

# **JTP Final Project**

## **Med Bud**

Author: Naman Singh

Date: May 2025

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Overview</b>	<b>1</b>
<b>3</b>	<b>System Architecture</b>	<b>2</b>
3.1	Architecture Diagram . . . . .	2
3.2	Flowchart . . . . .	3
<b>4</b>	<b>Frontend UI</b>	<b>4</b>
4.1	About Page . . . . .	4
4.2	Prediction Tab . . . . .	5
4.3	Profile Tab . . . . .	6
4.4	History Tab . . . . .	7
<b>5</b>	<b>Backend and Machine Learning</b>	<b>8</b>
5.1	Model Implementation . . . . .	8
<b>6</b>	<b>API Endpoints</b>	<b>9</b>
<b>7</b>	<b>Authentication and Security</b>	<b>9</b>
<b>8</b>	<b>Database Schema (Supabase)</b>	<b>10</b>
<b>9</b>	<b>Important !!!</b>	<b>11</b>
<b>10</b>	<b>Deployment on AWS EC2</b>	<b>12</b>
<b>11</b>	<b>Conclusion</b>	<b>14</b>

# 1 Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

## 2 Overview

The Medicine Recommendation System, **Med Bud**, is an intelligent healthcare application combining machine learning with a modern frontend to predict diseases and offer health recommendations.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

## 3 System Architecture

### 3.1 Architecture Diagram

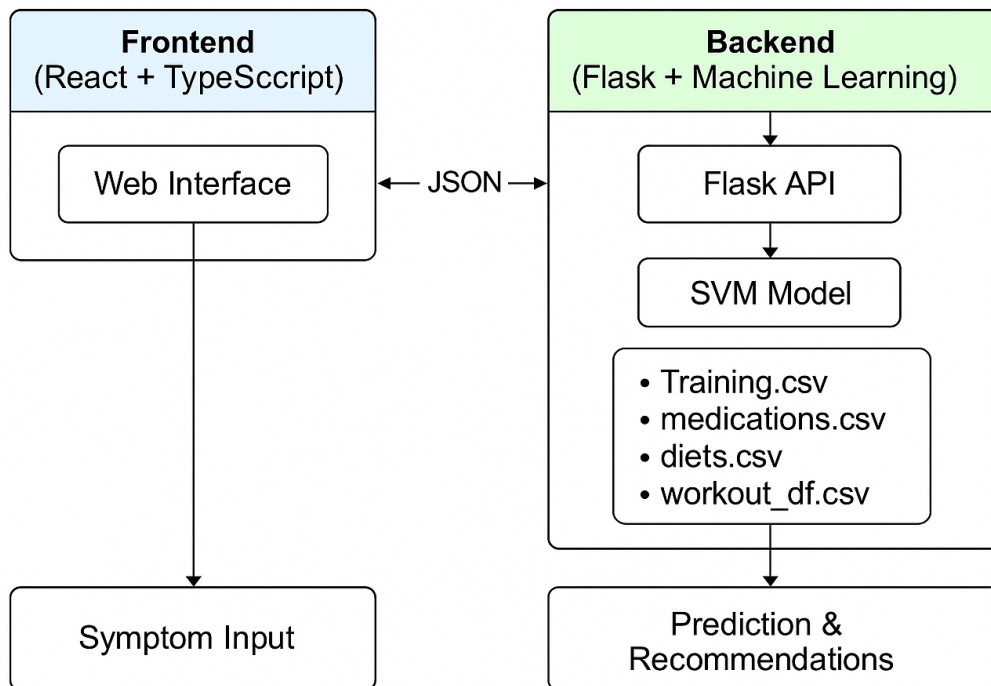


Figure 1: System Architecture

#### Description

The architecture of **Med Bud** consists of a **React + TypeScript** frontend and a **Flask-based** backend. Users input symptoms through a web interface, which sends data via **JSON** to the backend. The backend uses a trained **SVM model** and datasets (**Training.csv**, **medications.csv**, **diets.csv**, **workout\_df.csv**) to predict diseases and generate personalized recommendations. The results are returned to the frontend and displayed to the user.

