

Project Overview

The application is a web-based platform that allows healthcare providers to view and analyze medical images such as X-rays, CT scans, and MRIs. The platform provides tools to view images, annotate them, and perform image analysis using machine learning models. The platform is designed as a companion tool for radiologists and other healthcare providers to help them diagnose diseases. It uses the DICOM standard to retrieve and display medical images from healthcare providers' Picture Archiving and Communication System (PACS) servers.

Project Architecture

The project is divided into the following directories:

- **backend**: Contains the backend code. Used FastAPI as the web framework and SQLAlchemy as the ORM. Jinja2 is used as the templating engine for email templates.
- **frontend**: Contains the frontend code. Used Next.js 14 as the frontend framework and Material-UI as the UI library. Auth.js is used for authentication.
- **ml**: Contains the ML models and integration API.
- **viewer**: Contains the OHIF viewer. There is also a proxy server to handle the DICOM requests.
- **k8s**: Contains the Kubernetes deployment files.

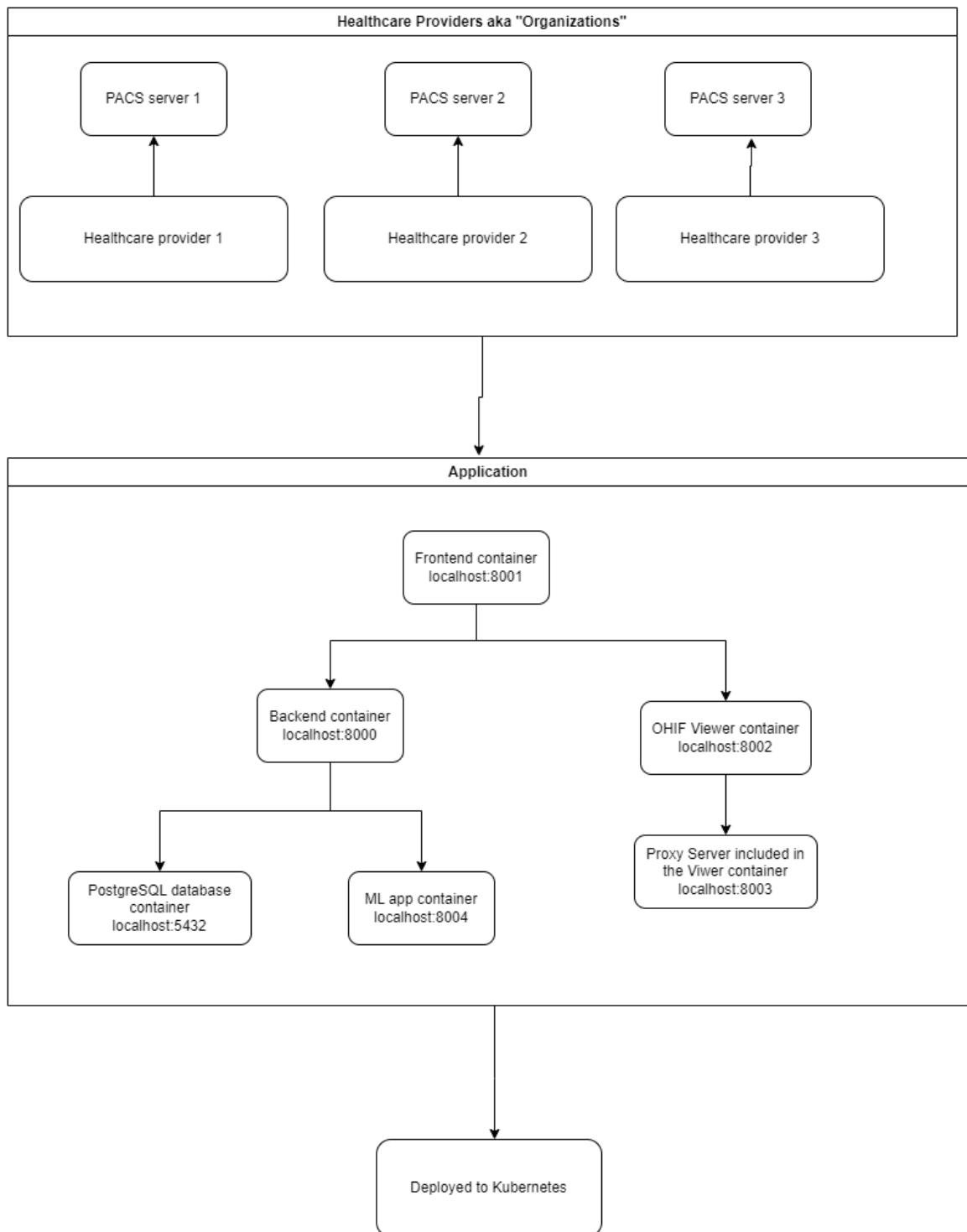
All the directories contain a **README.md** file that explains how to run the code in a local development environment. All applications are containerized using Docker and can be deployed to Kubernetes. The following ports are used by the applications:

- Backend: 8000
- Frontend: 8001
- Viewer: 8002
- Proxy: 8003
- ML: 8004

You can also simply run the applications using the `docker-compose` command instead of running them individually. The `docker-compose.yml` file is located in the root directory. To run the applications, use the following command:

```
docker compose up
```

Overall Architecture Diagram



Backend

The backend is a FastAPI application that serves the frontend and provides an REST API. Used Domain Driven Design (DDD) architecture to structure the codebase. The **src** directory contains the following directories:

- **application:** Contains the application business logic. This is where the use cases are implemented using the Command Query Responsibility Segregation (CQRS) pattern.
 - **commands:** Contains the command definitions and handlers. Commands are used to perform write operations such as creating, updating, and deleting entities.
 - **queries:** Contains the query definitions and handlers. Queries are used to perform read operations such as fetching entities from the database.
 - **services:** Contains the service classes that contain the business logic that is shared across multiple use cases.
 - **models:** Contains the data transfer objects (DTOs) that are used to transfer data between the application and the frontend app.
 - **email_templates:** Contains the email templates that are used to send emails to the users.
 - **utils:** Contains utility and helper functions that are used across the application.
- **domain:** Contains the domain entities, value objects and enums. The domain entities are the core business objects that are persisted in the database. The value objects are immutable objects that are used to represent the attributes of the domain entities.
- **infrastructure:** Contains the infrastructure code such as database connection settings, repositories, and unit of work implementation. Used SQLAlchemy as the ORM to interact with the database.
 - **migrations:** Contains the Alembic migration scripts that are used to create and update the database schema. To read more about migrations, please read migrations in */backend/docs/migration-guide.md*
- **presentation:** Contains the presentation layer code such as the API routes and seeding database.
 - **routers:** Contains the API routes that define the REST endpoints. To view the API documentation, visit the */docs* endpoint.
 - **seed_db.py:** Contains the script to seed the database with initial data.
- **core:** Contains implementation of the architecture related core classes such as Dependency Injection and Mediator.

Use Cases

To create a new use case, follow these steps:

1. Create a new folder in the *application/commands* or *application/queries* directory with the name of the use case.

2. Create a new command or query class in the folder with the name of the use case. Give the class name **command.py** or **query.py**.
3. Implement the command or query handler in the `handler.py` file.
4. Decorate the handler with the `@Mediator.register_handler({Command or query class})` decorator. In the constructor (`__init__`) of the handler, inject the required services and unit of work classes. To access the database session, you need to first inject **UnitOfWork** class then create a repository for the entity you want to interact with. For example,

```
@Mediator.register_handler(CreatePatientCommand)
class CreatePatientHandler(RequestHandler[CreatePatientCommand, Result]):
    def __init__(self, uow: UnitOfWork):
        self.uow = uow

    def handle(self, command: CreatePatientCommand) -> Result:
        patient_repo = self.uow.get_repository(Patient)
        # Perform database operations here by calling the repository methods.
```

Dependency Injection

The application uses the custom built Dependency Injection (DI) container to manage the dependencies. The DI container is implemented using the `DIContainer` class in the `core` module. To register a new dependency, you need to decorate the class with the `@DIContainer.register_service()` decorator. For example,

```
@DIContainer.register_service(ServiceLifecycle.TRANSIENT)
class MyService:
    def __init__(self, dependency: Dependency):
        self.dependency = dependency
```

There's also FastAPI built-in dependency injection used in the API routes. This DI is used only in the endpoint router functions. In other parts of the application, the custom DI container is used.

