# System Design Document

## 1. Introduction

### 1.1 Purpose

The purpose of this application is to enhance the digital reach and service delivery of small to medium-sized healthcare providers to their patients. By offering a user-friendly digital platform in a cost-effective multi-tenant SaaS environment, the app aims to help these providers increase their market share through improved outreach and customer satisfaction.

### 1.2 Scope

#### 1.2.1 In Scope

Initially, this app will let consumers register and create their medical profile. They can then book in-patient or virtual appointments and pay for it online through multiple channels. They can upload their past medical history, download prescriptions, book follow-up visits to a doctor amongst features described in this document.

Healthcare providers can manage their services list, onboard healthcare practitioners, remove practitioners, manage appointments, and upload prescriptions and medical history for patients. They will have the option to generate management reports of their financial, operational, or customer data.

Doctors can view their appointments, update patient medical history, change appointment status, create patient prescriptions, and generate a printout of the prescription for the patient.

The app will be hosted on a multi-tenant SaaS platform, where the data of each service provider will be kept secure and distinct from other providers. Patient medical records (EHR) will be maintained in the client facility. This app will fetch, insert, and update EHR data but would not store the data in the app’s multi-tenant cloud platform.

All authentication processes would comply with relevant data protection laws (e.g., HIPAA, GDPR) to ensure patient data remains secure and private.

The process of registration of a patient is fully automated. The registration of healthcare staff will be created by a system admin. Login for a healthcare professional or doctor would be created by healthcare staff while onboarding individual doctors.

Only a patient can create an appointment but is not allowed to delete an appointment. They can reschedule an appointment before 24 hrs. Doctors can modify appointments but cannot delete them. Appointments can only be deleted by healthcare staff.

Patients can only view prescriptions, whereas doctors and healthcare staff can create and modify them. Management reports can be generated by healthcare staff only.

Patients will be given an option of an AI-driven symptom checker. This chatbot will analyze patients’ symptoms and suggest possible causes and the type of doctor to consult.

#### 1.2.2 Out of Scope

Integration with third-party suppliers such as ambulance operators, pharmacists, and medical tourists are out of scope of this app presently. Currently, this app doesn’t support the IPD journey of patients.

Since this is a multi-tenant SaaS environment, this app would not follow an aggregator model. Hence, each patient will be linked to a single healthcare provider app.

### 1.3 Definitions, Acronyms, and Abbreviations

* **SaaS:** Software as a Service. A software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet.
* **EHR:** Electronic Health Record. A digital version of a patient’s paper chart. EHRs are real-time, patient-centered records that make information available instantly and securely to authorized users.
* **HIPAA:** Health Insurance Portability and Accountability Act. A US law designed to provide privacy standards to protect patients’ medical records and other health information provided to health plans, doctors, hospitals, and other healthcare providers.
* **GDPR:** General Data Protection Regulation. A regulation in EU law on data protection and privacy for all individuals within the European Union and the European Economic Area.
* **IPD:** In-patient Department. Refers to hospital services provided to patients who are admitted for an overnight stay or longer.
* **SME:** Small to Medium-sized Enterprise.

### 1.4 References

No external documents were provided as references for this specification.

### 1.5 Overview

This document specifies the requirements for a multi-tenant SaaS application designed to digitalize patient journeys for small to medium-sized healthcare providers. It details the product’s purpose, scope (both in and out), user roles and characteristics, overall constraints, assumptions, and dependencies. The core of the document outlines the specific functional, non-functional, and technical requirements necessary for the application’s development, including details on system architecture, data handling (specifically regarding EHRs), security, and compliance. The aim is to provide a clear and comprehensive guide for the development team.

## 2. Overall Description

### 2.1 Product Perspective

The Healthcare Patient Journey Digitalization Application is a standalone product operating as a multi-tenant SaaS platform. It is designed to interface with existing systems, primarily external Electronic Health Record (EHR) systems maintained by the client healthcare providers. The application acts as a digital front-end and management tool for provider services, appointment scheduling, and patient interactions, while relying on the client’s facility for the persistent storage of core patient EHR data. It is not an aggregator model, meaning each patient is linked to a specific healthcare provider tenant.

### 2.2 Product Functions

The application provides core functionalities for different user types: - **Patients:** Registration, profile management, appointment booking (in-patient/virtual), online payment, medical history upload, prescription viewing/download, follow-up booking, appointment rescheduling, and AI symptom checking. - **Healthcare Providers (via Staff):** Service list management, practitioner onboarding/removal, staff/doctor account creation, appointment management (including deletion), patient medical history/prescription management, and management report generation. - **Healthcare Practitioners (Doctors):** Appointment viewing/management, patient medical history/prescription management (creation/modification/printout), and appointment modification. - **System Administrators:** Creation of Healthcare Staff accounts.

### 2.3 User Characteristics

* **System Admin:** Users with administrative privileges responsible for initial setup and account creation for healthcare staff. Requires technical understanding of the system administration console.
* **Healthcare Staff:** Non-technical or semi-technical users at healthcare providers responsible for managing the provider’s configuration within the application, onboarding practitioners, managing appointments, and generating reports. They interact with a dedicated staff interface.
* **Healthcare Practitioner (Doctor):** Medical professionals who use the system primarily to view their schedules, manage appointments, and update/create patient medical records and prescriptions. They interact with a dedicated doctor interface. Requires basic computer literacy.
* **Patient:** The primary end-users, potentially with varying levels of digital literacy. The application is designed for ease of use with minimal clicks. Patients access the system via a patient-facing interface (likely web/mobile).