## 3.2 Flowchart

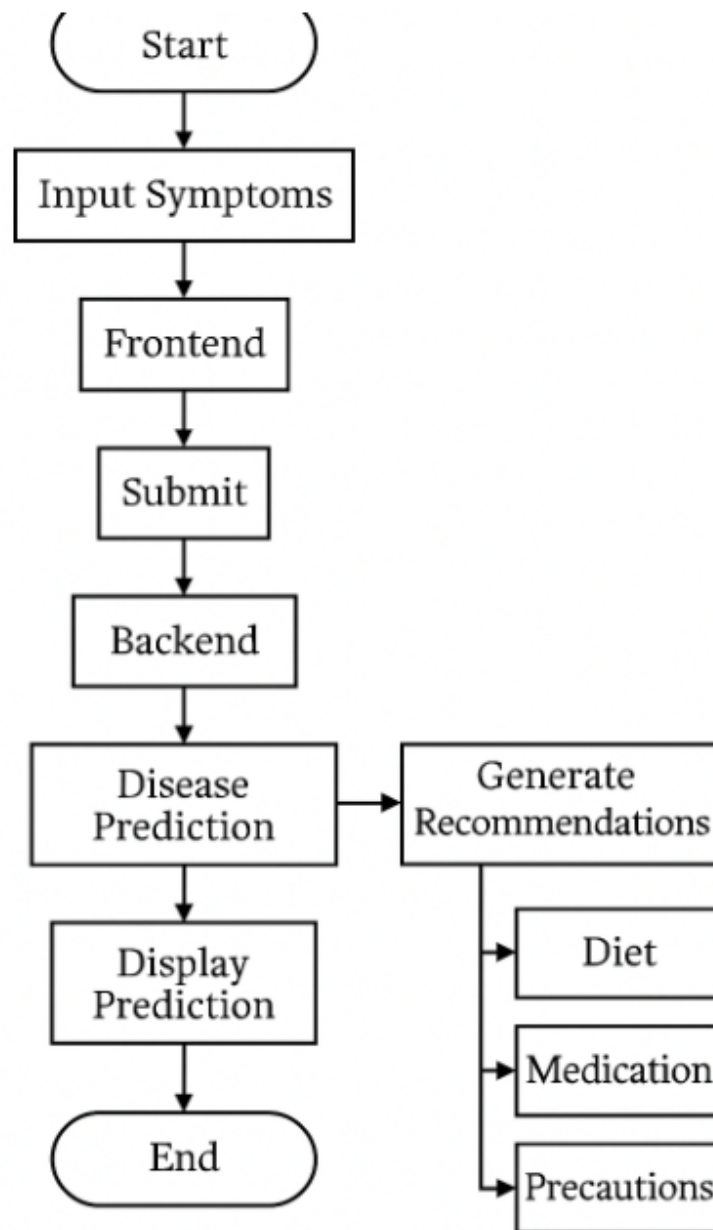


Figure 2: System Flowchart

### Description

The flowchart illustrates the step-by-step process of the **Med Bud** application. It begins with the user inputting symptoms through the frontend interface. Upon submission, the data is sent to the backend, where the system processes it to predict the disease. The predicted result is displayed, and simultaneously, the system generates personalized recommendations. These include tailored advice on **diet**, **medication**, and **precautions**, helping users take informed next steps.

## 4 Frontend UI

### 4.1 About Page

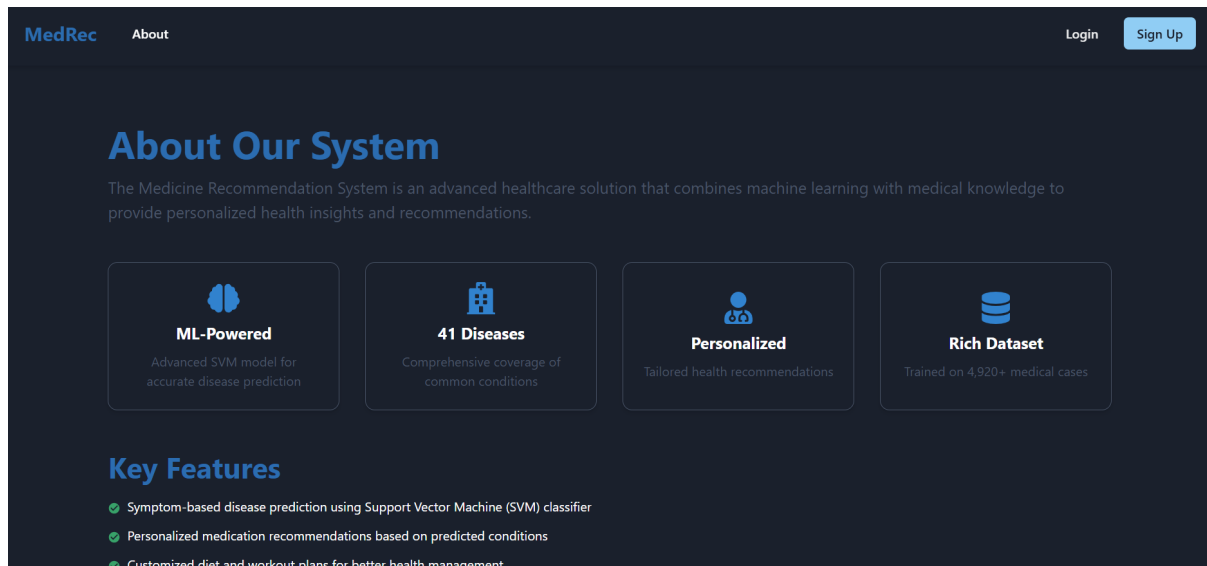


Figure 3: About Page UI

#### Description

The **About Page** introduces the core purpose and strengths of the **Med Bud** system. As an AI-driven healthcare platform, it merges **machine learning** with expert medical insights to deliver personalized recommendations based on user symptoms. This section highlights four key features: a robust **SVM model** for disease prediction, support for **41 diseases**, highly **personalized** outputs, and access to a **rich dataset** with over 4,920 real-world cases. Additionally, it emphasizes **key features** such as symptom-based prediction, customized medication plans, and targeted diet and workout suggestions.

From a development perspective, modifications to this page can be made in the frontend codebase under the file path: `src/pages/About.tsx`. This file contains the JSX layout and logic for the About screen. To enhance the content, developers can insert additional descriptive elements .

