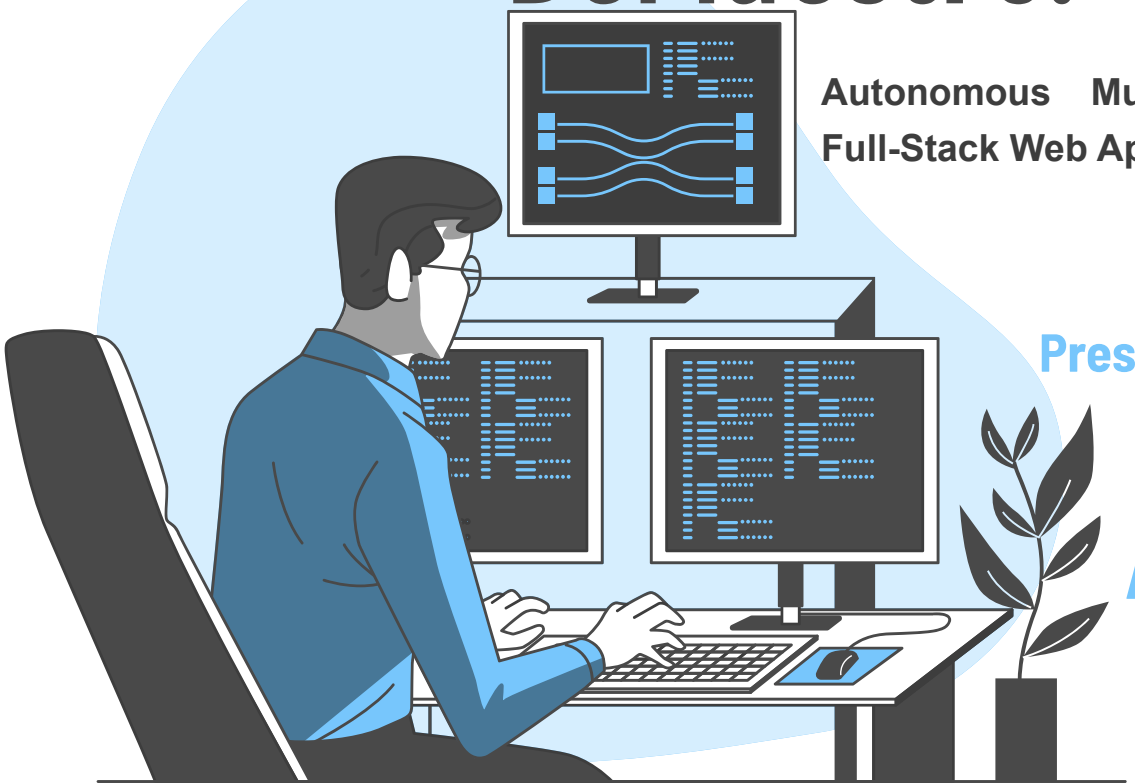
BRAUDE
College of Engineering, karmiel

DeMaestro:



Autonomous Multi-Agent System for Full-Stack Web Application Synthesis

26-1-D-25

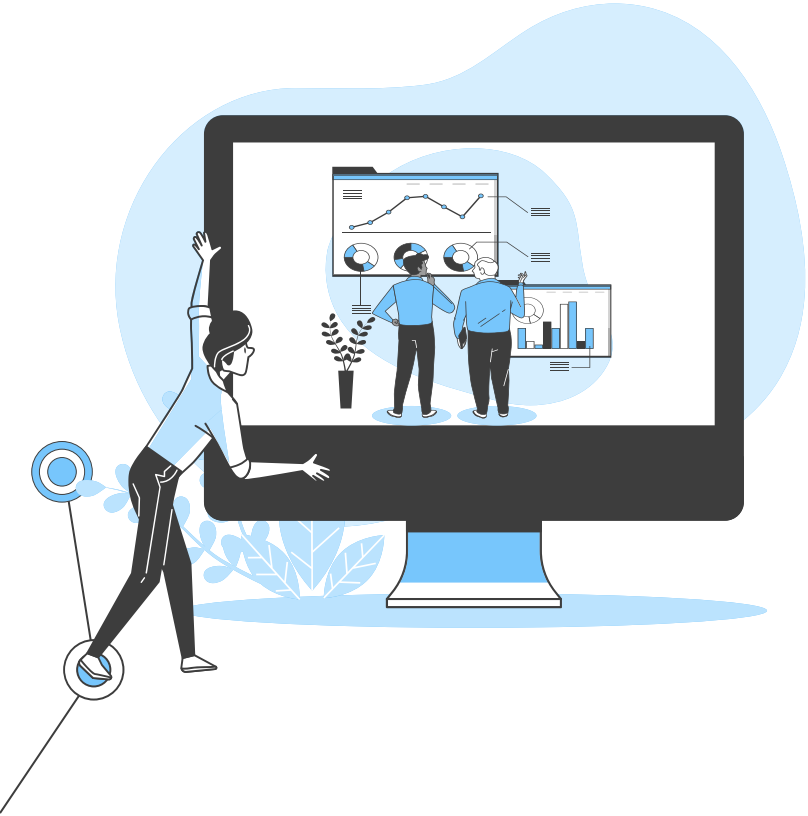
Presented by:

Owise Zoubi
Mohammed Atamneh

Advisor:

Natali Levi

Introduction & Motivation



Developing a full-stack web application requires knowledge of multiple technologies, including frontend development, backend logic, databases, and system integration. For students, beginners, and non-expert users, transforming an idea into a runnable application is often difficult and time-consuming. Even experienced developers spend significant effort on project setup and configuration before actual development begins. These challenges create high technical barriers and slow down innovation.

Problem Definition

The main problem addressed in this project is the difficulty of converting informal user requirements into a complete and runnable full-stack web application. User requirements are typically expressed in natural language, which is often ambiguous or incomplete, making automated generation unreliable.

- Ambiguous and incomplete user requirements
- No automatic clarification or validation
- Weak connection between requirements and code
- High technical barrier for non-expert users

Related Work

- **Approach: "One-Shot" Generation (Prompt to Code).**
- **Pros: Generates beautiful User Interfaces (UI) instantly.**

Limitations:

- **No Clarification: It guesses requirements instead of asking.**
- **Hallucinations: Often generates broken backend logic or fake database connections.**
- **Hard to Scale: Great for prototypes, but hard to turn into a real, secure application.**

