SOFTWARE REQUIREMENTS

SPECIFICATION

for

DigiMED

Prepared by

Adam Mitry, Ahmed Abdelkader,

Hoda Hussein, Mohamed Khalil Brik,

Rana Taher, Yasmina Mahdy

Version 2.0

November 1, 2024

Revision History

Date	Reason for Changes	Version
2024-10-23	Initial draft	1.0
2024-11-1	Added sprint 2 requirements	2.0

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive overview of the software requirements for DigiMED, a digital healthcare platform aimed at facilitating efficient and secure interaction between doctors, patients, and healthcare administrators. This Software Requirements Specification (SRS) will guide the development process by clearly defining both functional and non-functional requirements, ensuring the final product meets the needs of all users and complies with healthcare industry standards.

1.2 Scope

This core subset of DigiMED's features is centered on providing essential functionalities within three main user views: **Patient View, Doctor View** and **Admin View**.

Patient View: Aimed at empowering patients with access to their medical information and appointment management, the Patient View includes:

- **Appointment Scheduling**: Allows patients to view, book, and manage their appointments with healthcare providers.
- **Medical Records Access**: Enables patients to securely view essential parts of their medical history, providing transparency and continuity of care.
- **Prescription Management**: Provides patients with access to their current prescriptions, enhancing adherence to medication schedules.

Doctor View: Provides the doctor with the ability to interact with patients in an effortless, seamless, and smart manner.

- **Appointment Scheduling**: Allows doctors to avail their schedules and manage their appointments with patients.

- **Medical Records Access**: Enables doctors to update and record patients' medical conditions, tests, etc. for their and the patients' later reference.
- **Prescription Management**: Enables doctors to assign and add the appropriate prescriptions to their patients' profiles.

Admin View: Designed to enable administrators to manage essential system components and ensure smooth platform operation, the Admin View includes:

- **User Management**: Allows admins to view, create, edit, and delete profiles for doctors, patients, and other healthcare roles. Admins can assign user roles and manage basic access permissions.
- **Scheduling Management**: Enables admins to create and modify doctor schedules, ensuring efficient use of resources. Admins can also oversee patient appointments, allowing for easy management of the appointment pipeline.
- **Access Control**: Provides role-based permissions management, ensuring that each user has appropriate access based on their role, helping maintain data security and regulatory compliance.

1.3 Terms

Patient View: The DigiMED interface for patients, providing access to basic health information, appointments, and prescriptions.

Doctor View: The DigiMED interface for doctors, providing access to appointments, patients' medical records and prescriptions.

Admin View: The DigiMED interface for administrators, offering control over essential backend functions, including scheduling and user management.

Admin: Authorized personnel responsible for managing core user profiles, basic access, and scheduling.

Patient: A user accessing DigiMED to manage their healthcare needs, including appointments, medical records, and prescriptions.

Doctor: A user accessing DigiMED to practice their medical profession and interact with patients.

Appointment: A scheduled healthcare session between a patient and a doctor.

Schedule: A doctor's availability and booked appointments, managed by admins for optimized scheduling.

Prescription: A doctor-authorized medication plan available to patients through DigiMED.

Access Control: The ability for admins to assign permissions, limiting or allowing user access to specific features.

1.4 Abbreviations

SRS: Software Requirements Specification

HIPAA: Health Insurance Portability and Accountability Act

GDPR: General Data Protection Regulation

UI: User Interface

UX: User Experience

DBMS: Database Management System

PMRM: Patient Medical Records Management

2 Product Overview

2.1 Product Perspective

2.1.1 System Interfaces

2.1.2 User Interfaces

2.1.3 Hardware Interfaces

2.1.4 Software Interfaces

2.1.5 Communication Interfaces

2.1.6 Memory Constraints

2.1.7 Operations

2.1.8 Site Adaptation Requirements

2.1.9 Interface with Services

2.2 Product Functions

2.3 User Characteristics

2.4 Limitations

2.5 Assumptions and Dependencies

2.6 Apportioning of Requirements

1. **Share and Update a Doctor's Schedule:** Enables administrative staff to share and make updates to a doctor’s schedule as necessary.
2. **Access to Doctor's Schedule and Appointments:** Grants certain administrative users access to view and manage doctor schedules and appointments.
3. **Contact Patients Scheduled with Doctor:** Allows administrative staff to contact patients with scheduled appointments.
4. **Optimize Doctor’s Schedule:** Enables optimization of a doctor’s schedule based on patient needs and availability.
5. **Update Patient Medical Records:** Allows doctors to update and add new details to a patient's medical records.
6. **Write a Prescription for Patient:** Enables doctors to create and issue prescriptions directly for their patients.
7. **View My Scheduled Appointments:** Allows doctors to view all upcoming appointments in their schedules.
8. **Overwrite My Schedule:** Provides doctors the option to adjust their schedules directly when needed.
9. **Contact Patients' Previous Doctors:** Allows administrative staff to contact patients' previous doctors for additional medical information.
10. **Approve or Modify Online Appointment Requests:** Grants doctors the ability to approve or request modifications for online appointment bookings.

2.7 Specified Requirements

1. Profile Management

1.1 Edit an Existing Profile

1.2.1 The system shall enable the editing of the following profiles:

- **1.1.1 Patients:**
    - **1.1.1.1** The system shall allow patient users to update their profile.
    - **1.1.1.2** Patient users may also update fields that they had not previously entered.
    - **1.1.1.3** The fields that a patient may modify are: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, blood type, height, weight.
    - **1.1.1.4** The system shall send an HTTP PUT request to the Database Management Service (/profiles/patients) with updated information.
- **1.1.2 Doctors**
    - **1.1.2.1** The system shall allow doctor users to update their profile.
    - **1.1.2.2** Doctor users may also update fields that they had not previously entered.
    - **1.1.2.3** The fields that a doctor may modify are: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, specialty and subspecialty.
    - **1.1.2.4** The system shall send an HTTP PUT request to the Database. Management Service (/profiles/doctors) with updated information.

1.2 Retrieve a Profile

1.2.1 The system shall enable users to retrieve their profiles.

1.2.2. The system shall enable users with permission to view the profile of another user.

1.2.3 The system shall enforce access control based on user roles for retrieving profiles.

1.2.4 The system shall allow retrieval of the following profiles:

- **1.2.4.1** Patients
    - **1.2.4.1.1** The system shall retrieve a patient using PatientNatID.
    - **1.2.4.1.2** It shall send a POST request to /profiles/patients/retrieve.
    - **1.2.4.1.3** Only the patient in question or a doctor with permission may retrieve a patient profile.
- **1.2.4.2** Doctors
    - **1.2.4.2.1** The system shall retrieve a doctor using DoctorNatID.
    - **1.2.4.2.2** It shall send a POST request to /profiles/doctors/retrieve.
    - **1.2.4.2.3** Any user may retrieve the profile of a doctor.

1.3 Delete a profile (Admin Only)

1.3.1 The system shall permit admins to permanently remove a profile from the system.

1.3.2 Only users with the **"admin"** role shall be allowed to delete profiles.

1.3.3 The system shall validate the user's role through the Role header before deleting any record.

1.3.4 The system shall allow the deletion the following types of profiles:

- 1.3.4.1 Patient:
    - **1.3.4.1.1** Deletion using PatientNatID.
    - **1.3.4.1.2** The system shall send an HTTP DELETE request to /profiles/patients.
- 1.3.4.2 Doctor:
    - **1.3.4.2.1** Deletion using DoctorNatID.
    - **1.3.4.2.2** The system shall send an HTTP DELETE request to /profiles/doctors.

