Project Planning

Date	27 June 2025
Team ID	LTVIP2025TMID31381
Project Name	HealthAI
Marks	4 marks

3.1. Project Planning

The development of the HealthAI application adopted an iterative and modular approach, prioritizing core functionality and user experience for a local development and demonstration context. Key Phases and Activities:

1. Ideation & Conceptualization:

- Activities: Problem identification, brainstorming core features (AI-driven insights, personalization, user management), initial technology stack selection (Python, LLM, Database, UI Framework).
- Deliverables: Problem statement, list of core functionalities.

2. Requirement Analysis:

- Activities: Detailed definition of functional and non-functional requirements. Deep dive into technology stack selection rationale. Design of the dataflow diagram to visualize system interactions. Focus on security requirements like password hashing and environment variables.
- Deliverables: This comprehensive documentation section.

3. Project Design:

- Activities: Formulation of the solution architecture, outlining the interaction between frontend, backend, LLM, and database. Detailed proposed solution covering how each functional requirement is addressed, including the authentication flow and data linking. Decision on @st.cache_resource for optimization.

Project Planning

- Deliverables: Solution architecture diagram, detailed proposed solution descriptions.

4. Implementation:

- Setup: Setting up the local development environment (VS Code, Python environment). Installation of necessary Python libraries (transformers, torch, streamlit, pymongo, hashlib). Configuration for GPU acceleration if available locally (e.g., CUDA setup).
- Database Integration: Implementation of pymongo for MongoDB connection. Development of CRUD (Create, Read, Update, Delete - primarily Create and Read) operations for users, patients, and health_records collections.
- LLM Integration: Loading and initializing the IBM Granite model and tokenizer using Hugging Face pipeline. Creation of the generate_response helper function for consistent LLM calls.
- Streamlit UI Development: Building the multi-section Streamlit application (app.py). Implementing st.session_state for managing user login status. Designing interactive components for each health functionality.
- Core Logic Implementation: Coding the distinct functions for Symptoms Identifier, Home Remedies, Treatment Plans, Health Analytics, and Patient Chat, integrating LLM calls and database interactions.
- Security & Error Handling: Implementing basic password hashing. Integrating ObjectId.is_valid() checks for robust input validation on MongoDB IDs. Adding try-except blocks for graceful error handling across database and LLM operations. 5. Testing:
- Unit Testing (Informal): Testing individual functions (e.g., add_patient_mongodb, generate_response) in isolation within the local Python environment.
- Integration Testing: Verifying the seamless data flow and interaction between the Streamlit UI, Python backend, LLM, and MongoDB.
- Functional Testing: Systematically testing each major feature (Sign-up, Login, Patient Management, Symptoms Identifier, etc.)

against the defined functional requirements using various inputs (detailed in "GenAI Functional Test" section).

- Performance Testing (Observational): Monitoring LLM response times and overall application responsiveness, especially given the large model size and local hardware capabilities.
- User Experience Review: Informal checks on UI intuitiveness and clarity of instructions. Tools & Environment:

- **Development Environment:** VS Code with Python extension, Local Python environment.
- **Version Control:** GitHub (for tracking code changes and collaborative development).
- **Local Execution:** Running Streamlit application locally via streamlit run app.py.
- **Deployment (Conceptual for professor's context):** For demonstration purposes, the application can be run directly from VS Code. For broader accessibility or a more permanent setup, platforms like Streamlit Community Cloud (for Streamlit apps) or a custom server (e.g., using Gunicorn/Nginx) would be typical. Netlify is primarily for static sites or frontend-only applications and would not be suitable for deploying the entire Python backend with Streamlit and LLM. Timeline (Conceptual for a typical project lifecycle):
- **Phase 1 & 2 (Weeks 1-2):** Ideation, detailed requirement analysis, and initial architectural sketching.
- **Phase 3 (Weeks 3-4):** Detailed design of solution architecture and proposed solution, including database schema decisions and authentication flow.
- **Phase 4 (Weeks 5-8):** Core implementation, focusing on LLM integration, Streamlit UI, MongoDB CRUD operations, and linking all components. Iterative development of each functional module.
- **Phase 5 (Week 9):** Dedicated testing phase, including functional and basic performance tests, bug fixing, and comprehensive documentation.
- **Future Work (Beyond initial scope):** Continuous improvement, advanced features, production deployment considerations (e.g.,

Project Planning

moving beyond local GPU inference to a dedicated model serving solution). Key Risks & Mitigation Strategies:

- **LLM Performance:** Mitigated by using GPU acceleration if available locally. @st.cache_resource for model loading prevents repeated downloads within a Streamlit session.
- **Data Privacy:** Mitigated by using environment variables for credentials and basic password hashing. Further production-level security (e.g., salting/peppering passwords, robust encryption, role-based access) would be needed for a real-world application.
- **MongoDB Connectivity:** Addressed by robust error handling in get_mongodb_client and @st.cache_resource for a persistent client connection.
- **Computational Resources:** Large models require significant resources; local machine capabilities will dictate performance.
- **LLM Hallucinations/Accuracy:** Mitigated by including prominent medical disclaimers in all AI-generated responses and designing prompts to encourage cautious, informative output rather than definitive diagnoses.