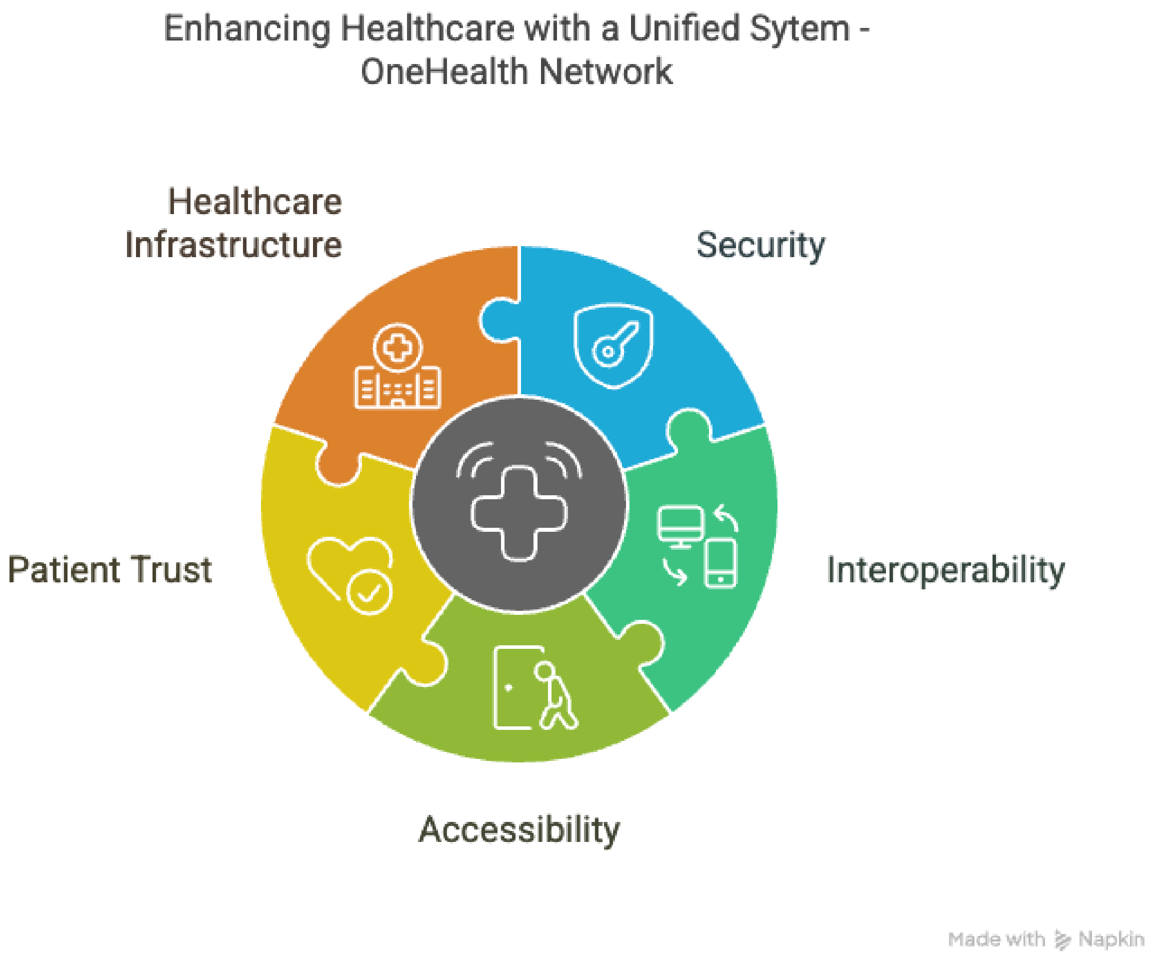
OneHealth Network

Software Design Document

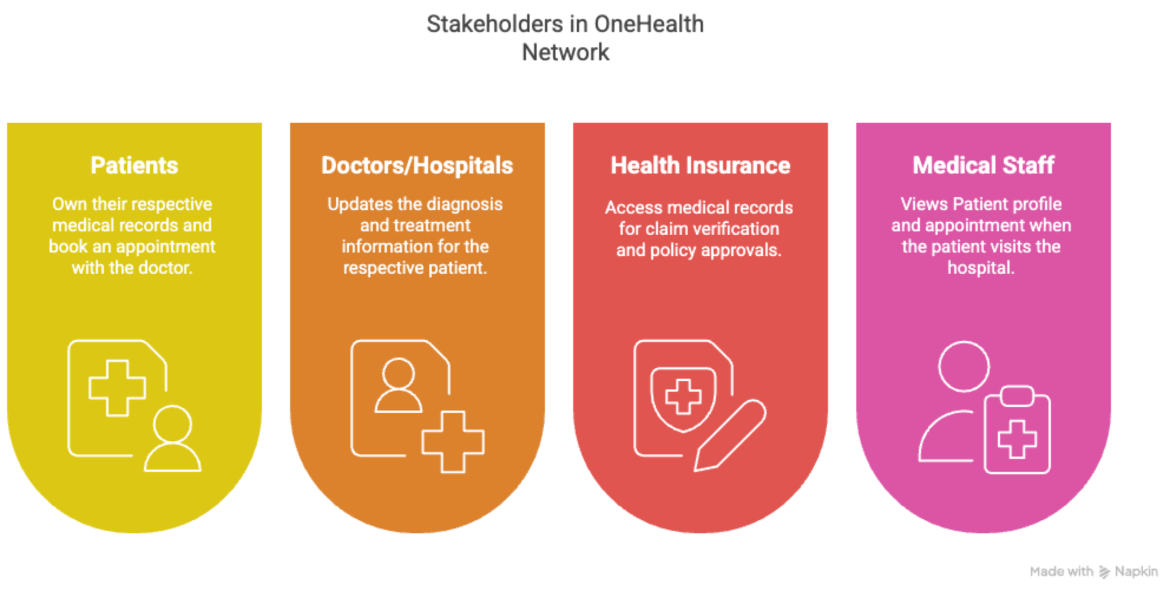
1. Project Overview

1.1 Project Scope

The absence of a centralized, safe, and user-friendly platform for maintaining medical records across the country presents serious difficulties for the healthcare industry. Patients frequently battle with disjointed medical records from various clinics and hospitals, which can result in ineffectiveness of treatment, diagnosis, and care continuity. Citizens can safely keep, update, and share their medical records with approved healthcare providers thanks to our integrated web platform, which functions as a unified, national healthcare system. In the end, this system strengthens the country's healthcare infrastructure by ensuring smooth interoperability, enhancing patient trust, improving medical decision-making, and promoting universal access to healthcare information.


Enhancing Healthcare with a Unified Sytem - OneHealth Network
Healthcare Infrastructure
Security
Patient Trust
Interoperability
Accessibility
Made with Napkin

1.2 Stakeholders and Stake


Stakeholders in OneHealth Network
Patients
Own their respective medical records and book an appointment with the doctor.
Doctors/Hospitals
Updates the diagnosis and treatment information for the respective patient.
Health Insurance
Access medical records for claim verification and policy approvals.
Medical Staff
Views Patient profile and appointment when the patient visits the hospital.
Made with Napkin

Stakeholders	**Stake**
🧍 Patients	Own their respective medical records and book an appointment with the doctor
🩺 Doctors/Hospitals	Updates the diagnosis and treatment information for the respective patient
🛡️ Health Insurance agencies	Access medical records for claim verification and policy approvals.
🧍 Medical Staff	Views Patient profile and appointment when the patient visits the hospital

