CERTIFICATE

This is to certify that the project entitled

“Online Medical History Management System”

Has been satisfactorily completed by:

1) **Sayad Arbaz**
2) **Asif Raju Shaikh**
3) **Omkar Suresh Mandavkar**
4) **Mirza Adnan Baig Nusratullah**
5) **Umer Shafik Karbhari**
6) **Hrishikesh Ramesh Helge**

Under supervision and guidance for partial fulfillment of the

PG Diploma in Advanced Computing (DAC) (24 weeks, Full Time) Course

of

Center for Development of Advanced Computing (C-DAC), Pune.

at

Academy of Information Technology (YCP)

Nariman Point, Mumbai, 400 021.

Faculty **Course Co-ordinator**

Batch: PG DAC SEP 2022 Batch
Date: 16th March 2023

PROJECT PLAN

Project Name: Online Medical History Management System
Customer: Doctors' practitioners, clinics, and multiple-specialty hospitals.
Project Manager: Sayad Arbaz
Start Date: 15 Jan 2023
Plan Date: 15 Jan 2023

Introduction

Introducing our Online Medical History Management System! A revolutionary solution for efficiently managing and storing medical reports. Our platform enables patients to easily signup and securely store their medical records.
With our system, doctors can effortlessly access patient's medical history by getting verified through a One-Time Password (OTP) sent to the patient's email. This ensures that only authorized healthcare professionals can access the data. Furthermore, doctors can add medicine data to the patient's database for comprehensive record-keeping.
To maintain the highest standards of security and confidentiality, only verified doctors will be displayed on the system. Our system is powered by a cutting-edge micro-services architecture of Spring Boot (J2EE) & Express.js (Node.js), with image data securely stored on an external Content Delivery Network (CDN).
Our user-friendly platform boasts a sleek and intuitive front-end implemented using the latest version of ReactJS(V6). We also use a reliable and robust RDBMS- MySQL database for seamless data management.
With our Online Medical History Management System, we aim to transform the way healthcare is managed and delivered. Join us on this exciting journey towards a healthier future!

Phase 1: Planning and Analysis

- Define project scope, objectives, and deliverables
- Identify stakeholders and their requirements
- Conduct a feasibility study to assess technical, financial, and operational feasibility of the project
- Develop a project plan with timelines, milestones, and resource requirements
- Conduct a risk assessment to identify potential risks and mitigation strategies
- Design the database schema and API specifications

Phase 2: Back-end Development

- Set up the development environment and infrastructure
- Develop the micro-services architecture using Spring Boot (J2EE) and Express.js (Node.js)
- Implement the user authentication and authorization system using OTP verification
- Develop the API endpoints for managing medical records and adding medicine data
- Integrate the external CDN for storing image data
- Write unit tests and conduct integration testing

Phase 3: Front-end Development

- Develop the user interface using ReactJS(V6)
- Implement the user registration and login functionalities
- Develop the user dashboard for managing medical records and medicine data
- Implement the data visualization features for displaying medical history
- Conduct user acceptance testing and fix bugs

Phase 4: Admin Panel Development

- Develop the admin panel for verifying doctors
- Implement the role-based access control for admin and doctors
- Develop the feature for adding and managing doctors
- Integrate the email notification system for OTP verification and admin approvals
- Write unit tests and conduct integration testing

Phase 5: Deployment and Maintenance

- Deploy the system on the cloud infrastructure
- Configure the system for scalability, availability, and security
- Develop the disaster recovery and backup plan
- Conduct user training and provide user support
- Monitor and optimize system performance
- Conduct periodic maintenance and upgrades

Overall, the project planning will involve five phases: planning and analysis, back-end development, front-end development, admin panel development, and deployment and maintenance. The planning phase will involve defining project scope and objectives, conducting a feasibility study, and designing the database schema and API specifications. The development phases will involve implementing the system functionalities using Spring Boot, Express.js, and ReactJS, integrating the external CDN, and conducting testing. The deployment and maintenance phase will involve deploying the system on the cloud infrastructure, providing user support, and monitoring and optimizing system performance.