Frontend

The frontend is a Next.js application that uses Material-UI as the UI library. The **src** directory contains the following directories: -

- **components**: Contains the reusable components that are used across multiple pages.
- **app**: Contains app routes along with the layout and page components.
 - Each page has a corresponding directory with the page components. For example, the `app/auth/signin` directory contains the components for the sign-in page (`app/auth/signin/page.tsx`).
- **core**: Contains shared services, hooks, utils, and DTOs.

Prefer to use PascalCase file naming convention for all source files except for special Next.js files such as **index.ts**, **page.tsx**, **layout.tsx**, etc.

Services

- **ApiService:** It's a singleton service that is used to make API requests to the backend. It uses native fetch API to make requests. These methods are server-side rendered (SSR) compatible. To call these methods in client components (components where marked as "use client"). Use useSWR hook from SWR library to fetch data from the API.
- **IPInfoService:** A singleton service that is used to get the user's IP address and location information. It uses the ipinfo.io API to get the information. The token is stored in the .env file.

ML

The ML directory contains the ML models and integration API. The **src** directory contains the following directories:

- **dto:** Contains the data transfer objects (DTOs) that are used to transfer data between the backend and the ML models.
- **models:** Contains the ML models that are used to predict disease based on the input data.
- **services:** Contains the service classes that are used to interact with the ML models and perform predictions and send the results back to the backend.
- **utils:** Contains utility and helper functions that are used across the ML application. Some of the utility functions are used to preprocess the input data before passing it to the ML models such as converting the DICOM images to PNG format.

Read the guide to understand how the ML models are integrated with the backend application in `/backend/docs/ml-interaction.md`

Viewer

The viewer directory contains the OHIF viewer. In the `ohif` directory, contains precompiled OHIF viewer code. Only the **app-config.js** can be modified to change the configuration of the viewer. If you want to add a new extension or add a new functionality to the viewer, you need to clone the OHIF viewer [repository](#) and make the changes there. Then build the viewer and copy the compiled code to the `ohif` directory. The **express** server is used to serve static files of the OHIF viewer. The **cors-anywhere** package is used to handle the CORS issues and proxy the requests to the PACS server.

Kubernetes

In the ``k8s`` directory, contains the Kubernetes deployment files to deploy the applications to Azure Kubernetes Service (AKS). The following files are present in the directory.

- **frontend.yaml:** Contains the deployment and service manifest files for the frontend application.
- **ml.yaml:** Contains the deployment and service manifest files for the ML application.

- **viewer.yaml**: Contains the deployment and service manifest files for the viewer and proxy applications.
- **database.yaml**: Contains the deployment and service manifest files for the PostgreSQL database.
- **ingress.yaml**: Contains the ingress manifest file to route the traffic to the backend, frontend, ml, viewer and proxy services.
- **loadbalancer.yaml**: Contains an optional manifest file to create a load balancer service to expose the applications for testing purposes.
- **cert-issuer.yaml**: Contains the cert-manager issuer manifest file to issue the SSL certificates for the applications with Let's Encrypt.

There is a **helm** directory that contains PowerShell scripts to install or uninstall *nginx ingress* and *cert-manager* services.

CI/CD

The project uses GitHub Actions for CI/CD. The workflows are defined in the `.github/workflows` directory. The following workflows are defined:

- **backend-ci.yaml**: Contains the CI workflow for the backend application. It runs the tests and linters on every push to the main branch.
- **deploy-aks.yaml**: Contains the CD workflow to deploy the applications to AKS. It runs when a new release is created. The workflow builds the Docker images, pushes them to the Azure Container Registry, and then deploys the applications to AKS.

Onboarding Guide

This guide will help you get started to onboard new users and healthcare providers to the platform.

Healthcare Provider Onboarding

To add a new healthcare provider, the concept of an *organization* is used. An organization can have multiple users and patients associated with it. The organization can be a hospital, clinic, or any other healthcare provider. To add a new organization, you should have the **app_admin** role. Follow the steps to create a new organization.

