



# **DATABASE SYSTEM (CO2014)**

---

## **Assignment 2**

# **HOSPITAL MANAGEMENT SYSTEM**

---

Advisor: Do Thanh Thai

Students: Tran Tri Dung – 2252133

Nguyen Le Duy Khang - 2252303

Huynh Ngoc Khoa - 2211591

## Contents

<b>1</b>	<b>Member list &amp; Workload .....</b>	<b>3</b>
<b>2</b>	<b>Triggers .....</b>	<b>4</b>
2.1	Prevent Patient Deletion .....	4
2.2	After Appointment Insert .....	4
2.3	Trg_insert_user .....	4
2.4	Trg_update_doctor_speciality .....	5
	Display the number of patient and doctors before proceeding .....	5
	Add a patient to database .....	9
2.4.2	Check_email() .....	10
2.4.3	Booking appointment based on department: .....	11
	• Patient choose the department suitable for their demands and the system will choose the doctor have the free time suitable to patient. ....	11
2.4.4	Booking appointment based on doctor .....	12
	• Retrieve to the database to get the free time of the doctor in that day, after that show to the patient time that doctor is free .....	12
2.4.5	Update Information of Patient: .....	13
	• Ensure that patients and doctors can only change the attributes that not is ssn, email or id .....	13
2.4.6	Showing booking calendar of doctor in the future: .....	14
	• Ensure that the booking calendar always show the future day, and cannot change the report in the past .....	14
<b>3</b>	<b>User Groups and Set Application-level Permissions .....</b>	<b>15</b>
<b>4</b>	<b>Application Architecture .....</b>	<b>19</b>
4.1	Presentation Layer (UI Layer): .....	19
4.2	Application Layer: .....	19
4.3	Database Layer: .....	21
<b>5</b>	<b>Application Specifications .....</b>	<b>22</b>
5.1	Key Modules .....	22
5.2	Non-Functional Requirements .....	22
5.3	Security and Compliance .....	23
5.4	Deliverables .....	23
5.5	Technology Stack .....	23
5.6	System Design .....	23
5.6.1	Frontend (Presentation Layer) .....	24
5.6.2	Backend (Business Logic Layer) .....	24
5.6.3	Database Layer .....	24
5.6.4	Version Control and Collaboration .....	24
5.7	Advantages of the Technology Stack .....	25
5.8	Development Workflow .....	25
<b>6</b>	<b>BCNF normalization for the database schema .....</b>	<b>26</b>
<b>7</b>	<b>Compare Data between website and database when insert, delete or edit .....</b>	<b>27</b>
7.1	Insert .....	27
7.2	Edit .....	29
7.3	Delete .....	Error! Bookmark not defined.
<b>8</b>	<b>Indexing .....</b>	<b>31</b>
8.1	Setup and Table Creation .....	31
8.2	Query To find time using an Index .....	32
	The following code execute and the time difference between using index and no using index is shown below .....	32



<b>8.3</b>	<b>Results Comparison .....</b>	<b>33</b>
<b>8.4</b>	<b>Python Automation .....</b>	<b>34</b>
<b>8.5</b>	<b>Conclusion .....</b>	<b>35</b>

## 1 Member list & Workload

No.	Fullname	Student ID	Task	Contribution
1	Tran Tri Dung	2252133	Define user groups and permissions, define app architecture, Do functions, procedures, triggers and assertions, define detailed specifications	100%
2	Nguyen le Duy khang	2252303	Frontend development, normalize schema to BCNF, create data for application	100%
3	Huynh Ngoc Khoa	2211591	Backend development, indexing, Do functions, procedures, triggers and assertions, define detailed specifications	100%

## 2 Triggers

### 2.1 Prevent Patient Deletion

- Purpose: Prevent the deletion of a patient who has existing appointments.

```
CREATE TRIGGER PreventPatientDeletion
BEFORE DELETE ON Patient
FOR EACH ROW
BEGIN
    DECLARE appointment_count INT;
    SELECT COUNT(*) INTO appointment_count
    FROM Appointment
    WHERE ID_Patient = OLD.ID_Patient;

    IF appointment_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot delete patient with existing
appointments.';
    END IF;
END;
```

### 2.2 After Appointment Insert

- Purpose: Automatically create an invoice when a new appointment is added.

```
CREATE TRIGGER AfterAppointmentInsert
AFTER INSERT ON Appointment
FOR EACH ROW
BEGIN
    INSERT INTO Invoice (ID_Patient, Amount, IssueDate)
    VALUES (NEW.ID_Patient, 100.00, CURDATE());
END;
```

### 2.3 Trg\_insert\_user

- Purpose: Automatically insert a new user into the Doctor or Patient table based on their role.

```
CREATE OR REPLACE TRIGGER trg_insert_user
AFTER INSERT ON Users
FOR EACH ROW
BEGIN
    IF UPPER(:NEW.role) = 'DOCTOR' THEN
        INSERT INTO Doctor (ID, Speciality)
        VALUES (:NEW.id, NULL);
    ELSIF UPPER(:NEW.role) = 'PATIENT' THEN
        INSERT INTO Patient (ID)
        VALUES (:NEW.id);
    END IF;
END;
```

```
END IF;  
END;  
/
```

## 2.4 Trg\_update\_doctor\_speciality

Purpose: Automatically update a doctor's specialty based on the information added to the Doctor\_Assigned table.

```
CREATE OR REPLACE TRIGGER trg_update_doctor_speciality  
AFTER INSERT ON Doctor_Assigned  
FOR EACH ROW  
DECLARE  
    dept_speciality VARCHAR2(100);  
    current_speciality VARCHAR2(4000);  
BEGIN  
    SELECT Specialty INTO dept_speciality  
    FROM Department  
    WHERE ID = :NEW.Department_ID;  
    SELECT Specialty INTO current_speciality  
    FROM Doctor  
    WHERE ID = :NEW.Doctor_ID;  
  
    IF current_speciality IS NULL THEN  
        UPDATE Doctor  
        SET Specialty = dept_speciality  
        WHERE ID = :NEW.Doctor_ID;  
    ELSIF INSTR(current_speciality, dept_speciality) = 0 THEN  
        UPDATE Doctor  
        SET Specialty = current_speciality || ', ' || dept_speciality  
        WHERE ID = :NEW.Doctor_ID;  
    END IF;  
END;  
/
```

**Display the number of patient and doctors before proceeding**



Admin

Show Patients

Show Doctors

Show New Doctors

Log out

## All Patients

Submit

ID	Name	SSN	Email	Gender	DOB	RegistrationDate
28	Huynh Ngoc Khoa1	0482040002	huynhkhoa340@gmail.com	FEMALE	2004-01-03	2024-12-12
27	Huynh Tea	0000000001	tea1@gmail.com	MALE	2000-01-01	2000-01-01
63	User_2	0000000002	user_2@example.com	OTHER	2024-12-11	2024-12-11
62	User_1	0000000001	user_1@example.com	FEMALE	2024-12-12	2024-12-12
64	User_3	0000000003	user_3@example.com	MALE	2024-12-10	2024-12-10
65	User_4	0000000004	user_4@example.com	FEMALE	2024-12-09	2024-12-09
66	User_5	0000000005	user_5@example.com	OTHER	2024-12-08	2024-12-08
67	User_6	0000000006	user_6@example.com	MALE	2024-12-07	2024-12-07
68	User_7	0000000007	user_7@example.com	FEMALE	2024-12-06	2024-12-06
					2024-	

Admin

Show Patients

Show Doctors

Show New Doctors

Log out

## All Doctors

ID	Name	SSN	Email	Gender	DOB	RegistrationDate
29	Le Duc Nghia	0000000002	nghia@gmail.com	MALE	2000-01-01	2000-01-01
30	Le Duc Khang	0000000003	kahng@gmail.com	MALE	2000-01-01	2000-01-01
44	Huynh Ngoc Khang	0482040001	huynhkhoa240@gmail.com	FEMALE	2004-01-03	2024-12-12
46	Huynh Ngoc Khoa2	0482040000	huynhkhoa140@gmail.com	MALE	2004-01-03	2024-12-12

