Pre - Microsoft Azure HDS Demonstration

Building an EHR platform from scratch.

A. Information collected before implementing this project.

1) **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.
2) **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.
3) **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.
4) **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)

B. Our Project Plan:

1) **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.
2) **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."*

C. Noting down features of our project (Categorized) :

1) **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
2) **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
3) **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.)
4) **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")

5) **Backend (FastAPI):**
   - ➔ API endpoints:
     - GET /patients
     - GET /patient/{id}
     - POST /predict/tabular
     - POST /predict/image
   - ➔ AI model loading + inference
   - ➔ Connect frontend + model + data

D. Steps decided as of now:

1) **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.

2) **Build Backend API (FastAPI or Flask) - Serve patient and AI data via REST endpoints**

3) **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.**

4) **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

5) **Integrate AI Outputs into Backend & Frontend**
   - ➔ Save AI predictions as synthetic Observation resources
   - ➔ Display on frontend in Prediction's tab for each patient

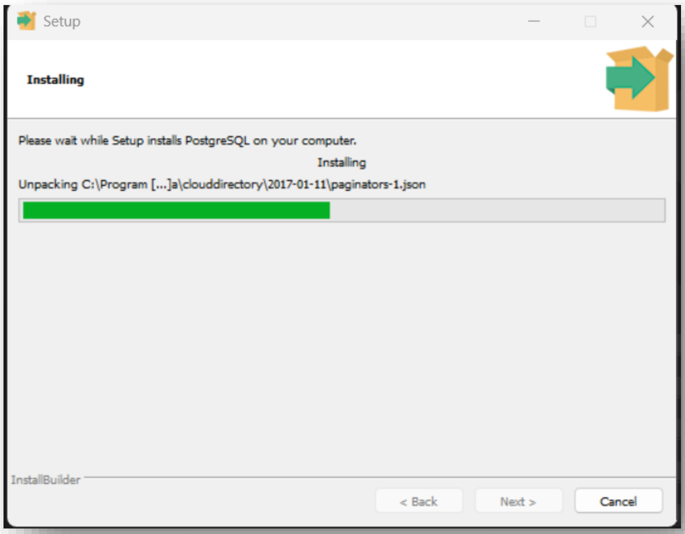
6) **Add Bonus Features (Later Phase)**
- ➔ DICOM viewer integration (OHIF, cornerstone.js)
- ➔ Graphs (vitals trends)
- ➔ Patient timeline view
- ➔ Power BI connection to structured CSVs or DB

E. 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

F. Let's Start!

1. Installing PostgreSQL database server.



Installing

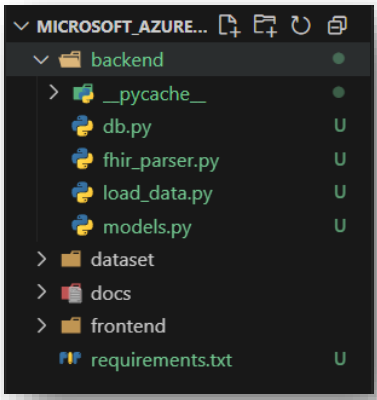
Please wait while Setup installs PostgreSQL on your computer.

Installing

Unpacking C:\Program [...]a\clouddirectory\2017-01-11\paginators-1.json

InstallBuilder

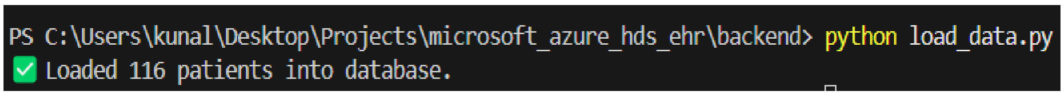
2. Create a very modular file structure, along with a python data pipeline.



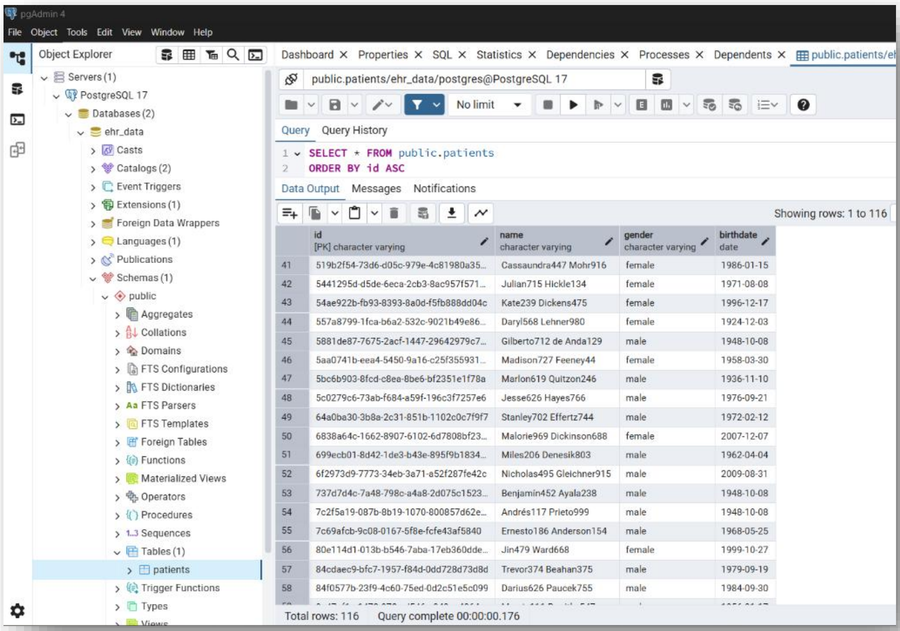
```
MICROSOFT_AZURE...
  backend
    __pycache__
    db.py            U
    fhir_parser.py   U
    load_data.py     U
    models.py        U
  dataset
  docs
  frontend
  requirements.txt   U
```

