



Cairo University
Faculty of Computers and Artificial Intelligence
Information Systems Department



Graduation Project Report

Devistant

A Software Engineer Assistant

By

20217021

Bishoy George Emil

20216128

Farah Mohamed Abdelhamid

20216067

Omar Ahmed Mohamed

20218003

Toka Abdelghafar Ebrahim

Under the supervision of:

Dr. Ayman El Kilany

TA. Seif Gad

*Department of Data Science, Faculty of Computers and Artificial
Intelligence, Cairo University, Egypt*

July 2025

Acknowledgements

First and foremost, we would like to express our deepest gratitude to **Dr. Ayman El Kilany**, whose vision, guidance, and encouragement were instrumental to the success of this project. The originality of the Devistant concept stems from his insightful direction and his constant push for innovation in the field of software engineering.

Dr. El Kilany not only helped shape the foundation of this project but also challenged us to explore emerging technologies such as **multi-agent systems, LangChain, and LangGraph**, inspiring us to step beyond traditional development tools and embrace modern, intelligent frameworks. His mentorship has significantly enriched our technical and problem-solving skills, and his passion for teaching left a lasting impact on our learning journey.

We also extend our sincere thanks to the faculty members of the **Computers & Artificial Intelligence - Data Science Department at Cairo University**, whose continuous support and feedback guided us throughout the academic year.

Finally, we are grateful to our families for their patience, support, and encouragement during this intense and rewarding journey.

Abstract

This project introduces Devistant, a software engineering assistant designed to automate and streamline the Software Development Life Cycle (SDLC). By providing a single, comprehensive project description, **Devistant** leverages a multi-agent system built with Large Language Models (LLMs), LangChain, and LangGraph to provide end-to-end support and generate a complete set of initial project artifacts in one shot, including requirement analysis, task prioritization, project timelines, UML diagrams, and documentation. The system architecture features multiple specialized agents, each handling a specific aspect of the SDLC, and a **master agent (Supervisor)** that synchronizes their outputs to ensure consistency.

The motivation behind **Deviant** is to address the slow and inefficient aspects of traditional software development, where engineers rely on fragmented tools and manual processes, resulting in delays and errors. **Devistant** helps by providing a smart, **all-in-one platform** that automates the early and time-consuming steps of starting a software project. It can take a high-level description and automatically create requirements, task lists in order of priority, and design documents like UML diagrams.

Devistant's significance lies in its potential to dramatically reduce the initial setup time for software projects and serve as a robust foundation for development. While the current implementation focuses on one-shot generation, future work will expand **Devistant** into a fully interactive chat-based workspace, allowing for real-time collaboration and iterative refinement of all project components. In conclusion, **Devistant** helps turn a project idea into a clear, organized development plan, making it a practical and powerful tool for modern software teams.

Table of Contents

CHAPTER 1	INTRODUCTION	1
1.1	Background and Motivation	1
1.2	Problem Statement	1
1.3	Objectives	2
1.4	Project Scope and Limitations	4
1.5	Project Methodology	5
1.6	Project Report Outline	7
CHAPTER 2	MARKET AND LITERATURE SURVEY	9
2.1	Market analysis	9
2.2	Literature Review	11
CHAPTER 3	DEVISTANT ANALYSIS	13
3.1	Problem Description	13
3.2	Requirements	13
3.3	Stakeholders and Responsibilities	16
3.4	System Scenarios	16
3.5	UML Diagrams	20
CHAPTER 4	DEVISTANT DESIGN AND IMPLEMENTATION	25
4.1	System Design	25
4.2	System Implementation	28
4.3	Technology Summary	32
CHAPTER 5	DEVISTANT TESTING AND EVALUATION	33
5.1	Testing	33
5.2	Evaluation	56
CHAPTER 6	CONCLUSIONS AND FUTURE WORK	58
6.1	Conclusion	58
6.2	Future Work	59
6.3	Lessons Learned	62

List of Figures

<i>Figure number</i>	<i>Page Number</i>
Figure 1. Devistant Use Case Diagram	20
Figure 2. Devistant Class Diagram	21
Figure 3. Devistant Sequence Diagram (One Shot)	22
Figure 4. Devistant Sequence Diagram (Chat-based)	23
Figure 5. Requirement analysis agent Sequence Diagram	24
Figure 6. Full Conversation Log Use Case Diagram	44
Figure 7. Full Conversation Log Class Diagram	45
Figure 8. Full Conversation Log ER Diagram	48

List of Tables

<i>Table number</i>	<i>Page Number</i>
Table 1: List of Abbreviations	6
Table 2 : Competitor Analysis	10
Table 3: Stakeholders and Responsibilities	16
Table 4: Technology summary	32
Table 5: Tests Conducted	34
Table 6: Routing Logic Tests	35
Table 7: Full Conversation Log Database Schema Design Tables	46 : 48 ,52
Table 8: Test Metrics summary	58

List of Abbreviations

Abbreviation	Meaning
SDLC	Software Development Life Cycle
LLMs	Large Language Models
UML	Unified Modeling Language
CI	Continuous Integration
CD	Continuous Delivery
NLP	Natural language processing
AI	Artificial Intelligence
PWA	Progressive Web App
JWT	JSON Web Token
API	Application Programming Interface
PCI-DSS	Payment Card Industry Data Security Standard

Chapter 1 Introduction

1.1 Background and Motivation

Traditional software project management relies heavily on manual tasks and fragmented coordination, often causing delays and errors. **Devistant** addresses this by using **LLMs**, **LangChain**, **LangGraph**, and a **multi-agent system** to automate and unify the SDLC, reducing setup time and improving consistency for software teams.

1.2 Problem Statement

The software development lifecycle (SDLC) is often hindered by fragmented tools and manual processes, particularly during critical phases such as requirement analysis, planning, and design. Traditional methods rely heavily on disjointed workflows, where engineers manually gather requirements, prioritize tasks, and generate design artifacts using separate tools. This lack of integration leads to:

- **Context switching:** Developers waste time moving between tools.
- **Data inconsistency:** Information gets lost or duplicated across platforms.
- **Collaboration gaps:** Teams may struggle to keep everyone aligned
- **Inconsistencies:** Miscommunication or errors in translating requirements into actionable tasks or design elements.
- **Scalability Challenges:** Difficulty in maintaining coherence across large or complex projects due to the absence of a unified system.

For instance, requirement analysis often involves iterative negotiations between stakeholders, which can delay project timelines if not streamlined. Similarly, design artifacts like use cases or class diagrams are typically created in isolation, risking misalignment with evolving requirements.

Devistant addresses these challenges by providing an intelligent, LLM-driven assistant that automates and synchronizes key SDLC tasks through specialized agents. The system aims to bridge the gap between fragmented tools and a cohesive development process, ensuring consistency, reducing manual effort, and enabling real-time adaptability to changes in project scope.

1.3 Objectives

Devistant AI aims to deliver an all-in-one, AI-powered workspace that provides complete software project planning and design support through both **One-Shot generation** and **Interactive Chat-based workflows**, and lays the groundwork for future iterative and collaborative features.

1.3.1 Primary Objective

1. Develop a Unified Software Engineering Assistant

- Create a multi-agent system where each agent specializes in a specific aspect of the SDLC (e.g., requirement analysis, task prioritization, UML generation).
- Integrate all agents under a master agent (Supervisor) to ensure synchronization and consistency across tasks.

2. Automate Repetitive Tasks:

- Automate requirement analysis, task prioritization, UML diagram generation, and documentation to reduce manual effort.
- This helps achieve a reduction in time for these tasks compared to the traditional Methods

3. Enable Iterative Learning:

- Provide a chat-based interface for live agent collaboration, feedback, and negotiation..
- Continuously adapt to user preferences and edits to improve accuracy and relevance.

1.3.2 Deliverables

1. Functional Prototype:

- Two fully implemented **Devistant** versions (One-Shot generation and Interactive Chat-based workflow), tests, evaluations, and fine-tuning notebook with datasets.
- Support for natural language inputs, multi-agent coordination, and real-time updates to project components.

2. Documentation and Reports:

- Complete technical documentation of system architecture, agent responsibilities, orchestration mechanisms, and diagrams.
- A final report outlining all development phases, testing outcomes, and lessons learned.

1.3.3 Intangible Objectives

1. Increase Productivity:

- Reduce the time and effort required for repetitive tasks, allowing developers to focus on higher-value activities.
- Improve collaboration and communication among team members by providing a unified platform for project management.

2. Enhance User Experience:

- Provide a user-friendly interface with both one-time and iterative refinement capabilities.
- Ensure that the system adapts to the specific needs and preferences of individual users or teams.

3. Promote Innovation:

- Demonstrate the potential of AI-driven tools to revolutionize software engineering workflows.
- Demonstrate the practical integration of LLMs, LangGraph, and LangChain in software engineering and inspire future enhancements.

1.4 Project Scope and Limitations

The scope of this project focuses on the development and implementation of **Devistant**, an AI-powered assistant designed to automate and streamline key tasks within the Software Development Life Cycle (SDLC). The primary tasks that Devistant aims to address include:

- **Requirement Analysis:** Automating the gathering and analysis of project requirements from stakeholders, helping to minimize manual effort and ensure accurate documentation.
- **Task Prioritization:** Utilizing AI to assist in prioritizing development tasks based on project requirements and urgency.
- **UML Diagram Generation:** Automatically generating Unified Modeling Language (UML) diagrams, such as use cases and class diagrams, from the requirements and design information.
- **Documentation Creation:** Automating the generation of technical documentation, including summaries and reports, to improve team efficiency and to keep all team members aligned and in case of handing work from one member to another.

The system leverages **Large Language Models (LLMs)** to process natural language input and provide intelligent responses, making interactions seamless. The multi-agent architecture ensures that specialized agents handle different aspects of the SDLC, with a master(Supervisor) agent synchronizing tasks to maintain consistency and efficiency.

While **Devistant** aims to address significant challenges in the SDLC, several limitations need to be considered:

- 1. Input Data Quality:** The effectiveness of the AI assistant is heavily dependent on the quality of the input data. Inaccurate, vague, or incomplete inputs may lead to suboptimal outputs in requirements analysis, task prioritization, and diagram generation.
- 2. Natural Language Understanding:** Although **Devistant** utilizes advanced LLMs, certain domain-specific or highly technical language may not be fully understood, potentially affecting the precision of generated outputs such as UML diagrams or database schemas.
- 3. Tool Integration:** **Devistant** in its current iteration does not integrate with external project management or version control tools (e.g., Jira, Git). This limits its ability to fully synchronize with existing development environments. Developers must manually transfer data between **Devistant** and other platforms, which can be time-consuming and prone to errors.
- 4. Real-time Collaboration Between Developers:** The system's ability to support real-time collaboration may face scalability challenges, particularly in large teams working on complex projects that may face performance bottlenecks. Ensuring seamless collaboration across a large user base may require further optimization.
- 5. User Onboarding and Adaptation:** As with any AI-powered tool, there may be an initial learning curve for users unfamiliar with such systems. The ease of adoption will vary depending on individual or team familiarity with AI-driven interfaces.

1.5 Project Methodology

This project adopted the Agile methodology to guide the design, development, and testing of Devistant. The development process was centered around short, iterative sprints, allowing the team to incrementally implement both the **One-Shot SDLC generation pipeline** and the **Interactive Chat-Based workflow**. This methodology enabled flexibility, continuous feedback, and efficient task delivery.

1.5.1 Agile Methodology

Although **Devistant** supports a one-shot execution version, the system was developed iteratively using Agile practices. Agile enabled rapid feature delivery, regular feedback incorporation, and adaptability to evolving system requirements.

Key features of the Agile approach:

- **Sprints:** The project has been broken down into manageable iterations, each focused on delivering a specific set of features, a specific agent, integration mechanism (e.g., agent logic, diagram generation, dependency mapping).
- **Sprint Reviews:** After each sprint, the implemented features were reviewed and evaluated to ensure alignment with project goals and user feedback.
- **Collaborative Planning:** Weekly planning meetings and daily discussions enabled team members to coordinate and adapt progress.

While the final product supported 2 versions of one-shot generation and an Interactive Chat-based System, the **Agile framework** was used to manage the software development process effectively.

1.5.2 One-Shot SDLC Generation Pipeline

The core execution methodology used by **Devistant** is the **one-shot generation pipeline**, where a complete SDLC workflow is generated in a single pass. Upon receiving the user's project description, the system automatically activates all agents to produce the following in sequence:

- Requirement Analysis
- Planning and Task Prioritization
- Project Management
- UML Diagram Generation (Use Case, Class)
- Database Schema Design
- Technical Documentation and Summary

This pipeline significantly reduces the manual effort and time traditionally required to go through each SDLC phase step by step. It provides a comprehensive blueprint that can later be reviewed and edited manually or interactively in future system versions.

1.5.3 Chat-Based Workflow version

The other enhanced version of **Devistant** is the **Chat-Based workflow**. This version enables step-by-step collaboration with the system. Users can:

- Interact with the Master Agent or individual sub-agents
- Modify and refine specific outputs (e.g., requirements, diagrams)
- Receive synchronized updates across dependent artifacts through LangGraph-powered dependency tracking

This flexible, **human-in-the-loop design** supports iterative project refinement and real-time adaptability.

1.5.4 Multi-Agent System Architecture

Devistant follows a **multi-agent architecture**, where each agent specializes in one area of the software development process. The architecture is designed to be scalable, modular, and adaptable using a hierarchical architecture design.

Key features of the Multi-agent approach:

- **Specialized Agents:** Each agent is responsible for a distinct SDLC task, such as requirement analysis, planning, design, database modeling, or documentation.
- **Master Agent (Supervisor):** Acts as a coordinator and ensures consistency by supervising all other agents, resolving conflicts, and ensuring cross-agent synchronization.
- **Agent Collaboration:** Agents communicate indirectly via shared context, shared memories, and dependency maps. They operate either sequentially (in one-shot version) or on demand (in chat-based version).