### 2.4 Constraints

* **Multi-tenant Architecture:** The system must securely segregate data for each healthcare provider tenant.
* **EHR Data Handling:** Patient EHR data must not be stored persistently within the SaaS platform’s cloud environment. It must be accessed or updated via integration with client EHR systems.
* **Regulatory Compliance:** All data handling, security, and privacy measures must strictly comply with relevant healthcare data protection laws (e.g., HIPAA, GDPR).
* **Patient-Provider Linking:** Each patient account must be technically linked to only one healthcare provider tenant instance.
* **Appointment Modification/Deletion Rules:** Specific rules apply to who can reschedule/modify (Patient, Doctor, Staff) and who can delete appointments (only Staff).
* **Prescription Access:** Patients can only view/download; Staff/Doctors can create/modify.
* **Report Access:** Only Healthcare Staff can generate management reports.

## 8.1 Functional Requirements

1. **Patient Registration:** The system shall allow patients to register for an account using an automated process.
2. **Patient Profile Management:** The system shall allow registered patients to create, view, and manage their personal and medical profile information.
3. **Appointment Booking:** The system shall allow patients to browse available slots for their linked healthcare provider and book either in-patient or virtual appointments with a healthcare practitioner.
4. **Online Appointment Payment:** The system shall allow patients to pay for their booked appointments securely online using multiple integrated payment channels.
5. **Patient Medical History Upload:** The system shall allow patients to upload their past medical history documents to their profile.
6. **Patient Prescription Access:** The system shall allow patients to view and download digital prescriptions issued by their healthcare practitioner through the application.
7. **Follow-up Appointment Booking:** The system shall allow patients to book follow-up appointments with a healthcare practitioner.
8. **Patient Appointment Rescheduling:** The system shall allow patients to reschedule their booked appointments up to 24 hours before the originally scheduled time. Patients are not permitted to delete appointments.
9. **AI Symptom Checker:** The system shall provide an AI-driven symptom checker functionality that analyzes symptoms entered by a patient and suggests possible causes and appropriate types of doctors to consult.

## 8.2 Non-Functional Requirements

1. **Usability:** The system shall provide a user-friendly interface and simple user journeys, particularly for patients, requiring minimal clicks and assistance.
2. **Scalability:** The system shall be designed to be scalable to support a large number of small to medium-sized healthcare providers and their respective patient bases within the multi-tenant architecture.
3. **Security:** The system shall implement robust security measures for all authentication processes, data handling, and access controls to protect sensitive healthcare data.
4. **Data Isolation:** The system shall ensure strict segregation of data, maintaining each healthcare provider’s data securely and distinctly separate from other providers within the multi-tenant platform.
5. **Compliance:** The system shall comply with relevant data protection laws, such as HIPAA and GDPR, in all aspects of data handling, storage constraints, and authentication.
6. **Privacy:** The system shall ensure the privacy of all patient and provider data in accordance with applicable regulations.
7. **Maintainability & Extensibility:** The system architecture shall be designed to facilitate the faster addition of newer healthcare services and support additional stakeholder journeys in the future.
8. **Performance:** The system shall provide responsive performance for typical user interactions, such as browsing appointments, loading profiles, and processing payments.
9. **Availability:** The system shall aim for high availability to ensure patients and providers can access required services reliably.

# System Architecture for Healthcare Patient Journey Digitalization Application

## 1. High-Level Architecture Diagram

graph LR  
 User[User Interface]  
 API[API Layer]  
 Service[Service Layer]  
 Database[Database]  
 EHR[EHR Integration]  
 Payment[Payment Gateway]  
   
 User --> API  
 API --> Service  
 Service --> Database  
 Service --> EHR  
 Service --> Payment

## 2. Component Diagram

graph TB  
 subgraph Frontend  
 Patient[Patient Interface]  
 Doctor[Doctor Interface]  
 Staff[Staff Interface]  
 Admin[Admin Interface]  
 end  
  
 subgraph Backend  
 Auth[Authentication Service]  
 Appointment[Appointment Management]  
 Prescription[Prescription Management]  
 MedicalHistory[Medical History Management]  
 Report[Report Generation]  
 Payment[Payment Processing]  
 end  
  
 Patient --> Auth  
 Patient --> Appointment  
 Patient --> Prescription  
 Patient --> MedicalHistory  
 Patient --> Report  
 Doctor --> Appointment  
 Doctor --> Prescription  
 Staff --> Appointment  
 Staff --> MedicalHistory  
 Staff --> Report  
 Admin --> Auth  
   
 Appointment --> Payment  
 Prescription --> Payment

## 3. Technology Stack Table

| Layer | Technology |
| --- | --- |
| Frontend | React, Vue.js |
| Backend | Node.js, Express |
| Database | MongoDB, PostgreSQL |
| EHR Integration | REST APIs, HL7 Integration |
| Authentication | OAuth2, JWT |
| Payment Gateway | Stripe, PayPal |
| Cloud Hosting | AWS, Azure |
| AI Services | Custom AI Module / Dialogflow |

## 4. Design Patterns Used

* **MVC (Model-View-Controller)**: For separating concerns in the web application architecture.
* **Singleton**: For managing shared resources, such as the database connection.
* **Repository Pattern**: For abstracting data access and making the application more maintainable.
* **Observer Pattern**: To react to changes in patient data and appointment status.
* **Strategy Pattern**: For different algorithms in the AI symptom checker.

