Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Ai Yoshida

DEVELOPMENT OF A WEB APPLICATION FOR CROSS-TIMEZONE MEETING SCHEDULING

SUPERVISOR

DÁVID SIK

BUDAPEST, 2025

Table of Contents:

1. Introduction

Recently, remote work and international teams have become more common, especially after COVID-19. Therefore, one big problem arises: it is difficult to schedule meetings among team members who live in different time regions.

I have experienced this problem while I was working remotely as an intern at Japanese company that provides programming educational platform for children. Usually, I needed to organize meetings among three countries, Hungary, Japan, and one more country like USA or India. Depending on the members who join the meeting, the time zones that I needed to consider were different. Each time I had to go through a troublesome process: firstly, search for each person's time zone (knowing country name is not enough, since U.S. has several time zones), secondly find available time (usually between 8:00 and 22:00) for each time zone and find overlapping time slots among three or four different time zones, finally, choose several timeslots from the candidates. Especially around period when summertime starts or ends, because of these dynamic changes, organizing meetings usually causes a lot of confusions and mistakes.

To solve this problem, I have searched several meeting scheduling tools, but most of them do not consider time differences, are made for large business teams or part of paid services and are not suitable for small teams or private use. This result gave me an idea of this project. I thought I can create a web application to solve this problem. Also, I was sure that there are some other people who face the same difficulties.

The purpose of this project is to develop a simple web application for cross time zone meeting scheduling. React is used for Frontend development, while Backend is implemented using Fast API, and SQLite for database. Main functions are to visualize users' time differences, to provide intuitive timeslot selection implemented by FullCalendar, and sharing meeting via URL.

2. Related Work and Technologies

This project was developed using JavaScript, FastAPI, React, FullCalendar, and SQLite. For the development environment Visual Studio Code was used. Also, moment.js library was used for time and time zone conversions. These technologies are well-supported, widely used and appropriate for building modern single-page applications. React is suitable for implementing a dynamic frontend while FastAPI is good to develop simple and fast backend using Python. SQLite is the database, which is light, so it was ideal for small scale project like this, since this is local environment development.

Before starting the project, I have done several research for existing tools which tried to solve the same problems. As a result, there are some web applications that calculate time differences, but most of the time with restrictions.

For example, World Clock Meeting Planner offers meeting scheduling functions with at most three different locations, but it requires user to set participants' time zones manually [1]. However, it is often troublesome for users since we cannot usually specify time zone information from country name. For instance, even if the user knows one participant lives in U.S, but it is not enough to schedule meeting since U.S has many time zones inside the country.

While the UI of everytimezone.com is practical, its meeting scheduling feature requires payment [2]. Therefore, it is not likely to use by students or individual users.

Calendly, Google Calendar, and Zoom Scheduler are popular meeting scheduling applications, but they do not have time zone difference visualization functions [3], [4], [5]. Most of the cases, users need to check the differences manually. Also, their main target is business use, so they assume that participants know each other's schedule or users' weekdays schedule in assigned for business.

To conclude, these existing meeting scheduling tools require manual steps to check the time differences, and not practical for small teams or individuals. Therefore, goal of my project is to develop simple, free, specializing for time difference adjustment web application meeting scheduling.

3. Project Goals and Requirements

3.1 Project Objectives

The main goal of this project is to develop a local web application that allows two or three people (each living in a different time zone) to schedule meetings easily.

3.2 Functional Requirements

The application should meet these following functional requirements:

- Users can create an account and can log in or log out.
- A unique ID is automatically assigned to each user when they create an account.
- E Each user can select up to 10 time slots, each fixed at 30 minutes, by clicking on the calendar when creating a new meeting schedule.
- Users can invite up to two contacts from their contact list.
- When invitees are selected, their time zones are visualized alongside the calendar to reflect time differences.
- When a user creates a new meeting poll, a unique URL is generated, which invitees can use to access the poll.
- The user ID is stored in localStorage, allowing the browser to save login information.

