5. Project Execution

5.1 Planning and Design

The initial phase of the CliniNSync project involved extensive planning and design, starting with brainstorming sessions within the team. The primary objective was to create a healthcare platform to manage electronic health records (EHRs), document upload/downloads, and provide an AI-powered chatbot for patients. We targeted three distinct user roles: patients, doctors, and admins.

We initially planned to use Firestore for storing documents and encrypting sensitive data. However, due to its data storage limits and difficulty in handling decryption with Flask, we switched to MongoDB, which better suited the needs of the project, especially with its GridFS integration for large file storage.

Design drafts for the user interface and system architecture were created. We chose React for the frontend due to its component-based structure and Flask for the backend to handle document management, chatbot services, and EHR functionality.

5.2 Implementation

During the implementation phase, we began by developing prototypes for each role. The patient role was implemented first, allowing them to upload EHR documents, receive insurance recommendations, chat with an AI bot about their uploaded PDFs, and book appointments with doctors. The doctor role followed, enabling them to view patient details, manage appointments, and access their patients’ EHR summaries. Finally, the admin role was created to manage users and appointments without accessing patient-specific data.

MongoDB was integrated as the primary database, with GridFS used for document storage. Flask was leveraged for backend services such as the AI chatbot, EHR management, and document handling. Local storage was used to store the email and role of the logged-in user, which was referenced for role-based data fetching and management.

6. Tools and Techniques Used

6.1 Tools

MongoDB: A NoSQL database used to store user information, appointments, and EHR documents. GridFS was employed to handle large file storage like PDFs.

Flask: A Python web framework used for backend services, including document management, chatbot functionality, and handling EHRs.

React: A JavaScript library for building the frontend, providing a responsive and dynamic user interface.

Material UI: A React component library used for designing the UI with pre-built components.

GitHub: A version control system used for team collaboration and code management.

LocalStorage: Used for storing the email and role of the logged-in user, which is later used for fetching relevant data.

6.2 Techniques

We applied Agile development principles, with regular team meetings to assess progress, address challenges, and refine the approach. MongoDB's GridFS was chosen for efficient document storage and retrieval, especially for large files like PDFs. The AI chatbot was developed using Flask to process questions related to the uploaded documents.

Instead of using JWT for authentication, email/password-based login was implemented, with the email and role stored in localStorage. This simplified the authentication process while allowing for role-based access control.

7. Partial Results

7.1 Initial Findings

During the initial stages of development, we encountered challenges with Firestore’s data storage limits and encryption/decryption processes in Flask. This led us to switch to MongoDB, which better suited our document storage needs. The basic patient functionalities, such as uploading documents and receiving summaries, were successfully implemented early on, with the chatbot offering preliminary responses to document-related queries.

7.2 Iterative Improvements

As the project progressed, we iterated on several aspects:

* Improved document handling by switching to MongoDB, which resolved the issues with Firestore’s storage limits and encryption complexities.
* Refined the chatbot’s responses for more accurate interaction with patients regarding their uploaded documents.
* Optimized the user interface for smoother navigation, especially when switching between roles.
* Added more robust role-based access control, ensuring that users could only access data relevant to their role.

8. Results and Discussion

8.1 Final Results

The final version of CliniNSync meets the key objectives:

* Patients can upload/download their EHR documents, interact with the AI chatbot, receive insurance recommendations, and book appointments.
* Doctors can manage appointments, view patient details, and access summaries of EHRs for their assigned patients.
* Admins can view user details and manage appointments but do not have access to patient-specific data.
* MongoDB with GridFS was successfully integrated for storing documents, and the system is fully functional with role-based access using localStorage for user authentication.

8.2 Discussion

The project achieved its primary goals of providing a secure, role-based healthcare platform. The switch from Firestore to MongoDB was crucial for resolving storage and encryption issues. The final system was able to handle large documents, provide role-specific functionalities, and deliver a positive user experience. While we initially planned to use JWT for authentication, using localStorage for storing the email and role simplified the process and met the requirements.

9. Prototype (Hardware/Software)

9.1 Prototype Description

The developed prototype is a web application with functionalities tailored to each user role. The key features include:

* Document upload/download for patients
* AI chatbot for patient queries related to uploaded PDFs
* Appointment management for doctors
* User management and doctor registration for admins
* The system uses MongoDB for storage, Flask for backend services, and React/Material UI for the frontend.

9.2 Development Process

The development process was iterative, with regular team collaboration and testing. Initial challenges included managing document storage and handling file encryption in Flask. These were addressed by switching to MongoDB and optimizing GridFS integration.

9.3 Testing and Validation

The prototype underwent several rounds of testing on localhost, focusing on document upload, chatbot interactions, and user access control. Feedback from early users helped refine the user interface and improve the overall functionality. The system was validated to ensure that it met security standards, particularly in handling sensitive patient data.

10. Conclusion

10.1 Summary

CliniNSync is a healthcare platform that successfully integrates document management, AI chatbots, and appointment scheduling for patients, doctors, and admins. MongoDB with GridFS was crucial for handling large files, and Flask ensured smooth backend processing. The project met its objectives and provided a role-based, secure environment for users to manage health-related data.

10.2 Personal Reflection

This project allowed me to deepen my understanding of full-stack development, particularly in terms of handling large data sets, ensuring security, and integrating AI-powered functionalities. It also provided valuable insights into the healthcare industry, especially regarding data privacy and the use of technology to enhance patient care. Overall, the project enhanced my technical skills and strengthened my problem-solving abilities.