These updates help communicate the system's mission and technical foundation more clearly to end users.

## 4.2 Prediction Tab

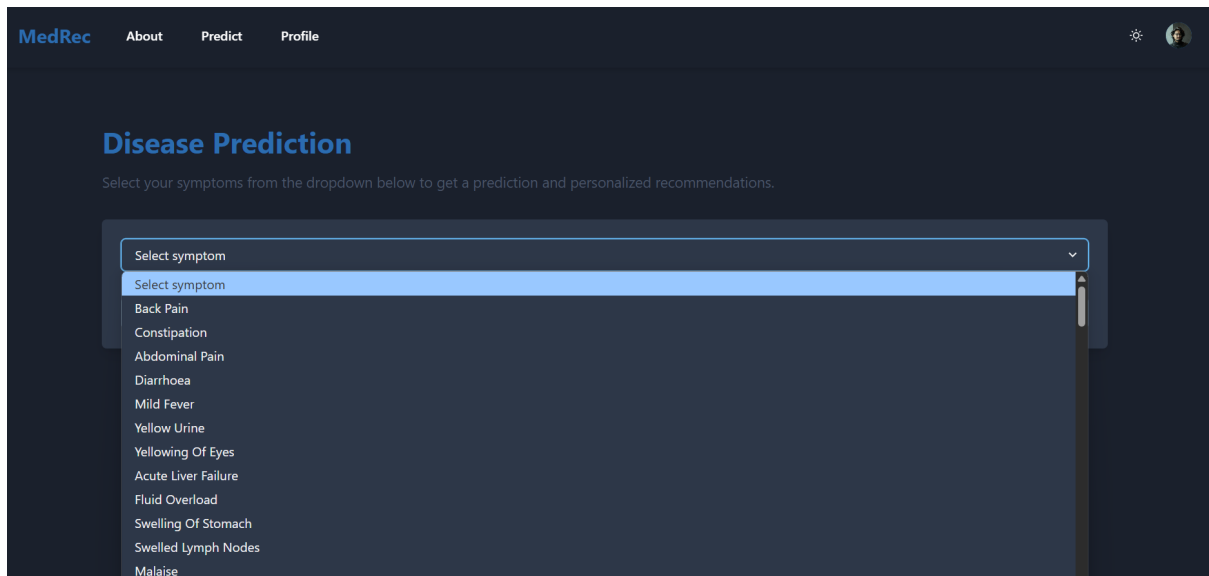


Figure 4: Prediction Tab UI

### Description

The **Disease Prediction** page provides an intuitive interface for users to select their symptoms and receive AI-driven diagnostic insights. A searchable dropdown allows users to choose from a comprehensive list of symptoms. Once selected, the symptoms are processed through the system's backend, which utilizes a trained **Support Vector Machine (SVM)** model to predict the most likely disease. The page is designed for simplicity and speed, offering a seamless experience on both desktop and mobile devices.

To modify the logic or layout of this page in the codebase, navigate to `src/pages/Prediction.tsx`. This component contains the core logic for rendering the dropdown, capturing user input, and triggering the prediction API. Developers can enhance this feature by customizing the dropdown behavior or improving UI responsiveness to user selections.

## 4.3 Profile Tab

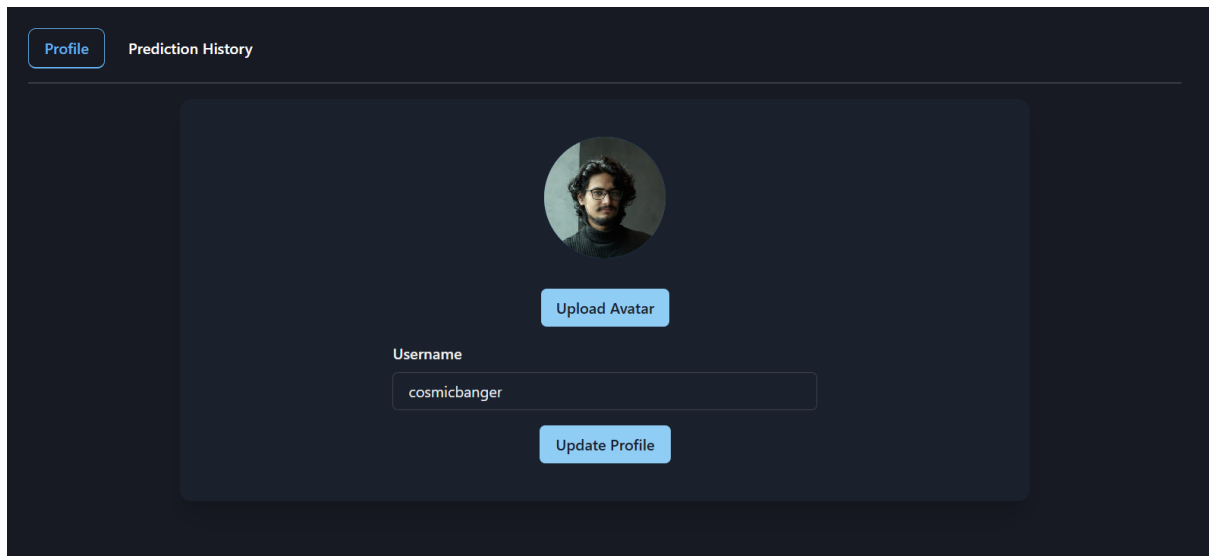


Figure 5: Profile Tab UI

### Description

The **Profile Tab** allows users to view and update their personal profile within the Med Bud system. Users can upload a custom avatar and modify their **username**, offering a personalized experience. Once updated, the new profile details are securely stored using **Supabase**, which acts as the backend-as-a-service for authentication and database management. The updates are reflected in real-time, ensuring a responsive and consistent user experience.