localhost:8000/admin.html?email=admin@gmail.com#

Figure 1 & 2: The number of patient and doctor on a date 13/12/2024



### Try adding a patient that already has existing email

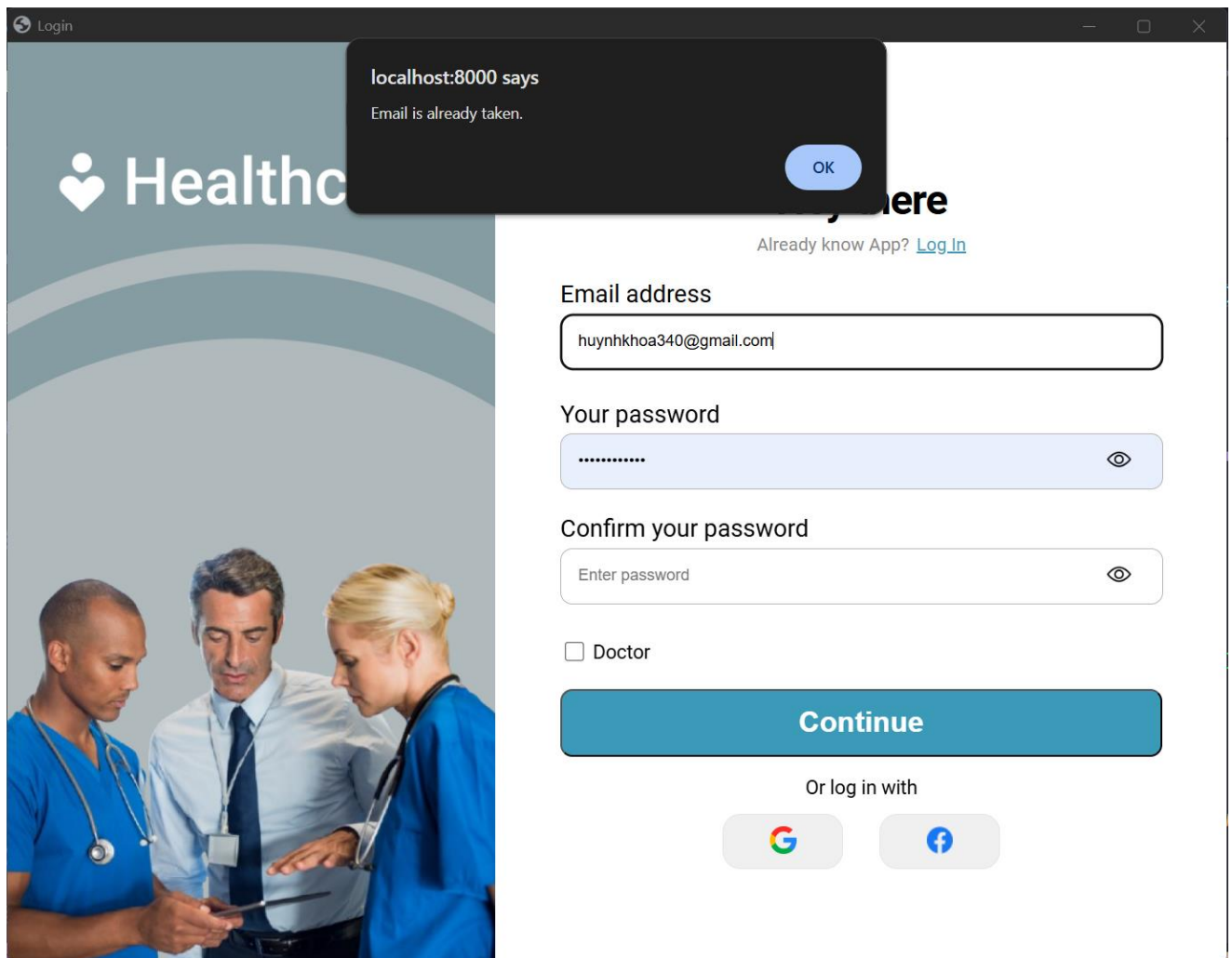
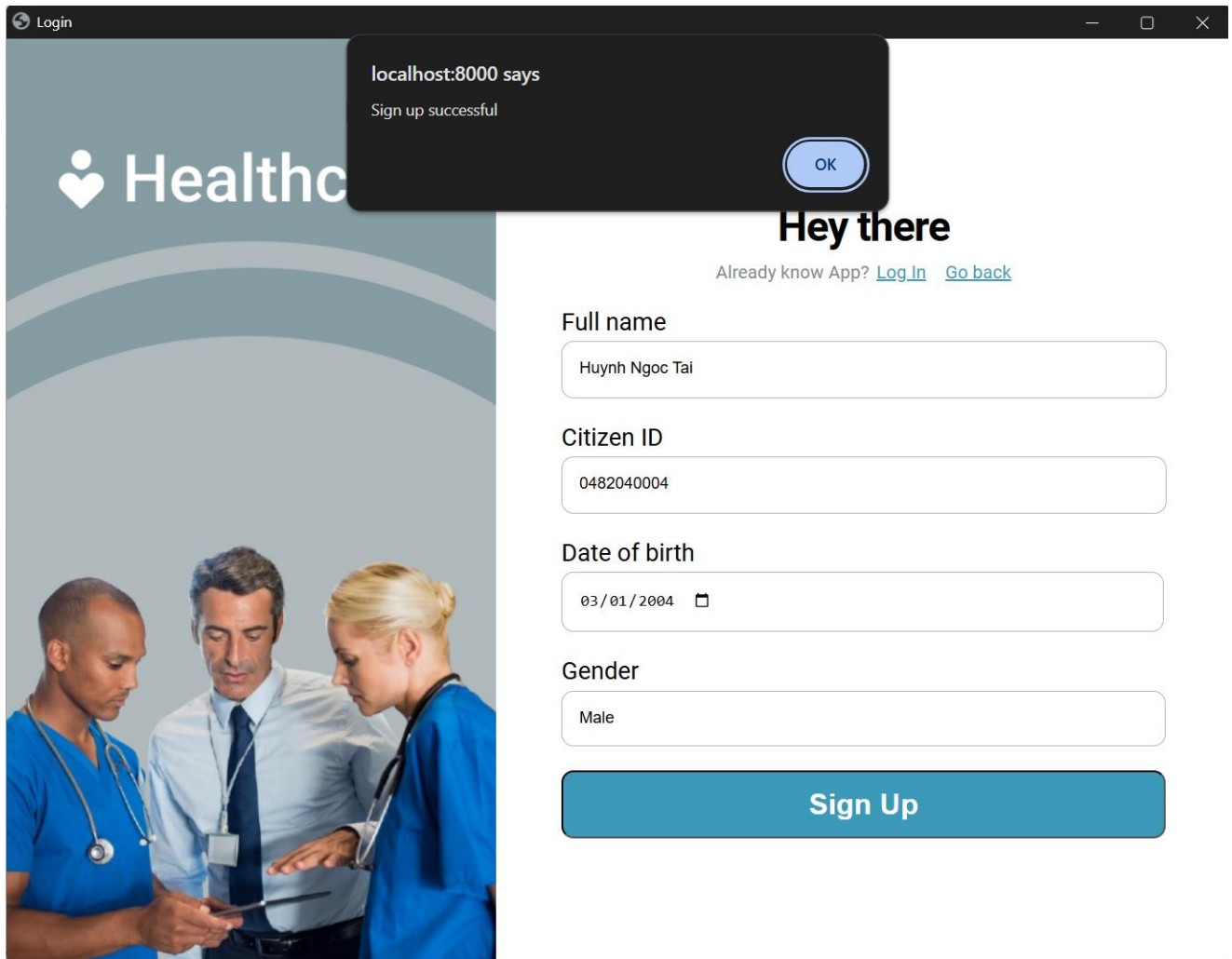


Figure 3: Error when registering account that have existing email

### Add a patient to database



The screenshot shows a web application interface for adding a new patient to a database. The interface is displayed within a browser window titled "Login".

**Success Message:** A dark overlay box displays the message "localhost:8000 says Sign up successful" with an "OK" button.

**Registration Form:**

- Header:** "Healthc" (partially visible) and "Hey there".
- Links:** "Already know App? [Log In](#) [Go back](#)".
- Fields:**
  - Full name:** Huynh Ngoc Tai
  - Citizen ID:** 0482040004
  - Date of birth:** 03/01/2004
  - Gender:** Male
- Button:** "Sign Up"

Figure 4: Adding new patient to database

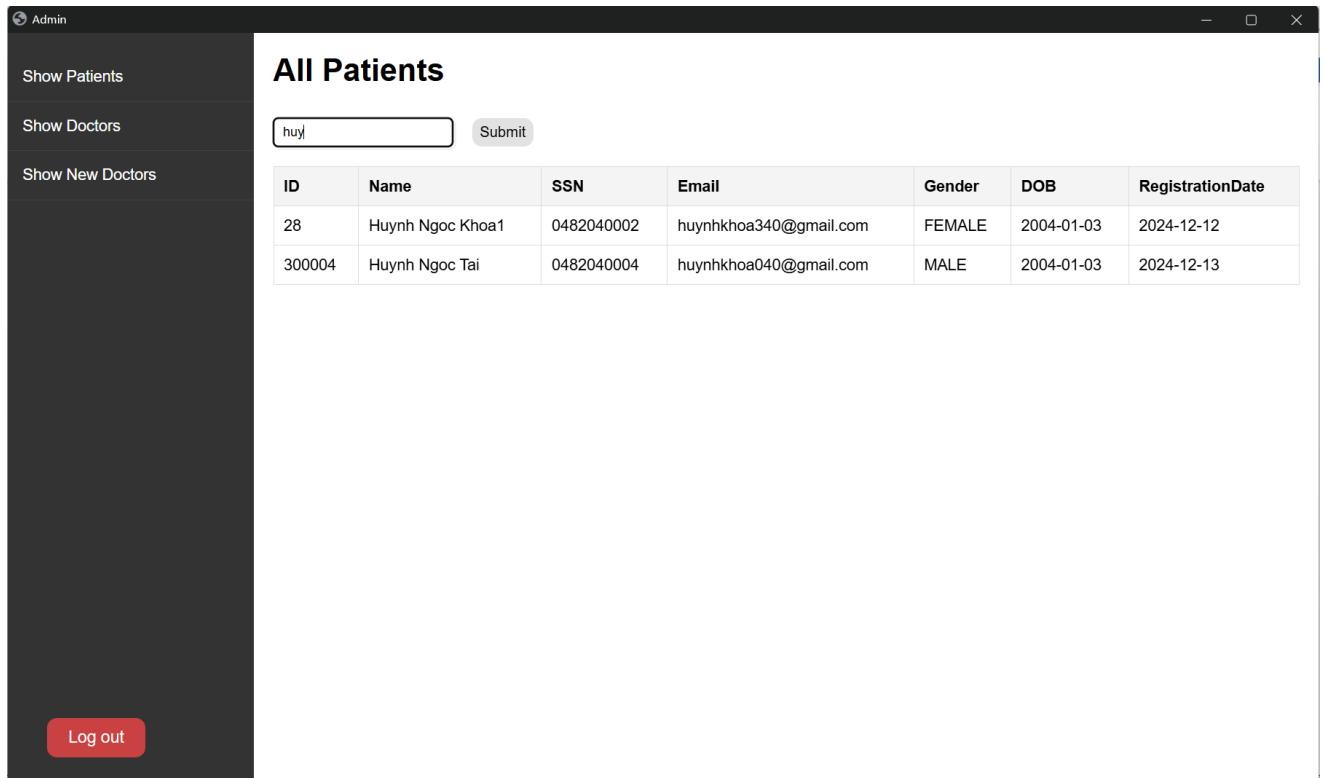


Figure 5: Patient is add to database that is showed by admin account

#### 2.4.2 Check\_email()

- Ensure that email is not registered in database

```
@eel.expose
def check_email(email):
    conn = connect_db()
    if conn:
        try:
            cursor = conn.cursor()
            query = """
            SELECT COUNT(*) from users where email = :email
            """
            print(email)
            cursor.execute(query, (email,))
            res = cursor.fetchone()
```

```
print(res[0])
if res:
    return res[0]
else:
    return 0
except oracledb.DatabaseError as e:
    print("Error during retrieve:", e)
    return str(e)
finally:
    cursor.close()
    conn.close()
else:
    return "Failed to connect to Oracle"
```

#### 2.4.3 Booking appointment based on department:

- Patient choose the department suitable for their demands and the system will choose the doctor have the free time suitable to patient.

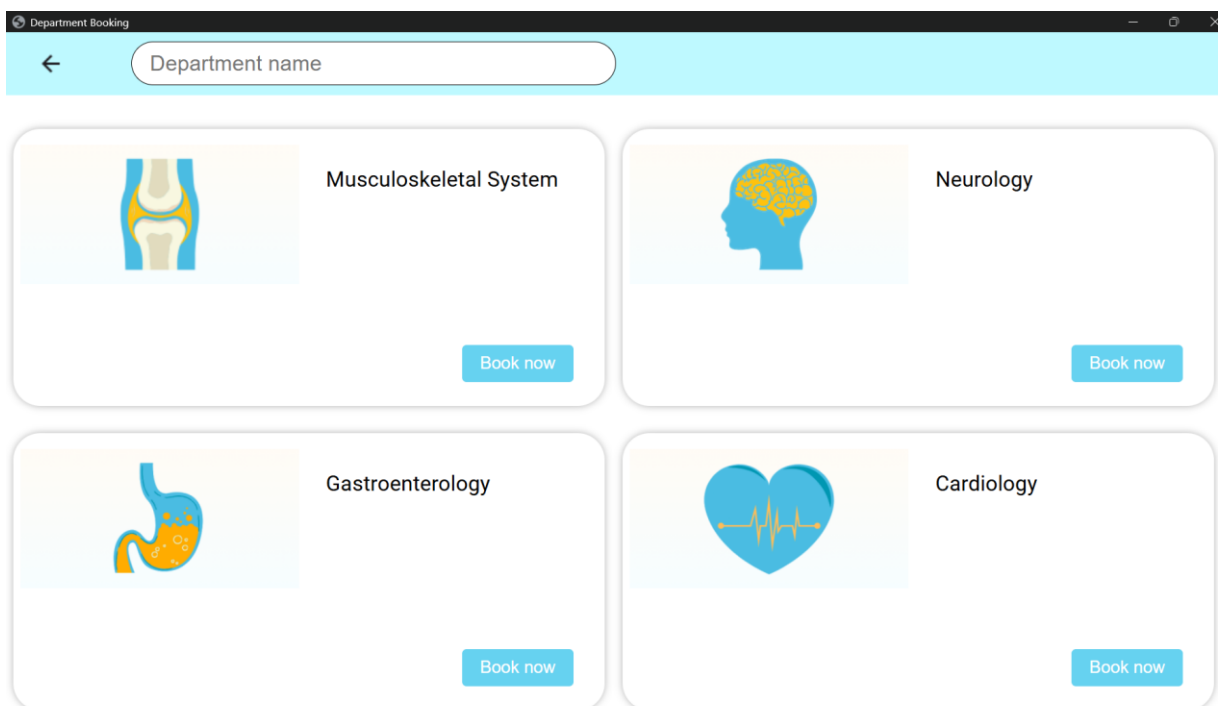


Figure 6: Finding department

←

Booking details

Patient information

Full name:

Citizen identification number:

Huynh Ngoc Khoa1

0482040002

Gender:

FEMALE

Date of birth:

2004-01-03

Schedule

Choose date:

13/12/2024

Submit

Chosen date: Friday 13 December 2024

AM

PM

10:00-10:10

10:10-10:20

10:20-10:30

10:30-10:40

10:40-10:50

10:50-11:00

11:00-11:10

11:10-11:20

11:20-11:30

11:30-11:40

11:40-11:50

11:50-12:00

Confirm

Figure 7: Choosing date

←

Booking details

localhost:8000 says  
Booking success!