This agent-based approach mirrors the structure of human software teams and supports future real-time or parallel execution.

1.5.5 LangChain and LangGraph Orchestration

To enable intelligent orchestration and flow control between agents, **Devistant** leverages:

- **LangChain:** Manages prompt templates, language model memory, input-output chains, and persistent context for each agent.
- **LangGraph:** Manages state transitions, execution flow, and conditional branching between agents. It supports both linear (one-shot) and dynamic (chat-based) workflows.

This setup allows **Devistant** to maintain execution context, adapt to user inputs, and support re-generation or rollback of outputs.

1.5.6 Dependency Map and Context Management

To address the **context window limitations** of LLMs, **Devistant** implements **dependency maps** that track relationships between components of the project. For example:

- A change in requirements automatically triggers updates to class diagrams and task plans.
- Each agent only receives the minimal necessary context, reducing token usage and avoiding information loss.

These maps are generated by the master agent and used to maintain consistency, prevent redundant operations, ensure traceability across the SDLC, and enable seamless synchronization across all project components

1.5.7 Diagram Rendering with Mermaid.js

All diagrams (Use Case, Class, Database) are generated using **Mermaid.js** syntax. Each design-related agent predicts the corresponding Mermaid code, which is:

- Rendered in real time in the frontend for user preview.
- Editable in either code or visual form.
- Exportable for integration into reports or documentation.

This approach allows lightweight, flexible, and easily modifiable diagrams that align with modern AI-powered development environments.

1.5.8 Chat-Based Workflow Implementation

The **Chat-Based version** of **Devistant** has been fully implemented, offering users an interactive and iterative software engineering experience. This version enables:

- **Conversational Interactions:** Users can chat with the Master Agent or target specific sub-agents.
- **Live Edits:** Project components such as requirements or diagrams can be revised and updated on demand.
- **Automatic Synchronization:** Dependency maps ensure that edits are propagated to all affected outputs.
- **Memory Management:** LangGraph supports state retention for longer, more complex conversations and workflows.

This version transforms **Devistant** from a static assistant into a dynamic, intelligent workspace for software teams

1.6 Project Report Outline

- **Chapter 2 : Presents the Market & Literature Survey:**

This chapter examines the current market landscape for AI-powered software engineering assistants. It discusses how organizations and developers currently manage software development tasks, especially requirement analysis, task planning, and documentation. It also explores industry trends, related tools, and identifies gaps that Devistant addresses. The chapter includes a competitor analysis and a literature review on the use of AI in the SDLC.

- **Chapter 3 : Presents the System Analysis of Devistant:**

This chapter examines **Devistant's** architecture and functionalities, addressing the problem description, system requirements, key stakeholders, and typical scenarios. It discusses specialized agents, their roles, the master agent's coordination, and includes UML diagrams and dependency maps.

- **Chapter 4 Presents Design and Implementation:**

This chapter describes the overall system design, including the multi-agent architecture, the one-shot execution flow, and how LangChain and LangGraph are used for orchestration. It also explains how Mermaid.js is used for diagram rendering, how agents interact through dependency maps, and how the system ensures consistency and modularity in outputs.

- **Chapter 5 presents Testing & Evaluation:**

This chapter covers the testing strategy used to validate **Devistant**, including unit and integration testing across agents, user testing, and evaluation of output accuracy. It documents sample test cases, performance metrics, and overall system reliability. Insights from the evaluation phase are also presented, showing how **Devistant** meets its functional goals.

- **Chapter 6: Work & lessons learned, Conclusion, and Future Work.**

This chapter summarizes the key outcomes of the project and how **Devistant** contributes to automating software engineering tasks. It reflects on the lessons learned, outlines current limitations, and proposes future enhancements such as a real-time chat interaction version, DevOps integration, and advanced team management features.

Chapter 2 Market and Literature Survey

2.1 Market Analysis

The rapid evolution of Artificial Intelligence (AI) has significantly influenced the landscape of software development tools. As modern software systems grow in complexity and the demand for shorter development cycles increases, there is a clear market need for intelligent platforms that automate core Software Development Life Cycle (SDLC) activities, such as requirement analysis, task planning, UML diagram generation, and documentation.

Devistant addresses this market gap by leveraging Large Language Models (LLMs) within a modular multi-agent architecture. The system supports both **One-Shot** and **Interactive Chat-Based** execution versions, enabling teams to automate the entire SDLC within a single unified platform.

2.1.1 Understanding the Market

A clear understanding of current practices in software engineering is essential for positioning Devistant effectively. Most development teams continue to rely on fragmented tools to manage different SDLC phases, leading to inefficiencies and communication gaps.

Key Observations:

- **Target Users:**
 - Software engineering teams, project managers, freelancers, and startups are aiming to streamline SDLC processes and enhance team coordination.
- **Current Challenges:**
 - Repetitive manual processes, such as requirement documentation.
 - Lack of integration between tools, leading to data silos and inconsistency.
 - Communication delays and version mismatches across platforms.

Market Potential:

The global market for AI in software engineering is rapidly expanding. Key drivers include the rise of AI-assisted coding, Agile automation, intelligent documentation tools, and enterprise-wide digital transformation initiatives.

2.1.2 Industry Trends

Several major trends are contributing to the growth of AI-powered assistants in software development:

- **Widespread Adoption of AI in Engineering Workflows:**
Organizations are increasingly integrating AI assistants for tasks such as code suggestions, requirement parsing, and documentation generation, significantly reducing manual effort and human error.
- **Growth of Agile and DevOps Practices:**
The demand for continuous integration and delivery has made automation essential. AI aligns well with Agile workflows by accelerating sprint planning, backlog prioritization, and daily reporting.
- **Natural Language Processing (NLP) in Project Management:**
NLP technologies are enabling tools that interpret human-written project descriptions or conversations, making AI systems accessible to non-technical stakeholders and reducing communication barriers.

2.1.3 Competitor Analysis

While several tools exist in the market, most address isolated aspects of the SDLC and do not offer comprehensive automation.

Tool	Strengths	Limitations
Jira (Atlassian)	Task and project tracking with some automation rules.	Does not offer AI-powered requirement analysis or technical diagram generation.
GitHub Copilot	Assists developers in writing code using AI.	Focuses solely on code generation, lacking support for planning, design, or documentation.
ClickUp / Monday.com	Project and team management with automation.	Limited AI capabilities in technical software development domains.

Devistant stands out as an all-in-one, LLM-powered software engineering assistant. It automates requirement analysis, task planning, UML, and database diagram generation, and technical documentation through a synchronized multi-agent system, making it a unique and comprehensive solution.

2.2 Literature Review

This section surveys recent research and industrial insights regarding AI applications in software engineering, particularly within the context of SDLC automation, natural language processing, and intelligent systems for project support.

Recent literature underscores the transformative role of AI in improving development efficiency, reducing errors, and enhancing collaboration.

- **Automated Requirement Engineering:** NLP-based agents can extract and structure requirements from natural language project descriptions, improving accuracy and reducing human dependency.
- **LLMs for Documentation and Code Generation:** Pretrained language models such as GPT have demonstrated effectiveness in generating coherent documentation, UML models, and code snippets.
- **Challenges Identified:**
 - Ensuring **data privacy** and model explainability.
 - Difficulty integrating AI systems with **legacy development tools**.
 - Continued need for **human oversight** to validate outputs.

These findings support **Devistant's** role as a **human-in-the-loop** assistant that complements developer workflows while enhancing speed and consistency.

2.2.1 How AI is used in SDLC

AI is applied in different stages of the SDLC to improve efficiency, accuracy, and automation. Here are some ways AI is making an impact:

1. Requirement Analysis

- AI tools can read and analyze project requirements from text documents, emails, or conversations.
- Natural Language Processing (NLP) helps convert human-written requirements into structured data that software teams can use.

Example: AI can automatically detect unclear or missing requirements and suggest refinements.

2. Task Prioritization

- AI can rank development tasks based on urgency, dependencies, and workload distribution.
- Machine learning models analyze past project data to suggest which tasks should be completed first.

Example: An AI system could recommend focusing on security features before UI improvements if security risks are detected.

3. UML Diagram Generation

- AI can automatically generate **Unified Modeling Language (UML) diagrams** (use case diagrams, class diagrams) based on textual descriptions.
- This reduces the time spent on manual diagram creation and ensures consistency.

Example: If a user describes a login system, AI can generate a UML use case diagram showing the interactions between the user and the system.

4. Documentation Automation

- AI-powered tools can create **technical documentation** by summarizing project details, requirements, and system architecture, reducing manual effort.
- Developers no longer need to manually write long reports, which saves time, as AI extracts key details and formats them automatically.

Example: AI can generate API documentation from code comments, saving time and ensuring clarity and completeness for developers

2.2.2 How Devistant Leverages AI in SDLC:

Devistant integrates recent advancements in artificial intelligence into a cohesive, end-to-end system that enhances every major phase of the Software Development Life Cycle (SDLC). Its architecture and intelligent orchestration enable the following capabilities:

- **End-to-End SDLC Automation:** **Devistant** can generate a complete software project plan in a single execution, including requirements, task prioritization, UML diagrams, database schema, and documentation.
- **Structured Diagram Generation:** All UML diagrams are generated in real time using Mermaid.js syntax. These diagrams are rendered live, editable, and exportable, ensuring both clarity and flexibility.
- **Interactive Learning and Adaptation:** The system continuously improves output quality by learning from user edits, preferences, and iterative feedback during interactive chat-based sessions.
- **Consistent and Synchronized Outputs:** Through a master agent and an internal dependency mapping mechanism, Devistant ensures consistency across all generated components. Updates in one part of the project (e.g., requirements) automatically propagate to dependent modules (e.g., class diagrams, task plans).

Devistant distinguishes itself from conventional tools by offering a unified workspace that integrates all major AI-enhanced SDLC functions. Its multi-agent design, real-time adaptability, and human-in-the-loop interactions make it a powerful platform for automation, collaboration, and productivity in modern software engineering.

Chapter 3 Devistant Analysis

3.1 Problem Description

The Software Development Life Cycle (SDLC) encompasses several stages, including requirement analysis, planning, design, and documentation. However, traditional development methods often rely on fragmented tools and manual workflows, which introduce inefficiencies, delays, and inconsistencies.

Common challenges faced by development teams include:

- **Manual Requirement Gathering:** Extracting and refining requirements from stakeholders is time-consuming and prone to misinterpretation and bias.
- **Task Planning & Prioritization Complexity:** Manually determining task priorities can be inconsistent and inefficient, especially across large or distributed teams.
- **Isolated Design and Documentation Tools:** UML diagrams and documentation are often created in disconnected platforms, leading to data loss or duplication.
- **Scalability Issues:** Managing evolving requirements and interdependent outputs across multiple tools becomes increasingly difficult as project complexity grows.

Devistant addresses these challenges by introducing a unified, AI-powered assistant that automates and synchronizes core software engineering tasks. Powered by a multi-agent architecture and a central master agent, Devistant ensures consistency across requirement analysis, task planning, design generation, and documentation, delivered in a single one-shot execution.

3.2 Requirements

Devistant is built to make software development easier by automating the Software Development Life Cycle (SDLC). It uses a group of specialized agents, each focused on a specific task like gathering requirements, planning tasks, or creating diagrams. A central Master Agent manages all the agents to keep their work consistent and connected.

Devistant offers two versions of the system:

- **One-Shot Version:** The user enters a full project description once, and Devistant generates the entire SDLC starting from requirements, tasks, diagrams till the documentation in one go.
- **ChatBot Version:** The user can interact with Devistant step-by-step, make changes at any stage, like updating requirements or diagrams, and get updated results. Each agent keeps a memory of the conversation, allowing for smoother and more flexible project development.

3.2.1 Functional Requirements

1. Requirement Analysis Agent

- Extracts and refines software requirements from user input.
- Uses NLP to detect missing or ambiguous requirements.
- Provides a structured output for review and iteration.

The Requirement Analysis Agent aids in transforming the user's project description and idea into a detailed requirement specification. Upon receiving the initial input, the agent provides a refined analysis. The user can then negotiate, adjust, and finalize the requirements with the agent's assistance, ensuring that all requirements are clear and comprehensive. The Requirement Agent may need to ask some questions to fully comprehend and understand exactly what you need (in the chatbot version).

2. Planning Agent

- Analyzes project scope to recommend task priorities.
- Adapts priorities based on dependencies and team workload.

The **Planning Agent** creates a clear, ordered list of tasks based on what the user provides. It helps break the project into manageable steps, making it easier to see what needs to be done and track progress.

3. Project Management Agent

The **Project Management Agent** supports both Agile and Waterfall methodologies, tailoring the project's timeline, sprints, and meetings according to the user's preferences. It dynamically adjusts workflows, helping to optimize the project's progress by tracking milestones, sprint completion, and team workload.

4. Design and Architecture Agents

- **Use Case Agent:** Creates use case diagrams and use case stories to represent user interactions.
- **Class Diagram Agent:** Suggests class structures and relationships, then renders UML class diagrams.
- **Database Agent:** Recommends database models and schema, tailored to the defined requirements and class design.

All diagrams are generated as **Mermaid.js code**, enabling both editable code and live visualization.

5. Documentation Agent

- Summarizes project details and automatically generates technical reports.
- Maintains version history and allows real-time collaboration.

This agent compiles essential documentation, summaries, and reports, ensuring that the user stays informed and can quickly understand the project's progress and outputs in case of handing over. It automatically generates technical reports, keeps a version history, and allows real-time collaboration on documentation.

6. Master Agent (Supervisor Agent)

- Synchronizes all specialized agents.
- Ensures updates in one area reflect across all related components.\
- Provides an interactive chat-based interface for user interaction.
- Handles the sub-agent's memory and states

The **Master Agent** coordinates all specialized agents, synchronizing their outputs and ensuring consistency across the development process. It integrates user feedback in real-time, updating and refining the project's deliverables as needed.