From a development standpoint, this feature is typically implemented in `src/pages/Profile.tsx`, where form input states are managed and API calls to Supabase are made. Developers can further enhance this functionality by adding input validation, profile deletion, or additional fields like bio or contact details.



## 4.4 History Tab

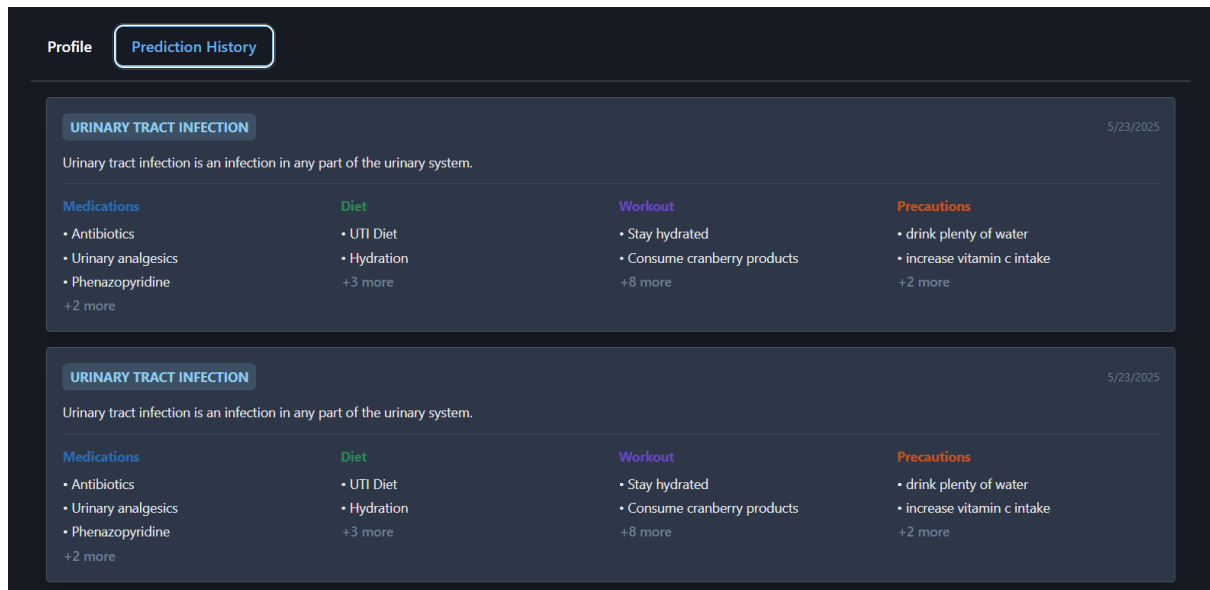


Figure 6: History Tab UI

### Description

The **Prediction History** tab allows users to review their past diagnosis results, including all generated health recommendations. Each card displays a predicted disease (e.g., Urinary Tract Infection), its brief description, and categorized suggestions for **Medications**, **Diet**, **Workout**, and **Precautions**. Additional items in each category are accessible via expandable sections, providing a complete view of the recommendations.

Each prediction record is stored in a **Supabase** backend as a structured object containing the disease name, timestamp, user ID, and arrays for medications, diets, workouts, and precautions. When a prediction is made, this data is saved in a dedicated Supabase table (e.g., `prediction_history`) using client-side API calls, typically located in `src/services/history.ts` or similar. Upon loading the history page, these entries are fetched and rendered dynamically in reverse chronological order, giving users a personalized medical history they can reference over time.

This functionality not only improves user experience by making past health data easily accessible, but also opens the door for future analytics and trend tracking per user.

## 5 Backend and Machine Learning

### Description

The backend of the Med Bud system is built using the **Flask** framework, designed to serve as a RESTful API that interfaces with a trained machine learning model. It handles HTTP requests from the frontend, processes symptom data, performs disease prediction, and returns a structured JSON response.

The application loads a **Support Vector Machine (SVM)** model trained on 132 binary symptom features. Upon receiving a list of symptoms via the `/predict` endpoint, it constructs a one-hot encoded input vector, runs the model inference, and predicts the most likely disease.

Additionally, the backend pulls related contextual information such as a **disease description**, **precautions**, **medications**, **diet plans**, and **workout routines** from preloaded CSV files. These files are parsed using **pandas** and the data is matched based on the predicted disease.

All interactions are CORS-enabled to support cross-origin requests from the React frontend, and the backend is designed to gracefully handle input validation, missing data, and internal errors. The server runs on port 5000 and listens for requests using **Flask's debug mode** during development.

