CS353 Project
Final Report
Spring 2025
Group 2

MEHMET EREN ANBAR 22002600
DENİZ ŞAHİN 22201690
FURKAN ÖZER 22203555
Isa Ahmad Khan 22101309
Furkan Mert Aksakal 22003191

Table of Contents

Table of Contents 1
Project Description 3
Contributions 4
Final E/R Diagram 5
List of Tables Schemas 5
SQL Queries of Functionalities 11
Login & Register 11
Register Patient 11
Register Doctor 11
Register Staff 12
Register Admin 12
Login 12
Appointment 13
Select doctor based on specialization and rating: 13
List available doctors for a given date & time slot 13
List available time slots of a doctor for a given date 14
Check if a time slot is available for a given doctor & date 14
Patient Books Appointment 15
List Appointments of a Given Patient 15
Cancel Appointments of a Given Patient 16
Update Time Slot & Date of a Given Appointment 16
Health Card 17
List of Medications Prescribed to a Given Patient 17
Doctor Views List of Medications available for prescription 17
Doctor Creates a medication 17
Doctor Prescribes Medication 17
Create Blood Test Result 17
List Blood Test Results Of a Given Patient 18
Report 18
Create Department Report 18
List All Reports For a Given Department 18
Create Doctor Report 19
List All Reports For a Given Doctor 19
Implementation Details 20
Advanced Database Components 20
User Manual 20
Login Page: 20
Registration Page: 21
Patient Dashboard: 21
Patient Appointment Calendar: 22
7.5 Patient Feedback: 22
7.6 Booking Appointment: 23

7.7 Patient Health Card:.....23
7.8 Doctor Schedule/Appointments:.....24
Declaring Unavailability:.....24
7.9 Doctor’s Inventory List:.....25
7.10 Staff Dashboard:.....25
Tests Page:.....26
Searching and Managing Inventory (Staff):.....26
Adding new Items:.....27
7.11 Admin Panel:.....27
Edit User:.....28
Add User:.....28

1. Project Description

This is the final documentation for a CS353 project. This project is a hospital management system. The project aims to create a website for all sorts of hospital management needs.

Hospital Administration and Management System (HAMS) is a web application that aims at scheduling hospital appointments, staff timetables, and medical equipment in an optimal way to better the overall procedure of healthcare. The system caters to the patients, physicians, hospital authorities, and health professionals with everyone having access to some functionalities particular to their undertakings. These functionalities render hospital tasks seamless and health resources efficiently exploited.

Patients can search physicians based on specialty and availability, schedule appointments, and keep their medical records, such as prescriptions, lab test results, etc., updated with the system. Patients are also offered the choice of filling out a feedback form for the physicians. Physicians can view and schedule their appointments, see patient histories, and order required medical supplies such as test kits and equipment. The system will be controlled by hospital administrators, monitor appointment patterns, and generate reports to ensure maximum hospital efficiency and the allocation of resources. At the same time, the medical staff can make requested resources available and assign them to the corresponding physicians and departments, leading to an efficient hospital facility.

Architecture in the system is capable of supporting more than one user role to provide streamlined communication among patients, doctors, and administrators. Every role supports some exclusive functionalities for effective user experience along with increased operating efficiency. Administrators, for example, are able to report doctor performances and trends for appointments so they can make the right decisions toward improving resource utilization as well as efficiency at the hospital. By having greater control over patient data and their schedules, doctors are able to improve patient care and minimize scheduling conflicts. Hospitals can function without delay or shortage since medical resources are tracked in real time. This application minimizes inefficiencies in manual processes, decreases human errors, and provides real-time feedback to all parties with automated appointment scheduling, resource assignment, and reporting capabilities. Hospital operations are streamlined by the system and improve patient satisfaction, optimize staff workload, and optimize hospital performance overall.

2. Contributions

Mehmet Eren Anbar:

- Created framework for frontend.
- Wrote in its entirety the patient appointment framework.
    - Department filtering.
    - Date and time availability.
    - Patient balance.

Furkan Mert Aksakal:

- Wrote in its entirety the patient dashboard, feedback and health card pages.
    - Retrieve patient blood tests.
    - Retrieve patient prescriptions.
    - Give feedback to doctors.

Deniz Şahin:

- Created framework for backend.
- Wrote the backend endpoints for doctor and staff.
    - Create blood test.
    - List and update equipment stock.
    - Declare unavailability.

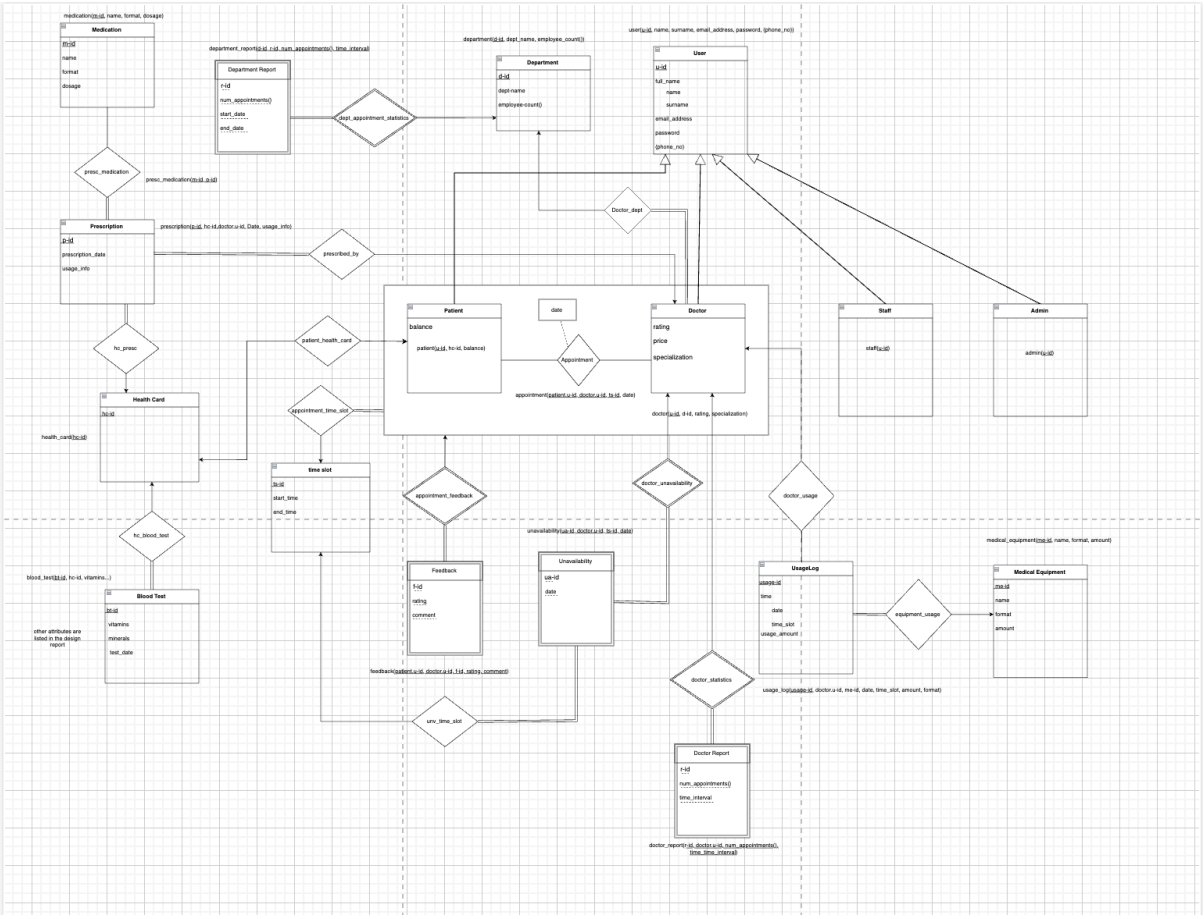
Furkan Özer:

- Wrote the entire admin page.
    - Define, delete and edit users.
    - Change patient balance.
    - Manage doctor specializations.