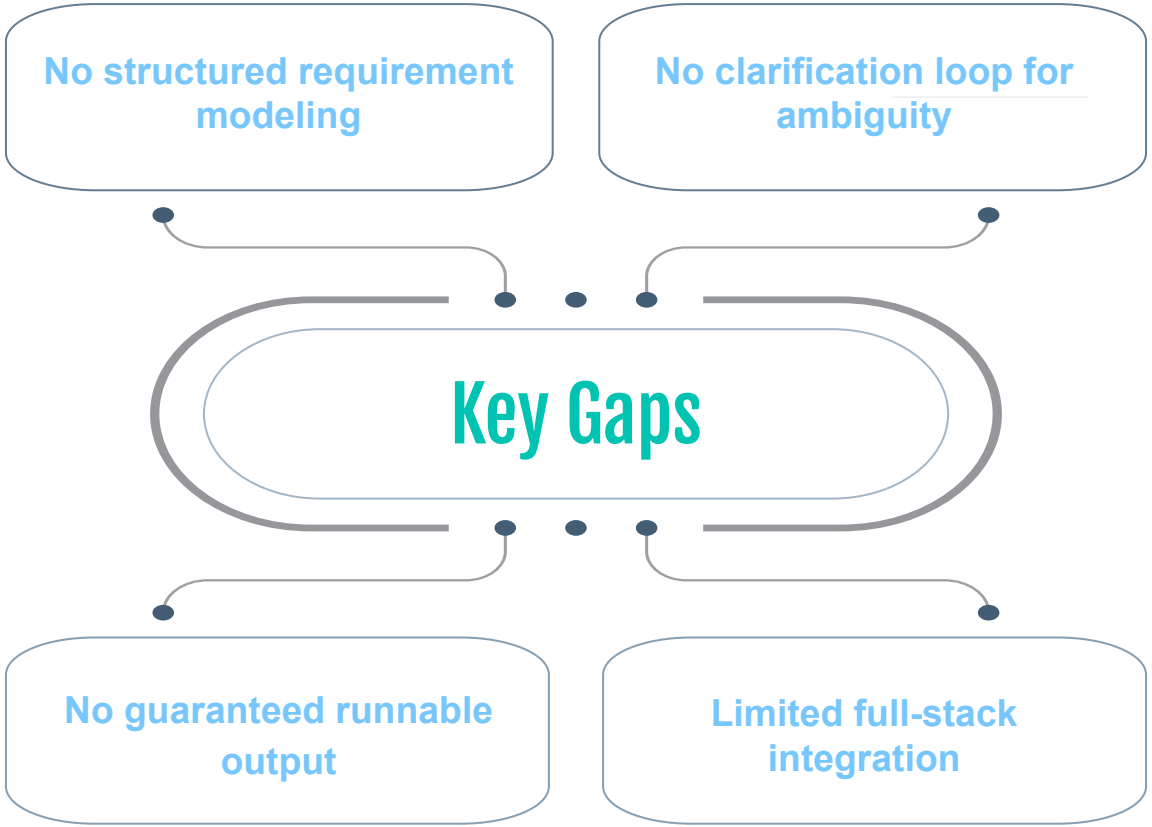
Related Work

- Allows application creation without writing code
- Visual drag-and-drop interface
- Fast prototyping for simple applications

Limitations:

- Limited flexibility for complex systems
- Difficult to export clean, full source code
- Strong platform dependency (vendor lock-in)
- Less control over backend logic and architecture

Gap Analysis

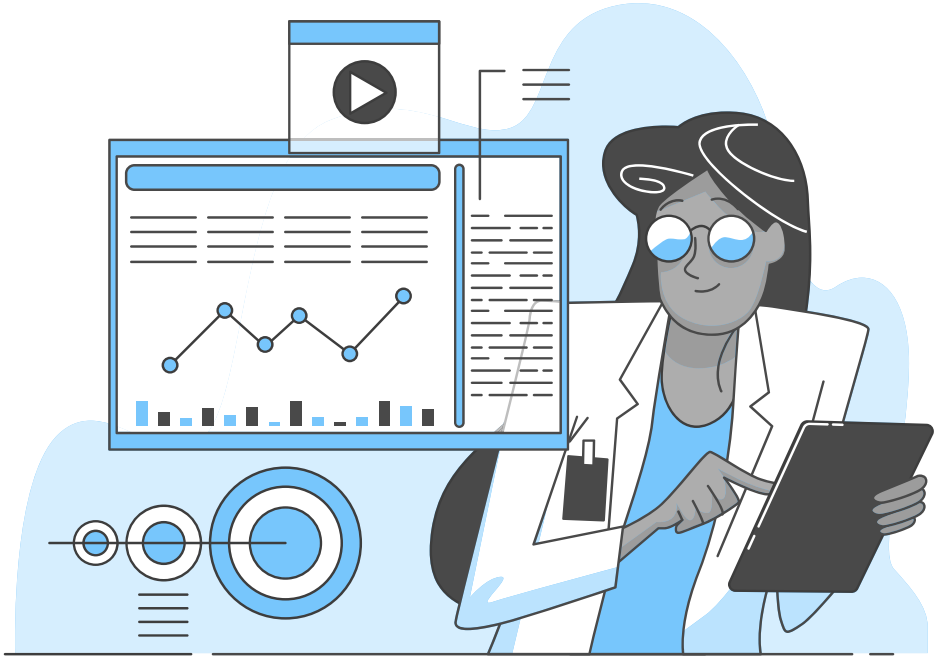


Proposed Solution

- DeMaestro is a role-based AI system for generating full-stack web applications
- Uses a single LLM (Google Gemini) performing multiple logical roles
- Follows a structured, multi-stage generation pipeline
- Includes requirement analysis and clarification before code generation
- Uses a fixed technology stack for consistency
- Separates user interface logic from AI processing
- Aims to produce complete and runnable source code