### 5.1 Model Implementation

#### Description

The disease prediction engine of the Med Bud system is driven by a supervised machine learning approach. Multiple classification models were evaluated, including **Support Vector Machine (SVC)**, **Random Forest**, **Gradient Boosting**, **K-Nearest Neighbors (KNN)**, and **Multinomial Naive Bayes**. Each model was trained using a dataset composed of 132 binary features, where each feature represents the presence or absence of a specific symptom.

The dataset was split into training and testing sets using `train_test_split()` from **scikit-learn**. During training, each model learned to map symptom combinations to one of 41 disease classes. The performance of each model was assessed using two key metrics: **accuracy score** and the **confusion matrix**. Accuracy provided an overall measure of correct predictions, while the confusion matrix offered insights into specific misclassification patterns.

After comparative evaluation, the **SVC with a linear kernel** emerged as the most reliable model in terms of consistent accuracy and interpretability. This model was then serialized using **pickle** and integrated into the Flask backend. At inference time, the API accepts a list of symptoms, converts them into a one-hot encoded vector aligned with the trained features, and passes it to the model for disease prediction.

To enhance the prediction context, the system retrieves a comprehensive disease description along with personalized recommendations—medications, dietary advice, workout suggestions, and precautions—using helper functions and structured CSV datasets. This modular and test-driven implementation ensures both robustness and extensibility for future upgrades or model replacements.

## 6 API Endpoints

- **Endpoint:** /predict
- **Methods:** GET, POST
- **Input:** Comma-separated symptoms
- **Output:** JSON with disease prediction and recommendations

## 7 Authentication and Security

User authentication is handled via **Supabase**, which provides secure sign-up and login flows. All authenticated routes, such as profile updates and prediction history retrieval, are protected using JWT tokens issued by Supabase. Input validation is performed both on the client side and the backend to prevent malformed requests and ensure data integrity. CORS policies are properly configured to enable safe cross-origin communication between the React frontend and Flask backend.

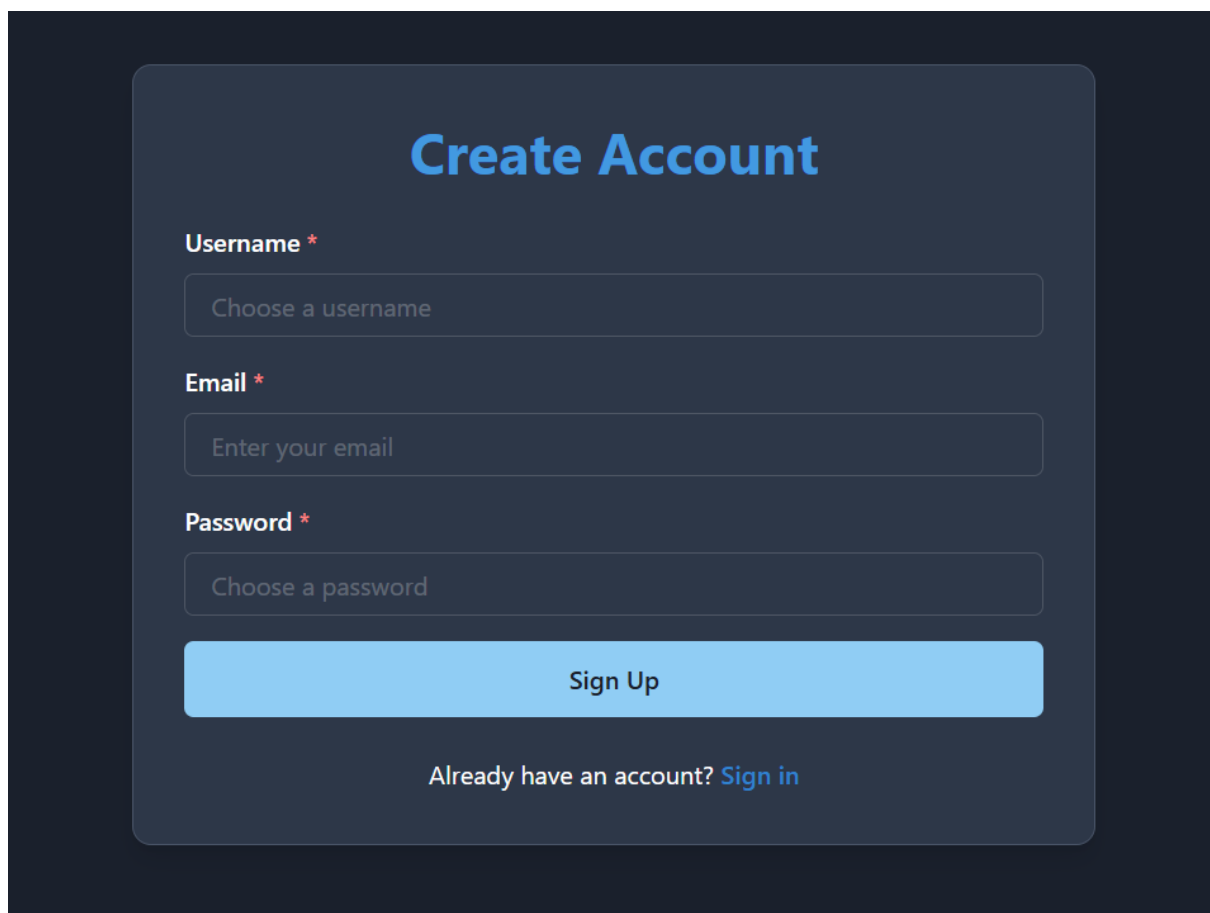
A screenshot of a 'Create Account' form. The form is centered on a dark blue background. It has a title 'Create Account' in a large, bold, light blue font. Below the title are three input fields, each with a label and a red asterisk: 'Username \*', 'Email \*', and 'Password \*'. The input fields have placeholder text: 'Choose a username', 'Enter your email', and 'Choose a password'. Below the input fields is a large, light blue button with the text 'Sign Up'. At the bottom of the form, there is a link that says 'Already have an account? Sign in'.