1.4 Create a Profile

1.4.1 The system shall allow users to create new profiles with relevant details.

1.4.2 The system shall allow the creation of the following profiles:

- 1.4.2.1 Patients:
  - **1.4.2.1.1** Patients can create a profile with the following fields: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, blood type, height, weight.
  - **1.4.2.1.2** The system shall send a POST request to /patients.
- 1.4.2.2 Doctor:
  - **1.4.2.2.1** Doctors can create a profile with the following fields: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, specialty and subspecialty.
  - **1.4.2.2.2** The system shall send a POST request to /profiles/doctors.

2. Patient Medical Records

2.1 Edit an Existing Profile

2.1.1 The system shall allow authorized doctors to modify details in an existing medical record for accuracy or updates.

2.1.2 The system shall enforce role-based access control, allowing only users with the **"doctor"** role to modify records.

2.1.3 The system shall allow doctors to edit the following fields:

- 2.1.3.1 Medical Conditions:
  - **2.1.2.1.1** Doctors shall be able to edit the **Notes** field related to a patient's medical condition.
  - **2.1.2.1.2** The system shall send an HTTP PUT request to the Database Management Service (/medicalConditions) with updated information.

- 2.1.3.2 Medical Tests:
    - **2.1.2.2.1** Doctors shall be able to edit the following fields: Test_Type, SubjectOfTest, Result, ImageOfScan, Date_TimeOfUpload.
    - **2.1.2.2.2** The system shall send an HTTP PUT request to the Database Management Service (/medicalTests) with updated information.
- 2.1.3.3 Doctor History:
    - **2.1.2.3.1** Doctors shall be able to edit the **startDate** of the treatment.
    - **2.1.2.3.2** The system shall send an HTTP PUT request to the Database Management Service (/doctorHistory) with updated information.

2.2 Retrieve a Record

2.2.1 The system shall enable users with permission to view a specific patient's medical record.

2.2.2 The system shall enforce access control based on user roles for retrieving records.

2.2.3 The system shall allow retrieval of the following records:

- 2.2.3.1 Medical Conditions:
    - **2.2.2.1.1** The system shall retrieve a medical condition using PatientNatID and MedCondition.
    - **2.2.2.1.2** It shall send a POST request to /medicalConditions/retrieve.
- 2.2.2.2 Medical Tests:
    - **2.2.2.2.1** The system shall retrieve a medical test using PatientNatID and TestID.
    - **2.2.2.2.2** It shall send a POST request to /medicalTests/retrieve.
- 2.2.2.3 Doctor History:
    - **2.2.2.3.1** The system shall retrieve treatment information using PatientNatID and DoctorNatID.
    - **2.2.2.3.2** It shall send a POST request to /doctorHistory/retrieve.

2.3 Delete a Record (Admin Only)

2.3.1 The system shall permit admins to permanently remove a patient's medical record from the system.

2.3.2 Only users with the **"admin"** role shall be allowed to delete records.

2.3.3 The system shall validate the user's role through the Role header before deleting any record.

2.3.4 The system shall delete the following types of records:

- 2.3.4.1 Medical Conditions:
  - **2.3.4.1.1** Deletion using PatientNatID and MedCondition.
  - **2.3.4.1.2** The system shall send an HTTP DELETE request to /medicalConditions.
- 2.3.4.2 Medical Tests:
  - **2.3.4.2.1** Deletion using PatientNatID and TestID.
  - **2.3.4.2.2** The system shall send an HTTP DELETE request to /medicalTests.
- 2.3.4.3 Doctor History:
  - **2.3.4.3.1** Deletion using PatientNatID and DoctorNatID.
  - **2.3.4.3.2** The system shall send an HTTP DELETE request to /doctorHistory.

2.4 Create a Record

2.4.1 The system shall allow doctors to add new medical records for patients with relevant details.

2.4.2 The system shall enforce role-based access control, allowing only doctors to create records.

2.4.3 The system shall allow the creation of the following records:

- 2.4.3.1 Medical Conditions:
  - **2.4.3.1.1** Doctors shall create a medical condition with PatientNatID, MedCondition, and Notes.
  - **2.4.3.1.2** The system shall send a POST request to /medicalConditions.
- 2.4.3.2 Medical Tests:
  - **2.4.3.2.1** Doctors shall create a medical test with PatientNatID, TestID, Test_Type, SubjectOfTest, Result, ImageOfScan, and Date_TimeOfUpload.
  - **2.4.3.2.2** The system shall send a POST request to /medicalTests.
- 2.4.3.3 Doctor History:
  - **2.4.2.3.1** Doctors shall create a treatment entry with PatientNatID, DoctorNatID, and startDate.
  - **2.4.2.3.2** The system shall send a POST request to /doctorHistory.

2.5 Access Management to Records

2.5.1 The system shall enforce permissions based on user roles, allowing or denying access to patient medical records.

2.5.2 The system shall check the Role header in all incoming requests to determine access levels. By default:

- **2.5.2.1** Doctors shall be able to create, edit, and retrieve medical records.
- **2.5.2.2** Admins shall have permissions to delete records.
- **2.5.2.3** Patients may be granted read-only access if required.

2.5.3 The system shall allow patients to modify privileges to their own medical records (future enhancement).

2.5.4 The AdminAuthorization class shall handle access control logic for routes restricted to admins.

3 Specific Requirements

3.1 External Interfaces

3.2 Functions

1. Profile Management

1.1.1.1 The system shall allow patient users to update their profile.

1.1.1.2 Patient users may also update fields that they had not previously entered.

1.1.1.3 The fields that a patient may modify are: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, blood type, height, weight.

- **Scenario:** A patient needs to update their profile, or add new fields not previously entered.
- **Function:** The system allows patient users to modify their own profiles.
- **Inputs:**
    - Patient ID (PatientNatID: string).
    - Updated details (e.g., Weight: float, Height: float, City: string).
- **Source:** API Gateway receives the request from the Patient UI.
- **Outputs:** A confirmation of successful updates.
- **Destination:** Updated records are stored in the Database Management System.
- **Action:**
    - The Patient UI sends an HTTP PUT request to the relevant endpoint (/profiles/patients).
    - The system forwards the request to the Database Management Service.
- **Requirements and Pre-condition:**
    - Only users with the "patient" role are authorized to perform this action.
    - Input fields must pass validation.
- **Post-condition and Side Effects:** The updated record is reflected in the database.

1.1.1.4 The system shall send an HTTP PUT request to the Database Management Service (/profiles/patient) with updated information.

- **Scenario:** The system communicates the updated profile data to the database.
- **Function:** Send a structured PUT request with the updated fields.
- **Inputs:** JSON payload containing updated details.
- **Source:** Internally generated request.
- **Outputs:** HTTP status indicating success or failure.
- **Destination:** Database Management System (/profiles/patients).
- **Action:** Format and send the request to the /patients endpoint.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - Endpoint must be available.
- **Post-condition and Side Effects:** Updated patient profile details are stored in the database.

