

CHAPTER 4 – RESULTS AND DISCUSSIONS

4.1 Initial Instruction

The initial instruction serves as the foundation for all AI-assisted development activities throughout the MBKM system project. It represents the context and requirement prompt initially provided to ChatGPT and repeatedly referenced throughout the conversation within the same project thread.

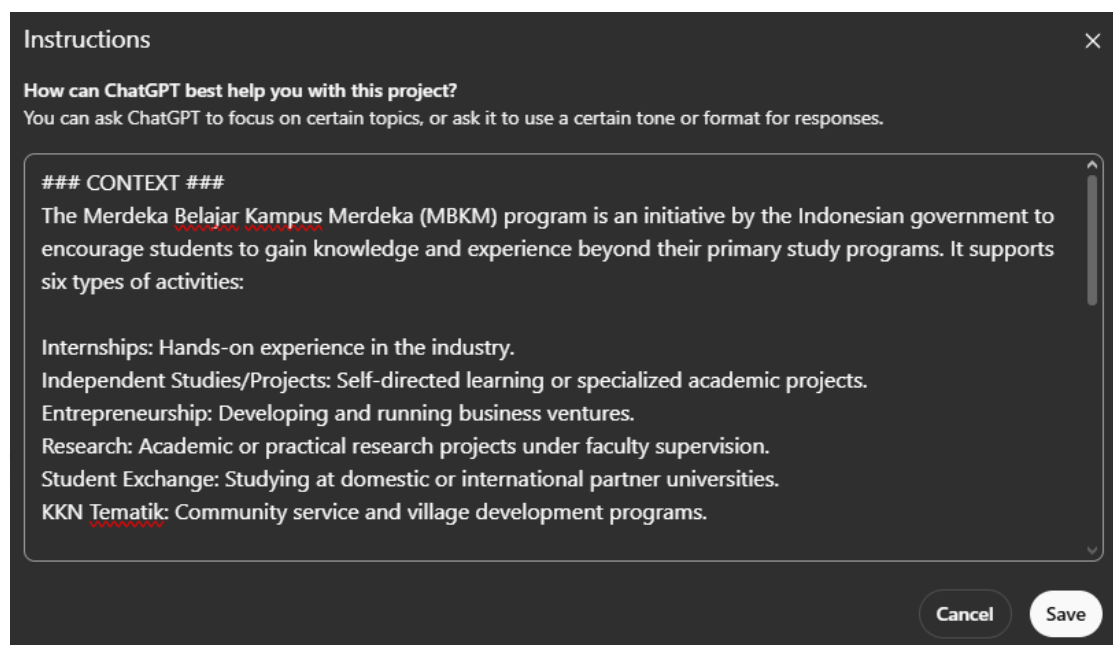


Figure 4.1 Initial Instructions for ChatGPT

Figure 4.1 shows the instruction written in natural language. It includes the following key components:

High-level project context: An overview of the MBKM program and its implementation at Swiss German University (SGU).

Project objective: A statement defining the goal of the MBKM portal development and the role of AI assistance.

User roles: A clear definition of the actors involved in the system, including students, staff, advisors, and supervisors.

Development approach: A summary of the Feature-Driven Development (FDD) methodology adopted for the project.

ChatGPT's role: A detailed outline of how ChatGPT is expected to support each phase of the Software Development Life Cycle (SDLC), including requirement engineering, design, implementation, testing, and documentation.

4.2 Requirement Elicitation

The requirement elicitation phase focused on gathering insights and detailed requirements for the MBKM system at Swiss German University (SGU). This phase aim to define the scope, functional needs, and constraints of the system, forming the foundation for the subsequent stages of development.

4.2.1 Initial preparation

Before conducting stakeholder interviews, ChatGPT was utilized for its knowledge retrieval capabilities to suggest interview questions based on industry best practices. Figure shows the prompt used. The following key areas were identified for discussion:

Strategic goals for the MBKM system.

Challenges in the current process for managing MBKM activities.

Integration requirements with existing systems.

Key features required by stakeholders, including must-have functionalities.

Compliance requirements with government regulations.

Figure 4.2 shows the output, ChatGPT suggested 28 questions for the Vice Rector, focusing on the strategic objectives of the MBKM program, the scope of activities, and the student journey across systems. For the interview with the Head of IT, ChatGPT proposed 46 questions addressing IT infrastructure, integration with existing systems, as well as functional and non-functional requirements. ChatGPT generated 41 questions for the MBKM staff, targeting the operational workflow, challenges in the current process, and desired system features. These questions were refined by the author to ensure their relevance and eliminate redundancy.

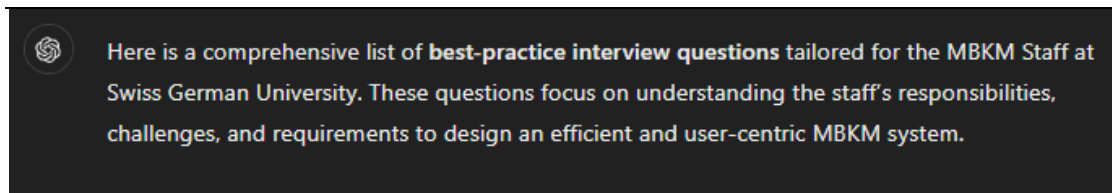


Figure 4.2 ChatGPT output based on the prompt

4.2.2 Stakeholder Interviews

Interviews were conducted with key stakeholders to gather insights. The Vice Rector emphasized the importance of ensuring a smooth student journey across different systems, minimizing administrative friction, and providing a seamless experience. They highlighted the need to align the MBKM system with SGU's strategic goals and student-focused approach while scoping the MBKM activities to five key options that SGU would implement, prioritizing practicality and alignment with available resources.

The Head of IT provided insights into the existing IT infrastructure and noted resource constraints. They discussed potential integration points with SGU's current university management system to avoid the creation of data silos. Additionally, they stressed the importance of implementing robust data security measures and ensuring the system's scalability to support future growth.

The MBKM staff shared operational challenges, including difficulties in tracking student progress and ensuring timely approvals. They suggested features such as real-time progress tracking, automated notifications, and integrated document management to address these issues and enhance the system's efficiency and usability.

4.2.3 Input Pre-processing using ChatGPT

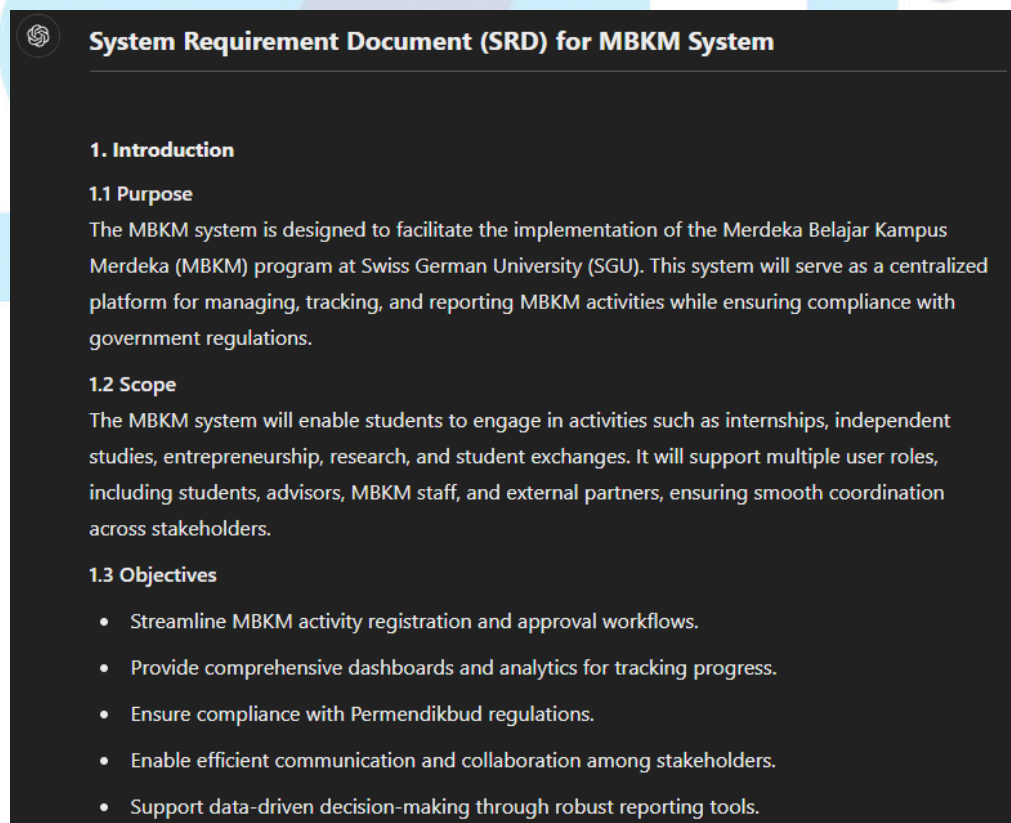
The interview data was processed using ChatGPT to generate a Software Requirement Document (SRD) using prompt shown in Figure 4.3. Structured prompts were crafted to guide ChatGPT in analyzing the interview data effectively. The transcribed interview data was formatted and fed into ChatGPT alongside the structured prompts. ChatGPT generated an initial draft of the requirements by identifying key system features and any gaps or ambiguities in the provided data

```
### TASK ###
1. Analyze the provided stakeholder interview transcripts thoroughly
to extract relevant system requirements.
2. Generate an initial System Requirement Document (SRD) that
adheres to industry best practices and standards.
3. Use professional language and organize the document in a clear,
logical format for easy reference.

### OUTPUT EXPECTATION ###
A detailed and structured System Requirement Document (SRD) that
can be used as the foundation for MBKM system design and
development.
```

