CSC 431

Tempo

System Architecture Specification (SAS)

Team 01

Alexandr Kim	Scrum Master
Enzo Carvalho	Requirements Engineer
Salvatore Puma	System Architect

Version History

Version	Date	Author(s)	Change Comments
1	3/25/2024	Alexander Kim, Enzo Carvalho, Salvatore Puma	
2	4/4/2024	Alexander Kim, Enzo Carvalho, Salvatore Puma	Revised structural diagram

Table of Contents

Table of Figures

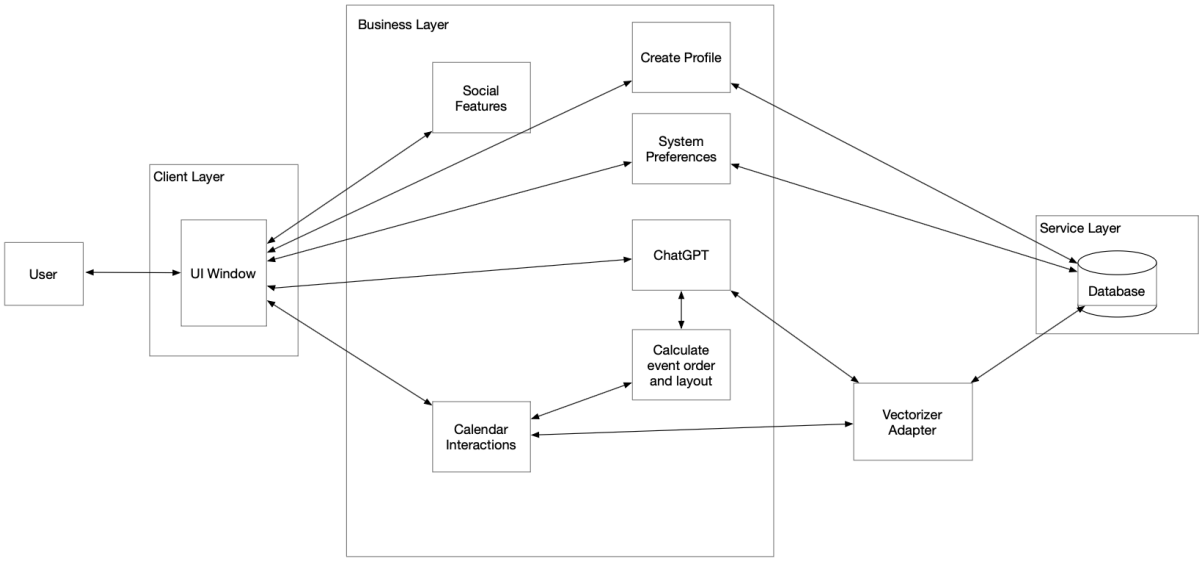
41. System Analysis

41.1 System Overview

This is the architecture specification of Tempo. Tempo uses a three-tier architecture, composed of a client layer, business layer and service layer. The client is the mobile application installed in every user's mobile device (Android or iOS), where very little calculations or storage are held, the user interface is provided, and accesses the business layer directly, service layer indirectly, through networking protocols such as DNS. The business layer does almost all of the heavy calculations, mainly the event order and layout calculations, and also provides the logic for the social features, creating a profile, system preferences, calendar interactions, and the chat access point that connects to chatGPT through the OpenAI API. This layer will be in the AWS cloud. The service layer is almost strictly used for the AWS cloud database (DynamoDB or the like).

The client layer's UI connects directly to all of the services and logic provided by the business layer, allowing the user access to features like the calendar, sharing/social, creating and manipulating your profile, and the AI chat. Then, some of the services in the business layer that control these features, specifically create profile and system preferences, connect directly to the service layer database. Others, like the ChatGPT and calendar interactions layer connect to the database through the vectorizer adapter, which uses machine learning algorithms to convert the inputs from these features into vectors in order to calculate a response using OpenAI's API, and stores everything in the database.