- 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:



```
PS C:\Users\kunal\Desktop\Projects\microsoft_azure_hds_ehr\backend> python load_data.py
✅ Loaded 116 patients into database.
```



pgAdmin 4
File Object Tools Edit View Window Help
Object Explorer
Dashboard Properties SQL Statistics Dependencies Processes Dependents
public.patients/ehr_data/postgres@PostgreSQL 17
No limit
Query Query History
Data Output Messages Notifications

```
SELECT * FROM public.patients
ORDER BY id ASC
```

Servers (1) > PostgreSQL 17 > Databases (2) > ehr_data > Schemas (1) > public > Tables (1) > patients

Showing rows: 1 to 116

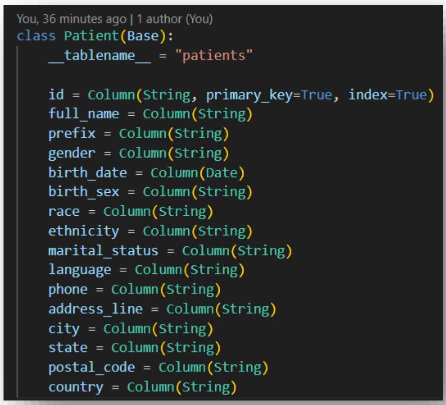
	id [PK] character varying	name character varying	gender character varying	birthdate date
41	519b2f54-73d6-d05c-979e-4c81980a35...	Cassaundra447 Mohr916	female	1986-01-15
42	5441295d-d5de-6eca-2cb3-8ac957f571...	Julian715 Hickle134	female	1971-08-08
43	54ae922b-fb93-8393-8a0d-f5fb888dd04c	Kate239 Dickens475	female	1996-12-17
44	557a8799-1fca-b6a2-532c-9021b49e86...	Daryl568 Lehner980	female	1924-12-03
45	5881de87-7675-2acf-1447-29642979c7...	Gilberto712 de Anda129	male	1948-10-08
46	5aa0741b-eea4-5450-9a16-c25f355931...	Madison727 Feeney44	female	1958-03-30
47	5bc6b903-8fcd-c8ea-8be6-bf2351e1f78a	Marlon619 Quitzon246	male	1936-11-10
48	5c0279c6-73ab-f684-a59f-196c3f7257e6	Jesse626 Hayes766	male	1976-09-21
49	64a0ba30-3b8a-2c31-851b-1102c0c7f9f7	Stanley702 Effertz744	male	1972-02-12
50	6838a64c-1662-8907-6102-6d7808bf23...	Malorie969 Dickinson688	female	2007-12-07
51	699ecb01-8d42-1de3-b43e-895f9b1834...	Miles206 Denesik803	male	1962-04-04
52	6f2973d9-7773-34eb-3a71-a52f287fe42c	Nicholas495 Gleichner915	male	2009-08-31
53	737d7d4c-7a48-798c-a4a8-2d075c1523...	Benjamin452 Ayala238	male	1948-10-08
54	7c2f5a19-087b-8b19-1070-800857d62e...	Andrés117 Prieto999	male	1948-10-08
55	7c69afcb-9c08-0167-5f8e-fcfe43af5840	Ernesto186 Anderson154	male	1968-05-25
56	80e114d1-013b-b546-7aba-17eb360dde...	Jin479 Ward668	female	1999-10-27
57	84cdaec9-bfc7-1957-f84d-0dd728d73d8d	Trevor374 Beahan375	male	1979-09-19
58	84f0577b-23f9-4c60-75ed-0d2c51e5c099	Darius626 Paucek755	male	1984-09-30

Total rows: 116 Query complete 00:00:00.176

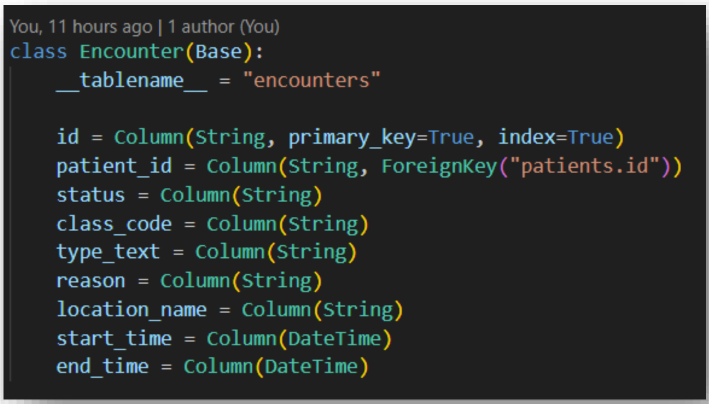
3. Now that we have tested our data pipeline, we created a FastAPI backend, and tested our API endpoints with this database server. Once our API works, we will create our entire database using the FHIR dataset.