Figure 4.3 Prompt for requirement analysis

Figure 4.4 shows the initial output from ChatGPT. It was reviewed by author to ensure alignment with SGU's operational needs. Any gaps or inaccuracies were addressed through additional prompts and iterations, or additional questions to the stakeholders. This feedback loop allowed for continuous improvement of the SRD.



System Requirement Document (SRD) for MBKM System

1. Introduction

1.1 Purpose

The MBKM system is designed to facilitate the implementation of the Merdeka Belajar Kampus Merdeka (MBKM) program at Swiss German University (SGU). This system will serve as a centralized platform for managing, tracking, and reporting MBKM activities while ensuring compliance with government regulations.

1.2 Scope

The MBKM system will enable students to engage in activities such as internships, independent studies, entrepreneurship, research, and student exchanges. It will support multiple user roles, including students, advisors, MBKM staff, and external partners, ensuring smooth coordination across stakeholders.

1.3 Objectives

- Streamline MBKM activity registration and approval workflows.
- Provide comprehensive dashboards and analytics for tracking progress.
- Ensure compliance with Permendikbud regulations.
- Enable efficient communication and collaboration among stakeholders.
- Support data-driven decision-making through robust reporting tools.

Figure 4.4 SRD generated by ChatGPT

4.3 Build Overall Model Phase

The "Build Overall Model" phase focused on defining the high-level architecture and structure of the MBKM system. The model represents both the domain knowledge and the high-level architecture of the MBKM system, ensuring that all team members, including AI, share a unified understanding of the system before decomposing it into features.

4.3.1 Collaborative Modelling Process

The modeling process was conducted through a collaborative approach between the author and ChatGPT. Drawing from the structured requirements obtained during the elicitation phase, the author prompted ChatGPT to:

Generate high level architecture
Generate use case diagrams representing key user interactions.
Model user workflows for each MBKM activity.
Identify domain entities and their relationships.
Draft initial database entity-relationship diagrams based on registration and tracking requirements.

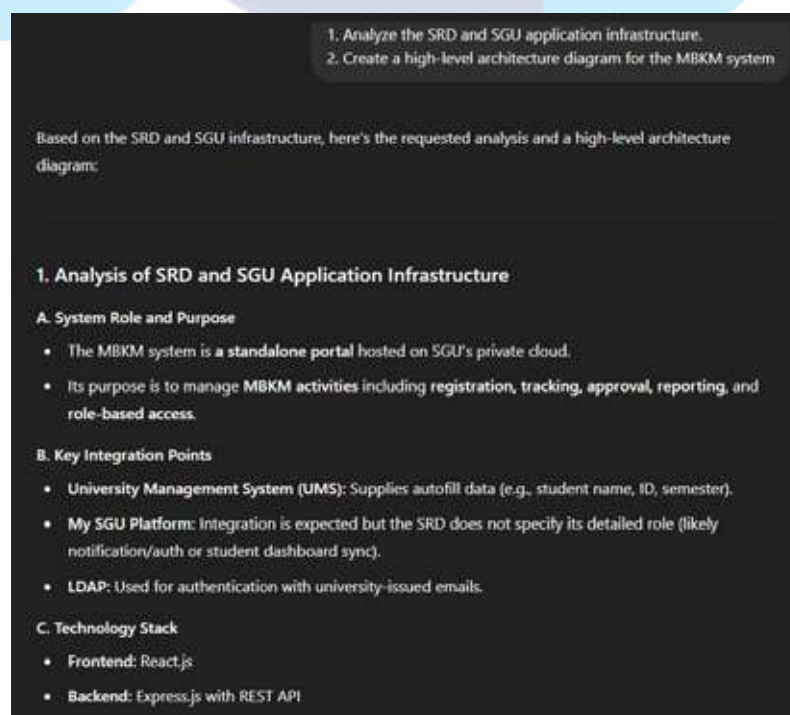


Figure 4.5 Snippet of modelling process

Figure 4.5 Shows snippet of the modelling process. ChatGPT then helped visualize these models into UML diagrams using PlantUML (Figure 4.6), which were included in the System Requirement Document (SRD) to ensure clarity and traceability for all stakeholders.