OK

Patient information

Full name:

Citizen identification number:

Huynh Ngoc Khoa1

0482040002

Gender:

FEMALE

Date of birth:

2004-01-03

Schedule

Choose date:

13/12/2024

Submit

Chosen date: Friday 13 December 2024

AM

PM

10:00-10:10

10:10-10:20

10:20-10:30

10:30-10:40

10:40-10:50

10:50-11:00

11:00-11:10

11:10-11:20

11:20-11:30

11:30-11:40

11:40-11:50

11:50-12:00

Confirm

Figure 8: Confirm booking success

#### 2.4.4 Booking appointment based on doctor

- Retrieve to the database to get the free time of the doctor in that day, after that show to the patient time that doctor is free

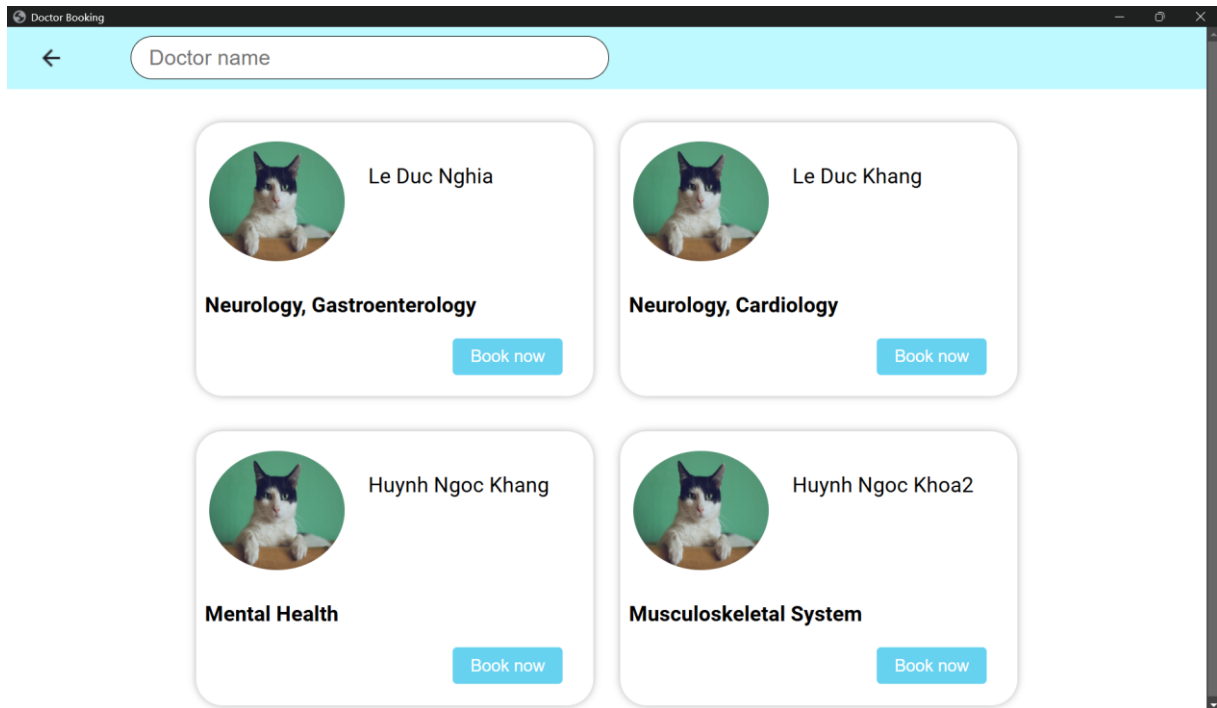


Figure 9: Choosing doctor

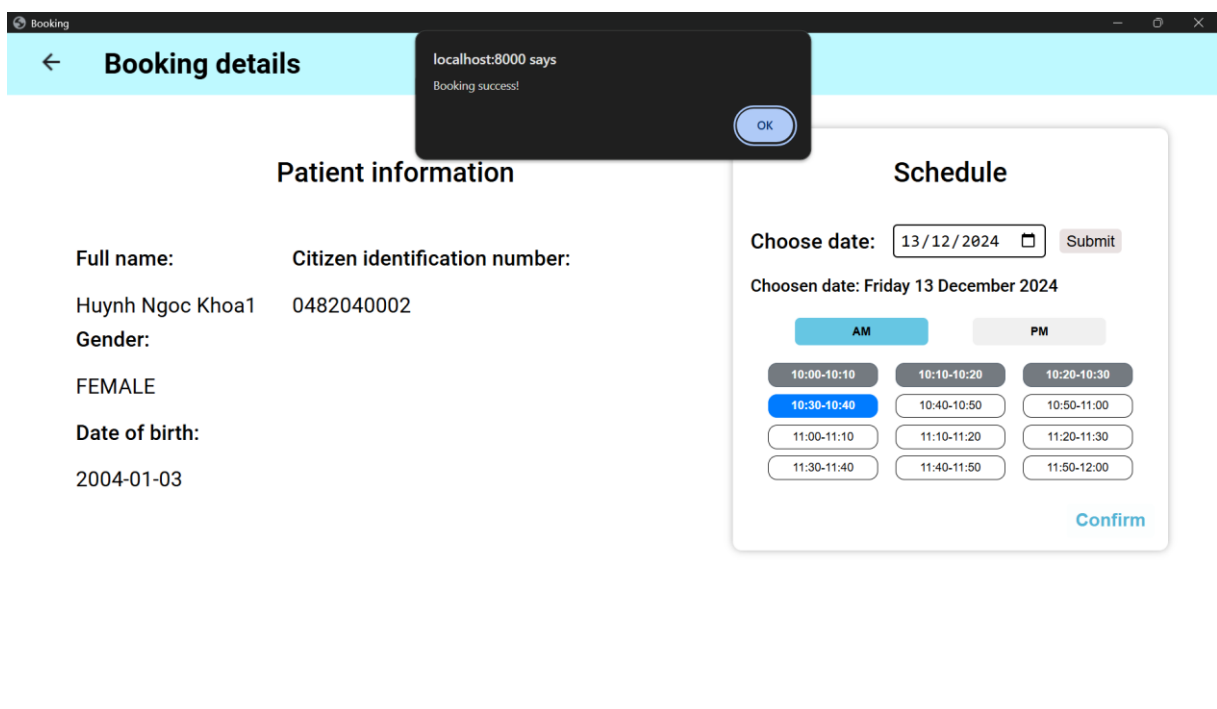


Figure 10: Confirm notification that success

#### 2.4.5 Update Information of Patient:

- Ensure that patients and doctors can only change the attributes that not is ssn, email or id

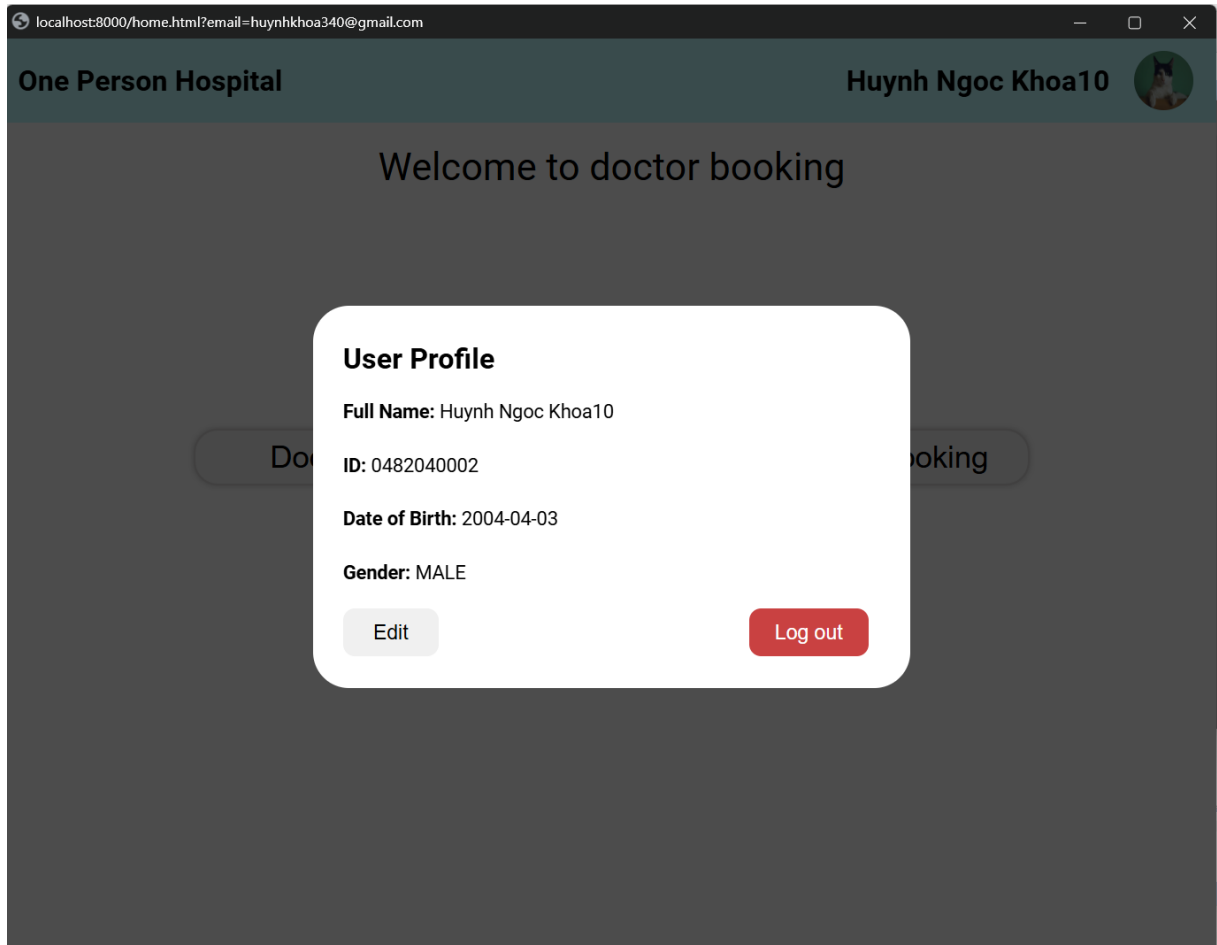


Figure 11: Editing profile

#### 2.4.6 Showing booking calendar of doctor in the future:

- Ensure that the booking calendar always show the future day, and cannot change the report in the past