Figure 7: Authentication

## 8 Database Schema (Supabase)

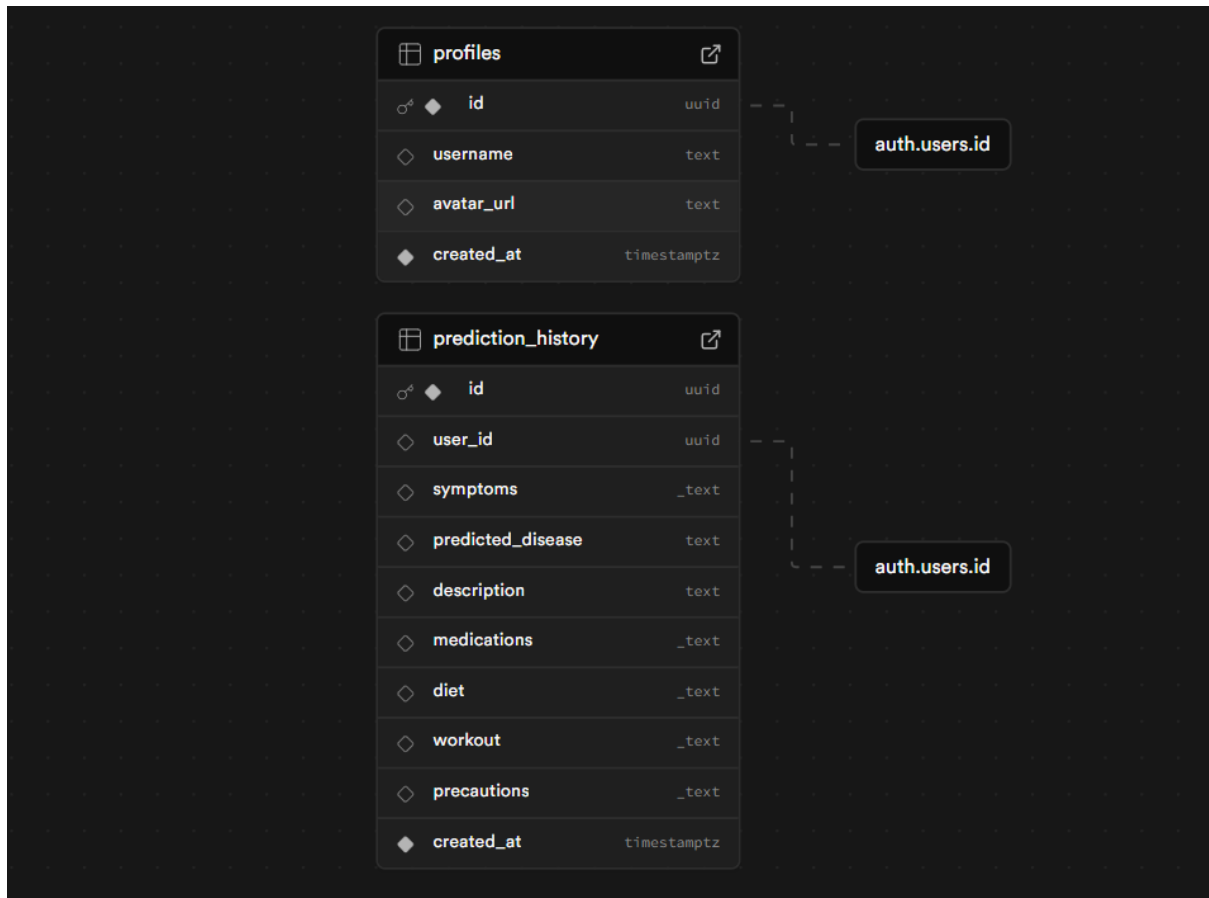


Figure 8: Database

The system leverages Supabase for storing user profiles and prediction history. The schema includes:

- **users:** Stores user ID, email, and profile metadata (e.g., username, avatar).
- **prediction\_history:** Stores timestamped records for each user containing predicted disease, list of symptoms, and structured fields for medications, diet, workout, and precautions.

All tables are indexed appropriately to support fast querying and real-time updates via Supabase client SDK.

## 9 Important !!!

To launch the full-stack Med Bud application using Docker, use the following command:

```
docker-compose up --build
```

The system will start:

- Frontend at <http://localhost:5173>
- Backend API at <http://localhost:5000>

To access the app over your network (e.g., mobile or other devices), follow these steps:

1. Run the command `ipconfig` in your terminal.
2. Copy the **IPv4 address** (e.g., `172.22.192.1`).
3. Open your browser and go to: `http://<your-ip>:5173` (e.g., `http://172.22.192.1:5173`).