The **Master Agent** also provides an interactive chat-based interface for seamless user interaction and smooth communication between agents using map dependency.

3.2.2 Non-Functional Requirements

1. **Scalability:** While the system is initially designed for a single user, the architecture must be scalable to support multiple simultaneous users in future iterations. It should efficiently handle increasing project sizes and complexities, ensuring smooth performance regardless of the project's scale.
2. **Performance:** The system must deliver prompt responses, with processing times of a few seconds for most interactions. This ensures a seamless user experience even with complex requests or large inputs, maintaining high responsiveness throughout the software development cycle.
3. **User Experience:** The system should feature an intuitive, user-friendly interface that minimizes the learning curve. The design must prioritize ease of navigation, with clear visual cues and interactions that guide the user through the process without confusion, enhancing usability and efficiency.
4. **Security:** All user data must be securely stored and protected against unauthorized access. The system should implement robust access controls, ensuring that only authorized users can interact with sensitive project data. Encryption and secure storage protocols should be enforced to protect data integrity and privacy.
5. **Integration:** The system must be designed with flexibility in mind, supporting API integration for potential future expansions. This will allow the seamless integration of additional features, third-party services, or external tools as the system evolves, ensuring its adaptability to new requirements and technologies.

3.3 Stakeholders & Responsibilities

Stakeholders	Responsibilities
Software Engineer (User)	Provides input at each stage, reviews and refines generated outputs, and approves final decisions
Project Manager	Oversees project progress, defines milestones, and ensures timely delivery
Development Team	Uses the tool for assistance in software engineering processes
System Administrator	Maintains and updates Devistant , ensuring reliability and security

3.4 System Scenarios

3.4.1 Requirement Analysis Workflow

1. The user inputs the initial project description and requirements into the system.
2. The Agent processes the input, refines the details, and identifies any ambiguities or missing elements using NLP.
3. **In Chat-Based Version only:** The Agent may ask 3 to 5 questions to get high-quality results.
4. The requirements agent updates the supervisor agent to then fill in the following agents: planning Agent, project management, design Agents, and documentation agent based on the finalized requirements, ensuring that all agents will be aligned with the user's project.

3.4.2 Planning & Project Management Workflow

1. The system analyzes finalized requirements to identify development tasks.
2. Tasks are ordered based on importance, logical flow, and dependencies.
3. The task list is generated and visualized for the user.

3.4.3 Design Agents Workflow

1. Based on requirements and tasks, the Design Agents generate Use Case and Class diagrams and Database Design.
2. Mermaid.js code is used to create each diagram.
3. The diagrams are rendered visually and are editable via Mermaid preview.
4. The system ensures consistency between diagrams and upstream inputs.

3.4.4 Documentation Workflow

1. The Documentation Agent collects and summarizes information from all agents.
2. It creates a clear technical report that includes the project's requirements, tasks, diagrams, and the reasons behind the design choices.
3. The report is presented to the user and can be edited or finalized for export.

3.4.5 Master Agent Workflow

Initializes the workflow by gathering inputs from the user and coordinating the execution of tasks across all specialized agents.

As the user interacts with the system, the **Master Agent** monitors the progress of each agent, ensuring that their outputs remain consistent and up to date.

Finally, Aggregate and integrate all information, generating a cohesive and updated project summary that includes all the agents' outputs, ready for final review or further refinement.

3.4.6 One-Shot Version Workflow

- The user provides the entire project description once at the beginning.
- The Master Agent provides a dependency map and triggers all agents to run in a specific order.
- Each agent produces its output based on the previous ones, starting with **requirements** → **planning** → **project management** → **designing** → **documentation**.
- The whole SDLC plan is generated in a single pass and presented to the user as a complete package.

This Version is ideal for users who prefer quick, full results with minimal interaction.

3.4.7 One Shot System Scenario :

User Story: The software engineer provides **Devistant** a full project description so it can understand requirements and generate all related outputs automatically in one pass.

Step-by-Step Flow:

User Input: "I'm building an e-commerce platform for local businesses."

System Action:

The Requirement Analysis Agent processes the entire input using Natural Language Processing (NLP). Even though there's no back-and-forth conversation, it intelligently infers details, fills in gaps, and creates structured requirements using general software templates.

Example Output (Auto-generated):

- Users can register via email or Google.
- Sellers can manage their inventory.
- Buyers can browse, purchase, and track orders.
- Admin can approve sellers and manage listings.

These finalized requirements are then passed directly to the Supervisor Agent, which orchestrates the rest of the process:

Planning Agent --> creates a prioritized task list.

Design Agents--> generate diagrams using Mermaid.js.

Database Agent --> suggests tables and schema.

Documentation Agent -->compiles a final report.

Outcome:

The user receives a complete, ready-to-use SDLC package all based on their initial input in the description, with zero follow-up questions required.

3.4.8 Chat Version Workflow

- The user starts by describing the project and can chat step-by-step with the system.
- The Agents may ask 3 to 5 questions to get high-quality results.
- Each agent keeps its memory, allowing the user to make changes at any stage, like updating requirements or re-generating a diagram.
- The Master Agent updates the dependency map and ensures all connected outputs are refreshed.
- The user can refine or explore different parts of the project interactively.

This Version supports real-time collaboration and is ideal for more detailed or evolving projects.

3.4.9 Chat Version scenario:

User Story:

The software engineer provides **Devistant** a full project description and answers follow-up questions from **Devistant** to ensure the generated requirements are clear, complete, and tailored to user needs.

Step-by-Step Interaction:

User Input: “I’m building an e-commerce platform for local businesses.”

Devistant (Requirement Analysis Agent):

To better understand your project, I have a few quick questions:

- What roles should the system support (e.g., admin, seller, buyer)?
- Should sellers manage their own inventory?
- Will users sign up with email or through social accounts?

User: “Only sellers and buyers for now. Yes to inventory management. And let’s allow email and Google sign-in.”

Devistant: “Got it! I’ve updated your requirements accordingly. Here’s the draft list, let me know if anything’s missing.”:

1//2//3//...

User: “Looks perfect. Let’s proceed.”

System Action:

The finalized requirements are passed to the Supervisor Agent, which coordinates the rest of the agents using a dependency map to ensure that:

Tasks, diagrams, and schema all align with the latest inputs

Only affected agents are re-triggered if something changes later

Documentation reflects the current state of the project

Outcome:

The user receives outputs that evolve with their input, ideal for dynamic projects or when clarity matters.

3.5 UML Diagrams

3.5.1 Use Case Diagram

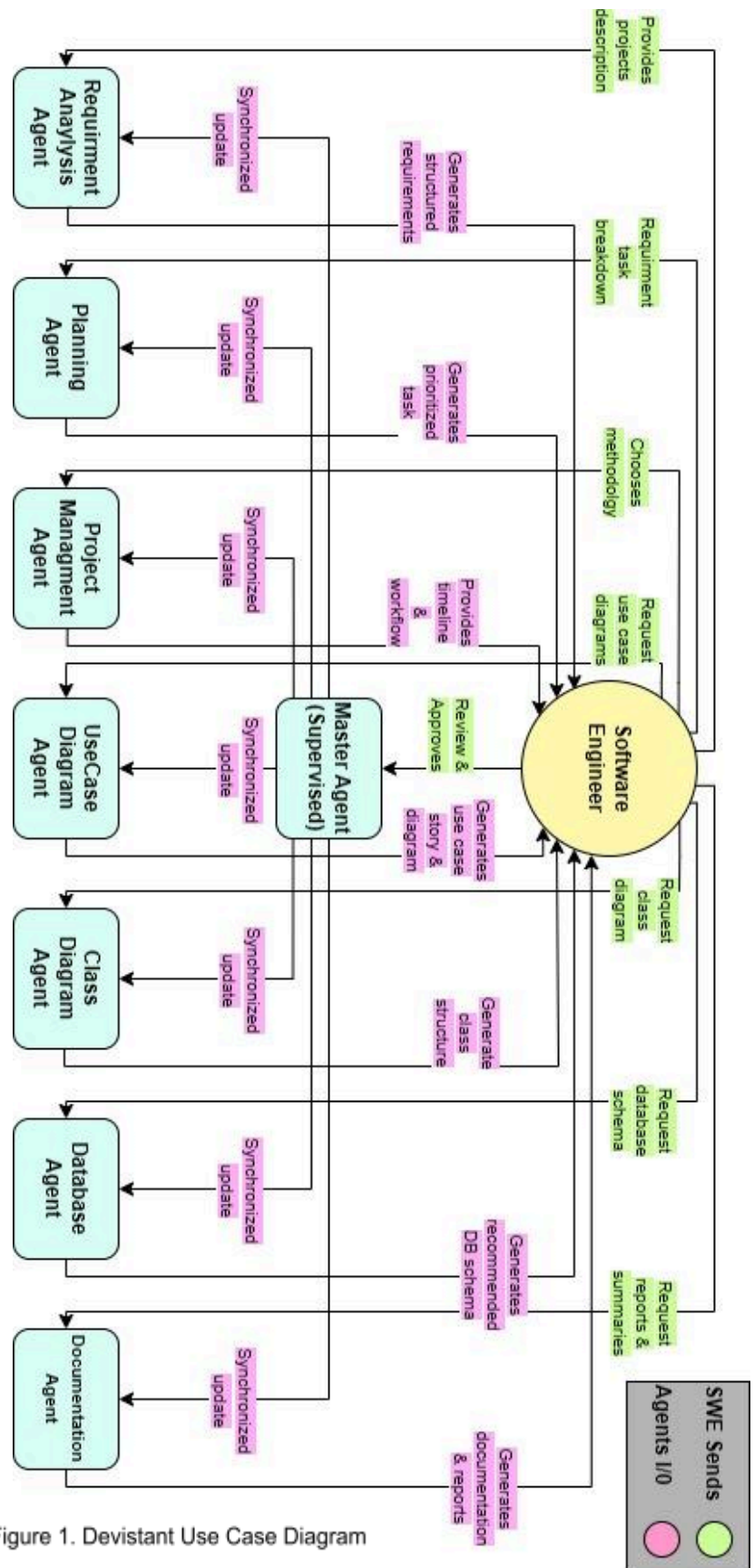
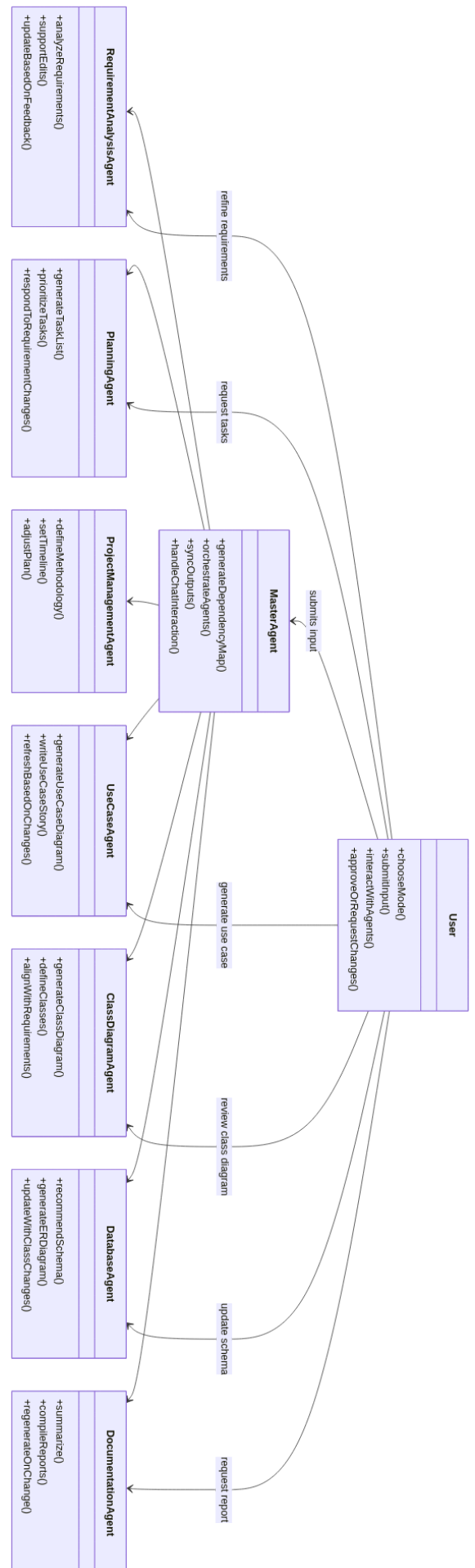