4. 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:

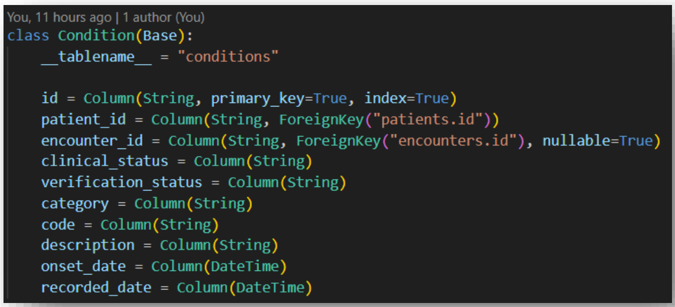


```
You, 36 minutes ago | 1 author (You)
class Patient(Base):
    __tablename__ = "patients"

    id = Column(String, primary_key=True, index=True)
    full_name = Column(String)
    prefix = Column(String)
    gender = Column(String)
    birth_date = Column(Date)
    birth_sex = Column(String)
    race = Column(String)
    ethnicity = Column(String)
    marital_status = Column(String)
    language = Column(String)
    phone = Column(String)
    address_line = Column(String)
    city = Column(String)
    state = Column(String)
    postal_code = Column(String)
    country = Column(String)
```

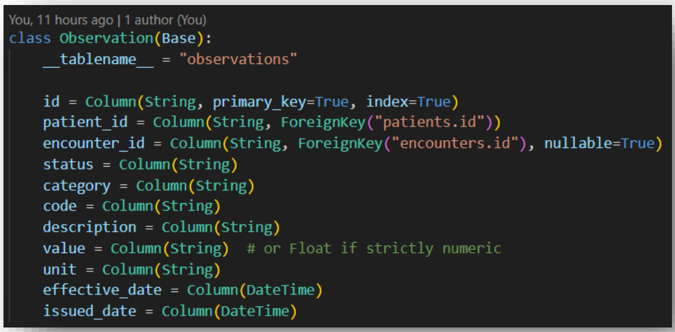


```
You, 11 hours ago | 1 author (You)
class Encounter(Base):
    __tablename__ = "encounters"

    id = Column(String, primary_key=True, index=True)
    patient_id = Column(String, ForeignKey("patients.id"))
    status = Column(String)
    class_code = Column(String)
    type_text = Column(String)
    reason = Column(String)
    location_name = Column(String)
    start_time = Column(DateTime)
    end_time = Column(DateTime)
```

➔ ***conditions*** and ***observations*** table model:

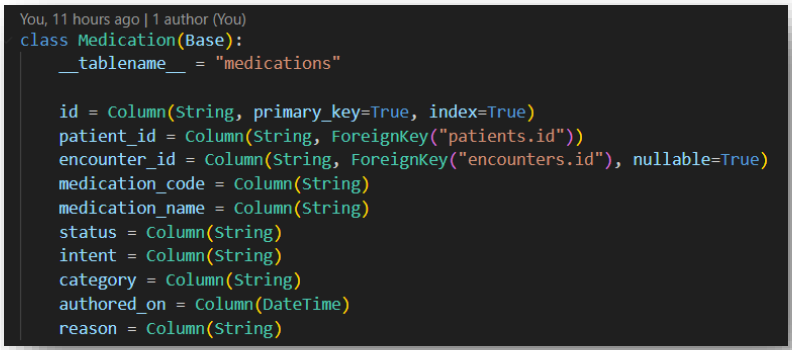


```
You, 11 hours ago | 1 author (You)
class Condition(Base):
    __tablename__ = "conditions"

    id = Column(String, primary_key=True, index=True)
    patient_id = Column(String, ForeignKey("patients.id"))
    encounter_id = Column(String, ForeignKey("encounters.id"), nullable=True)
    clinical_status = Column(String)
    verification_status = Column(String)
    category = Column(String)
    code = Column(String)
    description = Column(String)
    onset_date = Column(DateTime)
    recorded_date = Column(DateTime)
```

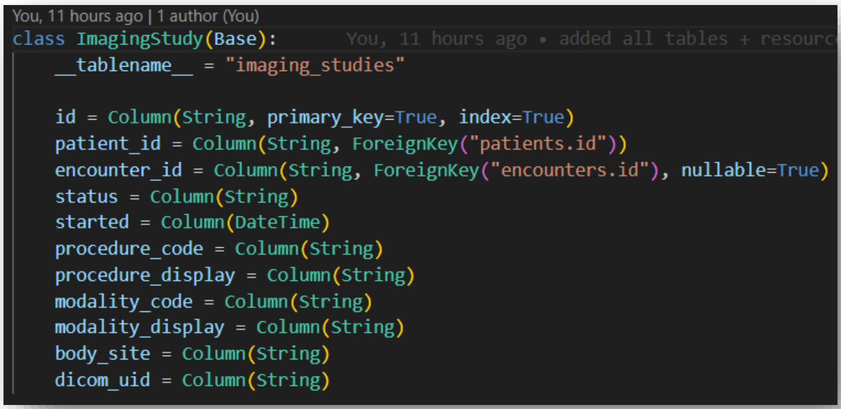


```
You, 11 hours ago | 1 author (You)
class Observation(Base):
    __tablename__ = "observations"

    id = Column(String, primary_key=True, index=True)
    patient_id = Column(String, ForeignKey("patients.id"))
    encounter_id = Column(String, ForeignKey("encounters.id"), nullable=True)
    status = Column(String)
    category = Column(String)
    code = Column(String)
    description = Column(String)
    value = Column(String)  # or Float if strictly numeric
    unit = Column(String)
    effective_date = Column(DateTime)
    issued_date = Column(DateTime)
```