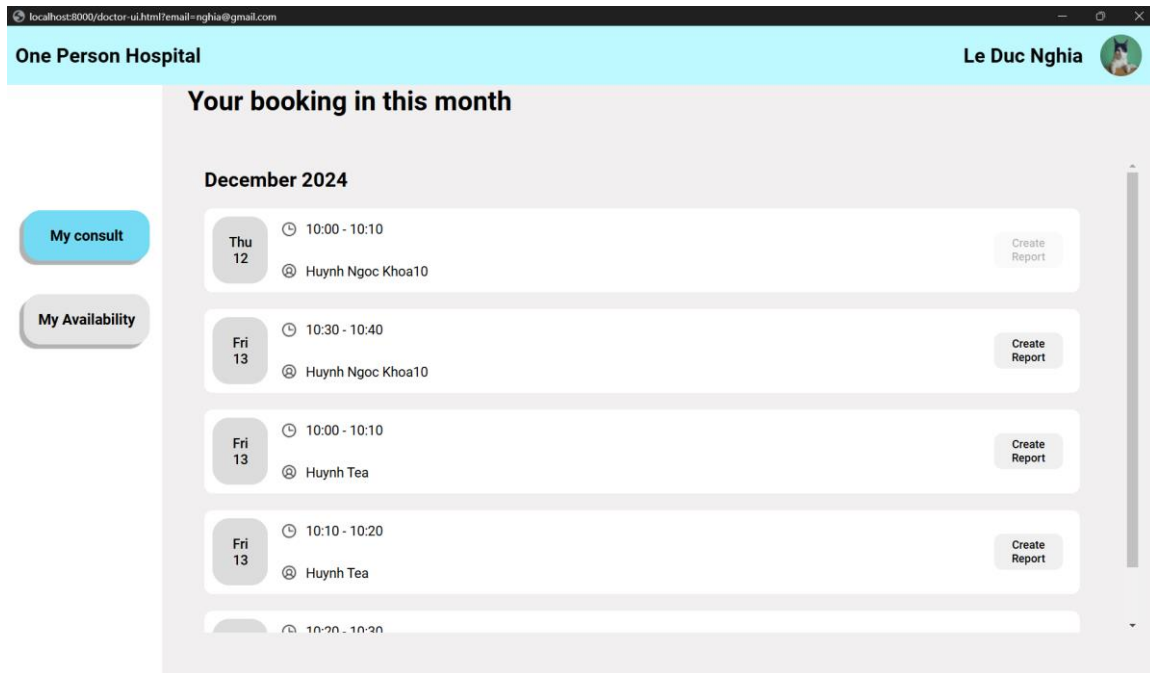


Figure 12: Doctor ui show booked appointment

### 3 User Groups and Set Application-level Permissions

User Group	Application-level Permissions
Admin	Manage entire system
Doctor	Handle patient records. They can see the list of all patients, all doctors, all departments, their personal information and appointments.
Patient	Access personal health information, personal information. They can see list of doctors, departments, their own appointments and can book an appointment.

Table 1: User Groups and Application-level Permissions

```
CREATE TABLE Department (
    ID INT PRIMARY KEY,
    Speciality VARCHAR(100)
);

CREATE Table Users (
    id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
```



```
SSN          VARCHAR(10) NOT NULL UNIQUE,
email        VARCHAR(40) NOT NULL,
password     VARCHAR(20) NOT NULL,
gender       VARCHAR(10) not null check (gender in ('MALE', 'FEMALE',
'OTHER')),
DOB          DATE,
role         VARCHAR(20) not null check (role in ('ADMIN','DOCTOR',
'PATIENT')),
registrationDate DATE
);

CREATE TABLE Doctor (
    ID INT PRIMARY KEY,
    Speciality VARCHAR(100),
    CONSTRAINT fk_doctor_id FOREIGN KEY (ID)
    REFERENCES Users(id)
    ON DELETE CASCADE DEFERRABLE
);

CREATE TABLE Doctor_Assigned (
    Doctor_ID INT NOT NULL,
    Department_ID INT NOT NULL,
    PRIMARY KEY (Doctor_ID, Department_ID),
    CONSTRAINT fk_doc_dep_doctor FOREIGN KEY (Doctor_ID)
        REFERENCES Doctor(ID) ON DELETE CASCADE DEFERRABLE,
    CONSTRAINT fk_doc_dep_department FOREIGN KEY (Department_ID)
        REFERENCES Department(ID) ON DELETE CASCADE DEFERRABLE
);

CREATE TABLE Room (
    ID INT PRIMARY KEY,
    Capacity INT,
    Room_type VARCHAR(10) not null check (Room_type in ('NORMAL',
'DELUXE', 'PRIVATE')),
```

```
price DECIMAL(10, 2)
);

CREATE TABLE Patient (
    ID INT PRIMARY KEY,
);

CREATE TABLE Patient_admission (
    Patient_ID INT,
    Room_ID INT,
    Date_start DATE,
    Date_end DATE,
    CONSTRAINT fk_patient_admission_pa_id FOREIGN KEY (Patient_ID)
        REFERENCES Patient(ID) ON DELETE SET NULL DEFERRABLE,
    CONSTRAINT fk_patient_admission_room_id FOREIGN KEY (Room_ID)
        REFERENCES Room(ID) ON DELETE SET NULL DEFERRABLE
);

CREATE TABLE Appointment (
    ID INT,
    Date_regis DATE,
    Time_regis TIMESTAMP,
    Patient_id INT,
    Doctor_id INT,
    PRIMARY KEY(ID, Patient_id, Doctor_id),
    CONSTRAINT fk_app_pa_ssn FOREIGN KEY (Patient_id)
        REFERENCES Patient(ID)
        ON DELETE SET NULL DEFERRABLE,
    CONSTRAINT fk_app_doc_ssn FOREIGN KEY (Doctor_id)
        REFERENCES Doctor(ID)
        ON DELETE SET NULL DEFERRABLE
);
```

```
CREATE TABLE Treatment (  
    ID INT,  
    Patient_id INT,  
    Doctor_id INT,  
    Date_prescribed DATE,  
    Diagnosis VARCHAR(100),  
    Invoice INT,  
    Feedback VARCHAR(200),  
    CONSTRAINT fk_treat_doc FOREIGN KEY (Doctor_id)  
    REFERENCES Doctor(ID)  
    ON DELETE SET NULL DEFERRABLE,  
    CONSTRAINT fk_treat_pa FOREIGN KEY (Patient_id)  
    REFERENCES Patient(ID)  
    ON DELETE SET NULL DEFERRABLE  
);  
  
CREATE Table Dosage_Treatment (  
    ID INT,  
    Patient_id INT,  
    Doctor_id INT,  
    Medical_name VARCHAR(10),  
    Medical_dosage VARCHAR(10),  
    CONSTRAINT fk_do_treat_doc FOREIGN KEY (Doctor_id)  
    REFERENCES Doctor(ID)  
    ON DELETE SET NULL DEFERRABLE,  
    CONSTRAINT fk_do_treat_pa FOREIGN KEY (Patient_id)  
    REFERENCES Patient(ID)  
    ON DELETE SET NULL DEFERRABLE  
);  
  
CREATE TABLE Medical_Report (  
    ID INT,  
    Diagnosis VARCHAR(100),  
    Patient_id INT,
```

```
PRIMARY KEY(ID, Patient_id),  
CONSTRAINT fk_med_pa_id FOREIGN KEY (Patient_id)  
REFERENCES Patient(ID)  
ON DELETE SET NULL DEFERRABLE  
);
```

## 4 Application Architecture

### 4.1 Presentation Layer (UI Layer):

The frontend layer where we interact with users (patients, doctors, admins). Here we are using technologies like Python, Oracle, HTML, CSS, JavaScript.

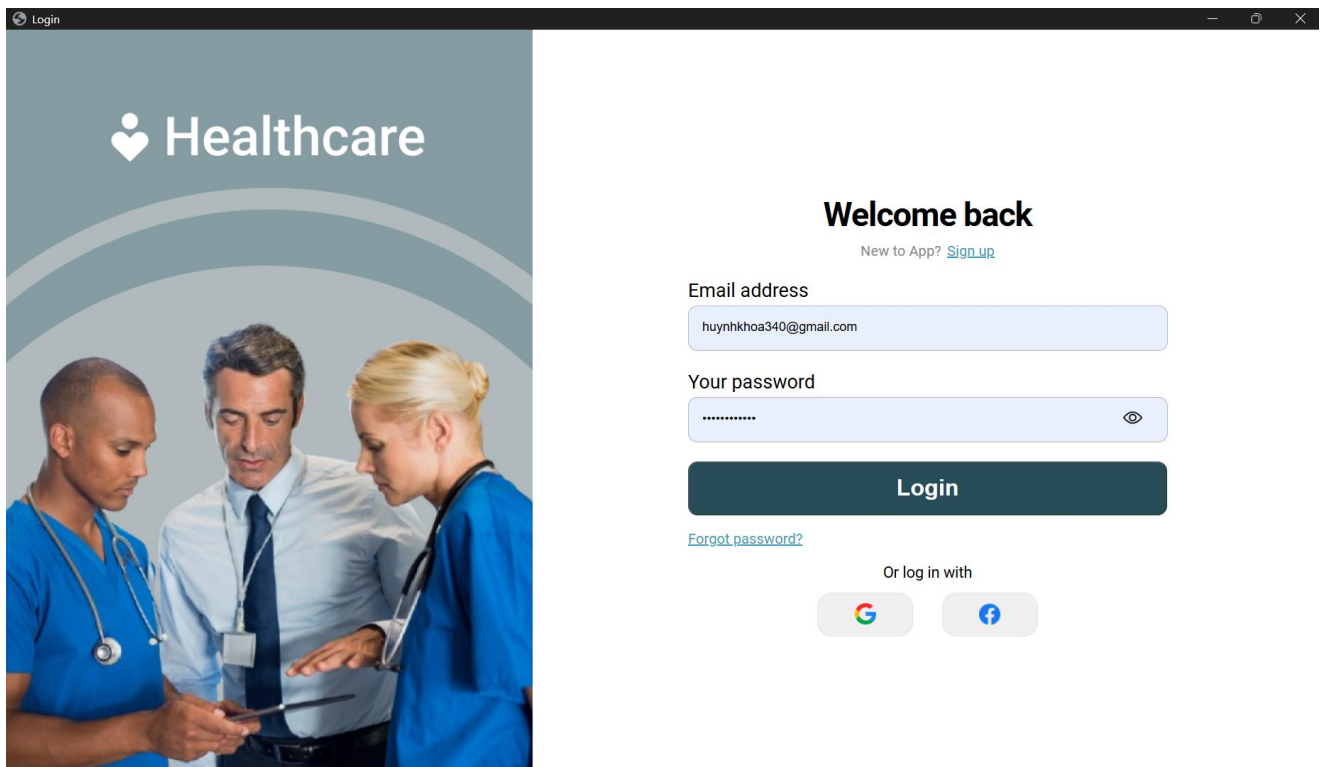
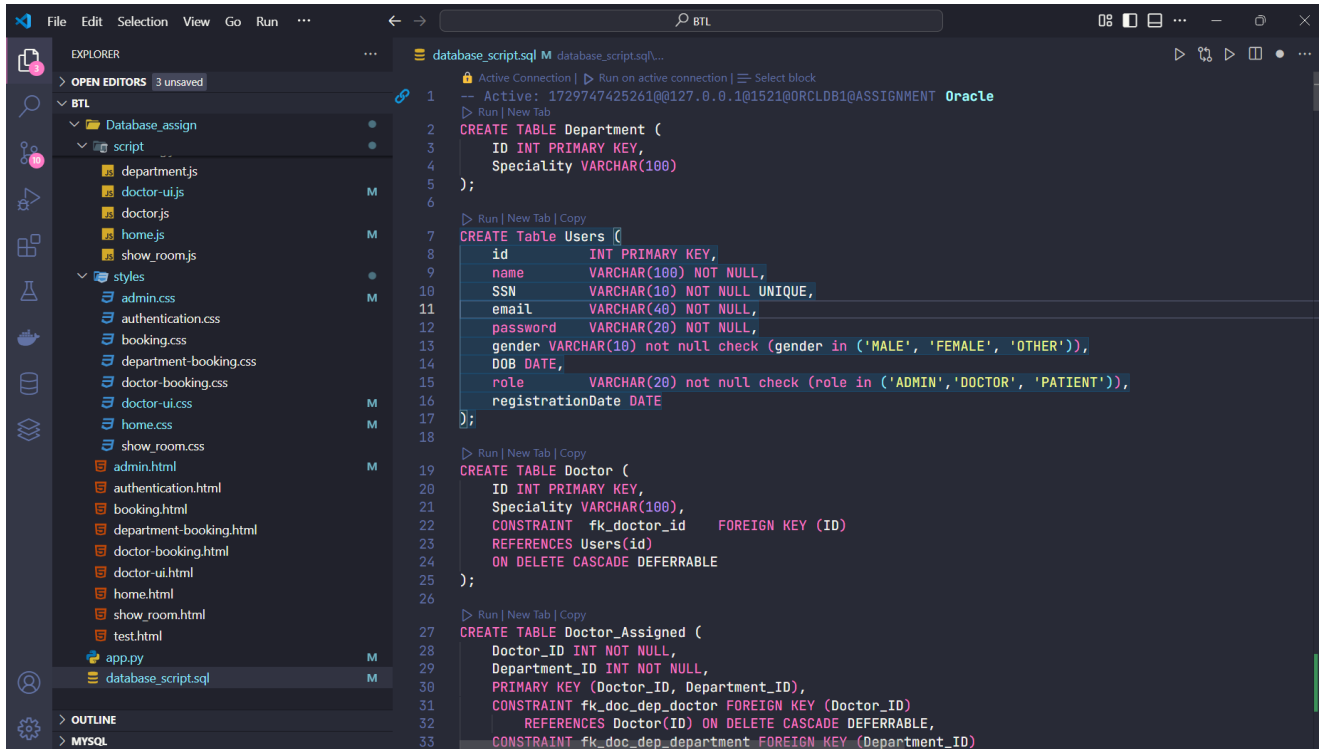


