**Pre - Microsoft Azure HDS Demonstration  
  
Building an EHR platform from scratch.**

1. **Information collected before implementing this project.**
2. **What kind of data is collected and used in EHR?**

* EHR data is stored in various formats depending on the use case. For structured, interoperable records across systems, FHIR is the preferred data standard — it’s modular, API-friendly, and widely supported by big tech. DICOM is used for imaging data like MRIs, where both pixel and metadata are important. While CSV is simpler and more ML-friendly, it lacks the clinical rigor and structure of FHIR. So here, I’ve worked with FHIR APIs and DICOM images for ML training and data pipelines.

1. **What is the role of Microsoft Azure Health Data Services?**

* Super organized health data - such as patient records, x-ray images, heart rate data, history of patient’s entire life’s medical records, etc.
* Main use case of this platform is -> to make sure the sensitive data is securely stored -> Easy to share between systems (like different hospitals) and can also generate meaningful insights like predicting patient health in future -> Can generate visuals using power bi.

1. **What is FHIR, DICOM and MedTech in short?**

* **FHIR** is just a universal standard way to format patient data. so that, different hospital systems can share it easily. For ex - name - age- medical conditions - etc. We don’t create or code FHIR. It is a ready-made format, also provided by Azure HDS. We just put data into it and do CRUD. Similarly, **DICOM** is a set standard for storing and sharing medical images like X-rays, or MRIs. It has a special file format for medical pictures which also include some extra information like patient names, date of scan, etc. **MedTech** is a tool to collect information from devices like smartwatches or blood sugar monitors and turn it into the FHIR format for the doctors to use.

1. **What do we do on Azure HDS exactly in a summarized manner?**

* Though we don't need to create Rest APIs from scratch as Azure HDS itself provides pre-built for FHIR and DICOM. We just need to call them using simple code to use them in our applications. (So, the skills needed here are -> Backend Operations including REST API and Data Ingestion)
* So based on the data storage and the fetching part (rest api and data ingestions), further we can use this data for particular use cases. for example, in Azure:
* we can store the billing data as FHIR documents and use them for analysis in our own built models on Azure Machine Learning.
* similarly, we can also use Xray or MRI data and store them as DICOM documents and use them for predictions using our models. -> The code required here will simply be to fetch the data, and then using this data as an input for our ML models.
* (So, the skills needed here are -> REST API again -> Understand Text extractions from FHIR and OpenCV/Image processing for DICOM)
* Displaying data using Power BI to show charts, etc. (So, the skills needed here is Power BI for data visualization)

1. **Our Project Plan:**
2. **Problem Statement:**

* Demonstrate an end-to-end healthcare data platform that mimics the capabilities of Microsoft Azure Health Data Services — including FHIR & DICOM data ingestion, AI integration, and interactive dashboards — built from scratch using open-source tools and deployed AI models.

1. **Use Case – Like a real-world story:**

* *“Dr. Smith logs into the EHR dashboard to review the clinical and imaging data of her patients. For each patient, she sees their vitals, diagnoses, lab reports, imaging results, and AI-powered risk predictions — all on one screen. The system also flags patients at high risk for adverse outcomes based on FHIR data and highlights abnormal findings in uploaded X-ray images using AI.”*

1. **Noting down features of our project (Categorized) :**
2. **FHIR Data Features:**

* Parse FHIR bundle (Patient, Encounter, Observation, Condition, etc.)
* Visualize patient history (timeline)
* Filter/search patients by condition
* Extract structured fields for ML

1. **DICOM Data Features:**

* Upload/view DICOM (using pydicom + OHIF or cornerstone.js)
* Run AI inference (e.g., pneumonia/COVID classifier)
* Show prediction overlays
* Link to FHIR ImagingStudy

1. **AI Features:**

* ML model for predicting patient risks (tabular)
* DL model for image classification (vision)
* Output saved in FHIR-like format
* Model evaluation view (accuracy, ROC, etc.)