Scope and Complexity Estimate

The Online Medical History Management System project involves building a web application for uploading and managing medical reports. The system will allow patients to sign up, store old medical records, and share their medical history with doctors. The doctors will need to be verified in the system using an OTP sent to the patient's email before they can access the medical records. Doctors can add medicine data to the patient's database. The system will have an admin panel for verifying doctors, managing roles and permissions, and adding doctors to the system.
The back-end of the system will be implemented using a micro-services architecture of Spring Boot (J2EE) and Express.js (Node.js). An external CDN will be used to store image data. The front-end will be implemented using ReactJS(V6) and will interact with the back-end through REST APIs. The system will use a RDBMS- MySQL database for data storage.

The scope of the project includes the following features:

- User authentication and authorization
- Patient registration and login
- Medical record management
- Medicine data management
- Doctor verification and approval
- Admin panel for managing doctors and roles
- Integration with external CDN for image storage
- User-friendly UI with data visualization features

The complexity of the project is medium to high due to the following reasons:

- Developing the micro-services architecture and integrating it with the external CDN requires significant technical expertise.
- Implementing the user authentication and authorization system using OTP verification requires complex logic and security measures.
- Designing and implementing the database schema requires careful consideration of data normalization and security.
- Developing a user-friendly UI with data visualization features requires knowledge of modern front-end technologies and UX design principles.
- Testing the system for functionality, performance, and security requires extensive testing and debugging.

The complexity of the project requires careful planning, implementation, and testing to ensure that it meets the requirements of the stakeholders and delivers a high-quality product.

Project time estimate

Week 1: Project planning and requirement gathering

- Define project scope and objectives
- Create a detailed project plan and timeline
- Identify and prioritize project requirements
- Determine team roles and responsibilities

Week 2: Medicine data management

- Develop a module for doctors to add medicine data to the patient's database
- Implement data visualization features for medicine data management
- Ensure the security and confidentiality of medicine data

Week 3: Back-end development

- Set up the micro-services architecture of Spring Boot and Express.js
- Create the user authentication and authorization system using OTP verification
- Implement data storage and retrieval using RDBMS- MySQL

Week 4: Front-end design and development

- Develop wireframes and UI/UX designs
- Implement front-end functionalities using ReactJS
- Integrate front-end with back-end APIs for user authentication and registration

Week 5: Testing and debugging

- Perform unit testing on front-end and back-end functionalities
- Conduct integration testing to ensure seamless communication between front-end and back-end
- Identify and resolve any bugs or issues in the system

Week 6: Doctor verification and approval

- Create an admin panel for managing doctor verification and approval
- Implement role-based access control to restrict access to the system
- Ensure that the system meets all security requirements

Week 7: Integration with external CDN for image storage

- Integrate the system with an external CDN for image data storage
- Optimize image retrieval for faster loading times
- Implement a module for uploading and displaying images in the system

Week 8: User acceptance testing and deployment

- Conduct user acceptance testing to ensure that the system meets all requirements and functions as intended
- Deploy the system to a production environment and ensure that it is stable and scalable
- Conduct training sessions for users and provide documentation for system usage and maintenance

Resource Allocation Report

Project Name: Online Medical History Management System

Duration: 2 Months

The following resources are allocated for the project:

Sayad Arbaz : Spring Boot
Asif Raju Shaikh : Spring Boot
Omkar Suresh Mandavkar : Node.js
Umer Shafik Karbhari : Node.js
Mirza Adnan Baig Nusratullah : ReactJS
Hrishikesh Ramesh Helge : ReactJS

Task Allocation:

Week 1:
• Project planning and requirement gathering (All resources)
• Designing the database schema (All resources)
• Setting up the development environment for Spring Boot and Node.js (Arbaz, Asif, Omkar, Umer)
• Creating a basic UI wireframe (Adnan)

