Heavens' Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

CSE 3100
Web-Based Application Project

HCV-Ai: Non-invasive Hepatitis C Virus Detection System

Submitted by

Kaif Ahmed Khan

Roll: 2103163

Submitted to

Md. Farhan Shakib
Lecturer

August 31, 2025

Contents

List of Figures

1 Introduction

The web application brings to life a machine learning framework developed for the non-invasive detection of Hepatitis C Virus (HCV), as outlined in our recent research. Designed with accessibility and accuracy in mind, the platform allows users—particularly patients and healthcare providers—to upload standard laboratory data and receive predictions about the possible stage of HCV infection: healthy, hepatitis, fibrosis, or cirrhosis.

The primary objective of the application is to make early and accurate HCV detection more accessible, especially in low-resource settings where traditional diagnostic methods are often invasive, costly, and difficult to access. By leveraging routine blood test results and state-of-the-art machine learning techniques—including synthetic data generation, feature selection, and explainable AI—this application aims to provide reliable, interpretable predictions that can assist in timely clinical decision-making.

This system addresses a critical gap in healthcare: the lack of affordable and scalable tools for multiclass liver disease classification. Whether you're a clinician, researcher, or concerned individual, this platform empowers you with actionable insights using only non-invasive lab data.

2 System Components

The system is composed of two main components: the front-end and the back-end, which work together to provide a seamless user experience and efficient data processing.

2.1 Frontend Components

The front-end of the system is built using Next.js, a React-based framework that enables server-side rendering and static site generation for improved performance. This component is responsible for the user interface, allowing patients and administrators to interact with the system. Users can register, log in, submit their medical data, and view their prediction results through a responsive and intuitive interface. Administrators, on the other hand, have access to dashboards where they can view predictions submitted by all users. The front-end communicates with the back-end through RESTful APIs, ensuring that user actions are efficiently processed and results are delivered in real time.

2.2 Backend Components

The back-end of the system is implemented using FastAPI[1] , a modern Python framework designed for high-performance APIs. This component handles authentication, authorization, and data management. It processes user-submitted medical data, forwards the input to the pre-trained machine learning model built with Scikit-Learn, and returns prediction results to the front-end. The back-end also

[1] `https://fastapi.tiangolo.com/`

manages the storage and retrieval of user data and prediction records using a relational database, implemented with SQLite for simplicity and reliability. Furthermore, Docker is used to containerize the application, ensuring consistency across development and deployment environments.

Together, the front-end and back-end form a robust system that supports medical prediction tasks, ensuring usability, performance, and scalability for both patients and administrators.