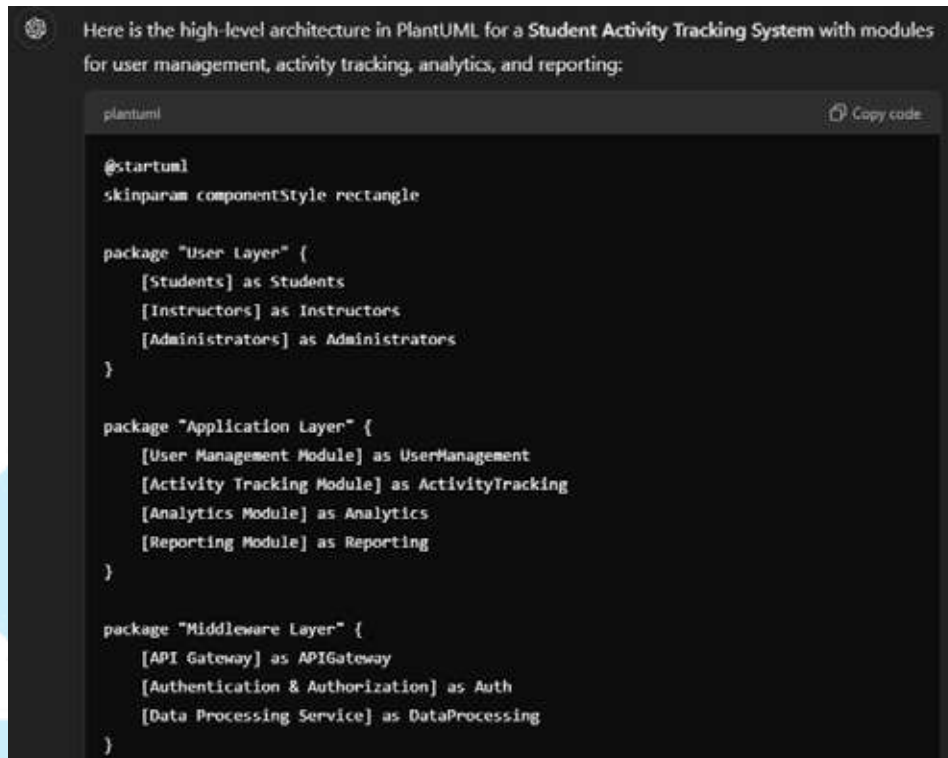


Figure 4.6 ChatGPT generated PlantUML script for architecture

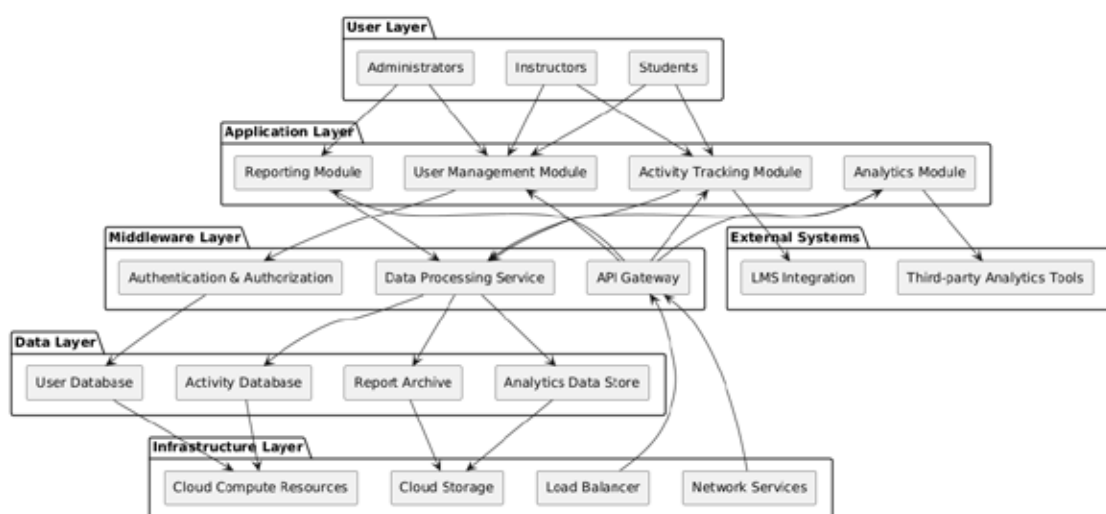


Figure 4.7 Initial component diagram generated using PlantUML

4.3.2 UML Artifact Generation

UML artifacts was generated during the modeling phase. Each diagram served a specific purpose in visualizing system behavior, structure, and data relationships.

The following UML artifacts were identified and produced:

Use Case Diagrams: Use case diagrams captured the core interactions between system actors and functionalities. These diagrams illustrated how students, MBKM staff, advisors, supervisors, and partners interact with the MBKM portal. The use cases were grouped by subsystems such as registration, activity tracking, and reporting.

Activity Diagrams: Activity diagrams were developed to represent workflow variations for each MBKM activity. These diagrams visualized the flow of actions, decisions, and approvals.

Class Diagram: The domain model represented conceptual entities and their relationships, including Student, Registration, MBKM Activity, Advisor, Supervisor, and Logbook. ChatGPT helped generate both the entity structure and the relationship using PlantUML syntax.

Entity-Relationship Diagram (ERD): A database-focused version of the domain model was also generated to map the logical structure of the backend. Primary keys, foreign keys, and join relationships were included based on the MBKM workflow logic.

Component Diagram: A high-level component diagram was created to visualize how frontend, backend, database, and external systems (e.g., university management system) interact. It included deployment boundaries and interfaces between modules such as registration, activity tracking, and reporting.

The integration of ChatGPT into the modeling process provided substantial support in quickly transforming textual requirements into structured UML diagrams. The AI-generated outputs often served as an effective starting point. However, a notable limitation observed during this phase was that the UML diagrams produced by ChatGPT, were often lacked visual and logical organization. Common issues included misalignment of connectors, resulting in cluttered diagrams and lack of layout control

(Figure 4.8). Although iterative prompting was used to refine the layout, it was found that repeated iterations did not consistently lead to improved diagram structure—in some cases, the outputs became more disorganized. As a result, it was more efficient for the author to manually adjust and clean up the diagrams after a few initial iterations, using the AI-generated version as a structural foundation rather than a final product. Figure 4.9 shows the final use case diagram included in the SRD.

Despite these limitations, the AI-generated UML artifacts effectively accelerated the author’s conceptualization of the system. While not final-ready, these initial drafts provided a structured starting point that eased the modelling process and reduced cognitive load. The author could then refine and organize the diagrams to meet documentation standards and improve clarity for stakeholder communication. Figure 4.10 Shows final diagram included in the SRD.

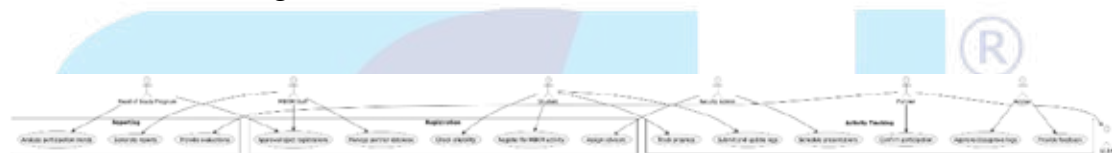


Figure 4.8 ChatGPT initial output for Use Case diagram

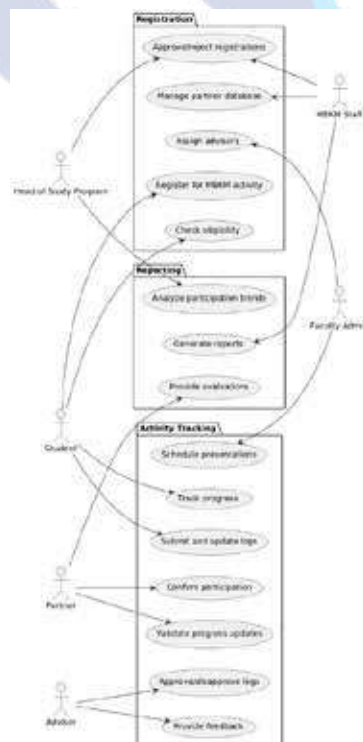


Figure 4.9 Use Case diagram after several iteration

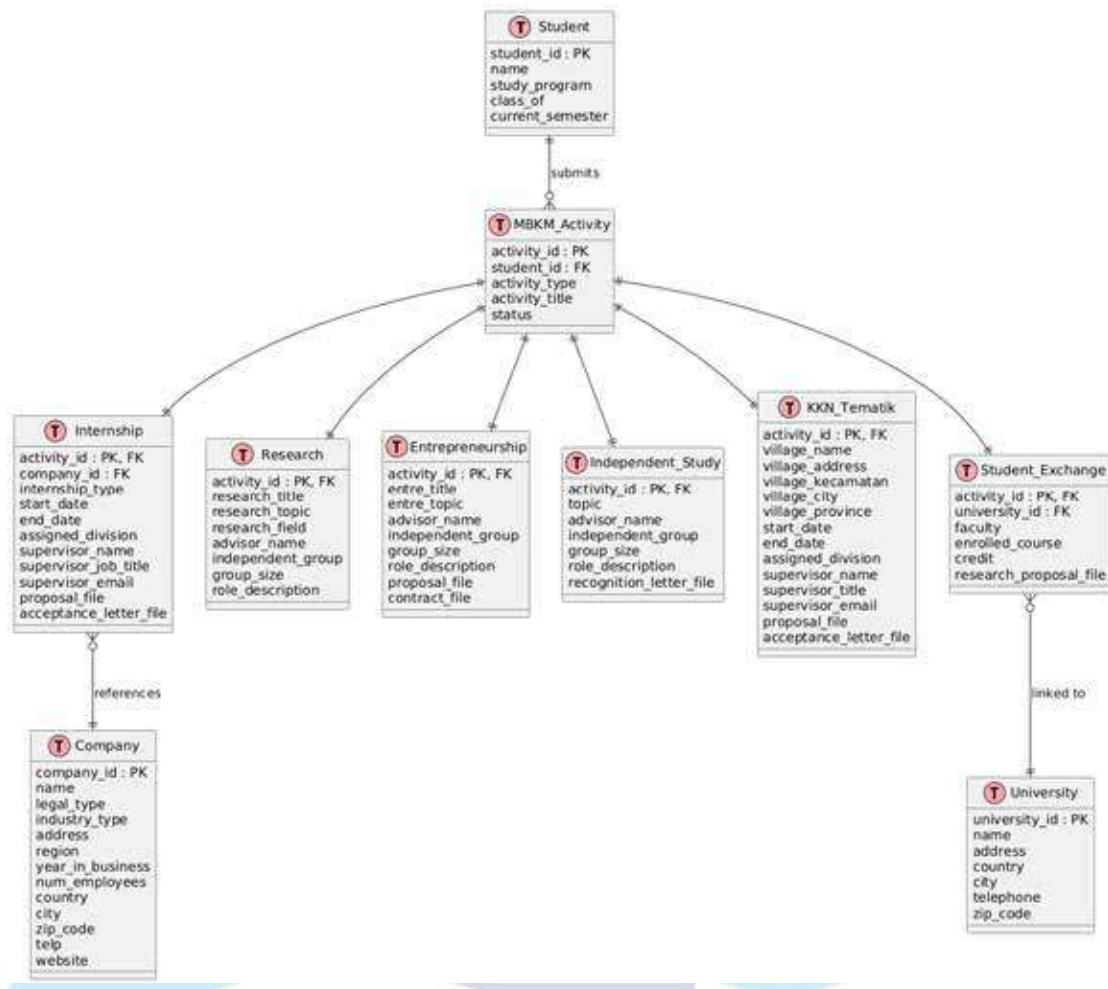


Figure 4.10 Finalized ERD for MBKM registration

4.4 Build Feature List

The Build Feature List phase served to translate the previously requirements into a clear, structured set of system features. This process was conducted iteratively through collaboration between stakeholders, the author, and ChatGPT.