➔ ***medications*** and ***imaging_studies*** table model



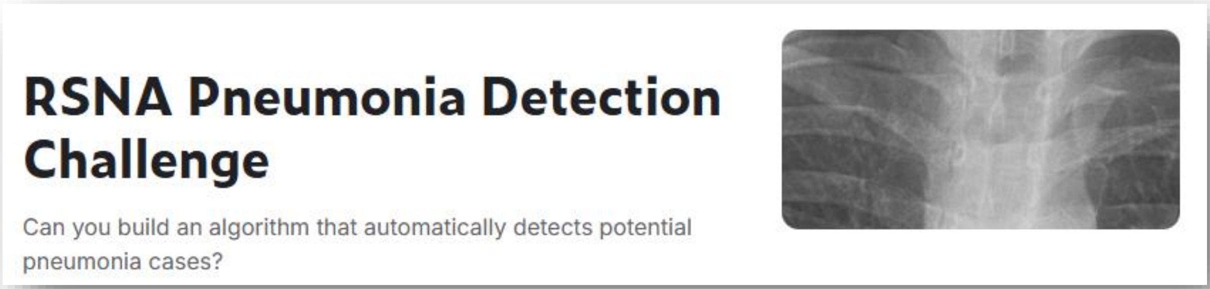
```
You, 11 hours ago | 1 author (You)
class Medication(Base):
    __tablename__ = "medications"

    id = Column(String, primary_key=True, index=True)
    patient_id = Column(String, ForeignKey("patients.id"))
    encounter_id = Column(String, ForeignKey("encounters.id"), nullable=True)
    medication_code = Column(String)
    medication_name = Column(String)
    status = Column(String)
    intent = Column(String)
    category = Column(String)
    authored_on = Column(DateTime)
    reason = Column(String)
```



You, 11 hours ago | 1 author (You)

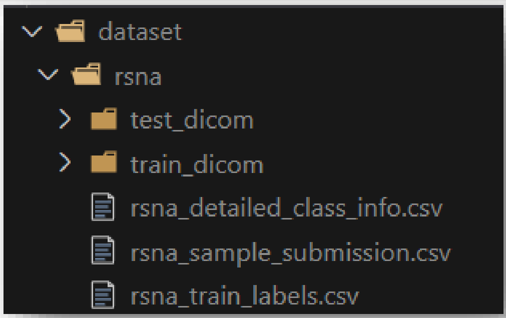
```
class ImagingStudy(Base):        You, 11 hours ago • added all tables + resourc
    __tablename__ = "imaging_studies"

    id = Column(String, primary_key=True, index=True)
    patient_id = Column(String, ForeignKey("patients.id"))
    encounter_id = Column(String, ForeignKey("encounters.id"), nullable=True)
    status = Column(String)
    started = Column(DateTime)
    procedure_code = Column(String)
    procedure_display = Column(String)
    modality_code = Column(String)
    modality_display = Column(String)
    body_site = Column(String)
    dicom_uid = Column(String)
```

5. 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.



RSNA Pneumonia Detection Challenge

Can you build an algorithm that automatically detects potential pneumonia cases?

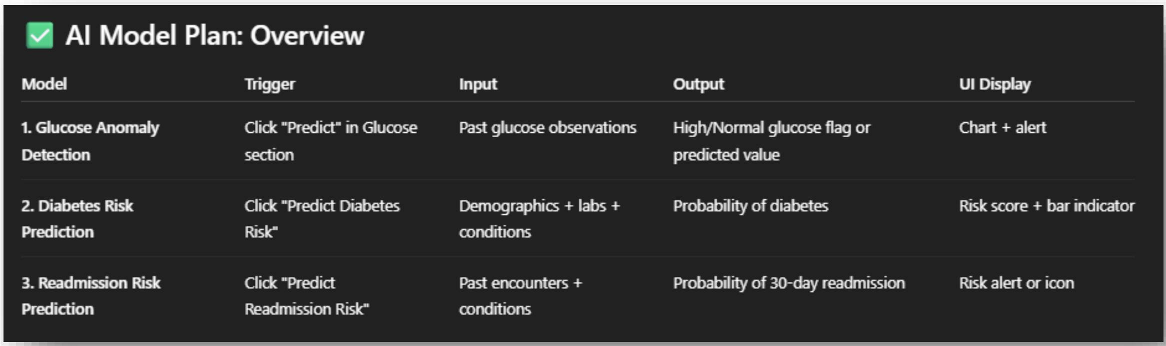


```
dataset
  rsna
    test_dicom
    train_dicom
    rsna_detailed_class_info.csv
    rsna_sample_submission.csv
    rsna_train_labels.csv
```

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

6. 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.



✅ AI Model Plan: Overview

Model	Trigger	Input	Output	UI Display
1. Glucose Anomaly Detection	Click "Predict" in Glucose section	Past glucose observations	High/Normal glucose flag or predicted value	Chart + alert
2. Diabetes Risk Prediction	Click "Predict Diabetes Risk"	Demographics + labs + conditions	Probability of diabetes	Risk score + bar indicator
3. Readmission Risk Prediction	Click "Predict Readmission Risk"	Past encounters + conditions	Probability of 30-day readmission	Risk alert or icon

7. 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
```

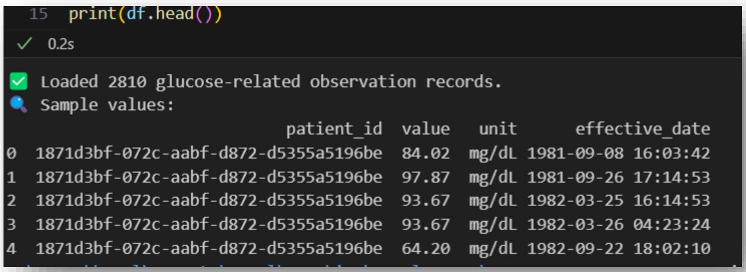
8. Okay, so let start with the ***Glucose Anomaly Detection Model*** in our project.

➔ 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.

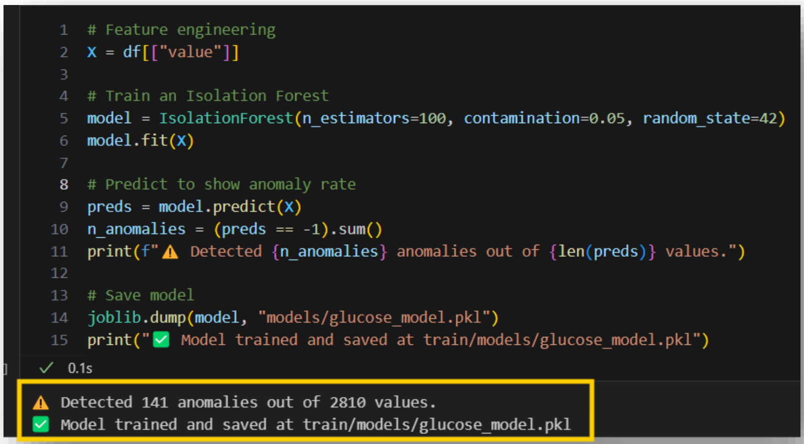
Bottom line: For datasets with numerical inputs, no labels, and where we want to flag rare/odd values, Isolation Forest is one of the best plug-and-play options.

➔ Output of fetched values from the Observation Table:



```
print(df.head())
✓ 0.2s

✅ Loaded 2810 glucose-related observation records.
🔍 Sample values:
                             patient_id  value   unit       effective_date
0  1871d3bf-072c-aabf-d872-d5355a5196be  84.02  mg/dL 1981-09-08 16:03:42
1  1871d3bf-072c-aabf-d872-d5355a5196be  97.87  mg/dL 1981-09-26 17:14:53
2  1871d3bf-072c-aabf-d872-d5355a5196be  93.67  mg/dL 1982-03-25 16:14:53
3  1871d3bf-072c-aabf-d872-d5355a5196be  93.67  mg/dL 1982-03-26 04:23:24
4  1871d3bf-072c-aabf-d872-d5355a5196be  64.20  mg/dL 1982-09-22 18:02:10
```

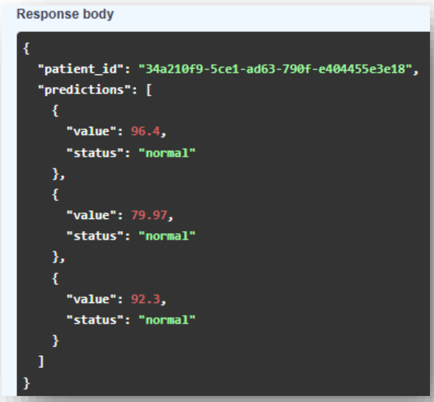
➔ Output of our model finding Anomalies:



```python
# Feature engineering
X = df[["value"]]

# Train an Isolation Forest
model = IsolationForest(n_estimators=100, contamination=0.05, random_state=42)
model.fit(X)

# Predict to show anomaly rate
preds = model.predict(X)
n_anomalies = (preds == -1).sum()
print(f"⚠ Detected {n_anomalies} anomalies out of {len(preds)} values.")

# Save model
joblib.dump(model, "models/glucose_model.pkl")
print("✅ Model trained and saved at train/models/glucose_model.pkl")
```

```
✓ 0.1s

⚠ Detected 141 anomalies out of 2810 values.
✅ Model trained and saved at train/models/glucose_model.pkl
```

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



Response body

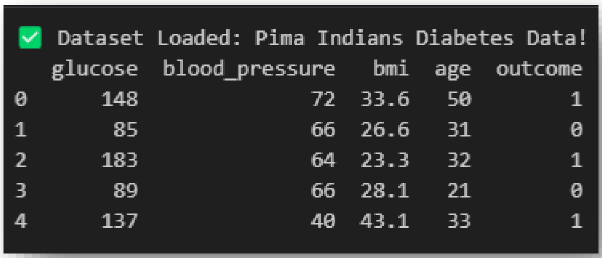
```json
{
  "patient_id": "34a210f9-5ce1-ad63-790f-e404455e3e18",
  "predictions": [
    {
      "value": 96.4,
      "status": "normal"
    },
    {
      "value": 79.97,
      "status": "normal"
    },
    {
      "value": 92.3,
      "status": "normal"
    }
  ]
}
```

➔ 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.

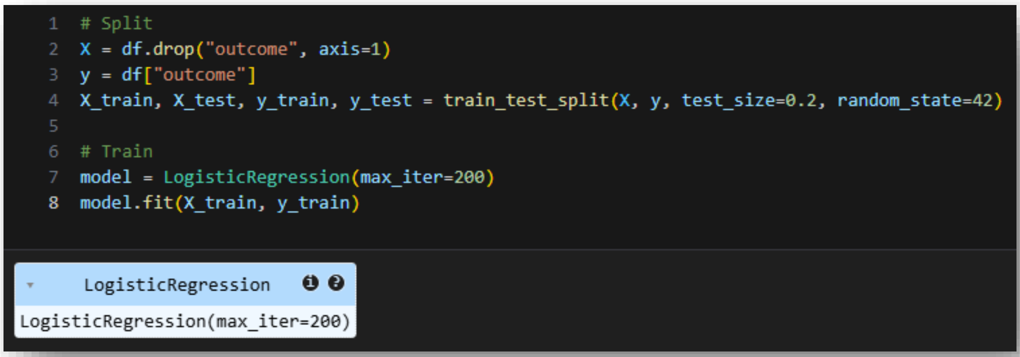
9. Moving on to our next ML model in this project for **Diabetes Prediction Model.**

➔ We used a ***supervised machine learning model*** to predict diabetes risk based on observations (like glucose, BMI, blood pressure, and age). We trained this on publicly available data (***Pima Indians dataset***) and apply it to patients from our own FHIR dataset when we extract compatible features. Note: The Pima Indians Dataset takes 8-9 parameters to train, but for this project demonstration we have taken only 4, which were the only ones available in our FHIR dataset.



