Project Design Phase
Solution Architecture

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

4.1. Solution Architecture

The HealthAI application adheres to a clear client-server architecture, typical for web-based AI applications, enabling modularity and clear separation of concerns.

graph TD

A[User (Web Browser)] -- HTTP/WebSocket --> B(Streamlit User Interface)

B -- Calls Python Functions --> C{Python Application Backend (app.py)}

C -- Text Generation Requests --> D[Hugging Face Transformers Pipeline]

D -- Loads/Infers --> E(IBM Granite model)

E -- Utilizes --> F(GPU/CPU Resources)

C -- Database Operations (CRUD) --> G(pymongo Driver)

G -- Network I/O --> H[MongoDB Database (Atlas/Local)]

subgraph Internal Backend Logic

C --- I(Streamlit Session State)

C --- J(User Authentication Logic)

C --- K(Patient Data & Health Record Management)

C --- L(LLM Prompt Engineering) end Architectural Components Breakdown:

1. User Device/Browser (Client):

The end-user's device (desktop, laptop, mobile) accessing the application via a web browser.

Project Design Phase
Solution Architecture

2. Streamlit User Interface (B):

- Presentation Layer: Built entirely in Python using the Streamlit library.
- Interactive Frontend: Provides all visual elements like text inputs, buttons, dropdowns, and markdown displays.
- Communication: Handles HTTP/WebSocket communication with the underlying Python backend.
- Session State (I): Utilizes st.session_state to manage application-wide state, crucial for maintaining user login status, username, and user ID across interactions within a single session.

3. Python Application Backend (app.py) (C):

- Core Logic: The brain of the application, written in Python, orchestrating all operations.
- User Authentication Logic (J): Contains functions (register_user, login_user) responsible for user registration, password hashing, and session management.
- Patient Data & Health Record Management (K): Manages all interactions with the MongoDB database (adding patients, retrieving profiles, inserting health records). It ensures data is linked to the correct user and patient.
- LLM Prompt Engineering (L): Dynamically constructs detailed prompts for the LLM based on user input and retrieved patient context.
- Error Handling: Implements robust error handling for database issues, invalid inputs, and LLM failures.

4. Hugging Face Transformers Pipeline (D):

- LLM Abstraction Layer: A high-level interface from the transformers library.
- Model Management: Handles loading the IBM Granite model and its tokenizer, preparing inputs, and extracting outputs. @st.cache_resource ensures this expensive operation is performed only once.

Project Design Phase
Solution Architecture

5. IBM Granite model (E):

- Generative AI Core: The pre-trained large language model responsible for generating all textual AI responses (disease predictions, home remedies, treatment plans, health insights, chat responses).

6. GPU/CPU Resources (F):

- Computational Engine: The underlying hardware (e.g., local GPU with CUDA, or CPU if no GPU is available/configured) where the computationally intensive LLM inference operations are performed.

7. pymongo Driver (G):

- Database Connector: The official Python driver for MongoDB, facilitating communication and data manipulation commands between the Python backend and the MongoDB server. @st.cache_resource maintains a persistent connection.

8. MongoDB Database (H):

- Persistent Data Store: A NoSQL document database where all application data is stored. This can be a local MongoDB instance or a cloud-hosted service (e.g., MongoDB Atlas).
- Collections: Contains:
  - users: Stores user credentials (username, hashed password) and their unique _id.
  - patients: Stores patient demographics (name, age, gender), with each patient document linked to a user_id.
  - health_records: Stores AI interaction history (symptoms, predictions, vitals, treatments), with each record linked to a patient_id.