4.4.1 ChatGPT-Assisted Feature Breakdown

ChatGPT was leveraged to assist in analyze requirements into detailed, actionable features. Structured prompts were used to guide the LLM, with prompt: "Analyze the requirement into key features and sub-features, considering user roles (students, staff, administrators) and system functionality." Figure 4.11 shows the ChatGPT output.

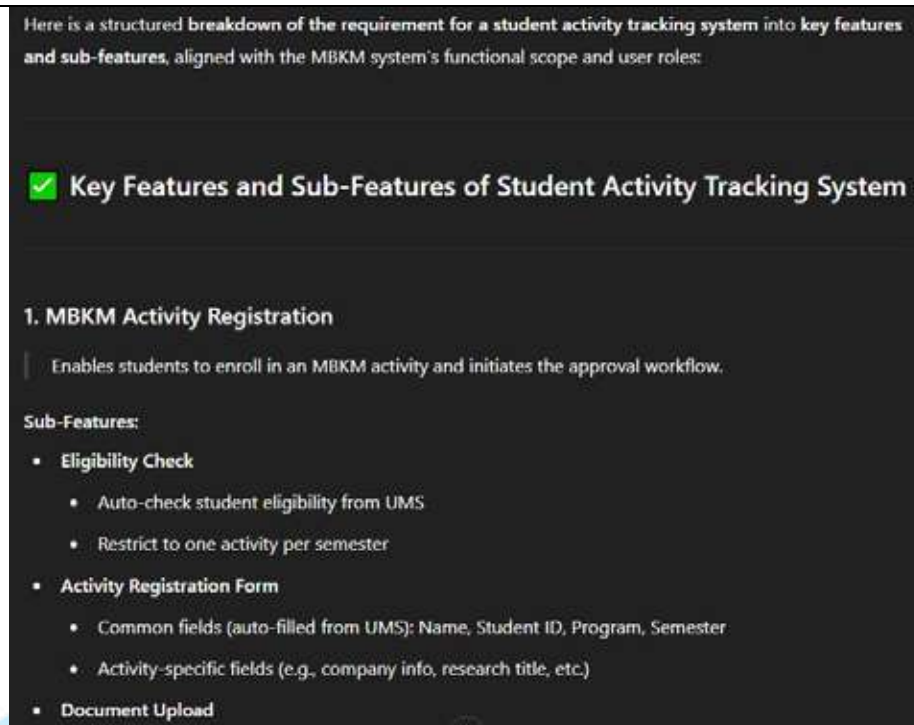


Figure 4.11 ChatGPT out in feature list phase

The outputs from ChatGPT include suggestions for categorizing features into core modules. Features suggested include:

User Management Features: Role-based access control, secure login and password recovery, and user profile management for students, staff, and external partners.

Registration Features: Activity-specific registration forms (e.g., internship, research, student exchange), multi-level approval workflows (student → MBKM staff → Head of Study Program), eligibility checks, and registration revision mechanisms upon rejection.

Activity Tracking Features: Real-time progress tracking dashboard, submission and approval workflows for MBKM activities, and integration with partner systems for activity verification.

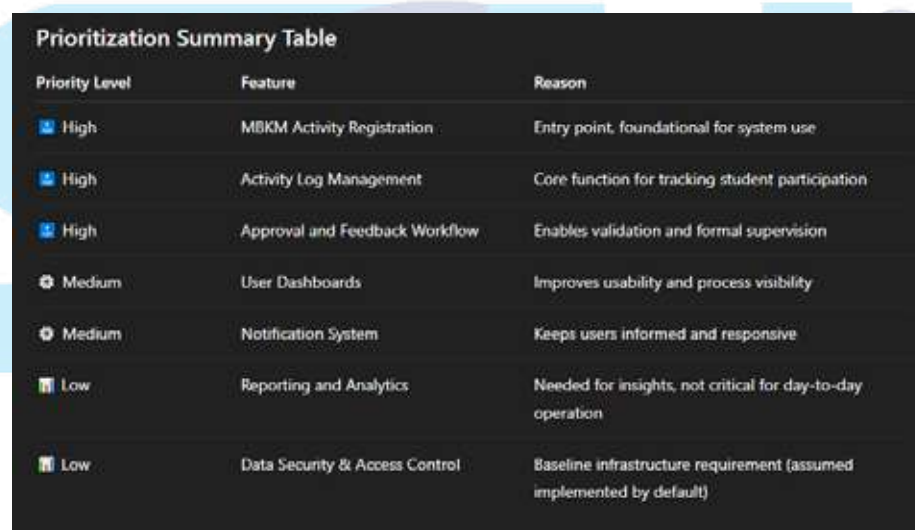
Analytics and Reporting Features: Customizable dashboards for KPIs (e.g., participation rates, completion rates), Automated generation of compliance reports, Exportable data formats for further analysis.

Notification Features: Automated email and SMS reminders for deadlines, Push notifications for approvals and updates, Alerts for anomalies, such as incomplete activity data.

Integration Features: APIs for data exchange with existing SGU systems and middleware for document management and student information systems.

4.4.2 Prioritization and Refinement

After identifying the list of features, the next step was to prioritize them based on their importance to system functionality, stakeholder needs, and development feasibility. Prioritization was conducted collaboratively between the author, stakeholders, and the SGU IT team. Figure 4.12 shows ChatGPT assisted in categorizing the features by suggesting logical groupings and identifying dependencies between modules.



Priority Level	Feature	Reason
High	MBKM Activity Registration	Entry point, foundational for system use
High	Activity Log Management	Core function for tracking student participation
High	Approval and Feedback Workflow	Enables validation and formal supervision
Medium	User Dashboards	Improves usability and process visibility
Medium	Notification System	Keeps users informed and responsive
Low	Reporting and Analytics	Needed for insights, not critical for day-to-day operation
Low	Data Security & Access Control	Baseline infrastructure requirement (assumed implemented by default)

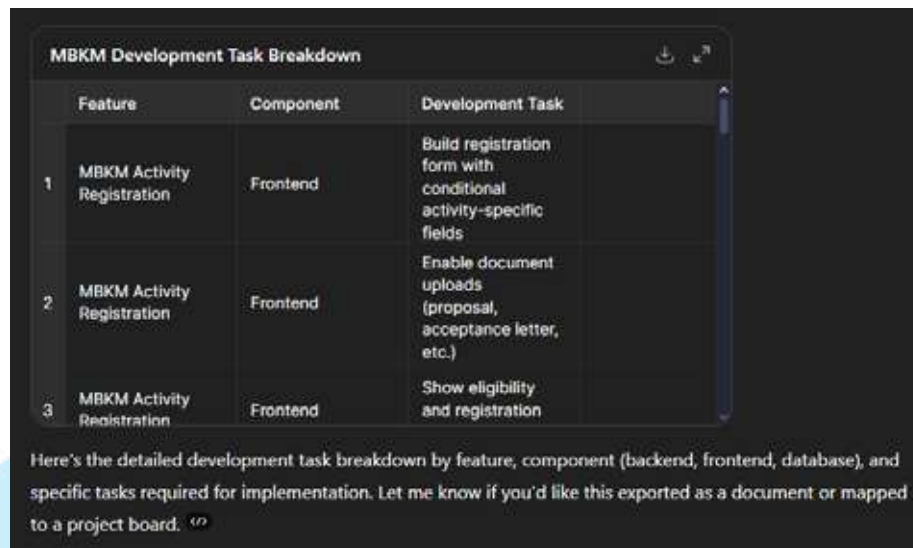
Figure 4.12 ChatGPT prioritization recommendation

Each item in the high priority category was deemed essential to the system’s minimum viable functionality. Features in the medium category were important for enhancing usability and administrative efficiency, while low features were classified as enhancements that could be considered for future releases or baseline infrastructure.

