Project Report

High Level Design Document

Group 5

Joe Lin A01079256
Louise Li A01377263
Blaise Klein A01300754
Reece Melnick A01349668
Inez Yoon A01066348

1. Project Summary

FANGO is a just-in-time AI-powered travel companion that helps users instantly understand the world around them. Travellers can take or upload a photo of any object—food, signs, products, landmarks—and FANGO:

- Identifies the object using OpenAI Vision.
- Translates it into the user's target language.
- Generates two safe, travel-friendly example phrases.
- Saves the result into the user's personal history to review later.

Unlike traditional apps that require lessons or preparation, FANGO is built for real-time learning during travel moments.

2. Problem Statement

Travellers frequently encounter unfamiliar objects, foods, or signage when abroad. Traditional translation apps require manual typing, which is slow and fails when the user doesn't know the word. Lesson-based language apps do not help in spontaneous, real-world situations.

FANGO solves this by enabling:

- Photo-based instant translation.
- AI-generated image detection and proper phrases.
- Secure user accounts syncing personal history.
- Lightweight micro-learning in real-time.

3. Key Architectural Choices

3.1 Django Backend

Django was chosen because:

- Clean, maintainable MVC architecture.
- Built-in admin, authentication, and security features.
- Excellent ORM (avoiding manual SQL).
- Easy integration with PostgreSQL and Redis.
- Python is familiar to the team.

Django handles:

- Authentication (JWT + Redis hybrid).
- Translation workflow.
- History storage.
- API endpoints consumed by React.
- OpenAI service integration.

3.2 PostgreSQL Database

PostgreSQL Database was chosen because:

- Strong relational schema support.
- ACID compliance ensures consistent user/translation data.
- Ideal for structured data: users, histories, languages, quizzes.
- Supports foreign keys, constraints, and indexing.
- More SQL-standard compliant than MySQL.

Also, PostgreSQL stores:

- Users
- Photos/paths
- Translations
- History
- Learning goals & difficulty

3.3 Redis

Redis adds:

- JWT Session Statefulness → revocable tokens
- Caching → reduces PostgreSQL load
- Rate limiting → protects from DoS attacks
- Fast key-value reads → ideal for token checks

This hybrid JWT authentication architecture makes FANGO significantly more secure than stateless token-only systems.

3.4 React + TypeScript Frontend

React and TypeScript were chosen because:

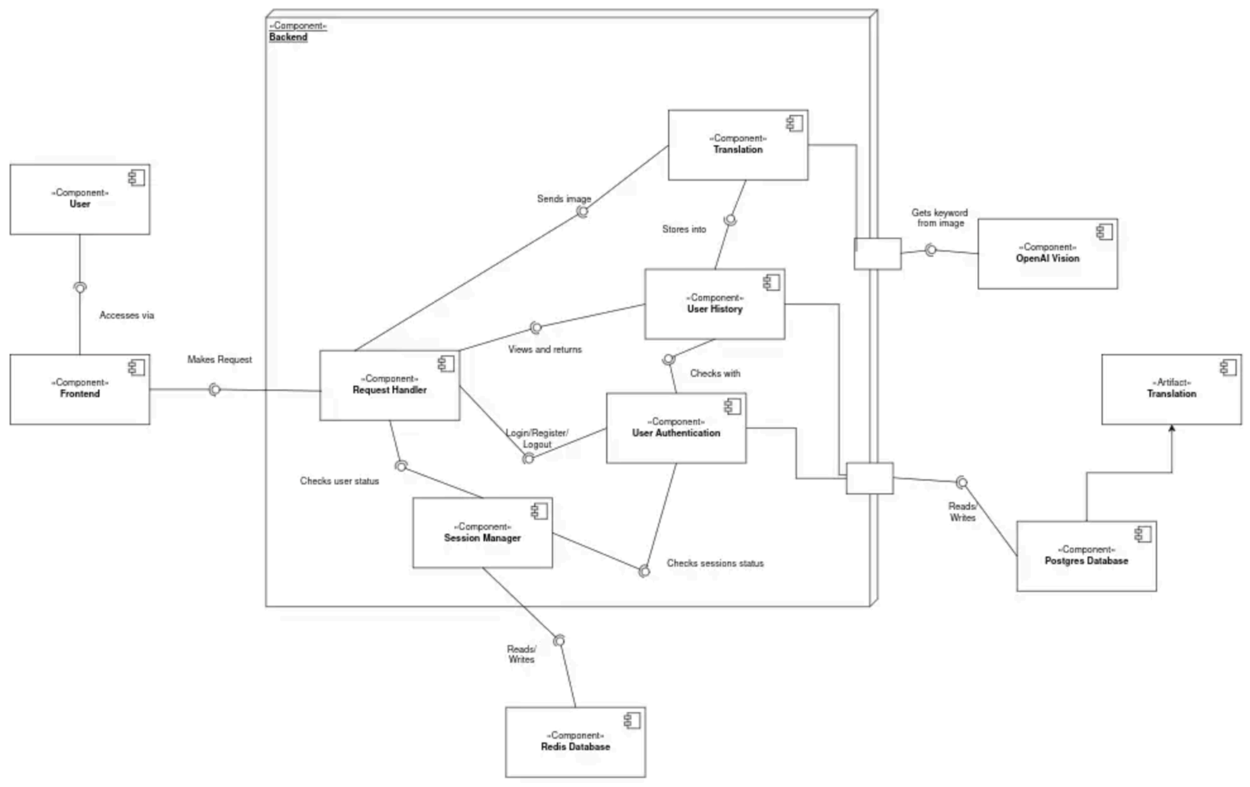
- Component-based architecture improves reusability.
- Virtual DOM → fast UI updates.
- Rich ecosystem (React Router, Zod, Axios, etc.).
- Full TypeScript type-safety.
- Great developer experience

React and TypeScript handles:

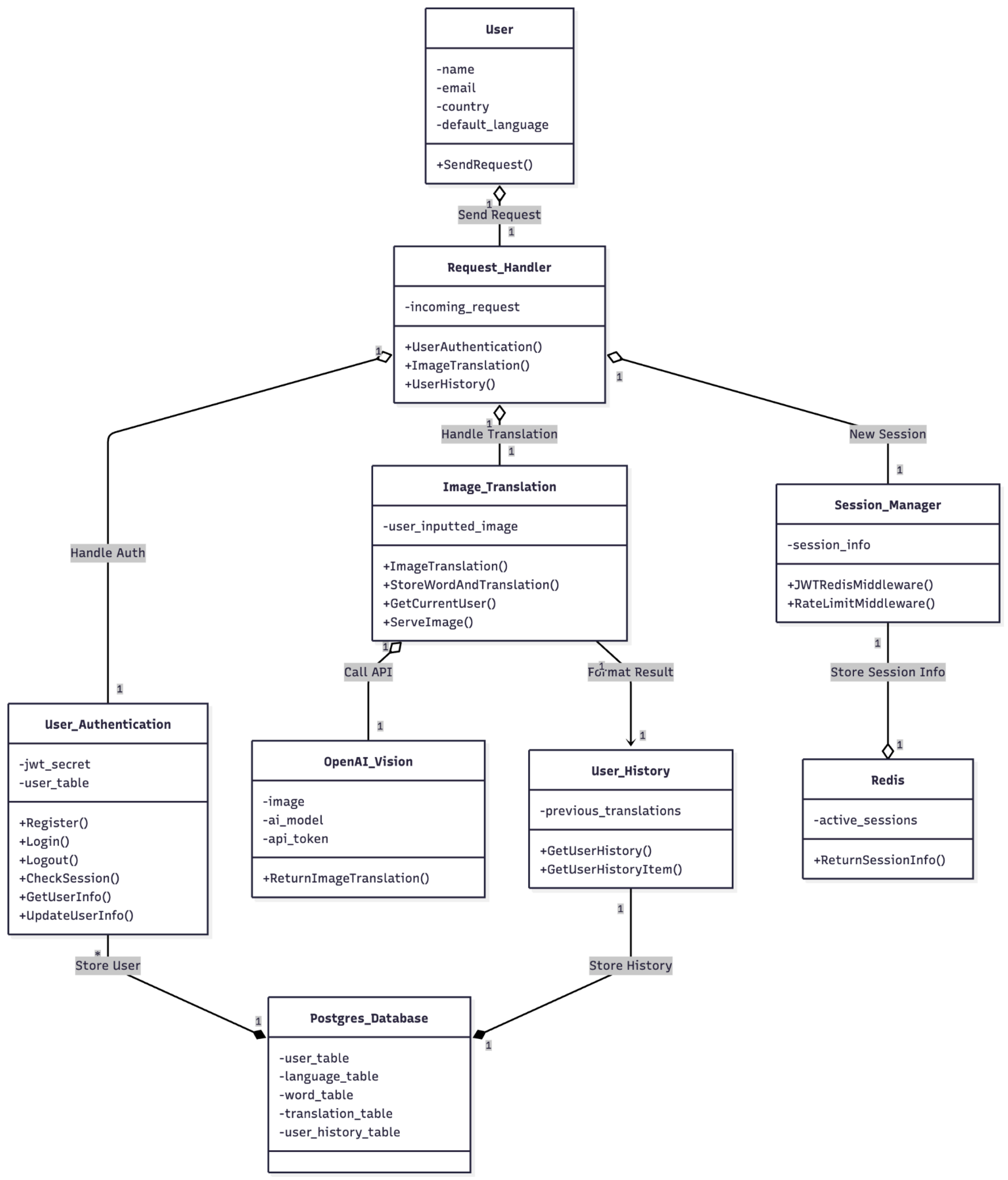
- Camera + image upload.
- Signup flow & multi-step forms.
- Translation results UI.
- History pages.
- Quiz & learning features.
- All communication with the backend API.