41.2 System Diagram



41.3 Actor Identification

The actors in the system are:

- New user: First time users of the app that have not yet made an account.
- Existing user: A returning user of the app that already has an existing account.
- Platform server: AWS will act as the platform server in all of the services the system may possibly require from it, such as authentication, cloud database, and elastic computing.

41.4 Design Rationale

1.4.1 Architectural Style

The app uses a three-tier structure:

- Client layer: Mainly responsible for the user interface, only supports small storage for offline features.
- Business layer: Responsible for all of the business logic, mainly the calculations for “translating” the inputs from the AI chat to the calendar and vice versa.
- Service layer: Mainly responsible for the cloud database and all of the other cloud services that could be used.

1.4.2 Design Pattern(s)

As of this stage, the only design pattern that makes it into the system is the adapter. It would be used as a vectorizer adapter, that will translate between the database on one side and both the calendar and ChatGPT (AI chat) on the other.

1.4.3 Framework

We decided to use the Flutter framework because it allows us to deploy our app on both Android and iOS from one single code base. On top of that, it is well documented, and easy to integrate with cloud services. Flutter will be used mainly for the front-end and a good amount of the back-end, with some JavaScript libraries making up the remainder of the back-end. Specifically, in the business layer, the JavaScript libraries will be used to make up the calendar view of the app, while Flutter will take care of all the other logic. We will also use AWS the database and cloud services like authentication and security.