1.1.2.1 The system shall allow doctor users to update their profile.

1.1.2.2 Doctor users may also update fields that they had not previously entered.

1.1.2.3 The fields that a doctor may modify are: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, specialty and subspecialty.

- **Scenario:** A doctor needs to update their profile, or add new fields not previously entered.
- **Function:** The system allows doctor users to modify their own profiles.
- **Inputs:**
    - Doctor ID (DoctorNatID: string).
    - Updated details (e.g., Specialty: string, Subspecialty: string).
- **Source:** API Gateway receives the request from the Doctor UI.
- **Outputs:** A confirmation of successful updates.
- **Destination:** Updated records are stored in the Database Management System.
- **Action:**

    - The Doctor UI sends an HTTP PUT request to the relevant endpoint (/profiles/doctors).
    - The system forwards the request to the Database Management Service.
- **Requirements and Pre-condition:**
    - Only users with the "doctor" role are authorized to perform this action.
    - Input fields must pass validation.
- **Post-condition and Side Effects:** The updated record is reflected in the database.

1.1.2.4 The system shall send an HTTP PUT request to the Database. Management Service (/profiles/doctor) with updated information.

- **Scenario:** The system communicates the updated profile data to the database.
- **Function:** Send a structured PUT request with the updated fields.
- **Inputs:** JSON payload containing updated details.
- **Source:** Internally generated request.
- **Outputs:** HTTP status indicating success or failure.
- **Destination:** Database Management System (/doctors).
- **Action:** Format and send the request to the /doctors endpoint.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - Endpoint must be available.
- **Post-condition and Side Effects:** Updated doctor profile details are stored in the database.

1.2 Retrieve a Profile

1.2.4.1.1 The system shall retrieve a patient using PatientNatID.

1.2.4.1.3 Only the patient in question or a doctor with permission may retrieve a patient profile.

- **Scenario:** A patient or doctor user requests to see a patient profile.
- **Function:** Retrieve a patient profile from the database.

- **Inputs:**
    - Patient ID (PatientNatID: string).
- **Source:** API Gateway processes the request.
- **Outputs:** Profile details.
- **Destination:** Database Management System (profiles/patients/retrieve).
- **Action:** The system sends a structured request to retrieve the profile.
- **Requirements and Pre-condition:**
    - Valid PatientNatID must be provided.
    - The user must have permission to view this profile.
- **Post-condition and Side Effects:** The record is retrieved and displayed to the user.

1.2.4.1.2 It shall send a POST request to /profiles/patients/retrieve.

- **Scenario:** The system retrieves a patient profile record.
- **Function:** Send a POST request to fetch the condition.
- **Inputs:**
    - JSON payload with PatientNatID.
- **Source:** Internal system process.
- **Outputs:** HTTP response containing the patient profile.
- **Destination:** Database Management System (/profiles/patients/retrieve).
- **Action:** Send a structured POST request.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The condition details are fetched from the database.

1.2.4.2.1 The system shall retrieve a doctor using DoctorNatID.

1.2.4.2.3 Any user may retrieve the profile of a doctor.

- **Scenario:** A user requests to see a doctor profile.
- **Function:** Retrieve a doctor profile from the database.

- **Inputs:**
    - Doctor ID (DoctorNatID: string).
- **Source:** API Gateway processes the request.
- **Outputs:** Profile details.
- **Destination:** Database Management System (profiles/doctors/retrieve).
- **Action:** The system sends a structured request to retrieve the profile.
- **Requirements and Pre-condition:**
    - Valid DoctorNatID must be provided.
- **Post-condition and Side Effects:** The record is retrieved and displayed to the user.

1.2.4.2.2 It shall send a POST request to /profiles/doctors/retrieve.

- **Scenario:** The system retrieves a doctor profile record.
- **Function:** Send a POST request to fetch the condition.
- **Inputs:**
    - JSON payload with DoctorNatID.
- **Source:** Internal system process.
- **Outputs:** HTTP response containing the patient profile.
- **Destination:** Database Management System (/profiles/doctors/retrieve).
- **Action:** Send a structured POST request.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The condition details are fetched from the database.

1.3 Delete a profile (Admin Only)

1.3.1 The system shall permit admins to permanently remove a profile from the system.

- **Scenario:** An admin wishes to delete a profile from the system.
- **Function:** The system allows admins to permanently delete patients.

- **Inputs:**
    - NatID (PatientNatID or DoctorNatID, string).
- **Source:** Admin UI sends a delete request through the API Gateway.
- **Outputs:** Confirmation of successful deletion.
- **Destination:** The profile is removed from the Database Management System.
- **Action:**
    - The Admin UI sends an HTTP DELETE request to the relevant endpoint (/profiles/patients or /profiles/doctors).
    - The system forwards the request to the Database Management Service.
- **Requirements and Pre-condition:**
    - Only users with the "admin" role are authorized to perform this action.
    - Valid NatID and record-specific identifiers must be provided.
- **Post-condition and Side Effects:**
    - The record is permanently deleted from the system.
    - Related data may also be deleted.

1.3.2 Only users with the **"admin"** role shall be allowed to delete profiles.

- **Scenario:** A user attempts to delete a profile.
- **Function:** Restrict delete privileges to admins only.
- **Inputs:** Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Access granted or denied.
- **Destination:** Internal role-checking logic (handled by the require_admin decorator in the AdminAuthorization class).
- **Action:** Validate the user's role before granting delete permissions.
- **Requirements and Pre-condition:**
    - The Role header must indicate "admin".
    - Unauthorized users are denied access to delete functionality.
- **Post-condition and Side Effects:** Only admins can delete records.

1.3.3 The system shall validate the user's role through the Role header before deleting any profile.

- **Scenario:** The system ensures only admins can delete profiles.
- **Function:** Validate user roles before performing deletion.
- **Inputs:** Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Validation success or failure message.
- **Destination:** Internal validation logic.
- **Action:** The system validates the Role header value before processing the delete request.
- **Requirements and Pre-condition:**
    - The Role header must indicate "admin".
    - Validation logic must block unauthorized roles.
- **Post-condition and Side Effects:** The delete operation is only processed for admins.

1.3.4 The system shall allow the deletion the following types of profiles:

1.3.4.1 Patient:

1.3.4.1.1 Deletion using PatientNatID.

1.3.4.1.2 The system shall send an HTTP DELETE request to /patients.

1.3.4.2 Doctor:

1.3.4.2.1 Deletion using DoctorNatID.

1.3.4.2.2 The system shall send an HTTP DELETE request to /doctors.

1.4 Create a Profile

1.4.2.1.1 Patients can create a profile with the following fields: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, blood type, height, weight.

- **Scenario:** A patient creates a new profile.
- **Function:** The system allows patients to create new profiles.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Other relevant data (e.g., First Name: string, Last Name: string, Date of Birth, date).
- **Source:** Patient UI sends a create request through the API Gateway.
- **Outputs:** Confirmation of successful record creation.
- **Destination:** Database Management System.
- **Action:**
    - The Patient UI sends an HTTP POST request to the relevant endpoint /profiles/patient.
    - The system forwards the request to the Database Management Service.
- **Requirements and Pre-condition:**
    - All required inputs must pass validation.
- **Post-condition and Side Effects:**
    - The new profile is added to the system.