1. Log in to the platform with your admin credentials.



Sign In

Email *

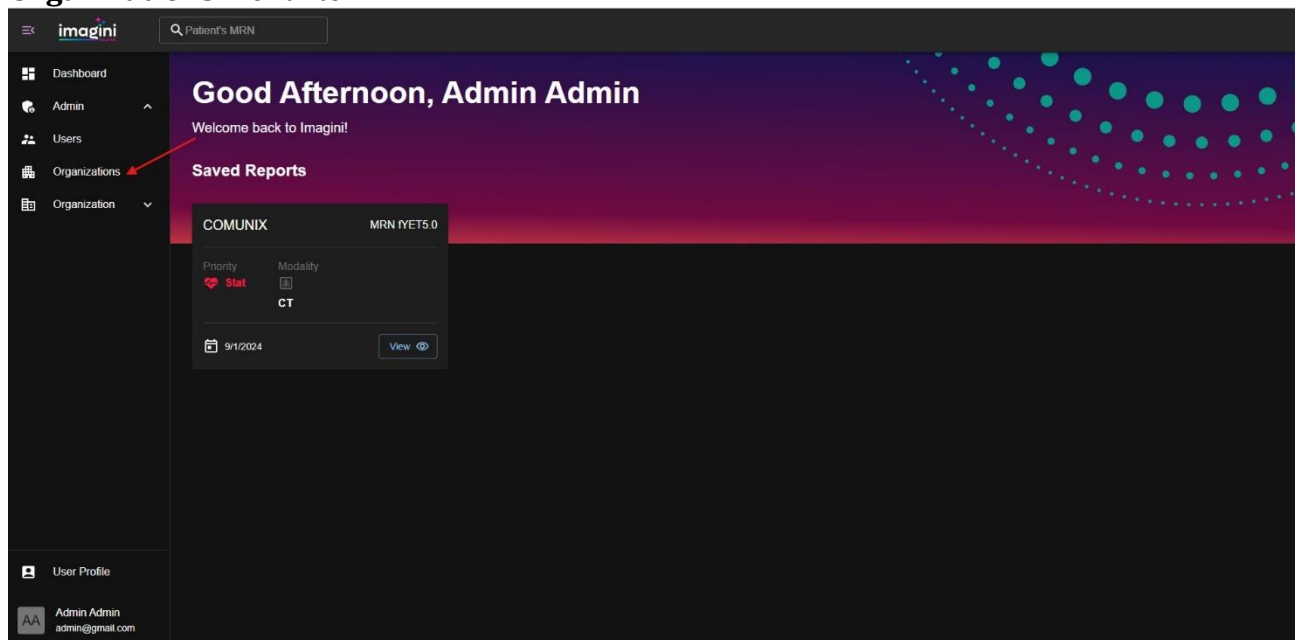
Password *

☐ Remember me [Forgot password?](#)