This architecture is designed to facilitate a multi-tenant environment while ensuring compliance with regulatory standards such as HIPAA and GDPR, providing a secure platform for healthcare providers and patients.

### Section 5: Data Flow Diagrams, ER Diagram, and Data Dictionary

**1. Data Flow Diagrams (Level 0 and Level 1)**

**Level 0 DFD:**

graph LR  
 A[Patient] -->|Registers| B[System]  
 A -->|Books Appointment| B  
 B -->|Confirms Appointment| A  
 A -->|Views Prescriptions| C[Prescription Service]  
 A -->|Uploads Medical History| D[Medical History Service]  
 A -->|Makes Payment| E[Payment Service]  
 A -->|Uses Symptom Checker| F[AI Service]

**Level 1 DFD:**

graph TB  
 subgraph Patient  
 A[Patient Interface]  
 end  
 subgraph System  
 B[Authentication Service]  
 C[Appointment Management]  
 D[Prescription Management]  
 E[Medical History Management]  
 F[Payment Processing]  
 G[AI Symptom Checker]  
 end  
  
 A --> B  
 A --> C  
 A --> D  
 A --> E  
 A --> F  
 A --> G  
   
 C --> F  
 D --> F

**2. ER Diagram**

erDiagram  
 PATIENT {  
 string id PK  
 string name  
 string email  
 string phone  
 string password  
 string healthcare\_provider\_id FK  
 }  
   
 DOCTOR {  
 string id PK  
 string name  
 string specialization  
 string schedule  
 string healthcare\_provider\_id FK  
 }  
   
 HEALTHCARE\_PROVIDER {  
 string id PK  
 string name  
 string contact\_info  
 }  
   
 APPOINTMENT {  
 string id PK  
 string patient\_id FK  
 string doctor\_id FK  
 dateTime appointment\_time  
 string status  
 string type  
 }  
   
 PRESCRIPTION {  
 string id PK  
 string patient\_id FK  
 string doctor\_id FK  
 string medication\_details  
 }  
  
 MEDICAL\_HISTORY {  
 string id PK  
 string patient\_id FK  
 string history\_document  
 }  
  
 PAYMENT {  
 string id PK  
 string appointment\_id FK  
 decimal amount  
 string status  
 }  
  
 HEALTHCARE\_PROVIDER ||--o{ DOCTOR : employs  
 DOCTOR ||--o{ APPOINTMENT : schedules  
 PATIENT ||--o{ APPOINTMENT : books  
 PATIENT ||--o{ PRESCRIPTION : receives  
 PATIENT ||--o{ MEDICAL\_HISTORY : submits  
 APPOINTMENT ||--o{ PAYMENT : includes

**3. Data Dictionary (Word-style Tabular Format)**

| Table Name | Column | Type | Constraints | Purpose |
| --- | --- | --- | --- | --- |
| PATIENT | id | string | PK | Unique identifier for each patient |
|  | name | string | NOT NULL | Patient’s name |
|  | email | string | UNIQUE, NOT NULL | Patient’s email address |
|  | phone | string | NOT NULL | Patient’s contact number |
|  | password | string | NOT NULL | Encrypted password for authentication |
|  | healthcare\_provider\_id | string | FK (HEALTHCARE\_PROVIDER.id) | Links patient to their healthcare provider |
| DOCTOR | id | string | PK | Unique identifier for each doctor |
|  | name | string | NOT NULL | Doctor’s name |
|  | specialization | string | NOT NULL | Doctor’s medical specialization |
|  | schedule | string | NOT NULL | Available schedule for appointments |
|  | healthcare\_provider\_id | string | FK (HEALTHCARE\_PROVIDER.id) | Links doctor to their healthcare provider |
| HEALTHCARE\_PROVIDER | id | string | PK | Unique identifier for each healthcare provider |
|  | name | string | NOT NULL | Name of the healthcare organization |
|  | contact\_info | string | NOT NULL | Contact information for the healthcare provider |
| APPOINTMENT | id | string | PK | Unique identifier for appointments |
|  | patient\_id | string | FK (PATIENT.id) | Links appointment to the patient |
|  | doctor\_id | string | FK (DOCTOR.id) | Links appointment to the doctor |
|  | appointment\_time | dateTime | NOT NULL | Date and time of the appointment |
|  | status | string | NOT NULL | Current status of appointment (e.g. booked/confirmed) |
|  | type | string | NOT NULL | Type of appointment (in-patient/virtual) |
| PRESCRIPTION | id | string | PK | Unique identifier for prescriptions |
|  | patient\_id | string | FK (PATIENT.id) | Links prescription to the patient |
|  | doctor\_id | string | FK (DOCTOR.id) | Links prescription to the doctor |
|  | medication\_details | string | NOT NULL | Details of prescribed medication |
| MEDICAL\_HISTORY | id | string | PK | Unique identifier for medical history |
|  | patient\_id | string | FK (PATIENT.id) | Links medical history to the patient |
|  | history\_document | string |  | Uploaded medical history document |
| PAYMENT | id | string | PK | Unique identifier for payment transactions |
|  | appointment\_id | string | FK (APPOINTMENT.id) | Links payment to the specific appointment |
|  | amount | decimal | NOT NULL | Total amount of the payment |
|  | status | string | NOT NULL | Status of the payment (e.g. completed, pending) |

