Project Report On

Docsy

Powered By –

Submitted by -

Souvik Ghosh
Kirti Garg
Akanksha Kushwaha
Raghib Shams
Suraj Pratap Singh
Kunnathu Pavan Kumar
Shailesh Kumar Sharma
Sushanth Ananthabhotla

Department of Product Engineering

Innovaccer Analytics Private Limited

Contents

1. Introduction

This project is a EHR Application System which is inspired by OpenEHR clinical Knowledge manager where patients can perform following tasks:-

1. They can login/register in the application using their credentials.
2. They can book an appointment.
3. They can view/download their prescription, lab reports, imaging reports.

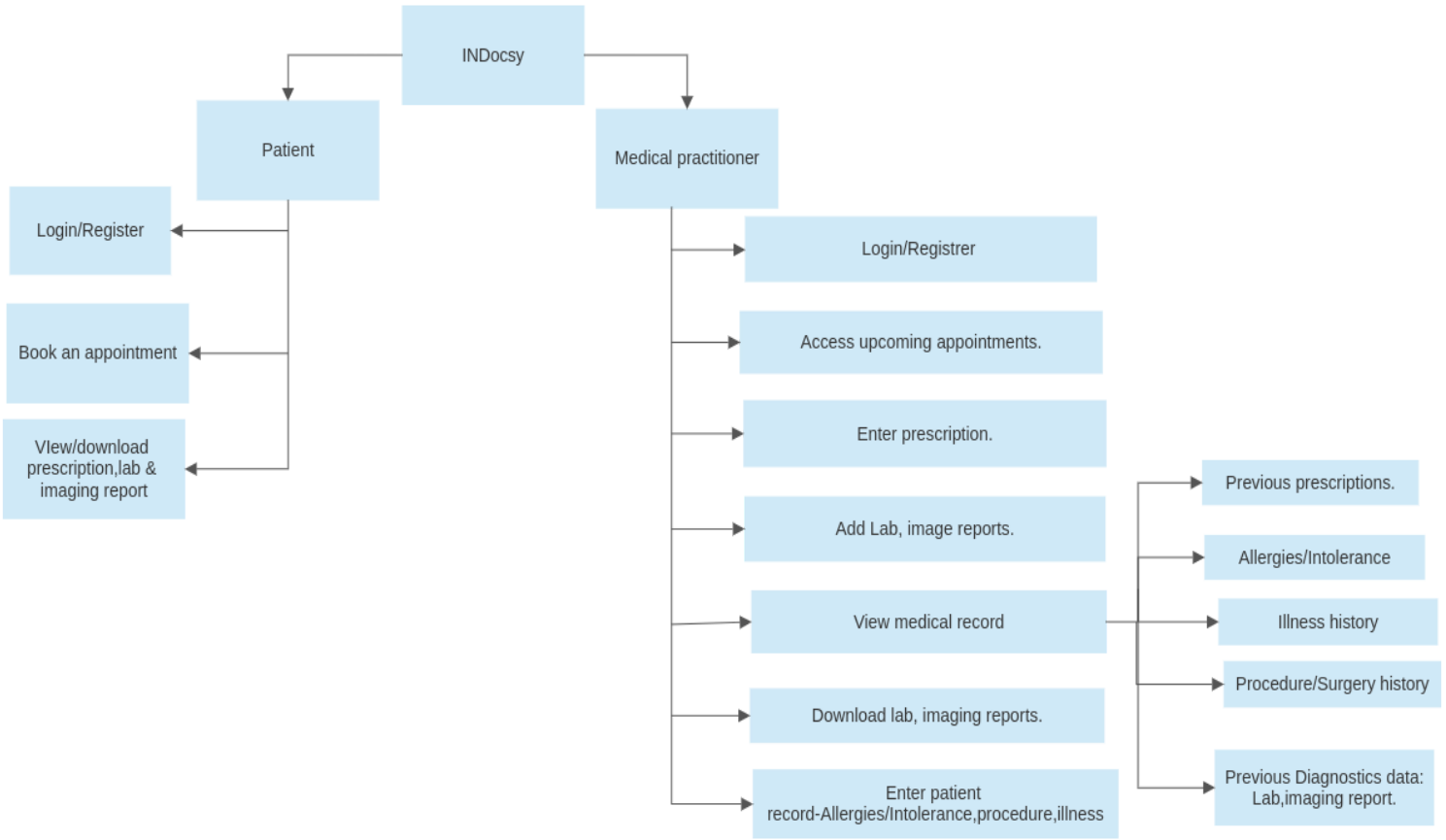
Medical practitioners can perform the following tasks:

1. They can login/register in the application using their credentials.
2. They can access upcoming appointments through their dashboard.
3. They can enter prescriptions for the particular patient.
4. They can add Lab,Imaging Records for the particular patient.
5. They can view medical records which contains :
    a. Previous prescriptions
    b. Allergies/Intolerances
    c. Illness History
    d. Procedure/Surgery History
    e. Previous Diagnostic Data - Lab Reports , Imaging Reports.
6. They can Download Lab & Imaging reports.
7. They can Enter patient’s Allergies, Procedures, Illness to their Record.

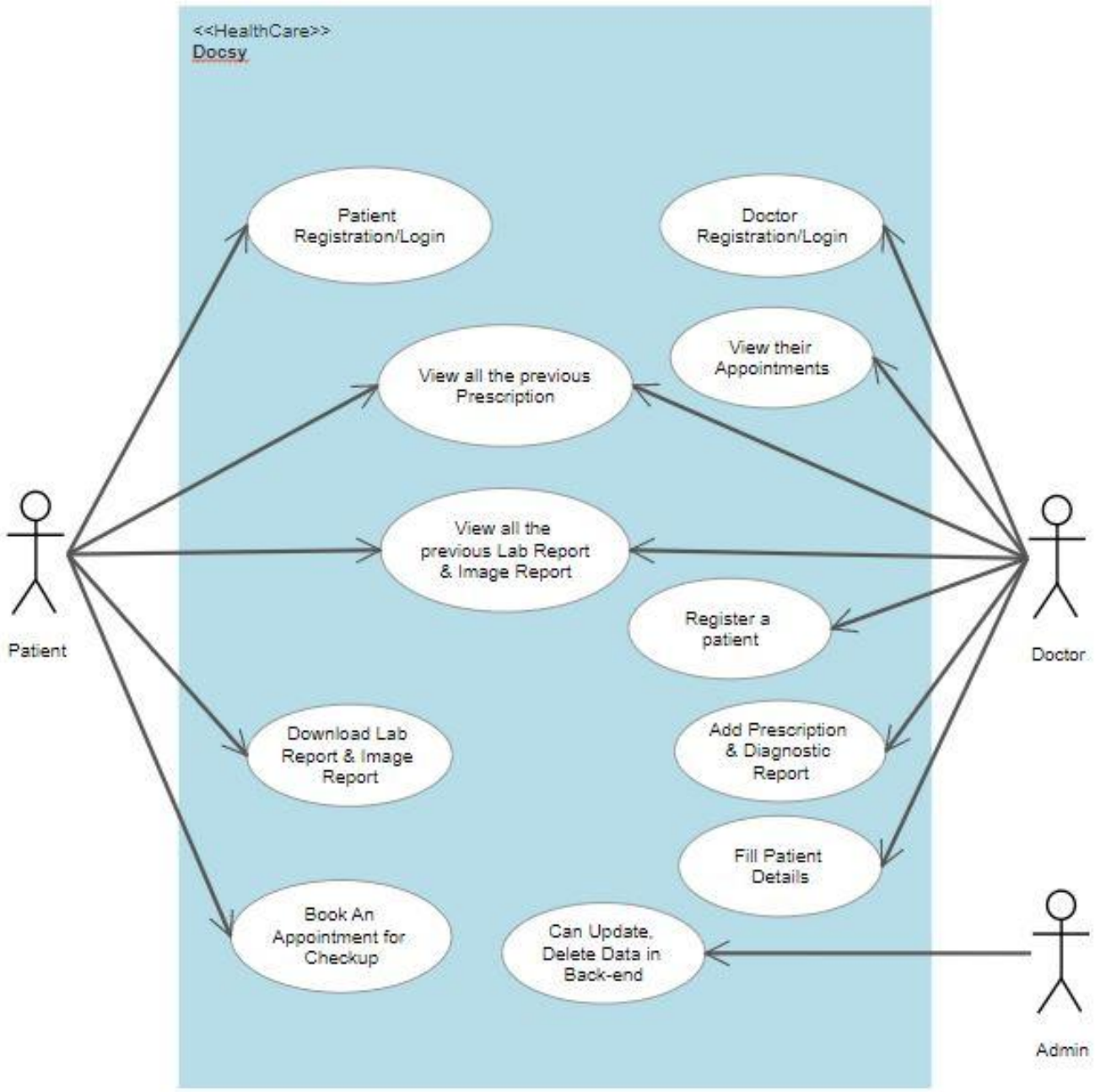
2. Features and workflow

Our website has several features –

https://imgur.com/a/wBunOma

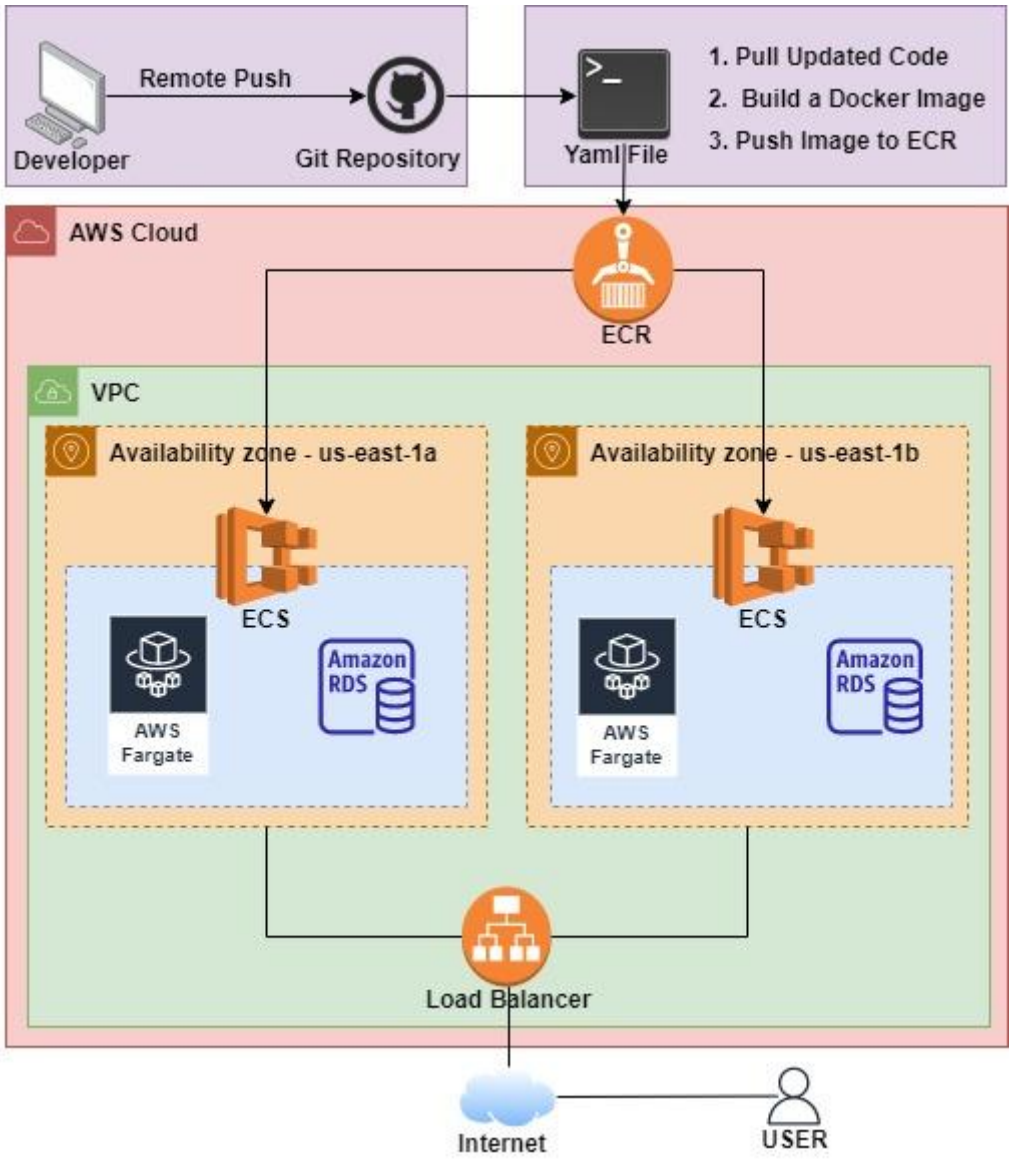


3. UML Diagram



UML Diagram for INDocsy

4. Architecture Diagram of Deployment



CI/CD Architecture

5. Local Testing & Deployment

Our project is totally a django based project. So, for the frontend part, we have used django templates only. Over here, we have used command **python3 manage.py runserver** to run our server locally at http://127.0.0.1:8000/ though django uses an 8000 port for hosting.

In case of UI testing we have made a separate folder named “UITesting” where our UI is being tested. For that, we have used sqlite3 present locally and for Deployment we have used Postgres only.

6. Deployment Flow

1. We will go to the ECS service in our AWS.
2. There first of all we will create the Cluster and give the requirements we want.
3. After creating Cluster now we will create Task Definitions in that cluster and the naming convention is very important here.