Figure 13: Presentation Layer

### 4.2 Application Layer:

The backend layer where we made logics of the system, navigations. Here we mainly use EEL which has Model View Templates architecture. Model represents the data that we want to present, which is from the database. View represents the request handler that returns the relevant templates and content. Template represents the HTML file (layout of the application).



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'Database\_assign', 'script', 'styles', and 'app.py'. The code editor shows the following SQL code:

```

1  -- Active: 172974742526100127.0.0.1@1521@ORCLDB1@ASSIGNMENT Oracle
2  -- Run | New Tab
3  CREATE TABLE Department (
4      ID INT PRIMARY KEY,
5      Speciality VARCHAR(100)
6  );
7
8  -- Run | New Tab | Copy
9  CREATE TABLE Users (
10     id INT PRIMARY KEY,
11     name VARCHAR(100) NOT NULL,
12     SSN VARCHAR(10) NOT NULL UNIQUE,
13     email VARCHAR(40) NOT NULL,
14     password VARCHAR(20) NOT NULL,
15     gender VARCHAR(10) not null check (gender in ('MALE', 'FEMALE', 'OTHER')),
16     DOB DATE,
17     role VARCHAR(20) not null check (role in ('ADMIN', 'DOCTOR', 'PATIENT')),
18     registrationDate DATE
19 );
20
21 -- Run | New Tab | Copy
22 CREATE TABLE Doctor (
23     ID INT PRIMARY KEY,
24     Speciality VARCHAR(100),
25     CONSTRAINT fk_doctor_id FOREIGN KEY (ID)
26     REFERENCES Users(id)
27     ON DELETE CASCADE DEFERRABLE
28 );
29
30 -- Run | New Tab | Copy
31 CREATE TABLE Doctor_Assigned (
32     Doctor_ID INT NOT NULL,
33     Department_ID INT NOT NULL,
34     PRIMARY KEY (Doctor_ID, Department_ID),
35     CONSTRAINT fk_doc_dep_doctor FOREIGN KEY (Doctor_ID)
36     REFERENCES Doctor(ID) ON DELETE CASCADE DEFERRABLE,
37     CONSTRAINT fk_doc_dep_department FOREIGN KEY (Department_ID)

```

Figure 14: Application Layer

### 4.3 Database Layer:

The layer where data is stored and triggers are created. We used Docker and VScode here for our application.

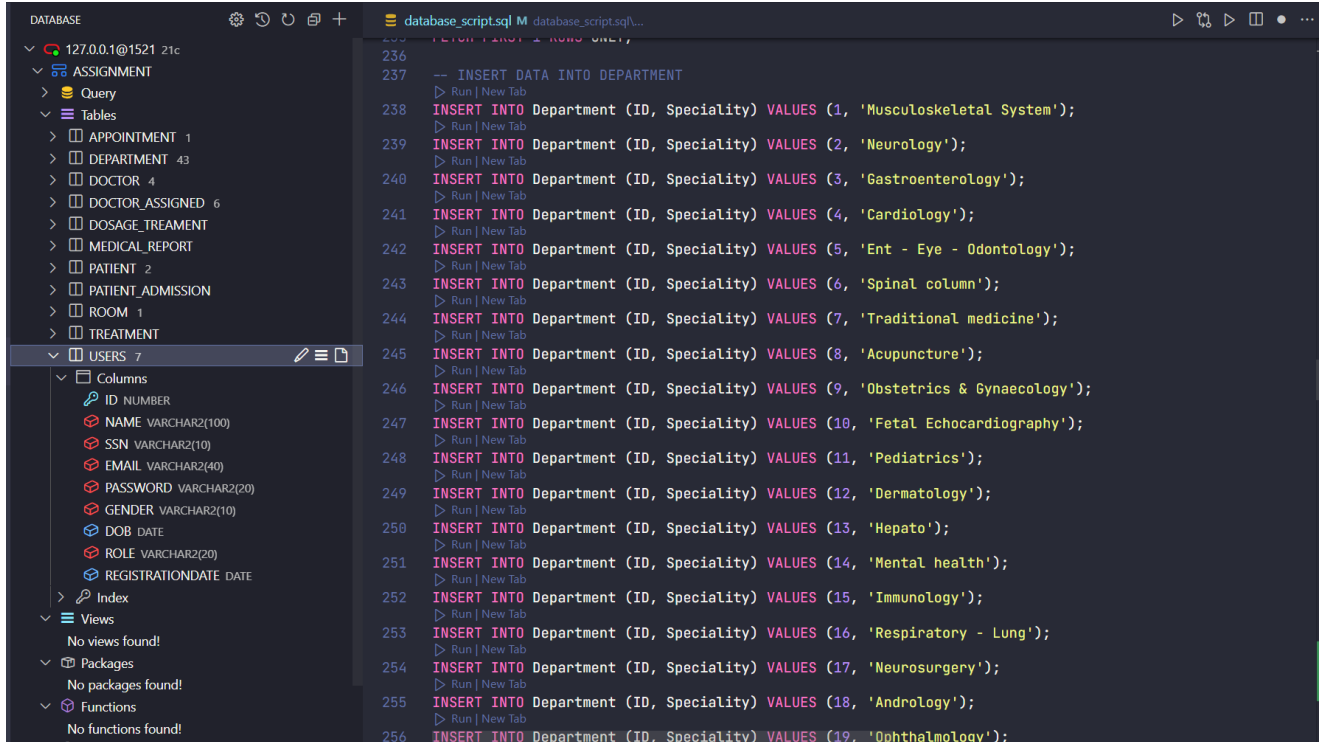


Figure 15: Database Layer

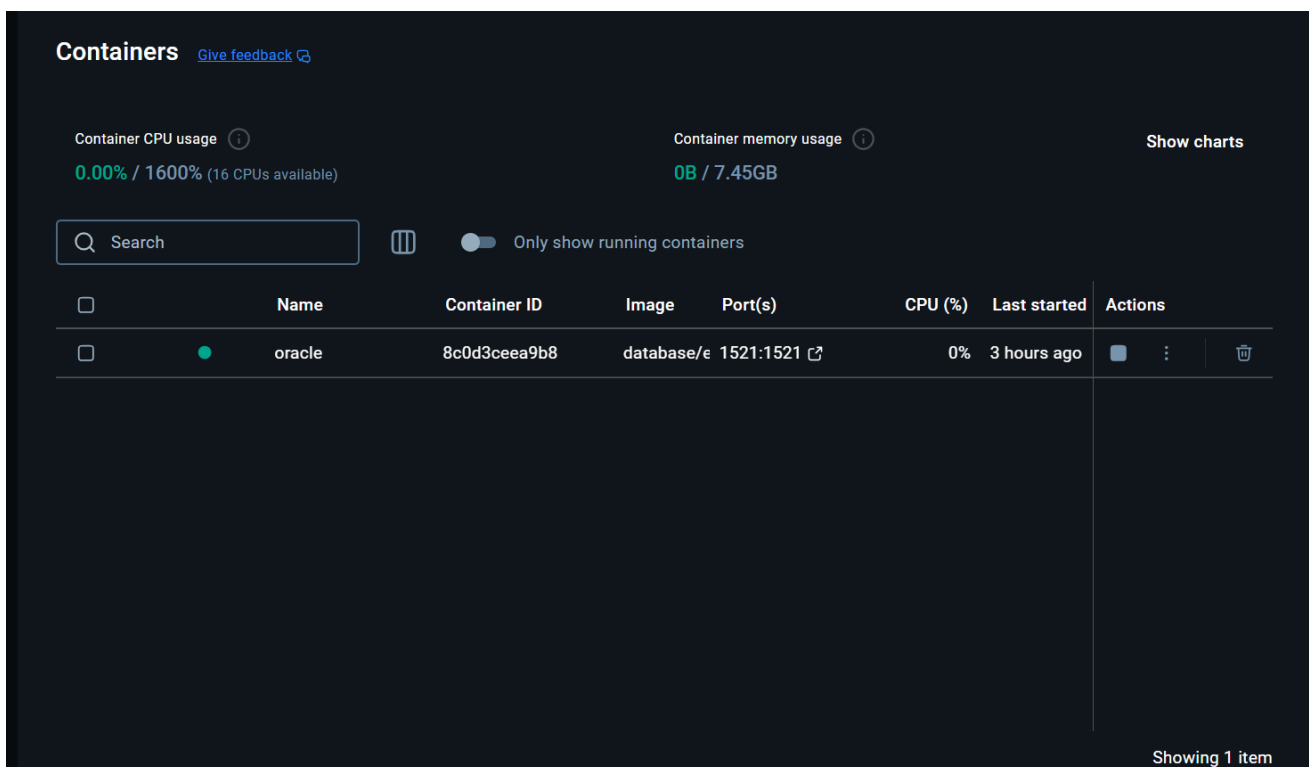


Figure 16: Docker server

## 5 Application Specifications

Hospital Management System (HMS) is a comprehensive application designed to manage hospital operations, including doctors, patient care, appointments and rooms. It ensures seamless interaction among various roles while maintaining data security and role-based access.

