[LangGraph Workflow Design 1](#_Toc1607284952)

[Defining the Workflow Architecture 1](#_Toc1941469932)

[Implementing LangGraph 1](#_Toc1554814244)

[Milestone 1: Analysis 2](#_Toc573493120)

[Milestone 2: Generate Project Setup using Tools 2](#_Toc308343269)

[Sample Folder Structure for Modular FastAPI Project 2](#_Toc229627057)

[Milestone 3: Autonomous Coding Workflow 3](#_Toc1348552507)

[Benchmarking Criteria for AI-Generated Code 4](#_Toc841128663)

[Milestone 4: Persistence & Iterations 4](#_Toc1115454027)

[Milestone 5: Deployment 4](#_Toc1443100673)

[Milestone 6: Documentation 4](#_Toc37766972)

[Milestone 7: LangSmith Logging & Debugging 5](#_Toc247098531)

[Milestone 8: FastAPI Endpoint for SRS Input 5](#_Toc162378142)

**Overview**

Build a system that takes a Software Requirements Specification (SRS) document as input, analyzes its content using LangGraph, and generates an AI-powered FastAPI project following best practices in software engineering. This includes unit test generation, code debugging, deployment, and documentation. This system should align with best practices in modern software engineering, incorporating structured workflows, automated debugging, extensive testing, and seamless deployment strategies. Additionally, it should leverage **agentic workflows**, **iterative feedback loops**, and **self-improving mechanisms** to enhance efficiency and reliability.

## **LangGraph Workflow Design**

### **Defining the Workflow Architecture**

* To build an efficient system, you need to identify the appropriate **nodes, agents, and tools** that facilitate:
* **Code Generation** – Transform structured SRS inputs into a well-defined FastAPI project with modular components.
* **Unit Testing** – Implement test-driven development (TDD) by generating and executing comprehensive test cases.
* **Debugging & Refinement** – Detect and resolve errors in real-time using automated feedback loops.
* **Iterative Improvements** – Enhance system performance by refining code based on runtime analysis and error logs.

### **Implementing LangGraph**

* Define **nodes** representing core operations such as **parsing**, **code generation**, **testing**, and **deployment**.
* Establish **edges** to determine the logical flow of execution across various stages of development.
* Maintain a **GraphState** to persist and track project-related data, including generated code, test results, and debugging iterations.
* Incorporate **automated feedback loops** to facilitate continuous learning and self-improvement in AI-driven development.

**Milestones**

**Note:** Milestones **1, 2, 3, 5, and 6** represent different components of the **LangGraph workflow**. These can be implemented as a **node, agent, or tool** depending on their function. LangGraph should handle iterative refinements using **LLM-driven automation** for AI tasks and **engineering logic** for structured execution.

## **Milestone 1: Analysis**

1. Build an AI workflow (LangGraph) to analyze the SRS document and extract key software requirements (Functional Requirements) for backend system generation.
2. **Key components to analyze for backend generation: for example -**
   1. Required API endpoints and their parameters.
   2. Backend logic (business rules, required computations).
   3. Database schema (tables, relationships, constraints).
   4. Authentication & authorization requirements.
3. Extract the database schema from the SRS screenshot using **Llama-3 Vision (Groq Preview)** to analyze the image and extract structured data, and use it to create models.

## **Milestone 2: Generate Project Setup using Tools**

**Backend:**

* 1. System should be able to Initialize a structured Python **FastAPI** project with a modular folder structure.
  2. System should be able to Create a virtual environment, activate it, and install required dependencies and required packages

**Database:**

* 1. System should be able to use agents and tools to Set up and integrate a PostgreSQL database.
  2. System should be able to Implement connection pooling and migrations (using Alembic or similar).
  3. System should be able to Validate that all prerequisites (Python, PostgreSQL, necessary packages) are met.