### Section 6: API Specifications and UI Wireframe Layout Descriptions

#### 1. External/Internal API Tables

| API Name | Method | Endpoint | Input Parameters | Output Parameters | Auth Required |
| --- | --- | --- | --- | --- | --- |
| Patient Registration | POST | /api/patients/register | { “name”: string, “email”: string, “phone”: string, “password”: string } | { “status”: string, “patient\_id”: string } | Yes |
| Patient Login | POST | /api/patients/login | { “email”: string, “password”: string } | { “status”: string, “token”: string } | No |
| Appointment Booking | POST | /api/appointments/book | { “patient\_id”: string, “doctor\_id”: string, “appointment\_time”: datetime } | { “status”: string, “appointment\_id”: string } | Yes |
| View Prescriptions | GET | /api/patients/{patient\_id}/prescriptions | { “patient\_id”: string } | [ { “prescription\_id”: string, “details”: string } ] | Yes |
| Upload Medical History | POST | /api/patients/{patient\_id}/history | { “patient\_id”: string, “document”: file } | { “status”: string } | Yes |
| Generate Management Reports | GET | /api/reports/generate | { “staff\_id”: string, “type”: string } | { “report”: object } | Yes |
| AI Symptom Checker | POST | /api/symptom-checker | { “symptoms”: string } | { “possible\_causes”: [string], “recommended\_doctor”: string } | No |
| Create Patient Prescription | POST | /api/patients/{patient\_id}/prescriptions | { “patient\_id”: string, “doctor\_id”: string, “medication\_details”: string } | { “status”: string, “prescription\_id”: string } | Yes |

#### 2. UI Wireframe Layout Descriptions

**A. Patient Interface**

* **Header**: Positioned at the top of the screen, stylish and modern with the application logo on the left and navigation links (Home, Profile, Appointments) on the right. Background color should be light with contrasting text.
* **Sidebar**: Vertical menu on the left featuring icons and labels (Home, Book Appointment, View Prescriptions, Upload History, Symptom Checker) for easy navigation. Visually separated by subtle background shading.
* **Content Area**: The larger section on the right displaying dynamic content based on user interaction. Includes forms for booking appointments, viewing prescriptions, etc. Styled with ample whitespace for clarity.
* **Footer**: A clean, minimalist section at the bottom with links to privacy policy, terms of service, and contact information.

**B. Healthcare Staff Interface**

* **Header**: Similar to the patient header but includes “Staff Dashboard” title. Extra menu links for managing doctors and generating reports.
* **Sidebar**: Contains options like “Manage Appointments,” “Generate Reports,” and “Doctor Management” with clear indication of selected menu option through active states.
* **Content Area**: Featuring tables and forms for managing data (appointments, reports). Information is presented with a clear grid layout making data easy to read.
* **Modal Windows**: For creating reports or adding/updating doctor information. Styled with soft shadows and rounded corners for a modern feel.

**C. Doctor Interface**

* **Header**: Displays the doctor’s name and specialized role prominently. Links for Profile, Appointments, and Prescriptions under the header.
* **Sidebar**: Options like “View Appointments,” “Manage Prescriptions,” and “Patient History” organized vertically for quick access.
* **Content Area**: Displays upcoming appointments and relevant data like patient history. Includes interactive elements like buttons to modify appointment status.
* **Visual Hierarchy**: Emphasis on important information with larger fonts for headings and contrasting colors for buttons.

**D. Admin Interface**

* **Header**: Contains the application name and quick access links for user management and settings.
* **Sidebar**: Comprehensive menu for all the management features including “User Management,” “System Reports,” and “Settings.”
* **Content Area**: Key statistics presented in cards (users, appointments, system health) for quick consumption, alongside tables for deeper data management.
* **Modal Pop-Ups**: For confirming deletion of users or generating reports, ensuring users can double-check actions before proceeding.

### Layout Styling

Throughout all interfaces, a color scheme should maintain a professional yet approachable aesthetic, likely using soft blues and grays with white backgrounds. Icons should be simple and intuitive, ensuring every user (regardless of technical skill) can navigate the application effortlessly. Font sizes must be compliant with accessibility standards, ensuring readability for all users.

The overall layout promotes a clean, organized, and user-friendly interface that fosters efficient interaction while adhering to compliance and privacy protocols essential for a healthcare application.

### Section 7: Security Overview

#### 1. Threat Modeling Using STRIDE

Using the STRIDE threat modeling framework, we identify potential threats for the Healthcare Patient Journey Digitalization Application as follows:

* **Spoofing**: Unauthorized users might impersonate legitimate users or healthcare providers to access sensitive information.
  + **Protection Mechanism**: Implement multi-factor authentication (MFA) for both patient and provider login processes, in addition to using OAuth2 for secure access delegation.
* **Tampering**: Data transmitted between clients and the server may be susceptible to interception and alteration.
  + **Protection Mechanism**: Utilize HTTPS for all data transmissions and apply data integrity checks (e.g., hashing) to ensure that data has not been tampered with in transit.
* **Repudiation**: Users may deny sending a transaction or performing an action.
  + **Protection Mechanism**: Maintain detailed audit logs with timestamps for all user actions (e.g., appointment bookings, modifications) to ensure accountability and non-repudiation.
* **Information Disclosure**: Sensitive medical information might be exposed to unauthorized users or applications.
  + **Protection Mechanism**: Implement data encryption at rest and in transit using AES and TLS, ensuring that sensitive information is accessible only to authorized personnel through Role-Based Access Control (RBAC).
* **Denial of Service (DoS)**: Attackers may overload the system’s resources, leading to service unavailability.
  + **Protection Mechanism**: Employ rate limiting and anomaly detection mechanisms to mitigate DoS attacks and ensure availability during peak usage.
* **Elevation of Privilege**: A user may gain unauthorized escalation of privileges to access restricted functionalities.
  + **Protection Mechanism**: Enforce strict access controls based on authenticated user roles (using RBAC) to limit unauthorized function access.