### 5.1 Key Modules

- **Patient Management:** Handles patient registration, health records, and personal details in a secure manner.
- **Department Management:** Allows administrators to manage hospital departments, including assigning heads and monitoring departmental activities.
- **Doctor Management:** Enables administrators to manage doctor profiles, while facilitating the assignment of doctors to departments.
- **Appointment Management:** Facilitates patient booking with preferred doctors for upcoming appointments.
- **Room Management:** Manages room assignments, tracks patient occupancy, and ensures efficient use of hospital resources.
- **Medical Records:** Maintains detailed records of diagnoses, test results and treatments for every patient.

### 5.2 Non-Functional Requirements

- **Security:** Implement user authentication and authorization to protect sensitive data.

- **Performance:** The application should handle up to 1000 concurrent users without significant performance degradation.
- **Usability:** The user interface should be intuitive and easy to navigate for all user types.
- **Scalability:** The system should be designed to accommodate future growth in user numbers and data volume.

### 5.3 Security and Compliance

- Implements secure authentication mechanisms.
- Ensures data privacy and compliance with healthcare regulations.

### 5.4 Deliverables

- Fully functioning HMS application.
- Documentation (user manual, API documentation).
- Test cases and test reports.
- Deployment scripts and configurations.

### 5.5 Technology Stack

The Hospital Management System (HMS) will be developed using the following technologies, ensuring a reliable and efficient system:

- **Programming Language:** Python
- **Framework:** EEL for python backend
- Eel bridges Python with web technologies, enabling Python code to handle business logic, database interactions, and computational tasks while delegating UI rendering to HTML, CSS, and JavaScript.
- Eel allows bi-directional communication between the Python backend and the JavaScript frontend.
- **Frontend:** HTML, CSS for creating a responsive and user-friendly interface
- HTML provides the structure of the UI, while JavaScript allows dynamic content rendering and real-time interaction.
- Adding JavaScript libraries (e.g., jQuery, Chart.js, or React) enhances functionality and responsiveness.
- **Database:** Vscod + docker to access oracle server for structured and scalable data management
- **Version Control:** Git for source code management and collaboration
- **Repository Hosting:** GitHub for centralized code storage and project management
- **Deployment:** Docker containers for consistency across development and production environments

### 5.6 System Design



The HMS application is designed using a layered architecture to enhance modularity and maintainability:

#### **5.6.1 Frontend (Presentation Layer)**

- Utilizes HTML, CSS, and EEL library to create interactive user interfaces.
- Offers role-based dashboards for administrators, doctors, and patients.
- Ensures cross-platform compatibility with responsive design.

#### **5.6.2 Backend (Business Logic Layer)**

- Developed using EEL, following the Model-View-Control (MVC) design pattern.
- Implements application logic, role-based access control, and session management.

#### **5.6.3 Database Layer**

- Uses Vscode with sqldeveloper to store and manage HMS data efficiently.
- Incorporates a well-structured schema with relationships, constraints, and optimized indexing.
- Utilizes EEL library in python to use cursor for database queries and transactions.

#### **5.6.4 Version Control and Collaboration**

- Employs Git for source code versioning, enabling effective tracking of changes and collaboration.
- Hosts the project repository on GitHub for centralized access and project management.
- Integrates GitHub Issues and Pull Requests for streamlined team workflows.

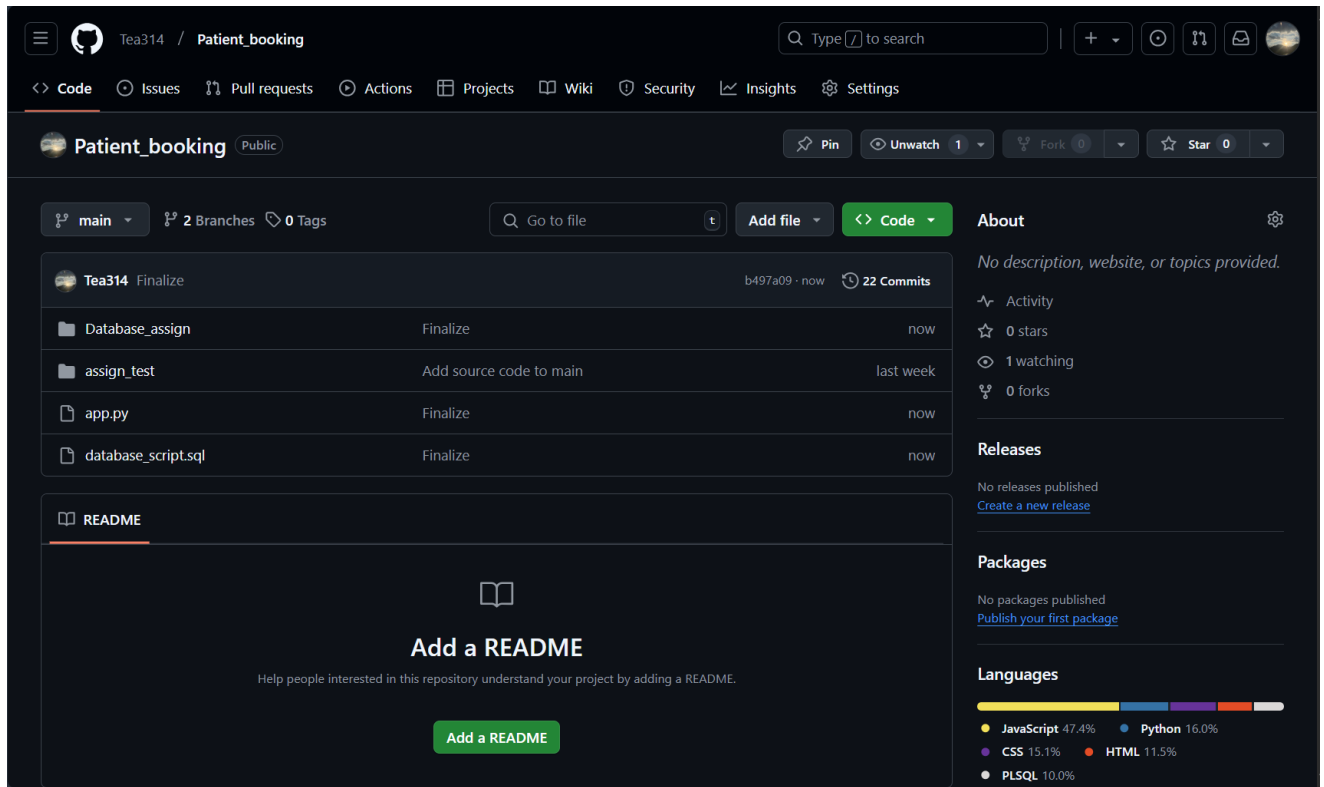


Figure 17: Project Github

## 5.7 Advantages of the Technology Stack

- **Python and Django:** Django's scalability and Python's simplicity accelerate development and maintenance.
- **HTML and CSS:** Enables interactive and responsive user interfaces.
- **Vscode with docker and sqldeveloper:** Ensures secure and efficient data storage and retrieval.
- **Git and GitHub:** Enhances collaboration, version control, and code review processes.

## 5.8 Development Workflow

The development process will follow an agile methodology, comprising the following stages:

1. Requirement Gathering and Prototyping
2. Database Design and Backend Development
3. Frontend Development and Integration
4. Version Control Management using Git and GitHub
5. Testing (Unit, Integration, and User Acceptance)
6. Deployment and Continuous Monitoring

## 6 BCNF normalization for the database schema

After thorough analysis of our relational schema, we have decided that they have met the requirement of BCNF. Below are the functional dependencies of all the relationships.

**User**(ID, Name, Gender, SSN, DOB, Role, Email, Password)

- ID → {Name, Gender, SSN, DOB, Role, Email, Password}

**Doctor**(ID, Specialty)

- ID → Specialty

**Department**(ID, Speciality)

- ID → Speciality

**Treatment**(ID, Doc\_ID, Paitent\_ID, Date, Diagnosis)

- {ID, Doc\_ID, Paitent\_ID} → {Date, Diagnosis}

**Patient**(ID)

**Room**(ID, Capacity, Room\_type, Price)

- ID → {Capacity, Room\_type, Price}

**Appointment**(Doc\_ID, Paitent\_ID, App\_ID, Date, Time)

- {Doc\_ID, Paitent\_ID, App\_ID} → {Date, Time}

**Medical\_Report**(Paitent\_ID, Med\_ID, Diagnosis)

- {Patient\_ID, Med\_ID} → Diagnosis

**Doc\_Assigned**(Doc\_ID, Dept\_ID)

**Dosage.Treatment**(Doc\_ID, Treat\_ID, Paitent\_ID, Medical\_name, Medical\_dosage)

- {Doc\_ID, Treat\_ID, Paitent\_ID} → {Medical\_name, Medical\_dosage}

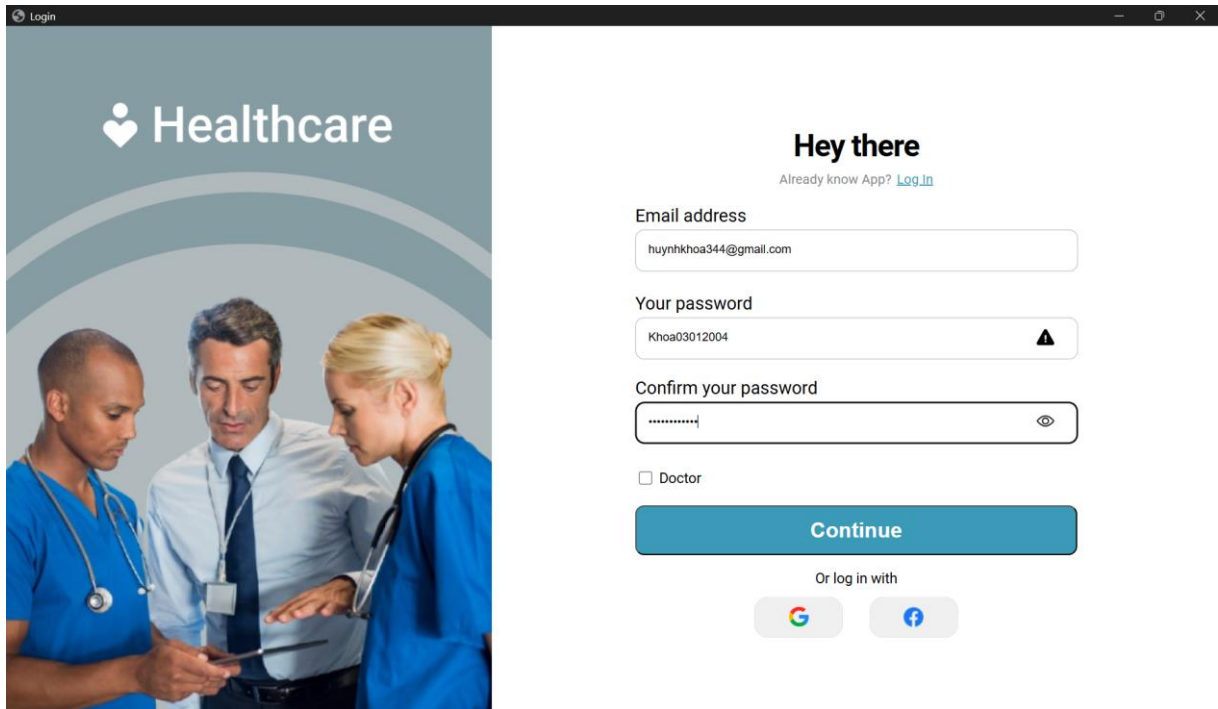
**Patient\_Admission**(Patient\_ID, Room\_ID, Date\_start, Date\_end, Total\_price)