2.3 Dataflow Diagram

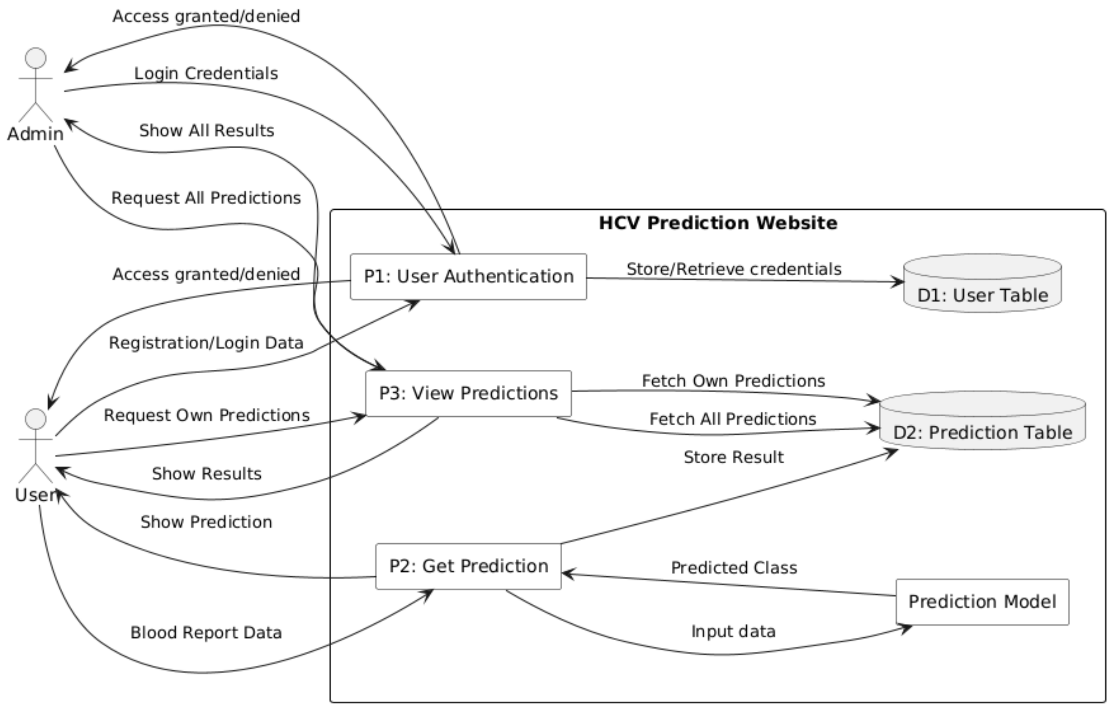


Figure 1: Dataflow diagram of the website

2.4 Sequence Diagram

2.5 Key Features & Page Screenshots

User Management

- User registration and login system with SQLite database authentication.
- Role-based access control: Patients (users) and Admins.
- Secure storage of user information.

Prediction System

- Input form for blood report data (patient submits laboratory values).
- Pretrained machine learning model processes the input.