PROJECT OBJECTIVES

1. Transform user requirements into full-stack source code
2. Support frontend, backend, and database generation
3. Handle unclear requirements using clarification questions
4. Provide project history per user
5. Deliver exportable, runnable application code



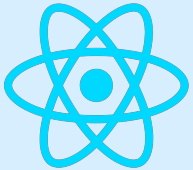
Technology Stack

DeMaestro utilizes modern technologies to ensure reliability, scalability, and efficient AI-driven application generation

Backend

Python

Frontend



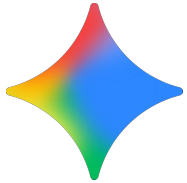
React.js

Tailwind CSS

Database & Cloud Services

Firebase (Auth + Firestore)

AI Model



Google Gemini API

System Requirements

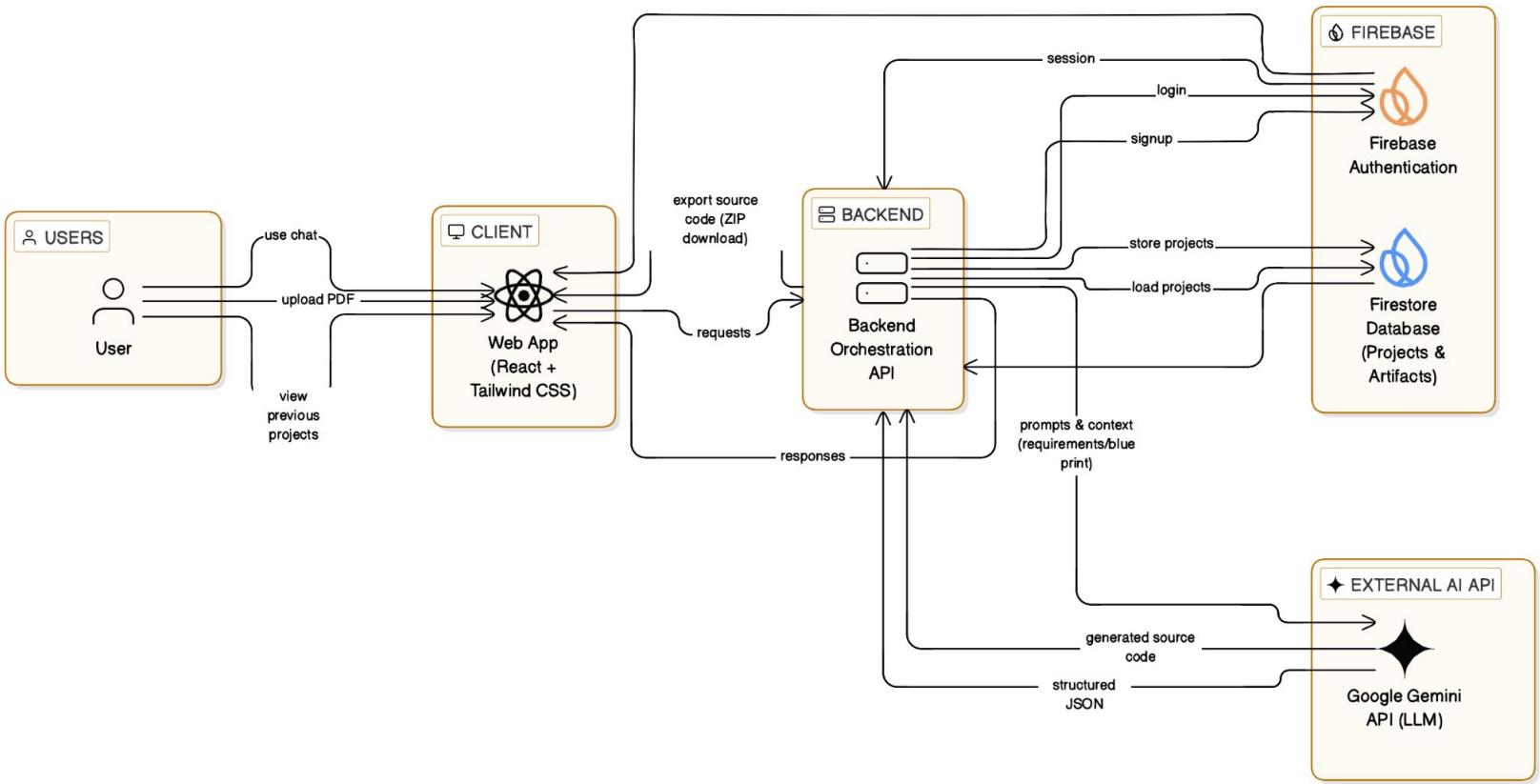
Functional Requirements (FR)