Isa Ahmad Khan:

- Wrote the front end for doctor and staff.
    - List available equipment.
    - Blood test form.
    - Doctor schedule.
- Connected the backend and frontend of the pages he wrote.

3. Final E/R Diagram



Click here for the full ER Diagram

4. List of Tables Schemas

medication(	m-id	**char**(5),
	name	**varchar**(50) **not null**,
	format	**varchar**(20) **not null**,
	dosage	**numeric**(5,1) **not null**,
	primary key (m-id)	
)		
prescription(	p-id	**char**(5),
	hc-id	**char**(5),
	doc-id	**char**(5),
	prescription_date	**date**,

```
                usage_info          varchar(100),
                primary key (p-id),
                foreign key (hc_id) references health_card,
                foreign key (doc-id) references doctor(u-id)
)
health_card(    hc-id               char(5),
                primary key(hc-id)
)
blood_test(     bt-id               char(5),
                hc-id               char(5),
                vitamins            varchar(100),
                minerals            varchar(100),
                cholesterol         numeric(5, 2),
                glucose             numeric(5, 2),
                hemoglobin          numeric(5, 2),
                white_blood_cells   numeric(5, 2),
                red_blood_cells     numeric(5, 2),
                test_date           date not null,
                primary key(bt-id),
                foreign key (hc_id) references health_card
)
department_report(  r-id            char(5),
                d-id                char(5),
                num_appointments    int not null,
                start_date          date not null,
                end_date            date not null,
                primary key(r-id),
                foreign key(d-id) references department
)
```

```
department(     d-id            char(5),
                dept_name       varchar(15) not null,
                employee_count  int,
                primary key(d-id)
)
patient(        u-id            char(5),
                hc-id           char(5),
                balance         numeric(8,2)
                primary key(u-id),
                foreign key(u-id) references user
                foreign key(hc-id) references health_card
)
unavailability( ua-id           char(5),
                ts-id           char(5),
                doc-id          char(5),
                date            date not null,
                primary key(ua-id),
                foreign key(doc-id) references doctor(u-id),
                foreign key(ts-id) references time_slot
)
feedback(       f-id            char(5),
                patient-id      char(5),
                doc-id          char(5),
                rating          numeric(2,1) not null,
                comment         varchar(200),
                primary key(f-id),
                foreign key(doc-id) references doctor(u-id),
                foreign key(patient-id) references patient(u-id)
)
```

```
user(            u-id              char(5),
                 name              varchar(50) not null,
                 surname           varchar(50) not null,
                 email_address     varchar(50),
                 password          varchar(15) not null,
                 {phone_no}        numeric(12),
                 primary key(u-id)
)
doctor_report(   r-id              char(5),
                 doc-id            char(5),
                 num_appointments  int,
                 start_date        date,
                 end_date          date,
                 primary key(r-id)
                 foreign key(doc-id) references doctor(u-id)
)
doctor(          u-id              char(5),
                 d-id              char(5),
                 rating            numeric(2,1),
                 price             numeric(5,0),
                 primary key(u-id),
                 foreign key(u-id) references user);
                 foreign key(d-id) references department);
)
staff(           u-id              char(5),
                 primary key(u-id),
                 foreign key u-id references user
)
admin(           u-id              char(5)
```

primary key(u-id)

foreign key(u-id) **references** user

)

usage_log(usage-id **char**(5),

doc-id **char**(5),

me-id **char**(5),

date **date not null**,

ts-id **char**(5) **not null**,

amount **int not null**,

format **varchar**(20),

primary key(usage-id),

foreign key (doc-id) **references** doctor(u-id),

foreign key (me-id) **references** medical equipment

)

medical_equipment(me-id **char**(5),

name **varchar**(50) **not null**,

format **varchar**(20),

amount **numeric**(5,0)

primary key(me-id)

)

presc_medication(m-id **char**(5),

p-id **char**(5),

primary key(m-id, p-id),

foreign key(m-id) **references** medication,

foreign key(p-id) **references** prescription

)

appointment(patient-id **char**(5),

doc-id **char**(5),

ts-id **char**(5),

```
                    date                    date,
                    primary key(patient-id, doc-id, ts-id, date),
                    foreign key(patient-id) references patient(u-id),
                    foreign key(doc-id) references doctor(u-id),
                    foreign key(ts-id) references time_slot
)
time_slot(          ts-id                   char(5),
                    start_time              time,
                    end_time                time,
)
```

5. SQL Queries of Functionalities

Login & Register

Register Patient

```
INSERT INTO user (u-id, name, surname, email_address, password, phone_no)
VALUES ('00001', 'deniz', ‘şahin’, 'deniz@şahin.com', '^’+R^’!+!>’^>1', 905555555555);

INSERT INTO patient (u-id, hc-id, balance)
VALUES ('00001', '00001', 0);

CREATE TRIGGER create_healthcard AFTER INSERT OF patient
REFERENCING NEW ROW AS nrow
FOR EACH ROW
BEGIN
        INSERT INTO health_card (hc-id)
        VALUES nrow.hc-id
END
```

Register Doctor

```
INSERT INTO user (u-id, name, surname, email_address, password, phone_no)
VALUES ('00002', 'eren', ‘anbar’, 'eren@anbar.com', '^’+R^’!+!>’^>1', 905555555555);

INSERT INTO doctor(u-id, d-id, rating, price)
VALUES ('00002', ‘00001’, 0.0, 300);

CREATE TRIGGER increment_department_employee_count
AFTER INSERT ON doctor
FOR EACH ROW
BEGIN
   UPDATE department
```

```
    SET employee_count = employee_count + 1
    WHERE d-id = NEW.d-id;
END;
```

Register Staff

```
INSERT INTO user (u-id, name, surname, email_address, password, phone_no)
VALUES ('00003', 'anbar', 'eren’, 'eren@eren.com', 'password', 905444444444);

INSERT INTO staff (u-id)
VALUES ('00003');
```

Register Admin

```
INSERT INTO user (u-id, name, surname, email_address, password, phone_no)
VALUES ('00004', 'mehmet', ‘eren’, 'mehmet@eren.com', 'password', 905333333333);

INSERT INTO admin (u-id)
VALUES ('00004');
```

Login

```
(SELECT u.u-id, u.name, u.surname,
      'patient' AS role
FROM user u NATURAL JOIN patient p
WHERE u.email_address = 'eren@anbar.com' AND u.password = '^’+R^’!+!>’^>1'
)
UNION

(SELECT u.u-id, u.name, u.surname,
      'doctor' AS role
```

```
FROM user u NATURAL JOIN doctor d
WHERE u.email_address = 'eren@anbar.com' AND u.password = '^'+R^'!+!>'^>1'
)
UNION

(SELECT u.u-id, u.name, u.surname,
    'staff' AS role
FROM user u
NATURAL JOIN staff s
WHERE u.email_address = 'eren@anbar.com'
AND u.password = '^'+R^'!+!>'^>1'
)
UNION

(SELECT u.u-id, u.name, u.surname,
    'admin' AS role
FROM user u NATURAL JOIN admin a
WHERE u.email_address = 'eren@anbar.com' AND u.password = '^'+R^'!+!>'^>1')
```