4.5 Plan by Feature

The Plan by Feature phase translated the prioritized feature list into a structured development plan. This phase involved decomposing each feature into implementable

units, assigning them to specific development cycles, and determining the technical tasks required for realization. ChatGPT was used as a co-creator to break down the features into backend logic, frontend interfaces, and database components, supporting the team in defining clear development scopes. Figure 4.13 shows the ChatGPT output.



	Feature	Component	Development Task
1	MBKM Activity Registration	Frontend	Build registration form with conditional activity-specific fields
2	MBKM Activity Registration	Frontend	Enable document uploads (proposal, acceptance letter, etc.)
3	MBKM Activity Registration	Frontend	Show eligibility and registration

Here's the detailed development task breakdown by feature, component (backend, frontend, database), and specific tasks required for implementation. Let me know if you'd like this exported as a document or mapped to a project board. ¹²

Figure 4.13 ChatGPT output for development task breakdown

Based on the prioritization from Section 4.4.2, features were assigned to specific development sprints. Sprint planning ensured that foundational features (e.g., registration) were implemented first, followed by features that depended on them (e.g., logging and reporting).

ChatGPT significantly enhanced the planning phase by accelerating the task decomposition process and suggesting backend/frontend/database alignment per feature. The AI-assisted planning substantially reduced the time and cognitive effort required in this phase.

4.6 Design by Feature

The "Design by Feature" phase translated system requirements into visual interface designs, preparing each feature for implementation. This phase focused on how each feature would be experienced by users, particularly in terms of layout, navigation, and usability. ChatGPT was used to generate wireframe suggestions based on the uploaded

Software Requirement Document (SRD), and the UI was then designed in Figma based on those suggestions.

4.6.1 Wireframe Suggestions via ChatGPT

To initiate the design process, the author uploaded the finalized Software Requirement Document (SRD) to ChatGPT and provided prompts to request wireframe layout suggestions for key features and user roles. Figure 4.14 shows ChatGPT output.

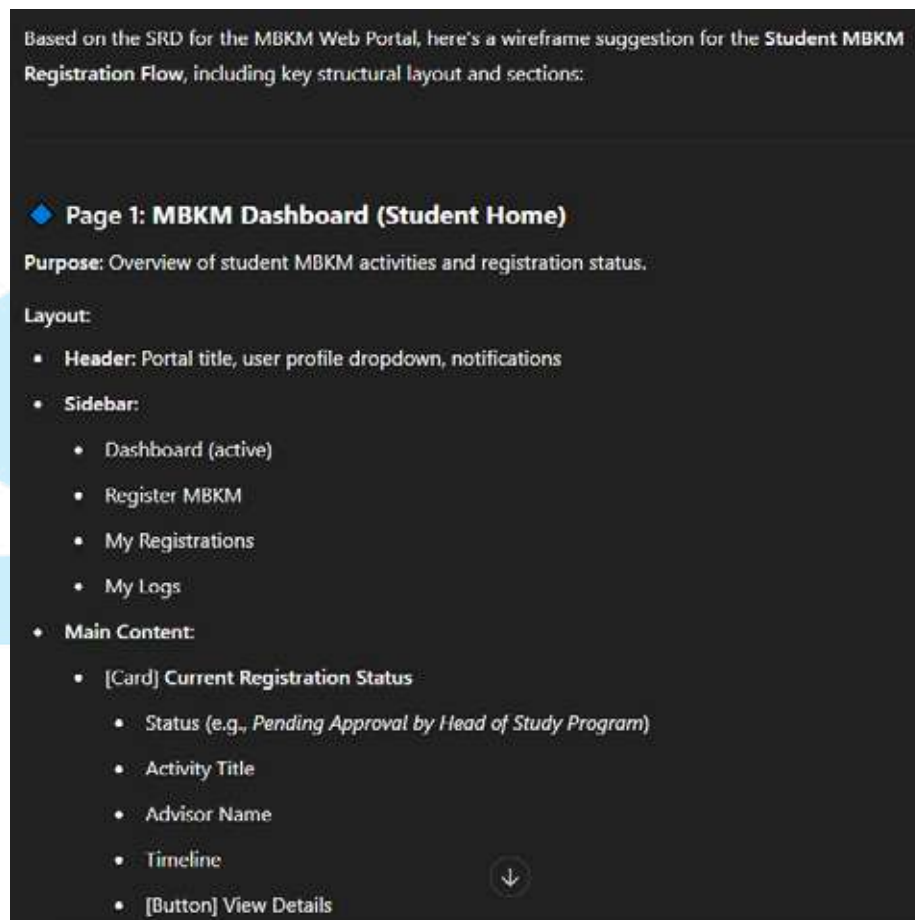


Figure 4.14 ChatGPT text-based wireframe output

These suggestions were provided in a text-based format describing layout zones and content areas. The author initially attempted to visualize these wireframes using PlantUML, generating structured diagrams based on ChatGPT's descriptions. Figure 4.15 shows the script and its visualization. However, the results were not visually satisfactory. As a result, the author decided to use the text-based wireframes directly as a reference to build the high-fidelity interface prototypes in Figma.

Here is a **PlantUML wireframe script** to visualize the **Student MBKM Registration Flow** using PlantUML's wireframe DSL:

```
plantuml
@startuml
skinparam handwritten true
skinparam monochrome true
skinparam componentStyle rectangle

title Student MBKM Registration Flow Wireframe

' Page 1: Dashboard
frame "Dashboard" {
    rectangle "Header: SGU MBKM Portal | Profile | Notifications" as Header
    rectangle "Sidebar:\n- Dashboard\n- Register MBKM\n- My Registrations\n- My Logs" as Sidebar
    rectangle "Main Content" {
        rectangle "Card: Current Registration Status\n- Status\n- Activity Title\n- Advisor Name\n- Ti"
        rectangle "Section: Start New Registration\n[Register for New MBKM Activity]" as NewReg
    }
}
}
```



Figure 4.15 ChatGPT PlantUML script and the visualization.

4.6.2 UI Design in Figma

Using the structural wireframe descriptions from ChatGPT, the author created UI designs in Figma. These Figma designs followed Swiss German University's visual branding and usability standards. Key screens designed included:

Dashboards for Each User Role; Student Dashboard displays MBKM activity progress, submission reminders, and alert notifications (Figure 4.16), Advisor/Supervisor Dashboard shows pending log approvals, student lists, and communication access, and MBKM Staff Dashboard shows centralized control panel for activity tracking and partner management.

MBKM Registration Workflows: Dedicated screens for registering to each MBKM activity (Internship, Research, Independent Study, Student Exchange,

and Entrepreneurship), with dynamic input fields and logic based on activity requirements (Figure 4.17).

Log Submission Workflows: Role-specific log submission forms for students, adapted to each MBKM activity, and integrated feedback loop with submission status indicators and file attachments.

Partner Management Interfaces: Internship and University Partner Management: Interface for adding, editing, and assigning industry partners.

Advisor and Supervisor Interfaces: Log review screens with filters and batch approval features and detailed student profiles with log histories and progress tracking.