1.4.2.1.2 The system shall send a POST request to /profiles/patients.

- **Scenario:** The system transmits a request to create a patient profile.
- **Function:** Send a structured HTTP POST request.
- **Inputs:**
    - JSON payload: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, blood type, height, weight.
- **Source:** Internal system process.
- **Outputs:** HTTP response indicating success or failure.
- **Destination:** Database Management Service (/profiles/patients).
- **Action:** Format and send the request to /profiles/patients.
- **Requirements and Pre-condition:**

    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The new patient profile is stored in the database.

1.4.2.2.1 Doctors can create a profile with the following fields: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, specialty and subspecialty.

- **Scenario:** A doctor creates a new profile.
- **Function:** The system allows doctors to create profiles.
- **Inputs:**
    - Doctor ID (DoctorNatID, string).
    - Other relevant data (e.g., First Name: string, Last Name: string, Specialty: string).
- **Source:** Doctor UI sends a create request through the API Gateway.
- **Outputs:** Confirmation of successful record creation.
- **Destination:** Database Management System.
- **Action:**
    - The Doctor UI sends an HTTP POST request to the relevant endpoint (/patients/profile).
    - The system forwards the request to the Database Management Service.
- **Requirements and Pre-condition:**
    - All required inputs must pass validation.
- **Post-condition and Side Effects:**
    - The new record is added to the system.

1.4.2.2.2 The system shall send a POST request to /profiles/doctors.

- **Scenario:** The system transmits a request to create a doctor profile.
- **Function:** Send a structured HTTP POST request.
- **Inputs:**

    - JSON payload: first name, middle name, last name, street, region, city, phone number, email, gender, date of birth, specialty and subspecialty. .
- **Source:** Internal system process.
- **Outputs:** HTTP response indicating success or failure.
- **Destination:** Database Management Service (/profiles/doctors).
- **Action:** Format and send the request to /profiles/doctors.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The new doctor profile is stored in the database.

---

2. Patient Medical Records

2.1 Edit an Existing Record

2.1.1 The system shall allow authorized doctors to modify details in an existing medical record for accuracy or updates.

- **Scenario:** A doctor needs to update a patient's medical record for corrections or updates.
- **Function:** The system allows doctors to modify records for accuracy.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Record-specific identifier (MedCondition, TestID, or DoctorNatID, string).
    - Updated details (e.g., Notes, Test_Type, startDate, string).
- **Source:** API Gateway receives the request from the Doctor UI.
- **Outputs:** A confirmation of successful updates.
- **Destination:** Updated records are stored in the Database Management System.
- **Action:**

    - The Doctor UI sends an HTTP PUT request to the relevant endpoint (/medicalConditions, /medicalTests, or /doctorHistory).
    - The system forwards the request to the Database Management Service.
- **Requirements and Pre-condition:**
    - Only users with the "doctor" role are authorized to perform this action.
    - Input fields must pass validation.
- **Post-condition and Side Effects:** The updated record is reflected in the database.

2.1.2 The system shall enforce role-based access control, allowing only users with the "doctor" role to modify records.

- **Scenario:** A user attempts to edit a medical record.
- **Function:** Restrict editing privileges to authorized users.
- **Inputs:** Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Access granted or denied.
- **Destination:** Internal role-checking logic.
- **Action:** Validate the user’s role before granting edit permissions.
- **Requirements and Pre-condition:**
    - The Role header must indicate "doctor".
    - Unauthorized users are denied editing access.
- **Post-condition and Side Effects:** Only doctors can modify records.

2.1.3 The system shall allow doctors to edit the following fields:

2.1.3.1 Medical Conditions

2.1.3.1.1 Doctors shall be able to edit the Notes field related to a patient's medical condition.

- **Scenario:** A doctor modifies the notes field in a patient’s condition record.
- **Function:** Update the Notes field.

- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Condition identifier (MedCondition, string).
    - New notes (Notes, string).
- **Source:** API Gateway processes a PUT request from the Doctor UI.
- **Outputs:** Confirmation of the updated notes field.
- **Destination:** Database Management Service endpoint /medicalConditions.
- **Action:** Send a JSON payload with the updated notes.
- **Requirements and Pre-condition:** Valid PatientNatID and MedCondition must exist in the database.
- **Post-condition and Side Effects:** The notes are updated in the system.

2.1.3.1.2 The system shall send an HTTP PUT request to the Database Management Service (/medicalConditions) with updated information.

- **Scenario:** The system communicates updated medical condition details to the database.
- **Function:** Transmit a PUT request with the updated data.
- **Inputs:**
    - JSON payload: PatientNatID, MedCondition, and updated Notes.
- **Source:** Internal system request after user input.
- **Outputs:** HTTP status indicating success or failure.
- **Destination:** Database Management System (/medicalConditions).
- **Action:** Format and send the PUT request to the database service.
- **Requirements and Pre-condition:**
    - The payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The database reflects the updated Notes.

2.1.3.2 Medical Tests

2.1.3.2.1 Doctors shall be able to edit the following fields: Test_Type, SubjectOfTest, Result, ImageOfScan, Date_TimeOfUpload.

- **Scenario:** A doctor updates details of a medical test.
- **Function:** Modify fields in a medical test record.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Test identifier (TestID, string).
    - Updated fields (Test_Type, SubjectOfTest, Result, ImageOfScan, Date_TimeOfUpload, string).
- **Source:** API Gateway processes the PUT request from the Doctor UI.
- **Outputs:** Confirmation of successful updates.
- **Destination:** Database Management System (/medicalTests).
- **Action:** Send an HTTP PUT request to the /medicalTests endpoint.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and TestID must exist in the database.
    - Fields must pass validation.
- **Post-condition and Side Effects:** Updated fields are reflected in the database.

2.1.3.2.2 The system shall send an HTTP PUT request to the Database Management Service (/medicalTests) with updated information.

- **Scenario:** The system communicates the updated test details to the database.
- **Function:** Send a structured PUT request with the updated fields.
- **Inputs:** JSON payload containing updated details.
- **Source:** Internally generated request.
- **Outputs:** HTTP status indicating success or failure.
- **Destination:** Database Management System (/medicalTests).
- **Action:** Format and send the request to the /medicalTests endpoint.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.

    - Endpoint must be available.
- **Post-condition and Side Effects:** Updated test details are stored in the database.

2.1.3.3 Doctor History

2.1.3.3.1 Doctors shall be able to edit the startDate of the treatment.

- **Scenario:** A doctor modifies the treatment start date for a patient.
- **Function:** Update the startDate field.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Doctor ID (DoctorNatID, string).
    - Updated start date (startDate, string).
- **Source:** API Gateway processes a PUT request from the Doctor UI.
- **Outputs:** Confirmation of the updated startDate.
- **Destination:** Database Management Service endpoint /doctorHistory.
- **Action:** Send a JSON payload with the updated startDate.
- **Requirements and Pre-condition:** Valid PatientNatID and DoctorNatID must exist in the database.
- **Post-condition and Side Effects:** The updated startDate is reflected in the system.

2.1.3.3.2 The system shall send an HTTP PUT request to the Database Management Service (/doctorHistory) with updated information.

- **Scenario:** The system communicates the updated treatment details to the database.
- **Function:** Send a structured PUT request with the updated startDate.
- **Inputs:** JSON payload containing updated startDate.
- **Source:** Internally generated request.
- **Outputs:** HTTP status indicating success or failure.
- **Destination:** Database Management System (/doctorHistory).
- **Action:** Format and send the request to the /doctorHistory endpoint.
- **Requirements and Pre-condition:**

    - JSON payload must include all required fields.
    - Endpoint must be available.
- **Post-condition and Side Effects:** Updated treatment details are stored in the database.