Appointment

Select doctor based on specialization and rating:

```
SELECT d.u-id, d.name, d.surname
FROM  doctor AS d
WHERE specialization = 'chest' AND rating > 4
```

List available doctors for a given date & time slot

```
SELECT d.u-id, d.name, d.surname
FROM doctor AS d
WHERE NOT EXISTS (
        SELECT *
```

```
        FROM appointment AS a
        WHERE a.doc-id = d.u-id AND a.date = '2025-04-01' AND a.ts-id = 3
)
AND NOT EXISTS (
        SELECT *
        FROM unavailability AS un
        WHERE un.doc-id = d.u-id AND un.date = '2025-04-01' AND un.ts-id = 3
)
```

List available time slots of a doctor for a given date

```
(SELECT ts-id, start_time, end_time
FROM time_slot)
EXCEPT
(
        (SELECT a1.ts-id, ts1.start_time, ts1.end_time
        FROM appointment AS a1, time_slot AS ts1
        WHERE a1.ts-id = ts1.ts-id AND a1.doc-id = ‘00001’ AND a1.date = '2025-04-01')
UNION
        (SELECT unav.ts-id, ts2.start_time, ts2.end_time
        FROM unavailability AS unav, time_slot AS ts2
        WHERE unav.ts-id = ts2.ts-id AND unav.doc-id = ‘00001’ AND unav.date =
        '2025-04-01')
)
```

Check if a time slot is available for a given doctor & date

This query returns the ts-id if such a time slot is available, otherwise returns an empty table.

```
SELECT ts.ts-id
FROM time_slot AS ts
WHERE ts.ts-id = 4 AND ts.ts-id NOT IN
(
        (SELECT a1.ts-id
```

```
        FROM appointment AS a1
        WHERE a1.doc-id = ‘00001’ AND a1.date = '2025-04-01')
UNION
        (SELECT unav.ts-id
        FROM unavailability AS unav
        WHERE unav.doc-id = ‘00001’ AND unav.date = '2025-04-01')
)
```

Patient Books Appointment

```
INSERT INTO appointment (patient-id, doc-id, date, ts-id)
VALUES ('3', '2', '2025-04-01', ‘3');

CREATE TRIGGER reduce_balance_after_appointment
AFTER INSERT ON appointment
FOR EACH ROW
BEGIN
        DECLARE doctor_price DECIMAL(5, 0);

        SELECT price INTO doctor_price
        FROM doctor
        WHERE u-id = NEW.doc-id;

        UPDATE patient
        SET balance = balance - doctor_price
        WHERE u-id = NEW.patient-id;
END
```

List Appointments of a Given Patient

```
SELECT a.date, ts.start_time, ts.end_time, d.name AS doctor_name, d.surname AS
doctor_surname
FROM appointment AS a
```

```
JOIN doctor AS d ON a.doc-id = d.u-id
JOIN time_slot AS ts ON a.ts-id = ts.ts-id
WHERE a.patient-id = ‘1';
```

Cancel Appointments of a Given Patient

```
DELETE FROM appointment
WHERE (patient-id, doc-id, ts-id, date) = ('1', '2', '3', '2025-04-01');
```

Update Time Slot & Date of a Given Appointment

```
UPDATE appointment
SET ts-id = 'NEW_TS_ID', date = 'NEW_DATE'
WHERE patient-id = 'PATIENT_ID'
AND doc.id = 'DOCTOR_ID'
AND ts-id = 'OLD_TS_ID'
AND date = 'OLD_DATE'
AND NOT EXISTS (
   SELECT * FROM appointment a
   WHERE a.doc.id = 'DOCTOR_ID'
   AND a.date = 'NEW_DATE'
   AND a.ts-id = 'NEW_TS_ID'

   UNION

   SELECT * FROM unavailability un
   WHERE un.doc.id = 'DOCTOR_ID'
   AND un.date = 'NEW_DATE'
   AND un.ts-id = 'NEW_TS_ID'
);
```

Health Card

List of Medications Prescribed to a Given Patient

SELECT p.p-id, m.name **AS** medication_name, m.format, m.dosage, p.prescription_date, p.usage_info

FROM prescription **AS** p, medication **AS** m, patient **AS** pat

WHERE p.m-id = m.m-id **AND** p.hc-id = pat.hc-id **AND** pat.u-id = '1'

Doctor Views List of Medications available for prescription

SELECT m-id, name, format, dosage

FROM medication;

Doctor Creates a medication

INSERT INTO medication (m-id, name, format, dosage)

VALUES ('0002', 'Aspirin', 'Tablet', 500.0);

Doctor Prescribes Medication

INSERT INTO prescription (p-id, hc-id, doc-id, date, usage_info)

VALUES ('002', '001', '0001', '2025-04-01', 'Take 3 tablets morning evening and night');

Create Blood Test Result

INSERT INTO blood_test (

bt-id, hc-id, vitamins, minerals, cholesterol, glucose, hemoglobin,

white_blood_cells, red_blood_cells, test_date

)

VALUES (

'2', '1', 'Vitamin A: 50mcg, Vitamin C: 20mg',

'Calcium: 9mg/dL, Magnesium: 2.5mg/dL', 180.00, 90.00, 13.5,

6.5, 4.7, 250000, '2025-04-01'

);

List Blood Test Results Of a Given Patient

```
SELECT bt.bt-id, bt.vitamins, bt.minerals, bt.cholesterol, bt.glucose,
    bt.hemoglobin, bt.white_blood_cells, bt.red_blood_cells, bt.test_date
FROM blood_test AS bt, patient AS pat
WHERE bt.hc-id = pat.hc-id AND pat.u-id = '1'
```

Report

Create Department Report

```
INSERT INTO department_report (r-id, d-id, num_appointments, start_date, end_date)
SELECT
        'DR' || LPAD(ROW_NUMBER() OVER (ORDER BY d."d-id")::text, 3, '0') AS
report_id,
        d."d-id",
        COUNT(*) AS num_appointments,
        :start_date,
        :end_date
FROM appointment a
JOIN doctor d
  ON a."doc-id" = d."u-id"
WHERE a.date BETWEEN :start_date AND :end_date
GROUP BY d."d-id";
```

List All Reports For a Given Department

```
SELECT *
FROM department_report
WHERE d-id = :department_id;
```

Create Doctor Report

```
INSERT INTO doctor_report (r-id, "doc-id", num_appointments, start_date, end_date)
SELECT
        'DR' || LPAD(ROW_NUMBER() OVER (ORDER BY a."doc-id")::text, 3, '0') AS report_id,
        a."doc-id",
        COUNT(*) AS num_appointments,
        :start_date,
        :end_date
FROM appointment a
WHERE a.date BETWEEN :start_date AND :end_date
GROUP BY a."doc-id";
```

List All Reports For a Given Doctor

```
SELECT *
FROM doctor_report
WHERE "doc-id" = :doctor_id;
```

6. Implementation Details

In HAMS, we have utilized React, to create a user interface. The user interface includes login and registration pages and feature-rich dashboards for different types of users. Styling was achieved through the assistance of Tailwind CSS.
On the back end, we used Django. It assisted us in giving a solid foundation to core modules such as user authentication, appointment scheduling, medical equipment tracking, and health record management. Raw SQL query support enabled us to implement performance database logic when necessary, as well as leverage Django's ORM for normal usage. There is a corresponding dedicated API endpoint for each module—accounts, appointments, health card, and equipment—that makes the system modular and simple to maintain.