- Accept requirements by **chat** or **PDF upload**
- Perform **clarification loop** when requirements are unclear
- Export **downloadable ZIP** of the generated project
- Support **signup/login** and **project history**
- Generate **blueprint + full-stack source code** (frontend + backend + DB)

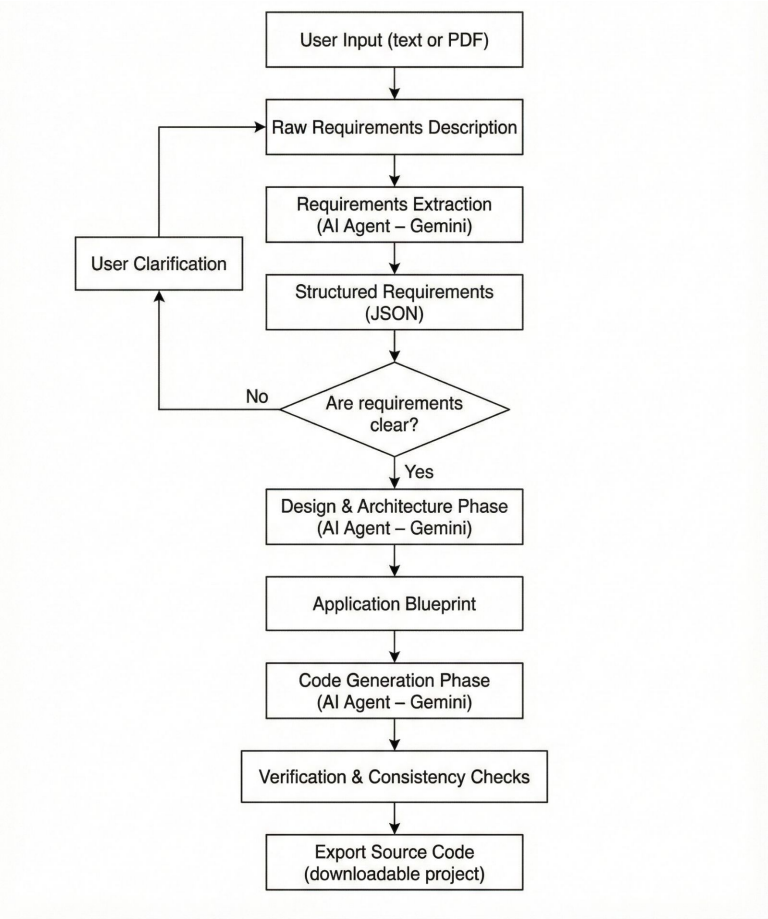
Non-Functional Requirements (NFR)

- **Security:** users can access only their own projects
- **Reliability:** most generated projects run locally without build/syntax errors
- **Usability:** user can submit requirements within ~60 seconds
- **Performance:** generation completes within ~5 minutes on average
- **Traceability:** logs show completed stages and stored artifacts