2.2 Retrieve a Record

2.2.1 The system shall enable users with permission to view a specific patient's medical record.

- **Scenario:** A user (doctor or patient) requests access to view a medical record.
- **Function:** Retrieve medical records.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Record-specific identifier (MedCondition, TestID, or DoctorNatID, string).
- **Source:** API Gateway receives requests from the user interface (Doctor UI or Patient UI).
- **Outputs:** The requested medical record.
- **Destination:** Database Management System.
- **Action:**
    - The user sends a request to view a record.
    - The system retrieves the record from the database and returns it.
- **Requirements and Pre-condition:**
    - The system must validate user permissions before granting access.
    - Inputs must be valid and match existing records in the database.
- **Post-condition and Side Effects:** The requested record is displayed to the user.

2.2.2 The system shall enforce access control based on user roles for retrieving records.

- **Scenario:** A user attempts to retrieve a medical record.
- **Function:** Restrict access to records based on user roles.
- **Inputs:** Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Access granted or denied.
- **Destination:** Internal role-checking logic.

- **Action:** Validate the user's role before granting access to records.
- **Requirements and Pre-condition:**
    - User roles must be validated before accessing records.
    - Unauthorized users are denied access.
- **Post-condition and Side Effects:** Only users with valid permissions can access records.

2.2.3 The system shall allow retrieval of the following records:

2.2.3.1 Medical Conditions

2.2.3.1.1 The system shall retrieve a medical condition using PatientNatID and MedCondition.

- **Scenario:** A user requests a medical condition record.
- **Function:** Retrieve a medical condition from the database.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Medical condition identifier (MedCondition, string).
- **Source:** API Gateway processes the request.
- **Outputs:** Medical condition details.
- **Destination:** Database Management System (/medicalConditions/retrieve).
- **Action:** The system sends a structured request to retrieve the condition.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and MedCondition must be provided.
    - The user must have proper permissions.
- **Post-condition and Side Effects:** The record is retrieved and displayed to the user.

2.2.3.1.2 The system shall send a POST request to /medicalConditions/retrieve.

- **Scenario:** The system retrieves a medical condition record.
- **Function:** Send a POST request to fetch the condition.

- **Inputs:**
    - JSON payload with PatientNatID and MedCondition.
- **Source:** Internal system process.
- **Outputs:** HTTP response containing the medical condition.
- **Destination:** Database Management System (/medicalConditions/retrieve).
- **Action:** Send a structured POST request.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The condition details are fetched from the database.

2.2.3.2 Medical Tests

2.2.3.2.1 The system shall retrieve a medical test using PatientNatID and TestID.

- **Scenario:** A user requests a medical test record.
- **Function:** Retrieve a medical test from the database.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Test identifier (TestID, string).
- **Source:** API Gateway processes the request.
- **Outputs:** Medical test details.
- **Destination:** Database Management System (/medicalTests/retrieve).
- **Action:** The system sends a structured request to retrieve the test.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and TestID must be provided.
    - The user must have proper permissions.
- **Post-condition and Side Effects:** The record is retrieved and displayed to the user.

2.2.3.2.2 The system shall send a POST request to /medicalTests/retrieve.

- **Scenario:** The system retrieves a medical test record.
- **Function:** Send a POST request to fetch the test.
- **Inputs:**
    - JSON payload with PatientNatID and TestID.
- **Source:** Internal system process.
- **Outputs:** HTTP response containing the medical test.
- **Destination:** Database Management System (/medicalTests/retrieve).
- **Action:** Send a structured POST request.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The test details are fetched from the database.

2.2.3.3 Doctor History

2.2.3.3.1 The system shall retrieve treatment information using PatientNatID and DoctorNatID.

- **Scenario:** A user requests treatment information.
- **Function:** Retrieve treatment data from the database.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Doctor ID (DoctorNatID, string).
- **Source:** API Gateway processes the request.
- **Outputs:** Treatment details.
- **Destination:** Database Management System (/doctorHistory/retrieve).
- **Action:** The system sends a structured request to retrieve treatment information.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and DoctorNatID must be provided.

    - The user must have proper permissions.
- **Post-condition and Side Effects:** The record is retrieved and displayed to the user.

2.2.3.3.2 The system shall send a POST request to /doctorHistory/retrieve.

- **Scenario:** The system retrieves treatment information.
- **Function:** Send a POST request to fetch treatment details.
- **Inputs:**
    - JSON payload with PatientNatID and DoctorNatID.
- **Source:** Internal system process.
- **Outputs:** HTTP response containing the treatment information.
- **Destination:** Database Management System (/doctorHistory/retrieve).
- **Action:** Send a structured POST request.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The treatment details are fetched from the database.

2.3 Delete a Record (Admin Only)

2.3.1 The system shall permit admins to permanently remove a patient's medical record from the system.

- **Scenario:** An admin deletes outdated or incorrect records from the system.
- **Function:** The system allows admins to permanently delete records.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Record-specific identifier (MedCondition, TestID, or DoctorNatID, string).
- **Source:** Admin UI sends a delete request through the API Gateway.
- **Outputs:** Confirmation of successful deletion.
- **Destination:** The record is removed from the Database Management System.
- **Action:**

    - The Admin UI sends an HTTP DELETE request to the relevant endpoint (/medicalConditions, /medicalTests, or /doctorHistory).
    - The system forwards the request to the Database Management Service.
- **Requirements and Pre-condition:**
    - Only users with the "admin" role are authorized to perform this action.
    - Valid Patient ID and record-specific identifiers must be provided.
- **Post-condition and Side Effects:**
    - The record is permanently deleted from the system.
    - Related data may also be deleted.

2.3.2 Only users with the "admin" role shall be allowed to delete records.

- **Scenario:** A user attempts to delete a record.
- **Function:** Restrict delete privileges to admins only.
- **Inputs:** Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Access granted or denied.
- **Destination:** Internal role-checking logic (handled by the require_admin decorator in the AdminAuthorization class).
- **Action:** Validate the user’s role before granting delete permissions.
- **Requirements and Pre-condition:**
    - The Role header must indicate "admin".
    - Unauthorized users are denied access to delete functionality.
- **Post-condition and Side Effects:** Only admins can delete records.

2.3.3 The system shall validate the user's role through the Role header before deleting any record.

- **Scenario:** The system ensures only admins can delete records.
- **Function:** Validate user roles before performing deletion.
- **Inputs:** Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Validation success or failure message.
- **Destination:** Internal validation logic.
- **Action:** The system validates the Role header value before processing the delete request.
- **Requirements and Pre-condition:**
    - The Role header must indicate "admin".
    - Validation logic must block unauthorized roles.
- **Post-condition and Side Effects:** The delete operation is only processed for admins.

2.3.4 The system shall delete the following types of records:

2.3.4.1 Medical Conditions

2.3.4.1.1 Deletion using PatientNatID and MedCondition.

- **Scenario:** An admin deletes a specific medical condition record.
- **Function:** Delete a medical condition from the database.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Medical condition identifier (MedCondition, string).
- **Source:** Admin UI sends a delete request via the API Gateway.
- **Outputs:** Confirmation of successful deletion.
- **Destination:** Database Management Service (/medicalConditions).
- **Action:** The system sends a structured DELETE request to /medicalConditions.
- **Requirements and Pre-condition:**

    - Valid PatientNatID and MedCondition must exist in the database.
    - The user must have admin permissions.
- **Post-condition and Side Effects:** The condition record is permanently removed from the system.

