# Project Planning Meeting Minutes – Implementation Phase

**Project:** Web-based Manga Platform with OCR and Translation
**Course:** Introduction to Software Engineering (HCMUS) – Instructor: Mr. Truong Phuoc Loc
**Meeting Date:** December 4, 2025
**Meeting Phase:** Implementation Planning Meeting

## Meeting Details

- **Time:** 15:00 – 17:00
- **Location:** Online (Discord video conference)
- **Attendees:** All group members (Group 07)
    - Le Anh Duy – 23127011
    - Tran Gia Huy – 23127199
    - Nguyen Thanh Luan – 23127296
    - Nguyen Thanh Tien – 23127539

**Purpose:** This meeting marks the beginning of the Implementation phase. With the Software Design Document completed, the team convened to plan the coding phase of the project. The goals were to allocate development tasks (front-end and back-end responsibilities), set up our collaborative development environment and workflow, establish milestones and a timeline for implementation, and ensure that everyone is aligned on how to realize the design in code. This meeting also addresses any final clarifications from the design and sets expectations for testing and deployment.

## Agenda

1. **Design Review for Implementation** – Quick review of design artifacts (ERD, class diagram, etc.) to ensure clarity on what to build. Address any ambiguities or changes needed before coding.
2. **Task Breakdown & Assignment** – Identify all major implementation tasks (frontend development, backend development, integration tasks) and assign them to team members. Define sub-teams for front-end vs back-end.
3. **Development Workflow & Tools** – Confirm the version control strategy (Git branching model), code review process, and project management tools usage (task board in Jira or GitHub Projects). Ensure everyone has access to the repository and any required services (e.g., the cloud database, API keys).

4. **Coding Standards & Practices** – Agree on coding conventions (naming, formatting) and any frameworks or libraries to use (reiterate the tech stack and any new libraries for convenience).
5. **Implementation Timeline & Milestones** – Establish internal milestones (e.g., “basic login working”, “upload & approval flow complete”) with target dates. Align these with the course deadlines (e.g., possibly an upcoming progress demo or the final submission deadline).
6. **Testing Strategy during Implementation** – Plan how and when we will create and run tests (unit tests, integration tests) during development, and who will be responsible for testing which parts.
7. **Deployment & Environment Setup** – Discuss where the application will be hosted for testing/demo (e.g., deploying the front-end and back-end, or running locally) and ensure environment variables and configurations are managed properly.
8. **Risk Assessment & Contingency** – Identify potential risks in implementation (e.g., integration difficulties, time constraints) and plan how to mitigate them.
9. **Next Steps & Checkpoints** – Plan regular check-in meetings or status updates, and set the date for the next team review (or code freeze date) before final testing.

**Discussion and Notes**

**1. Design Review for Implementation**

We began by reviewing critical parts of the design document to make sure there were no lingering questions as we start coding:

- **ERD Confirmation:** Nguyen Thanh Tien walked through the final ERD diagram, pointing out table names and relationships. All members agreed the database schema is clear and accounts for required data. We noted a minor adjustment: we will combine “Manga” and “Chapter” tables into perhaps a simpler structure for implementation (since each upload corresponds to a single chapter and for our project we might not have multiple chapters per title to implement fully). However, we decided to keep the schema as designed for future extensibility, but initial data might have one chapter per manga for now. No one raised objections, so the database schema stands as designed.
- **Class Diagram Clarity:** Tran Gia Huy highlighted the classes we will implement in code. Everyone confirmed understanding of the responsibilities of each module. For instance, ensuring that the **TranslationService class** will have a method to call the

external OCR API – we discussed the need to find a suitable API/SDK for OCR. We might use an open-source OCR (like Tesseract) for simplicity to demonstrate the feature if cloud API integration is problematic. This was noted as an implementation detail: we will try a simple approach first (perhaps using an open-source library on the server to parse text from images) because calling a paid API might be constrained. The design allowed either approach; we just need to pick one during implementation.

- **Any Design Changes?** The team asked if any design element now seemed unfeasible or needed simplification. One topic was the **comments feature**: implementing real-time comments (immediate display after submission) would require either frequent refreshing or use of websockets. We decided to implement it in a simple way (refresh comments list on submission). This doesn't change design significantly, just an implementation note. Another topic: the complexity of the **admin analytics**. Given time, we might not implement a full analytics dashboard; we could provide a basic page with counts (e.g., total users, total uploads, revenue sum). This is acceptable as it meets minimal requirements from that user story. So no major design changes were necessary, just conscious prioritization (comments and analytics will be implemented in a basic way).
- We confirmed that all team members have the final design document accessible as a reference throughout coding. We'll use it to guide our development and to ensure we meet each requirement.