Week 2:
• Implementing user registration and login functionality (Arbaz, Asif, Adnan)
• Implementing OTP verification (Omkar, Umer)
• Implementing the functionality for patients to upload medical reports (Arbaz, Asif)
• Implementing the functionality for doctors to access patient's medical history (Omkar, Umer)

Week 3:
• Implementing the functionality for doctors to add medicine data to patient's database (Omkar, Umer)
• Implementing the functionality for admin verification of doctors (Arbaz, Asif, Omkar, Umer)
• Implementing the functionality for displaying only verified doctors in the system (Omkar, Umer)

Week 4:
• Creating a responsive UI design (Adnan, Hrishikesh)
• Integrating the front-end with the back-end (Arbaz, Asif, Omkar, Umer, Adnan, Hrishikesh)

Week 5:
• Implementing external CDN to store image data (Arbaz, Asif)
• Implementing search functionality (Omkar, Umer)
• Testing and debugging the application (All resources)

Week 6:
• Deploying the application to the server (All resources)
• Final testing and bug fixing (All resources)
• Creating user manual and documentation (All resources)

Week 7:
• Implementing additional features based on user feedback (All resources)
• Enhancing the security of the system (Arbaz, Asif, Omkar, Umer)
• Performance optimization of the application (Adnan, Hrishikesh)

Week 8:
• Conducting final testing and bug fixing (All resources)
• Preparing for project handover and maintenance (All resources)

The above allocation is tentative and subject to change based on the progress of the project.

Note: Database normalization was a collective activity done by all the resources.

The above allocation is tentative and subject to change based on the progress of the project.

Roles and responsibilities

Sayad Arbaz : Spring Boot

Asif Raju Shaikh : Spring Boot

Omkar Suresh Mandavkar : Node.js

Umer Shafik Karbhari : Node.js

Mirza Adnan Baig Nusratullah : ReactJS

Hrishikesh Ramesh Helge : ReactJS

Production plan

- Conduct a thorough analysis of the requirements and design the architecture of the Online Medical History Management System, using microservices architecture of Spring Boot (J2EE) & Express.js (Node.js) for the back-end and ReactJS(V6) for the front-end.
- Develop a registration module for patients to signup and create their profiles, including storing old medical records and reports.
- Implement a verification module for doctors using OTP sent to the patient's email for authentication purposes.
- Develop a module to enable doctors to easily access the patient's medical history once verified, with the ability to add new medical reports and medicine data to the patient's database.
- Implement a module to ensure that doctors are not displayed in the system unless verified by the admin.
- Integrate an external Content Delivery Network (CDN) for storing image data.
- Develop a module to facilitate the search and retrieval of medical records and reports.
- Implement a module for generating reports and summaries for doctors to help them analyze the patient's medical history.
- Use MySQL RDBMS to store and manage the data in the system.
- Perform thorough testing and debugging of the system to ensure its reliability, security, and efficiency before deployment.

Revision process

- Review the project requirements and design to ensure that they meet the client's needs.
- Identify any gaps or inconsistencies in the project description and address them with the client.
- Evaluate the feasibility of using microservices architecture for the back-end, ReactJS for the front-end, and RDBMS-MySQL for database management.
- Determine the optimal approach for implementing user authentication and verification using OTP and email communication.
- Develop a detailed plan for integrating an external CDN to store image data securely.
- Review the data management strategies and data validation techniques implemented in the system, and identify any potential vulnerabilities.
- Perform comprehensive testing of all system components to ensure their reliability, security, and performance.
- Gather feedback from end-users and stakeholders to identify areas for improvement and incorporate the necessary changes.
- Ensure that the system documentation and training materials are up-to-date and comprehensive.
- Continuously monitor the system for issues and respond to them promptly to ensure smooth and uninterrupted service to the end-users.

Documentation translations

- Translate the project description into the target language, ensuring that all technical terms and concepts are accurately conveyed.
- Provide detailed documentation of the system's functionality and features, including instructions for patients and doctors to sign up and access the system.
- Translate the authentication and verification process using OTP and email communication into the target language.
- Provide a comprehensive guide for doctors to access and add medical data to the patient's database.
- Translate the information regarding the microservices architecture and the use of Spring Boot (J2EE) & Express.js (Node.js) into the target language.
- Provide documentation on how to store image data securely using an external CDN.
- Translate the instructions for searching and retrieving medical records and reports into the target language.
- Provide a guide for generating reports and summaries to help doctors analyze the patient's medical history.
- Translate the information regarding the use of MySQL RDBMS for database management into the target language.
- Ensure that all documentation is up-to-date and accurate, reflecting any changes or updates made to the system.

Document distribution plans

- Identify the stakeholders who need access to the system documentation, including patients, doctors, and administrators.
- Develop a distribution plan that ensures all stakeholders have access to the relevant documentation.
- Consider the best channels for distribution, such as email, online portals, or physical copies.
- Provide patients with clear instructions on how to access the system, store their medical records, and authorize access to doctors.
- Distribute user manuals to doctors, providing step-by-step instructions for accessing patient medical history and adding medicine data.
- Ensure that administrators have access to all system documentation, including technical manuals and user guides.
- Consider distributing the documentation in multiple languages, where necessary, to ensure that all stakeholders can access the system effectively.
- Provide regular updates to stakeholders on any changes or updates to the system, including new features or modifications to existing functionality.
- Include training materials to help stakeholders get up-to-speed with the system quickly, reducing the need for additional support.
- Develop a feedback mechanism to gather comments and suggestions from stakeholders, allowing for continuous improvement of the system documentation and distribution plan.

Testing plans

- Develop a comprehensive test plan that covers all aspects of the system's functionality, including user authentication, data storage, and system performance.
- Create test cases that cover both positive and negative scenarios, including edge cases and error handling.
- Use automated testing tools to ensure that the system is functioning as expected and that new features and updates do not cause unexpected issues.
- Conduct functional testing to ensure that all user interactions with the system are working as expected, including sign-up, login, and uploading medical records.
- Perform integration testing to ensure that all system components are working together as expected, including the front-end, back-end, and external CDN.
- Conduct load testing to ensure that the system can handle a large volume of users and data, and that system performance remains stable under load.
- Perform security testing to identify vulnerabilities and ensure that sensitive patient data is secure and protected from unauthorized access.
- Conduct usability testing to ensure that the system is easy to use and understand for both patients and doctors.
- Use regression testing to ensure that any updates or changes to the system do not cause issues with existing functionality.
- Monitor the system continuously to identify any issues or errors that may arise, and quickly address them to minimize system downtime and potential impact on users.

Maintenance requirements

- Regularly update the system software to ensure that it remains up-to-date and compatible with the latest security patches and updates.
- Monitor the system performance and address any issues or errors as they arise, to prevent downtime or data loss.
- Backup the system data on a regular basis to prevent data loss in case of system failure or other disasters.
- Ensure that the system is secure by implementing strong authentication mechanisms, encryption, and access controls to protect patient data from unauthorized access.
- Regularly test the system for vulnerabilities and address any security issues that are discovered.
- Perform regular maintenance on the system hardware to ensure that it remains in good working condition and is able to handle the system's workload.
- Provide user support to address any issues or concerns that users may have with the system.
- Regularly update the documentation to ensure that it remains accurate and up-to-date, including user guides, technical specifications, and maintenance procedures.
- Monitor system usage and capacity to identify areas of improvement or expansion needs to support increased user demand.
- Plan for future enhancements and upgrades to the system to ensure that it remains competitive and meets the evolving needs of users.

Guidelines for Preparing SRS Document

1.1 Purpose

The aim of this document is to explain the functionality of the project developed for an Online Medical History Management system for uploading and managing medical reports.