1.3 User Stories

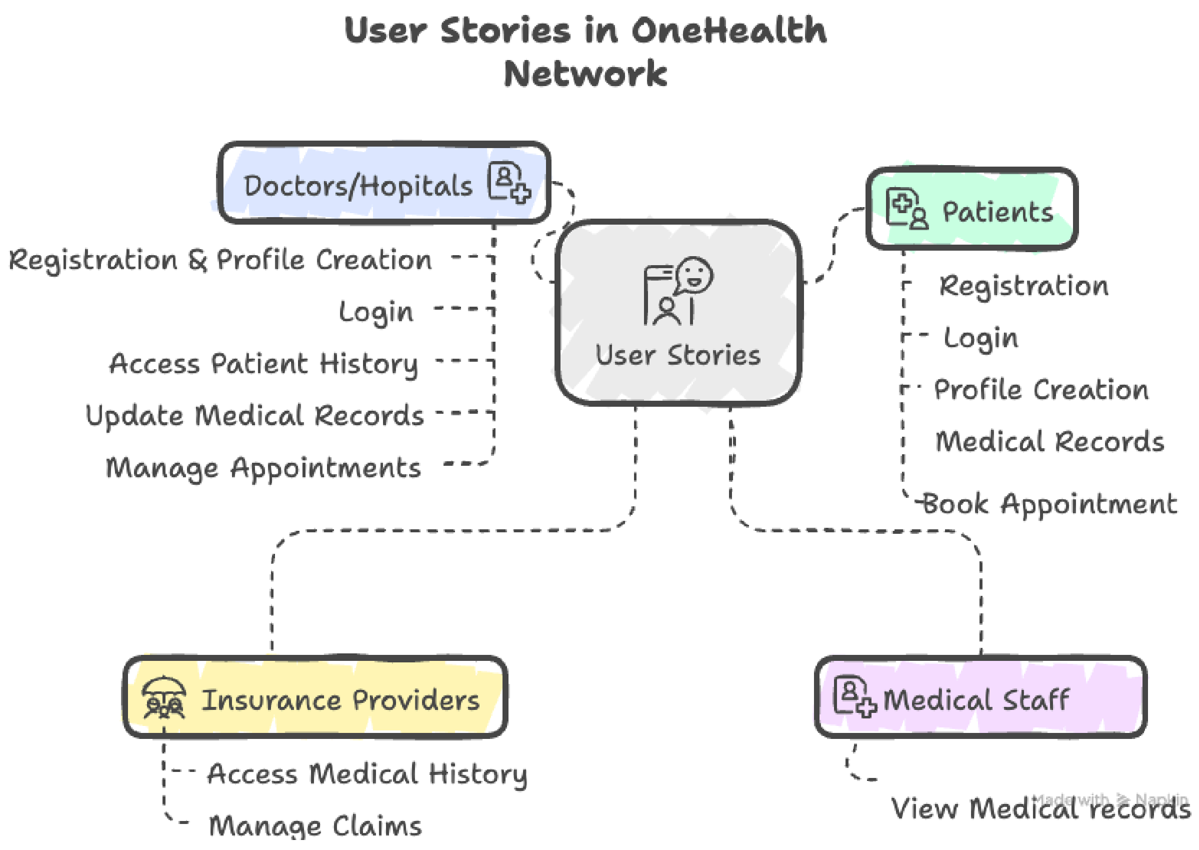


Figure - Overview of major stories of each stakeholder

Following are all the user-stories of each stakeholders-

1.3.1 Patient

Theme	**User Story**	**Tasks Involved**
Registration	As a Patient, he/ she want to register to the	1) Create a Patient model in models.py with fields (name, email, password, phone, etc.). 2) Design a user registration form (forms.py). 3) Create a signup view in views.py to handle registration. 4) Set up a URL route for the signup page in urls.py. 5) Create a signup.html template for the

	system, and create a profile (SignUp)	registration form. 6) Implement password hashing for security. 7) Add validation for email uniqueness and password strength. 8) Display success/error messages after registration.
Login	As a Patient, he/ she want to access the system with my profile (SignIn)	1) Create a login view in views.py to authenticate users. 2) Design a login form (forms.py). 3) Set up a URL route for the login page in urls.py. 4) Create a login.html template for the login form. 5) Implement session management after successful login. 6) Add error handling for invalid credentials. 6) Redirect authenticated users to the dashboard. 7) Implement "Remember Me" functionality (optional).
Profile Creation	As a Patient, he/ she want to create a health profile	1) Create a HealthProfile model in models.py (allergies, blood group, weight, height, etc.). 2) Link HealthProfile to the Patient model via ForeignKey. 3) Design a health profile form (forms.py). 4) Create a create_health_profile view in views.py. 5) Set up a URL route for health profile creation in urls.py. 6) Create a health_profile.html template. 7) Ensure only authenticated patients can access this feature. 8) Display success/error messages after submission.
	As a Patient, he/ she want to add, delete, edit my	1) Create an edit_health_profile view in views.py. 2) Design a form to update their health profile (forms.py). 3) Set up URL routes for edit/delete actions in urls.py.

Perform action	health profile/medical history	4) Create an edit_health_profile.html template. 5) Implement a delete_health_record view for deletions. 6) Add confirmation prompts before deletion. 7) Restrict access to the patient who owns the profile. 8) Log changes for audit purposes (optional).
Claim Insurance	As a Patient, he/ she want to file Insurance claim through the system	1) Create an InsuranceClaim model (patient, claim_amount, status, documents, etc.). 2) Design an insurance claim form (forms.py). 3) Create a file_claim view in views.py. 4) Set up a URL route for claims in urls.py. 5) Create a file_claim.html template. 6) Allow file uploads for supporting documents. 7) Validate claim data before submission. 8) Notify the patient via email upon claim submission. 9) Implement status tracking (Pending/Approved/Rejected).

1.3.2 Doctors

Theme	**User Story**	**Tasks Involved**
Registration & Profile Creation	As a Medical Staff, he/ she want to register to the system, and create a profile (SignUp)	1) Create a registration form for medical staff (name, email, password, role, specialization, etc.). 2) Add backend logic to validate and save staff details. 3) Assign a "Medical Staff" or "Doctor" role during registration. 4) Send a confirmation email (optional). 5) Redirect to the login page after successful registration.
Login	As a Medical Staff, he/ she want to access	1) Create a login form (email/username + password). 2) Authenticate and verify staff credentials. 3) Redirect to a staff-specific dashboard upon

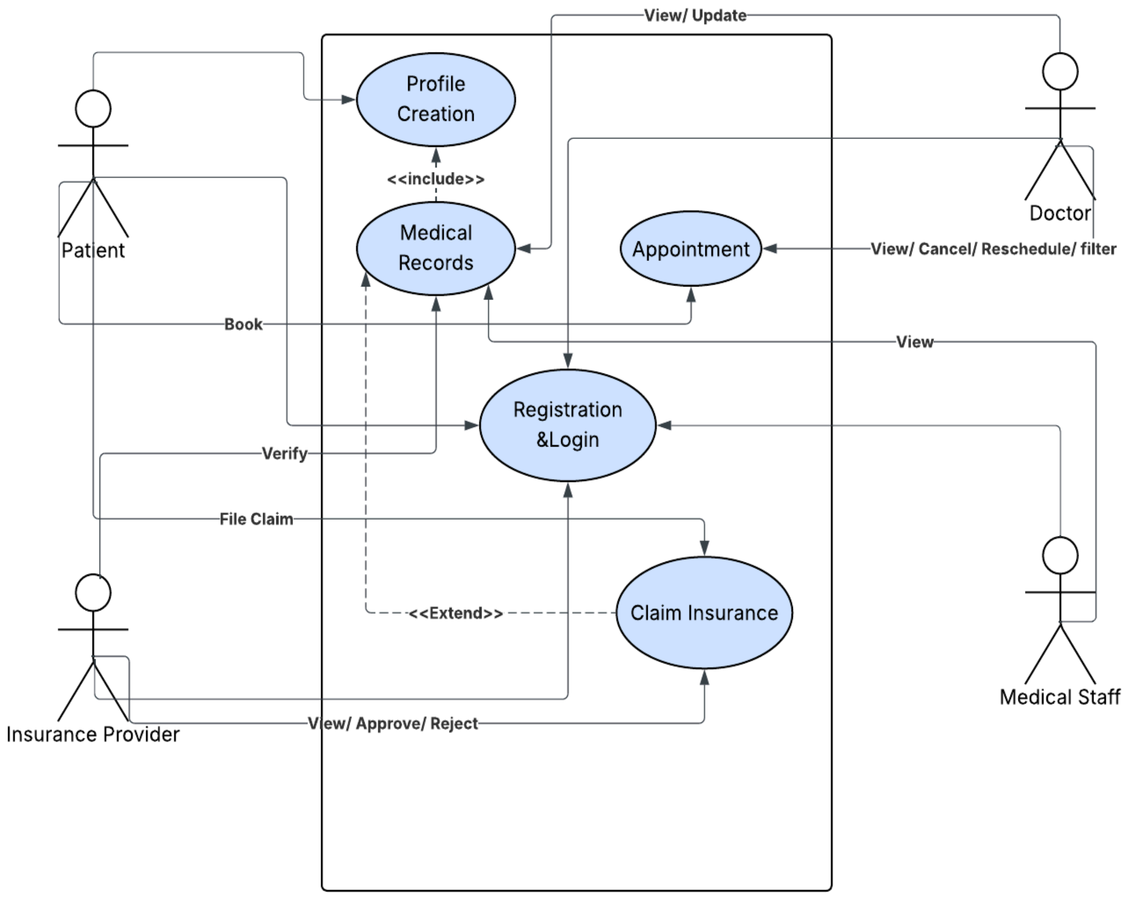
	the system with my profile (SignIn)	successful login. 4) Handle failed login attempts (error messages).
Perform Action	As a Doctor, he/ she want to see Patient's Medical History	1) Implement a search/filter to find patients (by name, ID, etc.). 2) Fetch and display the selected patient’s medical history (allergies, past treatments, etc.). 3) Ensure only authorized doctors can access patient records. 4) Display records in a readable format (table/cards).
Perform Action	As a Doctor, he/ she want to update my Patient's Medical history	1) Allow doctors to edit/add new medical records (diagnosis, prescriptions, notes). 2) Save changes securely with timestamps. 3) Show success/error messages after updates. 4) Restrict editing to only assigned patients (if applicable).
Perform Action	As a Doctor, he/ she should check my patients appointment	1) Fetch and display upcoming/past appointments for the logged-in doctor. 2) Filter appointments by date, status (confirmed/cancelled), or patient. 3) Allow rescheduling/cancelling appointments (if needed). 4) Show patient details linked to each appointment.