System Architecture Overview

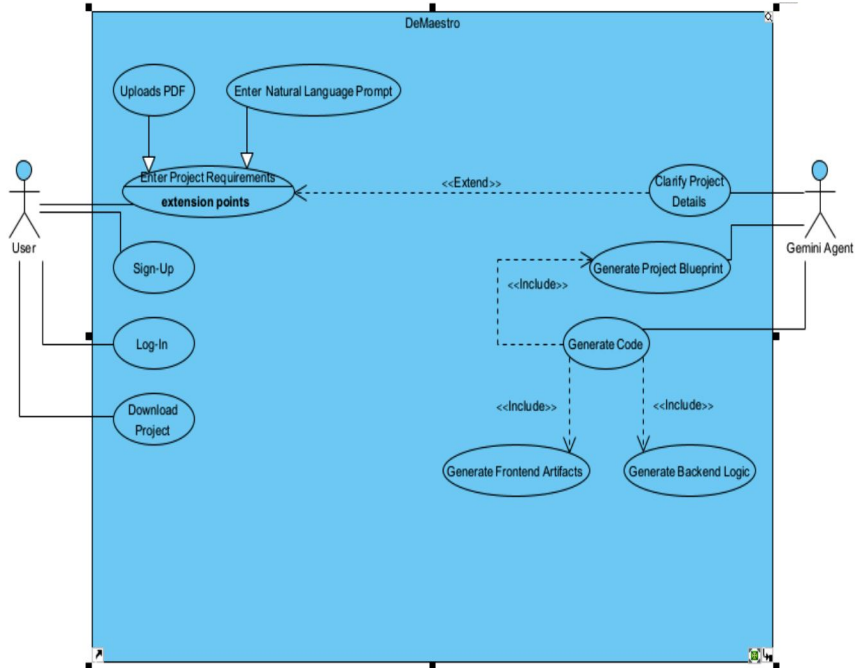


AI Agent Workflow (Gemini Process)