2.3.4.1.2 The system shall send an HTTP DELETE request to /medicalConditions.

- **Scenario:** The system sends a delete request to the Database Management Service for a medical condition.
- **Function:** Transmit a structured HTTP DELETE request.
- **Inputs:**
    - JSON payload: PatientNatID, MedCondition.
- **Source:** Internal system process.
- **Outputs:** HTTP response indicating success or failure.
- **Destination:** Database Management Service (/medicalConditions).
- **Action:** The system formats and sends the DELETE request.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The medical condition is deleted from the database.

2.3.4.2 Medical Tests

2.3.4.2.1 Deletion using PatientNatID and TestID.

- **Scenario:** An admin deletes a specific medical test record.
- **Function:** Delete a medical test from the database.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Test identifier (TestID, string).

- **Source:** Admin UI sends a delete request via the API Gateway.
- **Outputs:** Confirmation of successful deletion.
- **Destination:** Database Management Service (/medicalTests).
- **Action:** The system sends a structured DELETE request to /medicalTests.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and TestID must exist in the database.
    - The user must have admin permissions.
- **Post-condition and Side Effects:** The test record is permanently removed from the system.

2.3.4.2.2 The system shall send an HTTP DELETE request to /medicalTests.

- **Scenario:** The system sends a delete request to the Database Management Service for a medical test.
- **Function:** Transmit a structured HTTP DELETE request.
- **Inputs:**
    - JSON payload: PatientNatID, TestID.
- **Source:** Internal system process.
- **Outputs:** HTTP response indicating success or failure.
- **Destination:** Database Management Service (/medicalTests).
- **Action:** The system formats and sends the DELETE request.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The medical test is deleted from the database.

2.3.4.3 Doctor History

2.3.4.3.1 Deletion using PatientNatID and DoctorNatID.

- **Scenario:** An admin deletes a specific treatment record.
- **Function:** Delete a treatment record from the database.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Doctor ID (DoctorNatID, string).
- **Source:** Admin UI sends a delete request via the API Gateway.
- **Outputs:** Confirmation of successful deletion.
- **Destination:** Database Management Service (/doctorHistory).
- **Action:** The system sends a structured DELETE request to /doctorHistory.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and DoctorNatID must exist in the database.
    - The user must have admin permissions.
- **Post-condition and Side Effects:** The treatment record is permanently removed from the system.

2.3.4.3.2 The system shall send an HTTP DELETE request to /doctorHistory.

- **Scenario:** The system sends a delete request to the Database Management Service for a treatment record.
- **Function:** Transmit a structured HTTP DELETE request.
- **Inputs:**
    - JSON payload: PatientNatID, DoctorNatID.
- **Source:** Internal system process.
- **Outputs:** HTTP response indicating success or failure.
- **Destination:** Database Management Service (/doctorHistory).
- **Action:** The system formats and sends the DELETE request.
- **Requirements and Pre-condition:**

    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The treatment record is deleted from the database.

2.4 Create a Record

2.4.1 The system shall allow doctors to add new medical records for patients with relevant details.

- **Scenario:** A doctor creates a new medical record for a patient.
- **Function:** The system allows doctors to add new records to the database.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Record-specific details (e.g., MedCondition, TestID, DoctorNatID, or associated data, string).
- **Source:** Doctor UI sends a create request through the API Gateway.
- **Outputs:** Confirmation of successful record creation.
- **Destination:** Database Management System.
- **Action:**
    - The Doctor UI sends an HTTP POST request to the relevant endpoint (/medicalConditions, /medicalTests, or /doctorHistory).
    - The system forwards the request to the Database Management Service.
- **Requirements and Pre-condition:**
    - Only users with the "doctor" role are authorized to create records.
    - All required inputs must pass validation.
- **Post-condition and Side Effects:**
    - The new record is added to the system.
    - Optional: Notifications may be sent to the patient or relevant stakeholders.

2.4.2 The system shall enforce role-based access control, allowing only doctors to create records.

- **Scenario:** A user attempts to create a new medical record.
- **Function:** Restrict create privileges to authorized users.
- **Inputs:** Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Access granted or denied.
- **Destination:** Internal role-checking logic (handled by the AdminAuthorization class).
- **Action:** Validate the user's role before processing the create request.
- **Requirements and Pre-condition:**
    - The Role header must indicate "doctor".
    - Unauthorized users are denied access to create functionality.
- **Post-condition and Side Effects:** Only doctors can create records.

2.4.3 The system shall allow the creation of the following records:

2.4.3.1 Medical Conditions

2.4.3.1.1 Doctors shall create a medical condition with PatientNatID, MedCondition, and Notes.

- **Scenario:** A doctor adds a new medical condition for a patient.
- **Function:** Create a new record in the Medical Conditions table.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Medical condition identifier (MedCondition, string).
    - Notes (Notes, string).
- **Source:** Doctor UI sends the request via the API Gateway.
- **Outputs:** Confirmation of successful record creation.
- **Destination:** Database Management Service (/medicalConditions).

- **Action:** The system sends a structured POST request to /medicalConditions.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and MedCondition must be provided.
    - Notes must pass validation.
- **Post-condition and Side Effects:** The new medical condition is added to the system.

2.4.3.1.2 The system shall send a POST request to /medicalConditions.

- **Scenario:** The system transmits a request to create a medical condition.
- **Function:** Send a structured HTTP POST request.
- **Inputs:**
    - JSON payload: PatientNatID, MedCondition, Notes.
- **Source:** Internal system process.
- **Outputs:** HTTP response indicating success or failure.
- **Destination:** Database Management Service (/medicalConditions).
- **Action:** Format and send the request to /medicalConditions.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The new medical condition is stored in the database.

2.4.3.2 Medical Tests

2.4.3.2.1 Doctors shall create a medical test with PatientNatID, TestID, Test_Type, SubjectOfTest, Result, ImageOfScan, and Date_TimeOfUpload.

- **Scenario:** A doctor adds a new medical test for a patient.
- **Function:** Create a new record in the Medical Tests table.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Test identifier (TestID, string).

    - Additional fields: Test_Type, SubjectOfTest, Result, ImageOfScan, Date_TimeOfUpload (strings).
- **Source:** Doctor UI sends the request via the API Gateway.
- **Outputs:** Confirmation of successful record creation.
- **Destination:** Database Management Service (/medicalTests).
- **Action:** The system sends a structured POST request to /medicalTests.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and TestID must be provided.
    - Fields must pass validation.
- **Post-condition and Side Effects:** The new medical test is added to the system.

2.4.3.2.2 The system shall send a POST request to /medicalTests.

- **Scenario:** The system transmits a request to create a medical test.
- **Function:** Send a structured HTTP POST request.
- **Inputs:**
    - JSON payload: PatientNatID, TestID, Test_Type, SubjectOfTest, Result, ImageOfScan, Date_TimeOfUpload.
- **Source:** Internal system process.
- **Outputs:** HTTP response indicating success or failure.
- **Destination:** Database Management Service (/medicalTests).
- **Action:** Format and send the request to /medicalTests.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The new medical test is stored in the database.