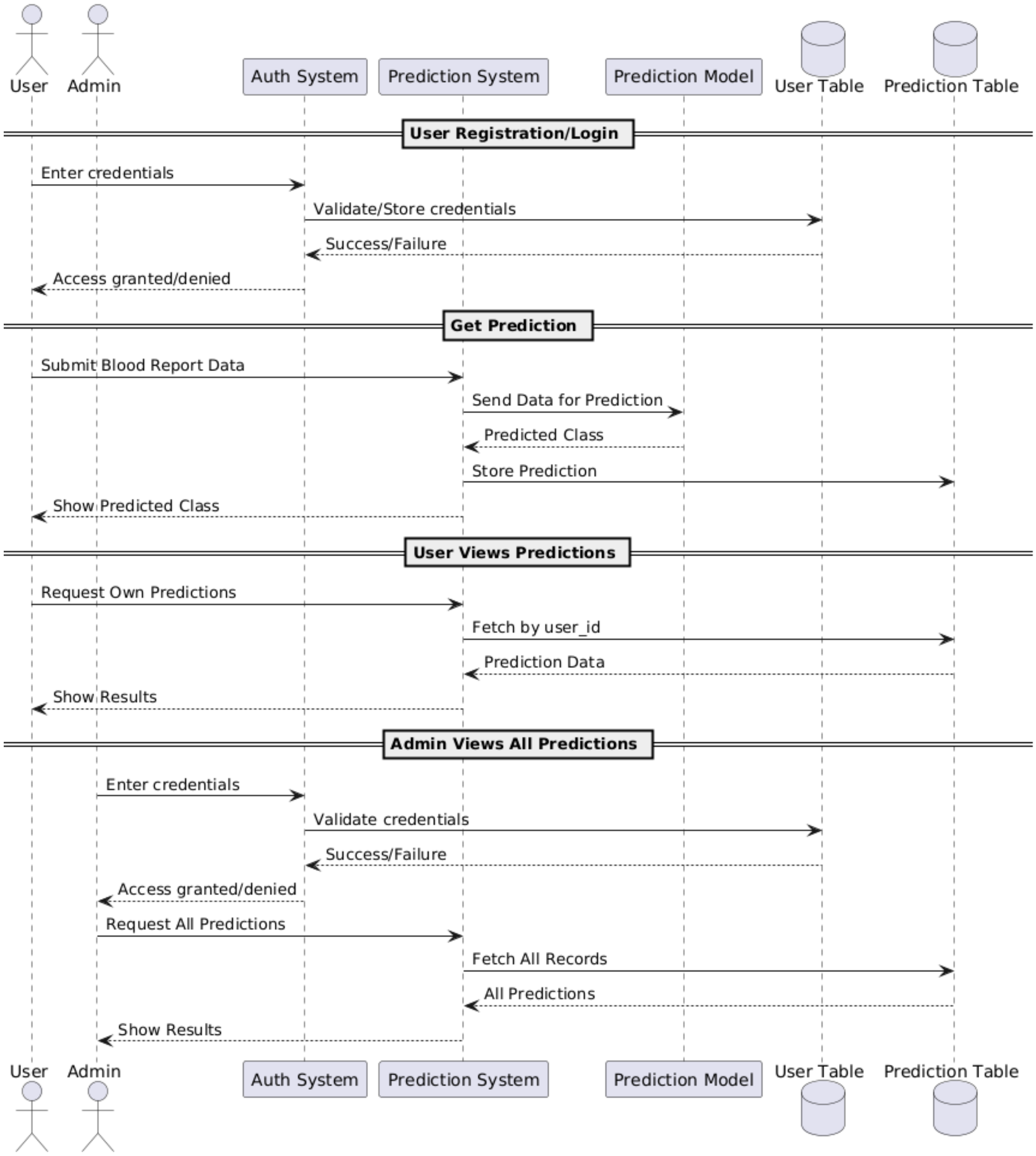
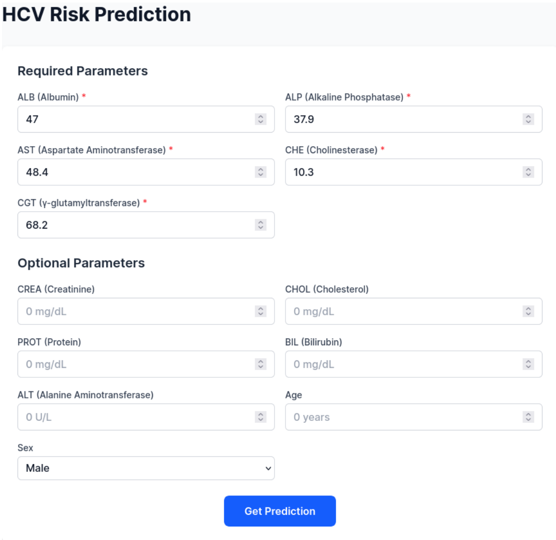
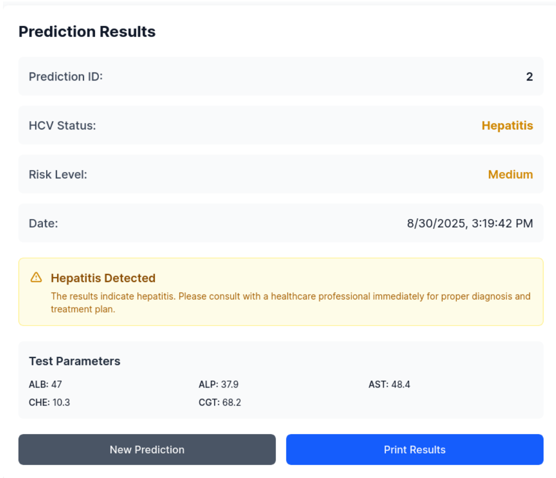