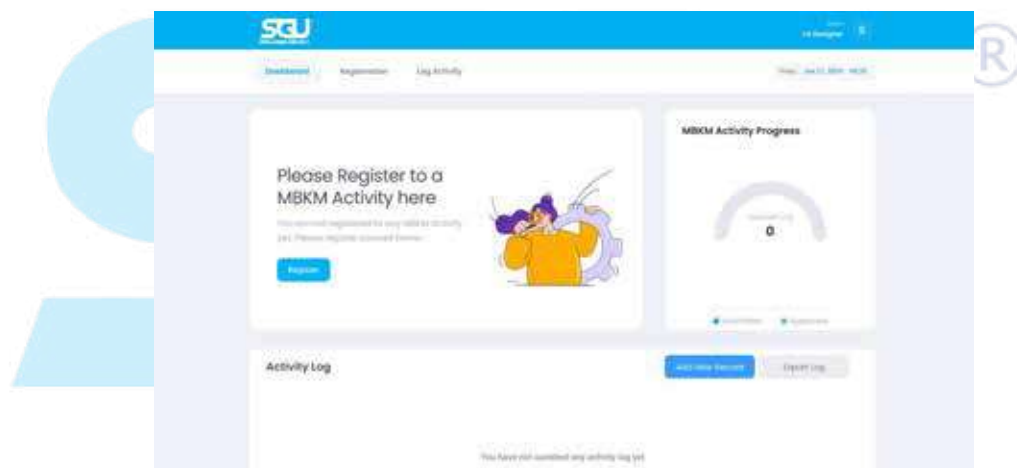


Figure 4.16 Student Dashboard design

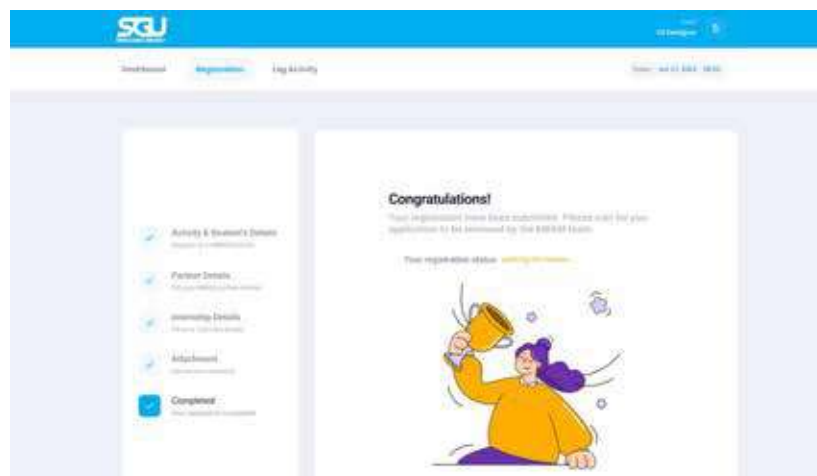


Figure 4.17 Student MBKM Registration design

4.6.3 Reflection on LLM-Assisted UI Design

ChatGPT contributed significantly to this phase by accelerating layout planning and reducing ambiguity in user interface expectations. While it could not create visual outputs directly, its ability to suggest layout logic and structure enabled the author to move quickly into design without starting from scratch.

4.7 Build by Feature

In this phase, the prioritized features were implemented by the SGU IST development team using the Metronic 8 admin template as the design foundation. Development was carried out using a component-based approach, with each feature built incrementally according to the previous phase. ChatGPT was used to assist developers in improving logic, optimizing layout structure, and debugging throughout the process. Notable use cases include:

Use Case 1: Table Layout Adjustment (Figure 4.18): A developer wanted to make the "Task" column more prominent in a table. ChatGPT's Suggestion:

- Swap the positions of "Status" and "Task" to move "Task" closer to "Date".
- Add width constraints to the "Task" column using `min-w-200px` or `w-100`.

✓ Solusi:

1. Tukar posisi kolom "Status" dengan "Task", agar "Task" lebih dekat ke "Date".

2. Beri class `w-100` atau `min-w-200px` ke kolom "Task" supaya dia jadi paling lebar.

💡 Contoh kode revisi:

jsx

Copy

Edit

```
<tr className="fw-bold text-muted">
  <th className="max-w-10px">No</th>
  <th className="min-w-50px">Date</th>
  <th className="min-w-200px">Task</th> {/* lebar lebih besar */}
  <th className="w-100px text-center">Status</th>
  <th className="w-150px text-center">Actions</th>
</tr>
```

Figure 4.18 Table Layout Adjustment by ChatGPT

Dennis Reynaldo

Use Case 2: Role Routing Logic Validation (Figure 4.19). The developer asked ChatGPT to validate the correctness of the following condition: `if (role === "Staff" || (role === "Supervisor" && password)) {}`. ChatGPT confirmed the logic was valid but suggested more readable alternatives:

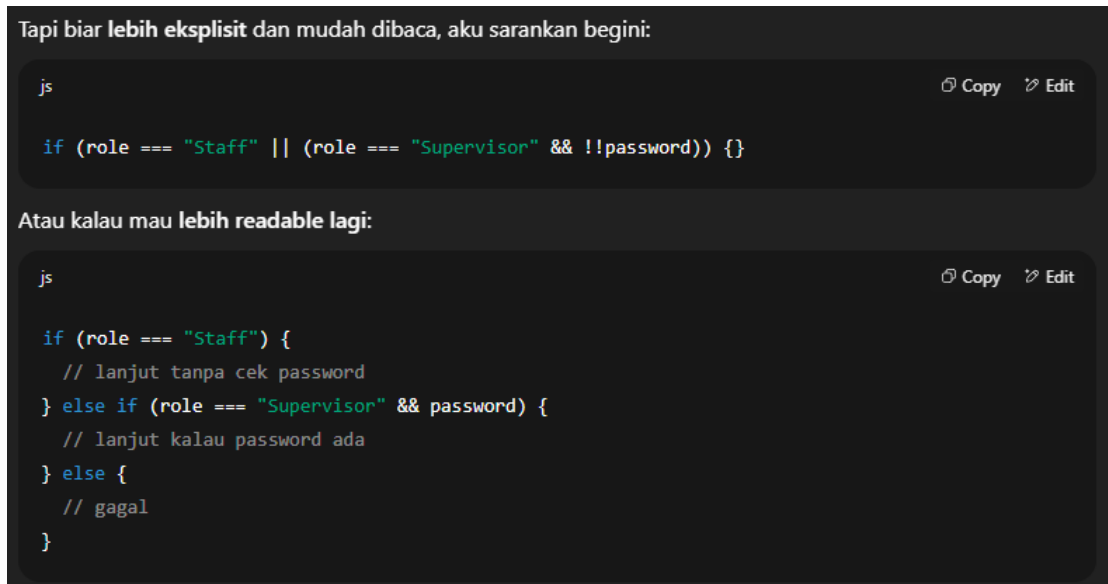


Figure 4.19 ChatGPT Role Routing Logic Validation

Use Case 3: Tooltip Not Visible (Figure 4.20). The developer asked ChatGPT why tooltip is not visible in his code, the developer pasted his code snippet for ChatGPT to analyze and revise. ChatGPT solution:

- Always render a real DOM element (e.g., `<span>`) inside `OverlayTrigger`.
- Always render `<Popover>`, but conditionally render its content, not the component itself.

Jika `status === false`, maka hasilnya `false`, dan `Popover` tidak akan muncul sama sekali. Tapi bukan itu saja — kamu juga pakai fragment (`<>...</>`) sebagai anak `OverlayTrigger`, yang tidak menghasilkan `DOM node` aktual, sehingga trigger klik-nya bisa tidak bekerja dengan benar.

✓ **Solusi:**

1. Pastikan ada elemen `DOM` nyata sebagai anak `OverlayTrigger` (misalnya `<span>`).
2. Selalu render `Popover`, tapi di dalam `Popover` bisa kasih kondisi kontennya.

🔧 **Perbaikan saran:**

```
jsx
const RegisterStatus = ({ data, status = true }) => {
  const renderPopover = (
    <Popover className="p-3">
      <Popover.Body className="p-0">
        {status ? (
          <div
            className="cursor-pointer bg-hover-light-success text-hover-success ps-3 pe-5 py-3 rou
          >
            <KTIcon iconName="dollar" className="fs-3 me-2" />
            <span>Change to paid</span>
          </div>
        ) : null}
      </Popover.Body>
    </Popover>
  )
  return (
    <OverlayTrigger
      trigger="click"
      placement="bottom"
      render={renderPopover}
    />
  )
}
```

Figure 4.20 ChatGPT for debugging

ChatGPT provided value across multiple areas of the build phase by assisting in UI design through layout optimization and utility-first class usage, clarifying logic by validating conditional structures, supporting debugging by identifying rendering and DOM hierarchy issues, and improving maintainability through code refactoring and modularization suggestions.

4.8 Testing

The testing phase aimed to ensure that all developed features in the MBKM system functioned correctly, aligned with the requirements, and provided a seamless experience for different user roles. This phase involved the generation and execution of test scenarios to validate functional requirements. A distinctive aspect of this phase was the use of ChatGPT to assist in test case generation based on the Software Requirement Document (SRD).