2.4.3.3 Doctor History

2.4.3.3.1 Doctors shall create a treatment entry with PatientNatID, DoctorNatID, and startDate.

- **Scenario:** A doctor adds a new treatment record for a patient.
- **Function:** Create a new record in the Doctor History table.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Doctor ID (DoctorNatID, string).
    - Start date (startDate, string).
- **Source:** Doctor UI sends the request via the API Gateway.
- **Outputs:** Confirmation of successful record creation.
- **Destination:** Database Management Service (/doctorHistory).
- **Action:** The system sends a structured POST request to /doctorHistory.
- **Requirements and Pre-condition:**
    - Valid PatientNatID and DoctorNatID must be provided.
    - Start date must pass validation.
- **Post-condition and Side Effects:** The new treatment entry is added to the system.

2.4.3.3.2 The system shall send a POST request to /doctorHistory.

- **Scenario:** The system transmits a request to create a treatment entry.
- **Function:** Send a structured HTTP POST request.
- **Inputs:**
    - JSON payload: PatientNatID, DoctorNatID, startDate.
- **Source:** Internal system process.
- **Outputs:** HTTP response indicating success or failure.
- **Destination:** Database Management Service (/doctorHistory).
- **Action:** Format and send the request to /doctorHistory.
- **Requirements and Pre-condition:**
    - JSON payload must include all required fields.
    - The endpoint must be reachable.
- **Post-condition and Side Effects:** The new treatment entry is stored in the database.

2.5 Access Management to Records

2.5.1 The system shall enforce permissions based on user roles, allowing or denying access to patient medical records.

- **Scenario:** Users request access to perform operations on medical records.
- **Function:** Restrict or grant access to medical records based on user roles.
- **Inputs:**
    - User role (Role, string).
    - Patient ID (PatientNatID, string).
    - Operation type (create, edit, retrieve, or delete).
- **Source:** Role header in the HTTP request.
- **Outputs:** Access granted or denied based on role validation.
- **Destination:** Internal access control logic.
- **Action:** Validate the user's role before processing the operation request.
- **Requirements and Pre-condition:**
    - User roles must be predefined and validated.
    - Unauthorized users are denied access.
- **Post-condition and Side Effects:** Only users with valid permissions can perform the requested operation.

2.5.2 The system shall check the Role header in all incoming requests to determine access levels. By default:

2.5.2.1 Doctors shall be able to create, edit, and retrieve medical records.

- **Scenario:** A doctor performs a CRUD operation on medical records.
- **Function:** Grant doctors the ability to create, edit, or retrieve records.
- **Inputs:**
    - Patient ID (PatientNatID, string).

    - Record-specific identifier (MedCondition, TestID, or DoctorNatID, string).
    - Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Confirmation of access for CRUD operations.
- **Destination:** Internal access control logic.
- **Action:** Validate the user's role as "doctor" before granting access.
- **Requirements and Pre-condition:**
    - The Role header must indicate "doctor".
    - Inputs must be valid and match existing records.
- **Post-condition and Side Effects:** Doctors can perform CRUD operations on medical records.

2.5.2.2 Admins shall have permissions to delete records.

- **Scenario:** An admin performs a delete operation on a record.
- **Function:** Grant admins the ability to delete records.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Record-specific identifier (MedCondition, TestID, or DoctorNatID, string).
    - Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Confirmation of access for delete operations.
- **Destination:** Internal access control logic.
- **Action:** Validate the user's role as "admin" before granting access.
- **Requirements and Pre-condition:**
    - The Role header must indicate "admin".
    - Inputs must be valid and match existing records.
- **Post-condition and Side Effects:** Admins can perform delete operations on medical records.

2.5.2.3 Patients may be granted read-only access if required.

- **Scenario:** A patient requests to view their medical record.
- **Function:** Grant patients read-only access to their medical records.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Medical record details for viewing.
- **Destination:** Patient UI displays the retrieved record.
- **Action:** Validate the user's role as "patient" and grant read-only access.
- **Requirements and Pre-condition:**
    - The Role header must indicate "patient".
    - Inputs must be valid and match existing records.
- **Post-condition and Side Effects:** Patients can view their own records but cannot edit or delete them.

2.5.3 The system shall allow patients to modify privileges to their own medical records (future enhancement).

- **Scenario:** A patient adjusts the sharing permissions of their medical records.
- **Function:** Allow patients to manage access to their own records.
- **Inputs:**
    - Patient ID (PatientNatID, string).
    - New permissions (list of users or roles with access).
- **Source:** Patient UI sends the request via the API Gateway.
- **Outputs:** Confirmation of updated permissions.
- **Destination:** Database Management System updates access rules.
- **Action:** The system processes the request to update access permissions.
- **Requirements and Pre-condition:**
    - Patients must authenticate themselves before modifying permissions.
    - Inputs must specify valid roles or user IDs.

- **Post-condition and Side Effects:** Access permissions for the patient's records are updated.

2.5.4 The AdminAuthorization class shall handle access control logic for routes restricted to admins.

- **Scenario:** Admin-only routes are accessed.
- **Function:** Restrict access to admin-specific routes using the AdminAuthorization class.
- **Inputs:**
    - Role (Role, string).
- **Source:** Role header in the HTTP request.
- **Outputs:** Access granted or denied.
- **Destination:** Internal role validation logic (require_admin decorator).
- **Action:** The system checks the Role header and denies access to non-admin users.
- **Requirements and Pre-condition:**
    - The Role header must indicate "admin".
- **Post-condition and Side Effects:** Only admins can access admin-restricted routes.

---

3. Scheduling System

3.1 Create a Schedule

3.1.1 The system should check the roles of the coming requests of creating a schedule.

3.1.1.1 Only Doctors should be allowed to create their schedule

- **Scenario:** A doctor wants to set their availability for appointments.
- **Function:** A doctor can create a new schedule.
- **Description:** The scheduling management service allows doctors to specify their available time slots.
- **Inputs:** Doctor ID (string), available dates and times (string).

- **Source:** The client request source is from the API Gateway; the doctor creates the schedule through the Doctor UI.
- **Outputs:** Confirmation of the newly created schedule.
- **Destination:** The schedule is saved in the Database management system.
- **Action:** The doctor sends their set times for appointments.
- **Requirements and Pre-condition:**
    - The system must validate the input for availability conflicts.
    - The system must check that the user requesting to create that schedule is authorized too.
- **Post-condition and Side effects:**
    - The schedule is set.

3.2 Create an Appointment in a Schedule

3.2.1 The system should check the requests coming for booking an appointment.

3.2.1.1 Patients should be allowed to book an appointment with the doctor of their choice.

- **Scenario:** A patient wants to book an appointment with a doctor or rebook an appointment.
- **Function:** A user can send a request to book an appointment on a specific date.
- **Description:** The appointment system allows users to book time slots in the available schedules and based on whether the doctor is giving access to only their current patients for follow ups.
- **Inputs:** Patient ID (string), Doctor ID (string), selected date and time (string).
- **Source:** The client request source is from the API Gateway; the user books the appointment through the Patient UI.
- **Outputs:** Confirmation of the booked appointment.
- **Destination:** The appointment is stored in the Database management system.
- **Action:** The user selects the time slot and confirms the appointment.
- **Requirements and Pre-condition:**

    - The system must ensure the time slot is available before confirming the appointment.
    - The system must check if the doctor is allowing new patients to schedule appointments.
- **Post-condition and Side effects:**
    - The appointment is confirmed.
    - notifications are sent to both doctor and patient.