#### 2. Mermaid Threat Diagrams

Below is a stylized threat diagram visualizing the threats identified using the STRIDE model with annotations on colors and layout for clarity.

graph TD  
 subgraph Threats  
 direction TB  
 Spoofing(Spoofing)  
 Tampering(Tampering)  
 Repudiation(Repudiation)  
 Disclosure(Information Disclosure)  
 DoS(Denial of Service)  
 Elevation(Elevation of Privilege)  
 end  
  
 style Spoofing fill:#FFDDC1,stroke:#BF616A,stroke-width:2px  
 style Tampering fill:#C1E1FF,stroke:#5GA3D5,stroke-width:2px  
 style Repudiation fill:#F6D86A,stroke:#F3A47D,stroke-width:2px  
 style Disclosure fill:#E7FFE7,stroke:#42C76D,stroke-width:2px  
 style DoS fill:#FFBFA4,stroke:#FF7F50,stroke-width:2px  
 style Elevation fill:#F8C8DE,stroke:#B388B6,stroke-width:2px  
  
 subgraph Countermeasures  
 direction TB  
 A1(Authentication Mechanism)  
 A2(Data Encryption)  
 A3(Audit Logging)  
 A4(Access Control)  
 A5(Availability)  
 A6(Role-Based Access Control)  
 end  
  
 style A1 fill:#FFDDC1,stroke:#BF616A,stroke-width:2px  
 style A2 fill:#C1E1FF,stroke:#5GA3D5,stroke-width:2px  
 style A3 fill:#F6D86A,stroke:#F3A47D,stroke-width:2px  
 style A4 fill:#E7FFE7,stroke:#42C76D,stroke-width:2px  
 style A5 fill:#FFBFA4,stroke:#FF7F50,stroke-width:2px  
 style A6 fill:#F8C8DE,stroke:#B388B6,stroke-width:2px  
  
 Spoofing --> A1  
 Tampering --> A2  
 Repudiation --> A3  
 Disclosure --> A2  
 DoS --> A5  
 Elevation --> A4  
 Elevation --> A6

#### 3. Authentication Mechanisms

* **OAuth2**: This protocol will be employed for secure authorization. It allows third-party applications to obtain limited access to HTTP services by allowing users to share their data without sharing their credentials. Patients will be able to access the application securely, while healthcare providers can manage their access through delegated permissions.
* **JWT (JSON Web Tokens)**: This method will encapsulate user information and claims within a token, which can be signed for integrity verification. Tokens will be exchanged at authentication to maintain session integrity and prevent spoofing attacks.
* **RBAC (Role-Based Access Control)**: This strategy will be implemented to manage user roles and permissions efficiently. Each user will have specific permissions tied to their role (e.g., patients, doctors, admin staff), ensuring that users can only access data and perform actions pertinent to their responsibilities.

#### 4. Data Encryption

All sensitive data will be encrypted both at rest and during transmission using industry-standard encryption protocols including AES (Advanced Encryption Standard) for data at rest and TLS (Transport Layer Security) for data in transit, ensuring the confidentiality and integrity of patient data.

#### 5. Compliance Needs

* **HIPAA**: The application will adhere to the Health Insurance Portability and Accountability Act which mandates strict rules on safeguarding patient information. Regular audits and risk assessments will be scheduled to ensure compliance.
* **GDPR**: All processes involving personal data will follow the General Data Protection Regulation guidelines concerning user consent, data minimization, and the right to access personal information. Users will have clear pathways to request data modifications or deletions.

This security section provides a robust framework that identifies potential vulnerabilities and outlines corresponding strategies to mitigate them, ultimately ensuring the security and compliance of the Healthcare Patient Journey Digitalization Application.