4.8.1 Uploading the SRD to ChatGPT for Test Case Generation

To begin the test case creation process, the finalized SRD containing detailed functional specifications for MBKM registration and activity tracking was uploaded into ChatGPT. The author then prompted ChatGPT to analyze the document and generate corresponding test cases for each feature and user role. Figure 4.21 shows ChatGPT output.

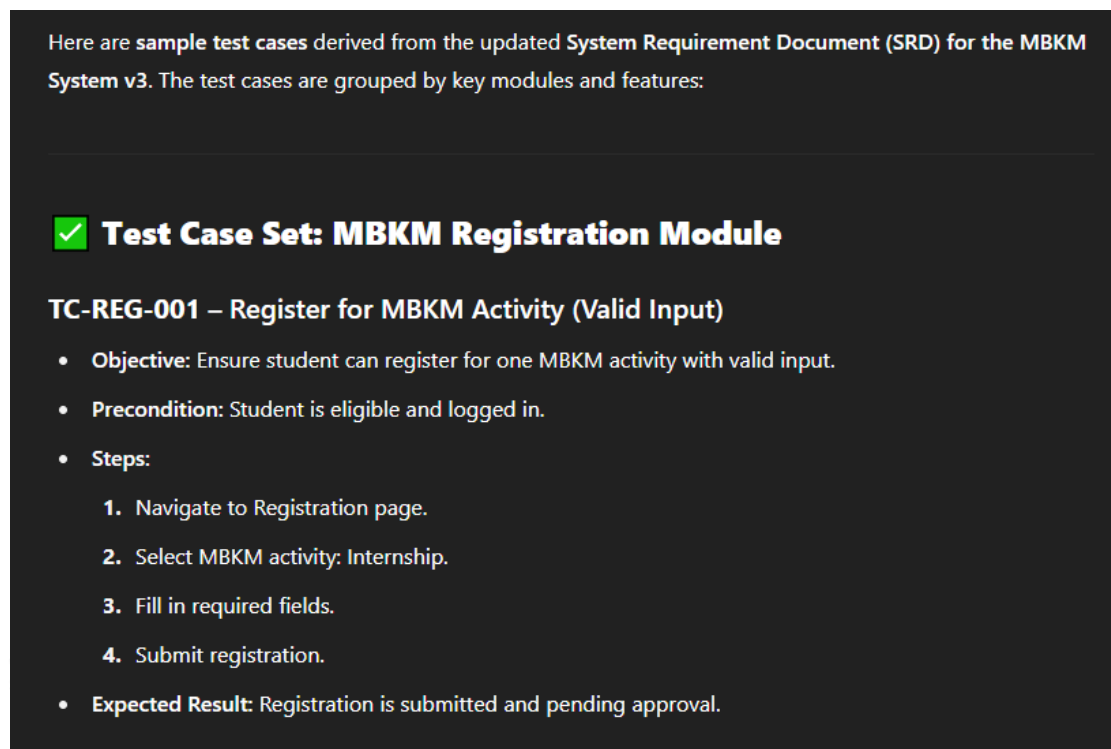


Figure 4.21 ChatGPT generate Test Case

ChatGPT successfully parsed the SRD and responded with a set of initial test scenarios. However, the first batch of outputs was too generic. These outputs failed to cover specific rules and constraints outlined in the SRD, such as the different approval processes between activity types or the minimum log threshold for activity completion.

4.8.2 Iterative Refinement through Prompt Engineering

To improve the quality of the test cases, the author iteratively refined the prompts and added clarification, including:

Specific rules from the SRD, such as: “Internship requires supervisor approval, while research requires advisor approval.”

Required business logic, like: “Every MBKM activity must include at least 40 approved logs to be considered complete.”

Visual and functional UI components, e.g., “Each student log page displays a chart showing log completion progress.”

seperate tc reg 001 and tc reg 002 for each mbkm activity

MBKM System Test Cases				
	Test Case ID	Title	Module	Precondition
1	TC-REG-003	Prevent Multiple Activity Registrations	Registration	Student has already registered
2	TC-APPROVAL-001	MBKM Staff Rejects Registration	Registration Workflow	Student has submitted registration
3	TC-APPROVAL-002	Head of Study Approves Registration	Registration Workflow	Staff has approved registration

Each MBKM activity now has its own registration test cases for both valid input (TC-REG-001-XXX) and missing required fields (TC-REG-002-XXX), where XXX reflects the activity (e.g., INT for Internship, RES for Research). Let me know if you'd like these grouped by activity or exported into a QA test case document.

Figure 4.22 Iteration done to enhance test case

With each iteration, as shown in Figure 4.22, ChatGPT’s outputs became more detail. The refined test cases included scenarios. These improvements enhanced the coverage and usability of the test cases for actual system validation. In the end, the test cases were restructured by the author to ensure a smooth and efficient testing process for all user types, minimizing the need for repeated back-and-forth clarifications during execution.

4.8.3 Test Execution and Validation

The SGU IT department used the refined test cases as a basis for manual testing across different modules. Testing covered:

Registration Workflows: Including activity-type validation, advisor assignment logic, and multi-role approval sequences.

Activity Tracking: Verifying the approval flow of logs by role (advisor/supervisor), progress bar logic, and data accuracy.

User Role Permissions: Ensuring users could only view or modify content appropriate to their assigned roles.

Dashboard Visualizations: Validating progress charts, report summaries, and alert behaviors.

Test results were logged in structured spreadsheets and internal QA documents. Bugs and edge cases identified during this process were documented and resolved prior to system deployment.

4.9 Summary of System Development

The development of the MBKM system was carried out using a modified Feature-Driven Development (FDD) model, with the author leveraging ChatGPT throughout each stage as a co-creator. This human-AI collaboration contributed significantly across all phases, from requirement elicitation, system modelling, UI design, coding, to testing. Each interaction was structured using prompt engineering principles to generate useful and context-aware outputs. The integration of ChatGPT was not intended to replace the developer, but rather to enhance the development process by reducing cognitive load, improving documentation structure, and offering alternative approaches to problem-solving.

In the Requirement Elicitation phase, ChatGPT effectively synthesized stakeholder interview results and generated structured requirement formats such as feature lists, use cases, and user stories. However, it required careful review, as the model occasionally misinterpreted implicit stakeholder needs or overgeneralized specific domain requirements.

During System Modeling, the AI generated various UML artifacts including use case diagrams, class diagrams, and activity diagrams. These served as a strong starting point but often suffered from layout and connector misalignment, requiring manual revision.

Iterative improvements did not always result in better output; thus, manual restructuring was often more efficient after a few attempts.

The Feature List and Plan by Feature phase demonstrated one of the clearest strengths of ChatGPT. Given a well-defined SRD and structured prompts, ChatGPT produced accurate and logically grouped feature sets aligned with user roles and system modules. These outputs required minimal revision, validating that with high-quality context and constrained tasks, ChatGPT can perform remarkably well in analytical structuring and categorization.

For the Design phase, ChatGPT was tasked with suggesting wireframe structure and layout logic, and converting text-based layouts into component blocks. While helpful for defining structure and screen relationships, its low visual fidelity and assumptions about standard HTML/CSS layout occasionally misaligned with frontend frameworks.

In the Build Phase, ChatGPT contributed by generating reusable code snippets, improving logic flow, suggesting validations, and offering refactoring recommendations. While the model helped in speeding up development, code sometimes lacked contextual fit or introduced syntactic errors, especially in deeply nested logic.

During Testing, uploading the SRD into ChatGPT enabled quick generation of structured test cases. These were further refined through prompt iteration and validated against actual feature behavior. However, coverage of negative paths and data realism in tests were areas where human oversight remained critical.

Table 4.1 summarizes the strengths and limitations of ChatGPT assistance in each development phase:

Table 4.1 Strengths and limitations of ChatGPT assistance

Development Phase	Strengths of ChatGPT Assistance	Limitations of ChatGPT Assistance
Requirement Elicitation	• Accelerated structuring of raw input • Generated use cases and user stories from interviews	• Missed implicit domain-specific needs • Tended to overgeneralize responses
System Modeling	• Provided UML diagram drafts • Accelerated conceptual modeling	• Poor layout control • Connector misalignment
Feature Breakdown	• Assisted in grouping features by user roles • Reduced effort in decomposing requirements	• Required reorganization based on domain insight
UI Design (Wireframe)	• Suggested layout logic and interface sections	• Low visual fidelity • Incompatible component assumptions
Coding	• Improved logic and error handling • Aided in refactoring	• Occasionally introduced syntax errors • Code often lacked project-specific context
Testing	• Quick generation of test cases • Helped ensure test structure consistency	• Lacked realistic test data • Edge case coverage incomplete
Documentation	• Drafted structured, well-formatted technical text	• Required accuracy and tone adjustments by author

Overall, ChatGPT proved to be a valuable co-creator, particularly in generating initial drafts and reducing the overhead of documentation and low-level logic implementation.

