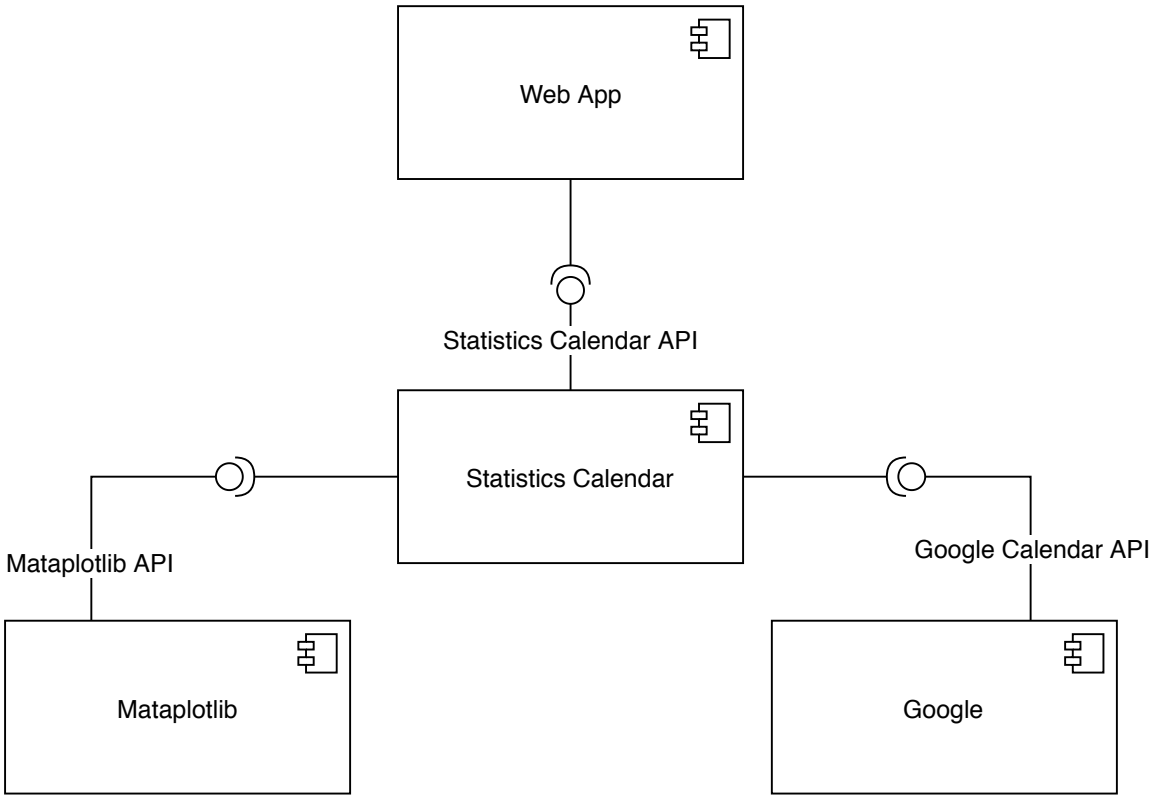

Figure 3: Components of Statistics Calendar System

Statistics Calendar (server-side) uses Google Calendar API to access the user's calendar related data. Whereas, Matplotlib API is to generate statistical images. With the help of these two components, Statistics Calendar provides API that is used by Web App (client-side). We do not use any Database because Google Calendar API allows us to save data on their cloud. Moreover, if we use, it may create synchronization problem because the database must be up to date with user's Google calendar.

3.4 Deployment

Figure 4 shows the deployment view of Statistics Calendar System.

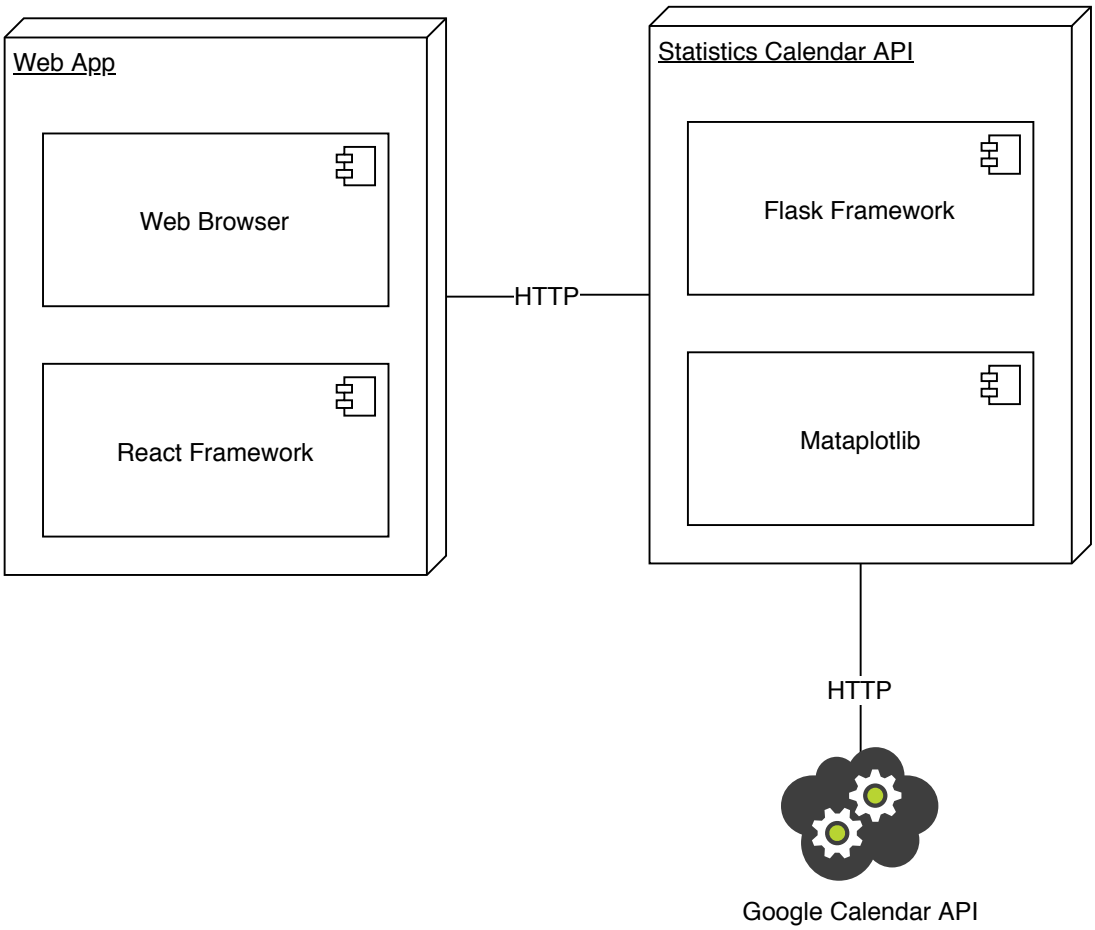

Figure 4: Deployment View of Statistics Calendar System

Web App runs on Web Browser and uses React framework. It communicates with Statistics Calendar API via HTTP protocol. Statistics Calendar API uses Flask framework to enable REST operation. In addition, it uses Matplotlib to generate statistical images. Moreover, it interacts with external service, i.e., Google Calendar API via HTTP to access user's Google calendar data.