### Why not use localhost for testing on other devices?

Hot reloading does not function properly when accessed via `localhost` from networked devices. Using the actual IPv4 address enables hot reload and allows real-time UI updates without restarting the dev server.

### Having trouble with continuous loading?

If the page keeps loading indefinitely, it's often due to caching issues. Be sure to:

- Clear your browser's cache and cookies.
- Perform a hard refresh using `Ctrl+F5` (Windows) or `Cmd+Shift+R` (Mac).

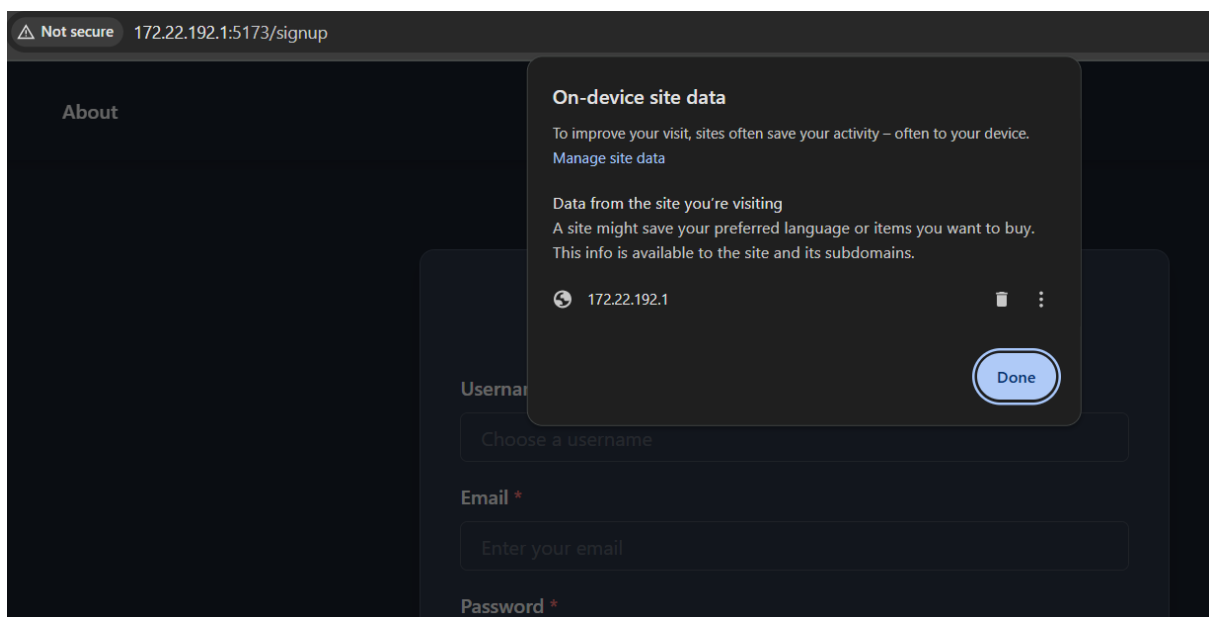


Figure 9: Remove cache

This ensures that stale cached assets or cookies do not interfere with updated builds or authentication sessions.

# 10 Deployment on AWS EC2

## Overview

To make the backend of **Med Bud** publicly accessible, an **AWS EC2** instance was used. This allows running the Flask server continuously and enables communication with the frontend hosted on Vercel. Below are the detailed steps followed to set up and deploy the backend service on EC2.

## Step-by-Step Deployment Guide

1. **Launched EC2 Instance:** An Amazon Linux 2 instance was launched using the AWS Console. A new key pair (`naman.pem`) was generated and downloaded securely.
2. **Configured Security Groups:** Inbound rules were updated to allow:
  - SSH (Port 22) – for terminal access
  - HTTP (Port 80) – optional for future web hosting
  - Custom TCP (Port 5000) – to expose the Flask server