Figure 2: Sequence diagram for showing how the user interacts with the website

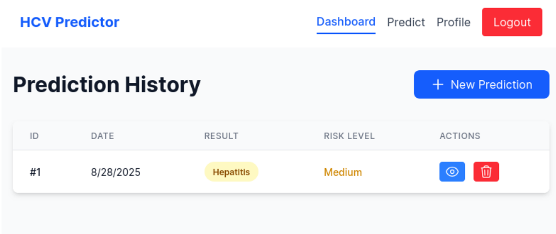
Figure 3: Homepage of HCV Predictor



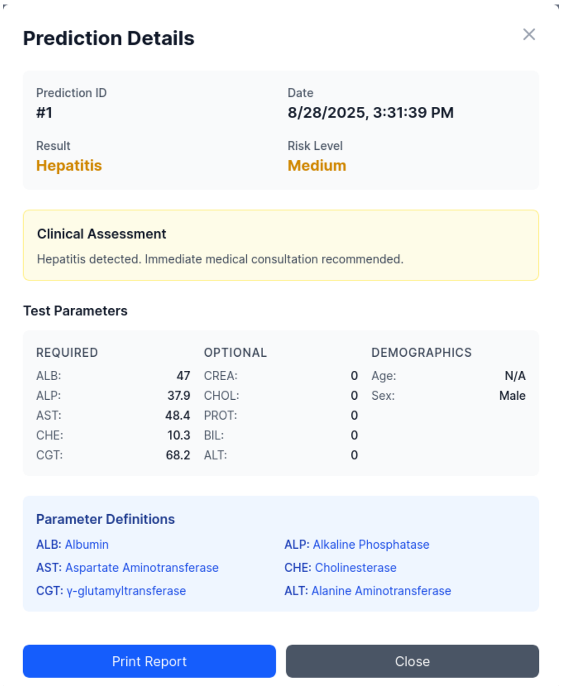
(a) Prediction form



(b) Prediction result

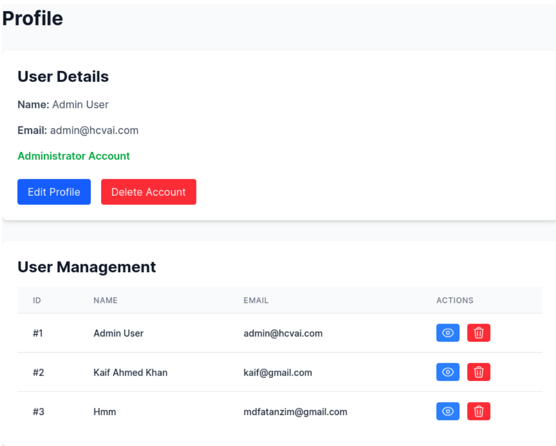


(c) Prediction history in dashboard

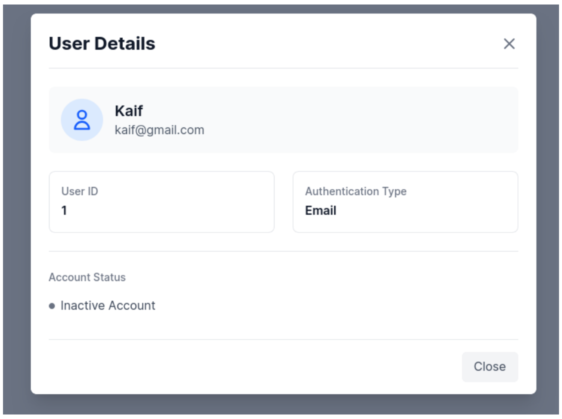


(d) Prediction details modal

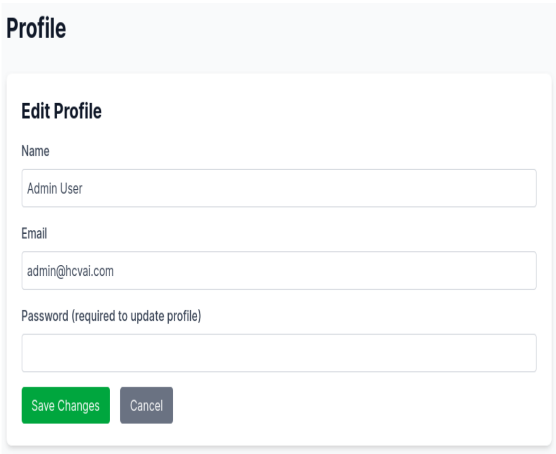
Figure 4: Prediction system and prediction history in dashboard.