4. System Architecture

4.1 Component Diagram

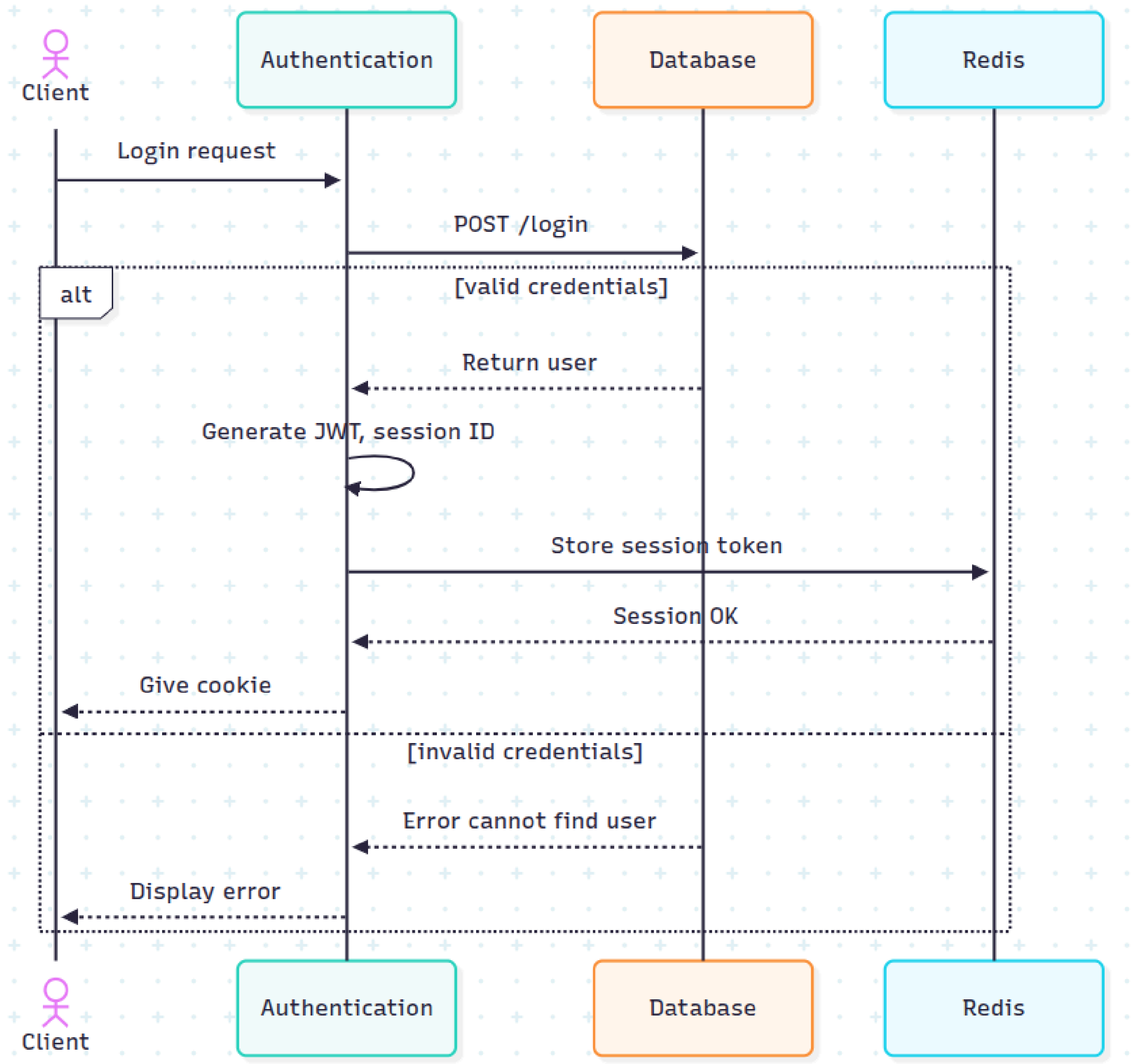


4.2 Class Diagram

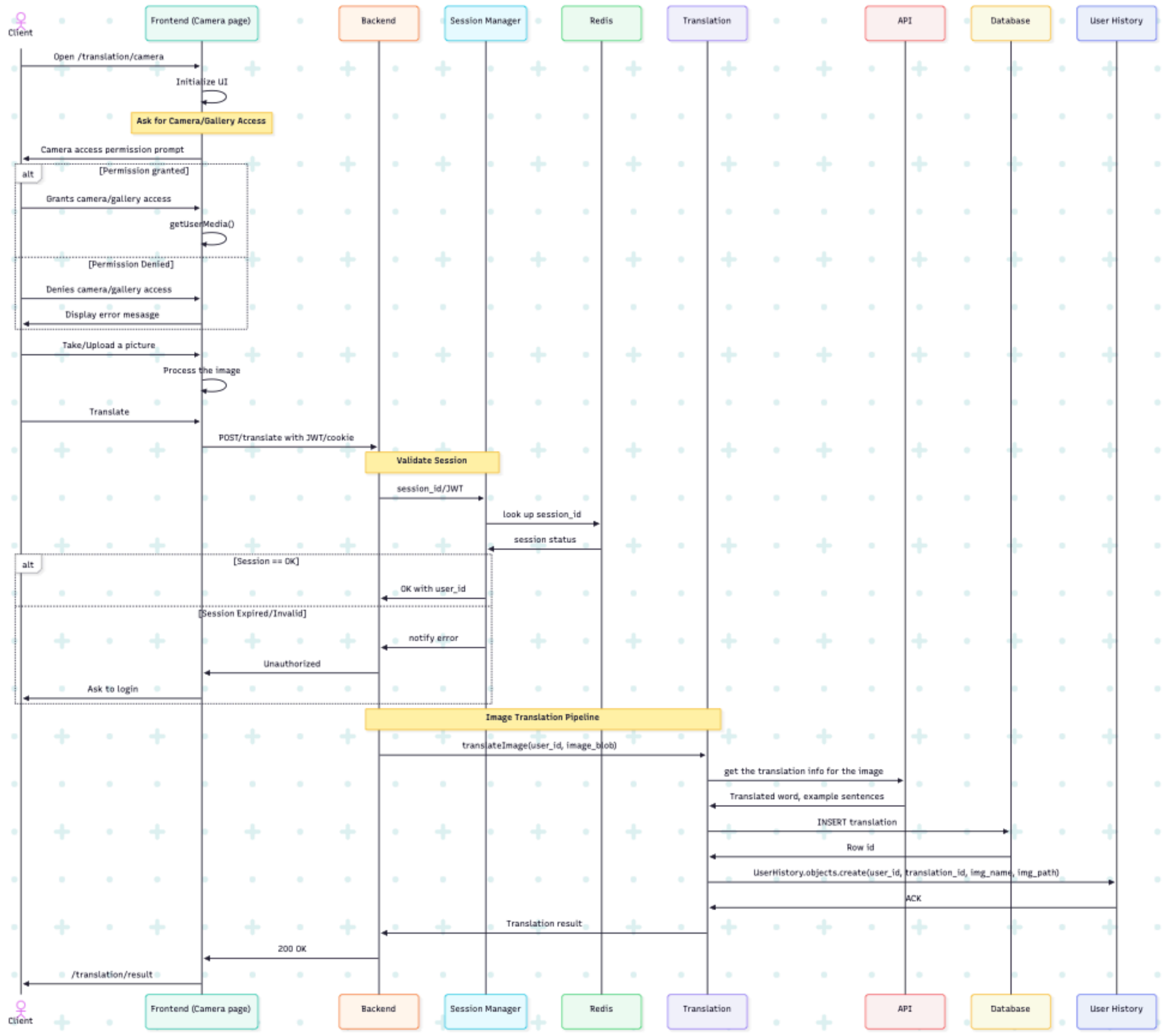


4.3 Sequence Diagram

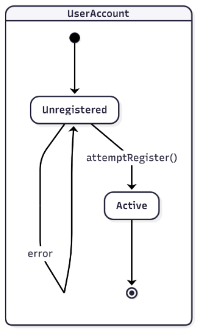
4.3.1 Account Login

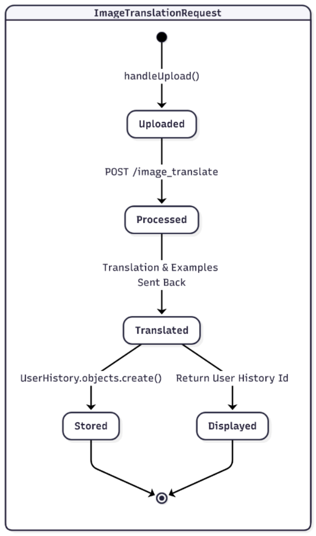


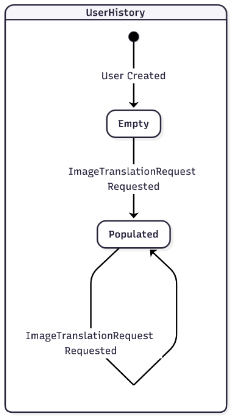
4.3.2 Camera Logic

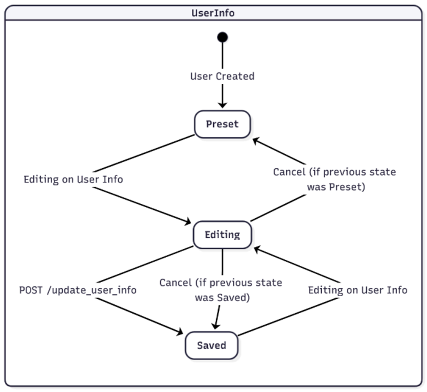


4.4 State Diagram






UserHistory
User Created
Empty
ImageTranslationRequest Requested
Populated
ImageTranslationRequest Requested