We had PostgreSQL as our database system, and employed it because of native Django support and native support of advanced SQL features like foreign key constraints, triggers, and stored procedures. Advanced features were needed to enforce domain logic like the availability of doctors, automatic status update of the appointment, and consistency of medical records.

7. Advanced Database Components

7.1 Trigger: create health_card after registering patient:

```
CREATE OR REPLACE FUNCTION create_healthcard()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO health_card (hc_id) VALUES (NEW.hc_id);
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


DROP TRIGGER IF EXISTS trg_create_healthcard ON patient;
CREATE TRIGGER trg_create_healthcard
BEFORE INSERT ON patient
FOR EACH ROW EXECUTE FUNCTION create_healthcard();
```

7.2 Trigger: reduce patient balance after appointment

```
CREATE OR REPLACE FUNCTION reduce_balance()
RETURNS TRIGGER AS $$
DECLARE
  doctor_price NUMERIC(5,0);
BEGIN
  SELECT price INTO doctor_price FROM doctor WHERE u_id = NEW.doc_id;
  UPDATE patient SET balance = balance - doctor_price WHERE u_id = NEW.patient_id;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


DROP TRIGGER IF EXISTS trg_reduce_balance ON appointment;
CREATE TRIGGER trg_reduce_balance
AFTER INSERT ON appointment
FOR EACH ROW EXECUTE FUNCTION reduce_balance();
```

7.3 Trigger: increment department count after adding doctor

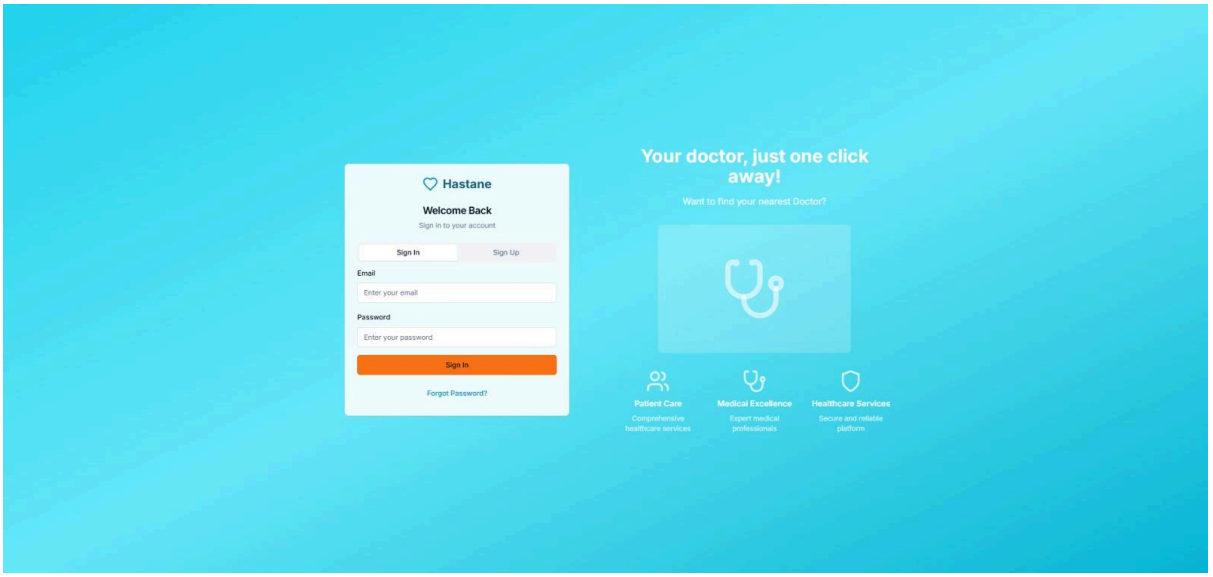
```
CREATE OR REPLACE FUNCTION increment_department_count()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE department SET employee_count = employee_count + 1 WHERE d_id =
NEW.d_id;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


DROP TRIGGER IF EXISTS trg_increment_department ON doctor;
CREATE TRIGGER trg_increment_department
AFTER INSERT ON doctor
FOR EACH ROW EXECUTE FUNCTION increment_department_count();
```

User Manual

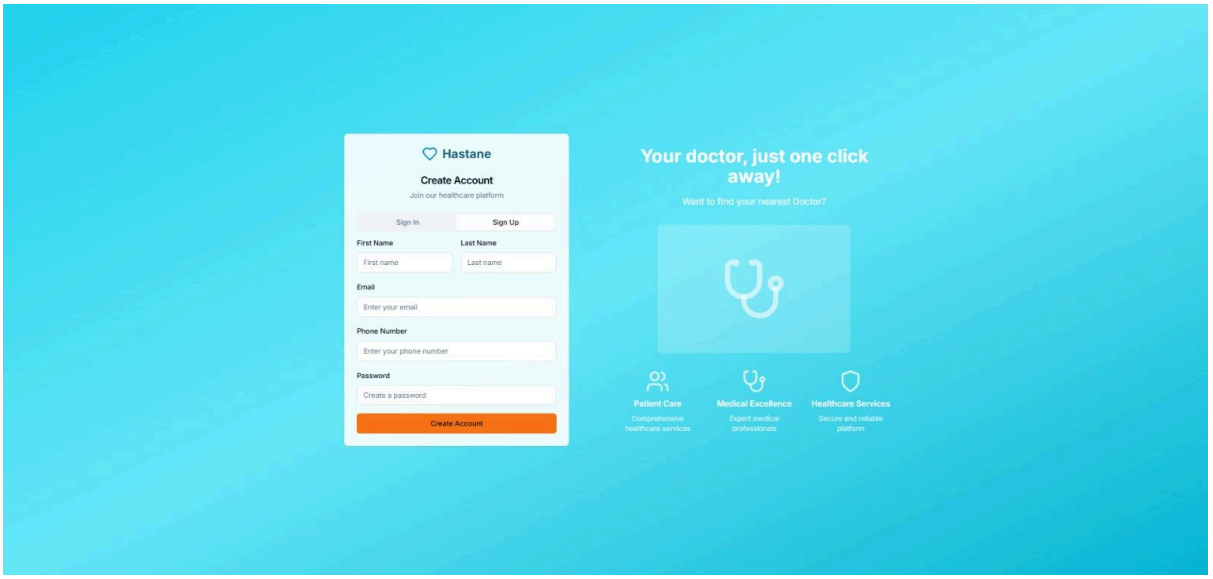
8.1 Login Page:

Users can login to the HAMS system by entering their Email and Password. If the user forgets their password, they can click on “Forgot Password” to reset it. If they have no account, users can click sign up to register.