- {Patient\_ID, Room\_ID} -> {Date\_start, Date\_end, Total\_price}

## 7 Compare Data between website and database when insert, delete or edit

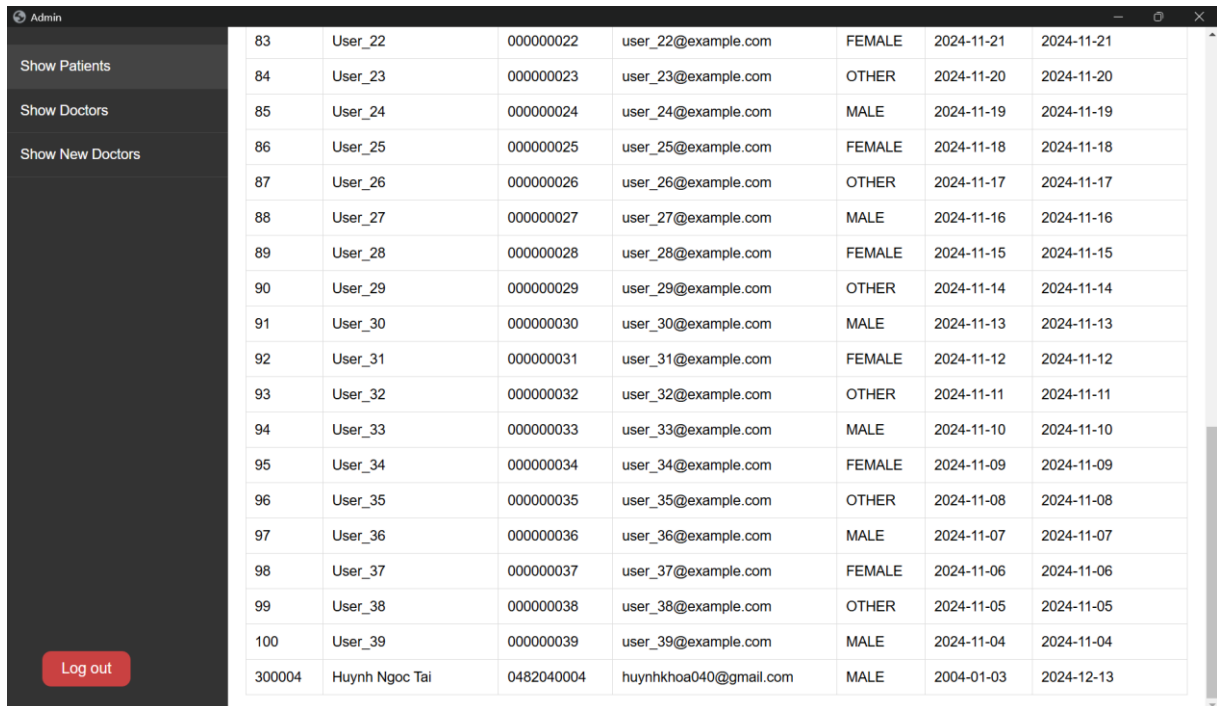
### 7.1 Insert

First, I inserted a new patient into our system:



The screenshot displays a web application interface for a healthcare system. On the left, there is a header with a heart icon and the word 'Healthcare' above a photograph of three medical professionals (two men and one woman) in blue scrubs looking at a tablet. On the right, the registration form is titled 'Hey there' with a link 'Already know App? Log In'. The form includes three input fields: 'Email address' containing 'huynhkhoea344@gmail.com', 'Your password' containing 'Khoa03012004' with a warning icon, and 'Confirm your password' which is masked with dots and has an eye icon. Below these fields is a checkbox labeled 'Doctor' which is unchecked. A large blue 'Continue' button is positioned below the checkbox. At the bottom, there is a section 'Or log in with' featuring two buttons with Google and Facebook logos.

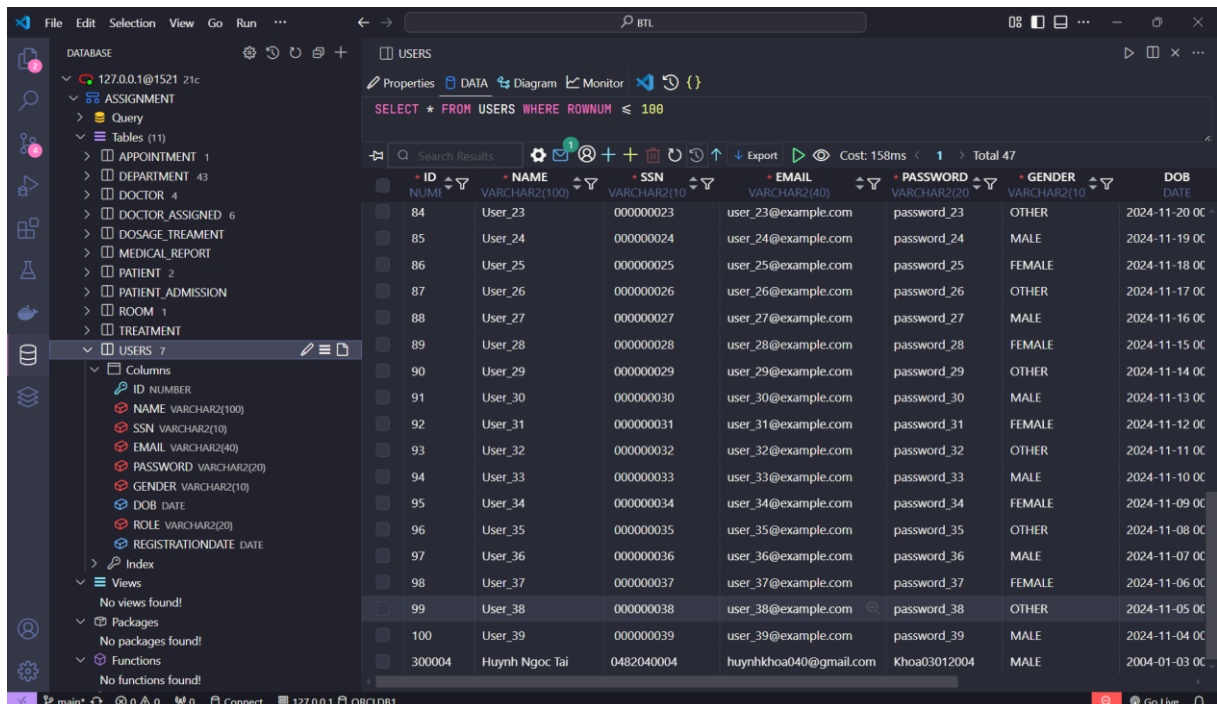
Figure 18: Register ui



83	User_22	000000022	user_22@example.com	FEMALE	2024-11-21	2024-11-21
84	User_23	000000023	user_23@example.com	OTHER	2024-11-20	2024-11-20
85	User_24	000000024	user_24@example.com	MALE	2024-11-19	2024-11-19
86	User_25	000000025	user_25@example.com	FEMALE	2024-11-18	2024-11-18
87	User_26	000000026	user_26@example.com	OTHER	2024-11-17	2024-11-17
88	User_27	000000027	user_27@example.com	MALE	2024-11-16	2024-11-16
89	User_28	000000028	user_28@example.com	FEMALE	2024-11-15	2024-11-15
90	User_29	000000029	user_29@example.com	OTHER	2024-11-14	2024-11-14
91	User_30	000000030	user_30@example.com	MALE	2024-11-13	2024-11-13
92	User_31	000000031	user_31@example.com	FEMALE	2024-11-12	2024-11-12
93	User_32	000000032	user_32@example.com	OTHER	2024-11-11	2024-11-11
94	User_33	000000033	user_33@example.com	MALE	2024-11-10	2024-11-10
95	User_34	000000034	user_34@example.com	FEMALE	2024-11-09	2024-11-09
96	User_35	000000035	user_35@example.com	OTHER	2024-11-08	2024-11-08
97	User_36	000000036	user_36@example.com	MALE	2024-11-07	2024-11-07
98	User_37	000000037	user_37@example.com	FEMALE	2024-11-06	2024-11-06
99	User_38	000000038	user_38@example.com	OTHER	2024-11-05	2024-11-05
100	User_39	000000039	user_39@example.com	MALE	2024-11-04	2024-11-04
300004	Huynh Ngoc Tai	0482040004	huynhkhao040@gmail.com	MALE	2004-01-03	2024-12-13

Figure 19: Result on website

Then, go to our database, we can check if there is a change in database:



ID	NAME	SSN	EMAIL	PASSWORD	GENDER	DOB
84	User_23	000000023	user_23@example.com	password_23	OTHER	2024-11-20 00:00:00
85	User_24	000000024	user_24@example.com	password_24	MALE	2024-11-19 00:00:00
86	User_25	000000025	user_25@example.com	password_25	FEMALE	2024-11-18 00:00:00
87	User_26	000000026	user_26@example.com	password_26	OTHER	2024-11-17 00:00:00
88	User_27	000000027	user_27@example.com	password_27	MALE	2024-11-16 00:00:00
89	User_28	000000028	user_28@example.com	password_28	FEMALE	2024-11-15 00:00:00
90	User_29	000000029	user_29@example.com	password_29	OTHER	2024-11-14 00:00:00
91	User_30	000000030	user_30@example.com	password_30	MALE	2024-11-13 00:00:00
92	User_31	000000031	user_31@example.com	password_31	FEMALE	2024-11-12 00:00:00
93	User_32	000000032	user_32@example.com	password_32	OTHER	2024-11-11 00:00:00
94	User_33	000000033	user_33@example.com	password_33	MALE	2024-11-10 00:00:00
95	User_34	000000034	user_34@example.com	password_34	FEMALE	2024-11-09 00:00:00
96	User_35	000000035	user_35@example.com	password_35	OTHER	2024-11-08 00:00:00
97	User_36	000000036	user_36@example.com	password_36	MALE	2024-11-07 00:00:00
98	User_37	000000037	user_37@example.com	password_37	FEMALE	2024-11-06 00:00:00
99	User_38	000000038	user_38@example.com	password_38	OTHER	2024-11-05 00:00:00
100	User_39	000000039	user_39@example.com	password_39	MALE	2024-11-04 00:00:00
300004	Huynh Ngoc Tai	0482040004	huynhkhao040@gmail.com	Khoa03012004	MALE	2004-01-03 00:00:00

Figure 20: Result in database

We can conclude that the result when inserting on website can affect to the database.