5. Key Challenges & Resolutions

Challenge 1 : React data handling

- Problem:
    - It was not clear how to handle data from multiple frontend pages to backend with React state.
- Resolution:
    - Researched on React state, context and props. We decided to use state for small data, context for medium data and props for the whole sign-up data. Data handling was safe and fast with the combination of state, context and props.

Challenge 2 : Image Translation

- Problem:
    - Outdated and unreliable OpenAI documentation.
- Resolution:
    - Delegate time into finding and reading appropriate, updated documentation for our translation feature. Additionally, we also had to conduct additional testing to ensure everything was working as intended.

Challenge 3 : Issues with CI

- Problem:
    - Had initial issues setting up workflow for github actions, without settings access (only owner has access to it). Also had issues with docker automatically testing as environment variables did not translate over without creating a temporary .env file and copying the variables over.
- Resolution:
    - Switched ownership of Github repository for troubleshooting in environment variable and secrets settings. Went through many iterations of testing for workflow and fixed testing on environment to Testing.

Challenge 4 : Management of Git State

- Problem:
    - Git version mismatch and problems caused by merging old versions
- Resolution:
    - Early on we had pull requests from feature branches to a dev branch. This allowed any problems to be rolled back easily as the last working commit was always before a new branch got merged. Later problems arose from old versions of code being remerged into dev, but this was solved by merging from the dev branch into the feature branch first.

Challenge 5 : Sessioning

- Problem:
    - JWT Statelessness
- Resolution:
    - Integrated Redis to store JWT session tokens, and having the system check session validity by comparing against Redis. The stored session token can be overwritten to enable token revocation at any point.

Challenge 6 : Security

- Problem:
    - DOS vulnerability
- Resolution:
    - Redis was also used to store rate limit keys that ensured users did not overload the server with requests. These rate limit keys would be checked for every single request, to ensure the request limit has not been exceeded.

Challenge 7 : Performance

- Problem:
    - Frequent redundant database queries
- Resolution:
    - To improve performance, we reduced the number of queries made by simply caching the frequently accessed data, specifically the user information, into the Redis session keys.

6. Lessons Learned

6.1 Backend

- Hybrid JWT systems dramatically improve security.
- API reliability improves with strict Pydantic validation.
- Breaking the backend into services (OpenAI, utils, middleware) improves maintainability.

6.2 Frontend

- React Context helps simplify multi-step flows.
- Strong typing with TypeScript prevents runtime bugs.
- Animations (Framer Motion/SpringMotionLayout) improve UX but require careful performance tuning.

6.3 Architecture

- Redis is extremely powerful and solves multiple categories of problems.
- Storing only file paths (not images) reduces DB size.
- A clear separation of concerns (frontend, backend, AI service) keeps the project scalable.

7. Future Improvements

7.1 Short Term

- Daily Quiz
  - Frontend pages are already created.
  - If backend logic is added for personalization, users can improve their language skills by reviewing their search history with the daily quiz feature.

7.2 Long Term

- Offline mode using local ML models
  - This feature will give more convenience for travellers who cannot connect to the Internet all the time.
  - This feature would give us a deep understanding of how ML models work.

8. Conclusion

The FANGO architecture supports real-time, photo-based translation with a secure and scalable design. Using Django, PostgreSQL, Redis, and a modular React frontend allowed the team to implement:

- AI-powered translation
- Secure authentication
- Real-time phrase generation
- Reliable user history

This architecture provides a strong foundation for future enhancements such as offline translation or AR-based overlays.