Figure 1. Devistant Use Case Diagram



3.5.2 Class Diagram

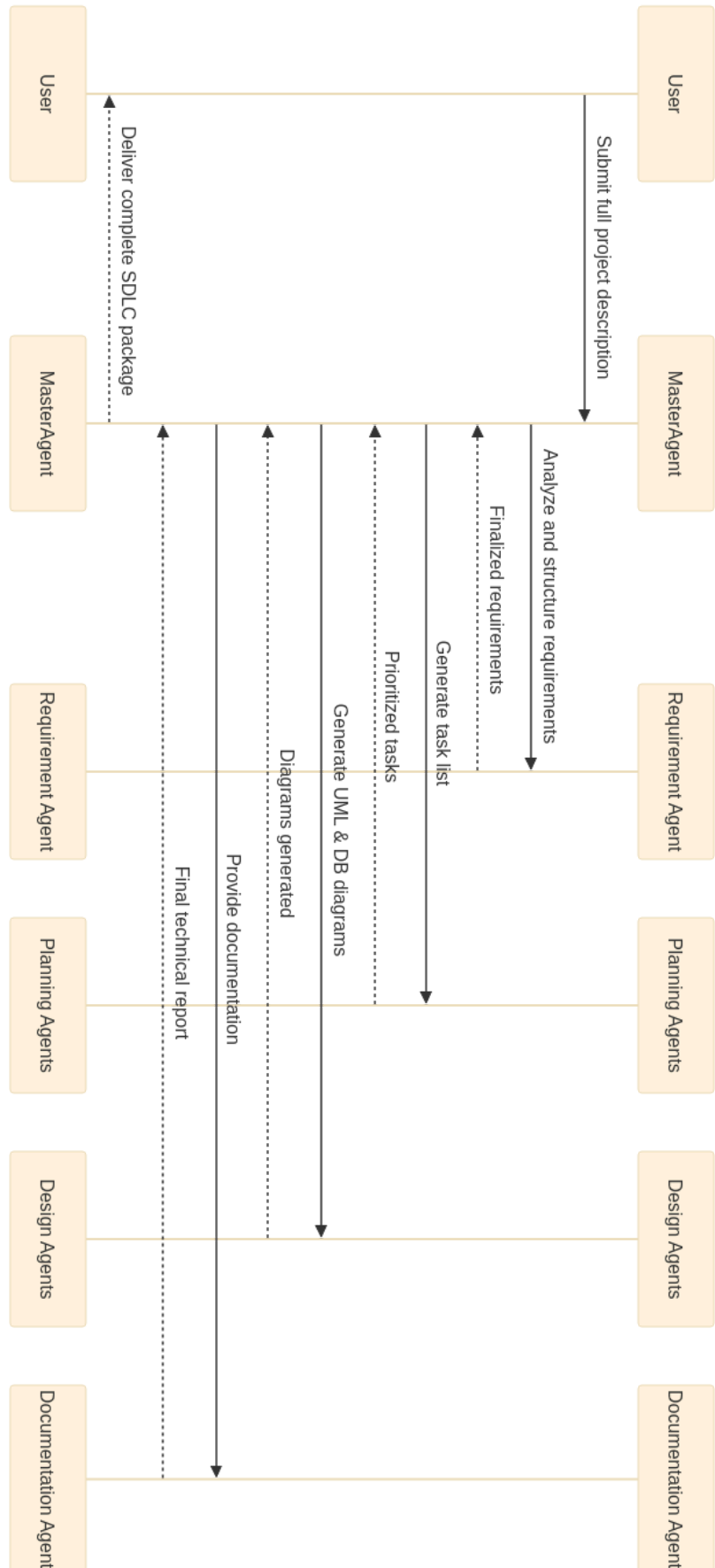
Figure 2. Devistant Class Diagram



3.5.3 Sequence Diagrams

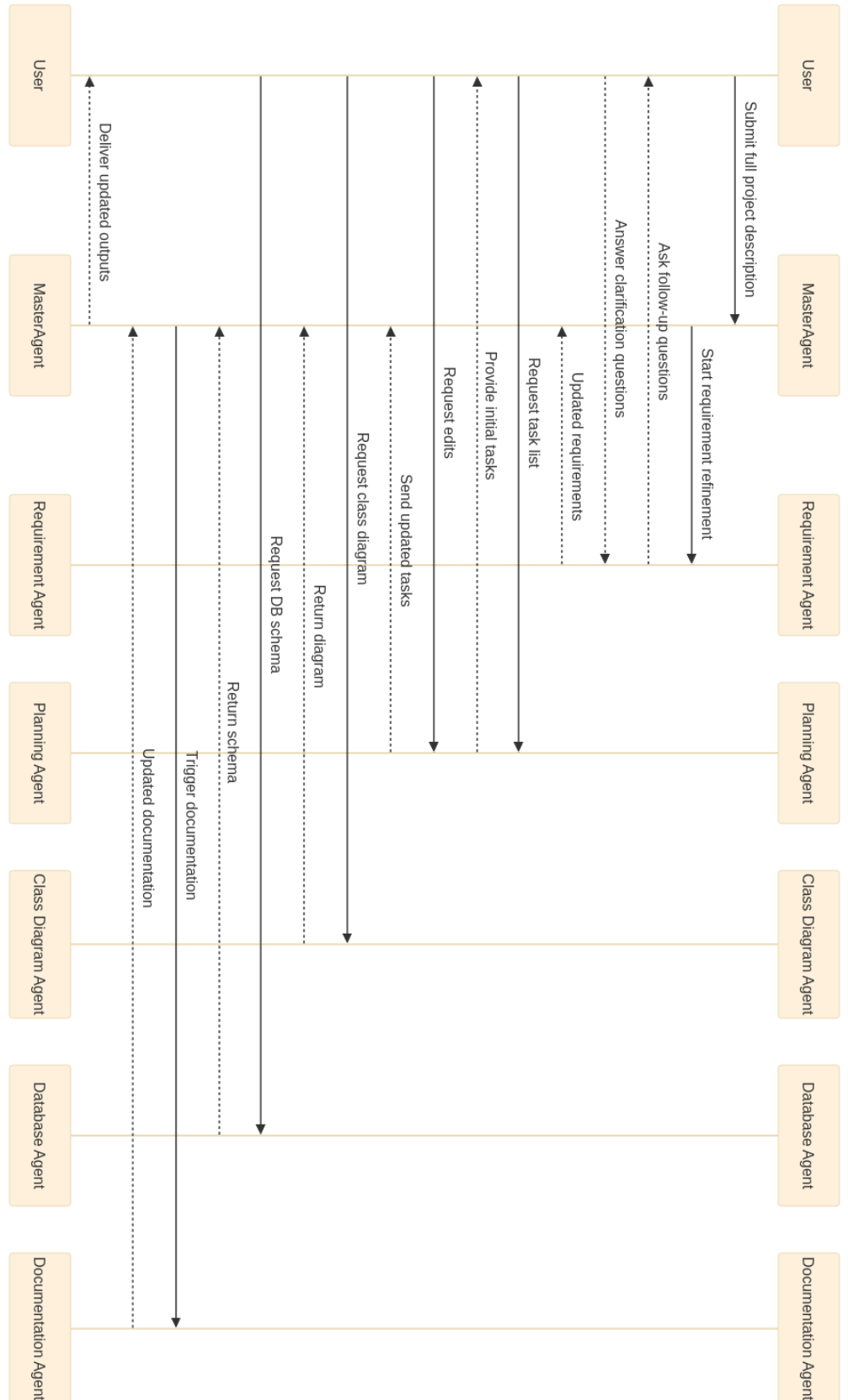
One Shot Version

Figure 3. Devistant Sequence Diagram (One Shot)



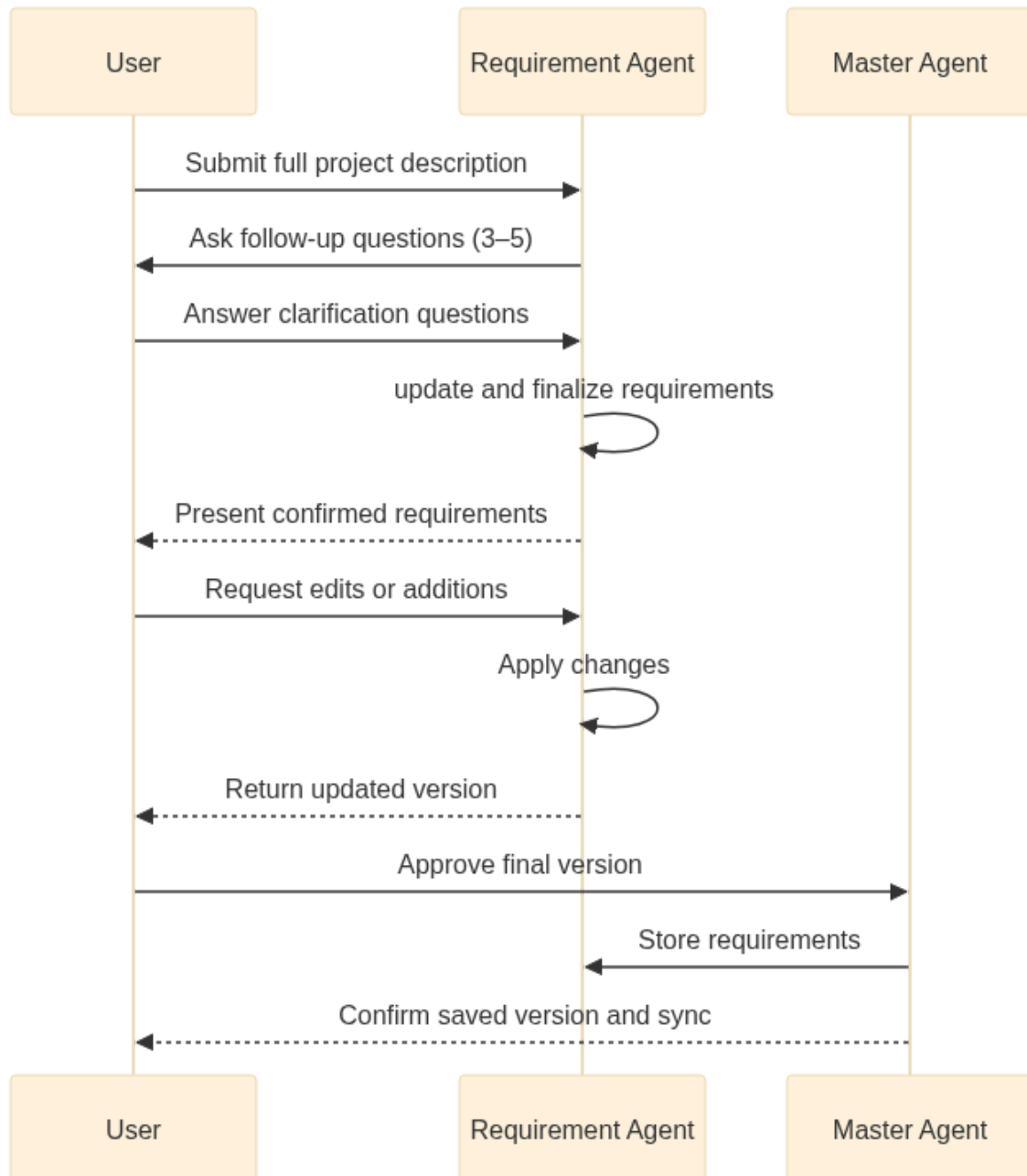
Chat Based Version

Figure 4. Devistant Sequence Diagram (Chat-based)



Sample: Requirement Agent Sequence Diagram

Figure 5. Requirement Analysis Agent Sequence Diagram



Chapter 4 Devistant Design & Implementation

Devistant is built with two versions of operation: One-Shot Version and Chat Version. In One-Shot Version, the system generates a complete SDLC plan based on a single input from the user. In Chat Version, users can work step-by-step, interact with agents, and update parts of the project dynamically. This flexible architecture allows software engineers to choose the version that best fits their workflow. Under the hood, a Master Agent coordinates specialized agents such as “requirements, planning, diagram agents “ using a dependency map to maintain consistency and minimize context window issues in large models.

4.1 System Design

4.1.1 System Architecture Overview

Devistant employs a multi-agent architecture with a central supervisor coordinating specialized agents. The system is built on LangGraph for workflow orchestration and uses the Command Pattern for agent communication.

Core Components:

- **Supervisor Agent:** Central coordinator that routes user requests to appropriate specialized agents
- **7 Specialized Agents:** Requirements Analysis, Planning, Project Management, Use Case, Class Diagram, Database, and Documentation agents
- **State Management System:** Maintains project context and conversation history
- **Memory Management:** Handles persistent storage and context sharing between agents
- **Diagram Generation Engine:** Creates visual Mermaid diagrams for use cases, class structures, and database schemas

Architecture Layers:

- **Presentation Layer:** Gradio web interface for user interaction
- **Orchestration Layer:** LangGraph workflows managing agent execution
- **Agent Layer:** Specialized agents with domain-specific expertise using LangChain
- **Memory Layer:** Context management and persistent storage
- **LLM Integration Layer:** Multiple language model providers (Groq, HuggingFace, Ollama)

4.1.2 One-Shot Execution Flow

In One-Shot Version, the system executes a complete SDLC workflow automatically:

User Input→**Supervisor**→**[Sequential Agent Execution]**→**Final Documentation**

Sequential Flow:

1. **Requirements Analysis** → Extracts and structures project requirements
2. **Planning** → Creates development roadmap and task breakdown
3. **Project Management** → Establishes methodology and timeline
4. **Use Case** → Generates user stories and use case diagrams
5. **Class Diagram** → Designs system architecture and class structure
6. **Database** → Creates database schema and ER diagrams
7. **Documentation** → Compiles comprehensive project documentation

4.1.3 Dependency Mapping

The system implements a sophisticated dependency map, ensuring agents receive relevant context:

```
agent_dependencies = {
  "supervisor": ["all"],
  "requirements_analysis": [],
  "planning": ["requirements_analysis"],
  "project_management": ["requirements_analysis", "planning"],
  "use_case": ["requirements_analysis"],
  "class_diagram": ["requirements_analysis", "use_case"],
  "database": ["requirements_analysis", "class_diagram"],
  "documentation": ["all"]
}
```

Context Flow:

- Each agent receives output from its dependencies
- Project memory maintains shared state across agents
- Context manager handles information sharing and prevents redundancy

4.1.4 Design Patterns Used

1. Command Pattern

- Agent interactions implemented as commands with `Command[Literal[...]]` return types
- Enables undo/redo functionality and request queuing

2. Factory Pattern

- The `AgentFactory` class dynamically creates agent instances
- Supports alternative naming conventions and agent mapping

3. State Pattern

- `MessagesState` and agent-specific state classes manage workflow states
- Enables proper state transitions and persistence

4. Observer Pattern

- Context sharing between agents using a notification system
- Memory manager notifies dependent agents of updates

5. Singleton Pattern

- Global project memory and conversation history
- Single instance of configuration and model managers

4.1.5 Chat-Based Execution Flow

In Chat version, users interact with individual agents through an intelligent routing system:

User Message → Supervisor Analysis → Agent Selection → Agent Response → Context Update

Intelligent Routing Logic:

- Natural language analysis to determine intent
- Agent capability matching
- Context-aware routing based on conversation history
- Fallback routing for edge cases

4.2 System Implementation

4.2.1 Technologies Used

Core Programming Language:

Python 3.13 is the main implementation language chosen for:

- Rich AI/ML ecosystem
- Excellent LangChain integration
- Rapid prototyping capabilities
- Strong community support

Technology Selection Rationale:

- **LangGraph:** Chosen for its state-of-the-art agent orchestration capabilities and LangChain integration
- **Gradio:** Selected for rapid UI development and built-in chat interface components
- **Multiple LLM Providers:** Ensures redundancy and allows model optimization for specific tasks
- **Mermaid.js:** Industry standard for programmatic diagram generation

4.2.2 Agent Implementation Details

Base Agent Architecture: *Python*

class BaseAgent(ABC):

def __init__(self, agent_name: str, agent_role: str, config: Dict[str, Any]):

self.agent_name = agent_name

self.agent_role = agent_role

self.llm = self._create_llm()

self.memory = AgentMemory(agent_name=agent_name)

@abstractmethod

def process_user_input(self, user_input: str, project_id: str) -> str:

pass

Specialized Agent Features:

- **Requirements Analysis Agent:** Interactive Q&A for comprehensive requirements gathering
- **Planning Agent:** Strategic development planning with rationale
- **Project Management Agent:** Methodology selection with justification
- **Use Case Agent:** User story generation and Mermaid use case diagrams
- **Class Diagram Agent:** UML class design with relationship analysis
- **Database Agent:** Schema design with ER diagram generation
- **Documentation Agent:** Comprehensive documentation compilation

LLM Model Assignment Strategy:

- **Groq Models:** Primary for speed and efficiency
- **Hugging Face Models:** for complex reasoning tasks, and the finetuned models we tried using.
- **Ollama Models:** Local execution using models like phi4, llama, qwen, etc.
- **Model Selection:** Based on task complexity and response time requirements

4.2.3 Diagram Generation Using Mermaid.js**Implementation Architecture: *Python***

```
def render_mermaid_diagram(mermaid_code: str, diagram_type: str):
```

```
    """ Renders Mermaid diagram code to PNG and SVG files """
```

```
    # File management
```

```
    output_dir = "diagrams"
```

```
    mmd_file = f"{diagram_type}_diagram.mmd"
```

```
    # CLI rendering with fallback
```

```
    mmdc_paths = ["mmdc", "C:\\Users\\...\\mmdc.cmd", "npx mmdc"]
```

```
    # Generate PNG and SVG outputs
```

```
    subprocess.run([mmdc_cmd, "-i", mmd_file, "-o", png_file])
```

```
    subprocess.run([mmdc_cmd, "-i", mmd_file, "-o", svg_file, "-e", "svg"])
```

Diagram Types Supported:

- **Use Case Diagrams:** graph TB syntax with actor-use case relationships
- **Class Diagrams:** classDiagram syntax with proper UML relationships
- **Database ER Diagrams:** erDiagram syntax with cardinality notation

Generation Process:

- **LLM Generation:** Agents generate Mermaid code using specialized prompts
- **Validation:** Syntax validation and error checking
- **Rendering:** CLI-based conversion to PNG/SVG (using Node.js)
- **Fallback:** Raw Mermaid code output if CLI unavailable
- **Storage:** Files saved in an organized directory structure

Quality Assurance:

- Syntax validation before rendering
- Multiple CLI path attempts for cross-platform compatibility
- Error handling with graceful degradation
- Visual validation through browser preview links

4.2.4 Fine Tuning

Technology Used: Hugging Face Transformers for using & fine-tuning FLAN-T5 model.

The goal of fine-tuning was to:

1. Teach the model how to generate correct Mermaid diagram code (like class diagrams, use case diagrams) from simple text instructions.
2. Make sure the model gives output that can be used directly in Mermaid tools to draw diagrams.

outcomes

1. The model started generating much better Mermaid diagrams after fine-tuning.
2. The diagrams were more correct and ready to render without manual fixes.

example

prompt = "generate mermaid classDiagram for a library system"

print(generate(prompt))

```
class Book {
  +int id
  +String title
}
class Library {
  +int id
  +String name
}
```

Library --> Book : contains

We tried fine-tuning a model to generate Mermaid diagrams, but we decided not to use it because:

- We couldn't find a big enough dataset to train the model well.
- The model's outputs were not good, the diagrams were often wrong or needed fixing.
- The model was slower and less efficient than using LLMs with APIs directly.
- We chose to use LLMs for better and faster diagram generation in Devistant.

4.2.5 User Interface Design

Gradio Interface:

- **Main Chat Window**
 - Large scrollable area displaying conversation between the user and AI agents
 - Shows both user messages and agent responses with clear visual separation
 - Maintains full conversation history during the session
- **Message Input Box**
 - A text area where users type their questions or requests
 - Supports multi-line input for complex queries
 - Auto-focuses for continuous conversation flow
- **Send Button**
 - Primary action button to submit messages
 - Triggers agent processing and response generation
- **Clear Chat Button**
 - Resets conversation history and project memory
 - Starts a fresh session when needed
- **Example Prompts**
 - Pre-written sample questions to guide users
 - Covers common use cases like "build a website" or "create a todo app"
 - Clickable shortcuts for quick interaction
- **Title Header**
 - Displays the system name and capabilities overview
 - Lists main features like requirements analysis, planning, etc
- **Simple User Flow:**

User types question → Clicks Send → AI analyzes and routes to appropriate agent
→ Agent responds → User continues conversation

4.3 Technology Summary

Category	Technology	Purpose
Core Language	Python	Main programming language
AI Framework	LangChain	LLM integration and tools
Workflow Engine	LangGraph	Agent orchestration and routing
User Interface	Gradio	Web-based chat interface
LLM Providers	Groq API	Primary language models
	Hugging Face	Fine Tuned Models
	Ollama	Local model execution
Fine Tuning	FLAN-T5 model	Try specialized methods for generating mermaid code
System Evaluation	LangSmith	Evaluate correctness, conciseness, hallucination
Diagram Engine	Mermaid.js	Visual diagram generation
Diagram CLI	Mermaid CLI	PNG/SVG file conversion
Environment	Python dotenv	Configuration management
Data Validation	Pydantic	Input/output validation
Memory Management	LangChain Memory	Conversation persistence
State Management	TypedDict	Type-safe state handling
Runtime	Node.js + npm	Mermaid CLI dependencies

Key technology Choices:

- **Gradio:** Chosen for rapid deployment and built-in chat components
- **Multiple LLM Providers:** Ensures reliability and cost optimization
- **Mermaid.js:** Industry standard for programmatic diagrams
- **LangGraph:** State-of-the-art agent workflow management

This implementation represents a sophisticated blend of modern AI technologies, thoughtful architectural design, and user-centered interface development, creating a comprehensive tool for software development assistance.

Chapter 5 Devistant Testing and Evaluation

This chapter presents a comprehensive evaluation of the Devistant system to ensure it meets its intended functional and quality requirements. It outlines the testing methodology, tools used, and the results of various system test cases. In addition, the chapter assesses the overall effectiveness of the system from multiple perspectives, including development cost, execution time compared to conventional systems, and its broader environmental, social, and political impacts. This evaluation aims to provide a clear understanding of the system's performance, reliability, and potential real-world influence.

5.1 Testing

To ensure Devistant functions work as expected, a comprehensive testing strategy was adopted. Our strategy includes end-to-end simulation with natural language chat flows, routing logic, validation, memory consistency checks, and system readiness confirmation.

5.1.1 Testing Strategy

The testing approach included:

- **Chat-Based Functional Simulation:** A full conversation was simulated to ensure that agents respond appropriately, build on context, and produce expected artifacts (requirements, plans, diagrams, documentation, etc.).
- **Unit Testing:** Each agent's logic and the supervisor's routing decisions were tested using deterministic messages.
- **Memory Integration Testing:** Tests ensured that the conversation memory and project memory retain and build context across multiple messages.
- **Setup Verification:** Pre-run validation checks for environment variables, required files, and folder structures.

5.1.2 Tools Used

- Python for test scripting
- dotenv for managing API keys
- LangChain and GROQ for LLM-based logic
- print() logs for manual inspection of system responses and flow
- Terminal execution of test scripts for validation

5.1.3 Tests Conducted

Test File	Description	Covered Aspects
run_complete_test.py	Simulates an entire user session from requirements gathering to documentation generation	Functional flow, integration, full conversation
test_conversation_memory.py	Verifies that conversation and project memory persist and evolve correctly	Memory consistency, agent response build-up
test_devistant_chat.py	Tests agent routing, project memory updates, scenario handling	Supervisor logic, project data handling, use case coverage

5.1.4 Chat-Based Scenario Testing

A real conversation was carried out to simulate a client describing a project and receiving responses from the appropriate agents. Example:

- **User Input:** "I want to build a website to sell hoodies online with user registration, shopping cart, and payment processing."
- **Expected Behavior:** The "requirements_analysis" agent responds with 5 guiding questions about the project.
- **Result:** Agent responds with a series of structured questions. "Passed"

Subsequent messages were used to simulate full planning, management, and design flow. Agents routed properly and produced artifacts as expected.

5.1.5 Conversation Memory Testing

Tested via "test_conversation_memory.py":

- Ensured the system builds context across turns
- Verified number of stored messages in conversation_history
- Confirmed requirements are generated only after enough input is gathered

Result: Memory built correctly across 3-step interactions and led to proper requirements generation.

5.1.6 Routing Logic Tests

Tested via “test_devistant_chat.py” using predefined messages and expected agent destinations.

Input Message	Expected Agent	Result
“Create a development plan based on those requirements”	“planning”	Passed
“Create use cases and user stories...”	“use_case”	Passed
“Hello, how are you?”	“__end__”	Passed

Routing logic handled all cases properly.

5.1.7 Setup & Environment Validation

Verified that:

- .env file with GROQ_API_KEY exists
- Essential files like devistant_chat.py and README.md are present
- diagrams/ directory is available or created as needed

System environment setup passed.

5.1.8 Success Criteria

All of the following were verified during testing:

- Requirements agent triggers question-based exploration
- Planning and project management agents justify decisions
- Use case and diagram agents generate correct Mermaid file.
- Conversation memory is retained and correctly used
- Supervisor routes requests to the appropriate agent
- System produces complete documentation in final step

5.1.9 Full Conversation Log

Chat-Based Version

User: “ I want to build a website to sell hoodies online with user registration, shopping cart, and payment processing”

Deviant:

Functional Requirements:

- User registration with email/password authentication and optional social login (Google/Facebook)
- Product catalog with filters/sorting for hoodie styles, sizes, colors, and price ranges
- Shopping cart with add/remove items, quantity adjustments, and cart persistence for logged-in users
- Payment processing integration with Stripe/PayPal for credit card, Apple Pay, and PayPal payments
- Order confirmation system with email receipts and order tracking
- Admin dashboard for managing inventory, orders, and user accounts
- Product review system with star ratings (user-submitted reviews)
- Wishlist functionality to save favorite items
- Discount code/Coupon application during checkout
- Mobile-responsive design for seamless shopping on all device sizes

Non-Functional Requirements:

- Payment system must be PCI-DSS compliant (security)
- Site should load in under 2 seconds on mobile (performance)
- Architecture must scale to 10,000+ concurrent users during sales (scalability)
- 99.9% uptime guarantee for product availability (reliability)
- 100% W3C compliant HTML/CSS for maximum browser compatibility (accessibility)
- User interface must meet WCAG 2.1 AA standards for accessibility (inclusive design)
- Inventory system synchronizes in real-time across all sales channels (consistency)

Would you like to add any specific features like subscription models, personalized recommendations, or social sharing capabilities? Also, do you have any technical constraints (e.g., existing backend systems, preferred programming languages)?

User: “maybe add a loyalty program”

Devistant:

Functional Requirements:

- User registration with email/password authentication and optional social login (Google/Facebook)
- Product catalog with filters/sorting for hoodie styles, sizes, colors, and price ranges
- Shopping cart with quantity management, saved carts, and checkout functionality
- Payment processing integration (e.g., Stripe, PayPal) with secure transaction handling
- Loyalty program system:
 - Points awarded for purchases, referrals, and product reviews
 - Points tracked in user dashboard with redemption options (e.g., \$1 = 10 points)
 - Tiered rewards (e.g., "Silver," "Gold" tiers with exclusive discounts or free shipping)
 - User interface for viewing loyalty points, reward history, and available redemptions
- Order history and tracking for purchased items

Non-Functional Requirements:

- Security: SSL encryption, PCI-DSS compliance for payment processing, secure user authentication
- Performance: Load time <2 seconds on product pages, 99.9% uptime during peak traffic
- Scalability: Support for 10,000+ concurrent users during sales events
- Usability: Intuitive navigation for loyalty program features (e.g., "My Rewards" section)
- Data Management: Secure storage of user points history and redemption logs

Would you like to clarify business goals (e.g., revenue targets, brand positioning) or technical constraints (e.g., platform preference: Shopify, custom-built, etc.) to refine these requirements further?