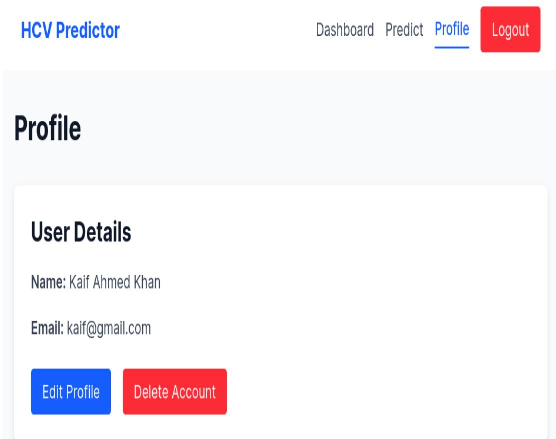
(a) Admin profile page



(b) User details modal



(c) Edit profile details



(d) Non-admin profile page

Figure 5: User Management. Link: https://172.104.62.100/profile

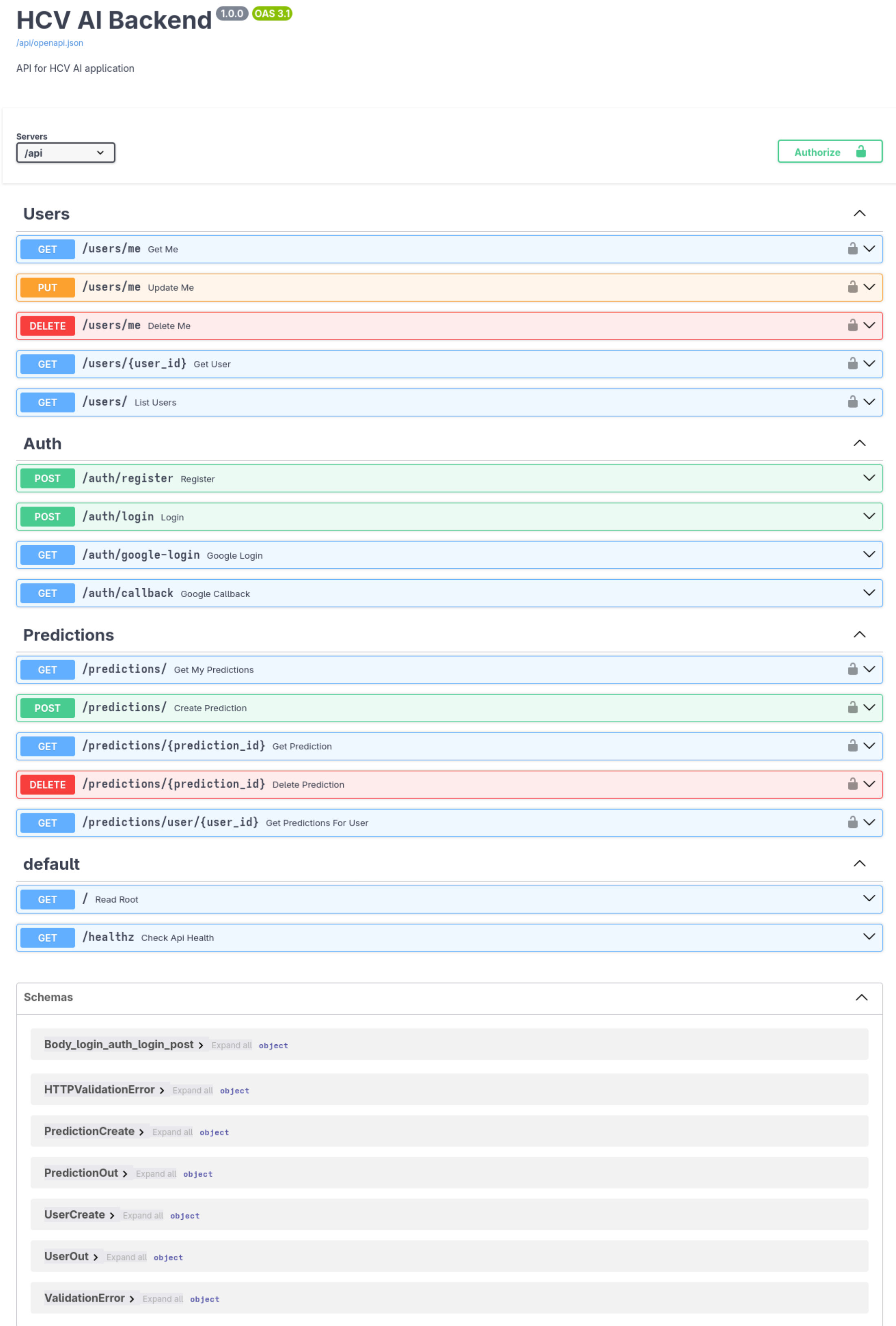


Figure 6: FastAPI backend documentation @ `https://172.104.62.100/api/docs`

- Prediction of HCV stage (4 classes: 0–3).
- Prediction result is displayed instantly to the user.

Prediction History

- Patients can view their own previous predictions.
- Admins can view all users' predictions for monitoring and analysis.

Database Integration

- SQLite database with two main tables:
  - **Users Table**: Stores user credentials and profile information.
  - **Predictions Table**: Stores patient input, prediction results, and timestamps.

Security and Roles

- Authentication system for both patients and admins.
- Role-specific permissions (patients see only their own data, admins see all data).

3 Tools & Technologies

Next.js

The technology stack for the medical prediction website was carefully chosen to balance performance, scalability, and ease of development. Next.js was selected for the frontend because it provides a modern React framework with support for server-side rendering and static site generation, ensuring fast page loads and a responsive user interface. The complete Next.js application implementation is available at `https://github.com/ahmed-kaif/hcv-frontend/`.

FastAPI

FastAPI was chosen for the backend due to its lightweight design, asynchronous capabilities, and automatic API documentation, which allows for efficient handling of requests and easy integration with machine learning models. The backend API is available `https://github.com/ahmed-kaif/hcv-ai/`.

SQLite

SQLite serves as the database system because it is lightweight, file-based, and easy to set up without requiring complex configuration, making it well-suited for rapid development and deployment in small to medium-scale applications.