## 7.2 Edit

Subsequently, when there are some mistakes when recording patient information, for instance in this case, patient Khoa10 was born in 2000 not 2004, so I will fix it by editing on this website:

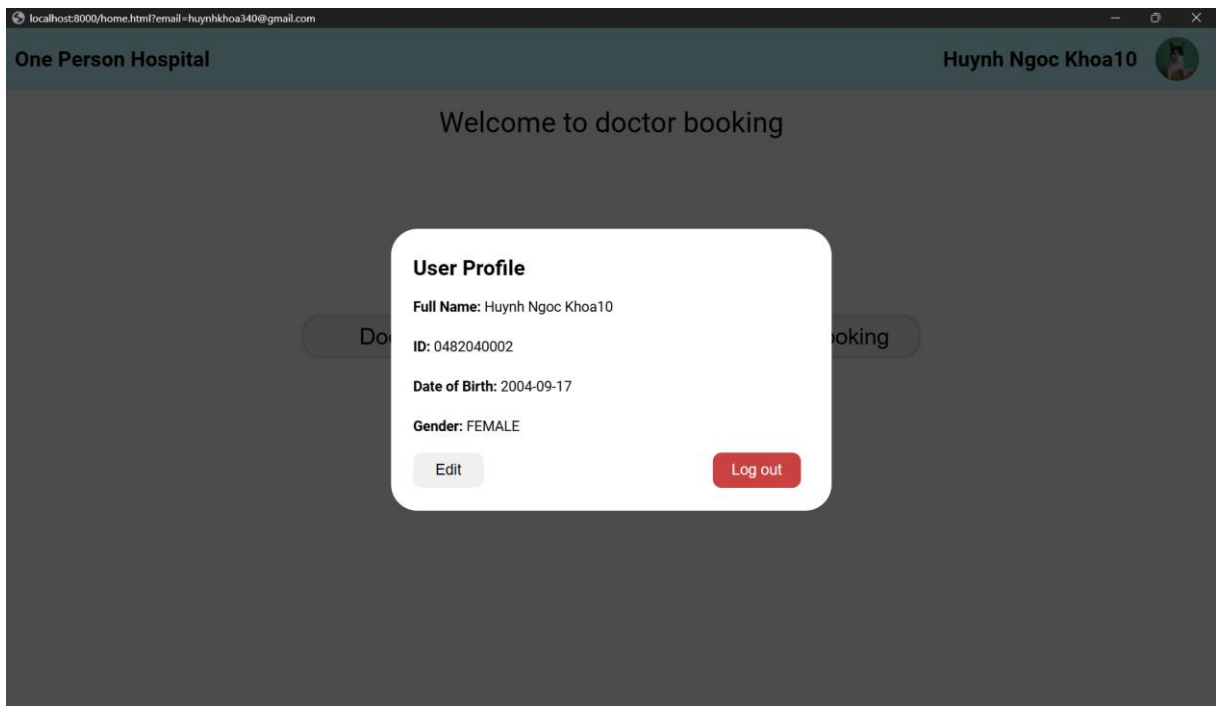


Figure 21: Edit patient information

Then this is the result on our website:



ID	Name	SSN	Email	Gender	DOB	RegistrationDate
28	Huynh	0482040002	huynhkhoea340@gmail.com	MALE	2000-09-17	2024-12-12
27	Huynh Tea	0000000001	tea1@gmail.com	MALE	2000-01-01	2000-01-01
63	User_2	0000000002	user_2@example.com	OTHER	2024-12-11	2024-12-11
62	User_1	0000000001	user_1@example.com	FEMALE	2024-12-12	2024-12-12
64	User_3	0000000003	user_3@example.com	MALE	2024-12-10	2024-12-10
65	User_4	0000000004	user_4@example.com	FEMALE	2024-12-09	2024-12-09
66	User_5	0000000005	user_5@example.com	OTHER	2024-12-08	2024-12-08
67	User_6	0000000006	user_6@example.com	MALE	2024-12-07	2024-12-07
68	User_7	0000000007	user_7@example.com	FEMALE	2024-12-06	2024-12-06
69	User_8	0000000008	user_8@example.com	OTHER	2024-12-05	2024-12-05
70	User_9	0000000009	user_9@example.com	MALE	2024-12-04	2024-12-04
71	User_10	0000000010	user_10@example.com	FEMALE	2024-12-03	2024-12-03
72	User_11	0000000011	user_11@example.com	OTHER	2024-12-02	2024-12-02
73	User_12	0000000012	user_12@example.com	MALE	2024-12-01	2024-12-01
74	User_13	0000000013	user_13@example.com	FEMALE	2024-11-30	2024-11-30

Figure 22: Result on admin

Go to the database, we can see:

ID	NAME	SSN	EMAIL	PASSWORD	GENDER	DOB
28	Huynh	0482040002	huynhkhoea340@gmail.com	Khoa03012004	MALE	2000-09-17 00
29	Le Duc Nghia	0000000002	nghia@gmail.com	Nghia12345	MALE	2000-01-01 00
26	admin	0000000000	admin@gmail.com	Admin12345	OTHER	2000-01-01 00
30	Le Duc Khang	0000000003	kahng@gmail.com	Khang12345	MALE	2000-01-01 00
27	Huynh Tea	0000000001	tea1@gmail.com	Khoa112345	MALE	2000-01-01 00
44	Huynh Ngoc Khang	0482040001	huynhkhoea240@gmail.com	Khoa03012004	FEMALE	2004-01-03 00
46	Huynh Ngoc Khoa2	0482040000	huynhkhoea140@gmail.com	Khoa03012004	MALE	2004-01-03 00
63	User_2	0000000002	user_2@example.com	password_2	OTHER	2024-12-11 00
62	User_1	0000000001	user_1@example.com	password_1	FEMALE	2024-12-12 00
64	User_3	0000000003	user_3@example.com	password_3	MALE	2024-12-10 00
65	User_4	0000000004	user_4@example.com	password_4	FEMALE	2024-12-09 00
66	User_5	0000000005	user_5@example.com	password_5	OTHER	2024-12-08 00
67	User_6	0000000006	user_6@example.com	password_6	MALE	2024-12-07 00
68	User_7	0000000007	user_7@example.com	password_7	FEMALE	2024-12-06 00
69	User_8	0000000008	user_8@example.com	password_8	OTHER	2024-12-05 00
70	User_9	0000000009	user_9@example.com	password_9	MALE	2024-12-04 00
71	User_10	0000000010	user_10@example.com	password_10	FEMALE	2024-12-03 00
72	User_11	0000000011	user_11@example.com	password_11	OTHER	2024-12-02 00

Figure 23: Result in database

We can conclude that the result when editing on website can affect to the database.



## 8 Indexing

The index structures are additional files on disk that provide secondary access paths, which provide alternative ways to access the records without affecting the physical placement of records in the primary data file on disk. They enable efficient access to records based on the indexing fields that are used to construct the index. Basically, any field of the file can be used to create an index, and multiple indexes on different fields - as well as indexes on multiple fields - can be constructed on the same file. A variety of indexes are possible; each of them uses a particular data structure to speed up the search.

The most prevalent types of indexes are based on ordered files (single-level indexes) and use tree data structures (multilevel indexes, B+-trees) to organize the index. Indexes can also be constructed based on hashing or other search data structures.

This section demonstrates the performance improvement achieved by implementing indexing in the Users table with a large dataset. We will compare the query execution time for a query without an index and with an index on the departmentID column.

### 8.1 Setup and Table Creation

The following table schema represents the Users table, which is used to store information about medical staff members, such as their ID, SSN, name, phone number, salary, and department affiliation.

```
CREATE Table Users (  
    id            INT PRIMARY KEY,  
    name          VARCHAR(100) NOT NULL,  
    SSN           VARCHAR(10) NOT NULL UNIQUE,  
    email         VARCHAR(40) NOT NULL,  
    password      VARCHAR(20) NOT NULL,  
    gender        VARCHAR(10) not null check (gender in ('MALE', 'FEMALE',  
'OTHER')),  
    DOB           DATE,  
    role          VARCHAR(20) not null check (role in ('ADMIN', 'DOCTOR',  
'PATIENT')),  
    registrationDate DATE  
);
```

We populate the table with a large number of rows for the demo. A sample population script might look like this:

```
BEGIN  
    FOR i IN 1..1000000 LOOP  
        INSERT INTO Users (id, name, ssn, email, password, gender, dob,  
role, registrationDate)  
            VALUES (
```



```
        i,  
        'User_' || i,  
        TO_CHAR(i, '0000000000'),  
        'user_' || i || '@example.com',  
        'password_' || i,  
        CASE MOD(i, 3)  
            WHEN 0 THEN 'MALE'  
            WHEN 1 THEN 'FEMALE'  
            ELSE 'OTHER'  
        END,  
        TRUNC(SYSDATE) - MOD(i, 365),  
        'PATIENT',  
        TRUNC(SYSDATE) - MOD(i, 100)  
    );  
END LOOP;  
COMMIT;  
END;  
/
```

## 8.2 Query To find time using an Index

The following code execute and the time difference between using index and no using index is shown below.

```
SET SERVEROUTPUT ON;  
  
DECLARE  
    start_time NUMBER;  
    end_time NUMBER;  
BEGIN  
    start_time := DBMS_UTILITY.GET_TIME;  
    FOR rec IN (SELECT * FROM users WHERE name = 'User_83') LOOP  
        NULL;  
    END LOOP;  
    end_time := DBMS_UTILITY.GET_TIME;
```

```
DBMS_OUTPUT.PUT_LINE('Time before creating index: ' || (end_time -  
start_time) || ' hundredths of seconds');  
EXECUTE IMMEDIATE 'CREATE INDEX idx_users_name ON Users(name)';  
start_time := DBMS_UTILITY.GET_TIME;  
FOR rec IN (SELECT * FROM users WHERE name = 'User_83') LOOP  
    NULL;  
END LOOP;  
end_time := DBMS_UTILITY.GET_TIME;  
DBMS_OUTPUT.PUT_LINE('Time after creating index: ' || (end_time -  
start_time) || ' hundredths of seconds');  
EXECUTE IMMEDIATE 'DROP INDEX idx_users_name';  
END;  
/
```

### 8.3 Results Comparison

The following table summarizes the execution times for the query before and after indexing:

Scenario	Execution Time (Example)
Without Index	0.044 seconds
With Index	0.006 seconds

Table 2: Execution Time Comparison for HMS.Medical Staff Table

## 8.4 Python Automation

The process can be automated using a Python script, which connects to the Oracle database, executes the queries, and measures the execution time:

```
def indexing():  
    conn = oracledb.connect(  
        user="KHOA", # Oracle username  
        password="Khoa0301#", # Oracle password  
        dsn="localhost/ORCLDB1", # DSN: hostname/servicename (or  
SID)  
    )  
    cursor = conn.cursor()  
  
    start_time = time.time()  
    cursor.execute("SELECT * FROM users WHERE name = 'User_83'")  
    data = cursor.fetchall()  
    end_time = time.time()  
    print(f"Execution Time Without Index: {end_time -  
start_time:.2f} seconds")  
    cursor.execute("CREATE INDEX idx_users_name ON Users(name)")  
  
    start_time = time.time()  
    cursor.execute("SELECT * FROM users WHERE name = 'User_83'")  
    data = cursor.fetchall()  
    end_time = time.time()  
    print(f"Execution Time With Index: {end_time - start_time:.2f}  
seconds")  
    cursor.execute("DROP INDEX idx_users_name")
```



```
cursor.close()  
conn.close()
```

## 8.5 Conclusion

The experiment demonstrates a significant performance improvement when using indexing. Without an index, the query scans the entire table, which can be slow for large datasets. By creating an index on the name column, the query performance improves drastically, making it more efficient for large-scale applications. This shows the importance of indexing for optimizing database performance in real-world scenarios.