**SOFTWARE PROJECT FINAL REPORT**

**Team Members**

* Shardul Panchal
* Parth Patel

**Date:**

December 4, 2024

**Table of Contents**

1. **Introduction**  
   1.1 Purpose and Scope  
   1.2 Product Overview  
   1.3 Structure of the Document  
   1.4 Terms, Acronyms, and Abbreviations
2. **Project Management Plan**  
   2.1 Project Organization  
   2.2 Lifecycle Model Used  
   2.3 Risk Analysis  
   2.4 Hardware and Software Resource Requirements  
   2.5 Deliverables and Schedule
3. **Requirement Specifications**  
   3.1 Stakeholders for the System  
   3.2 Use Cases  
   3.3 Rationale for Use Case Model  
   3.4 Non-functional Requirements
4. **Architecture**  
   4.1 Architectural Style(s) Used  
   4.2 Architectural Model  
   4.3 Technology, Software, and Hardware Used  
   4.4 Rationale for Architectural Style and Model
5. **Design**  
   5.1 User Interface Design  
   5.2 Component Design  
   5.3 Database Design  
   5.4 Traceability from Requirements to Design Models
6. **Test Management**  
   6.1 Complete List of Test Cases  
   6.2 Traceability of Test Cases to Use Cases  
   6.3 Techniques for Test Case Generation  
   6.4 Test Results and Assessments  
   6.5 Defects Report
7. **Conclusions**  
   7.1 Outcomes of the Project  
   7.2 Lessons Learned  
   7.3 Future Development
8. **References**

**1. Introduction**

**1.1 Purpose and Scope**

The purpose of this project is to develop a web-based Healthcare Appointment System to streamline the appointment booking process for patients and healthcare providers. This project reduces administrative workload, enhances patient satisfaction, and provides a user-friendly experience.

The scope includes:

* Patient registration and authentication.
* Doctor selection and appointment booking.
* Admin dashboard for managing appointments.

**1.2 Product Overview**

The Healthcare Appointment System allows patients to register, book, and manage their appointments. Admins can manage doctor schedules and oversee patient bookings. The system features a responsive design for accessibility on desktops, tablets, and smartphones.

**1.3 Structure of the Document**

This document provides a detailed explanation of the system’s purpose, scope, design, architecture, testing, and outcomes.

**1.4 Terms, Acronyms, and Abbreviations**

* **Next.js**: A React framework for building web applications with server-side rendering and static site generation capabilities.
* **Appwrite**: A backend-as-a-service platform used for managing APIs, databases, and user authentication.
* **TailwindCSS**: A utility-first CSS framework that simplifies styling by providing reusable design classes.
* **ShadCN**: A library of accessible and reusable UI components built on TailwindCSS for rapid interface development.
* **EHR (Electronic Health Records)**: Digital records used by healthcare providers to store and manage patient information securely.
* **Reusable Design and UI Components**: The project incorporates prebuilt design systems from TailwindCSS and accessible UI components from ShadCN to accelerate development and maintain a consistent user experience.

**2. Project Management Plan**

**2.1 Project Organization**

The team roles were distributed as follows:

* **Developers**: Responsible for design, implementation, and testing (Shardul Panchal & Parth Patel).
* **Project Manager**: Oversaw the timeline and deliverables.

**2.2 Lifecycle Model Used**

The **Waterfall Model** was chosen for its structured approach:

1. Requirements analysis.
2. System design.
3. Implementation.
4. Testing.
5. Deployment.

**2.3 Risk Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Probability** | **Impact** | **Mitigation Strategies** |
| Data Security Breaches | Medium | High | Implement encryption, security audits. |
| System Downtime | Low | High | Deploy redundant systems and failover mechanisms. |
| Integration Issues | Medium | Medium | Use phased integration with regular testing. |
| User Resistance | Medium | Medium | Provide training and user-friendly interfaces. |

**2.4 Hardware and Software Resource Requirements**

* **Hardware**: Laptop or PC with Windows/macOS.
* **Software**: Node.js, Appwrite, Next.js, TailwindCSS, and a modern web browser.

**2.5 Deliverables and Schedule**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Start Date** | **End Date** | **Duration** |
| Requirements Analysis | 09/15/2024 | 09/20/2024 | 5 Days |
| Figma Designing | 09/21/2024 | 09/27/2024 | 7 Days |
| Frontend Development | 09/29/2024 | 10/15/2024 | 17 Days |
| Backend Integration | 10/16/2024 | 10/23/2024 | 7 Days |
| Testing | 10/31/2024 | 11/03/2024 | 4 Days |
| Final Deployment | 11/11/2024 | 11/15/2024 | 5 Days |

**3. Requirement Specifications**

**3.1 Stakeholders for the System**

* **Patients**: End-users who book appointments.
* **Healthcare Providers**: Doctors, nurses, and specialists who manage appointments and schedules.
* **Administrative Staff**: Handles appointment management and resolves scheduling conflicts.

**3.2 Use Cases**

**Patient Use Case**

1. Register for an account.
2. Log in to view the dashboard.
3. Book, reschedule, or cancel appointments.
4. Receive confirmation of booked appointments.

**Admin Use Case**

1. Log into the admin dashboard.
2. Manage doctor schedules (add/remove availability).
3. Oversee patient bookings and resolve conflicts.

**3.3 Rationale for Use Case Model**

The use case model ensures that all critical workflows for patients and administrators are accounted for and directly map to system functionalities.

**3.4 Non-functional Requirements**

* **Scalability**: The system must handle up to 500 concurrent users during peak hours.
* **Availability**: Minimum 99% uptime.
* **Security**: All data in transit and at rest must be encrypted.
* **Responsiveness**: Load time should not exceed 2 seconds on average.

**4. Architecture**

**4.1 Architectural Style(s) Used**

The system employs a **component-based architecture**:

* **Frontend**: Built using Next.js and styled with TailwindCSS.
* **Backend**: Appwrite serves as the backend for API handling and database management.

**4.2 Architectural Model**

* **Frontend Components**:
  + User registration and login pages.
  + Appointment booking and management dashboards.
* **Backend Components**:
  + APIs for user authentication and appointment management.
  + Database collections for users, appointments, and schedules.

**4.3 Technology, Software, and Hardware Used**

* **Frontend**: Next.js, TailwindCSS, ShadCN.
* **Backend**: Appwrite for backend services.
* **Hardware**: Desktop or laptop with an internet connection.

**4.4 Rationale for Architectural Style and Model**

The architecture ensures modularity, scalability, and maintainability. The reuse of TailwindCSS and ShadCN UI components accelerated development while maintaining consistency.

**5. Design**

**5.1 User Interface Design**

* **Patient Dashboard**: Simple layout with options to view and manage appointments.
* **Admin Dashboard**: Detailed overview of doctor schedules and patient bookings.

**5.2 Component Design**

Frontend components were built to:

* Handle user interactions, such as form submissions for registration and booking.
* Provide real-time feedback for user actions, e.g., confirmation messages.

**5.3 Database Design**

Collections used:

1. **Users Collection**:
   * Fields: id, name, email, role (patient/admin).
2. **Appointments Collection**:
   * Fields: id, patientId, doctorId, dateTime, status (scheduled, cancelled).
3. **Schedules Collection**:
   * Fields: id, doctorId, availableSlots.

**5.4 Traceability from Requirements to Design Models**

Each functional requirement is mapped to:

* A corresponding UI component in the front end.
* An API endpoint in the backend.

**6. Test Management**

**6.1 Complete List of Test Cases**

1. **TC-001**: Validate user registration functionality.
2. **TC-002**: Ensure successful login for valid credentials.
3. **TC-003**: Verify appointment booking without conflicts.

**6.2 Traceability of Test Cases to Use Cases**

Each test case corresponds to a use case, ensuring end-to-end coverage.

**6.3 Techniques for Test Case Generation**

* **Equivalence Partitioning**: To test different inputs effectively.
* **Boundary Value Analysis**: To ensure system reliability at edge cases.

**6.4 Test Results and Assessments**

* **Unit Tests**: Passed with 95% coverage.
* **System Tests**: Completed successfully for all core workflows.

**6.5 Defects Report**

* Minor UI inconsistencies were resolved during testing.

**7. Conclusions**

**7.1 Outcomes of the Project**

The system successfully implemented patient and admin workflows, ensuring usability and scalability. Automated notification features were excluded for future implementation.

**7.2 Lessons Learned**

* Effective task allocation streamlined development.
* The modular design reduced integration complexity.

**7.3 Future Development**

* Integration with EHR systems.
* Development of a mobile app for iOS and Android.

**8. References**

* Next.js Documentation: https://nextjs.org/docs
* Appwrite API Documentation: https://appwrite.io/docs
* TailwindCSS Documentation: <https://tailwindcss.com/docs>