1. **Dashboard/UI Features (React):**

* Login screen (mock roles)
* Patient list + filters
* Patient profile page -> Tabbed views: Clinical | Imaging | Predictions
* Graphs/charts using D3 or Chart.js
* Notification cards (e.g., “Patient at risk of readmission”)

1. **Backend (FastAPI):**

* API endpoints:
  + GET /patients
  + GET /patient/{id}
  + POST /predict/tabular
  + POST /predict/image
* AI model loading + inference
* Connect frontend + model + data

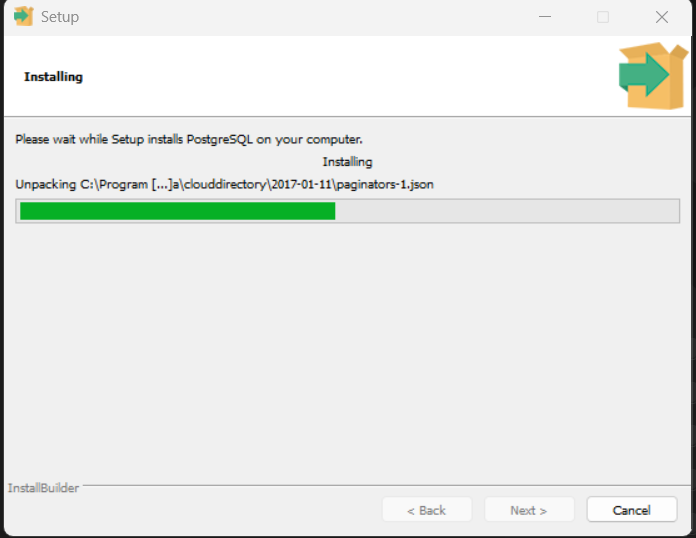
1. **Steps decided as of now:**
2. **Parse and Normalize FHIR Data (Backend)**

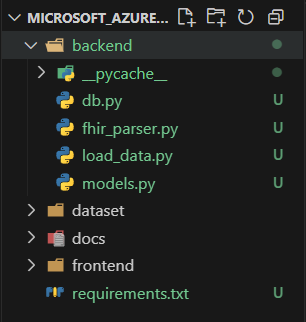
* Install FHIR parsing libraries:
* Create a fhir\_parser.py inside your backend directory.
  + Read all .json files.
  + Extract key resources: Patient, Observation, Condition, Encounter, MedicationRequest.
* Save extracted data into PostgreSQL.

1. **Build Backend API (FastAPI or Flask) - Serve patient and AI data via REST endpoints**
2. **Build Frontend Skeleton (React) - Build a clean UI to show patient dashboard.**
   * + **Home Page – A column of all the patients with the named and other simple details with a search bar on top which will display all the patients when name typed in.**
     + **Once we click on a patient name, we will open that patient’s dashboard. This dashboard will have the 3 tabs we discussed above.**
3. **AI Model Integration**
   * + Extract features like vitals, age, comorbidities
     + Train AI models (scikit-learn) to predict:
       - Readmission risk
       - Diabetes likelihood
       - Length of stay
       - Download public DICOM datasets (e.g., RSNA Pneumonia Dataset)
       - Train a CNN model (PyTorch or TensorFlow)
       - Save model & inference function
       - Hook it to POST /predict/image
4. **Integrate AI Outputs into Backend & Frontend**
   * + Save AI predictions as synthetic Observation resources
     + Display on frontend in Prediction’s tab for each patient
5. **Add Bonus Features (Later Phase)**
   * + DICOM viewer integration (OHIF, cornerstone.js)
     + Graphs (vitals trends)
     + Patient timeline view
     + Power BI connection to structured CSVs or DB
6. **Tech Stack decided for this project:**

|  |  |
| --- | --- |
| **Layer** | **Stack** |
| Frontend | React + Tailwind + Chart.js / D3 |
| Backend API | FastAPI or Flask |
| ML/CV Models | Scikit-learn + PyTorch |
| DICOM Parser | pydicom, SimpleITK |
| FHIR Parser | fhir.resources (Python) |
| Database | SQLite / PostgreSQL |
| Visualization | Power BI (optional) |
| Deployment | Docker + GitHub Actions |
| Optional | HAPI FHIR Server |

1. **Let’s Start!**
2. Installing PostgreSQL database server.



