Calendar Application - Project Report

**1. System Architecture and Design Decisions**

**1.1 Frontend Architecture**

For the frontend, I decided to use **React** as the framework to implement the user interface. React was chosen because of its flexibility, component-based structure, and wide community support, making it an ideal choice for handling dynamic content like calendar events. I also used **React Router** for managing navigation between views (monthly and daily calendar views).

**1.2 Backend Architecture**

For the backend, I opted for **Spring Boot** in Java (JVM-based language), as it is a robust framework for building scalable web applications. The application’s backend serves as an API layer that communicates with the database to manage user data and events.

**1.3 Database Choice**

I used **PostgreSQL** for data storage. It was chosen for its stability, scalability, and strong support for relational data management, which is ideal for handling event data, user management, and recurring event functionality.

**1.4 Design Choices**

* The application is designed to be responsive, ensuring it functions seamlessly across both desktop and mobile devices.
* I used **Material-UI** for UI components, which provides a modern, consistent design system that is both intuitive and customizable.
* Authentication is handled via **JWT (JSON Web Tokens)**, ensuring secure login and session management.

2. Used LLMs and AI-Based Tools

During the development of the calendar application, I leveraged AI tools to enhance productivity and support various development tasks:

2.1 LLM for Code Assistance

I used ChatGPT to generate code snippets for backend routes, frontend components, and event handling logic. Additionally, the LLM was used to explain complex concepts regarding the integration of frontend and backend.

2.2 Testing Assistance

AI was also used to generate unit and integration test cases. I used LLMs to create mock data and simulate different user actions to ensure the application’s functionality was robust, such as testing event creation, deletion, and notifications.

2.3 Documentation Support

For documentation and report writing, I used LLM-based tools to draft parts of the report, generate explanations for design decisions, and ensure proper grammar and clarity in the final write-up.

**3. Where Did the AI Tools Work Well?**

**3.1 Code Generation**

The AI-based tools helped me write repetitive boilerplate code quickly. For example, generating basic routes for event CRUD (Create, Read, Update, Delete) operations in the backend was done efficiently using prompts in the LLM.

**3.2 Testing and Debugging**

AI tools provided valuable input during the creation of unit and integration tests. For example, when I needed to ensure event creation functionality worked properly, the AI suggested edge cases that I had not considered, such as overlapping event times and missing event descriptions.

**3.3 Documentation and Reporting**

The AI greatly accelerated my documentation process. It helped draft reports, describe system architecture, and explain technical decisions clearly. By automating part of this, I was able to focus more on the development side.

**4. Where Did the AI Tools Not Work as Intended?**

**4.1 Code Generation Limitations**

Although the LLM generated useful snippets, it often made assumptions about the implementation, which were not always in line with my project’s structure. For example, it suggested using a different database than what I had chosen, leading to some initial confusion. I had to modify the generated code to fit the architecture I had designed.

**4.2 Lack of Contextual Awareness**

The LLM sometimes struggled to understand the context of certain technical constraints I was working with, particularly when it came to integrating multiple frameworks. This occasionally led to the AI generating incompatible solutions, such as suggesting frontend libraries incompatible with my backend setup. I had to refine my prompts and provide more specific guidance to achieve better results.

**5. Development Process**

**5.1 Tracking Issues and Pull Requests**

Throughout development, I used **GitHub Issues** to track progress and manage the features implemented for the calendar application. Each feature (e.g., user registration, event creation) was tracked as an issue with relevant acceptance criteria.

I followed a **Gitflow workflow**, creating separate branches for each feature and submitting pull requests for review. This allowed for a clean and organized development process, ensuring that the main branch always contained stable, tested code.

**5.2 Continuous Integration (CI) Pipeline**

I implemented a CI pipeline using **GitHub Actions**, which ran tests, linters, and code formatters every time a pull request was made. This ensured high-quality, error-free code was merged into the main branch.

**6. Testing**

**6.1 Login Screen**

**The user is prompted to input their email address and password to access their calendar**

**A login screen with blue and white text

AI-generated content may be incorrect.**

**6.2 Calendar View**

**The calendar is clear and well-organized with days displayed in grid format.**

A screenshot of a calendar

AI-generated content may be incorrect.

**6.3 Add New Event Screen**

adding a new event to the calendar. This pop-up window enables the user to enter key event details.

A screenshot of a computer

AI-generated content may be incorrect.

**6.4 Calendar View with Event**

A screenshot of a calendar

AI-generated content may be incorrect.

**7. Submission**

The calendar application was developed and tested using **PostgreSQL** for data storage, **React** for the frontend, and **Spring Boot** for the backend API layer. The application is hosted in a **GitHub repository** and follows the **Gitflow workflow** for version control.

**7.1 Repository Setup**

The GitHub repository includes:

* Source code for the frontend and backend
* A **README.md** file that provides instructions on setting up the development environment
* A **. gitignore** file to prevent unnecessary files from being committed
* Automated tests for both frontend and backend

**7.2 Required**

To run the application, the following tools are required:

* **Java JDK 17 or higher**
* **Node.js LTS version 18 or higher**
* **PostgreSQL** for the database
* **Postman** for Testing API
* **Apache Maven**
* **Jave Spring**