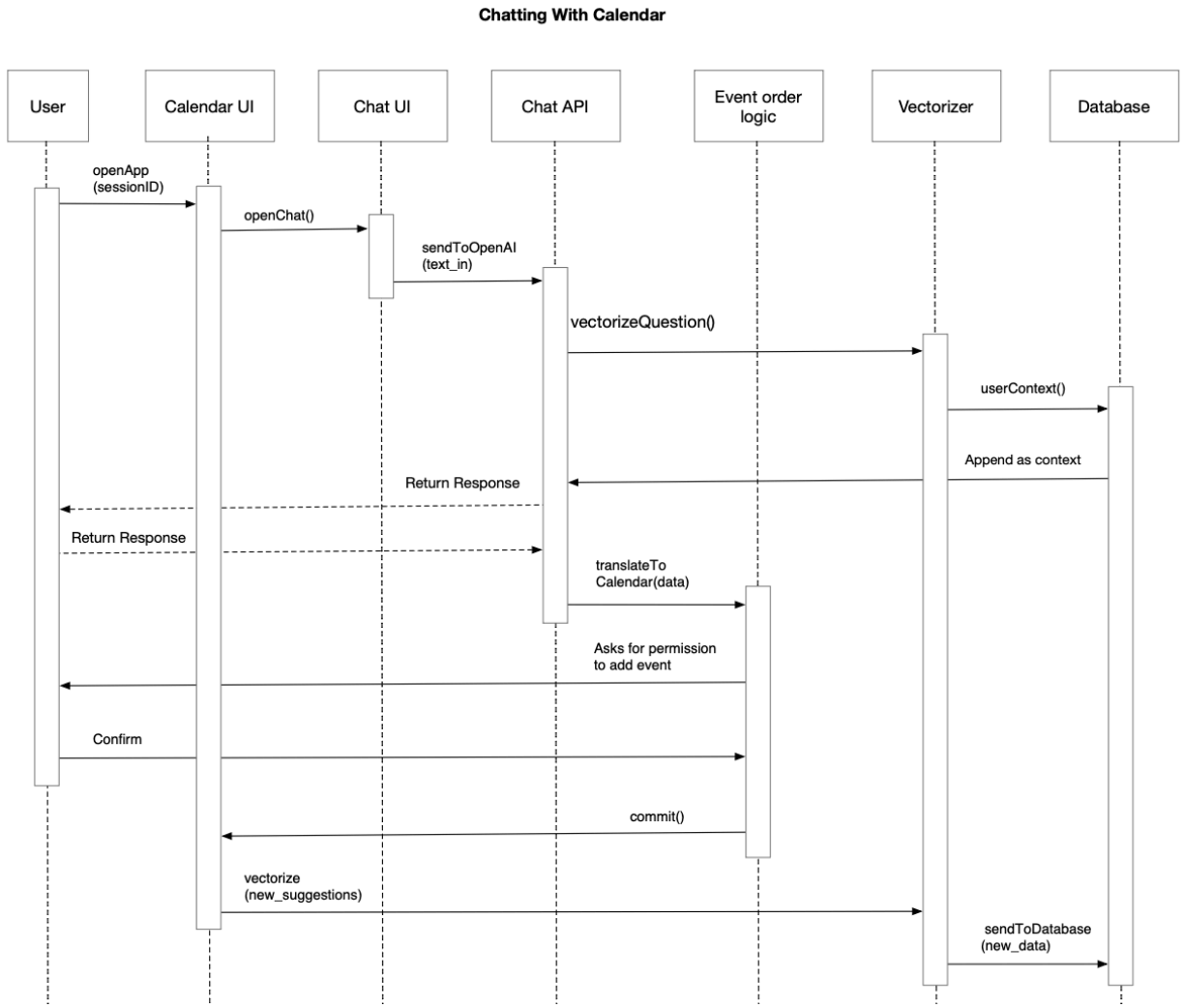
42. Functional Design

<Identify all significant workflows as sequence diagrams using the following format>