Scikit-Learn

Scikit-Learn was used for the machine learning component due to its simplicity, extensive functionality for classification tasks, and seamless integration with Python, enabling quick development and deployment of the pretrained HCV prediction model.

Nginx

Nginx was selected as the web server and reverse proxy because of its efficiency in handling concurrent connections, low resource usage, and strong production reliability. It enables seamless routing between the Next.js frontend and FastAPI backend, while providing HTTPS support, caching, and static file serving. Its lightweight design and proven scalability make it an ideal choice for a cost-effective and maintainable deployment. The following configuration script was used for serving both the frontend app and backend API in local machine and remote VPS server.

```
server {
    listen 80;
    server_name _;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name _;

    ssl_certificate     /etc/ssl/cert.pem;
    ssl_certificate_key /etc/ssl/privkey.pem;
    # Proxy Next.js frontend
    location / {
        proxy_pass http://frontend:3000;
        proxy_set_header   Host $host;
        proxy_set_header   X-Real-IP $remote_addr;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Proto $scheme;
    }
    # Proxy FastAPI backend
    location /api/ {
        proxy_pass http://backend:8000/;
        proxy_set_header   Host $host;
        proxy_set_header   X-Real-IP $remote_addr;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Proto $scheme;
    }
}
```

Docker

Docker was chosen to ensure consistent and portable deployments across different environments. By containerizing the Next.js frontend, FastAPI backend, and supporting services e.g. nginx, it eliminates dependency conflicts and simplifies setup. Docker also streamlines scaling, updates, and integration with hosting platforms, making the system easier to manage and more reliable.