4. In Task Definition we will use the Fargate environment instead EC2.
5. In the Task definition we will add the container name and URI of our image with the latest tag.
6. Fargate is **a serverless environment for running containers that allows users to completely remove the need to own, run, and manage the lifecycle of a compute infrastructure**.
7. In all these processes we must take care of VPC,Networks and Security Groups.
8. Now we will create the Service through the Task Definition we have created.
9. Now we copy the Task definitions json file from the Task definition console.
10. After all these steps we will start our Start, Build, Push and Deploy process through Github Actions.
11. Our Dockerfile builds the image through Github Actions.
12. After that the image will be pushed to ECR through Github Actions.
13. So we have the latest image in the ECR.
14. Now our image will run as a container and will be deployed on fargate.

Problems

1. We know that Fargate is serverless so after every deployment we will be assigned a different Public Ip.
2. So to get rid of this I have used load balancer as a reverse proxy.
3. Very careful with naming conventions and ports.

Benefits of reverse proxy

1. Load balancing.
2. Caching content and web acceleration for improved performance.
3. More efficient and secure SSL encryption.
4. Protection from server attacks and related security issues.

Implementation of Sonarqube

Sonar covers the following sections of code quality

1. Architecture and Design
2. Unit tests
3. Duplicated code
4. Potential bugs
5. Complex code