42.1 Diagram Title

42.2 Chatting With Calendar

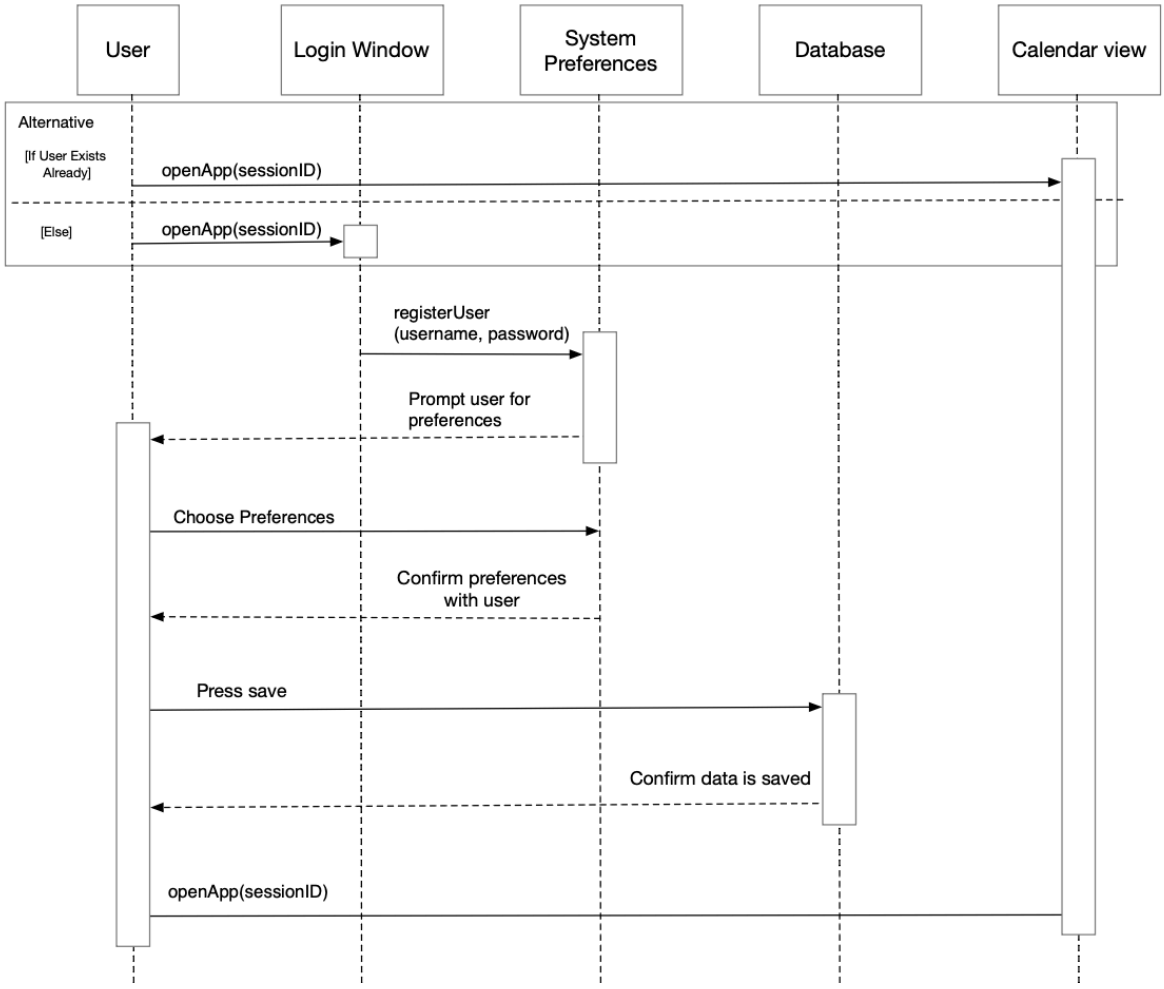
42.3 User Onboarding



Chatting With Calendar

- Considering this is a returning user or a first time user after they just finished registering their account, they will be prompted with the calendar UI after opening the app, which calls the openApp(sessionID) function.
- Once in the calendar UI and upon clicking on the chat icon they will be sent to the chat UI through the function openChat().
- After interacting with the chat UI and a user hits send, a chain of events happens, starting with the call to the function text_out sendToOpenAI(text_in).
- After the API receives the text input from the user, it triggers a function called userContext() that retrieves information from the database. Once it receives that, it processes and combines the user input with the context given by the database, which is just previously stored information that helps the chat make decisions. Finally, it calls the function translateToCalendar(data) which asks the user for permission to add the new suggestion to the calendar while showing a preview (goes to the event order logic).
- The event order logic now calls the function commit() which permanently adds the newly made suggestions to the calendar if the user approves it, else it sends the user back to the chat UI to start the whole process again.
- Once the new suggestions have been added permanently to the calendar, they are sent to the vectorizer adapter upon the automatic function call vectorize(new_suggestions).
- Lastly, the vectorizer sends the now vectorized data to the database with the sendToDatabase(new_data) function.

User Onboarding



User Onboarding

- For first time users of the app, the user will open the app, calling the openApp(sessionID) function and be prompted with the login window. For existing users, they will be sent directly to the main/calendar view of the app through openApp(sessionID) as well. The sessionID will be null for new users of course, which is an automatically generated string that uniquely identifies a user (using cookies).
- From the login window, upon filling in the necessary information, the user will be sent to the system preferences page by clicking the “register” button, which calls the registerUser(username, password) function.
- Once in the system preferences page, the user will choose their preferences from the built-in ones, and make new ones by clicking the plus sign button. Once they click “save”, their information will be sent to the database through the sendTo(username, password, preferences) function, where all of their information will be stored in the database.
- Once their information is stored, they will be automatically redirected to the calendar view by the mainPage(sessionID) function, where the sessionID function is an automatically generated string that identifies that user and keeps them logged in for a certain period of time. Once that session runs out, existing users will have to login again by inputting their credentials at the login window. They will be authorized and logged in or not with the loginAuthorization(username, password) function.

43. Structural Design

The following class diagram represents an overview of the component breakdown in our system. A primary focus for this design is modularity