1.3.3 Health Insurance Agencies

Theme	**User Story**	**Tasks Involved**
Registration & Profile Creation	As a Insurance Provider, they want to register to the system, and create a profile (SignUp)	1) Create a registration form for medical staff (name, email, password, role, specialization, etc.). 2) Add backend logic to validate and save staff details. 3) Assign a "Medical Staff" or "Doctor" role during registration. 4) Send a confirmation email (optional). 5) Redirect to the login page after successful registration.
Login	As a Insurance	1) Create a login form (email/username + password).

	Provider, they want to access the system with their profile (SignIn)	2) Authenticate and verify staff credentials. 3) Redirect to a staff-specific dashboard upon successful login. 4) Handle failed login attempts (error messages).
Perform Action	As a Insurance Provider, they want to see Patient's Medical History	1) Implement a search/filter to find patients (by name, ID, etc.). 2) Fetch and display the selected patient’s medical history (allergies, past treatments, etc.). 3) Ensure only authorized doctors can access patient records. 4) Display records in a readable format (table/cards).
Perform Action	As a Insurance Provider, they want to access Patient’s medical claims	1) Allow doctors to edit/add new medical records (diagnosis, prescriptions, notes). 2) Save changes securely with timestamps. 3) Show success/error messages after updates. 4) Restrict editing to only assigned patients (if applicable).

1.4 Use Case Diagram

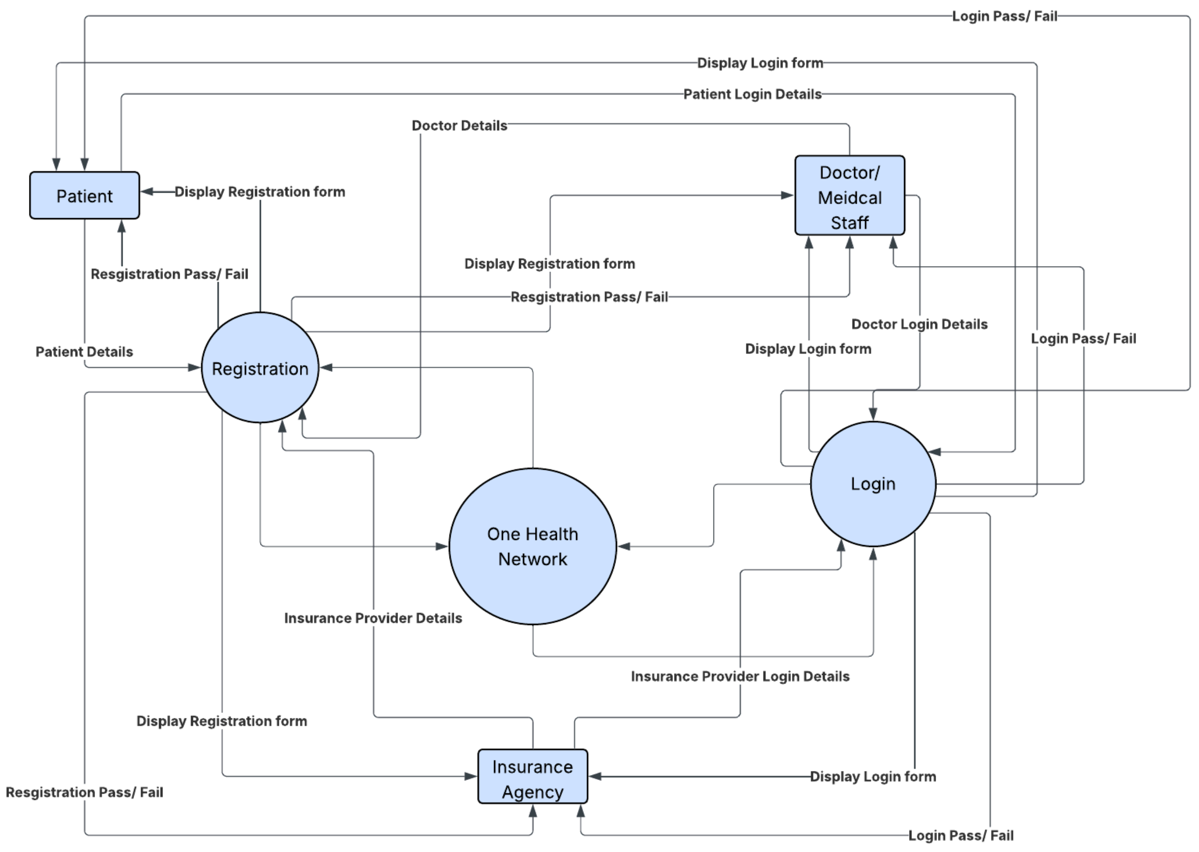


- Registration and login is the use case which is common for all the actors/ users.
- Patients can create their own health profile which includes the details of medical history/ records.
- Patient books an appointment with the doctor and the doctor owns the right to accept or cancel or reschedule the appointment.
- To reimburse the amount spent to the hospital, the patient files a health insurance claim, where the medical records of the patient can also be referred.