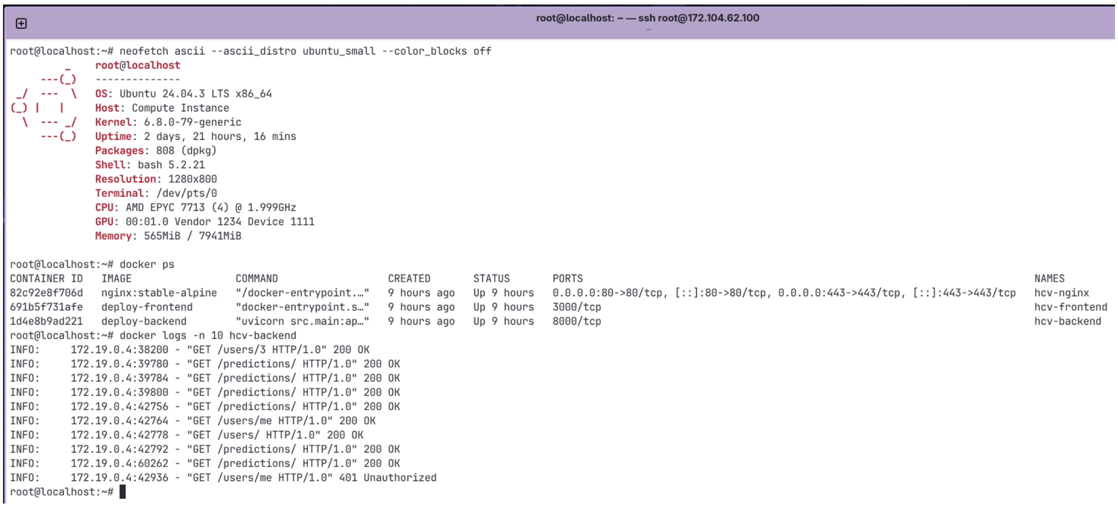


Figure 7: Docker running on linode ubuntu server vps

This combination of technologies ensures that the system is easy to develop, performant, and maintainable, while also allowing for future migration to more scalable databases if the application grows.

4 Sustainability & Environmental Considerations

In developing the medical prediction website, sustainability and environmental impact have been considered in both design and deployment. The following strategies are applied:

Performance Optimization

The backend, implemented using FastAPI, is lightweight and asynchronous, which reduces server load and ensures efficient handling of requests. The frontend, built with Next.js, supports static site generation and server-side rendering, which decreases repeated computation and improves response times. These optimizations reduce unnecessary resource consumption and improve the overall energy efficiency of the system.

Energy Efficiency in Coding & Infrastructure

The codebase follows modular and efficient practices to minimize redundant operations. Database queries are optimized to avoid excessive computation in SQLite. Containerization with Docker allows for consistent and isolated environments, reducing wasted resources across deployments. Additionally, horizontal scaling strategies are preferred over overprovisioning, ensuring resources are used only when required.

Long-Term Impact

By deploying the application on modern cloud platforms such as Render[2] the system benefits from shared, energy-efficient infrastructure. This reduces the need for dedicated high-power servers. The chosen stack (FastAPI+Next.js) supports scalability, which means the application can grow without significant increases in energy use per user. In the long term, the platform aims to provide medical predictions with minimal environmental footprint while supporting sustainable digital health practices.