```
✅ Dataset Loaded: Pima Indians Diabetes Data!
   glucose  blood_pressure   bmi  age  outcome
0      148              72  33.6   50        1
1       85              66  26.6   31        0
2      183              64  23.3   32        1
3       89              66  28.1   21        0
4      137              40  43.1   33        1
```

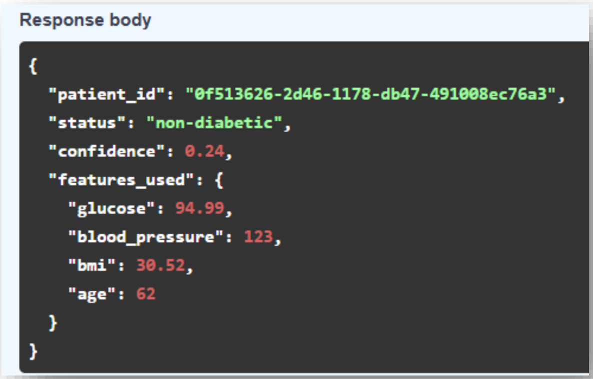
➔ We chose and trained a Logistic Regression model on the public dataset, which we will be using on our FHIR dataset values.



```
# Split
X = df.drop("outcome", axis=1)
y = df["outcome"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=200)
```

➔ Why chose Logistic Regression Model to predict Diabetes?
So, this model is particularly useful when you need to understand the relationship between multiple independent variables (in our case BMI, Systolic Blood Pressure, Age, etc) and the probability of a binary outcome (Yes or No), or when you need to classify data into two categories.



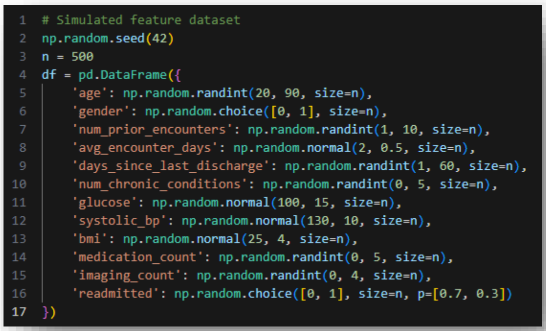
Response body

```
{
  "patient_id": "0f513626-2d46-1178-db47-491008ec76a3",
  "status": "non-diabetic",
  "confidence": 0.24,
  "features_used": {
    "glucose": 94.99,
    "blood_pressure": 123,
    "bmi": 30.52,
    "age": 62
  }
}
```

10. Moving on to our ML model in this project for FHIR dataset – ***Patient Readmission Risk Prediction***

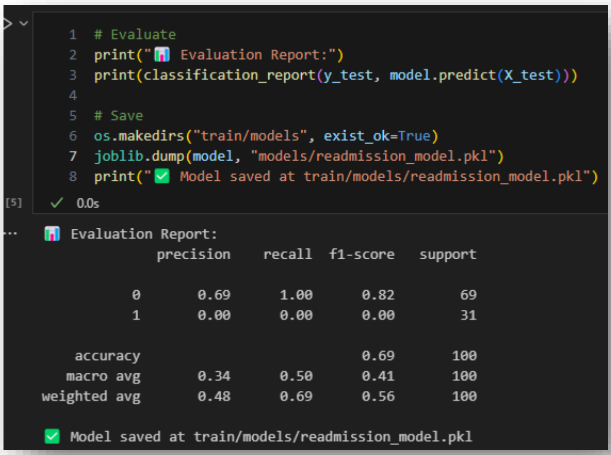
➔ For this prediction we will be using Random Forrest Classifier - used for patient readmission risk prediction because it is effective at handling complex, non-linear relationships in data, are less prone to overfitting, and can provide insights into feature importance, all of which are valuable in predicting readmission risk.

➔ Creating our own dataset:



```python
# Simulated feature dataset
np.random.seed(42)
n = 500
df = pd.DataFrame({
    'age': np.random.randint(20, 90, size=n),
    'gender': np.random.choice([0, 1], size=n),
    'num_prior_encounters': np.random.randint(1, 10, size=n),
    'avg_encounter_days': np.random.normal(2, 0.5, size=n),
    'days_since_last_discharge': np.random.randint(1, 60, size=n),
    'num_chronic_conditions': np.random.randint(0, 5, size=n),
    'glucose': np.random.normal(100, 15, size=n),
    'systolic_bp': np.random.normal(130, 10, size=n),
    'bmi': np.random.normal(25, 4, size=n),
    'medication_count': np.random.randint(0, 5, size=n),
    'imaging_count': np.random.randint(0, 4, size=n),
    'readmitted': np.random.choice([0, 1], size=n, p=[0.7, 0.3])
})
```

➔ Model output:

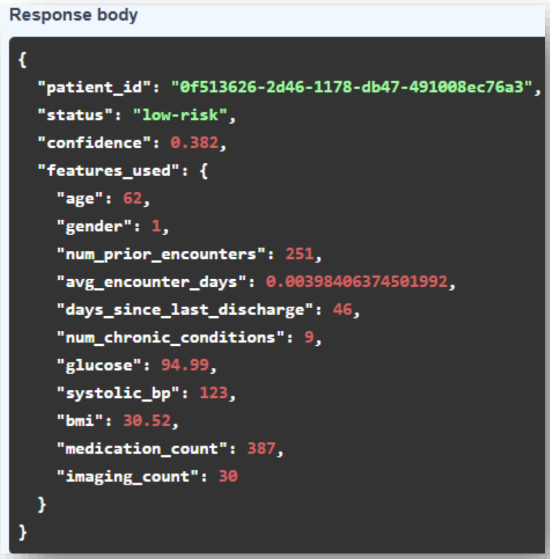


