Mid Term Project – AI Code Reviewer

Members :

Sher Shah Khan – 46119

Muhammad Talha – 47460

Github Link : https://github.com/shershahkx/Gen_AI_Mid_Term

AI Code Reviewer - Project Report
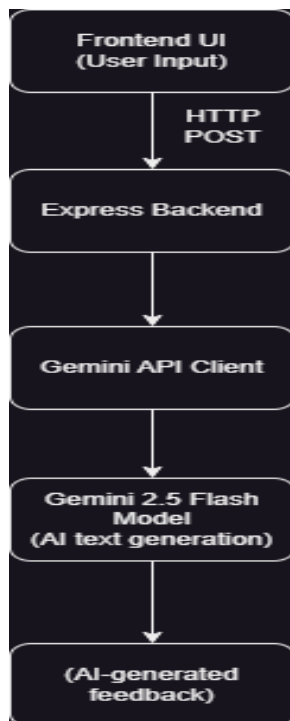
1. Project Scope

The **AI Code Reviewer** project is a lightweight Node.js backend that integrates **Google Gemini (Generative AI)** to automatically analyze and review code snippets.
The primary goal is to assist developers by providing **AI-driven feedback** under three categories:

- **Readability** – evaluates code clarity, naming conventions, and formatting.

- **Best Practices** – identifies unsafe coding patterns or design issues.

- **Optimization** – suggests performance or simplicity improvements.

This project demonstrates how **AI APIs** can be embedded into developer tools to enhance code quality checks, acting as an intelligent assistant for software teams or students.

2. Design

3. Implementation Steps

Set up Environment:

Install dependencies

```
npm install express cors body-parser dotenv @google/generative-ai
```

Backend Initialization

API Integration

API Endpoint

4. Input/Output

AI Feedback

Readability

- Issue: Missing Javadoc comments.
- Suggestion: Add Javadoc comments for the class and its public methods to explain their purpose, parameters, and return values.

Best Practices

- Issue: Using generic `Exception` for "not found" scenarios.
- Suggestion: Use more specific custom exceptions (e.g., `ReservationNotFoundException`) or return `Optional<Reservation>` for `getReservationById`.
- Issue: `getAllReservations()` returns the internal mutable list directly, breaking encapsulation.
- Suggestion: Return an unmodifiable view of the list using `Collections.unmodifiableList()` or a defensive copy to protect the service's internal state.

Optimization

- Issue: `getReservationById`, `cancelReservation`, and `updateReservation` operations involve linear searches (O(n) time complexity).
- Suggestion: Consider using a `HashMap<Integer, Reservation>` to store reservations, allowing for average O(1) time complexity for ID-based lookups, updates, and removals.

5. Reflection

Challenges Faced

1. Response Extraction — The Gemini API sometimes returns output in multiple formats (raw text), requiring dynamic extraction logic.

2. Prompt Tuning — It took several iterations to craft a prompt that ensures concise, structured, and relevant feedback instead of verbose explanations.

3. Error Handling — Managing incomplete API responses or network errors required careful try/catch wrapping.

6. Learnings

- Gained hands-on experience integrating **Google's Generative AI SDK** into a Node.js app.

- Learned how to handle asynchronous API calls, dynamic object parsing, and response normalization.

- Understood the importance of **prompt design** for achieving predictable AI behavior.

7. **Prompt Design Insight**

The prompt emphasizes clarity, brevity, and structure, ensuring consistent output under three headings. By specifying format and tone, the AI consistently returns concise and professional reviews.