**Section 8: Non-Functional Requirements and Quality Attributes**

| **Attribute** | **Description** | **Implementation Strategy** |
| --- | --- | --- |
| **Performance** | The system should handle 1000 concurrent users with a latency of less than 2 seconds for critical actions like appointment bookings and payments. | Load testing during development, using tools like JMeter or LoadRunner to simulate multiple users. Optimize APIs and database queries to improve response times. |
| **Uptime** | The system must maintain an uptime of 99.9% to ensure reliable services for patients and providers. | Utilize a cloud infrastructure with failover strategies such as load balancing and auto-scaling features when traffic exceeds normal levels. Implement monitoring tools for real-time health checks. |
| **Latency** | User interactions (like loading profiles or appointment statuses) should have a response time of less than 1 second. | Optimize front-end code for performance, minimize HTTP requests, and leverage content delivery networks (CDN) to serve static assets quickly. |
| **Scalability** | The application should scale horizontally to accommodate up to 10,000 healthcare providers and their associated patients. | Implement a microservices architecture to allow independent scaling of services (e.g., user management, appointment services). Use containerization (Docker) for resource efficiency. |
| **Maintainability** | The codebase should support modular updates and the introduction of new features without significant downtime. | Adhere to SOLID principles in software design, write comprehensive documentation, and maintain automated testing suites to minimize regression issues. |
| **Availability** | System should be designed to recover from failures without significant downtimes, ensuring continuous service. | Implement automatic backups and database replication. Leverage disaster recovery strategies and regular recovery testing. Ensure adequate logging for post-failure analysis. |
| **Data Isolation** | Each tenant’s data must be isolated to prevent unauthorized access from other tenants. | Use separate databases or dedicated schemas for each healthcare provider in a multi-tenant architecture. Ensure access controls are enforced rigorously during data interactions. |
| **Security** | Must protect all patient data and application access under relevant regulations (HIPAA, GDPR). | Implement strong authentication (e.g., multi-factor) and data encryption techniques at rest and in transit. Conduct regular security audits and compliance checks. |
| **Compliance** | Ensure all aspects of data handling comply with necessary healthcare regulations. | Regularly review compliance requirements, conduct training for developers on best practices, and involve compliance teams in the development process. |
| **User Experience** | The application should have an intuitive user interface that minimizes user friction. | Apply user-centered design principles based on user research. Conduct usability testing with real users to guide design decisions and constantly iterate based on feedback. |

This table encapsulates critical non-functional requirements and quality attributes necessary for the successful deployment of the Healthcare Patient Journey Digitalization Application, ensuring it meets operational expectations for performance, reliability, and user satisfaction.

### Section 9: User Journey Flowcharts and State Machine Diagrams

#### 1. User Journey Flowcharts

**Patient Journey Flowchart**

flowchart TD  
 A[Start] --> B[Register]  
 B --> C[Create Medical Profile]  
 C --> D[Book Appointment]  
 D -->|In-patient| E[Select Doctor]  
 D -->|Virtual| F[Choose Virtual Slot]  
 E --> G[Pay Online]  
 F --> G  
 G --> H[Receive Confirmation]  
 H --> I[Upload Medical History]  
 I --> J[View Prescriptions]  
 J --> K[Follow-up Appointment]  
 K --> L[Use AI Symptom Checker]  
 L --> M[End]  
 style A fill:#F9F9F9,stroke:#333,stroke-width:2px  
 style M fill:#F9F9F9,stroke:#333,stroke-width:2px

**Healthcare Provider Journey Flowchart**

flowchart TD  
 A[Start] --> B[Login]  
 B --> C[Manage Services]  
 C --> D[Onboard Practitioners]  
 D --> E[Manage Appointments]  
 E --> F[Handle Prescriptions]  
 F --> G[Generate Management Reports]  
 G --> H[Logout]  
 H --> I[End]  
 style A fill:#F0FFFF,stroke:#333,stroke-width:2px  
 style I fill:#F0FFFF,stroke:#333,stroke-width:2px

#### 2. State Machine Diagrams

**Appointment State Machine Diagram**

stateDiagram-v2  
 [\*] --> Idle  
 Idle --> Booked : Appointment Booking  
 Booked --> Confirmed : Confirm Appointment  
 Confirmed --> Completed : Appointment Completed  
 Confirmed --> Cancelled : Cancel Appointment  
 Completed --> [\*]  
 Cancelled --> [\*]  
   
 state Booked {  
 [\*] --> Pending  
 Pending --> Confirmed : Confirm with Provider  
 Pending --> Cancelled : Patient Cancels  
 }

**Prescription State Machine Diagram**

stateDiagram-v2  
 [\*] --> Created  
 Created --> Approved : Doctor Approves  
 Approved --> Active : Prescription Active  
 Active --> Refilled : Refill Request  
 Active --> Completed : Patient Completes  
 Active --> Cancelled : Prescription Canceled  
 Refilled --> Active  
 Completed --> [\*]  
 Cancelled --> [\*]

This comprehensive presentation covers user journey flowcharts detailing the interaction paths of both patients and healthcare providers along with state machine diagrams illustrating the statuses and transitions of appointments and prescriptions within the system. By ensuring these visualizations adhere to the specified styling and clarity, they provide an effective framework for understanding user processes and system behavior.

### Section 10: Deployment Architecture and CI/CD Flow Explanation

#### 1. Deployment Architecture Diagram

graph TD  
 subgraph Load Balancer  
 LB[Load Balancer]  
 end  
  
 subgraph Infrastructure  
 A1[Web Server: Nginx/Apache]  
 A2[APP Server: Node.js/Express]  
 A3[API Gateway]  
 A4[Database Cluster: PostgreSQL/MongoDB]  
 A5[EHR API]  
 A6[Payment Gateway: Stripe/PayPal]  
 A7[Monitoring: Grafana/ELK]  
 end  
  
 subgraph Users  
 U1[Patients]  
 U2[Healthcare Staff]  
 U3[Doctors]  
 U4[Admins]  
 end  
  