1. Create a very modular file structure, along with a python data pipeline.

- Here we created 4 files, that read and parse the complex JSON files (FHIR).

- Extract specific structured data.

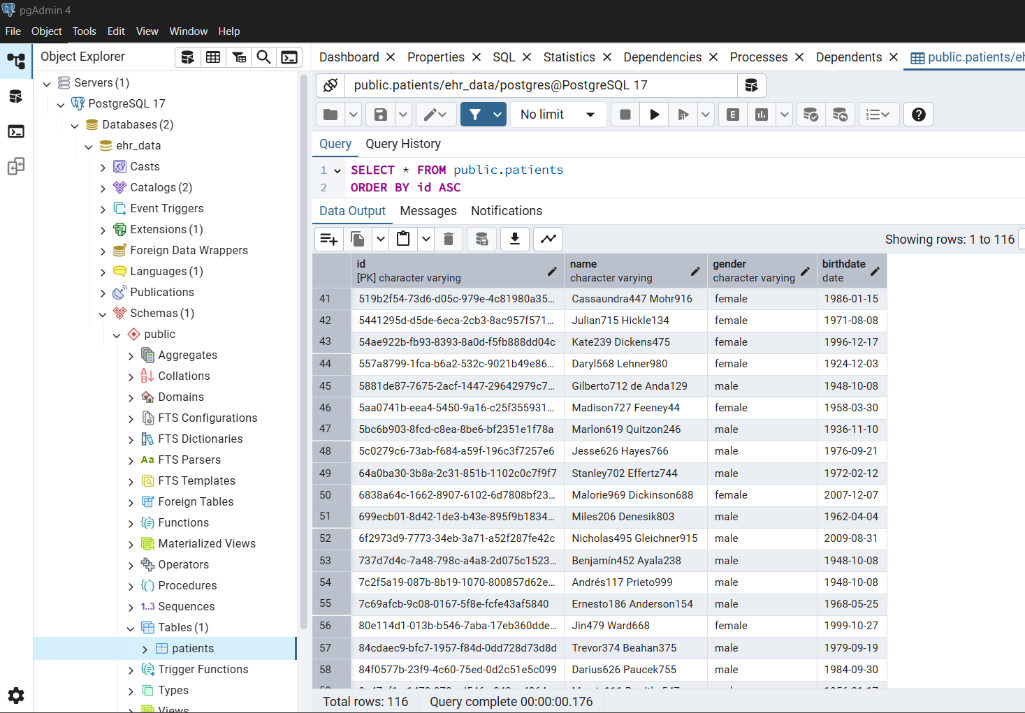
- Connects to our local PostgesSQL database.

- Creates a table if not already.

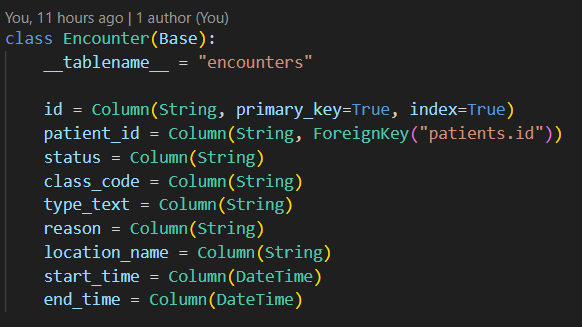
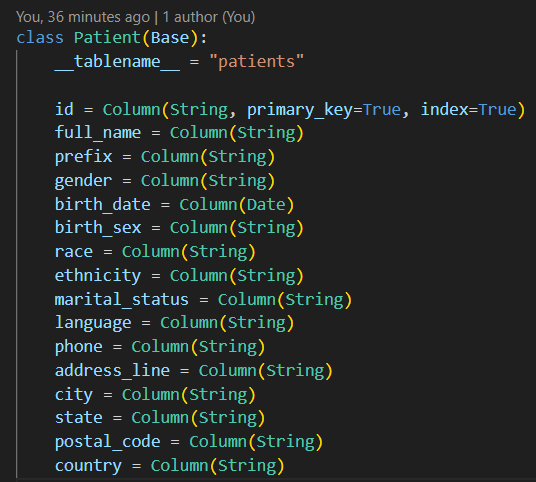
- Inserts the parsed data into that table.