5 Project Management & Cost Estimation

Project Timeline

The development phase is designed to be lightweight and completed in approximately 6 weeks:

- Requirement Analysis and Design: 0.5 week
- Backend Development (FastAPI): 1 week
- Frontend Development (Next.js): 1 week
- Database Setup (SQLite): 0.5 week
- Integration and Dockerization: 1 week
- Testing and Debugging: 1 week
- Deployment and Documentation: 1 week

Cost Estimation

The project is planned with minimal cost by leveraging free and open-source technologies:

- **Development Effort:** The project is developed by a single developer as part of an academic exercise, so direct labor cost is negligible. If monetized, the equivalent effort can be valued at approximately $500–$800 (including the developer hiring cost).
- **Operational Costs:** VPS like Linode or AWS EC2 with minimal hardware configuration cost upto $5-$10 per month (for backend and database) and Render (hosting) are sufficient for deployment. Docker is open-source and free. A custom domain is optional, costing about $10 per year.

Overall Estimate: The project can be developed with near-zero cost using free tiers, with only optional expenses for a domain name.

[2] `https://hcv-ai.onrender.com/`

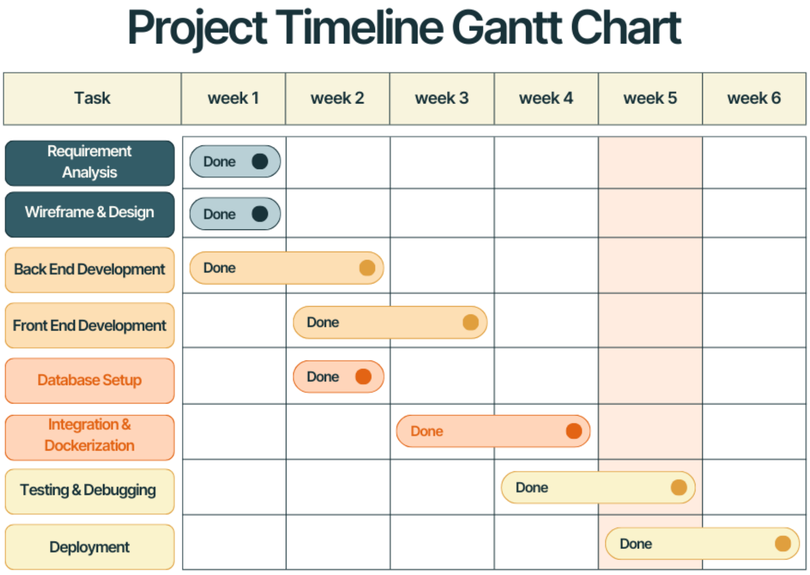


Figure 8: Gantt chart of project timeline.

6 Conclusion

The hepatitis C virus infection stage prediction website successfully delivers a platform where users can register, log in, submit their blood report data, and receive predictions for HCV stages using a pretrained machine learning model. Users can view their prediction history, while administrators have access to all prediction records for monitoring purposes. The integration of FastAPI, Next.js, SQLite, and Docker has resulted in a lightweight, efficient, and scalable system.

During development, several lessons were learned and challenges encountered. Implementing the backend API with FastAPI highlighted the importance of asynchronous programming to efficiently handle multiple requests. Integrating the machine learning model required careful preprocessing and validation to ensure accurate predictions. Designing the database schema and optimizing queries was crucial for reliable storage and retrieval of prediction data. Deploying the application with Docker and on cloud platforms presented challenges related to environment configuration and dependency management. Additionally, managing overlapping frontend and backend development tasks emphasized the need for careful project scheduling.

For future improvements, the system could incorporate enhanced user authentication mechanisms such as multi-factor authentication to strengthen security. Visualization features could be added to display prediction trends over time for each user, and the machine learning model could be continuously retrained with anonymized new data to improve accuracy. This project lays a solid foundation for an efficient, scalable, and user-friendly medical prediction system while providing opportunities for future enhancement.