It is the outcome of rigorous consideration of the requirements of the customer, by the various groups involved in the development of the system.

This document will provide a baseline for the design of the database, user interfaces, coding, and evaluation of test plans. It will be used as a solid foundation for continued product evaluation and improvement.

The existing manual system of medical record management has several drawbacks, such as delays, inefficiencies, and an increased risk of errors.

The new system will provide patients with an efficient and secure platform to store and manage their medical records, while allowing doctors to easily access and update the records.

The system will also ensure the privacy and security of the sensitive medical information stored in the system through advanced security measures and verification processes.

The implementation of this system is expected to improve the quality of healthcare delivery, reduce administrative costs, and enhance patient outcomes.

1.2 Definitions, Acronyms, and Abbreviations

C-DAC Center for Development of Advanced Computing

SRS Software Requirement Specifications

1.3 References for Requirement Analysis and Design

IEEE recommended practices for SRS. ANSI / IEEE Std 830 – 1993.
Different documents, registers related to Online Medical History Management System project.

1.4 Overview

The remaining part of SRS consists of

1) Complete description of the various product functions.
2) Logical characteristics of product functions.
3) Hardware and software configuration for servers and clients operating at various levels.

Business Model

Business Model for Online Medical History Management System:

1. Value Proposition:
- Convenient and secure access to medical records for patients and doctors
- Efficient and streamlined management of medical records
- Improved communication between patients and doctors

2. Customer Segments:
- Patients who want to store and access their medical records online
- Doctors who want to access their patients' medical records easily and securely

3. Channels:
- Online platform accessible through a web browser or mobile application
- Email notifications for OTP verification and updates

4. Customer Relationships:
- Self-service access to medical records for patients
- Personalized support for doctors and admin users
- Responsive customer service for technical support

5. Revenue Streams:
- Subscription fees for doctors and admin users
- Fees for additional storage or premium features
- Commission from referral partnerships with healthcare providers

6. Key Resources:
- Reliable and secure cloud storage for medical records
- Robust backend system built on Spring Boot and Express.js
- Scalable frontend system built on ReactJS
- Qualified technical team for maintenance and support

7. Key Activities:
- Design and development of the platform
- Implementation of security protocols and privacy policies
- Integration of third-party services for image storage and communication
- Continuous monitoring and improvement of the system

8. Key Partners:
- Healthcare providers for referral partnerships
- Cloud storage providers for image data storage
- Regulatory bodies for compliance and accreditation

9. Cost Structure:
- Development and maintenance costs for the platform
- Hosting and storage costs for cloud services
- Salaries and benefits for technical team
- Legal and regulatory compliance costs

Functional Requirements

2.1 Patient Signup and Login

The system shall provide a user-friendly interface for patients to signup and login using their email and password.

2.2 Medical Record Upload

The system shall allow patients to upload their medical reports in PDF or image formats. Patients shall be able to add descriptions and tags to the uploaded reports for easy search and retrieval.

2.3 Medical Record Management

The system shall provide patients with the ability to view and manage their uploaded medical reports. Patients shall be able to edit descriptions and tags, and delete reports that are no longer needed.

2.4 Doctor Verification

The system shall send a one-time password (OTP) to the patient's email for doctor verification. Only verified doctors shall be able to access patient data.

2.5 Doctor Access to Medical History

The system shall provide doctors with access to patient's medical history upon successful verification. Doctors shall be able to view and download patient medical reports, and add medicine data to the patient's database.

2.6 Admin Verification of Doctors

The system shall allow admin users to verify doctors before they are granted access to patient data. Only verified doctors shall be displayed on the system.

Non-functional Requirements

3.1 Performance

The system shall be able to handle a large number of concurrent users and medical reports. The system shall also provide fast response times for users.

3.2 Security

The system shall use secure communication protocols to ensure the confidentiality and integrity of patient data. The system shall also use strong authentication mechanisms to prevent unauthorized access to patient data.