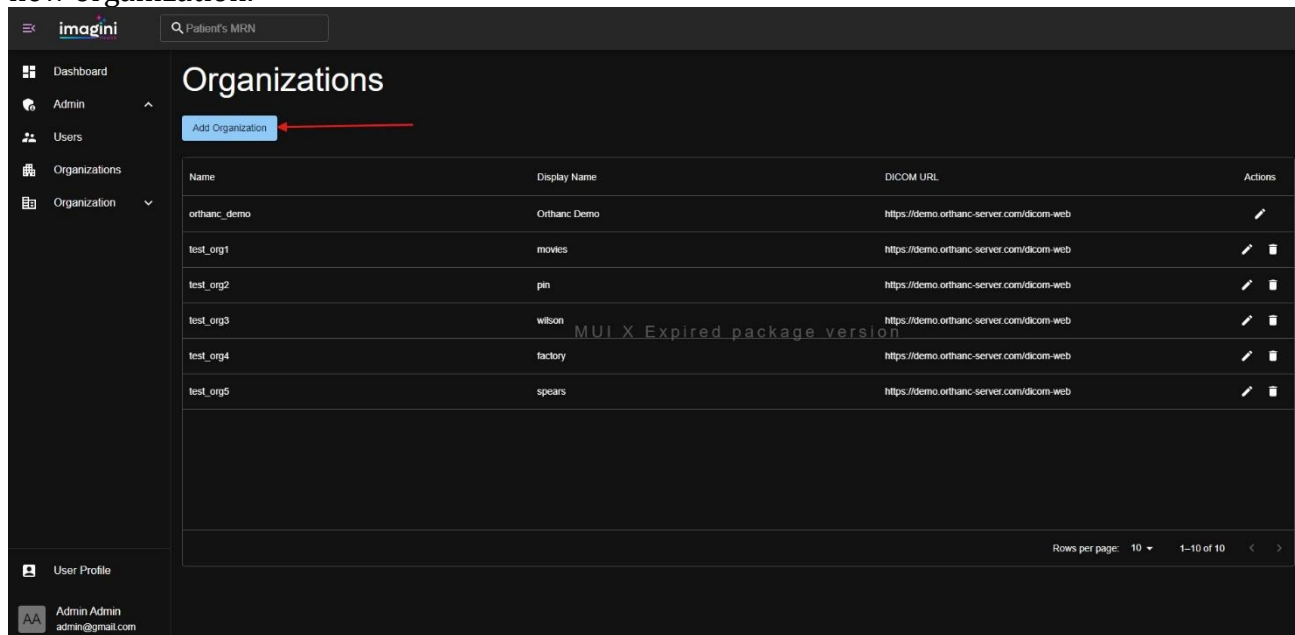
Sign In

[English](#) [Spanish](#)

2. Click on the **Admin** tab in the sidebar. It will open sub-menu items. Click on the **Organizations** menu item.



3. You will see a list of organizations. Click on the **Add Organization** button to create a new organization.



imagini

Search Patient's MRN

Dashboard

Admin

Users

Organizations

Organization

Create Organization

Name *

Organization name rules:

- Must be at least 4 characters long
- Must contain only lowercase letters, numbers, and underscore
- Must not start with a number
- Must not start or end with an underscore

Display Name

DICOM URL *

Website

Email

Address

Create

User Profile

Admin Admin
admin@gmail.com

4. Fill in the organization details such as name, address, phone number, and other details. Click on the **Create** button to create the organization. There are only name and DICOM URL fields required to create an organization. The organization name should be unique. The DICOM URL is used to connect the PACS server with the platform to fetch the patient's medical images. Make sure the provided DICOM URL is accessible from the platform.

How to add organization admin and regular users

After creating an organization, you can add users to the organization. There are two types of users that can be added to the organization: admin and regular users. The admin user can manage the organization settings, users, and patients. The regular user can only view the patients' data and perform image analysis.

Click on the **Users** menu item in the **Admin** tab. For example, you will see a list of users in the entire system. Click on the **Invite User** button to add a new user to the organization. It will open a dialog box to enter the user details. Enter user email address, specify role (None or Organisation Admin), and search for the organization name in the autocomplete field. Click on the *Invite* button to send an invitation to the user. The user will receive an email with an invitation link to join the organization. If the user doesn't have an account, they will be asked to create an account first.

imagini Patient's MRN

Dashboard Admin Organization

Users

Invite User

First Name	Last Name	Email	Role	Organization	Actions
Admin	Admin	admin@gmail.com	super_admin	orthanc_demo	
Len	Herrera	preferred2025@duck.com			
Hank	Phelps	exploration1920@live.com			
Diedre	Frederick	miniature1840@yandex.com			
Shawwna	Nielsen	auctions1975@live.com			
Joaquina	Chaney	argentina1931@yandex.com			
Lawanna	Dickson	fail1909@example.com			
Jeanette	Barnera	encyclopedia2072@duck.com			
Warner	Garcia	undo1894@yahoo.com			

Rows per page: 10 1-10 of 20

User Profile Admin Admin admin@gmail.com

imagini Patient's MRN

Users

Invite User

First Name	Last Name	Email	Role	Organization	Actions
Admin	Admin	admin@gmail.com	super_admin	orthanc_demo	
Len	Herrera	preferred2025@duck.com			
Hank	Phelps	exploration1920@live.com			
Diedre	Frederick	miniature1840@yandex.com			
Shawwna	Nielsen	auctions1975@live.com			
Joaquina	Chaney	argentina1931@yandex.com			
Lawanna	Dickson	fail1909@example.com			
Jeanette	Barnera	encyclopedia2072@duck.com			
Warner	Garcia	undo1894@yahoo.com			