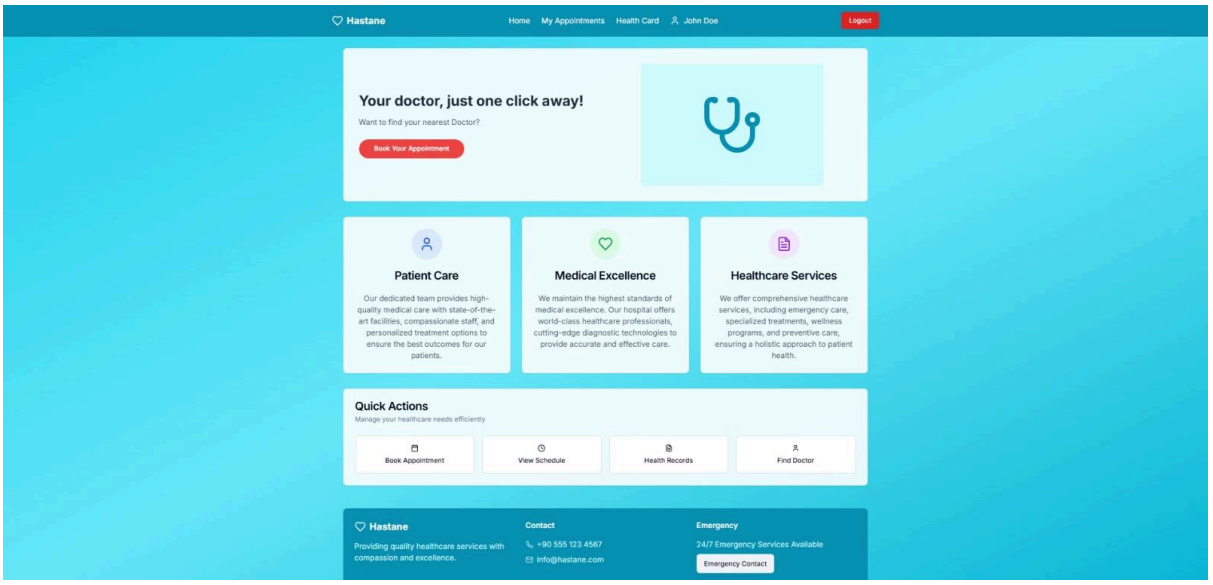
8.2 Registration Page:

To register, users need to provide First Name, Last name, Email, Phone Number and Password. After creating the account, the user will be prompted to the Patient Dashboard. While registering, if the email is already in use, the user will be prompted back to the Sign in page.



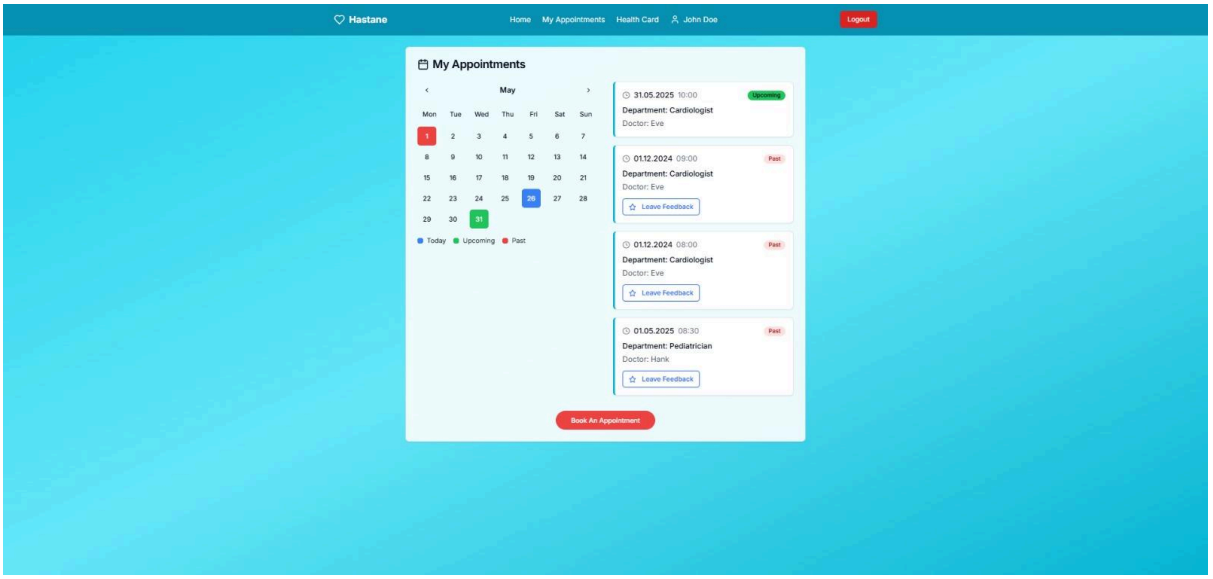
8.3 Patient Dashboard:

After logging in, the patient is prompted to the dashboard, where they can book their appointments, check their health cards and logout.



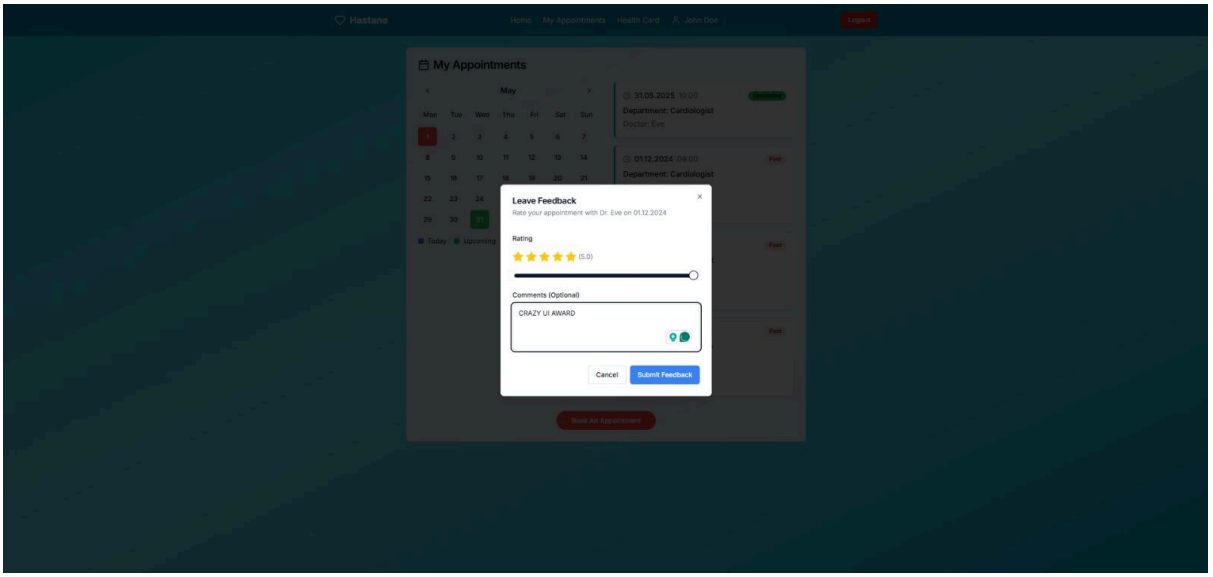
8.4 Patient Appointment Calendar:

Patients can check the dates of their past and upcoming appointments and details about those appointments



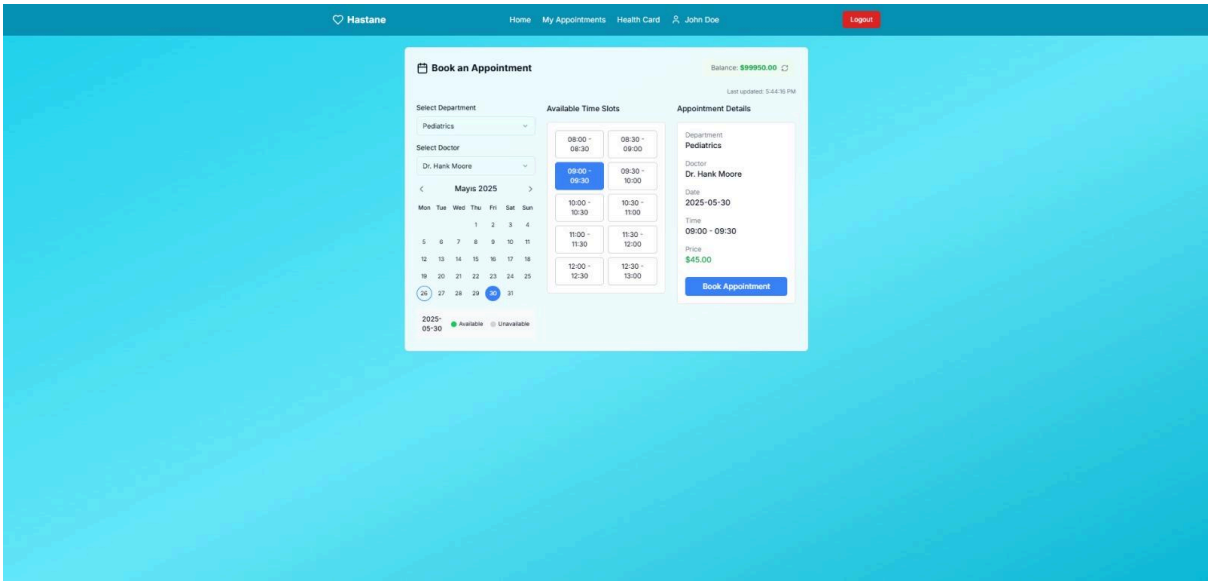
8.5 Patient Feedback:

Patients can leave feedback for the doctor, which includes comments and ratings out of five.