3.3 Non-Functional Requirements

In addition to the functional requirements above, the application should meet the following non-functional requirements:

- The application is fully functional in local environment and does not require internet connection.
- The UI should be intuitive, and time slot selection should be done smoothly.
- Manual testing is possible by using at least two different browsers.

4. System Design and Architecture

4.1 Architectural Overview

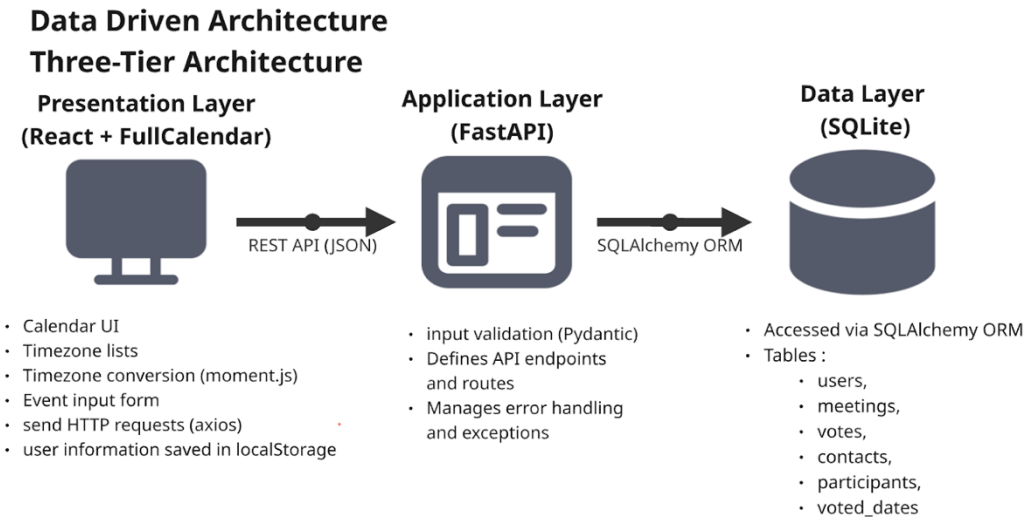

Figure 1: Architecture overview

This application is built based on three-tier architecture, following a data driven architecture approach. The presentation layer is responsible for rendering calendar UI, handling time zone and time slot conversions, and managing user's timeslot selection. It also sends HTTP request to the application layer using Axios. The application layer receives requests from frontend, performs input validation using Pydantic, defines RESTful API endpoints and routes, and handles error management. The data layer uses a local SQLite database which is accessed through SQLAlchemy ORM.

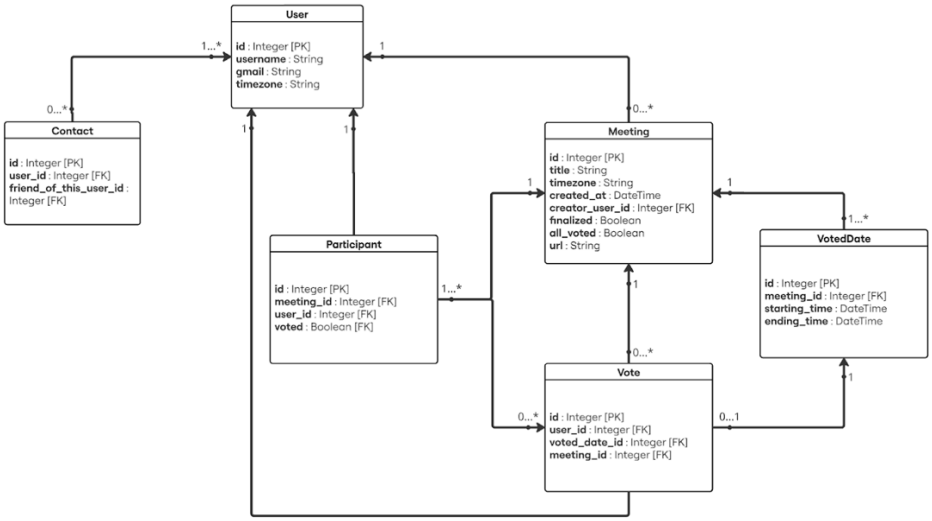

Figure 2: Database diagram

The database contains multiple tables including users, meetings, votes, contacts, participants, and voted_dates. This structure supports clear data flow and separation of concerns, making the application easier to maintain and scale.