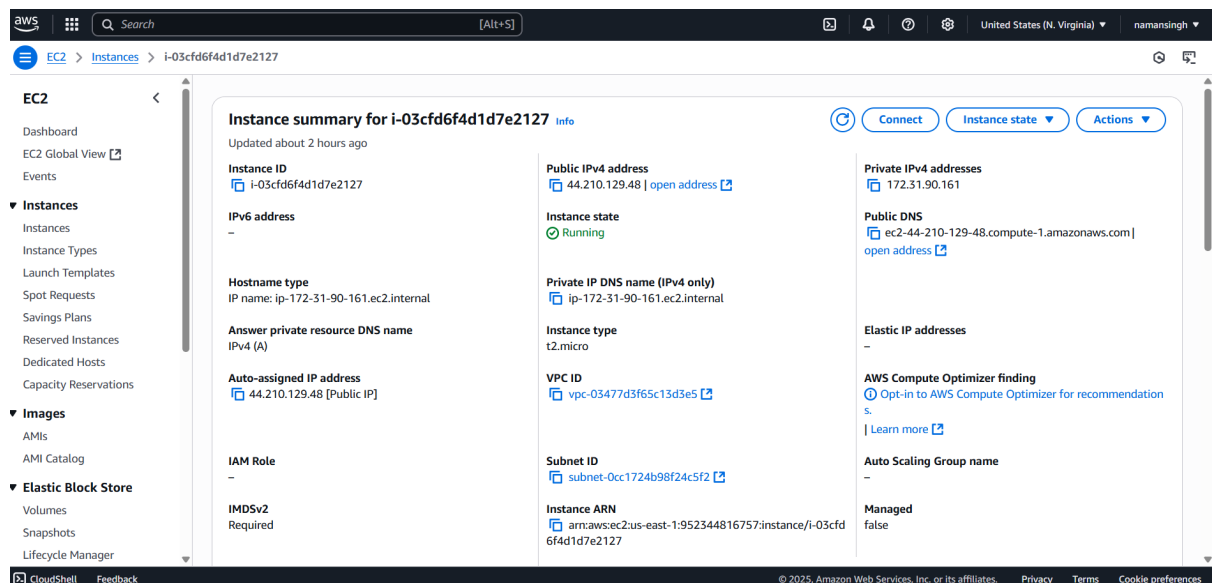


Figure 10: Security Group Configuration for EC2 Instance

This ensures that the Flask app can be accessed from the frontend deployed on Vercel.

3. **SSH into EC2:** Using the terminal:

```
ssh -i "naman.pem" ec2-user@<public-dns>
```

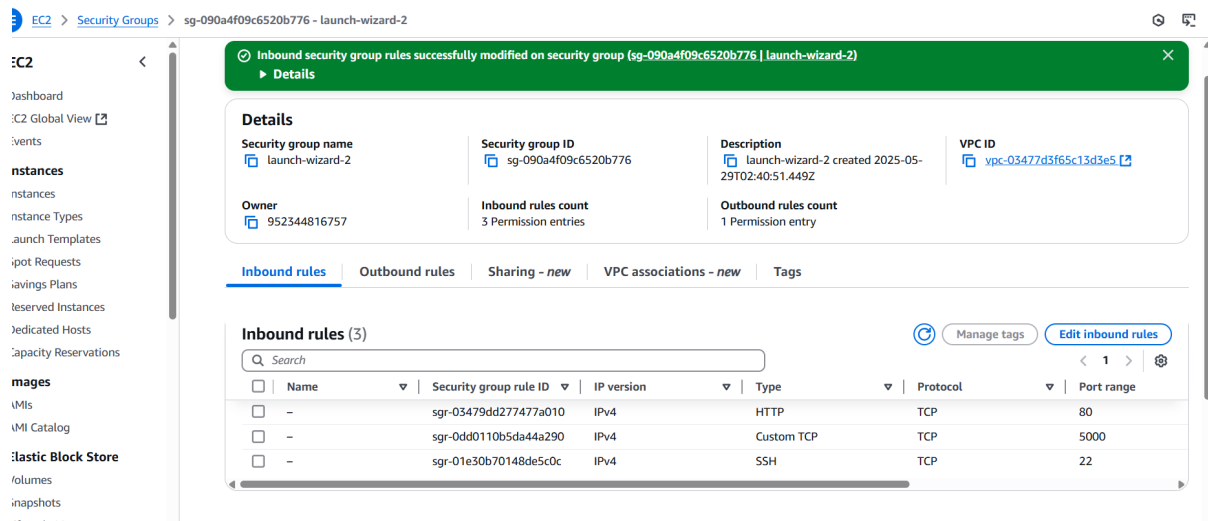


Figure 11: SSH Access to EC2 via PEM Key

Permissions for the PEM file were restricted using 'chmod 400'.

#### 4. Installed Dependencies:

```
sudo yum update -y
sudo yum install python3 git -y
```

#### 5. Cloned GitHub Repository:

```
git clone https://github.com/namansingh1314/MedBud.git
cd MedBud/Backend
```

#### 6. Started Flask Server: Activated a virtual environment and ran the app:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python app.py
```

The server now runs on port 5000 and is accessible from any IP.

#### 7. Connected to Vercel Frontend: On the Vercel dashboard, environment variable VITE\_API\_BASE\_URL was updated to:

```
http://<EC2-Public-IP>:5000
```

This ensures the live frontend communicates with the hosted backend.

## 11 Conclusion

Med Bud offers a practical and intelligent healthcare support system by combining machine learning with a responsive web interface. The backend effectively handles disease prediction using a trained SVM model, while the frontend ensures an intuitive user experience. With secure user authentication, structured data management via Supabase, and a modular architecture, the system is well-positioned for scalability.



# Bibliography and Resources

Below are the key resources used during the development and deployment of the Med Bud project:

- **GitHub Repository:**  
<https://github.com/namansingh1314/MedBud/tree/main>
- **EC2 Deployment Console:**  
AWS EC2 Console for Instance Access
- **YouTube Demonstration:**  
Demo Video on YouTube
- **Research Paper – Support Vector Machines:**  
Support Vector Machines: Theory and Applications
- **Official Documentation:**
  - Flask Documentation
  - Python Pickle Documentation
  - Docker Compose
  - Supabase Documentation
  - GitHub Actions