 U1 --> LB  
 U2 --> LB  
 U3 --> LB  
 U4 --> LB  
   
 LB --> A1  
 LB --> A2  
 A2 --> A3  
 A2 --> A4  
 A2 --> A5  
 A2 --> A6  
 A2 --> A7  
 style LB fill:#F0F0F0,stroke:#333,stroke-width:2px  
 style A1 fill:#E0E0E0,stroke:#333,stroke-width:2px  
 style A2 fill:#E0E0E0,stroke:#333,stroke-width:2px  
 style A3 fill:#E0E0E0,stroke:#333,stroke-width:2px  
 style A4 fill:#E0E0E0,stroke:#333,stroke-width:2px  
 style A5 fill:#E0E0E0,stroke:#333,stroke-width:2px  
 style A6 fill:#E0E0E0,stroke:#333,stroke-width:2px  
 style A7 fill:#E0E0E0,stroke:#333,stroke-width:2px

#### 2. CI/CD Flow Explanation

The Continuous Integration and Continuous Deployment (CI/CD) process for this multi-tenant SaaS application consists of various stages, each leveraging multiple tools to automate software delivery and improve quality. Here’s a detailed breakdown:

1. **Code Commit**:
   * Developers push code changes to the version control system (e.g., GitHub, GitLab).
2. **Continuous Integration**:
   * **Build Stage**:
     + **Tools**: Jenkins, GitLab CI
     + Code is automatically built into containers using Docker, ensuring consistency across environments.
   * **Testing Stage**:
     + **Tools**: Jest, Mocha for unit tests; Cypress for end-to-end tests
     + Automated tests are executed to validate code functionality.
3. **Containerization**:
   * The built application is packaged into containers to ensure a consistent deployment environment.
   * **Tools**: Docker
4. **Deployment Pipeline**:
   * **Staging Deployment**:
     + Automated deployment of the code to the staging environment for further testing.
     + **Tools**: Kubernetes, Helm for managing deployment configurations.
   * **Approval Process**:
     + Manual or automated approval steps ensure the changes are reviewed before production deployment.
5. **Production Deployment**:
   * Once approved, the code is deployed to the production environment.
   * Blue-Green or Canary deployment strategies are used to minimize downtime and mitigate risks.
   * **Tools**: Kubernetes to manage deployment scaling and traffic routing.
6. **Monitoring and Logging**:
   * Continuous monitoring of application performance and health is performed using:
     + **Tools**: Grafana for metrics visualization, ELK stack (Elasticsearch, Logstash, Kibana) for centralized logging.
7. **Feedback Loop**:
   * Monitoring results are analyzed to provide feedback to development teams for future improvements.

This CI/CD process ensures rapid and reliable delivery of updates to the application while maintaining quality through testing and monitoring protocols.

### Section 11: Monitoring Strategy

#### Monitoring Strategy Overview

Monitoring is critical for maintaining the health and performance of the Healthcare Patient Journey Digitalization Application. This strategy outlines the tools and practices used to achieve effective observability and alerting.

1. **Key Monitoring Tools**:
   * **Grafana**:
     + Purpose: Grafana is used for visualization of application metrics and monitoring dashboards.
     + Configuration: Dashboards display CPU usage, memory consumption, response times, and error rates.
   * **ELK Stack**:
     + **Elasticsearch**: Stores and indexes logs for fast searching.
     + **Logstash**: Collects and processes logs from various sources (containers, servers).
     + **Kibana**: Provides a web interface for visualizing logs and generating reports.
2. **Metrics to Monitor**:
   * Application Performance Metrics:
     + Response times for API endpoints.
     + Error rates for services (4xx and 5xx)
     + Service uptime.
   * Resource Utilization Metrics:
     + CPU and memory usage of application servers.
     + Disk I/O and capacity for databases.
   * User Metrics:
     + Active users, session durations, and interaction rates.
3. **Alerting Mechanisms**:
   * Define alert thresholds for key metrics to notify the operations team of potential issues, such as:
     + High error rates exceeding 5% for a sustained period.
     + Slow response times exceeding defined service level agreements (SLAs).
     + Resource utilization exceeding 80% capacity.
4. **Incident Response**:
   * Establish a runbook for incident management which includes:
     + Defined escalation procedures.
     + Roles and responsibilities for incident resolution.
     + Guidelines for conducting root cause analysis post-incident.
5. **Performance Reviews**:
   * Regular performance review meetings will be held to assess application health and discuss metrics trends. Adjustments to the monitoring setup may be made based on findings.

This monitoring strategy, leveraging Grafana and the ELK stack, ensures that the application runs reliably, complies with healthcare regulations, and meets user expectations for performance and availability. By actively monitoring system health and application performance, we can quickly respond to issues, ensuring minimal impact on both providers and patients.

### Section 12: Integration Sequence Diagrams + Functional Flow Explanations

#### 1. External Integrations with Diagrams

**A. Patient Registration Process**

sequenceDiagram  
 participant Patient  
 participant App  
 participant EHR as External EHR System  
 participant Auth as Authentication Service  
   
 Patient->>App: Register with personal data  
 App->>Auth: Validate credentials  
 Auth-->>App: Valid credentials  
 App->>EHR: Create Patient Record  
 EHR-->>App: Confirmation of record creation  
 App-->>Patient: Registration successful  
 %% Layout Styling  
 style sequence fill:#F9F9F9,stroke:#333,stroke-width:2px  
 classDef highlight fill:#ffcccc;  
 class App highlight;

**Description**: The Patient registers with the application, which validates the registration credentials through an authentication service. Once validated, the application creates a corresponding record in the external EHR system, ensuring patient data is correctly registered.