4.2 Project Structure

4.2.1 Frontend File Structure

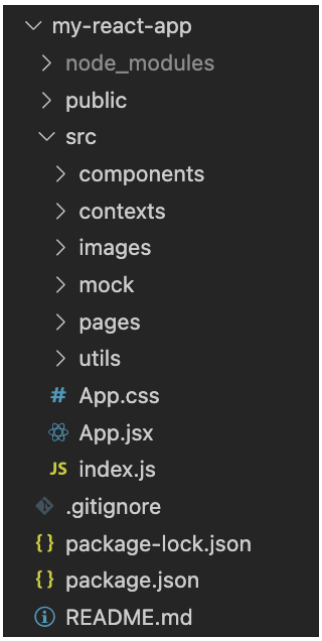

Figure 3: Frontend File Structure

The frontend file structure follows general React application structure. Main source files are located inside src directory, and depending on their roles, files are

organized into subfolders. There are reusable UI components such as buttons or forms in the component folder, while page unit components for routing are locateed in the pages folder.

4.2.1 Backend File Structure

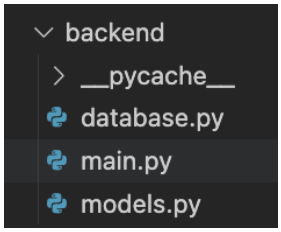

Figure 4: Backend File Structure

All backend files are located in the backend directory. The main py serves as entry point of application, and responsible for API routing and middleware configuration. The models.py defines SQLAlchemy ORM models and describes database table structure and their relations. The database connection and session are in database.py.

Detail of the development is published on the GitHub [6].

5. Implementation

5.1 Time zone Format: IANA

In this project, IANA time zone format (ex: “Asia/Tokyo”, “Europe/Budapest”) is used to describe user’s time zone. This format covers regions all over the world and constantly updated to the dynamic changes including daylight saving time or political decisions [7]. By implementing this format, this project can maintain consistency on time and time zones conversions.

5.2 Timeslot Format: ISO 8601

Time slots are handled using ISO 8601 string format. When a user selects a time on the calendar, the time is initially in the user’s local time, such as:

"2025-05-25T03:00+07:00”: Local time with offset (received from FullCalendar)

However, this local time is converted to UTC before saving:

"2025-05-25T03:00:00Z” : UTC format (converted using moment.js)

Using UTC helps avoid time zone conflicts when comparing or displaying times for other users. UTC time is also saved into SQLite as a TEXT value, making storage and comparison easier.

5.3 Interesting Source Code: Mistakes with Time Conversion

When we handle time slot conversion for time difference, we need to pay attention of this time format such as this; “2025-05-25T03:00:00". This is neither local time with offset nor UTC time which is with Z. Even if we save UTC time to the SQLite, it returns the time without Z. Therefore, we need to be careful when we need to convert this time using moment.js

The moment.js automatically convert time without explicit time zone into local time [8]. The moment.js may assume it is already in local time, which causes unexpected results. For example, in my case it automatically understands this time as Europe/Budapest time, UTC + 2, not UTC time. Therefore, it is necessary to clarify that the timeslot is UTC when we handle the timeslot without obvious offset declaration.