**2. Task Breakdown & Assignment**

We outlined all the major components to be implemented and assigned primary responsibility for each. The breakdown was done by dividing front-end and back-end tasks, as well as feature-based tasks. The assignment leverages each member's strengths and the role distribution we planned:

- **Frontend (Next.js) Development:** We will have two members focusing on the client side:
  - *Nguyen Thanh Luan* and *Nguyen Thanh Tien* were assigned to the Frontend Team. This aligns with the planned roles of 2 frontend developers. They will collaborate on building the Next.js application. We further divided front-end tasks:
    - Tien will take the lead on implementing the **Reader-facing pages**: Home page, Manga listing/search, Manga detail page, Reader (chapter viewing) page. This includes integrating the translation UI (e.g.,

        hooking the "Translate" button to call the backend API and display results). Luan will also handle the comments section component on the reader page.

        - Luan will focus on **User account and uploader/admin pages**: Login/Register forms, Profile/My List page, Upload Chapter page, and Admin Dashboard pages. This includes forms for upload and admin views for approvals. Tien will integrate the payment UI as well (e.g., a modal or page for purchasing a chapter).
        - The two front-end developers will work together on shared components (navigation bar, general layout, etc.) and ensure a consistent design. They will frequently merge their changes since pages are interconnected.
        - They will also need to implement front-end state management for things like user authentication state (likely using Next.js API routes or context). Possibly they will use Next Auth or a custom JWT handling in localStorage.

- **Backend (Node.js/Express) Development:** The other two members will focus on server-side:
    - *Le Anh Duy* and *Tran Gia Huy* form the Backend Team, aligning with having 2 backend developers. We subdivided backend tasks by feature as well:
        - Duy will lead development of **User authentication & management** (login, JWT issuance, registration logic), **Payment processing logic**, and **Integration with external OCR/translation**. Duy's tasks involve setting up the Express server scaffold, configuring the PostgreSQL database connection/ORM, and creating routes for auth (login, logout, signup). For the payment, Duy will implement the endpoints and logic to mark purchases (this might just simulate success as discussed). For translation, Duy will integrate an OCR library or API and ensure an endpoint (/translate etc.) can be called by the frontend.
        - Huy will focus on **Manga content management**: this includes routes for fetching manga and chapter data (list of manga, details, pages), **Upload & Chapter management** (endpoints to upload a chapter, save images, change status), and **Admin actions** (approve/reject API endpoints, feature highlight toggling). Huy will also design the database schema implementation (migrations or ORM models for all

    tables) given his involvement in design. Essentially, Huy ensures that all CRUD operations for manga, chapters, comments, etc., are implemented.

    - They will work together on overlapping areas. For example, the comment feature touches both their areas (Huy handling the database and API for posting comments, Luan/Tien showing it on UI). Similarly, purchase flows require coordination (Duy doing backend for purchase, Tien doing frontend UI update).

- We also identified some tasks that overlap or need coordination:
  - **Comments Feature:** Huy will create the API route for posting a comment (e.g., POST /api/chapters/:id/comments) and fetching comments. Luan will integrate it on the UI. They will coordinate on the data format (JSON fields).
  - **Favorites & Reading History:** These are minor features. Possibly Huy will implement simple endpoints for marking favorites and storing progress. Luan/Tien will call them accordingly (e.g., when user clicks “Favorite” or as the user reads a page, periodically update progress).
  - **Testing:** We noted that writing automated tests is part of the Testing phase later, but during implementation, each developer should do basic unit testing of their module. We decided that **All members** are responsible for testing their own code initially (e.g., using Postman to test APIs or running the web UI to ensure it works). Formal test cases will be executed later, but we want to integrate testing early. We might also create some sample data for testing (like a few manga and users in the database) to use during development.
- The task list was quite extensive, so we prioritized core features first (auth, reading, uploading, approving) and secondary features after (comments, ratings, analytics). If we find ourselves short on time, we prefer to have a working core system and can reduce scope on some extras (for instance, a very basic analytics page or skip advanced search filters).

**3. Development Workflow & Tools**

We agreed on how to collaborate effectively using our tools:

- **Git Repository:** The project repository on GitHub is set up with all members invited. We decided on a **Git branching model**: a main (or master) branch that remains stable (we won’t push broken code to main) and each feature or component will be developed on separate branches (e.g., frontend-reader-page, backend-auth, etc.).

After a feature is complete or at a stable point, a Pull Request will be opened to merge into main, and at least one other member should review the code (this ensures knowledge sharing and catches issues early). Given the tight schedule, code reviews will be quick but at least involve running the code and checking for obvious issues.

- **Communication:** Discord remains our primary communication tool for quick questions and sharing progress updates daily. We also plan short stand-up calls every 2-3 days to sync up (or daily quick text updates on what was done and blockers).