SonarQube receives files as an input and analyzes them along with barriers. Then calculates a set of metrics, stores them in a database and shows them on a dashboard. This recursive implementation helps in analysis of code quality and how code improves over time.

7. Technology Stack

We have created our application using Django Framework.

1.Frontend Technology -

1. HTML5
2. CSS3
3. Javascript
4. Bootstrap
5. Jquery

2. Back-End Technology –

1. Django
2. Django Template

3. Database Connection –

1. Postgres + Sqlite3

4. Testing -

- Manual Testing
- UI Testing - Using Selenium (POM based)
- Database Model Testing

4. DevOps

1. Amazon AWS(ECR, ECS, Fargate)
2. Sonarqube

8. Limitations

Few Limitations in this application are as follows:

1. This application gives same features or a display for all medical workers ie. be it a doctor,nurse or whoever logins will get the same dashboard/same set of features.
2. This Application does not consider doctor's availability for patients to book an appointment.
3. Medical Practitioners can not update the prescription for the same problem. they have to write it all over again and then perform further addition.
4. Patients can not hide any medical record from the doctor.

6. Future Scope

1. Pharmacist roles can be integrated with the present application system.
2. Reminders can be added for patients to take medicines on time and view booked appointments.
3. Creating individual roles and permissions to access applications for every medical worker.
4. Chat option for patients and doctors/caretakers.
5. Online video Consultation with doctor.
6. Family Medical Records option.
7. Adding more detailed /exhaustive records for patients.