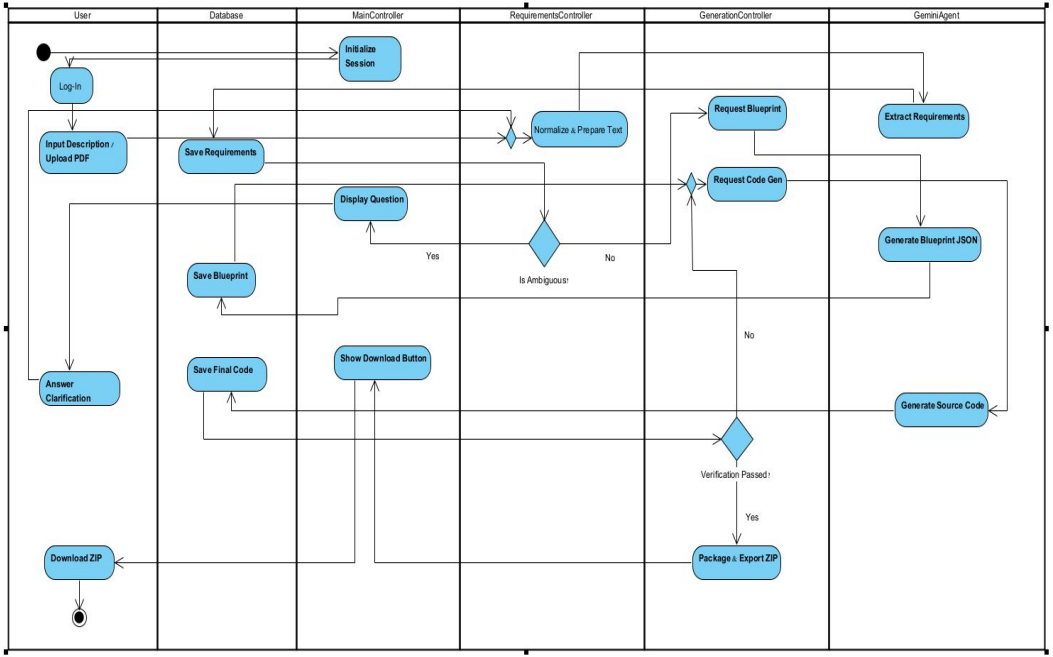


System Behavior

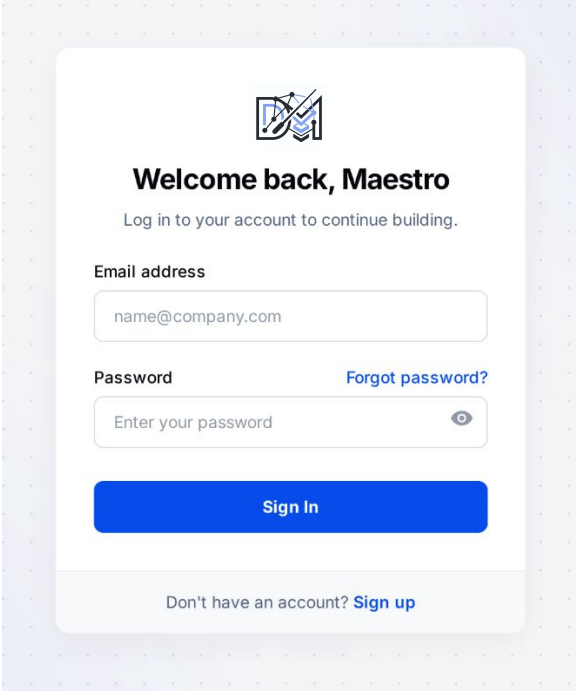
Use-Case Diagram

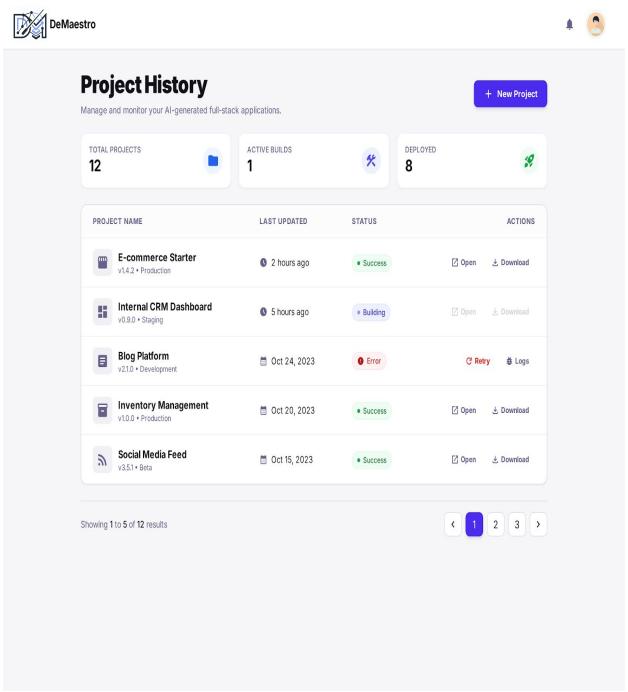


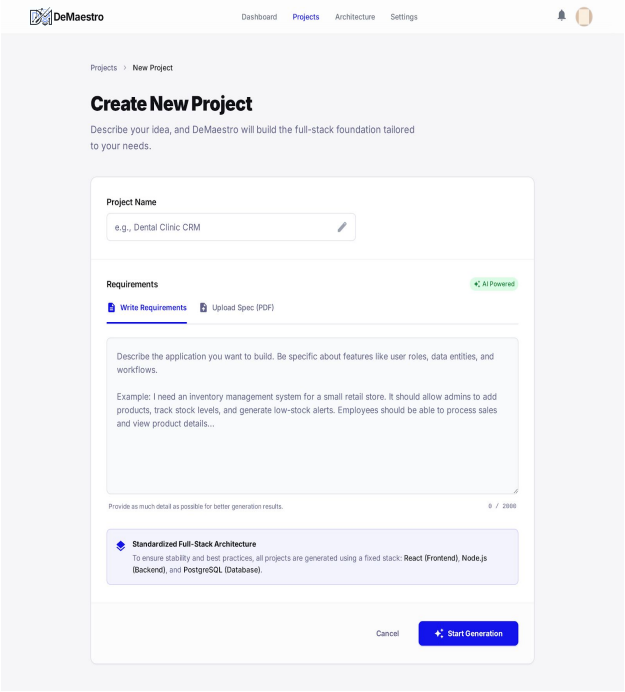
Activity Diagram



User Interface & Interaction Concept







Testing Strategy and Expected Results

- **Input Handling Tests:** Verify correct processing of text and PDF requirement inputs
- **Clarification Loop Tests:** Provide ambiguous inputs to confirm that clarification questions are generated
- **Generation Tests:** Validate that the system produces complete frontend, backend, and database code
- **Integration Tests:** Check consistency between frontend API calls and backend routes

Expected Results:

- Generated projects compile and run locally without syntax errors
- Clarification is triggered for incomplete requirements
- Exported ZIP files contain complete and structured source code
- Users can create, view, and re-download project history successfully

User Experience Testing

First-Time User Testing: Focus on new users to evaluate ease of learning and clarity of the workflow.

Task-Based Testing: Users perform basic actions such as creating a project, submitting requirements, answering clarification questions, and downloading the generated code.

System Usability Scale (SUS): Users complete a short usability questionnaire after using the system.

Acceptance Tests

ID	Requirement	Acceptance Test	Expected Result
AT1	FR1 – Requirement Input	User enters text or uploads a PDF	Input is accepted and displayed in the chat