### **Sample Folder Structure for Modular FastAPI Project**

project\_root/  
│── app/  
│ ├── api/  
│ │ ├── routes/  
│ │ │ ├── user.py  
│ │ │ ├── item.py  
│ │ │ └── \_\_init\_\_.py  
│ ├── models/  
│ │ ├── user.py  
│ │ ├── item.py  
│ │ └── \_\_init\_\_.py  
│ ├── services/  
│ ├── database.py  
│ ├── main.py  
│── tests/  
│── Dockerfile  
│── requirements.txt  
│── .env  
│── README.md

## **Milestone 3: Autonomous Coding Workflow**

1. **Generate Unit Tests using LLM:** System should be able to-
   1. Follow **Test-Driven Development (TDD)** principles.
   2. Generate **test cases first** using **Llama 3 70B** before generating implementation code.
   3. Use **pytest** for running tests and validating test coverage.
   4. Ensure tests cover all functional aspects, including edge cases.
2. **Generate Code using LLM:** System should be able to-
   1. Generate modular components including:
      1. Routes (API endpoints)
      2. Services
      3. Database Models
      4. main.py and other configurations
   2. Ensure the generated code adheres to best practices:
      1. Error handling
      2. Docstrings & documentation
      3. Logging & structured outputs
   3. System should be able to Write the generated code into respective .py files.
   4. Validate that the generated code is executable.
3. **Code Execution & Debugging:**
   1. System should be able to autonomously Run unit tests and validate code functionality.
   2. If tests fail or the application does not run, **regenerate or refine code using LLM.**
   3. System should be able to Ensure:
      1. The FastAPI backend starts successfully and connects to the database.
      2. API endpoints function correctly.

### **Benchmarking Criteria for AI-Generated Code**

* **Correctness:** Does the generated FastAPI project meet the extracted requirements?
* **Code Quality:** Is the AI-generated code structured, modular, and follows best practices?
* **Testing:** Do the unit tests cover all functional aspects?
* **Automation:** Does the workflow iteratively refine and improve the generated code?
* **Execution Success:** Does the project run without errors after generation?
* **Security:** Are best practices followed for input validation and authentication?

## **Milestone 4: Persistence & Iterations**

1. Ensure LangGraph retains previously generated code to maintain context.
   1. Example: If the system has already generated a **User model**, it should remember variable names, function names, and dependencies when generating new code (e.g., **referencing user\_id in routes and services**).
   2. Use **PostgreSQL** as database as a persistent store.
2. The workflow should align new generations with previous ones to prevent redundancy and maintain consistency.

## **Milestone 5: Deployment (Optional)**

1. Use **LLM** to generate a zip file aligned with the project requirements.

## **Milestone 6: Documentation**

1. Generate a **graph visualization** of the designed LangGraph AI workflow using **Graphviz** or similar tools like (**Mermaid**).
2. Use an LLM to generate project documentation:
   1. **README.md** (setup, usage, project structure)
   2. **API documentation** (endpoints, usage guide)

## **Milestone 7: LangSmith Logging & Debugging**

1. Create a **LangSmith** project to track logs for each execution.
2. Log key details, including:
   1. Graph execution steps
   2. API calls and responses
   3. Errors and debugging insights
3. Iterations and refinements in code generation
4. Logs should be structured to provide insights into system behavior over multiple runs.

## **Milestone 8: FastAPI Endpoint for SRS Input**

1. Expose the system as a FastAPI service that:
   1. Accepts **SRS documents** as input (only .docx format).
      1. Validates file format and handles errors (unsupported format, empty file).
   2. Uses the LangGraph workflow to generate the FastAPI project.
   3. Returns the **generated FastAPI project link/folder path and LangSmith logs** as output.

# BROWNIE: Dynamic Agent Creation (New Milestone)

* **Create an endpoint to generate dynamic agents** based on user-specified tasks.
* The system should retrieve relevant knowledge using **Retrieval-Augmented Generation (RAG)** by processing attached PDF documents.
* The new agents should be **autonomously developed** using the extracted knowledge.