```python
# Evaluate
print("📊 Evaluation Report:")
print(classification_report(y_test, model.predict(X_test)))

# Save
os.makedirs("train/models", exist_ok=True)
joblib.dump(model, "models/readmission_model.pkl")
print("✅ Model saved at train/models/readmission_model.pkl")
```

[5] ✓ 0.0s

```
📊 Evaluation Report:
              precision    recall  f1-score   support

           0       0.69      1.00      0.82        69
           1       0.00      0.00      0.00        31

    accuracy                           0.69       100
   macro avg       0.34      0.50      0.41       100
weighted avg       0.48      0.69      0.56       100

✅ Model saved at train/models/readmission_model.pkl
```

➔ API outcome:



Response body

```json
{
  "patient_id": "0f513626-2d46-1178-db47-491008ec76a3",
  "status": "low-risk",
  "confidence": 0.382,
  "features_used": {
    "age": 62,
    "gender": 1,
    "num_prior_encounters": 251,
    "avg_encounter_days": 0.00398406374501992,
    "days_since_last_discharge": 46,
    "num_chronic_conditions": 9,
    "glucose": 94.99,
    "systolic_bp": 123,
    "bmi": 30.52,
    "medication_count": 387,
    "imaging_count": 30
  }
}
```

11. Designing our ***Frontend*** for our FHIR Data and ***testing*** all our ***API endpoints*** from ***backend*** and displaying them aesthetically.

➔ So, we create a react application along with SCSS (Sassy CSS) in our Frontend folder.

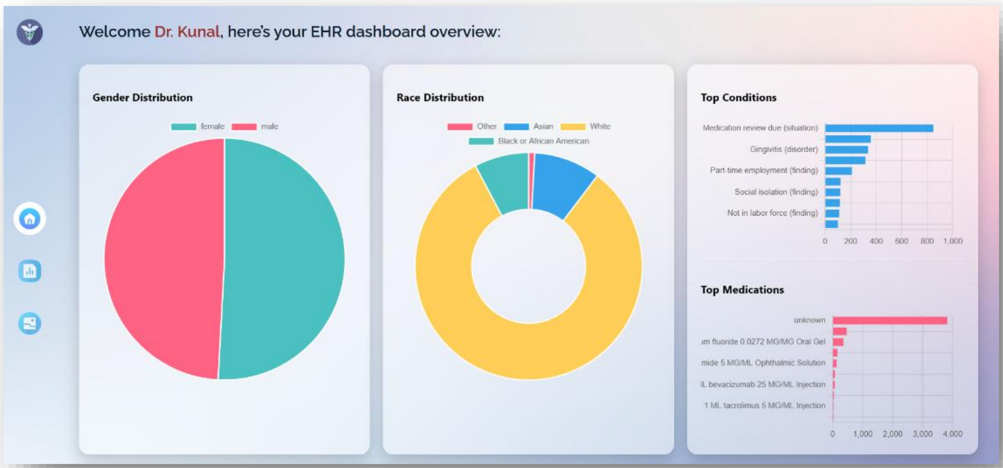
-- npx create-react-app frontend
-- npm install axios react-router-dom
-- npm install sass --save-dev

We will also be displaying multiple charts. For that we will be using:

-- npm install chart.js react-chartjs-2

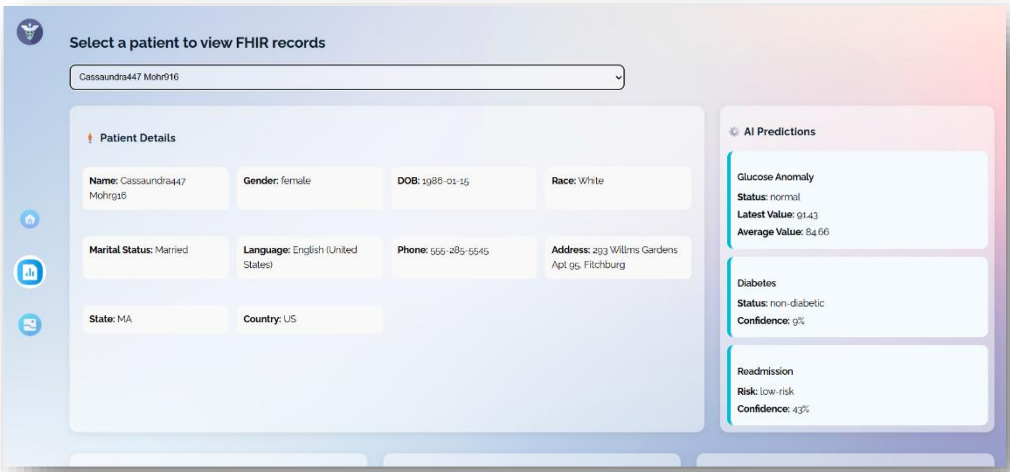
➔ We designed a simple structure for our UI.
-> Home Page – which displays all the global data from our FHIR dataset
-> FHIR – which lets user select a patient report and get all the details
-> DICOM – which lets user upload Pneumonia images and give predictions.

➔ Home Page:


Welcome Dr. Kunal, here's your EHR dashboard overview:

Gender Distribution
female
male
Race Distribution
Other
Asian
White
Black or African American
Top Conditions
Medication review due (situation)
Gingivitis (disorder)
Part-time employment (finding)
Social isolation (finding)
Not in labor force (finding)
0
200
400
600
800
1,000
Top Medications
unknown
ım fluoride 0.0272 MG/MG Oral Gel
mide 5 MG/ML Ophthalmic Solution
IL bevacizumab 25 MG/ML Injection
1 ML tacrolimus 5 MG/ML Injection
0
1,000
2,000
3,000
4,000

➔ FHIR Page:


Select a patient to view FHIR records

Cassaundra447 Mohr916

Patient Details

Name: Cassaundra447 Mohr916
Gender: female
DOB: 1986-01-15
Race: White
Marital Status: Married
Language: English (United States)
Phone: 555-285-5545
Address: 293 Willms Gardens Apt 95. Fitchburg
State: MA
Country: US

AI Predictions

Glucose Anomaly
Status: normal
Latest Value: 91.43
Average Value: 84.66

Diabetes
Status: non-diabetic
Confidence: 9%

Readmission
Risk: low-risk
Confidence: 43%

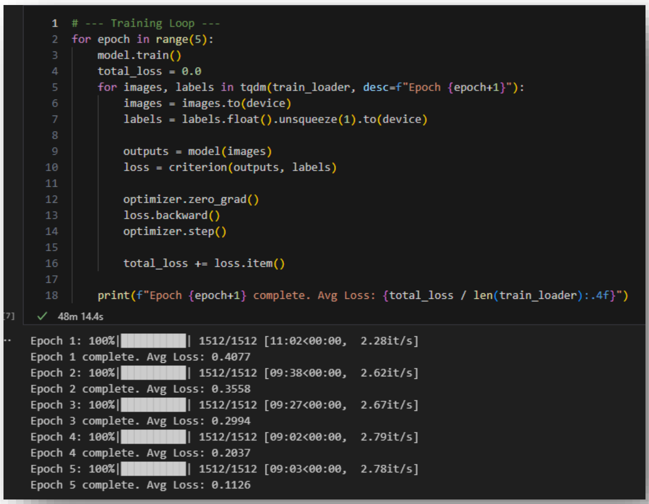
12. Lastly, we created our final ML model for our DICOM dataset. We used a ***RestNet18 CNN model*** to predict ***Pneumonia*** trained on our ***RSNA*** Pneumonia Dataset.

➔ Why we chose this model?

- Proven Performance on Medical Images: ResNet architectures are widely used in medical imaging tasks and have shown strong accuracy in detecting conditions like pneumonia from X-rays.
- Handles Deep Representations: ResNet18 uses residual connections, which help train deeper networks without vanishing gradients—ideal for extracting complex features in DICOM images.
- Lightweight and Fast: Compared to deeper ResNets (e.g., ResNet50), ResNet18 offers a good balance between speed and performance, making it suitable for real-time inference via APIs.
- Pretrained Weights: It comes with ImageNet pretrained weights, helping the model converge faster and generalize better, even with limited medical data.

In short: it's accurate, efficient, and well-suited for high-resolution grayscale DICOM data.

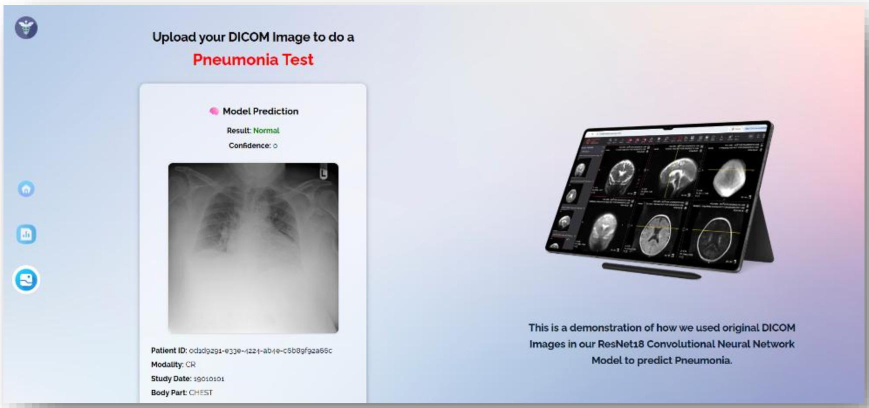
➔ Training our model on 5 Epochs (Roughly 50 Minutes of training):