Incorrect example (can cause bugs):

UTC -2 + 9 => 2025/05/25 10:00

```
start: moment("2025-05-25T03:00:00").tz("Asia/Tokyo").format(),
end: moment("2025-05-25T03:00:00").tz("Azia/Tokyo").format(),
```

Correct example (safe and clear):
UTC + 9 => 2025/05/25 12:00

```
start: moment.utc("2025-05-25T03:00:00").tz("Asia/Tokyo").format(),
end: moment.utc("2025-05-25T03:00:00").tz("Azia/Tokyo").format(),
```

6. Engineering Experiences

From this project development experience, I had gained several important learnings. I had some experience in application development before, mainly through university assignments and an internship However, implementations were usually very specific, and requirements were given. Therefore, this project was my first experience that I started from scratch and developed all of frontend, backend, and database unified as one web application. This gave me very significant insight into how each layer is connected and functions. I realized how important it is for each part to have a clear role. Understanding this flow helped me debug problems more easily and structure the project better.

Secondly, implementing time zone conversion was also a great experience for me to learn during this project. I had no knowledge about time zone format or conversion, so I started by doing research. Managing time differences between users in different regions was more complex than I expected, especially when considering daylight saving time and differences in UTC offsets. I learned how to use the IANA time zone format and the moment.js library to convert and display times properly for each user.

Lastly, database design was challenging for me. At the beginning of this project, I did not have enough knowledge about how to design a database structure properly. I started implementing the code without clearly thinking about how different tables or data should be connected. Later, I realized that this caused many problems. I had to change the database structure several times, and each time, it took time to fix the related code and sometimes created bugs. Through this experience, I learned that database design should be done carefully at the beginning, and it is very important to think about flexibility and future extensibility.

Though all of these experiences and challenges, it helped me understand how full stack applications are structured and function. I would like to apply this knowledge that I gained in this project to the next stage of development for my thesis.

7. Reference

[1] Time and Date AS, "World Clock Meeting Planner," Time and Date AS, 25 5 2025. [Online]. Available: https://www.timeanddate.com/worldclock/meeting.html. [Accessed 25 5 2025].

[2] "Every Time Zone," Every Time Zone, 25 5 2025. [Online]. Available: https://everytimezone.com/. [Accessed 25 5 2025].

[3] "Calendly," Calendly, 25 5 2025. [Online]. Available: https://calendly.com/. [Accessed 25 5 2025].

[4] "Google Calendar," Google, 25 5 2025. [Online]. Available: https://calendar.google.com/calendar/. [Accessed 25 5 2025].

[5] "zoom," Zoom Communications, 25 5 2025. [Online]. Available: https://www.zoom.com/. [Accessed 25 5 2025].

[6] A. Yoshida, "GitHub Repository - ProjectLab-meeting-app," Ai Yoshida, 25 5 2025. [Online]. Available: https://github.com/aiyoshida/ProjectLab-meeting-app. [Accessed 25 5 2025].

[7] "IANA Time Zone Database," Internet Assigned Numbers Authority (IANA), 25 5 2025. [Online]. Available: https://www.iana.org/time-zones. [Accessed 25 5 2025].

[8] "Moment.js," Moment.js Team, 25 5 2025. [Online]. Available: https://momentjs.com/docs/#/parsing/string-format/. [Accessed 25 5 2025].

8. Appendix: Weekly Work Daily Table

Week	Dates	Main task
1	Feb 10th – 16th	Topic selection and research
2	Feb 17th – 23rd	Kickoff meeting with the supervisor and research on the technology stack.
3	Feb 24th – Mar 2nd	Requirement definition
4	Mar 3rd – 9th	Writing a progress report and submitting it to the supervisor.
5	Mar 10th – 16th	Studied React using a Udemy course.
6	Mar 17th – 23rd	Learned and researched database design.
7	Mar 24th – Mar 30th	Created UI design using Figma
8	Mar 31st – Apr 6th	Frontend development: building pages and components.
9	Apr 7th – 13th	Frontend development: implementing calendar UI, time conversions, and time zone handling
10	Apr 14th – 20th	Backend development: implementing endpoints
11	Apr 21st – 27th	Backend development: modified and connected to the database.
12	Apr 28th – May 4th	Manual testing in a local environment
13	May 5th – 11th	Prepared the presentation slides.
14	May 12th – 18th	Wrote documentation
15	May 19th – 25th	Wrote documentation