1.5 Context Diagram

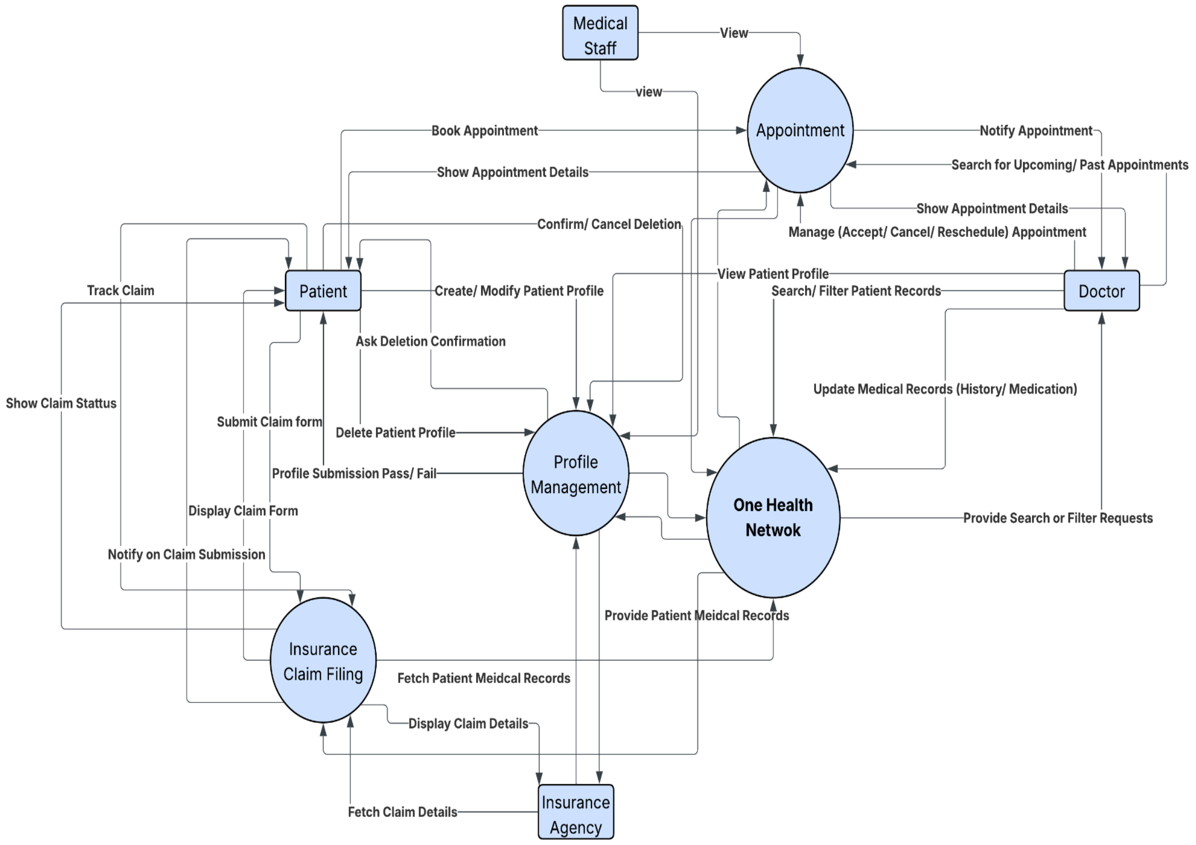
Note: Context diagram is split into two (Registration & Login and the rest), as putting everything together is making the diagram look clumsy.

1.5.1 Registration & Login



- All the four users (Patient, Doctor, Medical staff and Insurance agency) follow the same flow for both registration and login.
- If the user wants to register as a new user, a registration form is displayed and on filling up the fields, it gets submitted and a response will be shown if the registration is successful or failed.
- If the user wants to login, a login form is displayed and on filling up the fields, it gets submitted and a response will be shown if the login is successful or failed.

1.5.1 Others



- Profile management, Insurance claim filing and Appointment belong to the system "One Health Network".
- A patient can create, edit and delete his/ her health profile. During these operations on profile, the system asks for confirmation and sends the response once the action is completed.
- Medical staff, doctors and Insurance agencies can view the patient's health profile.
- While the patient wants to file an insurance claim, a claim form is displayed to the patient and once submitted the response will be shown.
- After successful claim submission, the patient can track the claim status.
- The medical staff can view the appointment of the patient.
- Doctors can filter or search patient records and manage the appointment by accepting or cancelling or rescheduling it.

1.6 Product Backlog

OneHealth Sprint Backlog

2. Architectural Overview

2.1 Architectural Decisions

The architecture of the **OneHealth Network** project was carefully designed to balance flexibility, security, scalability, and maintainability. Several alternative architectural designs were considered during the design process, but we have chosen Microservice Architecture. Let's discuss more on why this architecture is chosen.

2.1.2 Alternative Architecture

Monolithic Architecture

Description : In this approach, all components of the system would be tightly coupled and packaged together as one single unit. This would make the system easier to implement initially, as there is no need for communication between different layers.

Rationale for Rejection : This design would limit scalability and flexibility. As the application grows, it would be harder to manage, debug, and scale the system efficiently. Moreover, coupling components together would make it harder to update or replace individual features without affecting the entire system.

Service Oriented Architecture

Description : In this approach, it promotes modularity through reusable services. This architecture has better modularization than monoliths, It has established patterns for enterprise integration. This architecture supports multiple transport protocols and standards.

Rationale for Rejection : This architecture tends to involve heavier communication protocols like SOAP which is very complex, this is difficult to adopt lightweight RESTful or event driven patterns natively. This architecture has more governance overhead compared to microservice.

2.1.3 Chosen Architecture

After evaluation of the above architecture we have chosen **Microservice architecture** for multiple reasons like Modularity this architecture provides better modularity between each business capabilities. They can isolate different modules as its own service so they can work independently. The services can be scaled as needed without affecting other services. Team members can choose any appropriate tools according to their service. Microservice integrates well with containerization like docker and Devops tools.

2.1 Subsystem Architecture

This section discusses the Subsystem Architecture, represented by the UML package diagram that illustrates the dependencies between the packages of the system. The system is structured into four main layers: Application-Specific, Application-Generic, Middleware, and System-Software.

2.1.1. Subsystem Layers and Responsibilities

Application-Specific Layer

Responsibility: This layer contains the core components that are directly involved in the healthcare-specific functionalities of the system.

PatientRecordManagment: Manages the storage, retrieval, and processing of patient medical records.

DoctorRecordManagment: Manages the storage, retrieval, and processing of doctor records.

Insurance Management : Handles insurance claims, patient consent and compliance policies and access to patient data.

Architectural Style: This layer follows a Layered Architecture style, focusing on clear separations between domain-specific logic and infrastructure services. This modular structure makes it easier to maintain and scale specific functionalities independently.

Application-Generic Layer

Responsibility: This layer handles cross-cutting concerns that apply to the whole application, such as user authentication and notifications.

Authentication: Responsible for validating user credentials and managing session states.

NotificationService: Manages communication services such as email, push notifications, and alerts to users.

Architectural Style: A **Service-Oriented Architecture (SOA)** approach is applied here. Each service in this layer is loosely coupled, allowing independent scaling and replacement, while maintaining centralized control of essential functions like authentication.

Middleware Layer

Responsibility: The middleware layer enables the communication and coordination between the application components and the system software. It ensures that data flows securely and correctly across components.

APIGateway: Routes incoming requests to appropriate services and handles load balancing and request validation. We have different API's to fetch data of patients, doctor and insurance
Below are APIs we have used to send and fetch data

EMR Module APIs
Login Module
URL: /
View: index
Method: GET
Description: Renders the login page.

URL: /chgpwdpage/
View: changePwdPage
Method: GET
Description: Displays the change password page.

URL: /newuser/
View: newUser
Method: GET
Description: Displays the new user registration page.

URL: /login/
View: login
Method: POST
Description: Verifies user credentials and logs in.

URL: /createuser/

View: createUser
Method: POST
Description: Creates a new user account.

URL: /changepwd/
View: changePwd
Method: POST
Description: Changes the user's password.

URL: /logoff/
View: logoff
Method: GET
Description: Logs the user out.

Patient Contact Module
URL: /patientcontact/
View: index
Method: GET
Description: Renders the patient contact index page.

URL: /patientcontact/add/
View: add_contact
Methods: GET, POST
Description: Displays a form to add a new contact (GET) and saves the contact (POST).

URL: /patientcontact/edit/<id>/
View: edit_contact
Methods: GET, POST
Description: Displays a form to edit an existing contact (GET) and updates the contact (POST).

URL: /patientcontact/delete/<id>/
View: delete_contact
Methods: GET, POST
Description: Displays a confirmation page to delete a contact (GET) and deletes the contact (POST).

URL: /patientcontact/read/
View: readContactInfo

Method: GET
Description: Fetches and displays all contacts for the logged-in user.

URL: /patientcontact/update/
View: updateContactInfo
Method: POST
Description: Updates contact information for the logged-in user.

Patient Insurance Module
URL: /patientinsurance/
View: index
Method: GET
Description: Renders the patient insurance index page.

URL: /patientinsurance/add/
View: add_insurance
Method: GET
Description: Displays a form to add insurance information.

URL: /patientinsurance/edit/<id>/
View: edit_insurance
Method: GET
Description: Displays a form to edit insurance information.

URL: /patientinsurance/delete/<id>/
View: delete_insurance
Method: GET
Description: Displays a confirmation page to delete insurance information.