```
# --- Training Loop ---
for epoch in range(5):
    model.train()
    total_loss = 0.0
    for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}"):
        images = images.to(device)
        labels = labels.float().unsqueeze(1).to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch {epoch+1} complete. Avg Loss: {total_loss / len(train_loader):.4f}")
```

```
[7]  ✓ 48m 14.4s
Epoch 1: 100%|██████████| 1512/1512 [11:02<00:00,  2.28it/s]
Epoch 1 complete. Avg Loss: 0.4077
Epoch 2: 100%|██████████| 1512/1512 [09:38<00:00,  2.62it/s]
Epoch 2 complete. Avg Loss: 0.3558
Epoch 3: 100%|██████████| 1512/1512 [09:27<00:00,  2.67it/s]
Epoch 3 complete. Avg Loss: 0.2994
Epoch 4: 100%|██████████| 1512/1512 [09:02<00:00,  2.79it/s]
Epoch 4 complete. Avg Loss: 0.2037
Epoch 5: 100%|██████████| 1512/1512 [09:03<00:00,  2.78it/s]
Epoch 5 complete. Avg Loss: 0.1126
```

➔ DICOM Page UI:



Upload your DICOM Image to do a
Pneumonia Test

Model Prediction
Result: Normal
Confidence: 0

Patient ID: 0d1d9291-e33e-4224-ab4e-c5b89f92a66c
Modality: CR
Study Date: 19010101
Body Part: CHEST

This is a demonstration of how we used original DICOM Images in our ResNet18 Convolutional Neural Network Model to predict Pneumonia.