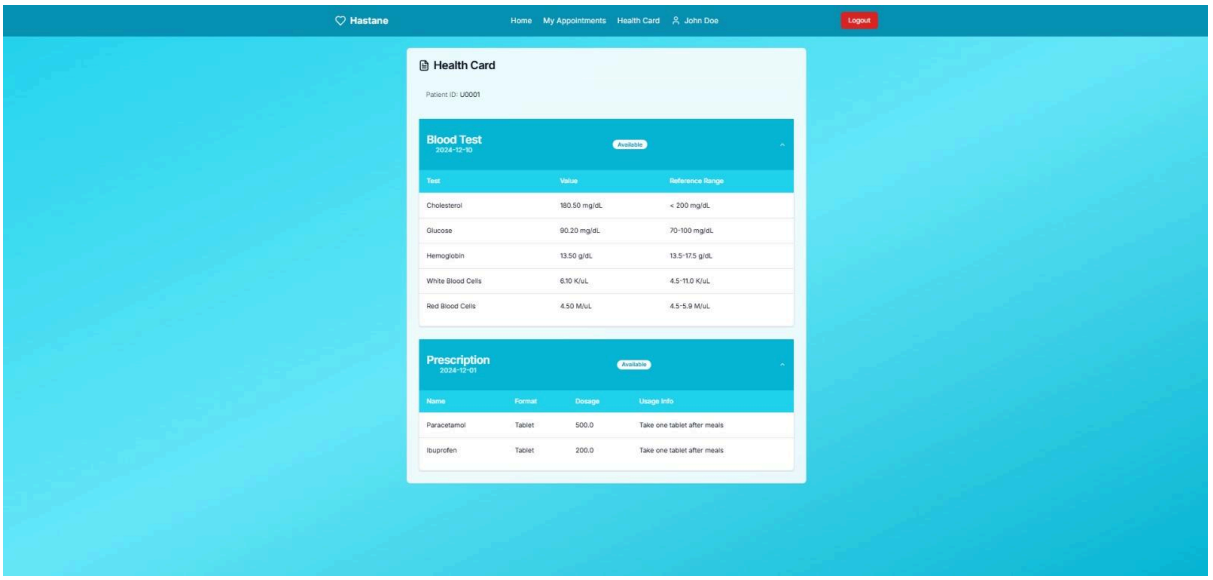
8.6 Booking Appointment:

Patients can book appointments for the doctor they want on a given day. They can search the doctor and/or filter them by department, availability dates.



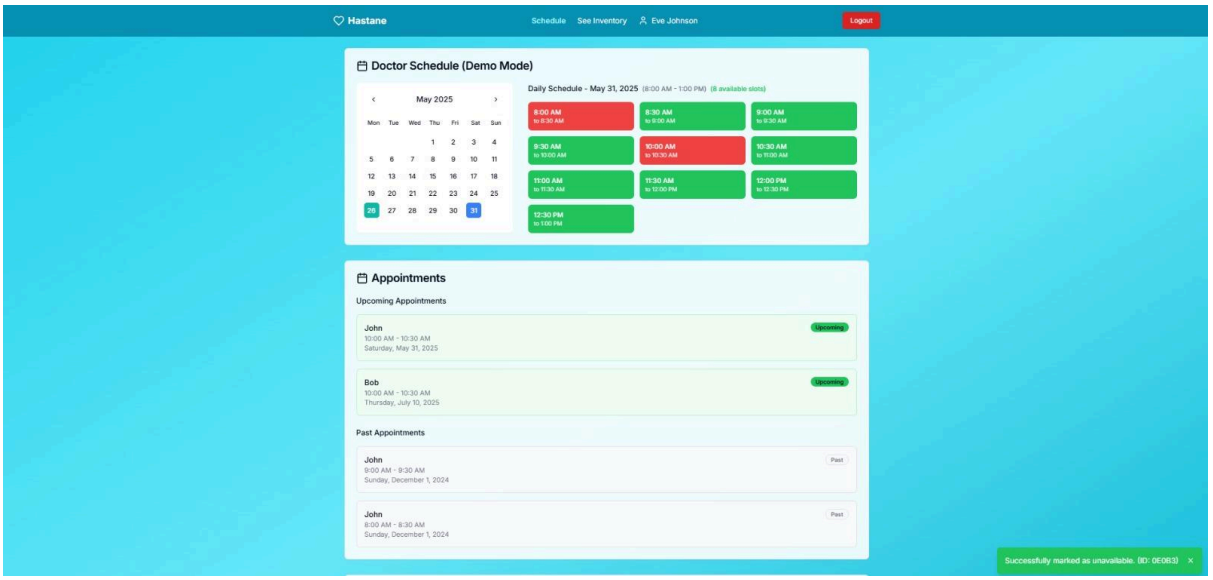
8.7 Patient Health Card:

Patients can view their Health Cards. The health card gives detailed information about Tests and prescriptions allotted.

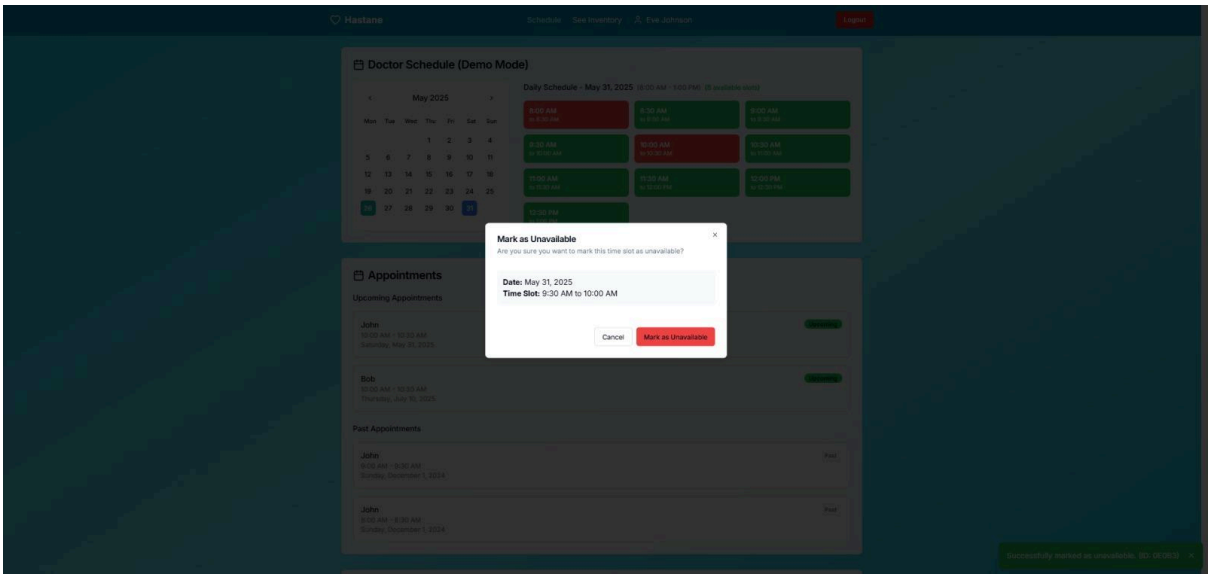


8.8 Doctor Schedule/Appointments:

Doctors can mark their working hours from their dashboard. They can mark certain hours as unavailable. They can also check their upcoming and past appointments.