URL: /patientinsurance/save/
View: saveInsurInfo
Method: POST
Description: Saves insurance information to the database.

Patient Medical History Module
URL: /patientmedicalhistory/
View: index
Method: GET
Description: Renders the patient medical history index page.

URL: /patientmedicalhistory/add/
View: add_medical_history
Method: GET
Description: Displays a form to add medical history.

URL: /patientmedicalhistory/edit/<id>/
View: edit_medical_history
Method: GET
Description: Displays a form to edit medical history.

URL: /patientmedicalhistory/delete/<id>/
View: delete_medical_history
Method: GET
Description: Displays a confirmation page to delete medical history.

URL: /patientmedicalhistory/save/
View: saveMedHistoryInfo
Method: POST
Description: Saves medical history information to the database.

Patient Medications Module
URL: /patientmeds/
View: index
Method: GET
Description: Renders the patient medications index page.

URL: /patientmeds/add/
View: add_medication
Method: GET
Description: Displays a form to add medication.

URL: /patientmeds/edit/<id>/
View: edit_medication
Method: GET
Description: Displays a form to edit medication.

URL: /patientmeds/delete/<id>/
View: delete_medication
Method: GET
Description: Displays a confirmation page to delete medication.

URL: /patientmeds/api/
View: Meds
Method: GET
Description: API endpoint for patient medications.

Dashboard Module
URL: /dashboard/
View: index
Method: GET
Description: Renders the dashboard page.

Hospitals/Doctors Module APIs
AuthApp Module
URL: /register/
View: register
Methods: GET, POST
Description: Displays a registration form (GET) and registers a new user (POST).

URL: /edit/
View: edit
Methods: GET, POST
Description: Displays a form to edit user details (GET) and updates the details (POST).

URL: /dashboard/
View: dashboard
Method: GET
Description: Renders the user dashboard.

URL: /
View: LoginView
Method: GET
Description: Displays the login page.

URL: /logout/
View: LogoutView
Method: GET
Description: Logs the user out.

URL: /password_change/
View: PasswordChangeView
Method: GET
Description: Displays the password change form.

URL: /password_reset/
View: PasswordResetView
Method: GET
Description: Displays the password reset form.

DataSecurity: Ensures compliance with privacy and security regulations, such as data encryption and access control.

IntegrationService: Facilitates communication between the OneHealth Network system and external systems or third-party services.

Architectural Style: This layer employs a Client-Server Architecture style, where the server-side components (APIGateway, DataSecurity) handle requests and ensure that the client-side (applications) can interact with the system securely.

System-Software Layer

Responsibility: The system software layer provides the necessary infrastructure and services to support the entire application.

Database: Stores and manages data, including patient personal data, Doctor personal data, Insurance providers data, Appointment booking data.

SystemServices: Provides lower-level services such as data storage management and network connectivity.

Architectural Style: This layer is based on **Repository Architecture**, where the database acts as a central repository of information that multiple components interact with to retrieve and store data.

2.1.2 Dependency Relationships and Justification

- Application-Specific Layer depends on Authentication for user validation and other shared services like notification handling.
- Middleware Layer serves as the communication bridge, handling data flow between the system's core business logic and its infrastructure. For instance, the APIGateway communicates with both the Application-Specific Layer and the Database.
- System-Software Layer provides the fundamental infrastructure, with components like the Database serving as a central data storage solution that multiple services interact with.

2.1.3 Architectural Styles Applied

Microservice Architecture (Primary): This architecture enables modularity, scalability and independent development and resilience.

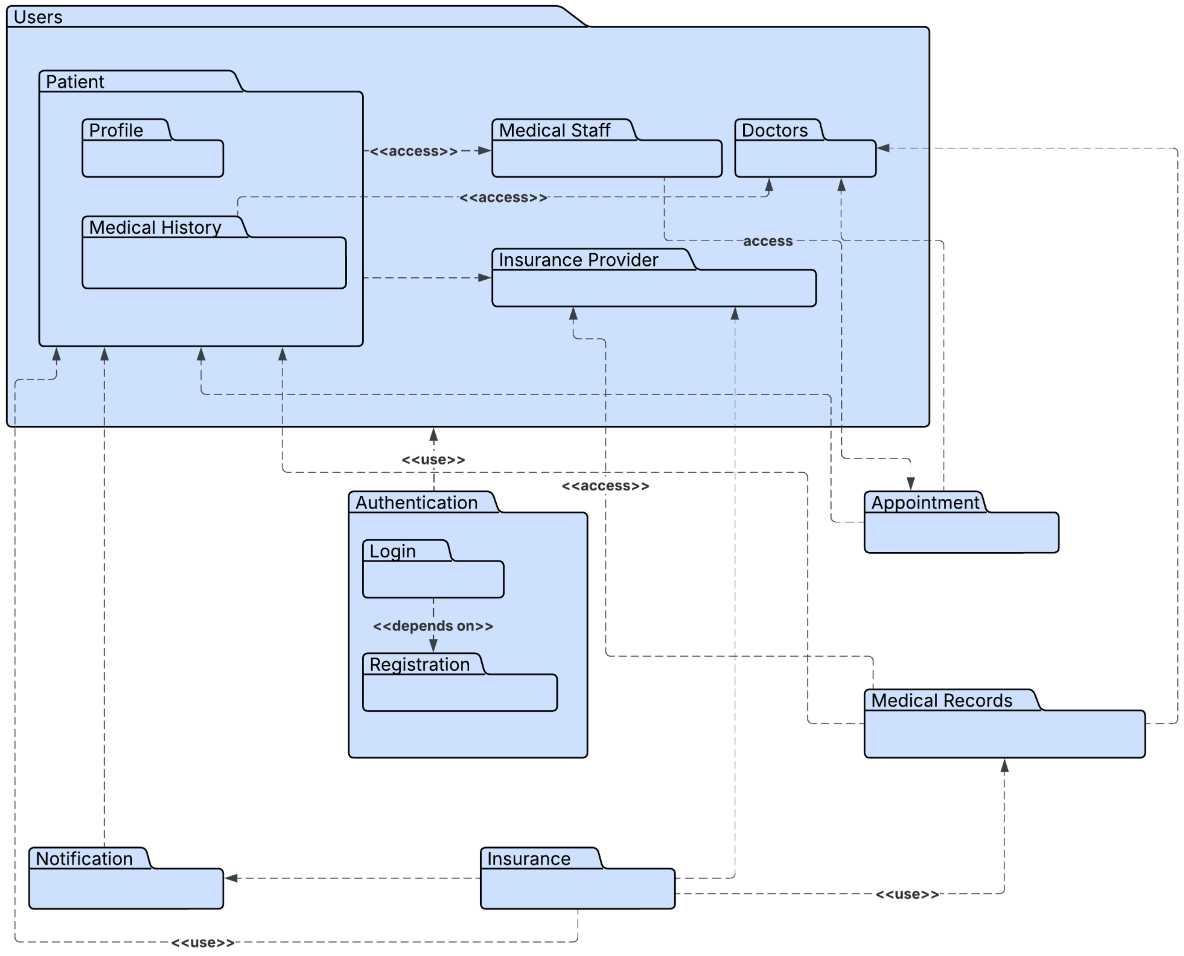
L**ayered Architecture**: The clear separation into different layers allows for modularity, where each layer performs a specific function, reducing system complexity and promoting maintainability.

Service-Oriented Architecture (SOA): Applied in the **Application-Generic Layer**, it supports reusable, independent services like authentication and notifications, which can be scaled or replaced without impacting other parts of the system.

Repository Architecture: Used in the **System-Software Layer** where a central database stores all application data. This architecture simplifies data management and ensures consistency across components.