However, the human developer remained central to refining outputs, making contextual decisions, and ensuring final system accuracy and usability.

4.10 Comparison Between Traditional and AI-Assisted Development

This section presents a comparison between the traditional development approach, represented by the thesis module, and the AI-assisted development of the MBKM system conducted in this study. The comparison focuses on development approach, shown in Table 4.2.

Table 4.2 Development Approach Comparison

Aspect	Thesis Module	AI-Assisted MBKM Module
Development Method	Waterfall	Feature-Driven Development with ChatGPT integration
Requirement Gathering	Manual interviews and documentation	Manual interviews and ChatGPT-assisted transcript analysis and SRD generation
Design Modeling	Manually created UMLs and architecture	ChatGPT-generated and refined UML
Code Development	ChatGPT-assisted code generation, refactoring, and validation setup	ChatGPT-assisted code generation, refactoring, and validation setup
Testing	Manually written test cases	ChatGPT-generated test cases and manual tuning
Documentation Process	Manual creation, slower iterations	AI-generated drafts, refined iteratively by the author

To validate the qualitative differences observed in methodology and process, this study includes a quantified analysis of the three key metrics outlined in Chapter 3: documentation completeness, development time, and user acceptance

4.10.1 Documentation Completeness

Documentation quality was evaluated using a 4-dimension rubric (Existence, Coverage, Clarity, Traceability), with each dimension scored on a 5-point scale across three core artifacts: Requirements Specification, UML Diagrams, and Test Cases.

Table 4.3 Documentation Completeness Scoring

Artifact	Dimension	Thesis Module (1–5)	MBKM Module (1–5)	Comments
Requirements	Existence	5	5	Both systems had complete requirement specs.
	Coverage	5	5	Covered all core features in both modules.
	Clarity	3	5	MBKM requirements were clearer and better structured.
	Traceability	2	4	MBKM linked requirements to test cases; thesis had limited linkage.
Subtotal		15 / 20	19 / 20	
UML Diagrams	Existence	1	5	MBKM included more complete diagrams.
	Coverage	3	4	Thesis diagrams limited in scope.
	Clarity	4	4	Both systems provided readable diagrams.
	Traceability	1	4	MBKM linked diagrams to features and flows.
Subtotal		9 / 20	17 / 20	
Test Cases	Existence	5	5	Full test cases were documented in both.
	Coverage	5	5	MBKM slightly broader in covering variations.
	Clarity	3	5	MBKM used a more consistent format.
	Traceability	1	3	MBKM test cases linked to requirements.
Subtotal		14 / 20	18 / 20	

Table 4.3 shows a significant improvement in the MBKM module compared to the thesis module. The requirements specification in the MBKM project scored 19/20, an increase from 15/20 in the thesis module, indicating clearer requirements coverage. Similarly, UML diagrams improved from 9/20 to 17/20, reflecting the AI's contribution in quickly generating and refining visual models,. Test case documentation also showed a notable increase, from 14/20 to 18/20, suggesting that LLM assistance significantly supported the generation of test cases.

Overall, the MBKM module achieved 54/60 (90%), compared to 38/60 (63.3%) for the thesis module. This represents a 26.7% improvement in documentation completeness, showing that AI assistance effectively enhanced the quality and coverage of development artifacts. These results indicate that LLMs, when integrated into the software engineering process, are especially valuable in supporting documentation-related tasks, which are often time-consuming and error-prone when performed manually.

4.10.2 Development Time

The comparison was scoped to one activity module in each system (the thesis activity module and one MBKM activity module), to ensure a fair comparison. The durations reflect the total time from the start of development up to testing, excluding deployment and post-release iterations. Table 4.4 shows the MBKM module was completed in less than half the time, reflecting a 62.5% time savings compared to the thesis module.

Table 4.4 Development Time Comparison

Module	Estimated Total Duration
Thesis Module	8 weeks
MBKM Activity Module	3 weeks

4.10.3 User Acceptance

User acceptance was evaluated through UAT survey. Respondents were asked to rate their overall satisfaction, usability experience, and willingness to use the system.

Table 4.5 User Acceptance Scoring

Section	Question	Thesis Module (Avg Score)	MBKM Module (Avg Score)	Difference
Usability	The system is easy to navigate.	4.0	4.5	+0.5
	The user interface is clear and intuitive.	4.0	4.5	+0.5
	I can complete tasks with minimal effort.	4.0	4.5	+0.5
	The visual layout is consistent and professional.	5.0	4.5	-0.5
	I feel confident using the system without needing support.	5.0	4.5	-0.5
Section Average (Usability Experience)		4.4	4.5	+0.1
Functionality	The system functions as expected based on my role.	4.0	4.0	0
	All necessary features are available for my tasks.	4.0	4.0	0
	The system responds quickly to my input.	4.0	4.0	0
	The system handles errors or invalid inputs appropriately.	4.0	4.0	0
Section Average (Functionality)		4.0	4.0	0
Satisfaction	I am satisfied with the overall performance of the system.	4.0	4.5	+0.5
	I would prefer this over a manual/paper-based process.	5.0	4.5	-0.5
	I would recommend this system to others.	4.0	4.0	0
Section Average (Overall Satisfaction)		4.3	4.3	0
Average Score (All Sections)		4.23	4.26	+0.13

Table 4.5 shows that the MBKM module achieved very similar results to the thesis module in terms of user acceptance. The MBKM system recorded a usability score of 4.5 compared to 4.4 for the thesis module, while functionality and overall satisfaction remained identical at 4.0 and 4.3 respectively. When averaged across all sections, the MBKM module scored 4.26, only slightly higher than the thesis module's 4.23.

These findings indicate that there is no significant difference in user acceptance between the two systems. Both were perceived as usable, functional, and satisfactory by respondents. However, this result is important: despite being developed in significantly less time, the AI-assisted MBKM module reached the same level of user satisfaction as the traditionally developed thesis module. This suggests that AI-assisted development can deliver outcomes comparable in quality to human-only development, while offering efficiency gains in the development process.

4.11 Challenges and Recommendations

The integration of Large Language Models (LLMs) such as ChatGPT into the development of the MBKM system demonstrated clear benefits in improving documentation completeness, reducing development time, and achieving comparable user acceptance to human-only development. However, several challenges emerged during the process, many of which highlight the limitations of current AI tools when applied in real-world software engineering contexts.

Inconsistent Output Quality: ChatGPT often generated artifacts that were structurally correct but lacked logical consistency or completeness. For example, some test case drafts omitted edge cases, and requirements occasionally included redundancies.

Recommendation: Treat AI outputs as **first drafts** requiring systematic validation by developers. Standardized review checklists should be applied, and structured prompts should be reused to minimize inconsistencies.

Limitation in Visual Output While ChatGPT was effective in generating PlantUML scripts for diagrams, the rendered outputs frequently lacked visual clarity and proper layout. Connectors were misaligned, labels were cluttered,

and iterative refinements sometimes worsened the diagrams instead of improving them. This issue required significant manual intervention to make diagrams usable for documentation and stakeholder communication.

Recommendation: Use ChatGPT to generate initial diagram scripts but refine them manually in dedicated modeling tools. Future work may explore integrating AI directly with UML editors or adopting specialized models trained for design visualization.

Shallow Domain Understanding: The AI provided useful outputs based on structured requirements but struggled with deeper domain-specific knowledge of MBKM regulations and academic workflows. This sometimes led to inaccurate assumptions about approval processes or reporting requirements.

Recommendation: AI-generated artifacts should always be validated against domain documents and stakeholder input. Domain experts must remain actively involved to ensure correctness.

Overall, the challenges highlight that LLMs serve best as co-creators rather than replacements for human developers. They are particularly strong in accelerating documentation completeness and reducing development time, but remain limited in areas such as visual output quality and deep domain reasoning.