User: “Now create a development plan based on those requirements”

Devistant:

Development Phases

The development of the e-commerce website will be broken down into the following phases:

1. Phase 1: Planning and Setup (Weeks 1-2)
 - a. Define the project scope, timeline, and budget.
 - b. Set up the development environment, including version control and project management tools.
 - c. Plan the database schema and architecture.
 - d. Determine the technology stack (e.g., frontend framework, backend language, database management system).
2. Phase 2: User Registration and Authentication (Weeks 3-4)
 - a. Implement user registration with email/password authentication.
 - b. Integrate optional social login (Google/Facebook).
 - c. Develop a user dashboard for profile management.
3. Phase 3: Product Catalog and Inventory Management (Weeks 5-8)
 - a. Design and implement the product catalog with filters and sorting.
 - b. Develop the inventory management system, including real-time synchronization across sales channels.
 - c. Integrate a search function for products.
4. Phase 4: Shopping Cart and Payment Processing (Weeks 9-12)
 - a. Implement the shopping cart with add/remove items, quantity adjustments, and cart persistence.
 - b. Integrate payment processing with Stripe/PayPal, ensuring PCI-DSS compliance.
 - c. Develop the order confirmation system with email receipts and order tracking.
5. Phase 5: Loyalty Program and Reviews (Weeks 13-16)
 - a. Implement the loyalty program system, including points tracking, redemption options, and tiered rewards.
 - b. Develop the product review system with star ratings and user-submitted reviews.
 - c. Integrate the wishlist functionality.
6. Phase 6: Admin Dashboard and Analytics (Weeks 17-20)
 - a. Develop the admin dashboard for managing inventory, orders, and user accounts.
 - b. Integrate analytics tools for sales tracking and user behavior analysis.
7. Phase 7: Testing, Optimization, and Deployment (Weeks 21-24)
 - a. Conduct thorough testing, including unit testing, integration testing, and user acceptance testing.

- b. Optimize the website for performance, ensuring load times under 2 seconds on mobile devices.
- c. Deploy the website to a production environment, ensuring 99.9% uptime and scalability.

Rationale Section

This development plan makes sense because it follows a logical sequence of building blocks. The initial phases focus on setting up the foundation (user registration, product catalog, and inventory management), which are essential for the subsequent phases. The shopping cart and payment processing integration are critical components that require a solid foundation. The loyalty program and review system build upon the existing functionality, enhancing the user experience. The admin dashboard and analytics integration provide valuable insights for business decision-making. Finally, thorough testing, optimization, and deployment ensure a high-quality and reliable website.

The phases build upon each other, allowing for incremental development and testing. This approach mitigates risks by identifying and addressing potential issues early on. For example, if the payment processing integration encounters unexpected challenges, it can be addressed before moving on to the loyalty program development.

This approach fits the project's complexity and requirements because it prioritizes the most critical components first. The technology stack will be chosen based on the project's specific needs, ensuring that the selected tools and frameworks support the goals. For instance, a frontend framework like React or Angular may be chosen for its scalability and performance benefits.

Success Factors

This development plan is likely to succeed because it:

- Breaks down the project into manageable phases, allowing for focused development and testing.
- Prioritizes critical components, ensuring a solid foundation for the website.
- Incorporates risk mitigation strategies, such as incremental development and testing.
- Addresses the project's specific challenges, including PCI-DSS compliance, performance, and scalability.
- Chooses a technology stack that supports the project's goals and requirements.

The chosen technology stack will play a crucial role in supporting the project's goals. For example, using a cloud-based platform like AWS or Google Cloud can ensure scalability and reliability. A content management system like WordPress or Shopify can provide a solid foundation for the product catalog and inventory management.

Development plan complete with strategic rationale. Ready for project management phase.

User: “Create a project management plan with sprints and timeline”

Devistant:

Project Management Plan

Project Name: E-commerce Website Development

Methodology Selection:

After analyzing the project requirements, development plan, and client needs, I recommend an Agile methodology for this project. Here's why:

Project Complexity and Uncertainty:

- The project involves developing a complex e-commerce website with multiple features, integrations, and technical requirements.
- There are many unknowns, such as the performance of the payment processing integration, the effectiveness of the loyalty program, and the user adoption of the website.

Team Size and Experience:

- The development team consists of experienced professionals, but there are some new members who require guidance and support.
- The team size is moderate, with around 10-15 members, which allows for effective communication and collaboration.

Client Involvement Needs:

- The client requires frequent updates and progress reports, which are essential for ensuring the project stays on track.
- The client also needs to be involved in testing and validation, which is critical for ensuring the website meets their expectations.

Timeline and Budget Constraints:

- The project has a tight timeline of 24 weeks, with a budget of \$200,000.
- The team needs to work efficiently and effectively to meet the deadlines and stay within budget.

Risk Factors and Change Likelihood:

- There are several risk factors, such as changes in the client's requirements, technical issues, and delays in testing and validation.
- The likelihood of changes is moderate to high, which requires a flexible and adaptive approach.

Justification:

Given these factors, an Agile methodology is the best fit for this project. Agile allows for:

- Flexibility: Agile enables the team to respond quickly to changes and adapt to new requirements.
- Collaboration: Agile fosters close collaboration among team members, stakeholders, and the client, ensuring everyone is aligned and informed.
- Iterative Development: Agile enables the team to develop the website in iterative cycles, allowing for continuous testing, validation, and improvement.
- Timeboxing: Agile allows for timeboxing, which ensures the team stays on track and meets the deadlines.

Implementation Strategy:

To implement Agile, we will use the following approach:

- Sprint Planning: We will plan and prioritize the work for each sprint (2-4 weeks) based on the project requirements and client needs.
- Daily Stand-ups: We will hold daily stand-ups to ensure everyone is aware of the progress, discuss any issues, and make adjustments as needed.
- Sprint Reviews: We will hold sprint reviews to demonstrate the progress, gather feedback, and make improvements.
- Sprint Retrospectives: We will hold sprint retrospectives to reflect on the process, identify areas for improvement, and implement changes.

Timeline:

The project timeline is divided into 7 sprints, with the following milestones:

- Sprint 1: Planning and Setup (Weeks 1-2)
- Sprint 2: User Registration and Authentication (Weeks 3-4)
- Sprint 3: Product Catalog and Inventory Management (Weeks 5-8)
- Sprint 4: Shopping Cart and Payment Processing (Weeks 9-12)
- Sprint 5: Loyalty Program and Reviews (Weeks 13-16)
- Sprint 6: Admin Dashboard and Analytics (Weeks 17-20)
- Sprint 7: Testing, Optimization, and Deployment (Weeks 21-24)

Team Structure and Roles:

The team will consist of the following roles:

- Scrum Master: Responsible for facilitating the Agile process, ensuring the team stays on track, and resolving issues.
- Product Owner: Responsible for representing the client's needs, prioritizing the work, and ensuring the team is aligned with the project goals.
- Development Team: Responsible for developing the website, integrating the features, and testing the website.
- Quality Assurance Team: Responsible for testing the website, identifying issues, and ensuring the website meets the client's expectations.

Success Framework:

To ensure the project's success, we will use the following framework:

- Risk Management: We will identify, assess, and mitigate risks throughout the project.
- Communication: We will maintain open and transparent communication among team members, stakeholders, and the client.
- Collaboration: We will foster close collaboration among team members, stakeholders, and the client to ensure everyone is aligned and informed.
- Continuous Improvement: We will reflect on the process, identify areas for improvement, and implement changes to ensure the project stays on track.

By using an Agile methodology, we can ensure the project stays on track, meets the client's needs, and delivers a high-quality website that meets the project requirements.

User: “Create use cases and user stories and a mermaid use case diagram based on the requirements and the plan”

Devistant:

Use Cases and User Stories

Use Cases

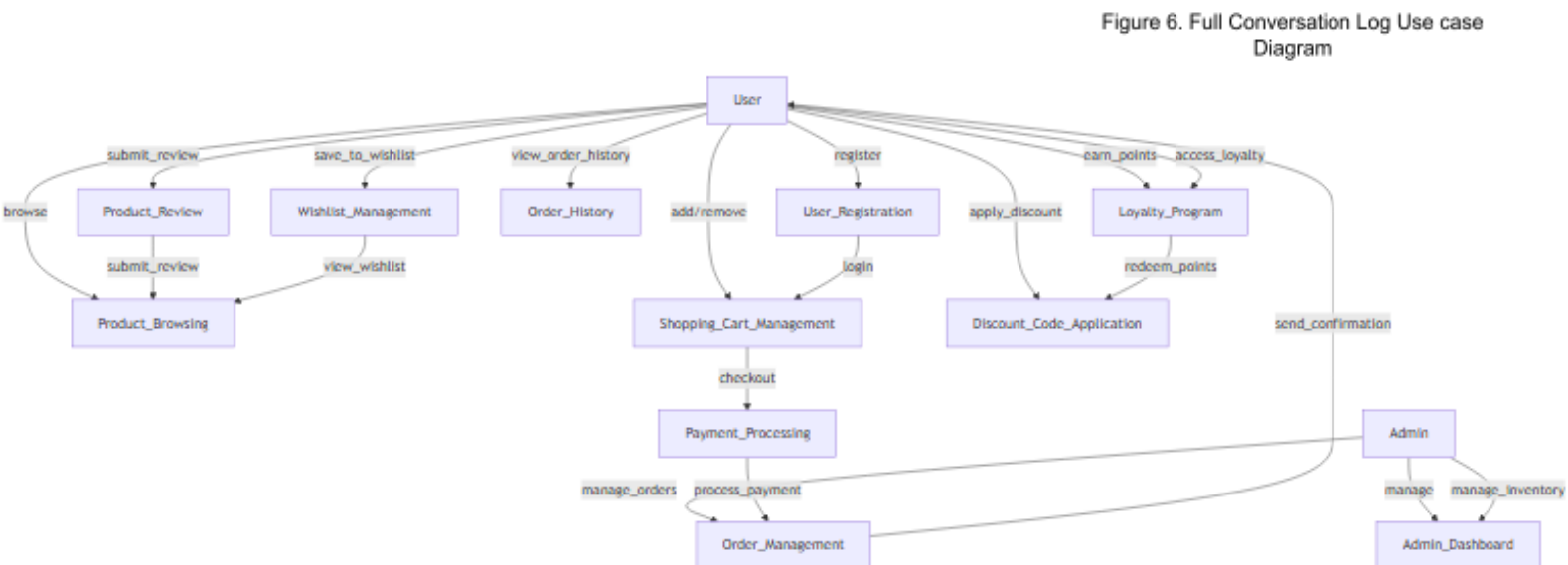
- User Registration: The system allows users to register for an account using email and password, with optional social login (Google/Facebook).
- Product Browsing: The system provides a product catalog with filters and sorting for users to browse hoodies by style, size, color, and price range.
- Shopping Cart Management: The system enables users to add/remove items, adjust quantities, and persist their cart for logged-in users.
- Payment Processing: The system integrates payment processing with Stripe/PayPal, ensuring PCI-DSS compliance, and supports credit card, Apple Pay, and PayPal payments.
- Order Management: The system generates order confirmations with email receipts and provides order tracking for users.
- Admin Dashboard: The system provides an admin dashboard for managing inventory, orders, and user accounts.
- Product Review: The system allows users to submit product reviews with star ratings.
- Wishlist Management: The system enables users to save favorite items to their wishlist.
- Discount Code Application: The system allows users to apply discount codes/coupons during checkout.
- Loyalty Program: The system awards points for purchases, referrals, and product reviews, and tracks user points with redemption options.
- Order History: The system provides users with order history and tracking information.

User Stories

1. As a user, I want to register for an account using email and password, so that I can access my account and order history.
2. As a user, I want to browse products by style, size, color, and price range, so that I can find the hoodie that meets my needs.
3. As a user, I want to add/remove items from my cart, so that I can manage my purchases.
4. As a user, I want to apply discount codes/coupons during checkout, so that I can receive discounts on my purchases.
5. As a user, I want to submit product reviews with star ratings, so that I can share my experience with others.
6. As a user, I want to save favorite items to my wishlist, so that I can easily find them later.
7. As a user, I want to view my order history and tracking information, so that I can stay informed about my purchases.
8. As an admin, I want to manage inventory, orders, and user accounts, so that I can ensure the smooth operation of the website.
9. As a user, I want to earn points for purchases, referrals, and product reviews, so that I can redeem rewards and discounts.
10. As a user, I want to access my loyalty program information, so that I can track my points and redemption options.

This use case diagram illustrates the interactions between the user, admin, and the e-commerce system, highlighting the various use cases and user stories. The diagram provides a visual representation of the system's functionality and the relationships between the different components.

Use Case Diagram (Mermaid)

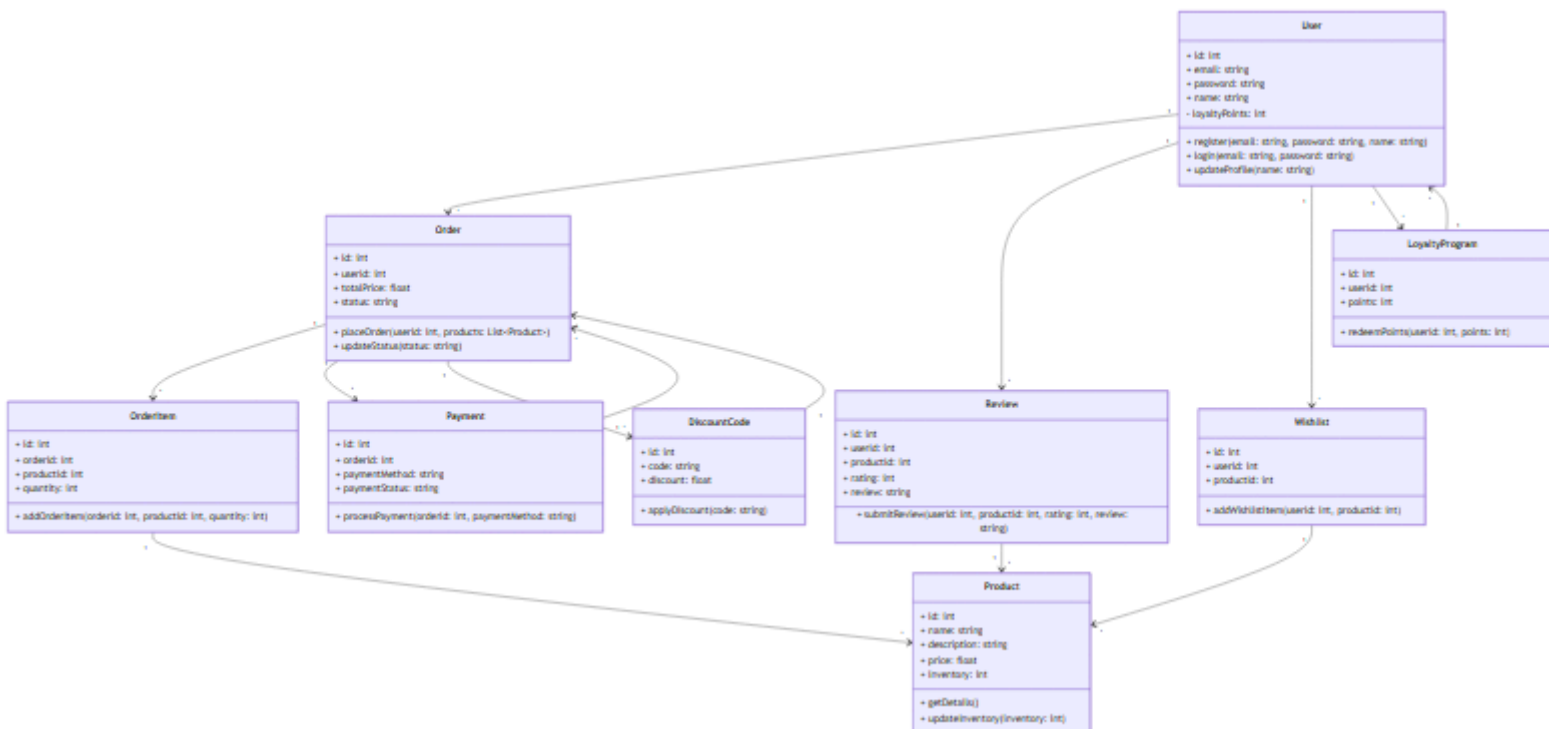


User: “Design the class structure”

Devistant:

Class Diagram

Figure 7. Full Conversation Log Class Diagram



The following UML class diagram represents the system architecture:

Key Design Decisions:

- Clear separation of concerns between classes
- Use of inheritance for code reuse
- Proper encapsulation with public/private attributes
- Well-defined relationships between classes

Class diagram complete. Ready for database design phase.