Declaring Unavailability:

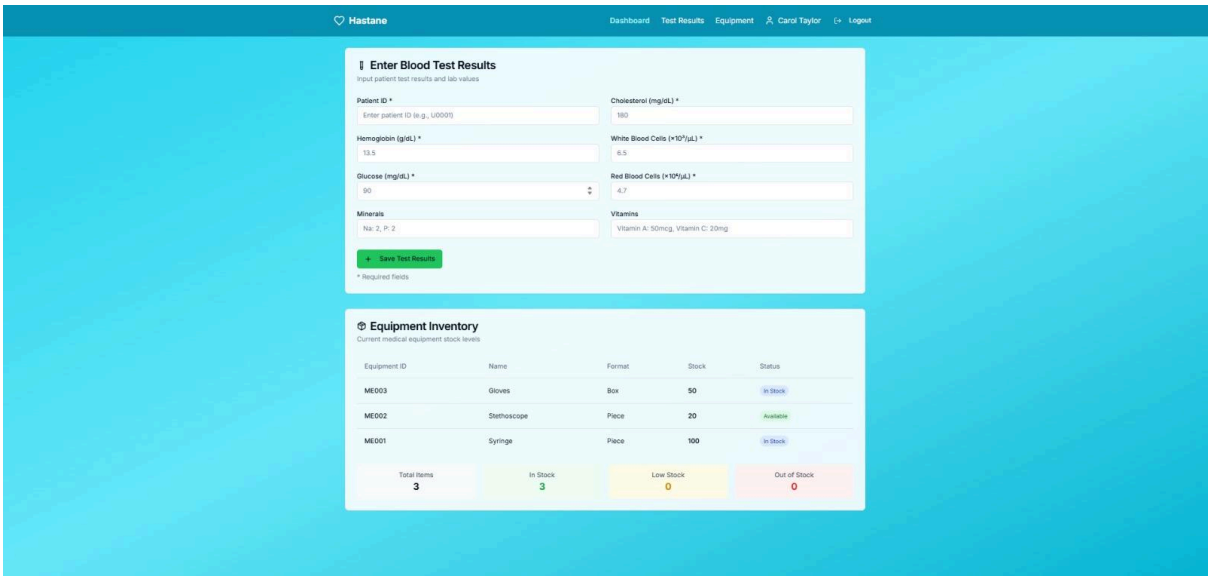


8.9 Doctor's Inventory List:

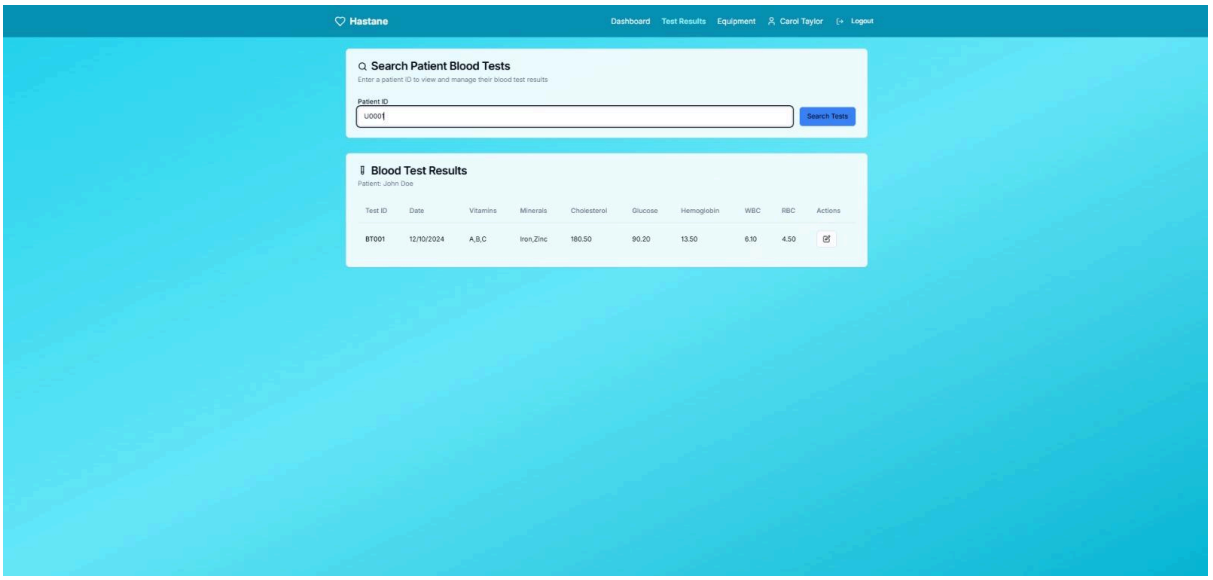
Doctors can check the inventory for all the items that are available.

8.10 Staff Dashboard:

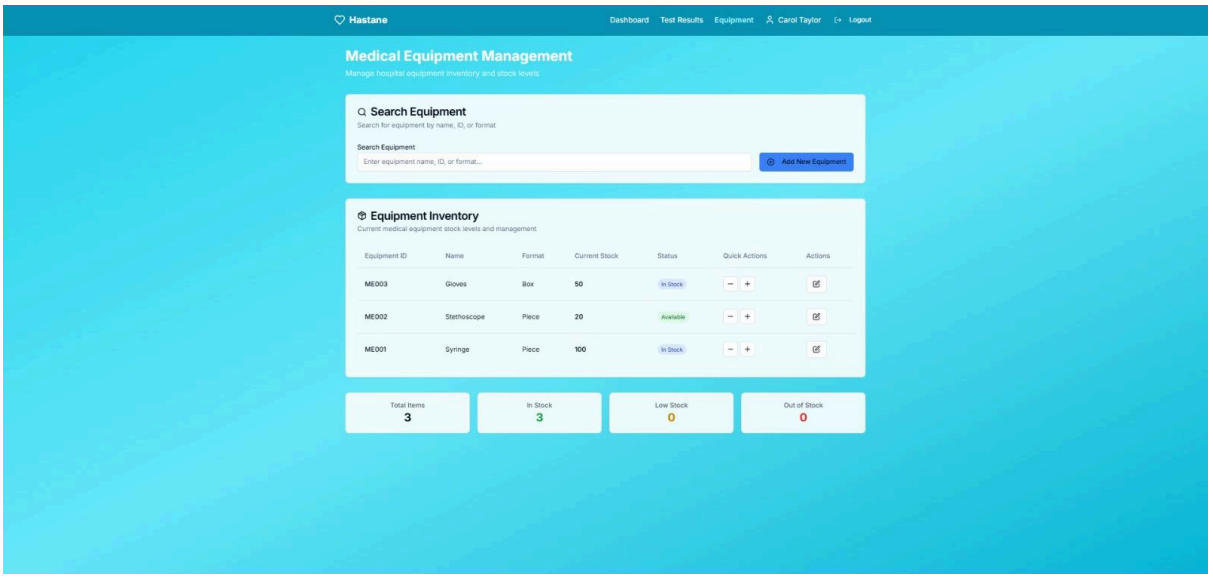
Staff Dashboard lets the staff add/search test results and manipulate inventory for equipments.