Rows per page: 10 1-10 of 20

User Profile Admin Admin admin@gmail.com

Invite a new user

Email: user@email.com

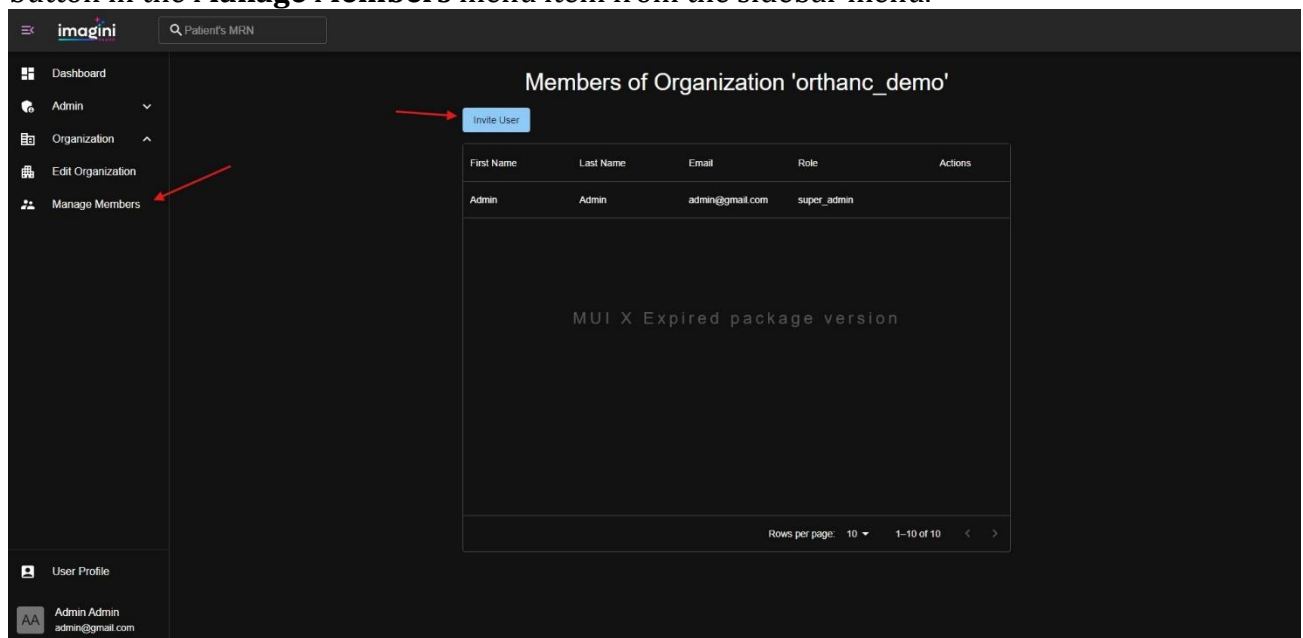
Role: Organization Admin

Organization: test_org1 - movies

Close Send Invitation

Adding a regular user is similar to adding an admin user. The only difference is the role selection. Select the **None** role to add a regular user.

Organization admins can also invite users to the organization by clicking on the **Invite User** button in the **Manage Members** menu item from the sidebar menu.



Note

The **Admin** menu in the sidebar is only visible to the users with the *app_admin* role. The **Organization** menu is only visible to the users with the *organization_admin* role.

Roles

The platform has the following roles:

- **super_admin**: The super admin role that has access to all functionalities in the system. This role is not visible in the UI. Only developers have access to this role to manage the system settings and troubleshoot issues.
- **app_admin**: The highest admin role that has access to all the organizations and users in the system. This role can create new organizations, users, and manage the system settings.
- **org_admin**: The admin role for the organization. This role can manage the organization settings, users, and patients.

How to use the platform

After onboarding the healthcare providers and users, they can use the platform to view patients' data and medical images. To search for a particular patient, type Patient ID in the search bar at topbar. It will show the patient DICOM studies and series. Click on the **+** icon to view patient's series associated with the study. Click on **Load in Viewer** button to view the medical images in the viewer.

the patient series table. You will see **View Report** button in the table. Click on the button to view the radiology report.

imagini

Q IYET5.0

COMUNIX


MRN #IYET5.0

1941-09-01

83 years old

Mar 4, 2004

CT HEAD/NECK 5.0 B30s



View in full screen

AA

Radiology Report

Patient Information

Patient Name: COMUNIX

Date of Birth: 1941-09-01 (83 years old)

Gender:

MRN: IYET5.0

Study Date: 9/1/2024

Referring Physician: Admin Admin

Clinical Information

clinical 1

Indication

indication 1

Technique

test technique

Findings

Lung Opacity

60.57%

Add this finding to your report?

Approve

Decline

Description

Pneumonia

61.12%

Description