User: “Design the database schema”**Devistant:**

Database Schema Design

The database schema will be designed using a relational database management system (RDBMS) to support the e-commerce website's functionality. The schema will include the following tables:

Users Table

Attribute Name	Data Type	Description
id	int	Unique user ID (PK)
email	varchar(255)	User email address
password	varchar(255)	User password (hashed)
name	varchar(255)	User name
loyalty_points	int	User loyalty points

Products Table

Attribute Name	Data Type	Description
id	int	Unique product ID (PK)
name	varchar(255)	Product name
description	text	Product description
price	decimal(10, 2)	Product price
inventory	int	Product inventory

Orders Table

Attribute Name	Data Type	Description
id	int	Unique order ID (PK)
user_id	int	Foreign key referencing the Users table (FK)
order_date	datetime	Order date and time
total_price	decimal(10, 2)	Total order price

Order Items Table

Attribute Name	Data Type	Description
id	int	Unique order item ID (PK)
order_id	int	Foreign key referencing the Orders table (FK)
product_id	int	Foreign key referencing the Products table (FK)
quantity	int	Order item quantity

Payments Table

Attribute Name	Data Type	Description
id	int	Unique payment ID (PK)
order_id	int	Foreign key referencing the Orders table (FK)
payment_method	varchar(255)	Payment method (e.g., credit card, PayPal)
payment_date	datetime	Payment date and time

Reviews Table

Attribute Name	Data Type	Description
id	int	Unique review ID (PK)
user_id	int	Foreign key referencing the Users table (FK)
product_id	int	Foreign key referencing the Products table (FK)
rating	int	Review rating (1-5)
review	text	Review text

Wishlists Table

Attribute Name	Data Type	Description
id	int	Unique wishlist ID (PK)
user_id	int	Foreign key referencing the Users table (FK)
product_id	int	Foreign key referencing the Products table (FK)

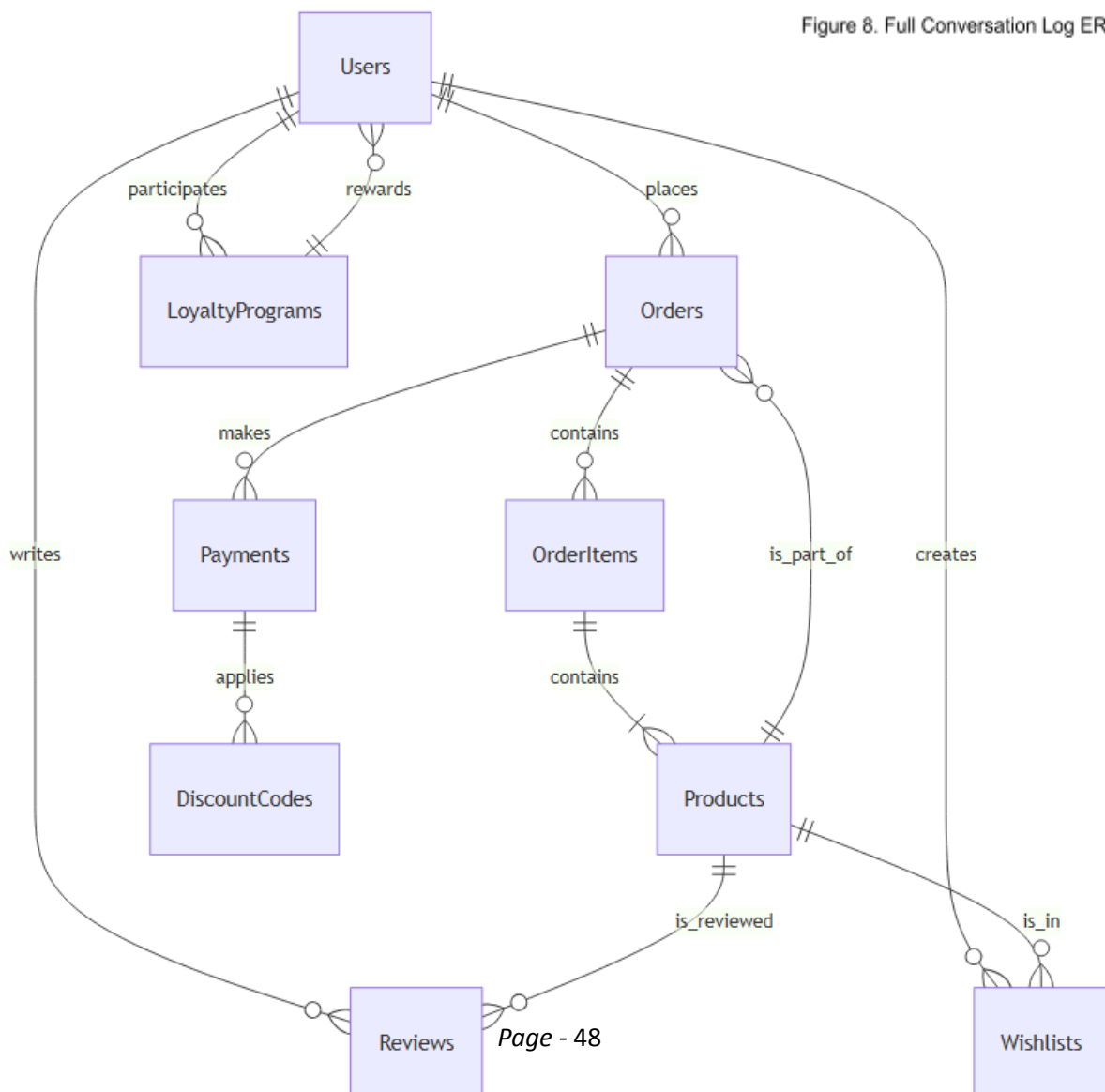
Loyalty Programs Table

Attribute Name	Data Type	Description
id	int	Unique loyalty program ID (PK)
user_id	int	Foreign key referencing the Users table (FK)
points	int	Loyalty points
redemption_history	text	Redemption history

Discount Codes Table

Attribute Name	Data Type	Description
id	int	Unique discount code ID (PK)
code	varchar(255)	Discount code
discount	decimal(10, 2)	Discount amount

ER Diagram



Indexing and Constraints

- Primary keys: id columns in each table
- Foreign keys:
 - + user_id in Orders, Reviews, Wishlists, and LoyaltyPrograms tables
referencing id in Users table
 - + order_id in OrderItems and Payments tables referencing id in Orders table
 - + product_id in OrderItems, Reviews, and Wishlists tables referencing id in Products table
- Unique constraints:
 - + email in Users table
 - + code in DiscountCodes table
- Indexes:
 - + email in Users table
 - + name in Products table
 - + order_date in Orders table

Database design complete. Ready for documentation phase.

User: “Summarize the project and create documentation”

Devistant:

Project Documentation

Project Documentation: Online Hoodie Sales Website

Introduction

This document outlines the scope, objectives, and requirements of the online hoodie sales website project. The project aims to design and develop a user-friendly e-commerce platform for selling hoodies online with user registration, shopping cart, and payment processing capabilities.

Executive Summary

The project involves creating a comprehensive online hoodie sales website that provides a seamless user experience, enables secure transactions, and streamlines inventory management. The website will cater to customers looking to purchase hoodies in various styles, sizes, and colors. To achieve this, the project will leverage industry-standard technologies, best practices, and rigorous testing to ensure a reliable and scalable solution.

Project Overview

The online hoodie sales website project is a web-based e-commerce platform that will enable users to browse, purchase, and manage their orders. The platform will feature user registration with email/password authentication and optional social login, a product catalog with filters and sorting, a shopping cart with item management, and payment processing capabilities. The project will be developed in phases, with a focus on user experience, security, and scalability.

Requirements Summary

A total of 30 requirements have been identified for the project, covering user registration, product catalog, shopping cart, payment processing, and other essential features. These requirements are detailed in the Requirements section of this document.

Scope and Objectives

The scope of this project includes:

- Developing a user-friendly online hoodie sales website with user registration and authentication
- Creating a product catalog with filters and sorting
- Implementing a shopping cart with item management and payment processing
- Ensuring secure transactions and data protection
- Providing a seamless user experience across various devices and browsers

The project objectives are to:

- Design and develop a scalable and secure e-commerce platform
- Ensure a seamless user experience and high customer satisfaction
- Provide a comprehensive product catalog and inventory management system
- Implement robust payment processing and transaction security measures
- Meet the project timeline and budget constraints

This document serves as a comprehensive guide to the project, outlining its scope, objectives, requirements, and technical specifications. It will be regularly updated to reflect changes, additions, and revisions to the project plan.

TECHNICAL IMPLEMENTATION

Development Approach:

Agile Methodology

The Agile methodology will be used for the development of the e-commerce website. The team will follow the Scrum framework, with daily stand-ups, bi-weekly sprints, and regular retrospectives. The development process will be iterative and incremental, with continuous improvement and adaptation to meet the changing requirements.

Tools and Technologies

- Front-end: React (16.13.1) with TypeScript (4.2.3) for building the user interface and user experience.
- Back-end: Node.js (14.17.0) with Express.js (4.17.1) for building the API and handling server-side logic.
- Database: MySQL (8.0.21) for storing and managing data.
- Payment Gateway: Stripe (for payment processing).
- Social Login: Google OAuth (for Google login) and Facebook OAuth (for Facebook login).

System Architecture

The system architecture will be designed to be scalable, maintainable, and secure. The architecture will consist of the following layers:

1. Presentation Layer: This layer will handle user input and output, using React for the front-end.
2. Application Layer: This layer will handle business logic, using Node.js and Express.js for the back-end.
3. Database Layer: This layer will handle data storage and retrieval, using MySQL as the database management system.

4. Integration Layer: This layer will handle integration with external services, such as the payment gateway and social login services.

Database Design

The database schema will be designed to support the e-commerce website's functionality. The schema will include the following tables:

Users Table

Attribute Name	Data Type	Description
id	int	Unique user ID (PK)
email	varchar(255)	User email address
password	varchar(255)	User password (hashed)
name	varchar(255)	User name
loyalty_points	int	User loyalty points

Products Table

Attribute Name	Data Type	Description
id	int	Unique product ID (PK)
name	varchar(255)	Product name
description	text	Product description
price	float	Product price
inventory	int	Product inventory

Orders Table

Attribute Name	Data Type	Description
id	int	Unique order ID (PK)
user_id	int	Foreign key referencing the users table
order_date	datetime	Order date and time
total	float	Order total

Implementation Guidelines

Front-end Development

- Use React hooks for state management and lifecycle methods.
- Use TypeScript for typing and code analysis.
- Use CSS-in-JS for styling.
- Use a front-end build tool, such as Webpack or Rollup, for bundling and optimization.

Back-end Development

- Use Node.js and Express.js for building the API.
- Use a back-end build tool, such as Webpack or Gulp, for bundling and optimization.
- Use a package manager, such as npm or yarn, for dependency management.

Database Development

- Use MySQL for database management.
- Use a database migration tool, such as Sequelize or TypeORM, for database schema management.

Testing and Deployment

- Use Jest or Mocha for unit testing and integration testing.
- Use a testing framework, such as Jest or Mocha, for end-to-end testing.
- Use a continuous integration and continuous deployment (CI/CD) tool, such as Jenkins or Travis CI, for automating testing and deployment.

Security

- Use HTTPS for secure communication between the client and server.
- Use SSL/TLS certificates for encrypting data in transit.
- Use authentication and authorization mechanisms, such as JSON Web Tokens (JWT) or OAuth, for securing user access.

API Documentation

The API will be documented using Swagger or OpenAPI, with endpoints for user registration, login, product browsing, shopping cart management, and payment processing.

Code Quality and Best Practices

The code will be written in accordance with best practices, including:

- Following the principles of SOLID design.
- Using meaningful variable and function names.
- Using comments and documentation to explain code intent.
- Following a consistent coding style.
- Using a linter and code formatter to enforce coding standards.

Continuous Integration and Continuous Deployment (CI/CD)

The CI/CD pipeline will be automated using a tool, such as Jenkins or Travis CI, to ensure that code changes are tested and deployed to production regularly.

Monitoring and Logging

The system will be monitored and logged using tools, such as Prometheus or Grafana, to ensure that issues are detected and resolved promptly.

Scalability and Performance

The system will be designed to scale horizontally and vertically, using load balancers and auto-scaling to ensure that the system can handle increased traffic and usage.

Backup and Recovery

The system will be backed up regularly, using a tool, such as mysqldump or pg_dump, to ensure that data is preserved in case of a disaster.

Compliance

The system will comply with relevant regulations, such as PCI-DSS and GDPR, to ensure that data is handled and stored securely.

Note: This is a high-level technical implementation guide and may require further clarification or details based on specific requirements and constraints.

SUMMARY

Project Status

The Devistant project has been successfully completed with all six phases - Requirements, Planning, Project Management, Use Cases, Class Diagram, and Database Design - successfully completed. The project has been thoroughly documented and tested to ensure that all requirements have been met.

Deliverables Summary

The following deliverables have been produced as a result of this project:

- Requirements Document: Outlining the functional and non-functional requirements of the Devistant system.
- Project Plan: Detailing the project timeline, milestones, and resources.
- Project Management Plan: Describing the project management approach and governance structure.
- Use Cases Document: Describing the user interactions with the system and the system's responses.
- Class Diagram: Visualizing the system's object-oriented design and architecture.
- Database Design Document: Outlining the database schema and data relationships.

Next Steps

Following the completion of this project, the next steps are:

1. **Development Phase:** The development team will start building the system based on the designs and documentation produced in this project.
2. **Testing and Quality Assurance:** The system will undergo thorough testing and quality assurance to ensure that it meets all the requirements and is free of defects.
3. **Deployment:** The system will be deployed to the production environment, and users will be trained on its operation.
4. **Maintenance and Support:** The system will be maintained and supported to ensure that it continues to meet the changing needs of the users.

Conclusion

The Devistant project has been a successful endeavor, producing a comprehensive set of deliverables that will form the foundation of the system's development. The next steps will focus on building, testing, and deploying the system, ensuring that it meets the requirements and expectations of the users.

Appendix: Generated Artifacts

- Diagrams saved in ./diagrams/ folder
- Use Case Diagram: use_case_diagram.png/svg
- Class Diagram: class_diagram.png/svg
- Database ER Diagram: database_diagram.png/svg

Documentation generated by Devistant AI Development Assistant

“All Diagrams saved in Diagrams Folder in .svg & .png & .mmd”

Access the folder by scanning QR code



5.1.10 Summary of Testing Outcomes

- **Test Coverage:** Full flow tested from input to final output
- **Failures:** None detected during automated or manual testing
- **Artifacts Generated:** Use case diagrams, class diagrams, database schema, and project documentation and saved in the Diagrams directory.
- **Readiness:** System is validated and ready for deployment

5.2 Evaluation

We used **LangSmith** in conjunction with Groq-hosted **LLM-as-judge** models (LLaMA 3-70B) to evaluate the performance of the **Devistant** assistant across three core quality metrics:

- Correctness (task intent match)
- Hallucination (factual grounding)
- Conciseness (brevity and clarity)

Each test involved simulated user queries (e.g., “create a development plan”), and the system response was evaluated against expected behavior. Multiple iterations were conducted (~6 experiments) with real-time agent interaction and isolated conversation memory.

5.2.1 Test Metrics Summary

Metric	Observed Range	Notes & Interpretation
Correctness	0.67 → 1.00	Mostly accurate responses with correct task handling. Assistant usually fulfills the intent.
Hallucination	0.67 → 1.00	Rare minor inaccuracies; overall grounded content.
Conciseness	0.00 → 0.33	Tends to be verbose; needs post-processing.
P50 Latency	3.82s → 9.92s	Acceptable for non-real-time tools.
P99 Latency	7.77s → 54.41s	Some agent chains introduce latency spikes.
Tokens	2400 → 4,000	moderate to high response length per task.

All test cases used an isolated chat history per interaction to ensure reproducibility.

5.2.2 Tools Used

- **LangSmith:** Evaluation harness and LLM-judge tooling
- **Groq + LLaMA 3-70B:** Evaluation models for correctness, hallucination, and conciseness
- **Python + LangChain + LangGraph:** Orchestration and agent supervisor logic
- **Mermaid.js:** For rendering agent-generated ER diagrams
- **Transformers:** For internal token usage tracking

5.2.3 Evaluation Insights

- **Correctness** was consistently high, showing that the multi-agent routing logic and specialized agents (requirements, planning, database) are effectively aligned with their tasks.
Conciseness scored low due to detailed, sometimes excessive output. This can be mitigated by trimming verbose agent responses.
- **Latency** is acceptable for interactive use cases but needs optimization for real-time assistants or mobile integrations.

5.2.4 Cost, Time, and Impact Evaluation

- **Cost (Approximate):** The experiment was conducted using the **Groq API**, which offers highly efficient inference at low cost. For ~6 runs totaling ~12,000 tokens, the estimated cost was **under \$1 USD ~ 50 EGP** using current Groq pricing.
 - Groq Playground provides a free tier quota to use some of its models for free.
 - LangSmith provides a free tier that covers the evaluation volume.
- **Time Compared to Traditional Systems:** **Devistant's** agent-based task decomposition provides faster planning and requirement coverage than typical human-assisted documentation workflows.

A manual requirement-documentation-planning-DB design cycle typically takes **several hours to days**. **Devistant** achieved this in **< 1 minute per case**.

- **Environmental Impact:** By using optimized inference (Groq), **Devistant** minimizes energy usage compared to larger models or GPU-heavy services. Evaluations were performed locally or in cloud environments with efficient compute.
- **Social & Political Impact:** **Devistant** automates planning, technical documentation, and architectural design. This:
 - Empowers small teams/startups with rapid prototyping tools.
 - Reduces reliance on expensive consultants or engineers for early-stage planning.
 - May reshape expectations around who can initiate and plan software systems, enabling broader innovation access.

Chapter 6 Conclusions and Future Work

6.1 Conclusion

The goal of this project was to build **Devistant**, a multi-agent software engineering assistant capable of automating and streamlining the Software Development Life Cycle (SDLC) using Large Language Models (LLMs), LangChain, and LangGraph. The system successfully addresses major pain points in traditional software engineering workflows, including fragmented tools, manual diagram generation, and inconsistent documentation.

Through the use of a **Supervisor Agent** and several specialized agents (e.g., requirements analysis, planning, project management, class diagram generation, database modeling, and documentation), Devistant enables both **one-shot SDLC generation** and **interactive, chat-based workflows**. This allows for rapid prototyping, iterative design, and consistent documentation within a unified platform.

System testing confirmed that Devistant meets its primary objectives. Agents responded correctly to user prompts, maintained context across long conversations, and generated coherent outputs aligned with user intent. Evaluations using LangSmith and LLM-as-a-judge demonstrated high correctness and low hallucination, affirming that the system performs reliably within its defined scope.

Ultimately, Devistant demonstrates that **AI-driven automation in software engineering is not only feasible but also practical**, providing tangible benefits in speed, accuracy, and collaboration compared to traditional methods.

6.2 Future Work

While Devistant has achieved its initial goals, several opportunities for improvement and expansion remain

6.2.1 Future work on Devistant

1. **Integration with External Development Tools :** Currently, Devistant operates in isolation. Integrating with platforms such as **Jira, GitHub, GitLab, or VS Code** would enable seamless transitions between AI-generated outputs and live development environments. This would allow real-time task tracking, version control, and collaboration.
2. **Programming & Code Generation Agents:** A major future enhancement is the inclusion of **programming agents** capable of:
 - Generating backend and frontend code (e.g., Flask, Django, React)
 - Creating RESTful APIs and route handlers based on class diagrams.
 - Producing test cases (unit, integration, end-to-end) for each requirement
 - Performing static code analysis and suggesting improvements

These agents would be tightly coupled with the existing planning, class diagram, and database agents, ensuring that generated code is **fully synchronized** with system design and requirements.

3. **Customizability and Personalization:** To make Devistant adaptable across different use cases and industries, future versions should allow users to:
 - Choose from **custom tech stacks** (e.g., Node.js vs. Python, SQL vs. NoSQL)
 - Define **project templates or presets** for recurring system types (e.g., e-commerce, CMS)
 - Set **custom formatting rules** for documentation and diagrams
 - Personalize interaction tone (e.g., formal, technical, beginner-friendly)

This would turn Devistant into a more versatile, user-driven assistant suitable for various development styles.

4. **Multi-User Real-Time Collaboration:** Devistant currently supports a single user session. Enabling **real-time multi-user collaboration** (e.g., multiple engineers reviewing and editing the same plan simultaneously) would bring it closer to enterprise-grade tools like Confluence or Google Docs but with AI- capabilities.
5. **Multi-Language Support (Natural and Programming):** Future versions can support:
 - Support **international languages** in the user interface (e.g., Arabic, French)
 - Allow users to choose **different programming languages** for code outputs
 - Automatically translate documentation to the user's preferred language
6. **Mobile and Lightweight Versions:** Developing a **mobile-friendly UI** or lightweight PWA (Progressive Web App) would improve accessibility for users working remotely or needing quick updates on the go.
7. **Enhanced Diagram Editing Interface:**

While Mermaid.js provides a fast rendering layer, adding a **drag-and-drop visual diagram editor** with real-time sync to Mermaid code would improve usability for non-technical users or visual-first thinkers.

8. **Advanced Testing and Debugging Agents:** Introduce dedicated agents for:
 - Test case generation and test coverage analysis
 - Bug detection based on code outputs
 - Continuous integration setup and simulation execution

This would shift Devistant from a planning/design tool to a **full development assistant** that supports post-design activities.

9. **Custom Model Fine-Tuning:** While general-purpose LLMs have served well, fine-tuning on **software engineering-specific datasets** or domain-specific corpora could improve the precision and efficiency of each agent particularly in code generation, architecture enforcement, and design validation.

10. Auto-Refactoring and Reverse Engineering Agents: Additional agents could allow Devistant to:

- Reverse engineer existing codebases into updated requirements, diagrams, and docs.
- Refactor legacy code into modern patterns aligned with best practices.

6.2.2 Vision

With these enhancements, Devistant can evolve from a smart assistant into a collaborative, full-lifecycle development companion capable of not only planning and designing but also building, testing, and adapting software in real time, tailored to each user's context.

6.2.3 Closing Remark

Devistant demonstrates a promising path forward for **AI-assisted software engineering**. By combining intelligent orchestration, modular agent design, and real-time adaptability, it sets the stage for a new generation of developer tools that are not only smarter but also more collaborative, scalable, and aligned with modern agile workflows.

6.3 Lessons Learned

Throughout the development of Devistant, this project has provided a hands-on, multidisciplinary learning experience that bridged theory with real-world AI applications. As the system evolved from a concept into a functioning multi-agent software assistant, a number of technical and conceptual lessons were gained

6.3.1 Lessons Learned

1. **Prompt Engineering & Prompting Techniques:** Building an AI system centered around LLMs revealed the importance of **prompt design** in guiding model behavior. Careful experimentation with few-shot, role-based, and chain-of-thought prompts improved accuracy, structure, and task alignment. Crafting prompts that balance creativity with constraint became a core competency.
2. **Understanding and Using LLMs Effectively:** This project deepened our understanding of **Large Language Models (LLMs)**, including their capabilities, limitations, and how to harness them effectively for different stages of the Software Development Life Cycle. We also explored trade-offs between general-purpose LLMs and specialized models.
3. **LangChain & LangGraph Frameworks:** Implementing Devistant introduced us to **LangChain**, which helped in building agent pipelines, managing memory, and interacting with models, and **LangGraph**, which offered a powerful, event-driven framework for orchestrating multi-agent workflows. These tools taught us how to structure conversational flows, state machines, and dependency-based systems in modern AI apps.
4. **LangSmith for Evaluation:** LangSmith provided a practical solution for **debugging and evaluating agent outputs**. We learned how to measure correctness, hallucination, and conciseness using LLM-as-a-judge techniques. This helped bring transparency to how well each agent performed and where improvements were needed.
5. **Multi-Agent Architecture & System Design:** Designing a **multi-agent system** taught me how to divide responsibilities across specialized components and synchronize them via a **supervisor agent** and shared memory. We explored

different patterns for communication, dependency mapping, and state consistency across agents.

6. **ML & AI System Design:** The full development cycle from requirement gathering to architecture, design, testing, and evaluation taught us how to structure a modern **AI-enhanced ML system**, considering trade-offs like token limits, context window, memory strategies, modularity, and human oversight.
7. **Agent Tools & Automation:** We implemented custom **agent tools** and functions to support automation within the chat-based workflow. The value of **human-in-the-loop** design became clear, especially when enabling iterative refinement or live corrections by the user.
8. **Frontend Development (Gradio):** While building the user interface for Devistant, I gained experience with **Gradio**, learning how to quickly deploy interactive chat UIs, render diagrams, and build visual workflows for technical users.
9. **Versioning & Artifact Management:** Tracking outputs from different agents (e.g., use case diagrams, class diagrams, documentation) emphasized the need for **versioning**, reproducibility, and traceability skills critical for managing large projects and collaborative environments.
10. **Real-World Software Engineering:** Finally, this project provided practical experience with:
 - a. Building automation for software design tasks
 - b. Designing scalable systems with **human-AI collaboration** in mind
 - c. Handling limitations of LLMs and designing fallback or refinement mechanisms

6.3.2 Reflection

This project was not only about implementing a system, it was about **learning how to think like a Data Scientist and an AI engineer**, balancing abstraction with practicality, and designing tools that enhance human capabilities without replacing them. It highlighted the power of modularity, orchestration, and conversational interfaces in shaping the future of software development.