**B. Appointment Booking Process**

sequenceDiagram  
 participant Patient  
 participant App  
 participant Auth  
 participant EHR  
 participant Schedule as Appointment Scheduler  
   
 Patient->>Auth: Login  
 Auth-->>Patient: Authenticate User  
 Patient->>App: Request Available Slots  
 App->>Schedule: Query Doctor Availability  
 Schedule-->>App: Return Available Slots  
 App-->>Patient: Display Available Slots  
 Patient->>App: Confirm Appointment Slot  
 App->>EHR: Update Appointment Schedule  
 EHR-->>App: Confirmation of Appointment  
 App-->>Patient: Appointment Confirmed  
 %% Layout Styling  
 style sequence fill:#F9F9F9,stroke:#333,stroke-width:2px  
 classDef highlight fill:#ccffcc;  
 class Schedule highlight;

**Description**: After logging in, the Patient requests available appointment slots, which are fetched from the Appointment Scheduler. Once the Patient confirms a slot, the appointment is updated in the EHR system, effectively securing the appointment.

**C. Payment Processing for Appointments**

sequenceDiagram  
 participant Patient  
 participant App  
 participant Payment as Payment Gateway  
   
 Patient->>App: Initiate Payment  
 App->>Payment: Process Payment  
 Payment-->>App: Payment Confirmation  
 App-->>Patient: Payment Successful  
 %% Layout Styling  
 style sequence fill:#F9F9F9,stroke:#333,stroke-width:2px  
 classDef highlight fill:#ccccff;  
 class Payment highlight;

**Description**: The Patient initiates payment for an appointment through the application, which communicates with an external payment gateway to process the payment. Upon confirmation from the payment system, the application notifies the Patient of a successful transaction.

**D. AI Chatbot Integration for Symptom Checker**

sequenceDiagram  
 participant Patient  
 participant App  
 participant AI as AI Service  
   
 Patient->>App: Submit Symptoms  
 App->>AI: Analyze Symptoms  
 AI-->>App: Return Possible Causes  
 App-->>Patient: Display Possible Causes  
 %% Layout Styling  
 style sequence fill:#F9F9F9,stroke:#333,stroke-width:2px  
 classDef highlight fill:#ffe6cc;  
 class AI highlight;

**Description**: The Patient submits symptoms to the application, which forwards this information to an AI service for analysis. The AI service returns possible causes, which the application then displays to the Patient, facilitating better health decisions.

#### 2. Description of Each Integration

1. **API Integration**:
   * APIs are used for communication between the application and external systems, such as EHRs and payment gateways. They enable seamless data transfer, allowing for patient records to be created and updated in real-time.
2. **Webhook Integration**:
   * Webhooks will be employed for real-time notifications from payment gateways to alert the application of payment statuses, ensuring instant updates to patient records without requiring further action from users.
3. **Triggers**:
   * Triggers in the application are utilized to initiate specific workflows when certain conditions are met, such as sending a confirmation email after a successful appointment booking.
4. **Fallback Logic**:
   * Fallback logic will be implemented to handle potential issues with integrations, such as retrying a payment transaction or attempting reconnections in case a third-party API is temporarily unresponsive.
5. **Retry Logic**:
   * The application will include retry logic for API calls that fail due to network issues, ensuring that requests are retried a set number of times before failing altogether, thus enhancing reliability.

This detailed overview of external integrations captures the critical workflows within the Healthcare Patient Journey Digitalization Application, encompassing interactions with third-party systems, maintaining compliance, and ensuring a user-friendly experience. The use of Mermaid diagrams not only provides clarity in flow but also visually represents the integration points in an accessible format.

### Section 13: Traceability Matrix

| **Functional Requirement ID** | **Feature/Component** | **Linked Design Section** | **Related Test Case ID** |
| --- | --- | --- | --- |
| FR-01 | Patient Registration | Registration Module | TC-01 |
| FR-02 | Patient Profile Management | Profile Management Module | TC-02 |
| FR-03 | Appointment Booking | Appointment Management Module | TC-03 |
| FR-04 | Online Appointment Payment | Payment Processing Module | TC-04 |
| FR-05 | Patient Medical History Upload | Medical History Management Module | TC-05 |
| FR-06 | Patient Prescription Access | Prescription Management Module | TC-06 |
| FR-07 | Follow-up Appointment Booking | Appointment Management Module | TC-07 |
| FR-08 | Patient Appointment Rescheduling | Appointment Management Module | TC-08 |
| FR-09 | AI Symptom Checker | AI Services Module | TC-09 |

### Mapping Explanation:

* **Functional Requirement ID**: Identifies each unique functional requirement within the Software Requirements Specification (SRS).
* **Feature/Component**: Indicates the feature or system component that fulfills the functional requirement.
* **Linked Design Section**: Specifies the relevant section in the design documentation that elaborates on how the feature/component is implemented.
* **Related Test Case ID**: Links to the associated test case, ensuring that all functional requirements are tested thoroughly and maintained throughout the Software Development Life Cycle (SDLC).

This traceability matrix ensures that every functional requirement has a designated component, is designed according to specific guidelines, and is subject to rigorous testing, thus promoting the successful delivery and quality assurance of the system as it progresses through development and deployment phases. This structured approach helps all stakeholders clearly understand and ensure compliance with the outlined requirements, ultimately enhancing the application’s integrity and performance in the healthcare landscape.