AT2	FR3 – Requirement Clarification	User submits a vague request	System asks clarification questions
AT3	FR7/FR8 – Backend & Frontend Generation	User completes clarification	Frontend and backend code are generated
AT4	FR9 – Integration Verification	Code generation completes	Frontend API calls match backend routes
AT5	FR12 – Source Code Download	User clicks “Download Project”	ZIP file is downloaded successfully
AT6	FR13 – Project Management	User revisits dashboard	Previous projects are visible
AT7	FR14 – User Authentication	User logs in	User accesses only their own projects
AT8	NFR5 – Usability	New user submits requirements	Done within ~60 seconds
AT9	NFR1 – Reliability	Generated project is run locally	Application starts without syntax errors

Expected Challenges

Understanding User Requirements

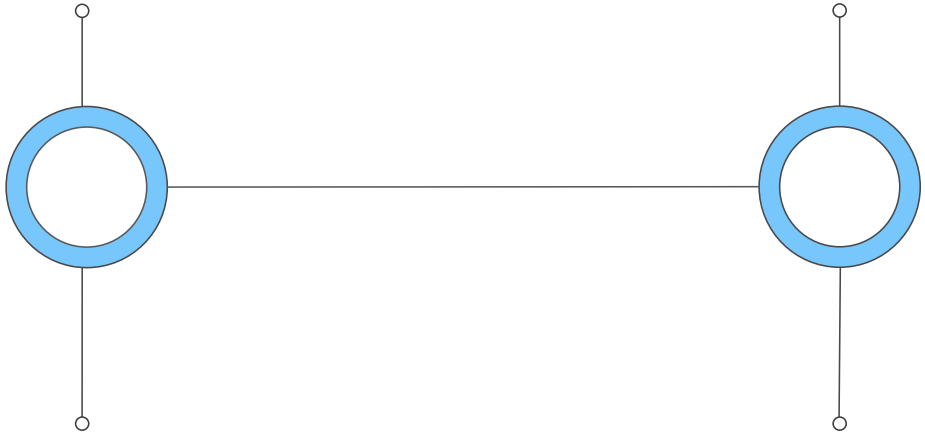
Users may provide unclear or incomplete descriptions, making it difficult for the system to fully understand the intended application

Full-Stack Integration

Ensuring that the frontend, backend, and database work together correctly is challenging and small mismatches can cause failures

Expected Challenges

Dependency on External AI Services



The system relies on external AI services, which may introduce limitations such as latency, usage limits, or availability issues

Performance & Scalability

Generating full applications is time-consuming, and system performance may degrade as the number of users increases.

Thank you!

any questions?