3.3 Reliability

The system shall be available for use 24/7. The system shall also have a backup and recovery mechanism in place to prevent data loss in case of system failure.

3.4 Technology Stack

The system shall be developed using the following technology stack:

- Back-end: Spring Boot (J2EE) & Express.js (Node.js) with micro-services architecture
- Front-end: ReactJS(V6)
- Database: RDBMS- MySQL
- External CDN for image data storage

Table of Contents (Business Model)

List of Functions

F-1: Patient Sign-up
F-2: Patient Medical Record Upload
F-3: Doctor Verification
F-4: Doctor Access to Patient Medical Records
F-5: Doctor Medicine Data Entry
F-6: Admin Verification

Functional Description

F-1: Patient Sign-up

1. This function allows patients to sign up to the system by providing their personal details, contact information, and creating a username and password.

Purpose: To enable patients to access the system and upload their medical records.
Organization Unit: Patients
Time (when performed): When a patient wants to access the system.
Documents: None
Frequency: One-time
Data Entities: Patient Personal Details, Contact Information, Username, Password

F-2: Patient Medical Record Upload
This function allows patients to upload their medical records to the system.
Purpose: To enable patients to store and manage their medical records.
Organization Unit: Patients
Time (when performed): When a patient wants to upload their medical record.
Documents: Medical Record
Frequency: As required
Data Entities: Medical Record Details, Patient ID

F-3: Doctor Verification
This function allows the system to verify the authenticity of a doctor by sending an OTP to the patient's email address.
Purpose: To ensure that only verified doctors have access to patient medical records.
Organization Unit: System
Time (when performed): When a doctor attempts to access a patient's medical record.
Documents: None
Frequency: One-time
Data Entities: Doctor Details, OTP

F-4: Doctor Access to Patient Medical Records
This function allows doctors to access patient medical records once they are verified.
Purpose: To enable doctors to view a patient's medical history.
Organization Unit: Doctors
Time (when performed): When a doctor wants to access a patient's medical record.
Documents: None
Frequency: As required
Data Entities: Patient ID, Medical Record Details

F-5: Doctor Medicine Data Entry
This function allows doctors to enter medicine data for a patient.
Purpose: To enable doctors to add medicine data to a patient's database.
Organization Unit: Doctors
Time (when performed): When a doctor wants to enter medicine data for a patient.
Documents: Medicine Data
Frequency: As required
Data Entities: Patient ID, Medicine Data

F-6: Admin Verification
This function allows the system to verify the authenticity of an admin.
Purpose: To ensure that only authorized personnel can access the system.
Organization Unit: System
Time (when performed): When an admin attempts to access the system.
Documents: None
Frequency: One-time
Data Entities: Admin Details, OTP

Glossary

1. SRS: Software Requirements Specification
2. Online Medical History Management System: A web-based system for managing medical records of patients and doctors.
3. Patient: An individual who uses the system to store and manage their medical records.
4. Doctor: A healthcare professional who has access to patient's medical history after verification.
5. OTP: One-Time Password, a unique password sent to patient's email for verification purposes.
6. Back-end: The server-side of the application which manages data processing and storage.
7. Microservices Architecture: An architectural style in which an application is broken down into small, independent services that communicate with each other using APIs.
8. Spring Boot: An open-source Java-based framework used to create microservices.
9. Express.js: A lightweight and flexible Node.js framework used to build web applications.
10. CDN: Content Delivery Network, a network of servers used to store and deliver content such as images and videos.
11. Front-end: The client-side of the application which is responsible for rendering and displaying the interface to the user.
12. ReactJS: A JavaScript library used for building user interfaces.
13. RDBMS: Relational Database Management System, a type of database management system that stores data in a structured format using tables.
14. Admin: An authorized user who can verify doctor's access to the patient's medical history.

Conclusion

This document has defined the software requirements for the Online Medical History Management System. The requirements have been defined to ensure the system provides a secure, reliable, and user-friendly platform for patients and doctors to manage medical reports. The system shall be developed using the defined technology stack to ensure optimal performance and scalability.