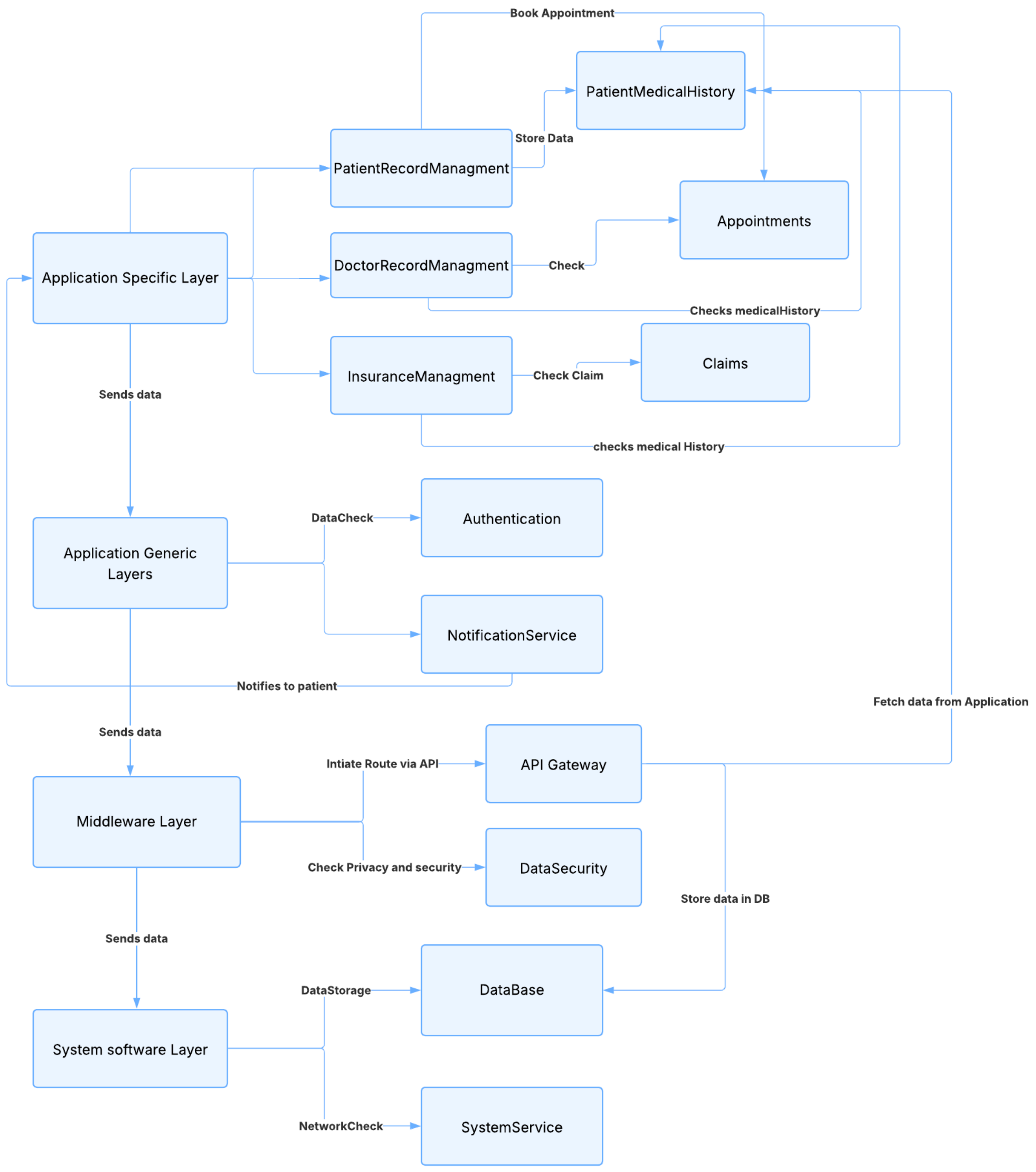
These are chosen for multiple reasons for example **Scalability** : Patient appointments and Insurance claims can come in high volumes and they can be scaled independently. **Maintenance** : each service and be changed and redeployed with affecting other services. **Extension** : any new features or any personalization can be added without disturbing any main components.

2.1.4 UML package diagram:



- All the users are packaged as sub - packages in users package.
- Patient's profile and medical history are sub - packages of the patient.
- Registration and login are sub - packages of authentication where it uses the user information.
- Insurance providers access medical records for verification.
- Patients book appointments and doctors can alter or view the appointments.
- Medical staff can also view the appointment.
- Doctors and patients use medical records.
- Insurance uses patient information and medical records to process the claim and sends responses as notification to patients.

2.1.5 Dependency Diagram



2.2 Deployment Architecture

For this we have a single threaded software system that runs on a local device and does not connect to any other systems running on different computers across a network.

“This software will run on a single processor.”

2.3 Persistent Data Storage

For our system we have chosen MongoDB, a NoSQL document based database for persistent data storage, we have chosen this for its flexibility, scalability and its ability to store complex documents. So this is ideal for our healthcare so it can deal with diverse data types.

We have created 4 collection(tables) in this db to store all the info from the application

Patient collection : which stores data regarding patients all their personal info is stored in this collection, Below is a display of columns in table format for easy understanding.

Field Name	Type	Description
_id	ObjectId	Unique patient identifier
first_name	String	First name
last_name	String	Last name
date_of_birth	Date	Date of birth
gender	String	Gender
email	String	Email address
phone	String	Contact number
address	String	Residential address
insurance	Object	insurance_id

Doctor collection: which stores data regarding doctors and their personal info, below is the display of columns in table format for this collection to understand easily.

Field Name	Type	Description
_id	ObjectId	Unique doctor identifier
first_name	String	First name
last_name	String	Last name
specialty	String	Medical specialty
email	String	Email address
phone	String	Contact number
license_number	String	Medical license ID
years_experience	Number	Number of years in practice

Insurance Collection : which stores data related to insurance companies, their policies and providers, below is the display of columns in table format for this collection to understand easily.

Field Name	Type	Description
_id	ObjectId	Unique insurance ID
provider_name	String	Name of insurance provider
plan_name	String	Specific insurance plan name
coverage_type	String	Type of coverage (Basic, Premium, etc.)
contact_number	String	Insurance company contact number

Appointments Collection: This store appointment details which patients are made through system like date, purpose to book, status ect, below is the display of columns in table format for this collection to understand easily.

Field Name	Type	Description
_id	ObjectId	Unique appointment ID
patient_id	ObjectId	Reference to patients._id
doctor_id	ObjectId	Reference to doctors._id
date	Date	Appointment date and time
reason	String	Purpose of visit
status	String	Scheduled, Completed, or Cancelled

2.4 Global Control Flow

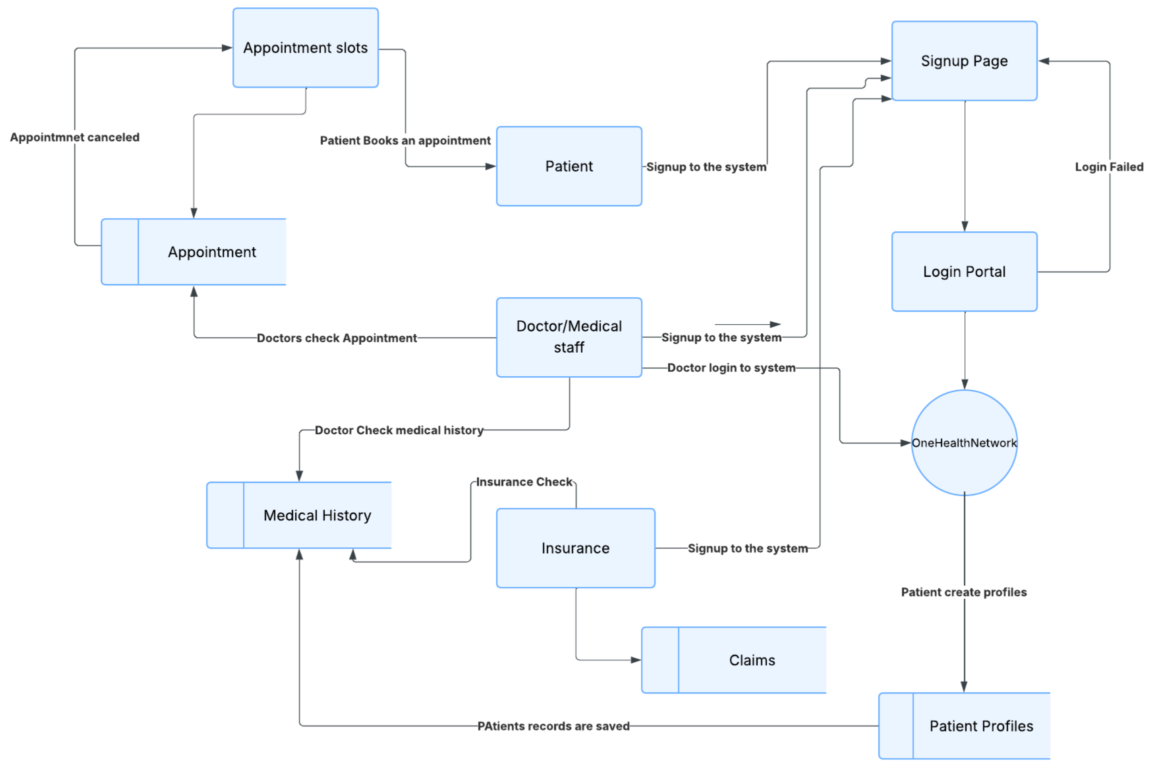
Our system adopts mostly event driven architecture, sometimes time dependency and limited concurrency in certain operations.