All members confirmed they have their development environment ready (Node.js, a code editor, etc.). We also confirmed connectivity to the centralized database: we chose to use a cloud PostgreSQL instance (e.g., Supabase or Railway) so everyone can point their local backend to the same database for easier integration testing. Le Anh Duy has created a Supabase project and will share the connection URL and credentials privately. We must be careful with credentials (we'll put them in an .env file not committed to Git).

**4. Coding Standards & Practices**

We spent a few minutes to agree on coding conventions so our codebase remains uniform:

- We will use **JavaScript/TypeScript**? We realized we should decide whether to use TypeScript for the Node and Next.js app. This is slightly extra work, but given we have design definitions, TypeScript will help catch errors and define interfaces (especially for data models). All members are somewhat familiar with TS, and it aligns with writing more robust code. We noted to configure our project for TS from the start.
- **Linting/Formatting:** We will use ESLint with a common style guide (perhaps Airbnb or Standard) and Prettier for automatic code formatting. This ensures things like indentations, quotes, etc., are consistent. Huy volunteered to set up ESLint/Prettier configs at the start of the repository.
- **Naming conventions:** Use clear, descriptive names for variables and functions. Database tables/fields use snake_case by convention (common in SQL), and in code we might use camelCase for variables. We will document any mapping as needed. We also decided to follow RESTful naming for API endpoints (e.g., GET /api/mangas, POST /api/chapters).
- **Commenting and Documentation:** We encourage documenting functions and complex logic with comments. Also, to maintain a simple Markdown README.md in

the repo to help any new developer (or for our instructor) understand how to run the project. The README will include how to set up environment variables, how to migrate the database, etc.

- **Reusability:** We will create reusable components/functions where possible. For example, frontend: a generic Modal component, or form input components. Backend: middleware for authentication (to protect routes), generic error handler, etc.
- **Testing during development:** We agreed each developer writes basic tests or at least test harnesses for their module. For instance, Duy will test the OCR function with a sample image to verify it works, Huy will test that an upload creates the correct DB entries, etc. We might use a tool like Postman to manually test API endpoints as we develop them.

By adhering to these standards, we hope to reduce integration issues when we merge our code.

**5. Implementation Timeline & Milestones**

We mapped out a timeline for the implementation phase. We have roughly from Dec 15 until likely the project final deadline (which might be end of December or early January depending on course schedule – possibly early January if there's a final presentation). However, we want to finish coding by end of December to allow time for thorough testing and any buffer.

Key milestones set:

- **Milestone 1: Core Backend Functionalities Ready – (Target: Dec 5, 2025).** By this date, we aim to have the core back-end endpoints implemented and the database schema in place. This includes user auth, uploading chapters, fetching chapters, approving chapters, and translation integration. Essentially, a reader should be able to browse and read a translated chapter by this point, and an uploader/admin should be able to perform an upload and approval cycle (though perhaps via API calls or rudimentary UI).
- **Milestone 2: Core Frontend Pages Functional – (Target: Dec 5, 2025).** The goal is that the main user flows on the frontend are working end-to-end with the backend. A user can register, log in, navigate the site, read a manga with translation, comment, and an admin can log in to their dashboard. The UI might still be basic/styled minimally at this point, but functionally everything connects. This assumes that by

then, the front-end team will catch up after the backend is mostly done in the earlier part of the week.

- **Milestone 3: Feature Completion & Polishing – (Target: Dec 10, 2025)** we handle remaining features and polish: e.g., payment flow (if not done yet), favorites list, analytics page for admin, UI improvements (better styling, responsiveness checks), and any bug fixes discovered. We also aim to add more validation (e.g., form validation on frontend, error messages from backend) to improve the user experience.
- **Milestone 4: Internal Testing & Freeze – (Target: Dec 15, 2025).** All implementation should be completed by this time. We will run all our test cases from the Test Plan, fix any bugs found, and prepare for final demonstration. We call this a code freeze date (only bug fixes after this, no new features).

**6. Testing Strategy during Implementation**

While a detailed test plan exists (and formal testing phase comes after coding), we discussed how to integrate testing into our development process:

- **Unit Tests:** We will write unit tests for critical functions where possible. For example, functions like password hashing/verification, translation text processing, and any complex logic (like role-checking middleware or payment calculation) could have small unit tests. We decided to use Jest (since it works well with Node and even with Next for some functions) for any unit tests. However, given time constraints, unit tests are secondary to actually building features – so our approach is pragmatic: write tests if it doesn't slow down development too much, otherwise prioritize manual testing.
- **Integration Tests:** Closer to completion, we will do end-to-end integration tests manually. For instance:
    - Create a new user, log in, upload a chapter, have an admin approve it, then as the user see it appear and attempt to purchase if it's paid, etc. This kind of scenario will be tested by the team manually to ensure all parts work together.
    - We might use our test cases document to guide these manual tests.
- **Bug Tracking:** If during development or testing a bug is found, we will log it (likely as an issue in our repository or on the project board) and assign someone to fix it. This ensures we keep track of known issues and verify their resolution.