Hardware & Network Interfaces

➪ Back-end Server Configuration

Intel Pentium-IV
128 MB RAM
1 Raid Controller Card
32 bit Ethernet Controller (100 Base T)
8 x 2.0 GB Fast SCSI/2 with Raid Support
2.88 MB FDD
48 x CD ROM Drive
SVGA Colour Monitor on PCI with 1MB RAM
101 Keys Keyboard
1 Microsoft Mouse with pad
4/8 GB DAT
One Serial & Two Parallel Ports
Internet Information Server(IIS)
Microsoft Transaction Server(MTS)

➪Front-end Client Configuration

Intel Pentium-III @ 650 MHz

128 MB SDRAM
10 GB Hard Disk Drive
1.44 MB Floppy Disk Drive
15” SVGA Digital Color Monitor
One Serial, One Parallel port and
One USB port.
104 Keys Keyboard
PS2 Mouse with pad
32 bit PCI Ethernet Card
48X CD Drive

Software Interfaces

➪Software configuration for back-end Services

Windows NT – Server 4.0 SQL Server 7.0

➪Software configuration for front-end Services

Virus Protection Software
Client Work station
Office 2000
Web Browser – Internet Explorer/Netscape
Norton Utilities
TCP/IP
Other client application software as per requirement

Communication Interfaces

Various network protocols such as ISDN, ATM will be used for Intranet connectivity. UTP Ethernet, TCP/IP Protocols will be used for Local Area Networks. Network hubs, routers, bridges, cables, patch cables, connectors will be required.

PROJECT CLOSURE REPORT

- **Overall Productivity Achieved**
  With the completion of this project, the patient can upload and view is medical history in text as well as image format. The doctor can view the previous medical history of a patient if the patient is visiting him for the first time.
  This makes it easier for the doctor to diagnose the patient without having to worry about his pre medical conditions.
  The doctor can also upload the medical prescriptions on a per visit basis.
  Over all, this application is efficient for the patient and doctor to efficiently manage their documents and medical records.

- **Estimated Start and End date of the project**
  The initial planning of the project started in the month of January with the design of database using Crowfoot notations, designing the initial diagrams using wireframe diagrams, and defining the functionality of the project using use case diagrams.
  The actual execution of the project started in February 2023, with the doctor and admin modules developed in spring boot and the patient module developed in a node.js environment.
  The UI is developed using the React.js library.
  The estimated end of project was calculated to be 25-02-2023.

- **Tools Used**

  1. **Eclipse**
     Eclipse is an open source and widely used SDK (studio development kit) that is well known for its plugins and friendly user interface. Eclipse allows to write, organize, compile, run and deploy the code and project in order to test it and make sure it functions in a way you want it to. To get full explanation on powers of eclipse use eclipse website.

  2. **Tomcat**
     Since this project is a web app, it will be deployed on a server. Tomcat is open source servlet foundation run on Apace servers. This means that when you compile your project it gets converted to a .jar file. This file is put on a server to make the application run on server.
     In this particular project since spring boot application is developed. In-built tomcat server is used.

- **Enabling Technologies**

1. **Spring**
   The Spring Framework is an open-source application framework for Java. It provides a comprehensive programming and configuration model for modern Java-based enterprise applications. The framework supports various programming models such as imperative, reactive, and functional programming, and it offers a wide range of features, including dependency injection, aspect-oriented programming, and transaction management.

Spring is widely used in enterprise applications and provides solutions for building web applications, microservices, batch processing, and integration applications. It is designed to be modular and flexible, allowing developers to use only the components that they need.

Some of the key features of the Spring Framework include:

- Dependency injection: Allows objects to be created and managed by a container rather than by application code.
- Aspect-oriented programming: Enables modularization of cross-cutting concerns such as security, logging, and caching.
- Data access: Provides a consistent approach to working with relational databases, NoSQL databases, and other data sources.
- MVC framework: Offers a model-view-controller architecture for building web applications.
- Integration with other technologies: Integrates with other popular frameworks and technologies, such as Hibernate, Struts, and JPA.