- Allows us to view that table in the pgAdmin panel.

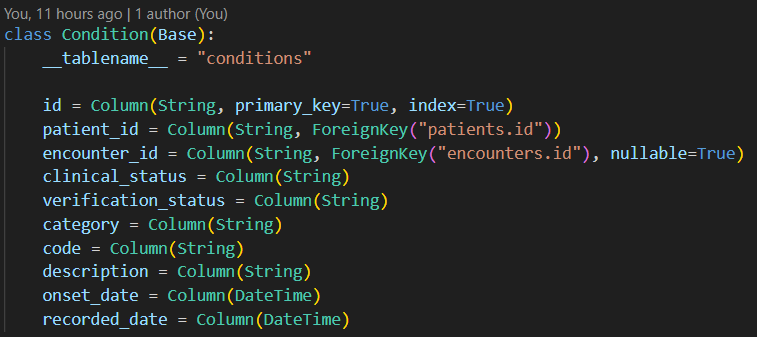
Here we tested our data pipeline, and tried to create a “patients” table with a few basic patient details, and checked it in the pgAdmin UI tool:

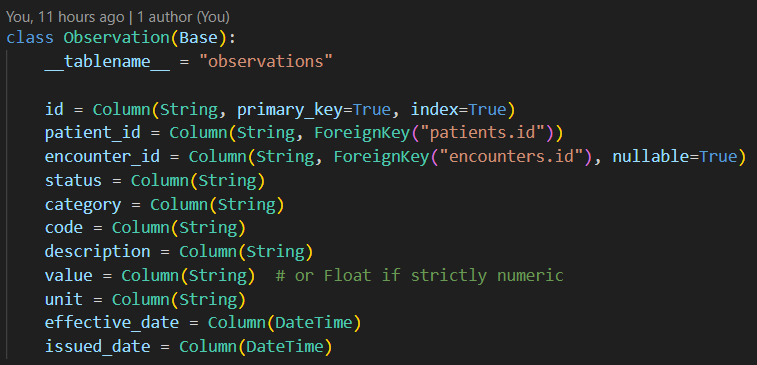


1. Now that we have tested our data pipeline, we created a FastAPI backend, and tested our API endpoints with this database server. Once our API works, we will create our entire database using the FHIR dataset.
2. Now we will create our entire ***database pipeline***, one after other.  
   Note: FHIR dataset has roughly 18 kinds of records of each patient. Hence, for this project we will select approx. ***6-7 for demonstrations***. Each table data model along with name and column names are displayed below.
   * + Starting with the ***patients*** and ***encounter*** table model:

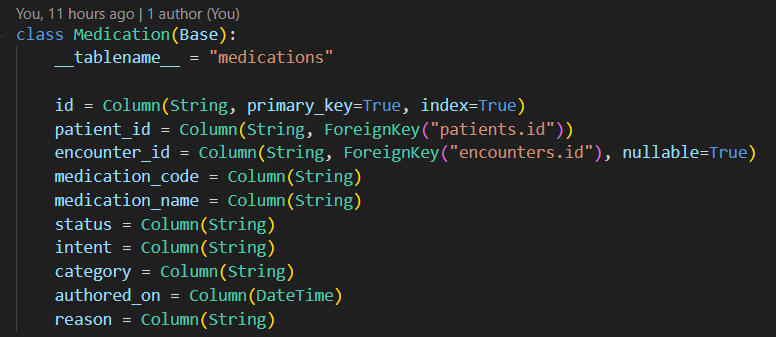


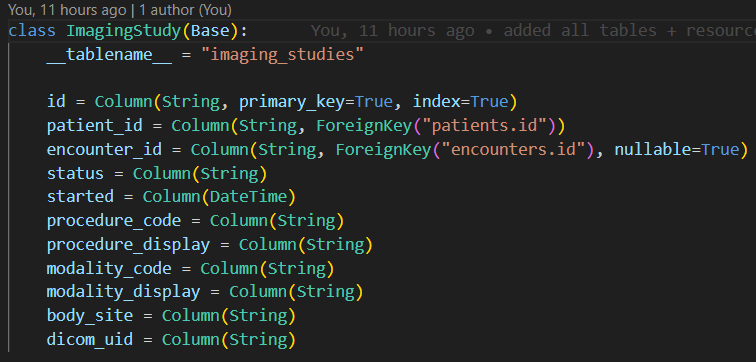
* + - ***conditions*** and ***observations*** table model:



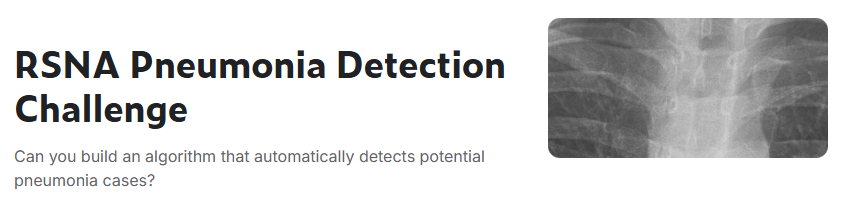


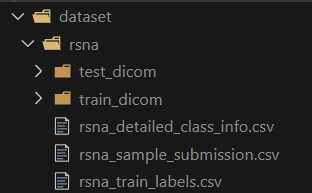
* + - ***medications*** and ***imaging\_studies*** table model





1. Now once we are done with the FHIR dataset, we will download a ***DICOM dataset*** from ***Kaggle***. Here we downloaded the ***RSNA Pneumonia Detection Dataset***, which we will be using to train and test our CNN AI model to predict Pneumonia.

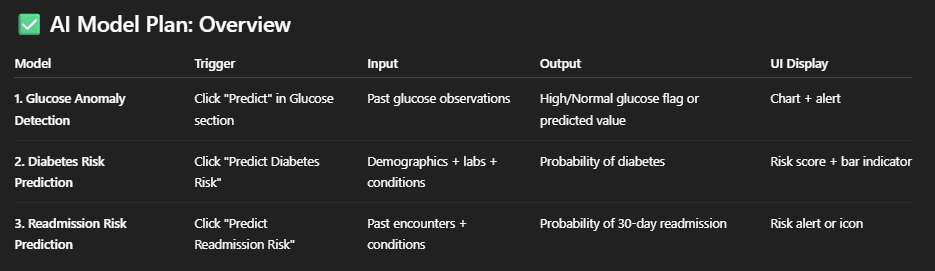




We will store all the DICOM ***meta data*** of the training data in our database.

1. Now we create and load AI Models for FHIR Data followed by the DICOM data before we visualize all the data on our React UI.
   * + Starting with ***FHIR Data AI Models*** Plan:

As we have roughly 6 types of tables for the FHIR data, we will select a few to make some predictions and test our ML models on it.



1. This is the ***directory structure*** we designed.

The workflow of these models is very simple, and the code maintained in our project is pretty sophisticated.

backend/

│

├── main.py ← This is our FastAPI server

├── db.py ← This file connects our project to postgresSQL

├── models.py ← Here we define all the db table schemas

├── fhir\_parser.py ← This file parses all the required info from FHIR data

├── dicom\_parser.py ← This file parses all the DICOM images

│

├── ml/ ← Contains all the ML models used by FastAPI api calls

│ ├── \_\_init\_\_.py

│ ├── glucose\_model.py ← Build + load glucose anomaly model

│ └── diabetes\_model.py ← etc tec

│

├── train/ ← Jupyter files to train our models on data & export

│ ├── glucose\_training.ipynb ← Notebook to experiment and train and others

│ └── models/ ← Stores all trained models, used in ml/.

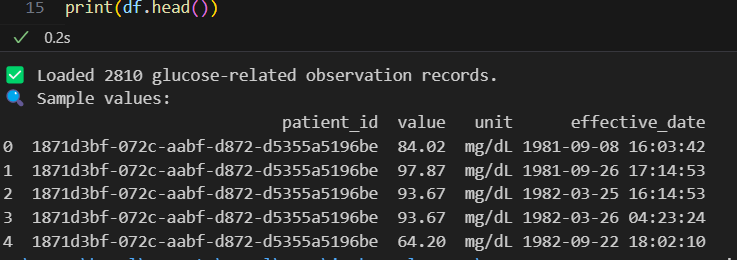
│ └── glucose\_model.pkl ← Exported trained model, and others

1. Okay, so let start with the ***Glucose Anomaly Detection Model*** in our project.

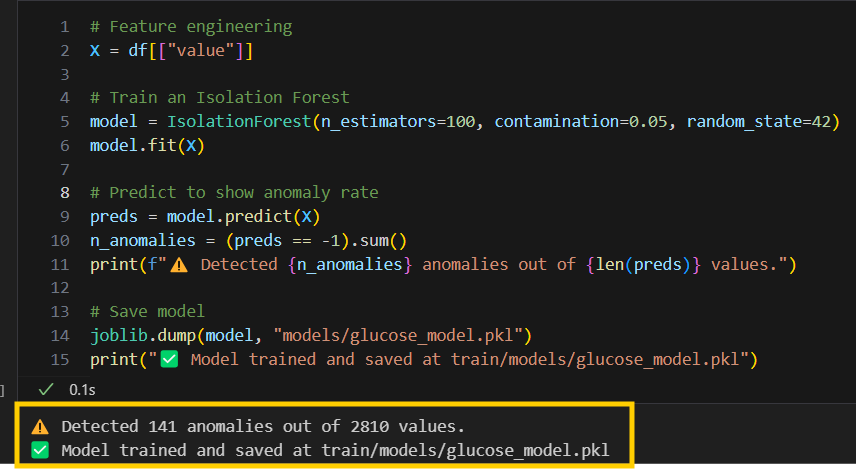
* We chose ***Isolation Forest model*** for Glucose Level Anomaly Detection because:
  + It's an unsupervised algorithm, meaning we don’t need labelled data ("normal" vs. "abnormal") — which is true in your case.
  + It’s fast and scales well with datasets like yours (a few thousand rows).
  + It works well for univariate or low-dimensional data like glucose values.
  + It detects outliers by randomly partitioning data points and isolating the rare ones quicker.

Bottom line: For datasets with numerical inputs, no labels, and where we want to flag rare/odd values, Isolation Forest is one of the best plug-and-play options.

* Output of fetched values from the Observation Table:



* Output of our model finding Anomalies:



* Testing our API Endpoint output (input taken is a patient id from the FHIR data):



* Were there any other options than Isolation Forest Model?
  + We could’ve used Autoencoder (Neural Network) but it is more useful if we consider more features such as time line, age, glucose, etc.
  + Similar to IF, we have LOF – Local Outlier Forest. This model works well if anomalies are locally sparse but is slower than IF on big datasets.

1. Moving on to our next ML model in this project for **Diabetes Prediction Model.**

* We chose