Procedural or event-driven: This system is predominantly event driven. It operates by listening and responding to the user generated events like login request, patient updating records, booking appointment and form submission. Each user can perform actions in a non linear fashion based on their role and the options which are available to them. A patient can view medical history, booked appointments or update any personal information, A doctor may update any medical records or notes of patients, they can view the booked appointment and check patient history. Admins can perform any backend updates, manage insurance records based on the events. This design gives flexibility and supports multiple workflows based on the role.

Time dependency : Time dependency is present only in the booking an appointment feature. While most of the applications are event driven just booking appointments involve time based constraints. The system checks for available time slots dynamically before booking any appointment. Timers and alerts may be used to notify patients for their next medical appointment. It maintains time-awareness for validating and scheduling appointments.

Concurrency : The system supports basic concurrency through async request handling Appointment booking Multiple users can attempt to book different appointments parallely.
Data access: Separate backend services handels patient requests, doctors request and insurance requests concurrently. While multithreading is not used explicitly at application level there are underlying web servers that can handle concurrent requests.

2.4.1 Data Flow Diagram

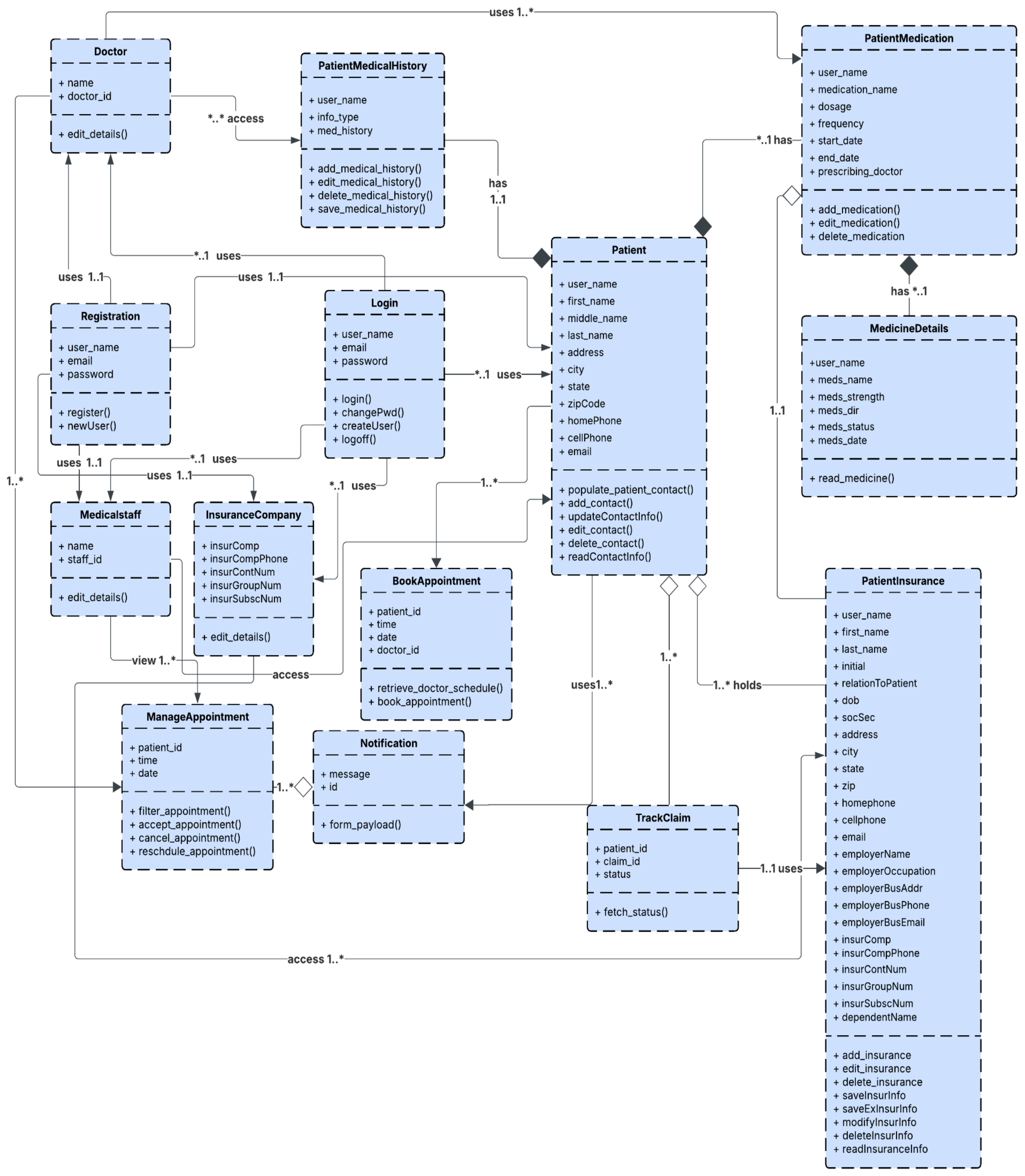


- Patients, Doctors and Insurance are the main actors for this system.
- Patients, doctors and Insurance providers all first need to signup page to create login detain and then login to OneHealth system.
- When login failed it reverted back to the signup page to see the errors.
- Once a patient logins to the system he creates a profile by entering all personal details.
- Patient checks for appointments with doctors and books the next availability slot.
- Patients claim the insurance through appropriate insurance providers.
- When doctors cancel the appointment it reverts back to the appointment page for patients.
- Doctors login into the system and check for appointments booked by patients.
- Doctors check the patient's medical records to know their history.
- Insurance providers login into the system to check claims made by patients.
- Insurance providers check medical records of the patient before claims.

3. Detailed System Design

3.1 Static View

3.1.1 UML Class Diagram



3.1.2 Class - Explanation

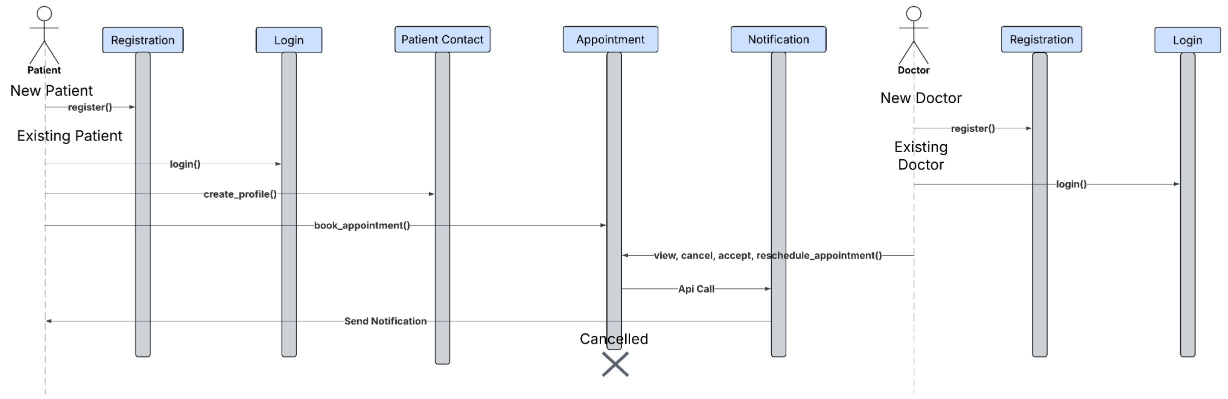
Class	Relation Class	Relation Type	Explanation
Doctor	Registration	Association - 1:1	Doctor is a user class, where one doctor can do only one registration
Patient	Registration	Association - 1:1	Patient is a user class, where one patient can do only one registration
Medical Staff	Registration	Association - 1:1	Medical staff is a user class, where one staff can do only one registration
Insurance Company	Registration	Association - 1:1	Insurance agency is a user class, where one company can do only one registration
Doctor	Login	Association - 1:*	Doctor is a user class, where one doctor can login multiple times
Patient	Login	Association - 1:*	Patient is a user class, where one patient can login multiple times
Medical Staff	Login	Association - 1:*	Medical staff is a user class, where one staff can login multiple times
Insurance Company	Login	Association - 1:*	Insurance Company is a user class, where one company can login multiple times
Doctor	Patient Medical History	Association - *..*	Any number of doctors can view multiple patient's medical history
Patient	Patient Medical History	Composition - 1..1	A patient can have only one medical history and history belong to patient
Patient	Patient Medication	Composition - 1..*	A patient can have multiple medications and medications belong to patient
Doctor	Patient Medication	Association - 1..*	A doctor can provide multiple medications
Patient Medication	Medicine Details	Composition - 1..*	One medication given for a patient can have multiple drugs and those will be there in medicine details and strongly coupled