3.2.1.2 Doctor may reserve appointments for specific patients

- **Scenario:** A doctor might have a patient that is not on the digimed system
- **Function:** A doctor can send a request to reserve an appointment on a specific date.
- **Description:** The appointment system allows the doctor to reserve it since it is their own schedule.
- **Inputs:** Patient ID (string), Doctor ID (string), selected date and time (string).
- **Source:** The client request source is from the API Gateway; the doctor books reserves the appointments through the doctor UI.
- **Outputs:** Confirmation of the booked appointment.
- **Destination:** The appointment is stored in the Database management system.
- **Requirements and Pre-condition:**
    - The system must ensure that this user is actually the doctor and is allowed to do this.
- **Post-condition and Side effects:**
    - The appointment is confirmed.
    - notifications are sent to the doctor.

3.3 Edit an Existing Schedule

3.3.1 The system should allow the existing schedule to be modified.

3.3.1.1 The doctor should be allowed to edit their own schedule.

- **Scenario:** A doctor needs to adjust their available hours.
- **Function:** A doctor can edit their schedule.
- **Description:** The scheduling system allows doctors to modify their availability.
- **Inputs:** Schedule ID (string), updated availability details (string), doctor ID.
- **Source:** The client request source is from the API Gateway; the doctor edits the schedule through the Doctor UI.
- **Outputs:** Confirmation of the updated schedule.
- **Destination:** The schedule is updated in the Database management system.
- **Action:** The doctor selects the schedule for a specific organization if they work in more than one and submits the updated details.
- **Requirements and Pre-condition:**
    - The system must validate the new availability against existing appointments.
- **Post-condition and Side effects:**
    - The schedule is updated, and notifications may be sent to affected patients.

3.4. Delete a Schedule

3.4.1 The system should allow a schedule to be deleted if it is no longer in use.

3.4.1.1 Doctors should be allowed to delete a schedule if they no longer work in a specific organization.

- **Scenario:** A doctor no longer wants to keep a specific schedule for a specific organization.
- **Function:** A doctor can delete a schedule.
- **Description:** The scheduling system allows doctors to remove their schedules when they are no longer needed.
- **Inputs:** Schedule ID (string), Doctor ID.
- **Source:** The client request source is from the API Gateway; the doctor deletes the schedule through the Doctor UI.
- **Outputs:** Confirmation of successful deletion.

- **Destination:** The schedule is removed from the Database management system.
- **Action:** The doctor selects the schedule to delete and confirms the action.
- **Requirements and Pre-condition:**
    - The system must check for any existing appointments before allowing deletion.
- **Post-condition and Side effects:**
    - The schedule is removed.
    - and patients with existing appointments may be notified of the change.
    - The doctor will receive a notification that the schedule got deleted.

3.4.1.2 The admin should be able to delete schedules of doctors that are no longer part of the system

- **Scenario:** A doctor no longer is a part of the digimed system.
- **Function:** administrator of the system can delete a schedule no longer in use.
- **Description:** The administrator can delete a schedule of a doctor that left the system to save the space for other data in the database.
- **Inputs:** Schedule ID (string), Doctor ID.
- **Source:** The administrator.
- **Outputs:** Confirmation of successful deletion.
- **Destination:** The schedule is removed from the Database management system.
- **Requirements and Pre-condition:**
    - The system must check for any existing appointments before allowing deletion.
    - The system must check that the person making this deletion is actually the administrator.
- **Post-condition and Side effects:**
    - A notification will be sent to the administrator and the patients of that doctor that their appointments were deleted.

3.5. Delete an Appointment

3.5.1 The system should allow an appointment to be deleted

3.5.1.1 A patient can cancel an appointment.

- **Scenario:** A patient wishes to cancel their appointment.
- **Function:** A user can delete an appointment.
- **Description:** The scheduling management system allows users to cancel their scheduled appointments.
- **Inputs:** Appointment ID (string), Patient ID, Doctor ID.
- **Source:** The client request source is from the API Gateway; the user cancels the appointment through the Patient UI.
- **Outputs:** Confirmation of appointment cancellation.
- **Destination:** The appointment is removed from the Database management system.
- **Action:** The user selects the appointment to cancel and confirms the cancellation.
- **Requirements and Pre-condition:**
    - The system must validate cancellation policies before confirming.
- **Post-condition and Side effects:**
    - The appointment is removed and both the doctor and patient may be notified.

3.5.1.2 A doctor can cancel an appointment

- **Scenario:** A doctor is no longer available to attend an appointment.
- **Function:** A doctor can cancel an appointment.
- **Description:** The scheduling management system allows doctors to cancel their appointments.
- **Inputs:** Appointment ID (string), Patient ID, Doctor ID.
- **Source:** The client request source is from the API Gateway; the doctor cancels the appointment through the Patient UI.
- **Outputs:** Confirmation of appointment cancellation.

- **Destination:** The appointment is removed from the Database management system.
- **Action:** The doctor selects the appointment to cancel and confirms the cancellation.
- **Requirements and Pre-condition:**
    - The system must validate cancellation policies before confirming.
- **Post-condition and Side effects:**
    - The appointment is removed and both the doctor and patient may be notified.

3.6. Retrieve a Schedule

3.6.1 The system should allow the schedule to be retrieved
3.6.1.1 A patient should be able to view the schedules of doctors

- **Scenario:** A patient wants to view available appointment slots for a doctor.
- **Function:** A user can retrieve a doctor's schedule.
- **Description:** The scheduling system allows users to see available time slots based on the doctor's schedule.
- **Inputs:** Doctor ID (string), Schedule ID.
- **Source:** The client request source is from the API Gateway; the user requests the schedule through the Patient UI.
- **Outputs:** The available time slots for the doctor.
- **Destination:** Data is retrieved from the Database management system.
- **Action:** The user searches for the doctor's schedule using the Doctor ID.
- **Requirements and Pre-condition:**
    - The system must allow schedule retrieval for authorized users.
- **Post-condition and Side effects:**
    - The user receives a list of available time slots to choose from.

3.7. Access Management to Scheduling

3.7.1 The system should allow the scheduling to be managed by authorized users
3.7.1.1 The admin should be able to control the access to scheduling of the users in the system

- **Scenario:** Different users require different levels of access to scheduling features.
- **Function:** The system manages user access levels to scheduling features.
- **Description:** The scheduling management system restricts or grants access to scheduling functions based on user roles.
- **Inputs:** User ID (string), access level (string).
- **Source:** The client request source is from the API Gateway.
- **Outputs:** Confirmation of access rights granted or denied.
- **Destination:** Access rights are managed through the Database management system.
- **Action:** The admin or authorized personnel adjusts user access levels.
- **Requirements and Pre-condition:**
    - The system must have defined roles and permissions.
- **Post-condition and Side effects:**
    - Users receive notifications if their access is changed.

3.3 Usability Requirements

3.4 Performance Requirements

3.5 Logical Database Requirements

3.6 Design Constraints

3.7 Standards Compliance

3.8 Software System Attributes

4 Verification

5 Supporting Information