- **Test Data:** We will set up some test data in the database to use for testing flows (for example, one admin user, a couple of normal users, some sample manga with one or two pages, etc.). This will save time repeatedly creating data. We can reset the database as needed since it's just our test environment.
- **Responsibility:** We didn't assign a separate "tester" role; instead each feature developer will at minimum test their own feature. Additionally, for cross-check, another team member will test it too (for example, once Huy implements uploading, Tien as a front-end dev will test uploading through the UI and report any backend issues, etc.). This echoes the principle of peer review and ensures two sets of eyes on each functionality.

We also briefly considered automated end-to-end testing tools (like Selenium or Puppeteer) but agreed we likely don't have time to set those up. We'll rely on manual testing and possibly recording a demo video as proof of a working system.

**7. Deployment & Environment Setup**

Although deployment (moving to production environment) might be part of a later phase (Deployment phase), we need to plan how we will host or run the system for demonstration:

- **Local vs Online Demo:** If possible, we'd like to deploy the application to a cloud platform for ease of demo (so that instructors can access it live). Potential options:
  - Deploy frontend on Vercel (as Next.js is easily deployable there).
  - Deploy backend on a free Node hosting (Heroku (no free tier now), or maybe Render.com or Railway).
  - Database is already cloud (Supabase), which is good.
    We will attempt to deploy, but as a fallback, we can run the system locally on a single machine for the presentation.
- We need to ensure environment variables (API keys, DB credentials) are configured appropriately in the deployment environment. We will not expose secrets in the code repository. For now, each has .env files for local dev. For deployment, we'll use the platform's secret storage.
- We also planned to test the app in an environment similar to production – for example, accessing it over the internet if deployed or at least emulating multiple users (maybe one in incognito as admin, one as user to test concurrently).

- **Documentation of Deployment:** We'll record the steps and config needed (this can also feed into our final report or user manual, e.g., how to run the system).

**8. Risk Assessment & Contingency**

We identified a few potential risks in the implementation phase and our plans to mitigate them:

- **Risk: Third-Party OCR/Translation issues** – If the chosen OCR or translation service is unreliable, slow, or hard to integrate, this could block the translation feature. *Mitigation:* Use a simple OCR library (like Tesseract OCR) that we can run locally to get text, and use a translation API for text (like Google Translate via HTTP). If that fails, we could simplify the feature to maybe just display text in one alternate language that we pre-translate (not ideal, but a fallback). We allocated this task early (Duy will spike on this early in implementation) to ensure we have a working approach or switch early if needed.
- **Risk: Time crunch / feature creep** – There is a lot to implement and limited time. If we run behind, some less critical features might not be completed. *Mitigation:* Prioritize core features (reading, uploading, approving, translating). If needed, we can cut or simplify: e.g., skip the "statistics dashboard" or implement only a very minimal version; reduce the payment integration to a simple flag without actual processing (just assume purchase on button click for demo purposes); not implement advanced search filters (just a simple search by keyword). The instructor is likely expecting the main functionalities, so we will focus on those first.
- **Risk: Integration bugs** – Often, issues arise when connecting front-end to back-end. *Mitigation:* Frequent integration testing as we go, not leaving it all to the end. We set the milestone for core integration by Dec 24 to catch bugs early. Also, by working in parallel with good communication (frontenders and backenders testing together), we hope to smooth out mismatches (like API request/response formats). The use of TypeScript can help ensure we share data models between backend and frontend (maybe define interfaces for data objects and reuse them) so that they stay consistent.
- **Risk: Team member availability** – The holiday season and possible exams or other coursework could reduce availability. *Mitigation:* We have a clear plan and task assignments; each person knows what to do, so if someone gets busy unexpectedly, we'll communicate and possibly help each other. Because we have overlapping knowledge (everyone has at least some idea of each part), if needed one can step in

to assist or take over a task. We also scheduled intermediate check-ins to surface any delays quickly.

- **Risk: Deployment challenges** – Sometimes deploying a full-stack app can be tricky (CORS issues, environment differences). *Mitigation:* Start deployment process early (as noted) and be ready to troubleshoot. If deployment fails, plan B is running locally for demo, which we know will work since we develop that way. We will also prepare a recorded demo as backup in case of any live demo issues.

Recording these risks made us more vigilant. We concluded that while the project is ambitious, careful planning and teamwork should allow us to complete it successfully.