Patient Insurance	Patient Medication	Aggregation - 1..1	Patient Insurance can possibly check the medication and for one claim can check that particular treatment details and even if the patient is deleted claim will remain
Patient	Patient Insurance	Aggregation - 1..*	A patient can claim 0 to multiple insurances
Patient	Track Claim	Aggregation - 1..*	A patient can track claims multiple times. Since, claim remains even if the patient is deleted, track claim also remains undeleted.
Patient Insurance	Track Claim	Association - 1..1	A patient insurance claim can have only one track claim
Insurance Company	Patient insurance	Association - 1..*	Insurance company can access any number of insurance claims of patients
Patient	Book Appointment	Association - 1..*	A patient can book multiple appointments
Doctor	Manage Appointment	Association - 1..*	A doctor can manage any number of appointments
Medical Staff	Manage Appointment	Association - 1..* (View)	A medical staff can view any number of appointments, which is a restricted relationship to the class
Notification	Manage Appointment	Aggregation - *..1	Multiple notifications can be sent for any alteration in appointment. And notification is loosely coupled with manage appointment
Patient	Notification	Association - 1..*	A patient can receive multiple notifications

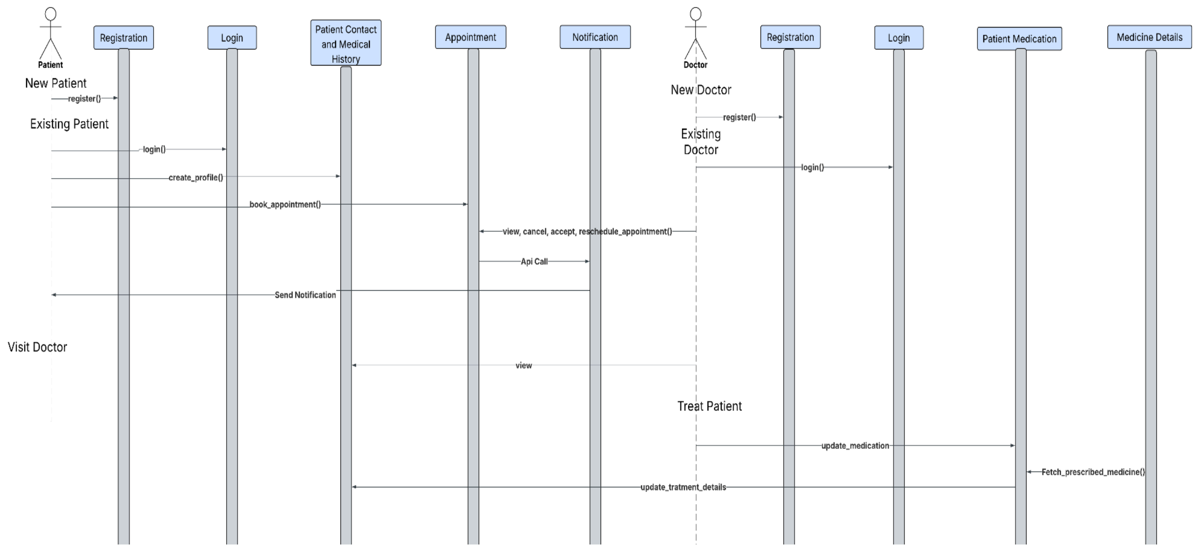
3.2 Dynamic view

3.2.1 Sequence Diagram

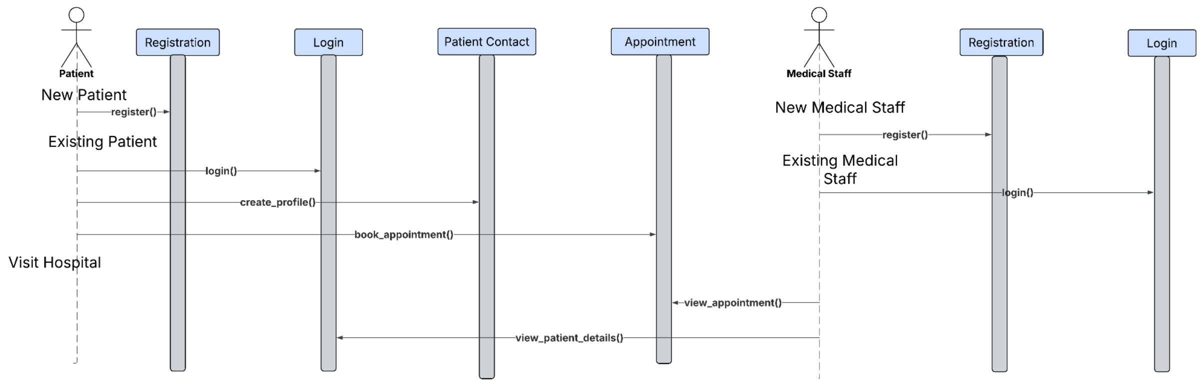
3.2.1.1 Appointment Cancellation Flow



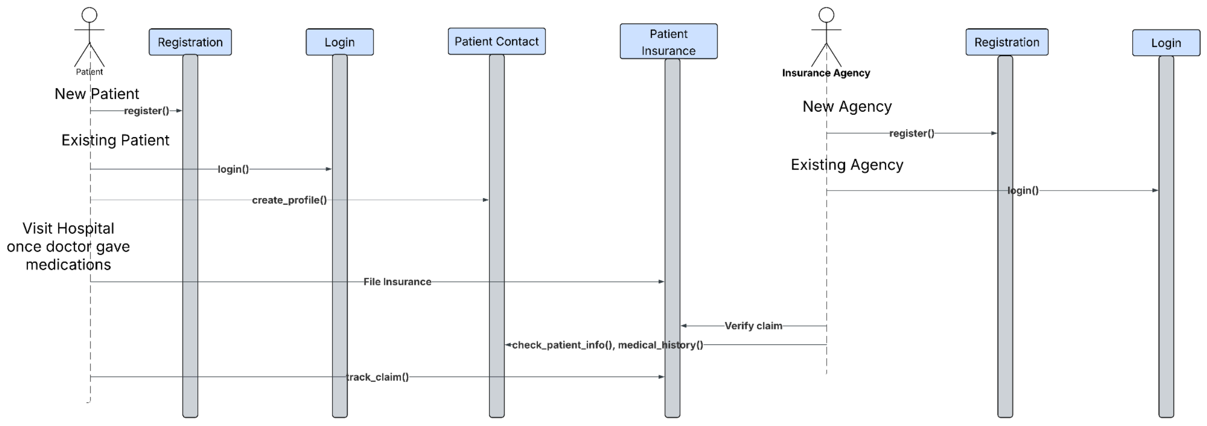
3.2.1.2 Successful or Rescheduled Appointment



3.2.1.3 Medical Staff



3.2.1.4 Filing Insurance Claim



3.3 Test Plan

Follow the link - One Health Network Test Plan