Overall, the Spring Framework is a powerful and flexible tool for building robust, scalable, and maintainable Java applications.

2. **Hibernate (Spring data JPA)**

Spring Data JPA is a module of Spring framework that provides an abstraction layer for working with database persistence in java application using the java Persistence API (JPA).

JPA is a specification for object-Relational Mapping(ORM) in Java, which defines a set of interfaces and annotations for mapping java objects to relational database tables. Spring data JPA provides a higher level of abstraction over JPA, making it easier to work with databases in java applications by eliminating boilerplate code and providing a simplified interface for common data access operations.

With Spring data JPA, developers can use a set of predefined repository interfaces to interact with the database, without the of writing custom data access code. The interfaces provide methods for common data access operations, such as inserting, updating, deleting, and querying data. Developers can also define custom queries using JPA query language JPQL, or native SQL.

Spring Data JPA also supports various JPA implementations including Hibernate, EclipseLink and OpenJPA. This allows developers to switch between different JPA providers without having to modify their code.

Overall Spring data JPA Simplifies the development of the data-driven applications in Java by providing a high-level, flexible and easy to use interface to work with relational databases.

3. Node.js (Express.js)

Express.js is a popular and widely used web application framework for Node.js. It provides a set of robust features and tools for building web applications and API's using Node.js. Express.js is known for its simplicity, flexibility, and sustainability making it an ideal choice for developing both small and large web applications.

Express.js is built on top of Node.js and leverages the power of Node.js to provide a fast, event-driven, non-blocking I/O model. It provides a simple, yet powerful, HTTP server API that makes it easy to develop web applications in Node,js.

Some of the key features of Express are:

- Routing: Express.js provides a powerful routing mechanism that makes it easy to define and handle HTTP requests based on URL's, HTTP methods and request parameters.
- Middleware: Express.js allows developers to use middleware functions to add additional functionality to their applications. Middleware functions can be used for handling requests, manipulating data and performing authentication and authorization.

Over all Express.js is a powerful and flexible web application framework for Node.js that simplifies the development of web applications and API's by providing a rich set of features and tools.

4. MySql (RDBMS)

MySQL is one of the most recognizable technologies in the modern big data ecosystem. Often called the most popular database and currently enjoying widespread, effective use regardless of industry, it's clear that anyone involved with enterprise data or general IT should at least aim for a basic familiarity of MySQL.

With MySQL, even those new to relational systems can immediately build fast, powerful, and secure data storage systems. MySQL's programmatic syntax and interfaces are also perfect gateways into the wide world of other popular query languages and structured data stores.

5. Cloudinary (External CDN)

Cloudinary is a cloud based image and video management platform that provides an API for developers to manage, manipulate and deliver media files in the cloud. The Cloudinary API is a RESTful API that provides a simpe an instrusive way to upload, transform and serve media files.

Cloudinary supports multiple programming languages such as Java, PHP, Python, Ruby, Node.js and many other. It also integrates with popular content management systems such as WordPress, Magneto etc.

Over all, the Cloudinary API simplifies media management in the cloud by providing a powerful set of features and tools that developers can use to optimize, transform and deliver media files at scale.

6. React.js (SPM tool)

React (also known as **React.js** or **ReactJS**) is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.

React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, React is only concerned with the user interface and rendering components to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

- **Defects**

  Defects can occur at any stage of the SDLC, but the impact and cost of fixing them may vary depending on when they are detected.

  Defects may occur during following period:

1. Requirement Gathering
2. Design
3. Development
4. Testing
5. Deployment
6. Maintenance

Over all its important to detect and fix defects as early as possible in the SDLC to minimize their impact and cost. This requires a robust testing strategy and focus on quality throughout the entire the entire software development process.