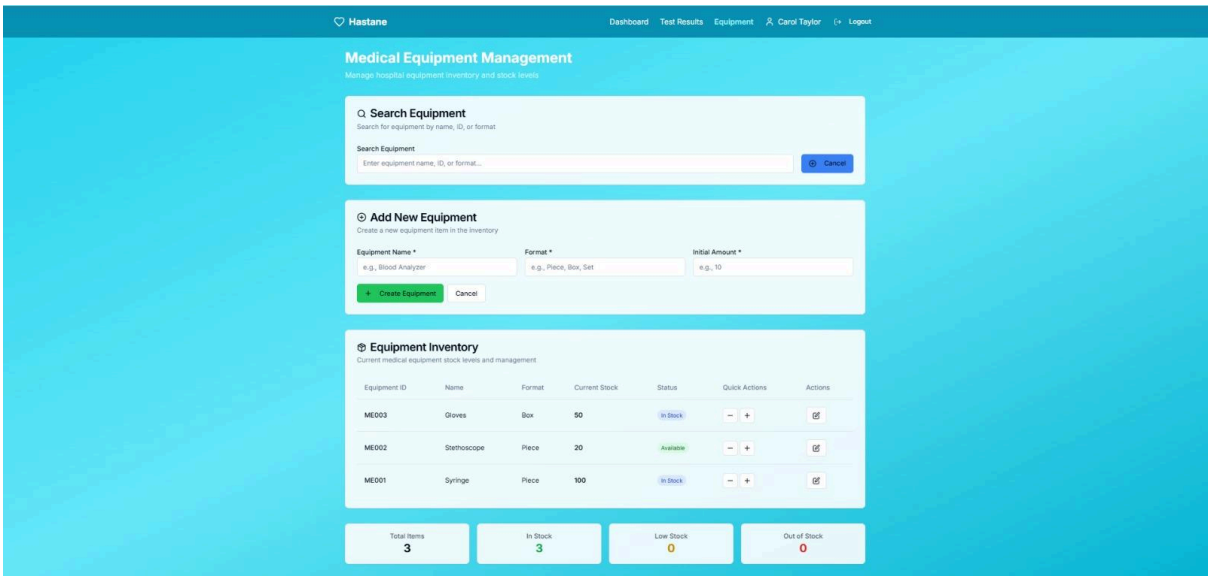
Tests Page:



Searching and Managing Inventory (Staff):

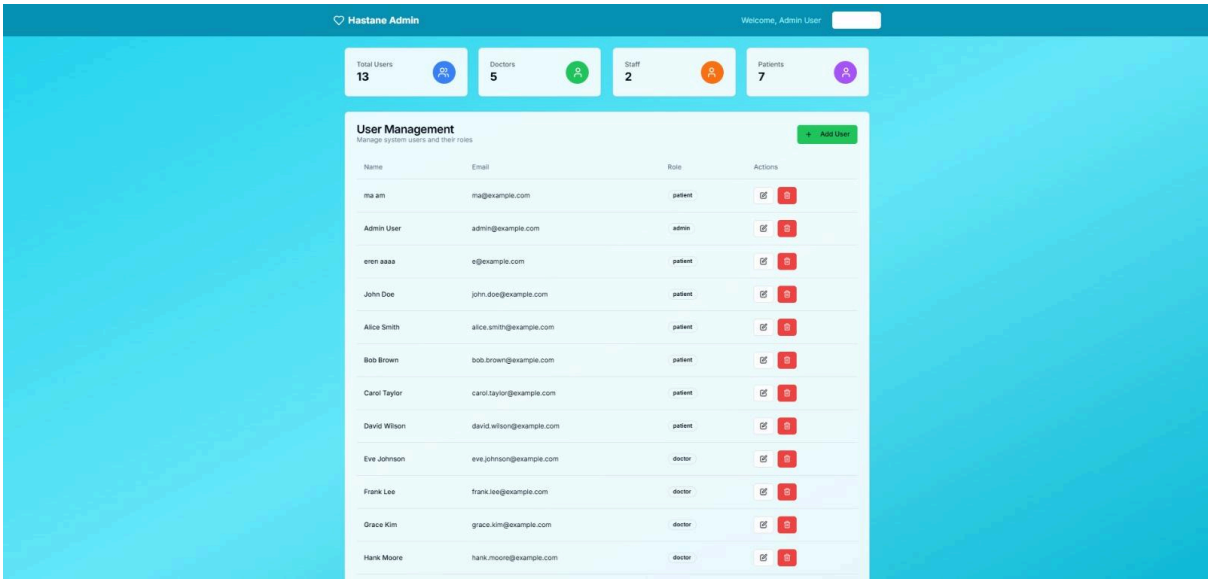


Adding new Items:

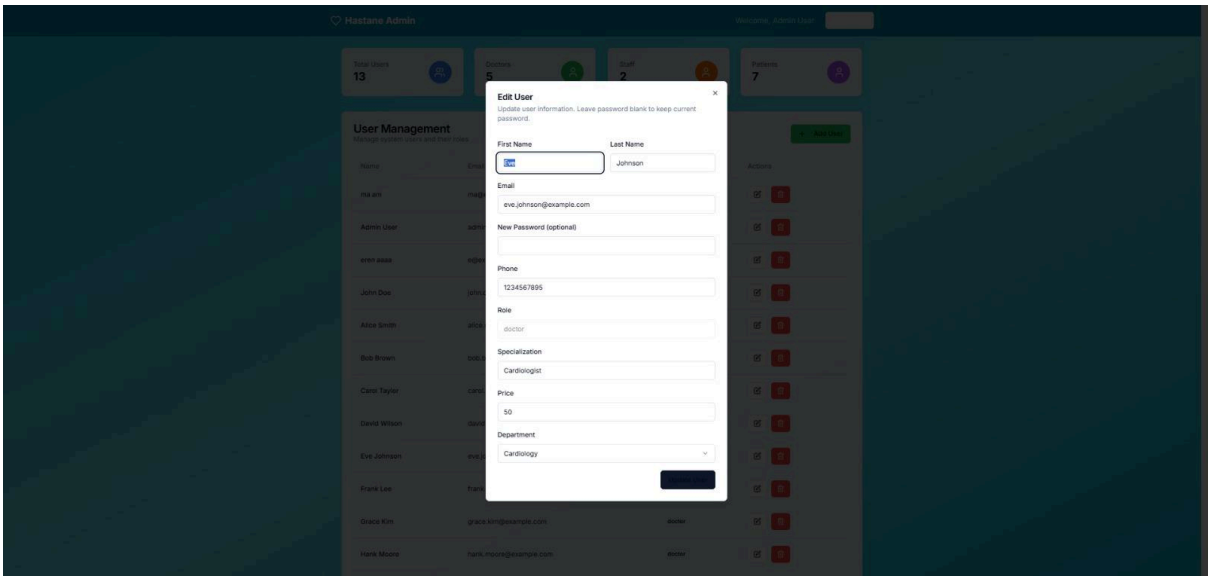


8.11 Admin Panel:

Admin can check and edit all the users in the system. They can add, delete and edit users in the system.



Edit User:



Add User: