PULSE

School of Science & Engineering

CSC 3326: DATABASE SYSTEMS

Pulse: Project Proposal

Manal Mehdaoui

Manal Houmimid

Taha Chadli

Supervised by:
Dr. Lamiae Bouanane

Table of Contents :

I. **Introduction:**

A. Project description**:** Organizational structure, mission, operations, problems and constraints.

B. Project objectives.

C. Project Planning.

II. Design:

A. Conceptual design:

1. Requirements' specification: main system processes.

B. Requirements Gathering.

A. Functional Requirements and Nonfunctional Requirements.

2. Business rules, entities, relationships.

3. Initial E-R model.

4. Table normalization.

5. Model verification: data manipulation operations and queries.

6. Final E-R model.

B. Logical design: tables, attributes' data types, constraints and views:

III. Implementation:

A. Database tables creation and population.

B. Application architecture: technology and tools used and their interaction/web user-friendly interface.

IV. Testing and fine-tuning:

V. User manual: Description of menus, forms, screens' snapshots...

VI. Conclusion:

VII. Future work (proposed future work to extend/improve your application):

1) Introduction:

In this project proposal, we aim to design a desktop and web-based application for hospital. Through this application, patients will be able to book appointments with their doctors. Moreover, our application aims to increase access to information such as health records, and upcoming and previous reservations, alongside the availability of a doctor, while also saving up a tremendous amount of time for both patients and doctors since both parties will be able to fill in data and access it at any time.
We chose to give our application the name: **Pulse**.
The team members are **Manal Mehdaoui**, **Manal Houmimid**, and **Taha Chadli**.

B. Project objectives.

Throughout this project, we will manage to apply the knowledge that we acquired form our Database System course. We have the opportunity to utilize what we covered in class in a real-world situation. Plus, we apply the topics related to the design and implementation of a database.

Moreover, we will have the chance to refine our project through following the important steps of a good design which include the normalization process, choosing the right primary keys for each table, and the right relationships.

All of these skills will be used in our project ‘Pulse’ where we create a real-life application with both a front end and a backend and a database which will hold all the information of the important members of the hospital application.

C. Project Planning.

Time management

- **Dates and tasks:**

Week1: (09/24/2022 - 09/30/2022) =**Information Gathering**

- First meeting with team members
- Creation of a group chat to share our project specifications
- Gathering data available.
- Dividing tasks

Week2: (3 October-9 October) = **Project Proposal**

- Writing the project proposal
- Specifying the entities, their attributes, and relationships
- Specifying the objectives and functionalities of the project
- Planning of the project

Week 3: (11/11/2022 - 11/30/2022) = **Database Design**

- Adding more requirements to have a specific goal
- Discuss the database design
- Creating sprints for the project
- Creating an initial ER model
- Dividing the project sprints among team members
- Updating the project's report

Week 4: (11/30/2022 - 12/06/2022) = **Database Creation**

- Start implementation of database tables
- Implementation of the frontend of the appointments

Week 5: (12/07/2022 - 12/12/2022) = **Software design/ project mid report**

- Software design
- Working and implementing the back end of the appointment.
- Implementing the authentication system that includes both login and sign-up.
- Update report

Week 6: (12/13/2022 - 12/14/2022) = **Code implementation/ Software implementation**

- Logical and relational design
- Complete writing code

Week 7: (12/15/20200) = **Testing**

- Review the code to make modifications if needed
- Review the front end to check if it is easy to use by the end-user

Week 8: (12/16/022) = **Review/ Testing/ Final report**

- Testing the website and fixing any bugs and issues faced
- Update and finalize the report
- Check the application before showing it to the client

- **Distribution of responsibilities:**
  - To be able to provide a good application and do a good project, responsibilities should be given equally between the team members. For the first part of this feasibility study Taha Chadli has taken care of the Introduction. Manal Mehdaoui has worked on the requirements gathering and the project management plan. On the other hand, Manal Houmimid has worked on the requirements specification.

- **Procedures and tasks distribution:**

Good communication between teammates and being responsible for reviewing the deliverables are important to obtain a good project result. Moreover, to deliver a good product clear communication with the client is very important to get clear stated requirements. So, splitting the tasks earlier between our team members will be beneficial and important so we can better manage the project and be able to integrate everything and review each other's work clearly.

Task	**Supervisor**
Database design	Taha Chadli
Database creation	Manal Mehdaoui, Manal Houmimid
Software design	Taha Chadli, Manal Mehdaoui
Software implementation	Manal Mehdaoui, Taha Chadli
Testing	Taha Chadli, Manal Mehdaoui

2) Requirements Gathering:

In this project we will be able to learn more concepts that are used in database systems and discover how to use them in a real-life application. Moreover, we will be able to apply different knowledge and information that we learned in class which focuses mostly on the design and other database concepts. As for the implementation part of the project we will be able to use our knowledge gained from the lab sessions.

- This project will be a web application that manages the functionality and database of a hospital set up. It will be a secure and easy way of storing information of the outdoor and indoor patients, doctors, rooms, and bill payments.

- **The requirements of these project are:**
  - User friendly web application
  - Access for doctors, patients, and staff
  - Creating appointments
- **The objectives of this project are:**
  - Recording information about doctors, patients, staff, and rooms
  - Generating bills for transactions
  - Easy access for doctors to see their patient's information and statues
  - Facilitating taking appointments for patients.
  - User friendly interface for the users
  - Maintaining records of indoor and outdoor patients
- **The functionalities to achieve these objectives**
  - A web application with front end and back end because the two sides to communicate and operate effectively so the website's functionality improves.
  - Using more than one programming language to help build the API (react, css, html, javasript)
  - For the databases, the SQL language will be the programming language used to manage and organize the data.
  - The website should have an authentication for doctors, patients, and staff
  - Creation of an account by providing specific information that include name, last name, CIN, email, and phone number
  - Easy login with generated ID or email and password after the creation of the profile.

- Booking appointments by freedom of choice of time and doctor
- Ability to cancel appointments both by the patient and doctor
- Calculations of charges and other utilities
- Ability to update, add, and delete records of all the users
- Show the rooms availability this is only shown for the staff

3) Requirements Specification:

A. Functional Requirements:

- **login:**

User Requirements: The user should be able to log in into the software.

System Requirements: Accepts identification using an ID or email and password while handling any exceptions that might occur.

Input/Output: The input must be either an ID or an email and a password. The output should be an answer by the system providing feedback on whether the user is successfully in or not.

Patient's View

- **Setting Appointment:**

User Requirements: The user should be able to set an appointment specifying the time of their preference describing their condition

System Requirements: The system should provide to the patient all the possible free times and the doctors available.

Input/Output: The input should be the preferred time slot plus the day of the patient and a brief description of their condition, the output should be a confirmation message from the system precising the name of the doctor handling the patient's needs.

- **Check Record**

User Requirements: The user could check their personal record which should include their past appointments, prescriptions, and doctors/nurses handling their case.

System Requirement: The System must provide all the possible data about the patient.

Input/Output: The input should be the patient's password as their record is classified as sensitive information. The output must handle the case of wrong passwords, if the password is correct the output should be all the necessary data about the patient.

Admin's View

- **Manage Staff**

  Customer Requirement: The admin should be able to first have access to the list of all employees, and second add/delete any employee off that list.

  System Requirement: It should accept whatever edits the admin makes.

  Input/Output: Input should be whatever edits the admin wants to make while the output must be a confirmation of the system

- **Manage Patients**

  Customer Requirement: The admin should be able to add patients and update their records.

  System Requirement: It should accept whatever edits the admin makes.

  Input/Output: Input should be whatever edits the admin wants to make while the output must be a confirmation of the system

Doctor's view:

- **Manage appointments**

Customer Requirement: the doctor has the ability to manage appointments, by canceling, confirming, or editing

System Requirement: It should accept whatever edits the doctor makes.

Input/Output: Input should be whatever edits the doctor wants to make while the output must be a confirmation of the system

- **View patient information:**

  Customer Requirement: the doctor can view all patients' information, and edit their condition and information related to their health

  System Requirement: It should accept whatever edits the doctor makes.

  Input/Output: Input should be whatever edits the doctor wants to make while the output must be a confirmation of the system

- **Manage surgery information:**

  Customer Requirement: the doctor can view all the surgeries done, and edit any information related to it

  System Requirement: It should accept whatever edits the doctor makes.

  Input/Output: Input should be whatever edits the doctor wants to make while the output must be a confirmation of the system

Pharmacist view

- **Manage medicine**

  Customer Requirement: the pharmacist can manage all medication, add medicine, remove medicine, and manage its quantity.

System Requirement: It should accept whatever edits the pharmacist makes.

Input/Output: Input should be whatever edits the pharmacist wants to make while the output must be a confirmation of the system

Receptionist view

- **Manage visitors**

Customer Requirement: the receptionist can manage all visitors, and their check-in and check-out time

System Requirement: It should accept whatever edits the receptionist makes.

Input/Output: Input should be whatever edits the receptionist wants to make while the output must be a confirmation of the system

B. Non-functional Requirements:

- **Security:** The data should be well protected from any external manipulations and stored correctly.
- **Efficiency:** The process of retrieving the data from the database should be performed as fast as possible
- **Usability:** The software must be easy to use for any possible users without causing them any confusion.

2. Business rules, entities, relationships.

- The hospital is divided into five **departments**: surgical, outpatient department, inpatient service, pharmacy department, and rehabilitation department. Each department is

identified by its name, the number of rooms in it, and the types and numbers of staff members, and each department should have a manager.

- The hospital is defined by its ID and has an address, a name, and a website name.

- Employees of the hospital can be divided into doctors, nurses, pharmacists, and receptionists.

- Each employee is defined by their unique employee ID, they also have a first and last

name and a specified number of working hours with the start and the end of their shift.

- Each patient is defined by their patient ID and has a first name and last name, additionally, they have an address, a national identity card, phone number, email, and date of birth.

- In addition to its patient ID, an inpatient also has the number of nights they will stay and may have to undergo one or more surgeries. Surgery is defined by its surgery code, type, date, and duration.

- A doctor may offer many surgeries. But only one doctor is in the surgery

- An outpatient can specify when they want to do their annual checkup.

- A patient can request many appointments with their doctor depending on the available time slots. A doctor may have many appointments.

- Each department should have at least one doctor and nurse.

- A medicine is identified by its code and has a description, in date, quantity on hand and minimum quantity, price, and discount rate. A pharmacist has many types of medicine available for sale. A pharmacist can have a minimum of 10 medicines per type in stock.

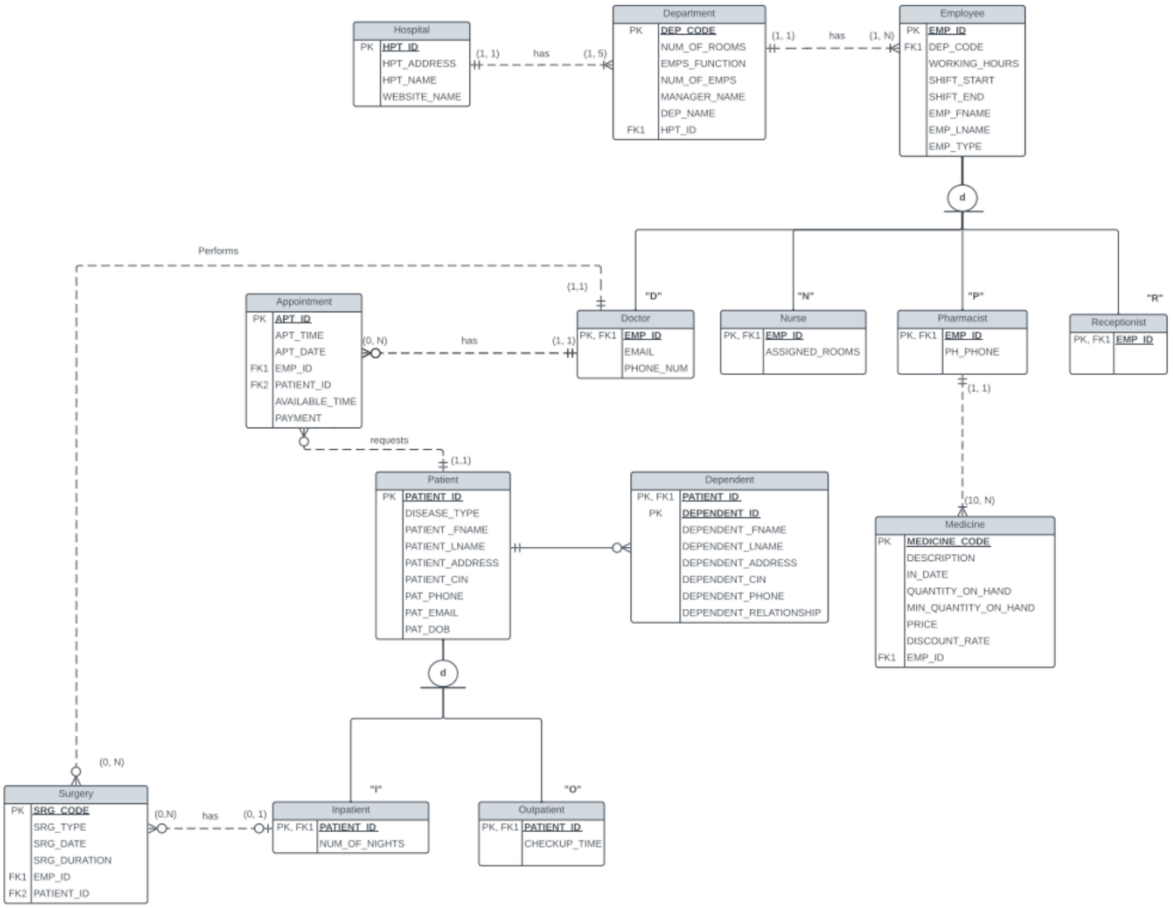
- A patient has a dependent which is a visitor. A visitor cannot exist without a patient.

- A patient can have many to no visitors. A visitor must have at least a patient.

- The visitor has a visitor id, a first name, last name, address, CIN, and phone number,

and his relationship with the patient.

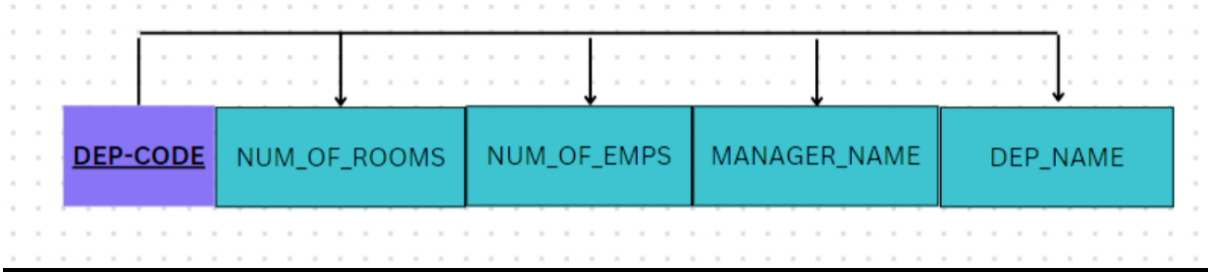
3. Initial E-R model.



4. Table normalization.

Dependency Diagram:

For the department table:



Candidate key(s): DEP_CODE

Non-prime dependency check: All the non-prime attributes which are NUM_OF_ROOMS, NUM_OF_EMPS, MANAGER_NAME, and DEP_NAME are fully dependent on each of the prime attributes.

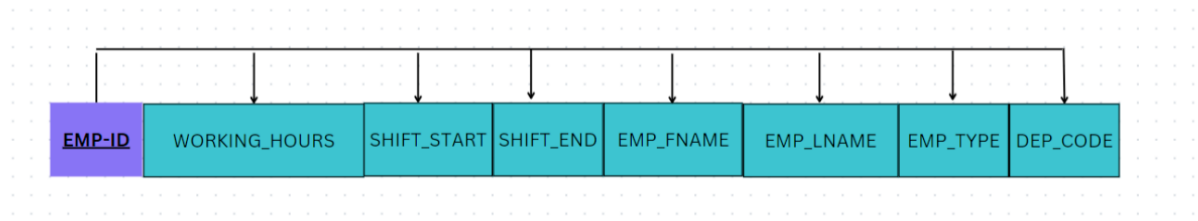
1NF (**DEP-CODE**, NUM_OF_ROOMS, NUM_OF_EMPS, MANAGER_NAME, DEP_NAME)

- All key attributes are defined.
- There are no repeating groups in the table.
- All attributes are dependent on the primary key.

Partial dependency check: The department table has no partial dependency because it only has a single candidate key. Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the tables because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the department table.

EMPLOYEE TABLE



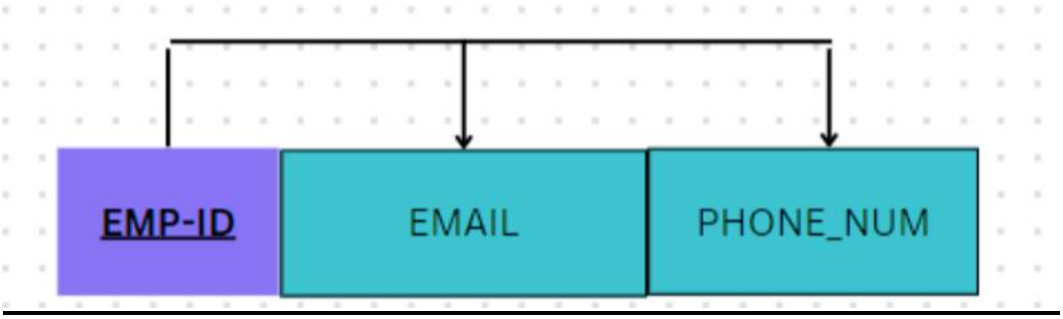
Candidate key(s): EMP_ID

Non-prime dependency check: All the non-prime attributes which are: WORKING_HOURS, SHIFT_START, SHIDT_END, EMP_FNAME, EMP_LNAME, EMP_TYPE, and DEP_CODE are fully dependent on each of the prime attributes.

Partial dependency check: The employee table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the employee table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the employee table.

Doctor Table:



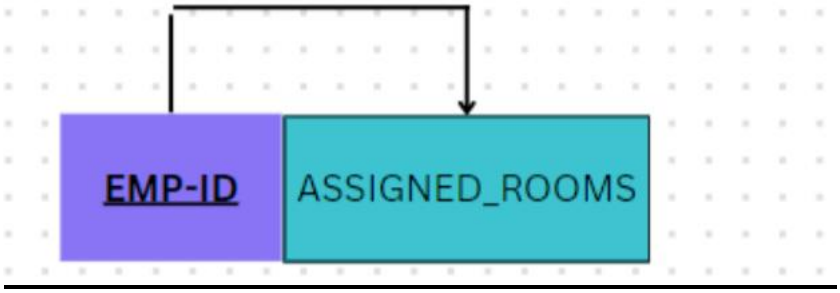
Candidate key(s): EMP_ID

Non-prime dependency check: All the non-prime attributes which are EMAIL, and PHONE_NUMBER are fully dependent on each of the prime attributes.

Partial dependency check: The doctor table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the doctor table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the doctor table.

Nurse Table



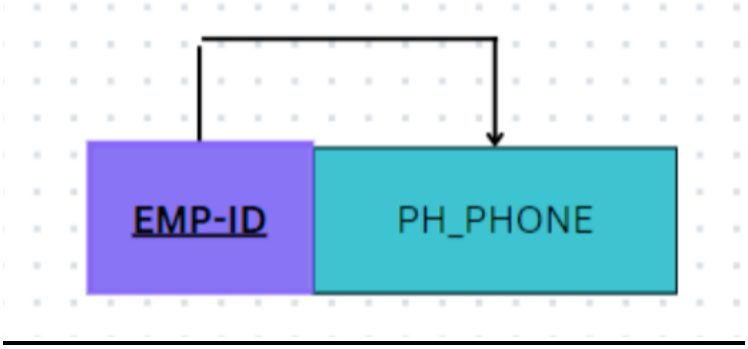
Candidate key(s): EMP_ID

Non-prime dependency check: All the non-prime attributes which are here only ASSIGNED_ROOMS are fully dependent on each of the prime attributes.

Partial dependency check: The NURSE table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the doctor table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the nurse table.

PHARMACIST TABLE:



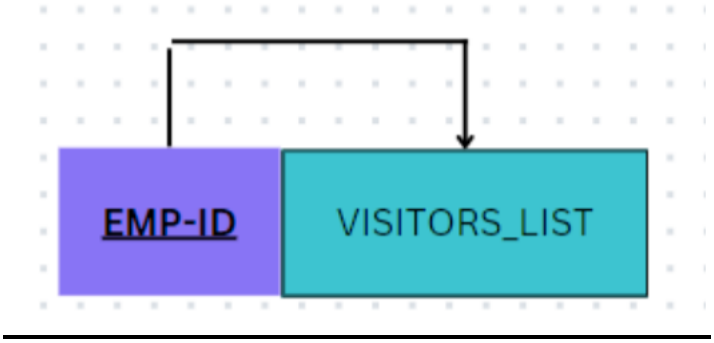
Candidate key(s): EMP_ID

Non-prime dependency check: All the non-prime attributes which are here only PH_PHONE are fully dependent on each of the prime attributes.

Partial dependency check: The pharmacist table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the pharmacist table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the pharmacist table.

RECEPTIONIST TABLE



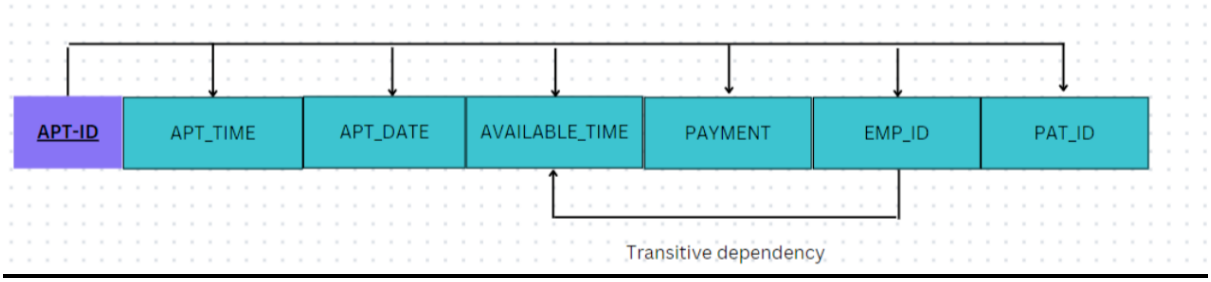
Candidate key(s): EMP_ID

Non-prime dependency check: All the non-prime attributes which are here only VISITORS_LIST are fully dependent on each of the prime attributes.

Partial dependency check: The receptionist table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the receptionist table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the receptionist table.

Appointment table:



Candidate key(s):APT_ID

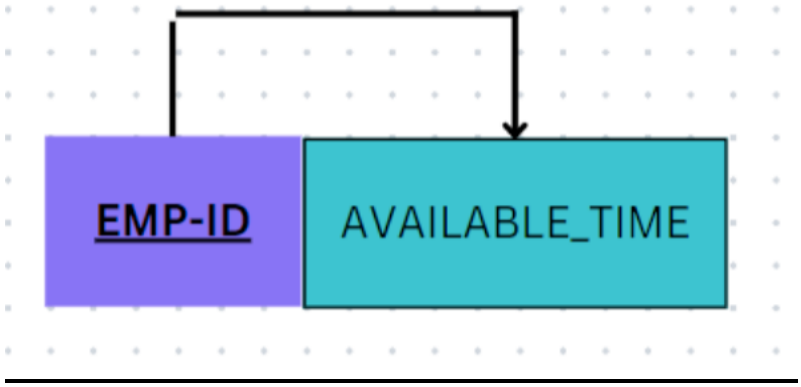
Non-prime dependency check: All the non-prime attributes which are here APT_TIME, APT_DATE, AVAILABLE_TIME, PAYMENT, EMP_ID, and PAT_ID are fully dependent on each of the prime attributes.

Partial dependency check: The receptionist table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is a transitive dependency in the Appointment table:

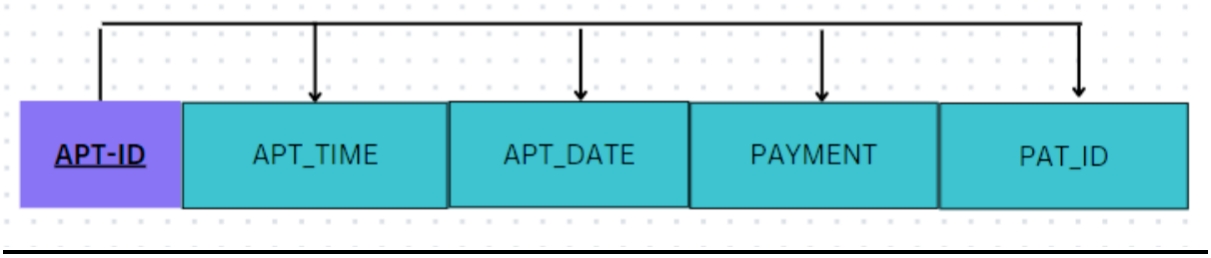
So, make a new table to eliminate transitive dependency

This new table is named **Availability**:

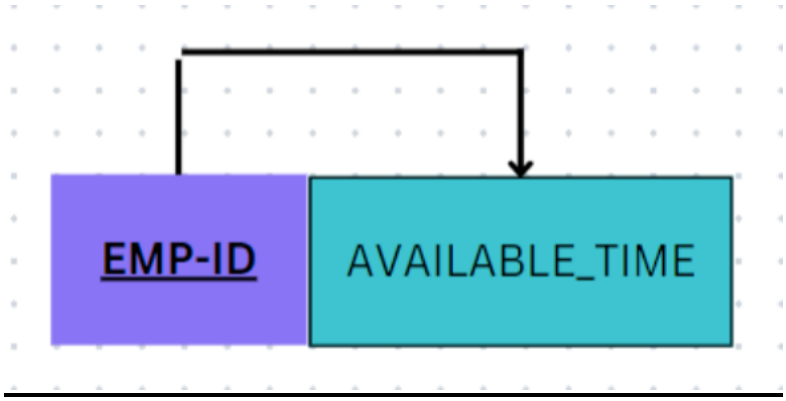


➔ The two tables that we have now id the appointment table and availability:

Appointment table:

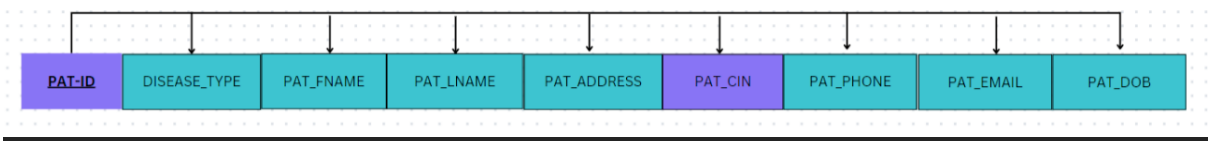


Availability table:



Both of the tables are 3NF :

Patient table:



Candidate key(s): PATIENT_ID, and PATIENT_CIN

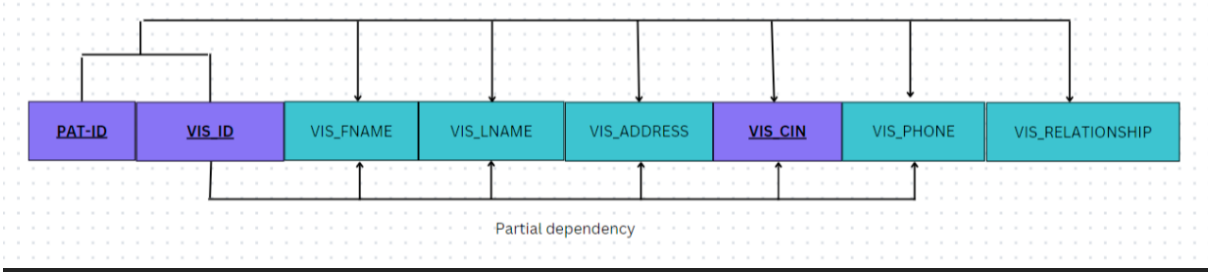
Non-prime dependency check: All the non-prime attributes which are DISEASE_TYPE, PAT_FNAME, PAT_LNAME, PAT_ADDRESS, PAT_PHONE, and PAT_EMAIL are fully dependent on each of the prime attributes.

Partial dependency check: The patient table has no partial dependency because it only has single candidate keys. Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the patient table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the patient table.

PS: *Even if the CIN is a candidate key, it cannot be a primary because it is descriptive.*

Visitor table:

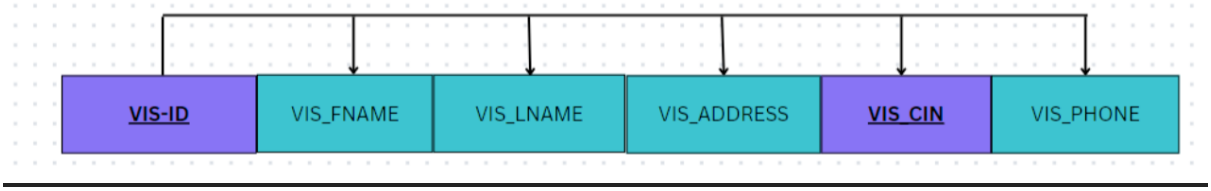


Candidate key(s): PAT_ID, VIS_ID/ VIS_CIN

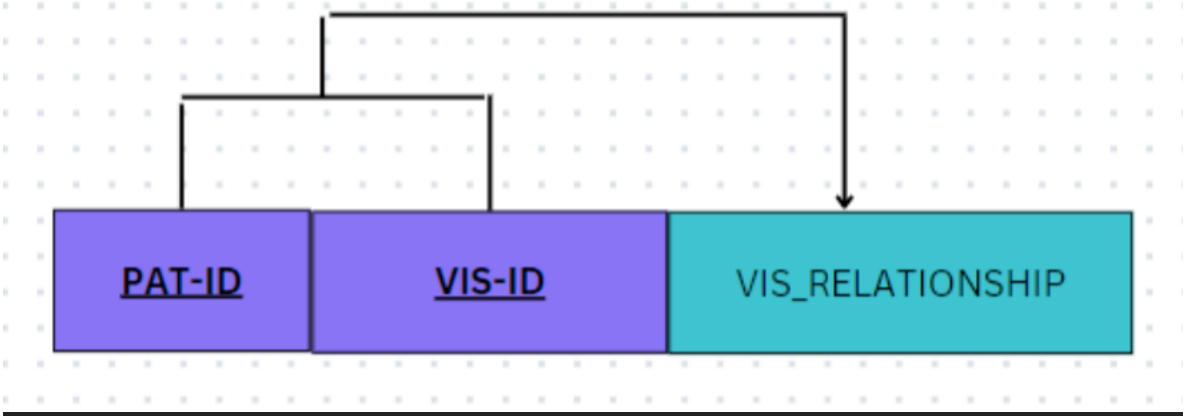
Non-prime dependency check: All the non-prime attributes which are VIS_FNAME, VIS_LNAME, VIS_ADDRESS, VIS_CIN, VIS_PHONE, VIS_RELATIONSHIP are fully dependent on each of the prime attributes.

Partial dependency check: The VISITOR table has a partial dependency

So, the **new Visitor table**



And the second table is named **Relationship:**

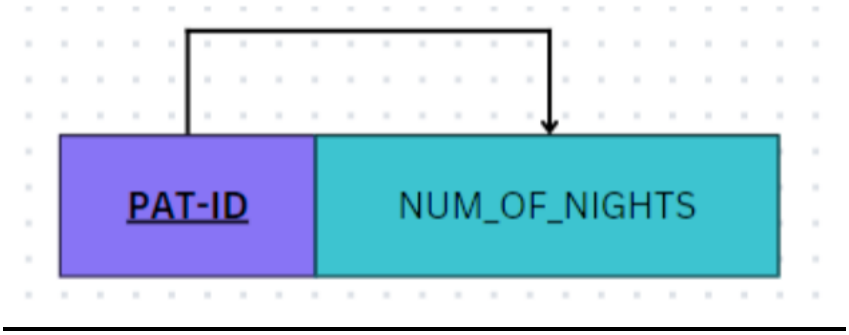


So the two table relationship and visitor are in the 2NF

Transitive dependency check: There is no transitive dependency in the patient and relationship table becauseno non-prime attribute can be used to determine another one. Therefore, the 3 NF is reachedfor the two tables.

PS: *Even if the CIN is a candidate key, it cannot be a primary because it is descriptive.*

Inpatient table:



Candidate key(s): PATIENT_ID

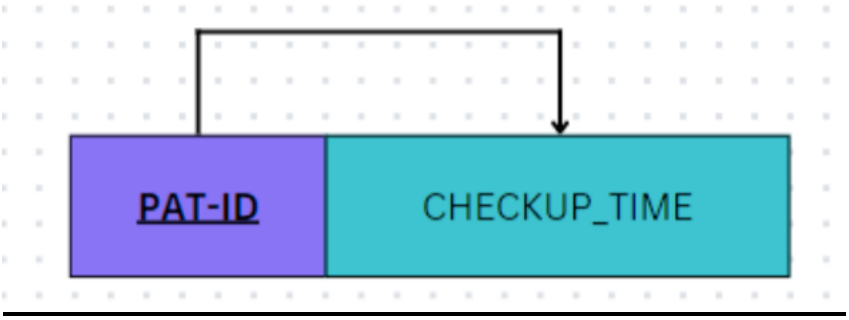
Non-prime dependency check: All the non-prime attributes which are only

NUM_OF_NIGHTS are fully dependent on each of the prime attributes.

Partial dependency check: The INPATIENT table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the inpatient table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the inpatient table.

OUTPATIENT TABLE:



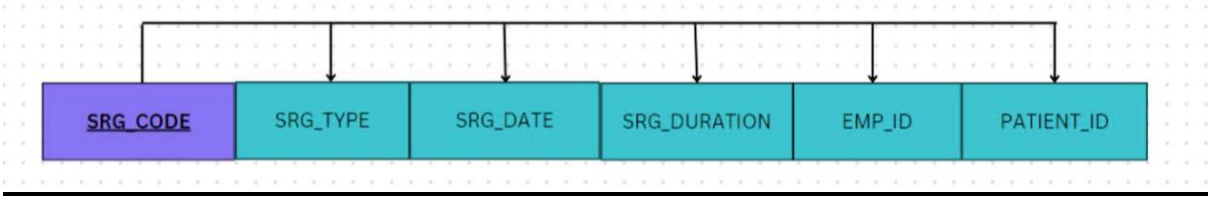
Candidate key(s): PATIENT_ID

Non-prime dependency check: All the non-prime attributes which are only CHECKUP_TIME are fully dependent on each of the prime attributes.

Partial dependency check: The PATIENT table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the patient table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the outpatient table.

Surgery table:



Candidate key(s): SRG_CODE

Non-prime dependency check: All the non-prime attributes which are SRG_TYPE, SRG_DATE, SRG DURATION, EMP_ID, and PATIENT_ID are fully dependent on each of the prime attributes.

Partial dependency check: The SURGERY table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the SURGERY table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the SURGERY table.

Medicine table:

Candidate key(s): MEDICINDE_CODE

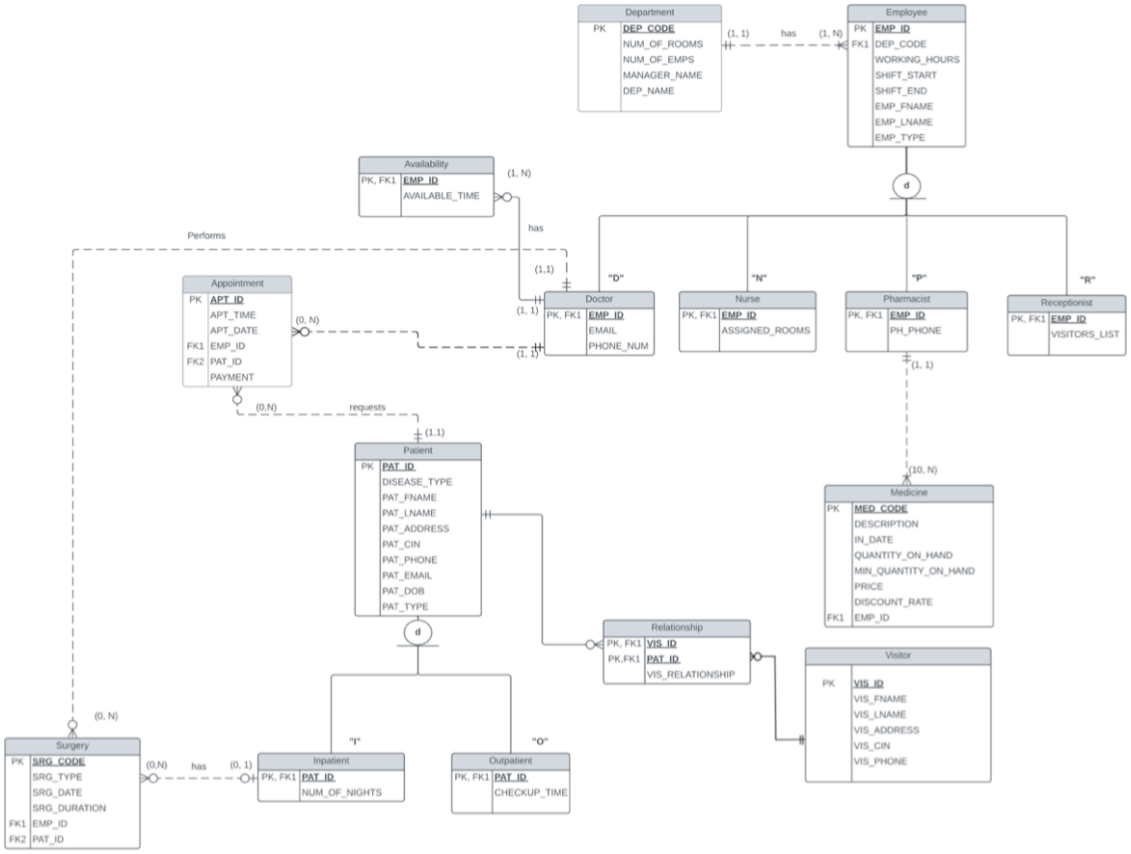
Non-prime dependency check: All the non-prime attributes which are DESCRPTION, IN_DATE, QUANTITY_ON_HAND, MIN_QUANTITY_ON_HAND, PRICE, DISCOUNT_RATE, and EMP_ID are fully dependent on each of the prime attributes.

Partial dependency check: The MEDICINE table has no partial dependency because it only has single candidate keys (1 primary key). Therefore, it is on 2 NF.

Transitive dependency check: There is no transitive dependency in the MEDICINE table because no non-prime attribute can be used to determine another one. Therefore, the 3 NF is reached for the medicine table.

ER diagram:

After the normalization process, the new ERD is as follows:



New Business rules:

1) The hospital is divided into five departments: surgical, outpatient department, inpatient service, pharmacy department, and rehabilitation department. Each department is identified by its name, the number of rooms in it, and the types and numbers of staff members, and each department should have a manager.

2) Employees of the hospital can be divided into doctors, nurses, pharmacists, and receptionists.

3) Each employee is defined by their unique employee ID; they also have a first and last name and a specified number of working hours with the start and the end of their shift.

4) Each patient is defined by their patient ID and has a first name and last name, additionally, they have an address, a national identity card, phone number, email, and date of birth.

5) In addition to its patient ID, an inpatient also has the number of nights they will stay and may undergo many surgeries. Surgery is defined by its surgery code, type, date, and duration.

6) A doctor may offer many surgeries. But only one doctor is in the surgery.

7) An outpatient can specify when they want to do their annual checkup.

8) A patient can request many appointments with their doctor depending on the available time slots. A doctor may have many appointments.

9) Each department should have at least one doctor and nurse.

10) A medicine is identified by its code and has a description, in date, quantity on hand and minimum quantity, price, and discount rate. A pharmacist has many types of medicine available for sale. A pharmacist can have a minimum of 10 medicines per type in stock.

11) A patient has a dependent which is a visitor. A visitor cannot exist without a patient.

12) A patient can have many to no visitors. A visitor must have at least a patient.

13) The visitor has a visitor id, a first name, last name, address, CIN, and phone number, and his relationship with the patient.

Data-Modeling Checklist:

- All the tables are in the 3NF, we will go through the checklist to make sure that the design part is relevant.

The business rules are concise and clear : They are written precisely, clearly, and simply.

Entity names:

- Nouns that are familiar to the business, short and meaning full.

- **Granularity:** no further details should be given.
- All entity names are unique within the model.
- Each entity represents a single subject

Abbreviations:

Emp ➔ employee

Pat ➔ patient

Dep ➔ department

Apt ➔ appointment

Srg ➔ surgery

Vis ➔ visitor

Med ➔ medicine

Ph ➔ pharmacist

Attributes names:

- All attributes' names are unique within the entity.
- They use the entity abbreviation as a prefix.
- They are descriptive of the characteristic.
- Not a reserved word.
- Each non-key attribute is fully dependent on the primary key of the entity it belongs to.

- All attributes are named after the naming convention

Relationship names:

- All relationship names are in active form
- All relationships clearly identify relationship participants

- All relationships clearly define participation and connectivity.

Entities:

- Each entity represents a single subject
- Each entity represents a set of distinguishable entity instances.
- All entities are in 3NF.
- Granularity: no further details should be given.

- PK should be clearly defined and support the selected data granularity.

Attributes:

- Simple and single-valued (atomic data).
- Documents default values, constraints, synonyms, and aliases.
- Not redundant unless this is required for transaction accuracy, performance, or maintaining a history.

- Nonkey attributes are fully dependent on the PK attribute.

Relationships:

- Clearly identifies relationship participants.
- Clearly define participation, connectivity, and document cardinality.

5. Model verification: data manipulation operations and queries.

Insert doctor:

```
25
26  insert into doctor (EMP_ID,emp_wh,
27      emp_password,
28      emp_shift_start,
29      emp_shift_end ,
30      emp_fname,
31      emp_lname,
32      emp_type,doc_email,doc_phone)
33      values(10,'24:00:00','james123','6:00:00','20:00:00','James','Kim','Doctor','james100@gmail.com','9761');
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 106 msec.
```

-→ Insert Receptionist:

```
25
26  insert into receptionist (EMP_ID,emp_wh,
27      emp_password,
28      emp_shift_start,
29      emp_shift_end ,
30      emp_fname,
31      emp_lname,
32      emp_type)
33      values(19,'24:00:00','safae22','6:00:00','20:00:00','Safae','Salami','Receptionist');
34
35
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 57 msec.
```

→ Insert Pharmacist

```
16
17  insert into pharmacist(emp_id,emp_wh, emp_shift_start, emp_shift_end, emp_fname,emp_lname,emp_type, ph_phone)
18  values (2,'24:00:00', '8:00:00','20:00:00','Mehdi', 'Zahiri', 'pharmacist',0661722181);
19
20
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 62 msec.
```

→ Insert nurse:

```
27
28  insert into nurse(EMP_ID,emp_wh,
29      emp_password,
30      emp_shift_start,
31      emp_shift_end ,
32      emp_fname,
33      emp_lname,
34      emp_type,assigned_rooms)
35      values(2,'24:00:00','karim122','6:00:00','20:00:00','Karima','Salami','nurse','7');
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 56 msec.
```

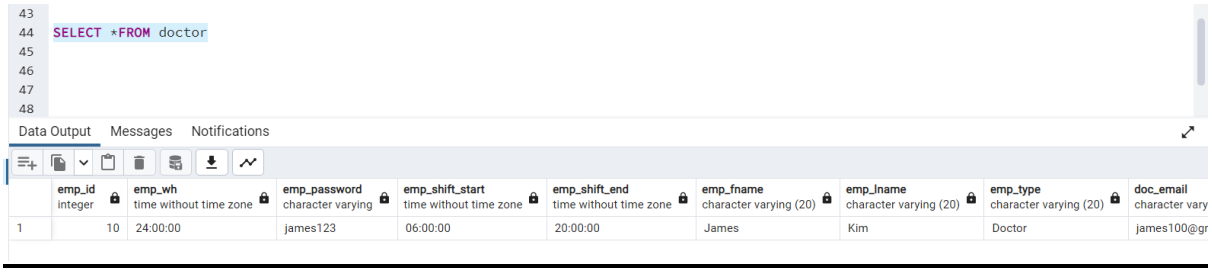
→ select all employees:

```
43
44  SELECT *FROM EMPLOYEE
45
```

Data Output | Messages | Notifications

	emp_id [PK] integer	emp_wh time without time zone	emp_password character varying	emp_shift_start time without time zone	emp_shift_end time without time zone	emp_fname character varying (20)	emp_lname character varying (20)	emp_type character varying (20)
1	10	24:00:00	james123	06:00:00	20:00:00	James	Kim	Doctor
2	19	24:00:00	safae22	06:00:00	20:00:00	Safae	Salami	Receptionist
3	2	24:00:00	[null]	08:00:00	20:00:00	Mehdi	Zahiri	pharmacist
4	2	24:00:00	karim122	06:00:00	20:00:00	Karima	Salami	nurse

→ select doctor:

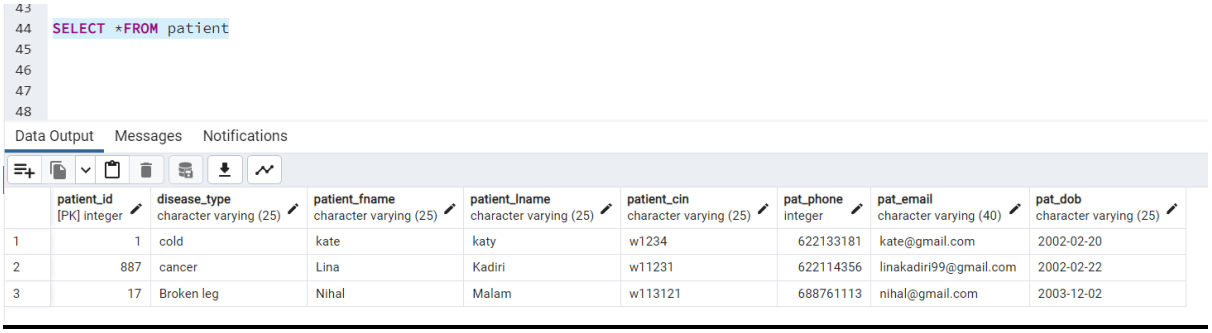


```
SELECT *FROM doctor
```

Data Output | Messages | Notifications

	emp_id integer	emp_wh time without time zone	emp_password character varying	emp_shift_start time without time zone	emp_shift_end time without time zone	emp_fname character varying (20)	emp_lname character varying (20)	emp_type character varying (20)	doc_email character vary
1	10	24:00:00	james123	06:00:00	20:00:00	James	Kim	Doctor	james100@gr

→ select patient:

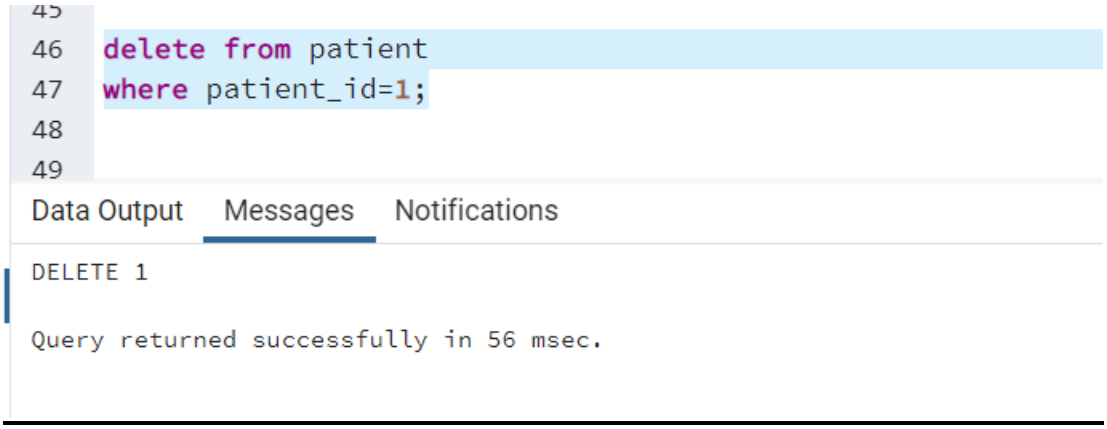


```
SELECT *FROM patient
```

Data Output | Messages | Notifications

	patient_id [PK] integer	disease_type character varying (25)	patient_fname character varying (25)	patient_lname character varying (25)	patient_cin character varying (25)	pat_phone integer	pat_email character varying (40)	pat_dob character varying (25)
1	1	cold	kate	katy	w1234	622133181	kate@gmail.com	2002-02-20
2	887	cancer	Lina	Kadiri	w11231	622114356	linakadiri99@gmail.com	2002-02-22
3	17	Broken leg	Nihal	Malam	w113121	688761113	nihal@gmail.com	2003-12-02

→delete patient



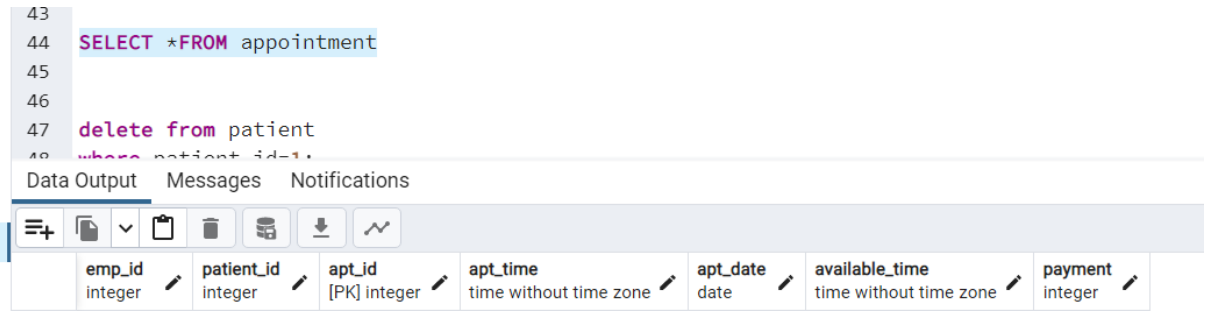
```
delete from patient
where patient_id=1;
```

Data Output | Messages | Notifications

```
DELETE 1

Query returned successfully in 56 msec.
```

→ Select appointment



```
SELECT *FROM appointment

delete from patient
```

Data Output | Messages | Notifications

emp_id integer	patient_id integer	apt_id [PK] integer	apt_time time without time zone	apt_date date	available_time time without time zone	payment integer

→ select department

```
44  SELECT *FROM department
45
46
47  delete from patient
```

Data Output | Messages | Notifications

	dep_code [PK] character varying (25)	num_of_rooms integer	num_of_emps integer	manager_name character varying (25)	dep_name character varying (25)
1	1	10	5	Khalid	Admissions

→ select medicine

```
44  SELECT *FROM medicine
```

Data Output | Messages | Notifications

	medicine_code character varying (25)	description character varying (30)	in_date date	quantity_on_hand integer	min_quantity_on_hand integer	price numeric	discount_rate integer
1	1	Doliprane 1g	2022-11-11	40	5	14.6	0
2	2	RHUMIX, Sachet 400MG	2022-12-12	58	10	22	0

→ update medicine

```
43
44  SELECT *FROM medicine
45  Update medicine
46  set description= '';
47
```

Data Output | Messages | Notifications

	medicine_code character varying (25)	description character varying (30)	in_date date	quantity_on_hand integer

→ update patient name:

```
47
48  Update patient
49  set patient_fname= 'Salim'
50  where patient_id=17;
51
```

→ select from dependent(visitor)

```
60  select *from dependent
61
```

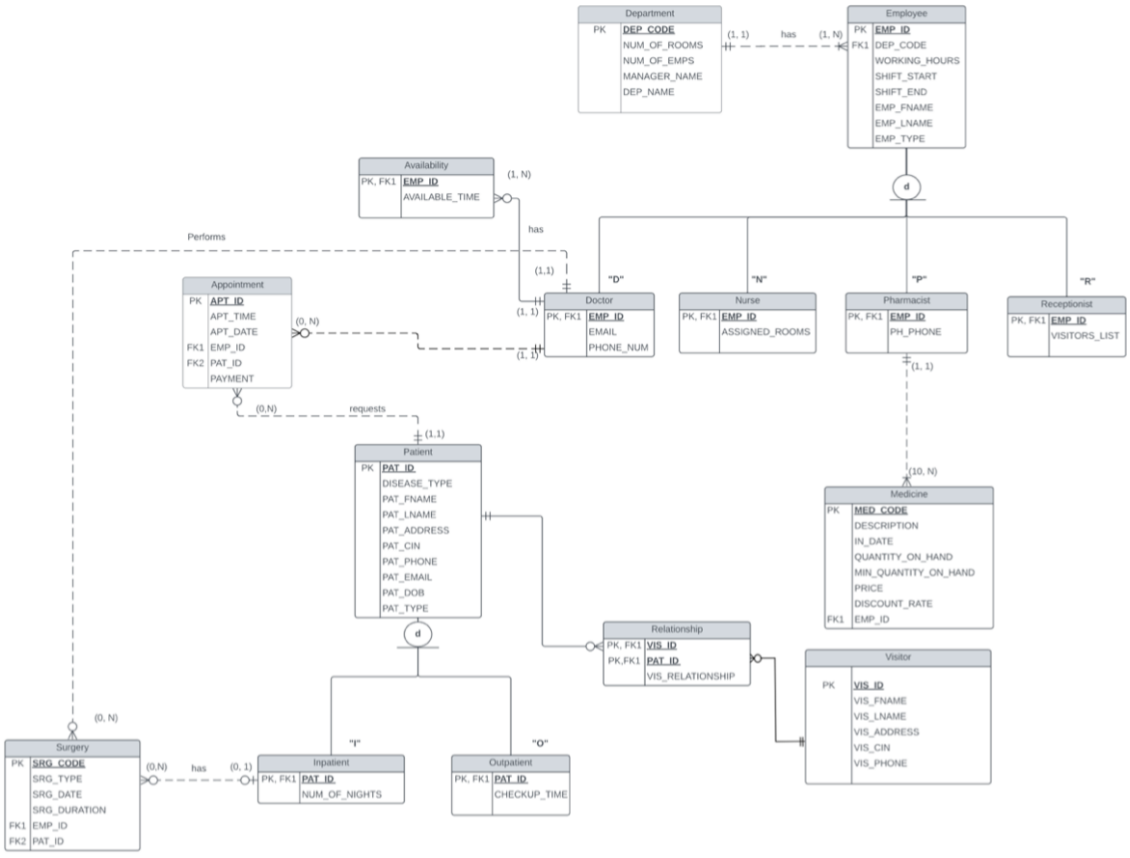
Data Output | Messages | Notifications

	dependent_id [PK] integer	patient_id integer	dependent_fname character varying (20)	dependent_lname character varying (20)	dependent_address character varying (30)	dependent_cin character varying (20)	dependent_phone character varying (30)	dependent_relationship character varying (30)
1	1	1	Salim	Radi	France	w1431	08817	Brother
2	12	1112	Karim	Kadiri	settat	w1123	0677195466	Brother
3	52	4	Kawtar	Karimi	ifrane	w1131	0677184114	Sister
4	90	1	Zahid	Kadiri	zahid100@gmail.com	z167s2	0625463876	Brother

→ delete from dependent:

```
delete from dependent
where patient_id=1;
```

6. Final E-R model.



B. Logical design: tables, attributes' data types, constraints and views:

1) Tables:

```
CREATE table employee (
    EMP_ID int,
    emp_wh time,
    emp_password varchar,
    emp_shift_start time,
    emp_shift_end time,
    emp_fname varchar (20),
    emp_lname varchar (20),
    emp_type varchar (20),
    CONSTRAINT Empkey PRIMARY KEY(EMP_ID)

);
```

```
create table doctor(
doc_email varchar (45),
doc_phone varchar (45)

)inherits (employee);

create table receptionist(


)inherits (employee);

create table pharmacist(
ph_phone varchar(45)

)inherits (employee);

create table nurse(
assigned_rooms varchar(25)

)inherits (employee);
```

```sql
create table hospital(
HPT_address varchar (40),
HPT_name varchar(25),
website_name varchar(25)

);

create table department(
dep_code varchar(25),
num_of_rooms int,
num_of_emps int,
manager_name varchar(25),
dep_name varchar(25),
CONSTRAINT depkey PRIMARY KEY(dep_code)
);
```

```sql
create table patient(
PATIENT_ID int,
DISEASE_TYPE varchar(25),
PATIENT_FNAME varchar(25),
PATIENT_LNAME varchar (25),
PATIENT_CIN varchar(25),
PAT_PHONE int,
PAT_EMAIL varchar(40),
PAT_DOB varchar (25),
CONSTRAINT patientkey PRIMARY KEY(PATIENT_ID)
);

create table inpatient(
num_of_nights int

)inherits (patient);
```

```
create table outpatient(
checkup_time varchar(25)

)inherits (patient);

create table surgery(
EMP_ID INT,
PATIENT_ID INT,
SRG_CODE varchar (25),
SRG_TYPE varchar(25),
SRG_DATE date,
SRG_DURATION int,
CONSTRAINT surgkey PRIMARY KEY(SRG_CODE),
CONSTRAINT EMP FOREIGN KEY (EMP_ID) REFERENCES employee(EMP_ID),
CONSTRAINT PAT FOREIGN KEY (PATIENT_ID) REFERENCES patient (PATIENT_ID)
);
```

```
create table appointment (
EMP_ID INT,
PATIENT_ID INT,
APT_ID int not null,
APT_TIME time,
APT_DATE date,
AVAILABLE_TIME time,
payment int,
CONSTRAINT appointmentkey PRIMARY KEY(APT_ID),
CONSTRAINT  EMP FOREIGN KEY (EMP_ID) REFERENCES employee(EMP_ID),
CONSTRAINT PAT FOREIGN KEY (PATIENT_ID) REFERENCES patient(PATIENT_ID)

);
```

```
create table dependent(
dependent_id int,
patient_id int,
dependent_fname varchar (20),
dependent_lname varchar (20),
dependent_address varchar (30),
dependent_cin varchar (20),
dependent_phone varchar(30),
dependent_relationship varchar(30),
CONSTRAINT dep PRIMARY KEY(dependent_id),
CONSTRAINT EMP FOREIGN KEY (patient_id) REFERENCES patient(patient_id)

);
```

```
create table medicine(

MEDICINE_CODE varchar(25),
DESCRIPTION varchar(30),
IN_DATE date,
QUANTITY_ON_HAND int,
MIN_QUANTITY_ON_HAND int,
PRICE decimal,
DISCOUNT_RATE int
);
```

Triggers:

```
CREATE FUNCTION insertpatient()
  RETURNS trigger AS
$$
BEGIN
        INSERT INTO employee(emp_id,emp_fname,emp_lname)
        VALUES(NEW.emp_id,NEW.emp_fname,emp_lname);

    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER pattrigger
  AFTER INSERT
  ON employee
  FOR EACH ROW
  EXECUTE PROCEDURE insertpatient();
```

```
CREATE FUNCTION update_med() RETURNS TRIGGER
AS $$

BEGIN
UPDATE medicine
SET reorder = 1
WHERE medicine_code = NEW.medicine_code;
RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER trmedUpd AFTER UPDATE ON
medicine FOR EACH ROW
WHEN (OLD.QUANTITY_ON_HAND IS DISTINCT FROM NEW.QUANTITY_ON_HAND
AND NEW.QUANTITY_ON_HAND < NEW.MIN_QUANTITY_ON_HAND) EXECUTE
FUNCTION update_med();
```

II. **Implementation:**

Data was randomly generated

A. Database tables creation and population.

Creation

```
CREATE table employee (
    EMP_ID int,
    emp_wh time,
    emp_password varchar,
    emp_shift_start time,
    emp_shift_end time,
    emp_fname varchar (20),
    emp_lname varchar (20),
    emp_type varchar (20),
    CONSTRAINT Empkey PRIMARY KEY(EMP_ID)

);
```

```
create table doctor(
doc_email varchar (45),
doc_phone varchar (45)

)inherits (employee);

create table receptionist(


)inherits (employee);

create table pharmacist(
ph_phone varchar(45)

)inherits (employee);

create table nurse(
assigned_rooms varchar(25)

)inherits (employee);
```

```sql
create table hospital(
HPT_address varchar (40),
HPT_name varchar(25),
website_name varchar(25)

);

create table department(
dep_code varchar(25),
num_of_rooms int,
num_of_emps int,
manager_name varchar(25),
dep_name varchar(25),
CONSTRAINT depkey PRIMARY KEY(dep_code)
);
```

```sql
create table patient(
PATIENT_ID int,
DISEASE_TYPE varchar(25),
PATIENT_FNAME varchar(25),
PATIENT_LNAME varchar (25),
PATIENT_CIN varchar(25),
PAT_PHONE int,
PAT_EMAIL varchar(40),
PAT_DOB varchar (25),
CONSTRAINT patientkey PRIMARY KEY(PATIENT_ID)
);

create table inpatient(
num_of_nights int

)inherits (patient);
```

```
create table outpatient(
checkup_time varchar(25)

)inherits (patient);

create table surgery(
EMP_ID INT,
PATIENT_ID INT,
SRG_CODE varchar (25),
SRG_TYPE varchar(25),
SRG_DATE date,
SRG_DURATION int,
CONSTRAINT surgkey PRIMARY KEY(SRG_CODE),
CONSTRAINT EMP FOREIGN KEY (EMP_ID) REFERENCES employee(EMP_ID),
CONSTRAINT PAT FOREIGN KEY (PATIENT_ID) REFERENCES patient (PATIENT_ID)
);
```

```
create table appointment (
EMP_ID INT,
PATIENT_ID INT,
APT_ID int not null,
APT_TIME time,
APT_DATE date,
AVAILABLE_TIME time,
payment int,
CONSTRAINT appointmentkey PRIMARY KEY(APT_ID),
CONSTRAINT  EMP FOREIGN KEY (EMP_ID) REFERENCES employee(EMP_ID),
CONSTRAINT PAT FOREIGN KEY (PATIENT_ID) REFERENCES patient(PATIENT_ID)

);
```

```
create table dependent(
dependent_id int,
patient_id int,
dependent_fname varchar (20),
dependent_lname varchar (20),
dependent_address varchar (30),
dependent_cin varchar (20),
dependent_phone varchar(30),
dependent_relationship varchar(30),
CONSTRAINT dep PRIMARY KEY(dependent_id),
CONSTRAINT EMP FOREIGN KEY (patient_id) REFERENCES patient(patient_id)

);
```

```
create table medicine(

MEDICINE_CODE varchar(25),
DESCRIPTION varchar(30),
IN_DATE date,
QUANTITY_ON_HAND int,
MIN_QUANTITY_ON_HAND int,
PRICE decimal,
DISCOUNT_RATE int
);
```

Triggers:

```
CREATE FUNCTION insertpatient()
  RETURNS trigger AS
$$
BEGIN
        INSERT INTO employee(emp_id,emp_fname,emp_lname)
        VALUES(NEW.emp_id,NEW.emp_fname,emp_lname);

    RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER pattrigger
  AFTER INSERT
  ON employee
  FOR EACH ROW
  EXECUTE PROCEDURE insertpatient();
```

```
CREATE FUNCTION update_med() RETURNS TRIGGER
AS $$

BEGIN
UPDATE medicine
SET reorder = 1
WHERE medicine_code = NEW.medicine_code;
RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER trmedUpd AFTER UPDATE ON
medicine FOR EACH ROW
WHEN (OLD.QUANTITY_ON_HAND IS DISTINCT FROM NEW.QUANTITY_ON_HAND
AND NEW.QUANTITY_ON_HAND < NEW.MIN_QUANTITY_ON_HAND) EXECUTE
FUNCTION update_med();
```

Population:

- Pharmacist:

```
insert into pharmacist(emp_id,emp_wh, emp_shift_start, emp_shift_end, emp_fname,emp_lname,emp_type, ph_phone)
values (2,'24:00:00', '8:00:00','20:00:00','Mehdi', 'Zahiri', 'pharmacist',0661722181);
```

- Receptionist:

```
25
26  insert into receptionist (EMP_ID,emp_wh,
27      emp_password,
28      emp_shift_start,
29      emp_shift_end ,
30      emp_fname,
31      emp_lname,
32      emp_type)
33      values(19,'24:00:00','safae22','6:00:00','20:00:00','Safae','Salami','Receptionist');
34
35
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 57 msec.
```

- Nurse

```
insert into nurse(EMP_ID,emp_wh,
    emp_password,
    emp_shift_start,
    emp_shift_end ,
    emp_fname,
    emp_lname,
    emp_type,assigned_rooms)
    values(2,'24:00:00','karim122','6:00:00','20:00:00','Karima','Salami','nurse','7');
```

- Doctor

```
insert into doctor (EMP_ID,emp_wh,
    emp_password,
    emp_shift_start,
    emp_shift_end ,
    emp_fname,
    emp_lname,
    emp_type,doc_email,doc_phone)
    values(10,'24:00:00','james123','6:00:00','20:00:00','James','Kim','Doctor','james100@gmail.com','9761');
```

- Employee

```
    INSERT INTO employee (EMP_ID, emp_wh,emp_shift_start,
    emp_shift_end, emp_fname,emp_lname,emp_type) VALUES
 (87, '20:00','6:00','21:00', 'Jake','Lin','Doctor');
```

- medicine:

```
insert into medicine (medicine_code, description, in_date,quantity_on_hand,min_quantity_on_hand,price,discount_rate)
values (2,'RHUMIX, Sachet 400MG','12-12-2022', 58,10,22,0);
```

- department

```
    INSERT INTO department (dep_code,
num_of_rooms ,
num_of_emps ,
manager_name ,
dep_name ) VALUES
 (2, 7,7,'Sam', 'Addmission');
```

- patient:

```
    INSERT INTO department (PATIENT_ID ,
DISEASE_TYPE ,
PATIENT_FNAME ,
PATIENT_LNAME ,
PATIENT_CIN ,
PAT_PHONE ,
PAT_EMAIL ,
PAT_DOB  VALUES
 (2,'cancer','Lina','Kadiri','w1234',0617723122,'lina@gmail.com','20-08-2002');
```

B. Application architecture: technology and tools used and their interaction / web user-friendly interface.

- **CSS and HTML:**

I used CSS to style the layout and appearance of my website or web application. This included setting font sizes and colors, aligning elements on the page, and adding background images and colors.

I created CSS classes and applied them to specific HTML elements to control their appearance. I also used CSS selectors to target elements based on their type, class, or ID.

I learned how to use CSS to create responsive designs that adapt to the size and orientation of different devices, such as smartphones and tablets. This involved using media queries and flexible grid layouts to create layouts that look good on any screen.

- **React bootstrap:**

used React Bootstrap to add pre-designed, responsive components to my React app. These included navbars, buttons, forms, tables, and other common UI elements.

I learned how to customize the appearance and behavior of React Bootstrap components using props and style objects. This allowed me to create a cohesive,

professional-looking interface for my app without having to design and build everything from scratch.

- **Javascript**

I used JavaScript to add interactivity and dynamic functionality to my web projects. This included creating event handlers to respond to user actions, modifying the DOM to update the page in response to user input, and using asynchronous programming techniques to make HTTP requests and work with APIs.

I learned how to use various JavaScript libraries and frameworks, such as React, Angular, and Vue, to build more complex and powerful web applications. I also learned how to use Node.js to build server-side applications and work with databases.

- **Postgres sql**

It is open-source and freely available, which makes it an attractive option for those who want to avoid the costs associated with proprietary software.

It supports a wide range of data types and can store and manage complex data structures, making it well-suited for applications that require the storage and manipulation of complex data.

It has a strong reputation for reliability and performance, and is able to scale to support large amounts of data and high levels of concurrency.

- **Node js**

Building web servers and APIs using JavaScript and a variety of popular libraries and frameworks, such as Express.js and Hapi.js.

Running server-side JavaScript code to power real-time web applications, such as chat applications or live streaming platforms.

Developing scalable, high-performance network applications by taking advantage of Node.js' non-blocking I/O model and event-driven architecture.

Creating command-line tools and other applications that can be run from the terminal or command prompt.

- **Express**

Building web servers and APIs for web applications, mobile apps, and other client-server systems.

Routing HTTP requests to the appropriate handler function based on the URL and HTTP method.

Parsing request data and generating response data in a variety of formats, including JSON, HTML, and XML.

Implementing middleware functions to perform tasks such as authentication, validation, and error handling.

- **React**

Creating reusable UI components that can be easily shared and composed to build complex UIs.

Handling state management and data flow within a UI in a declarative and efficient way.

Integrating with other libraries and frameworks, such as Redux and GraphQL, to build more powerful and scalable applications.

- **Axios:**

It is easy to use and has a simple API, which makes it quick and straightforward to make HTTP requests and handle responses.

It supports a wide range of features, such as automatic transformation of request and response data, support for canceling requests, and support for uploading and downloading binary data.

- **Tailwind CSS**

Tailwind CSS is a utility-first CSS framework that allows for easy customization of design elements in a React app through the use of class names. This means that it is quick and easy to add styling to your app without having to write out complex CSS rules.

Using Tailwind CSS can also help to improve the performance of your React app by reducing the amount of style rules that need to be processed. This can help to speed up the rendering of your app, making it more responsive and enjoyable to use.

IV. Testing and fine-tuning:

As the developer of Pulse, a hospital management system application built with React, it was my responsibility to ensure that the app was functioning correctly and performing optimally. To achieve this, I conducted extensive testing and fine-tuning.

I began by setting up automated testing frameworks and writing unit tests to validate the behavior of individual components and functions. This allowed me to catch any issues early on in the development process and ensure that the app was working as intended.

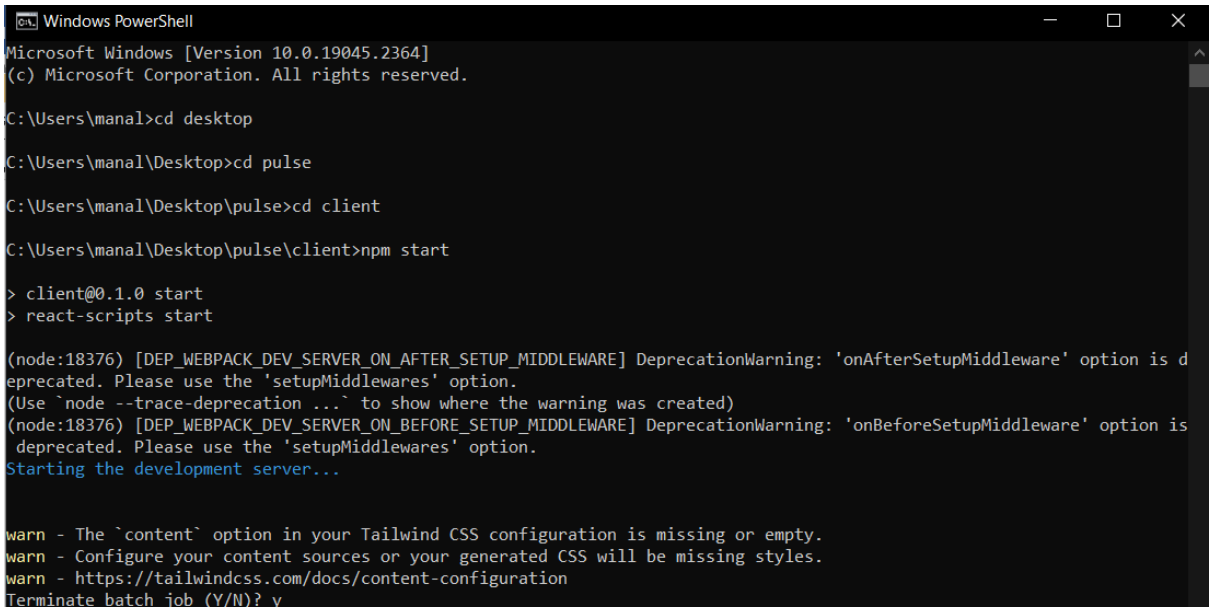
Next, I manually tested the app by interacting with it in the browser and simulating different user scenarios. This included testing the app on different devices and browsers to ensure that it was responsive and worked correctly on all platforms.

As I was testing, I identified and fixed any bugs or issues that I encountered. This included debugging code to find the root cause of problems and writing code to address them. I also optimized the app's performance by minimizing the number of re-renders, reducing the size of assets, and using techniques such as lazy loading to improve load times.

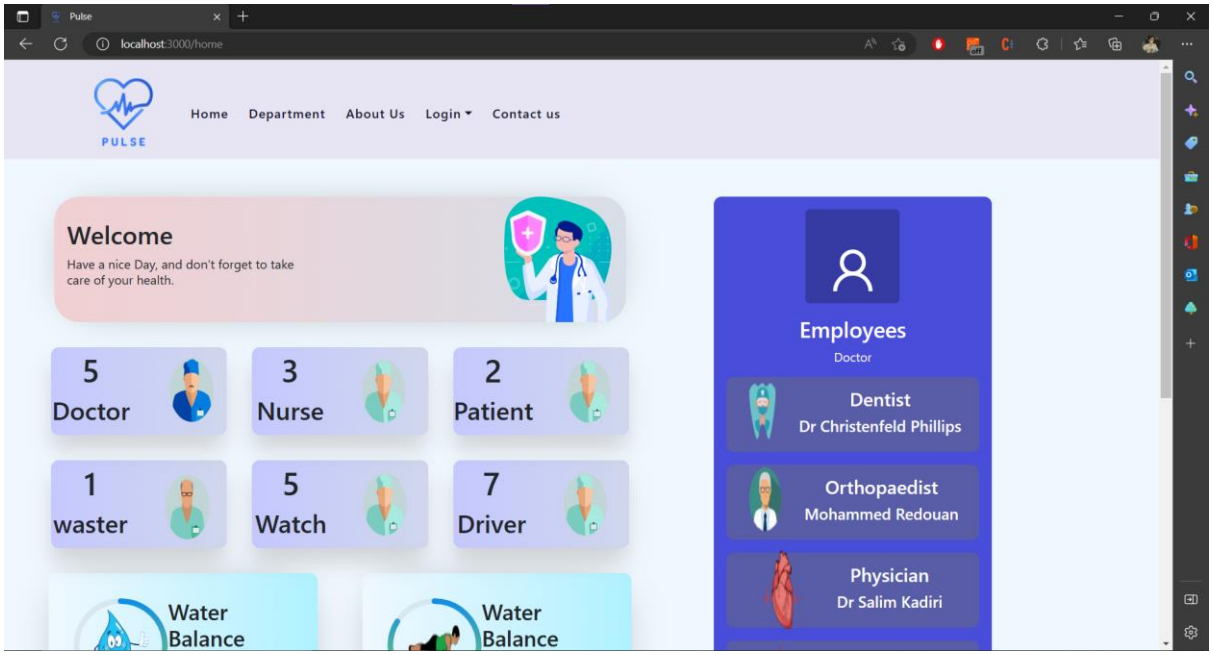
Overall, the testing and fine-tuning process allowed me to identify and fix any issues with the app and ensure that it was working smoothly and efficiently for all users. As a result, the hospital staff were able to use Pulse to efficiently manage their patients and streamline their workflow, leading to improved patient care and satisfaction.

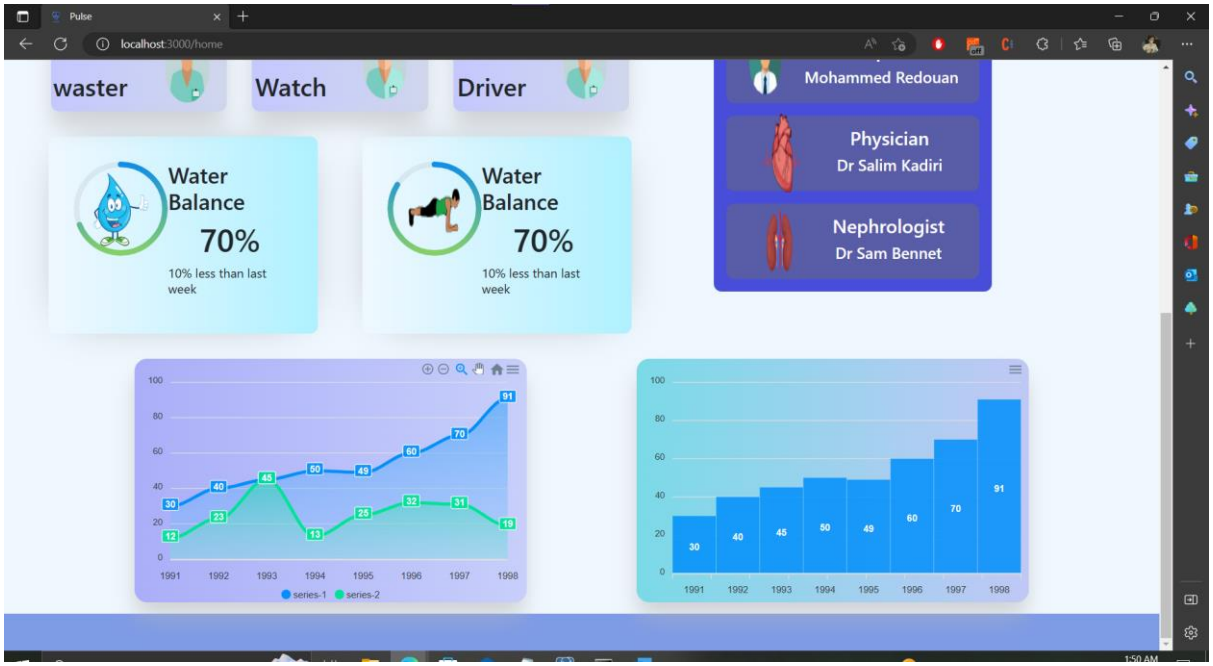
V. User manual: Description of menus, forms, screens' snapshots...

- Access cmd and enter to the file and do npm start to start project:

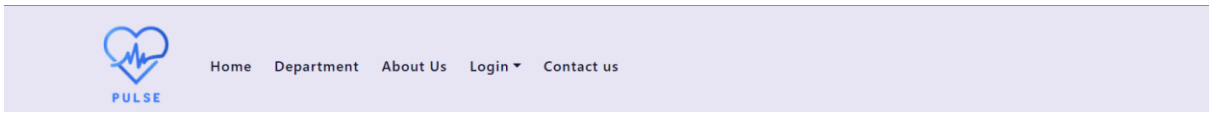


-Home Page:

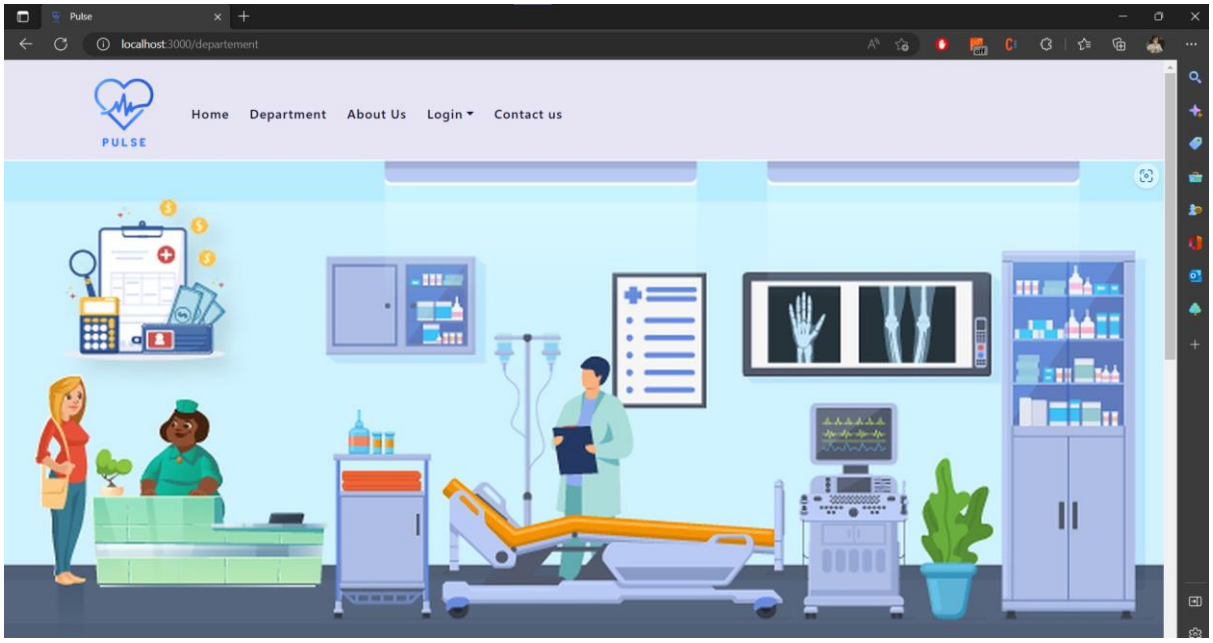


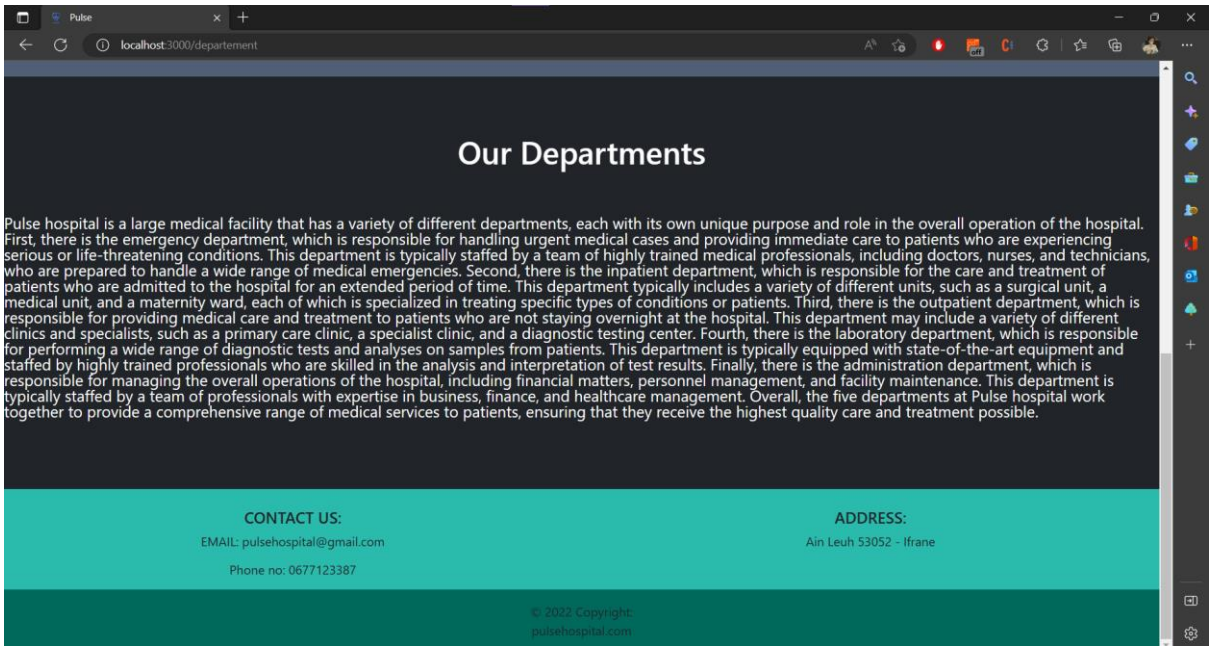


Navigation bar and logo:

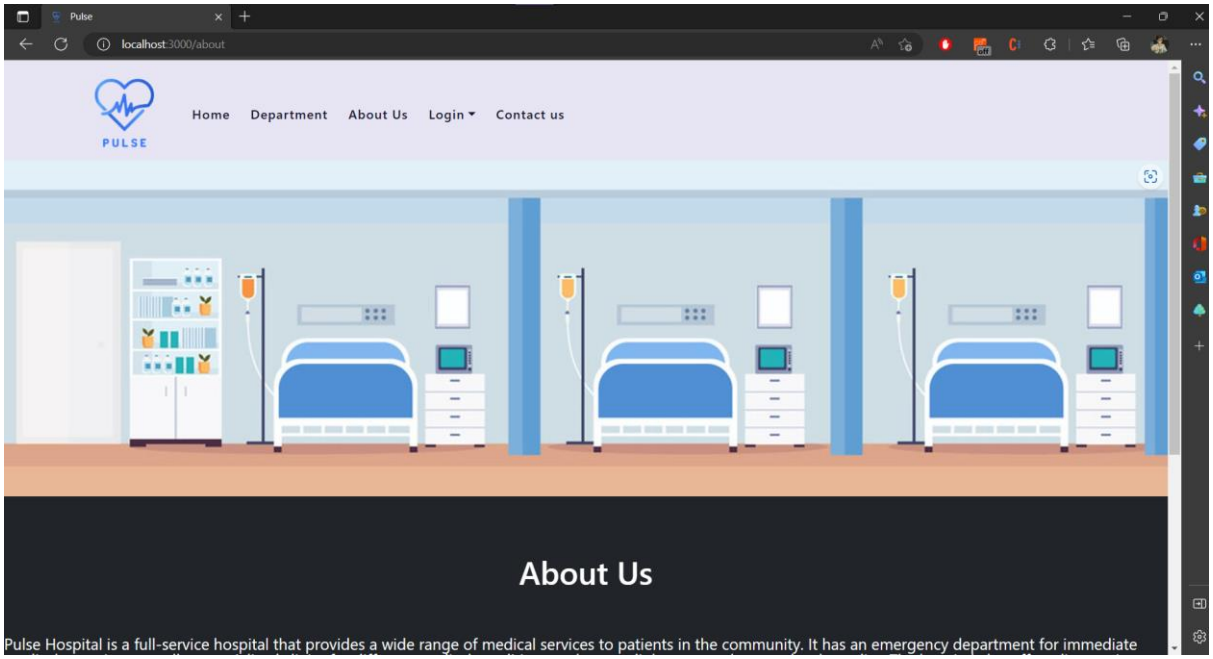


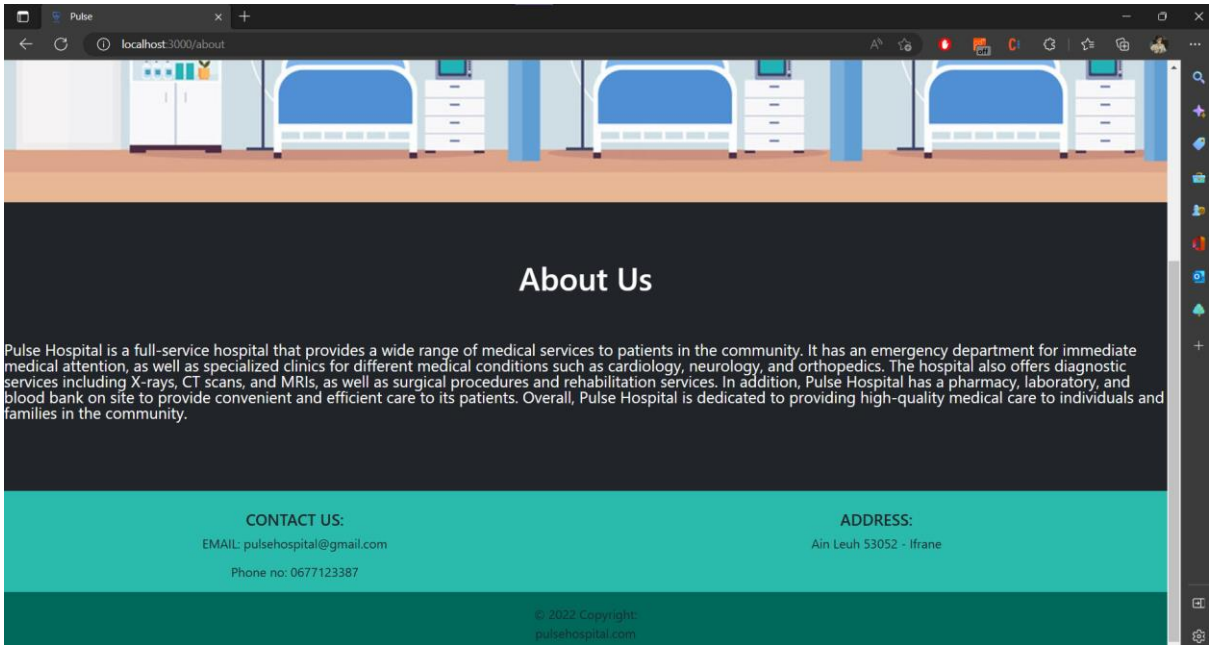
- Click on department you get info about our departments



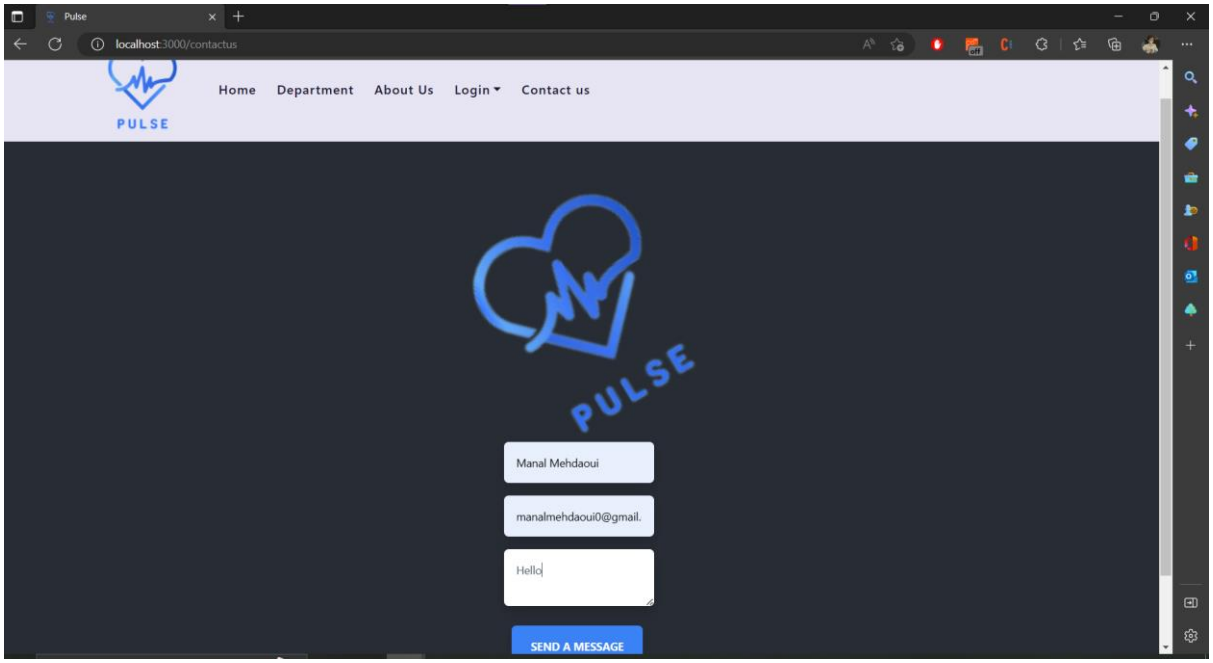


- Click on About Us you Get About Us Page:

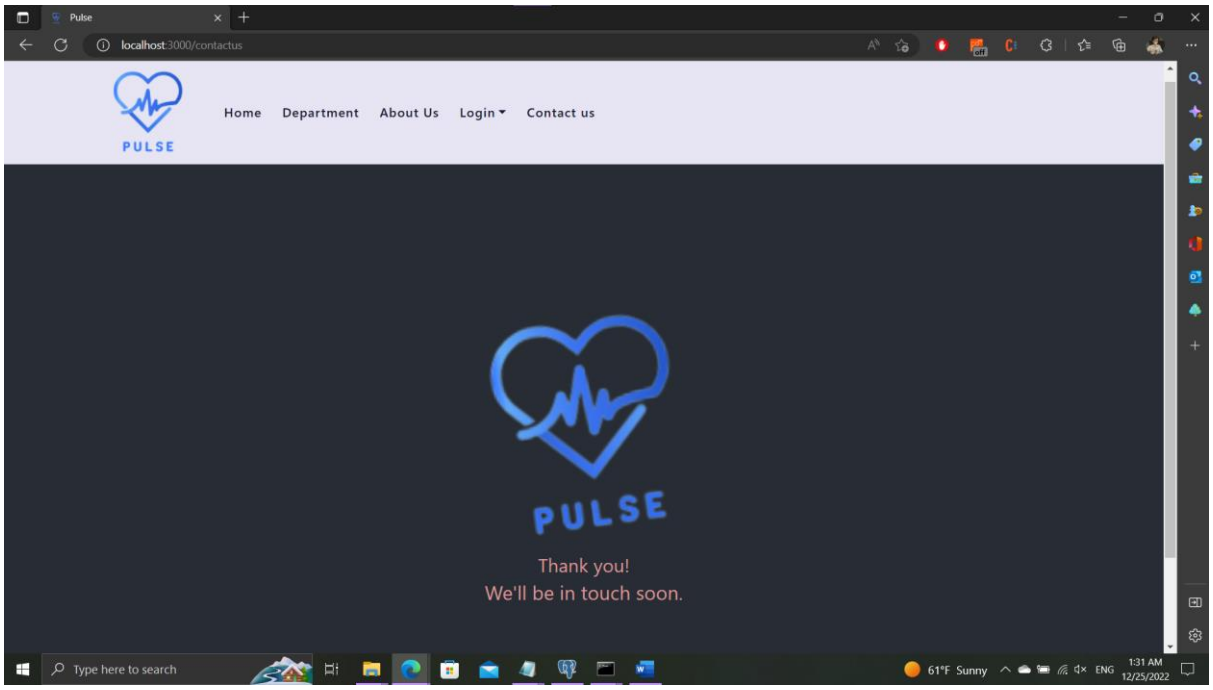




- Click on contact us you get contact us page, where you can enter your information and contact us



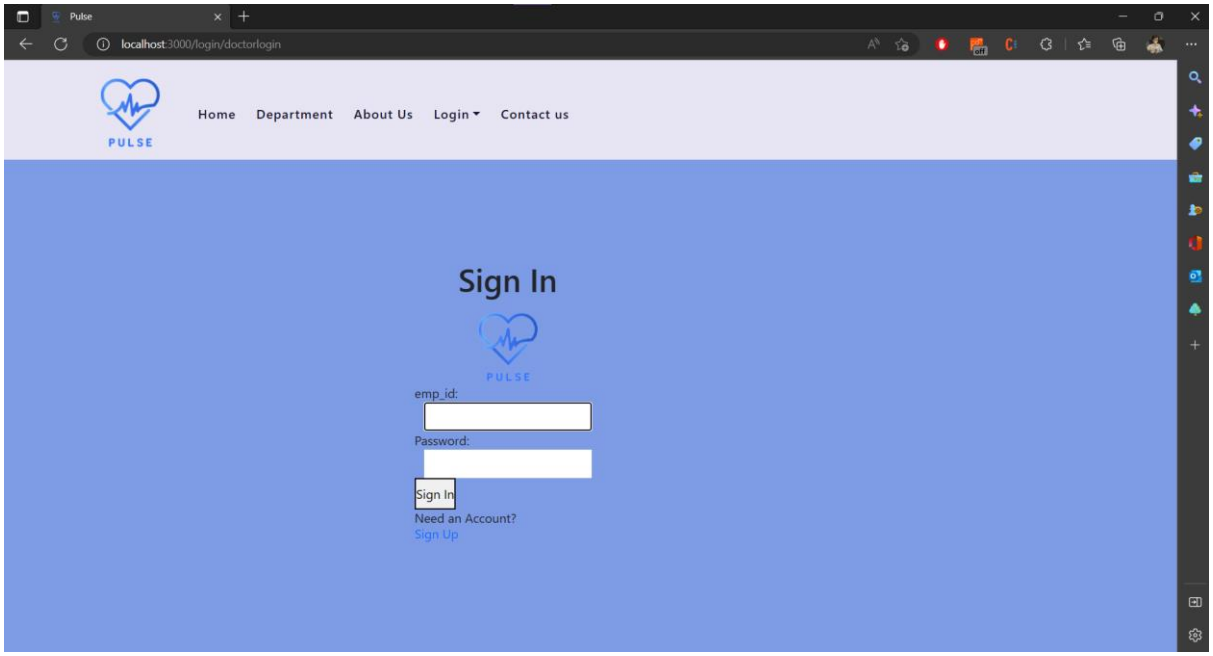
- Click on send and it will be sent



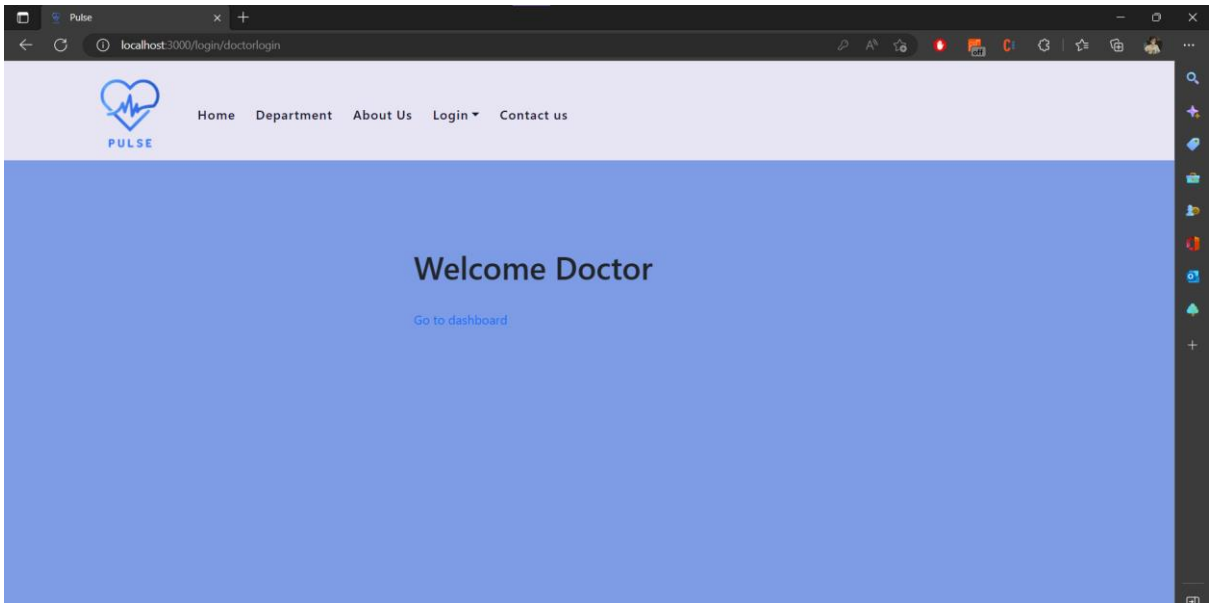
- **<u>Now you can Login as doctor/ patient/pharmacist/or receptionist</u>**

<u>As a doctor:</u>

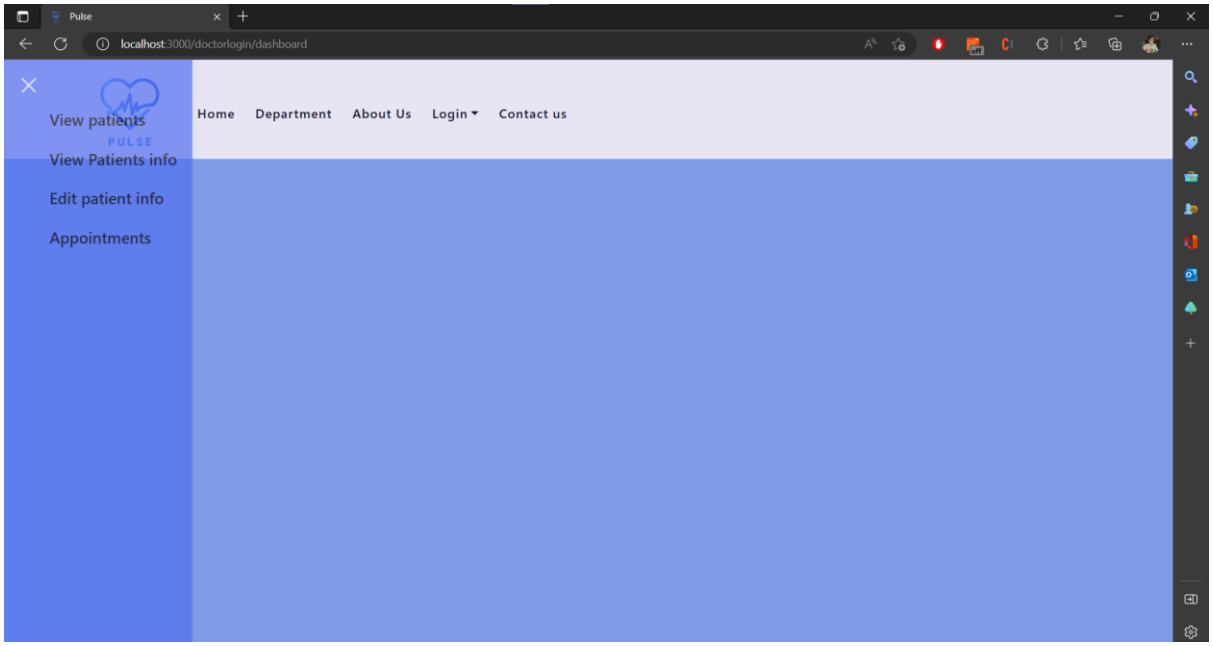
- You click on login doctor and enter the information for sign up



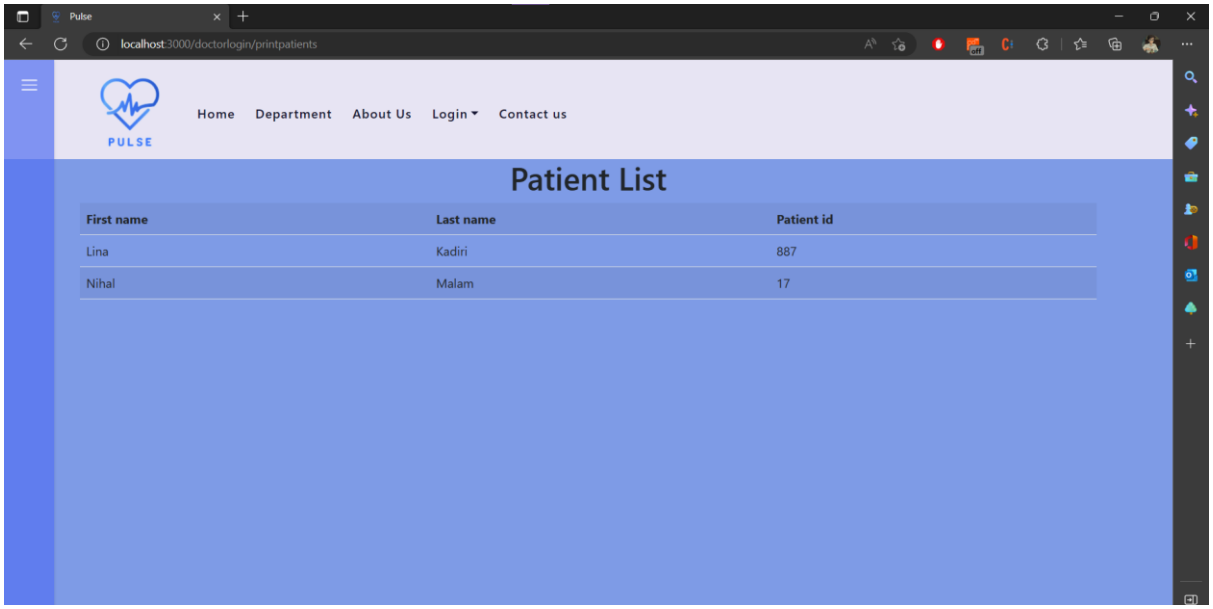
-after sign in



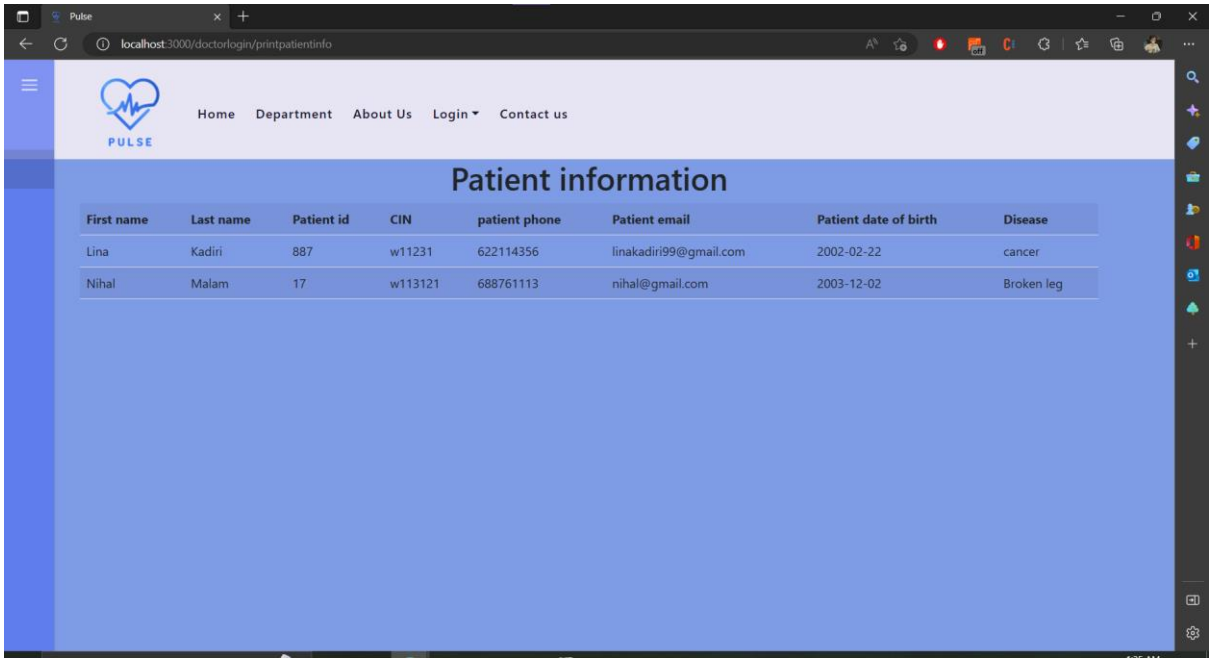
- Click on go to dashboard and see the sidebar



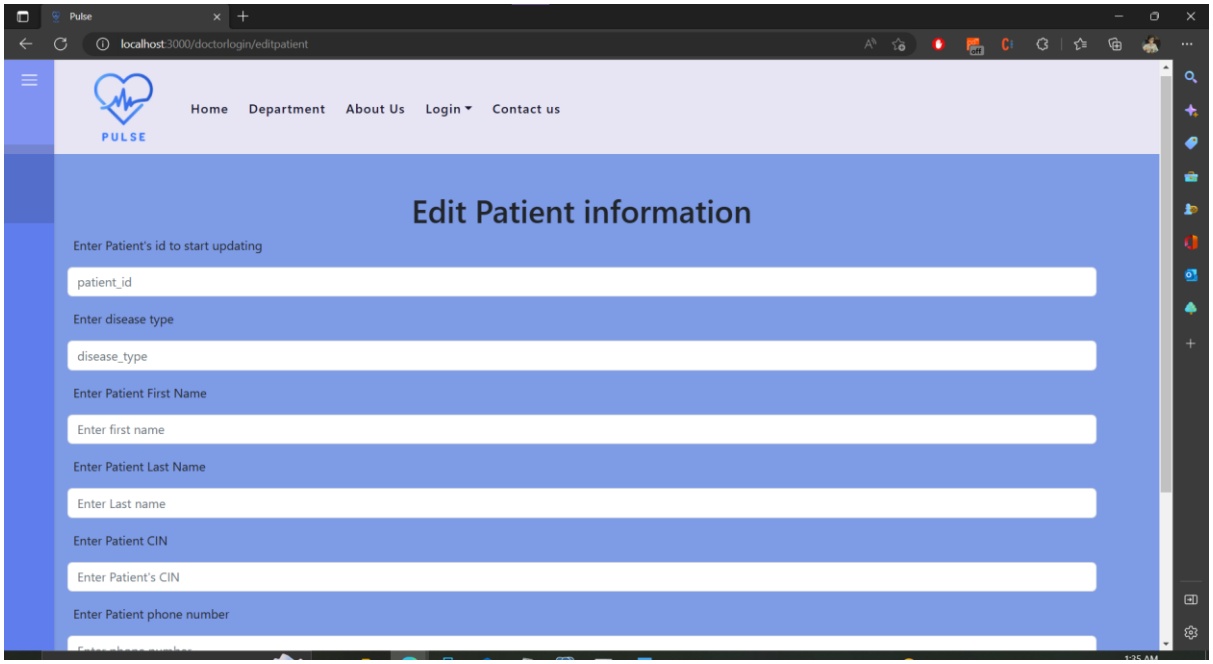
- Click on view patients to see patients

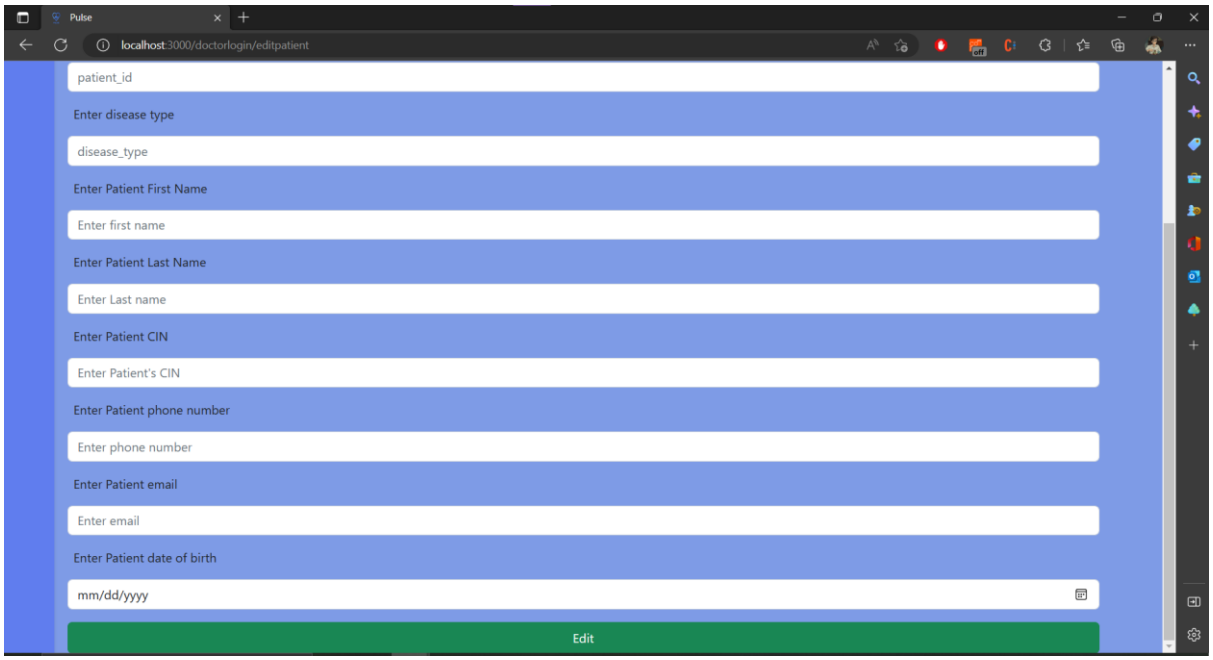


- Click on view patients info to see full patients information

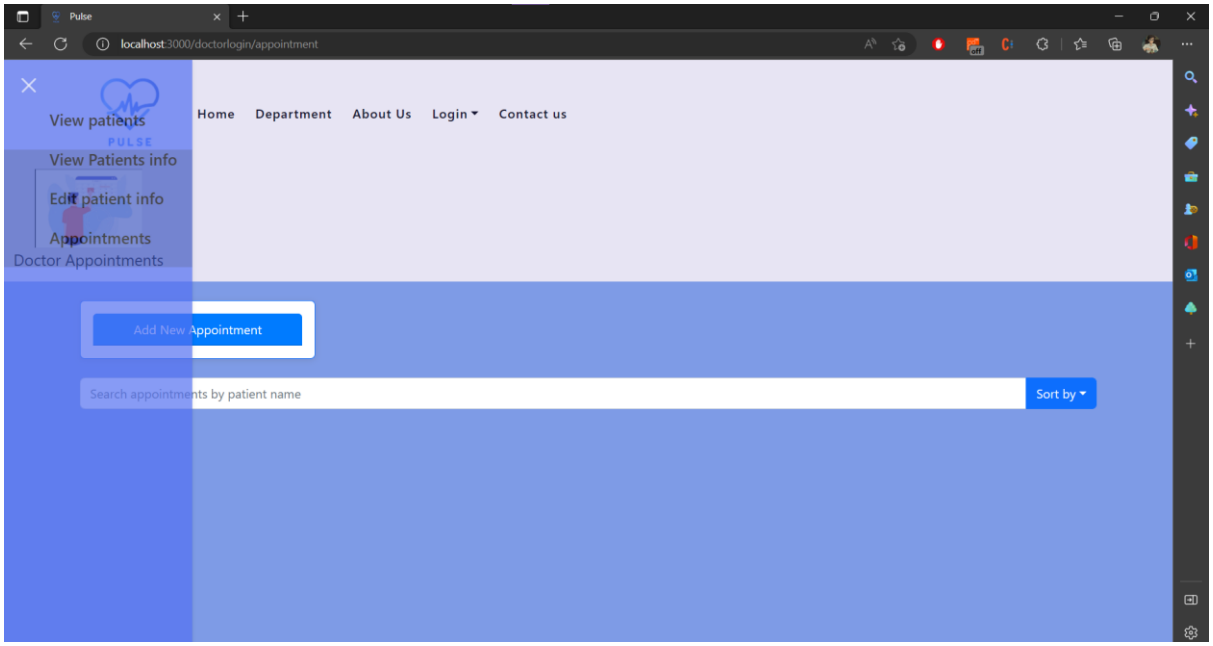


- Click on edit patient info to edit

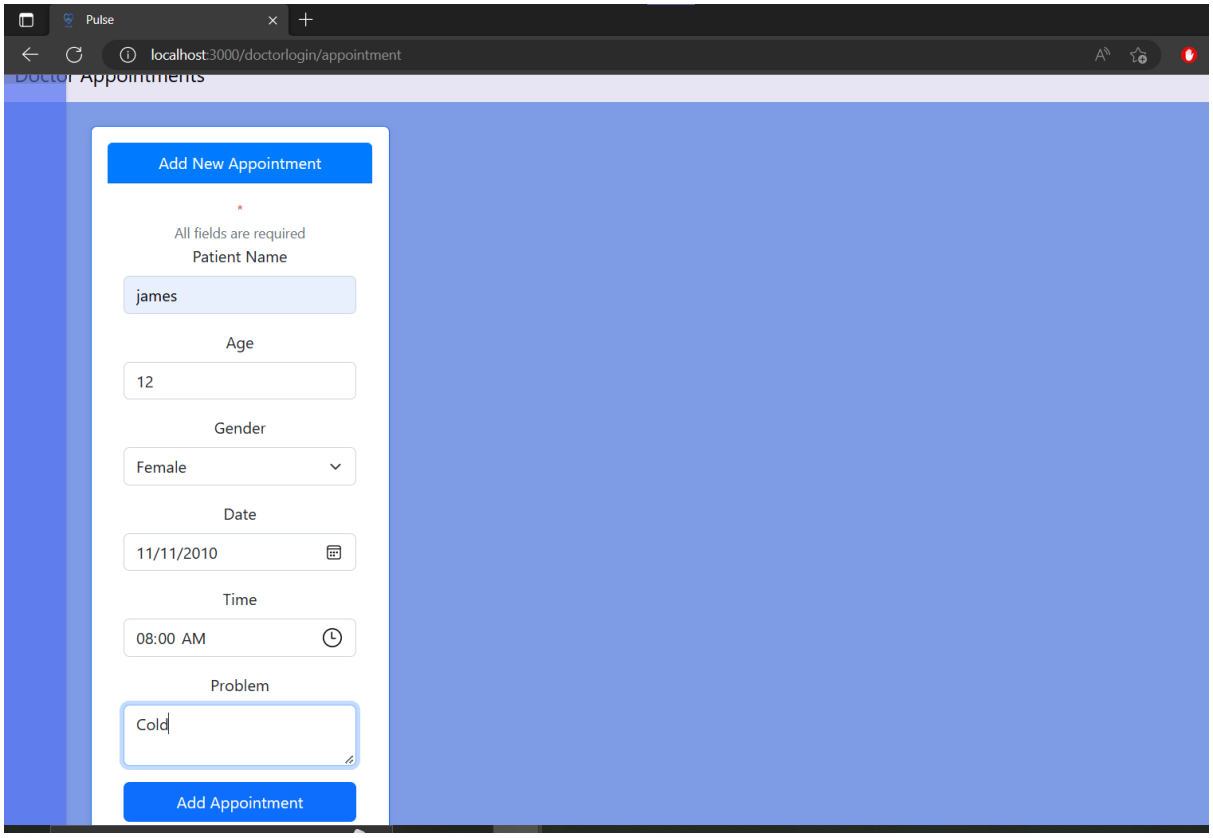




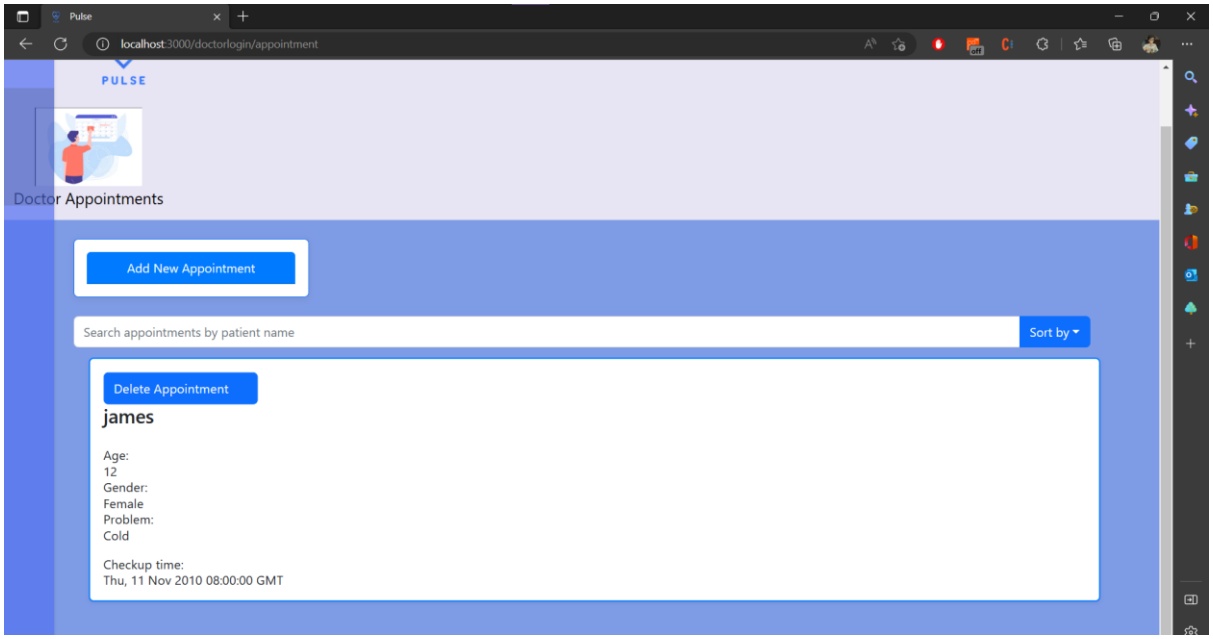
- Click on view appointments to manage appointments:



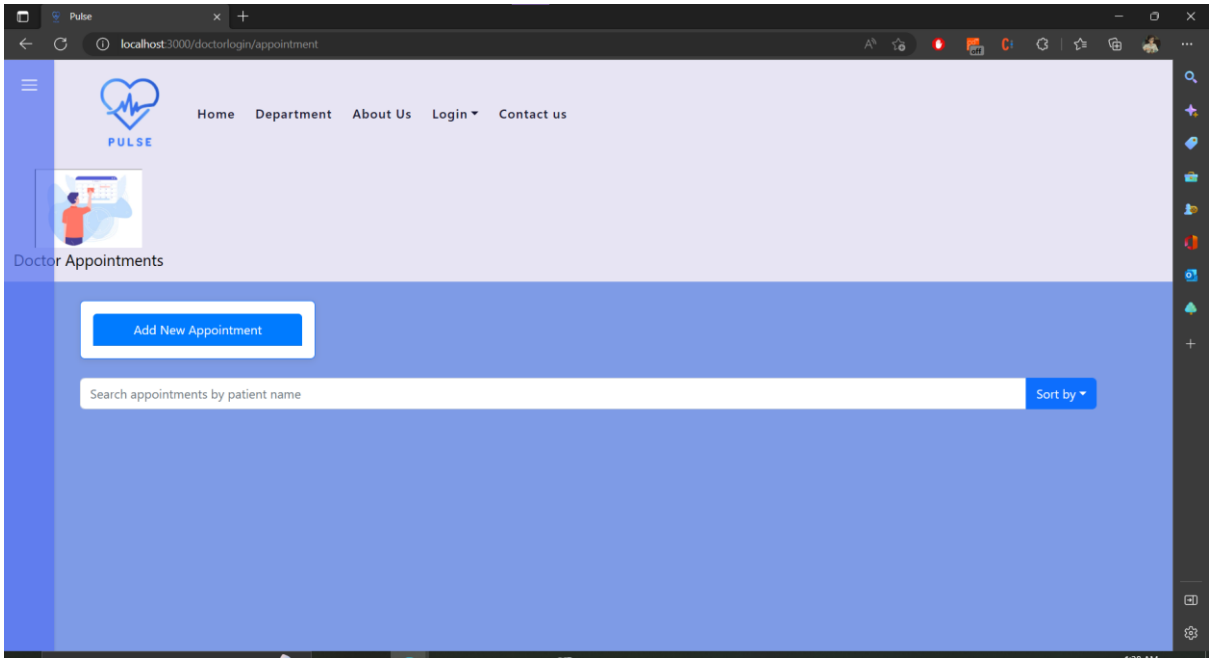
- Click on add appointment and put information



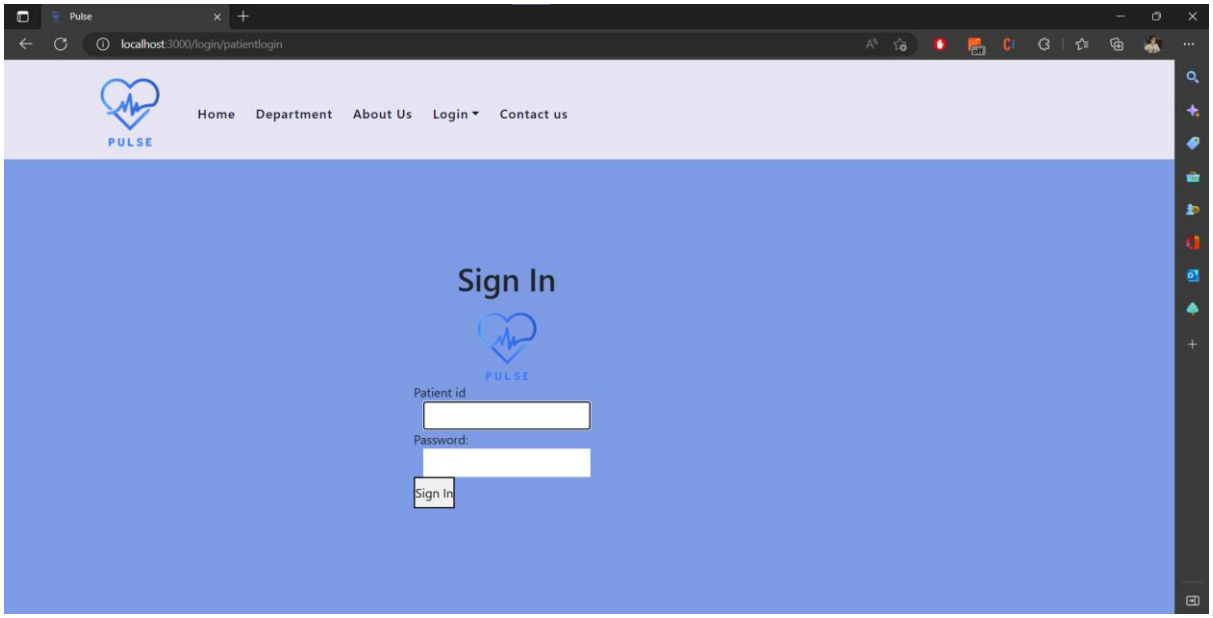
- And add appointment



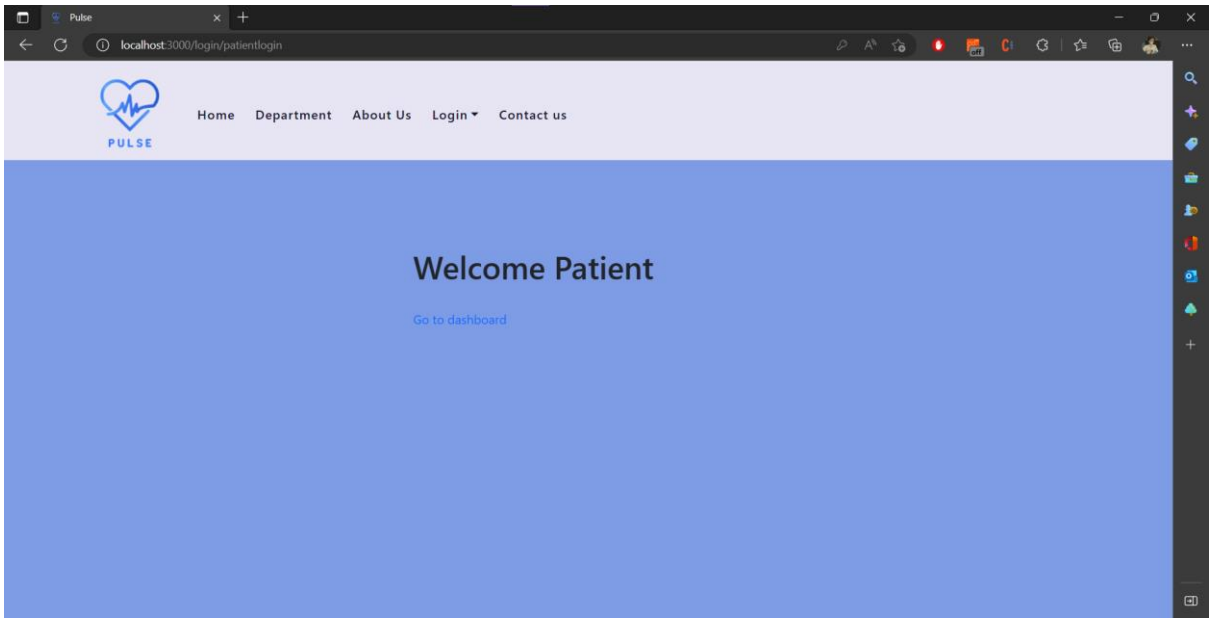
- You can delete appointment by clicking on delete appointment

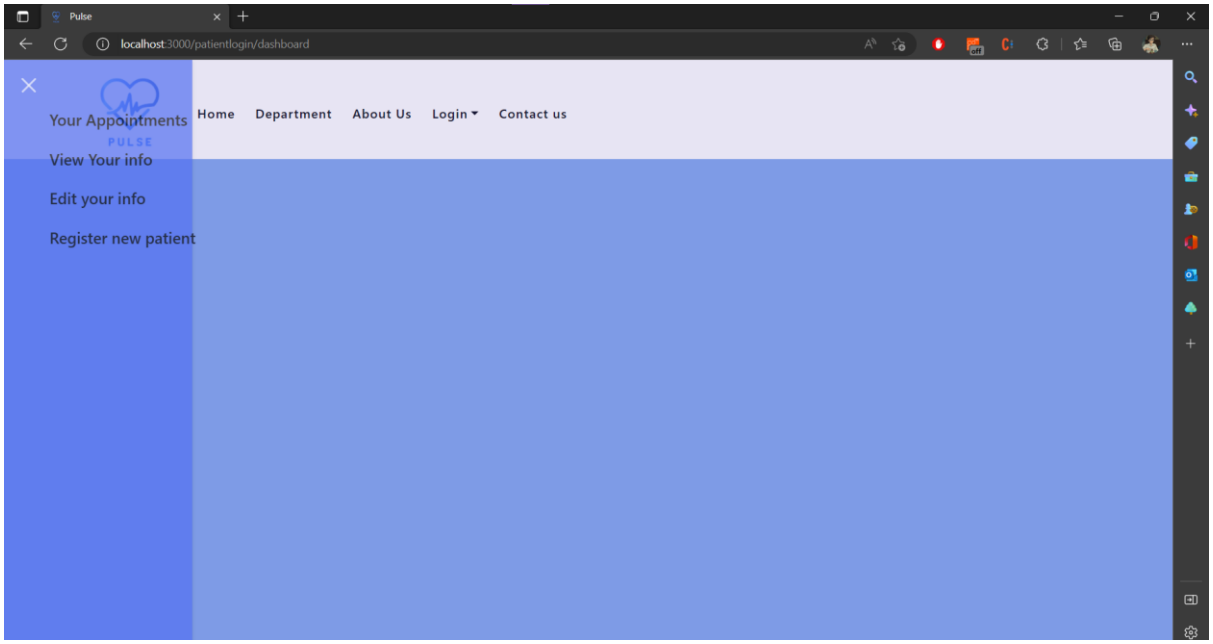


Now go login as a patient:



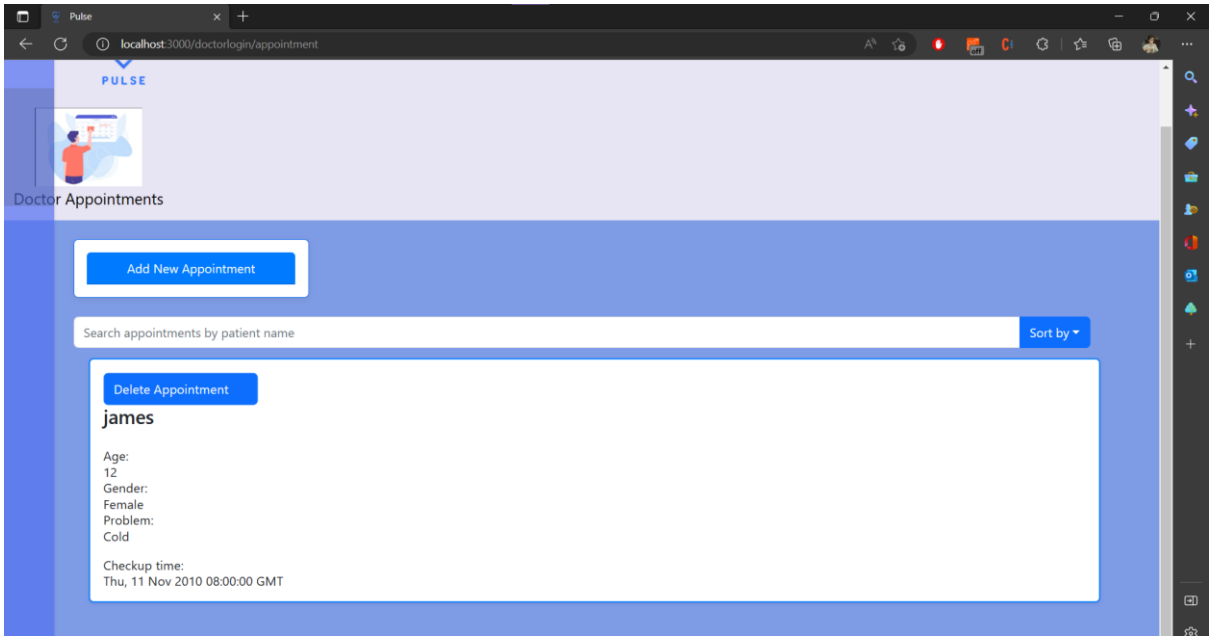
- Enter you info and go to dashboard



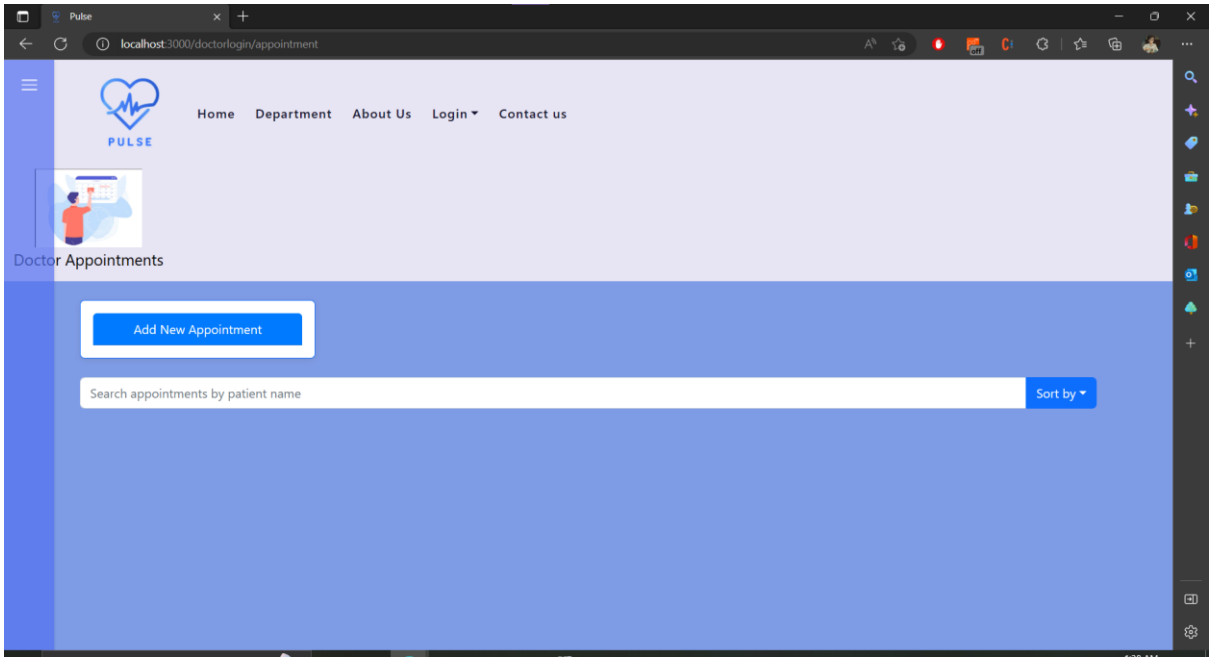


- To manage appointments click on appointments:

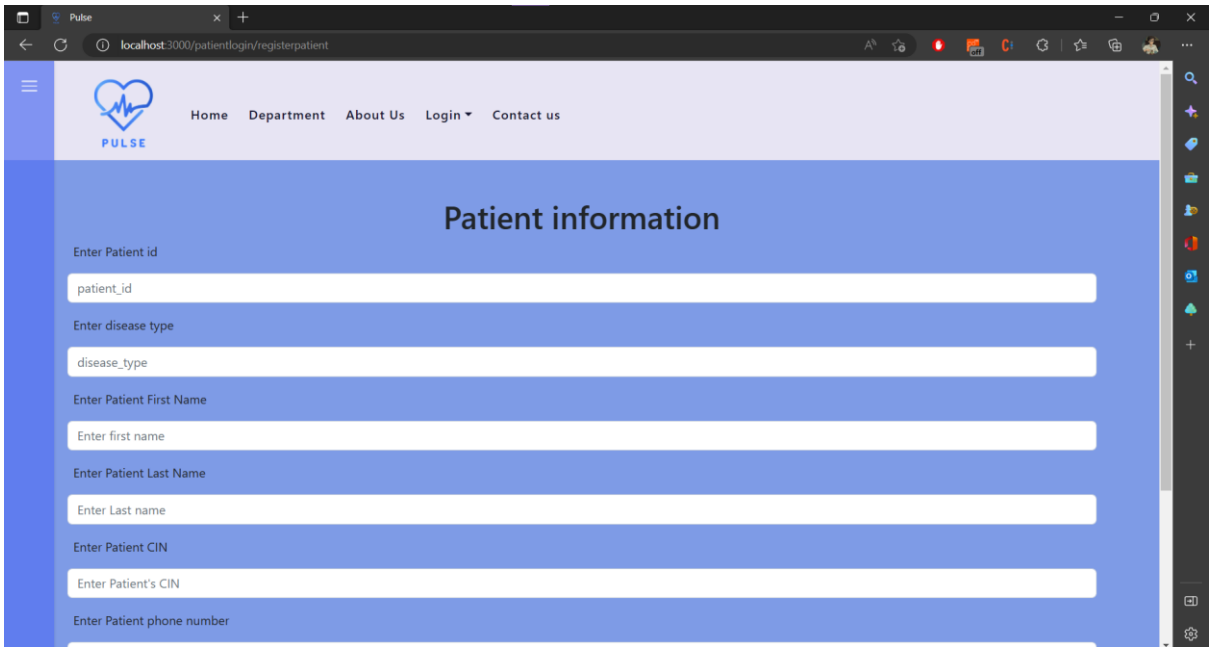
It is the same as the doctors appointments



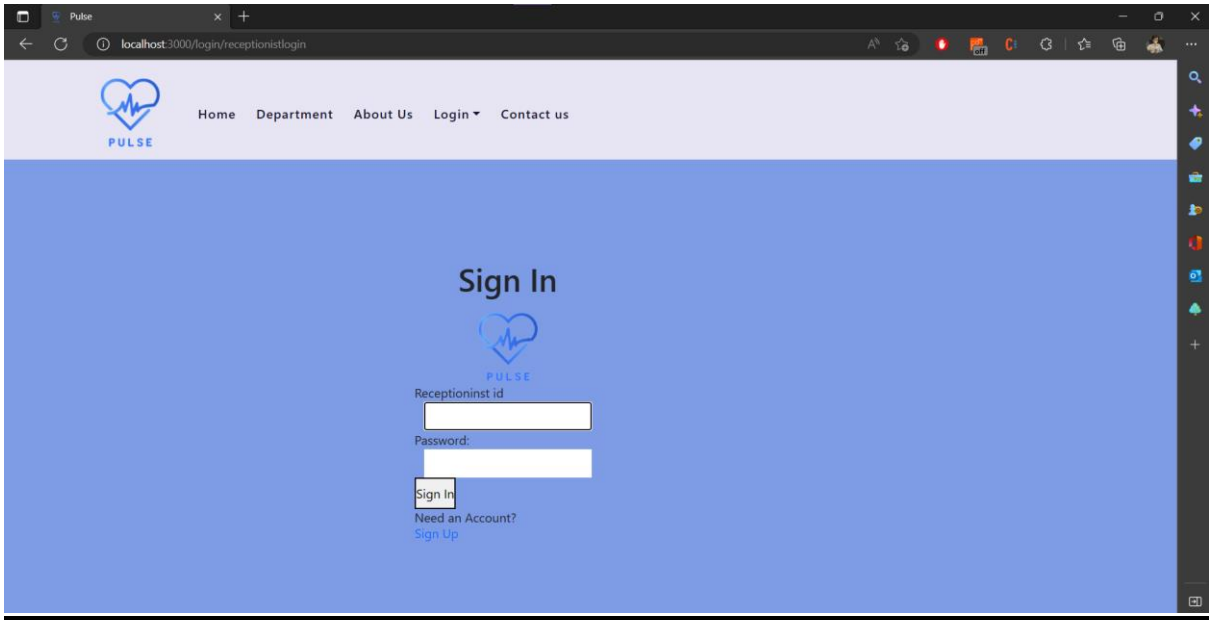
And you can delete



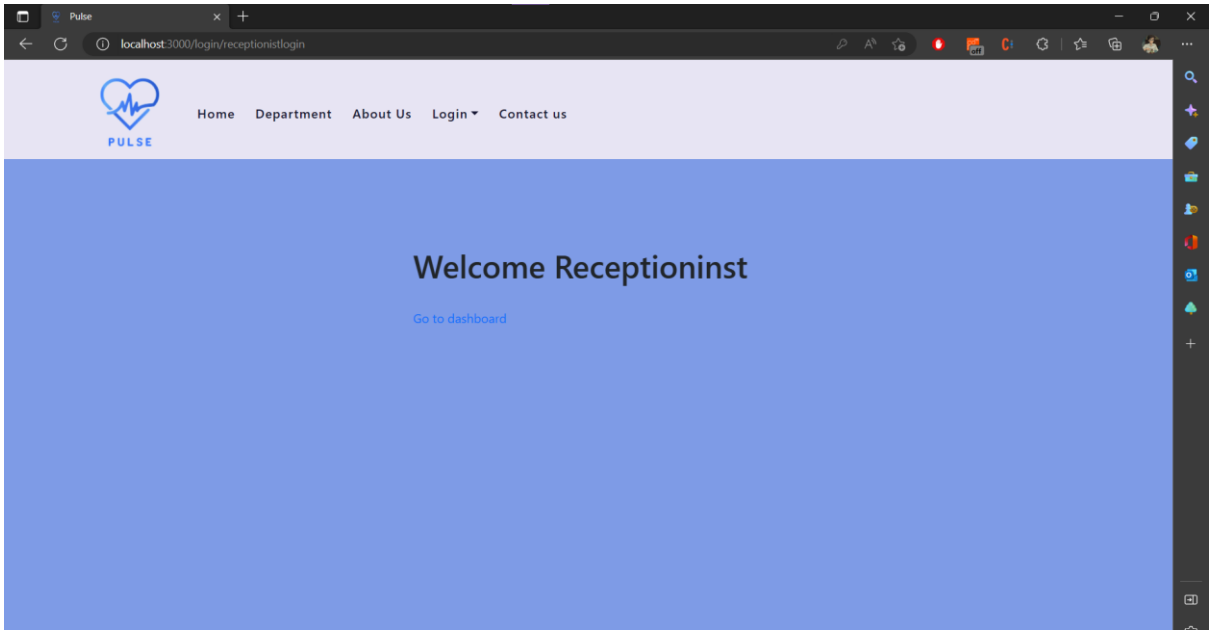
- Go to register a patient and add information

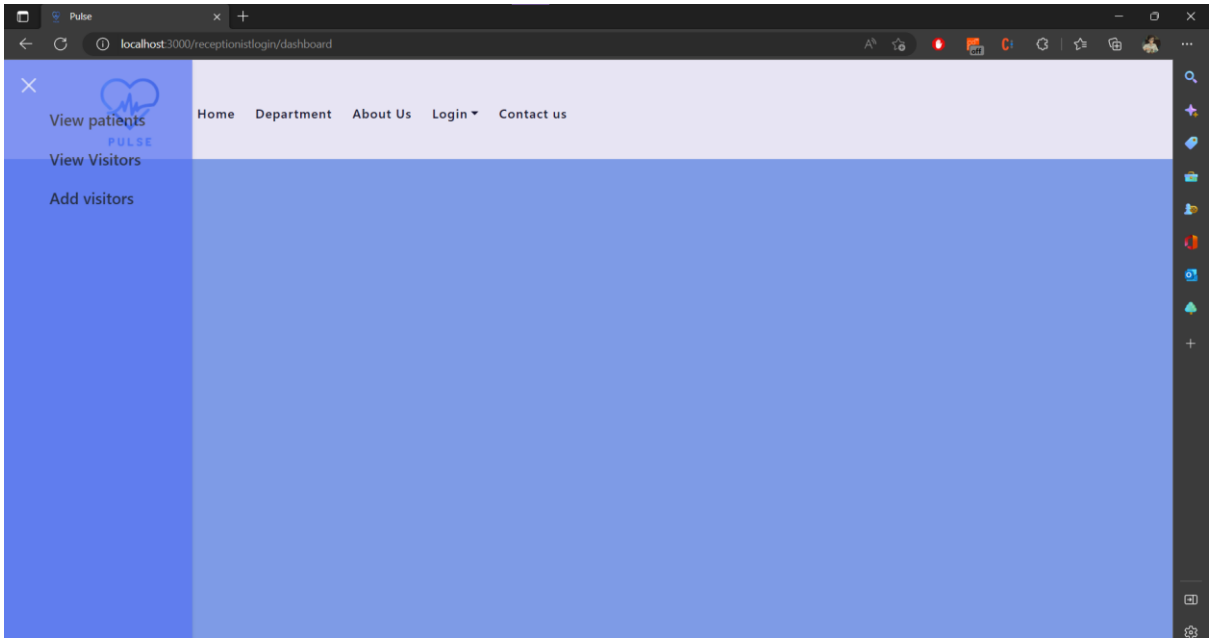


- **Now login as a receptionist:**

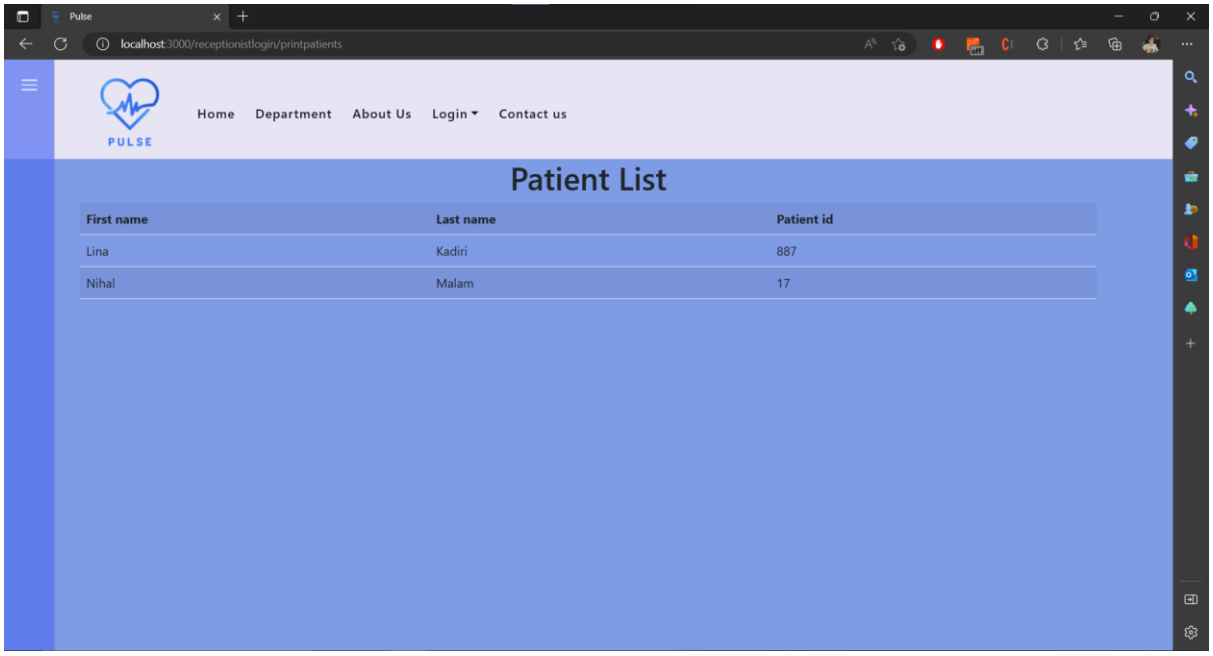


- Enter information and go to dashboard

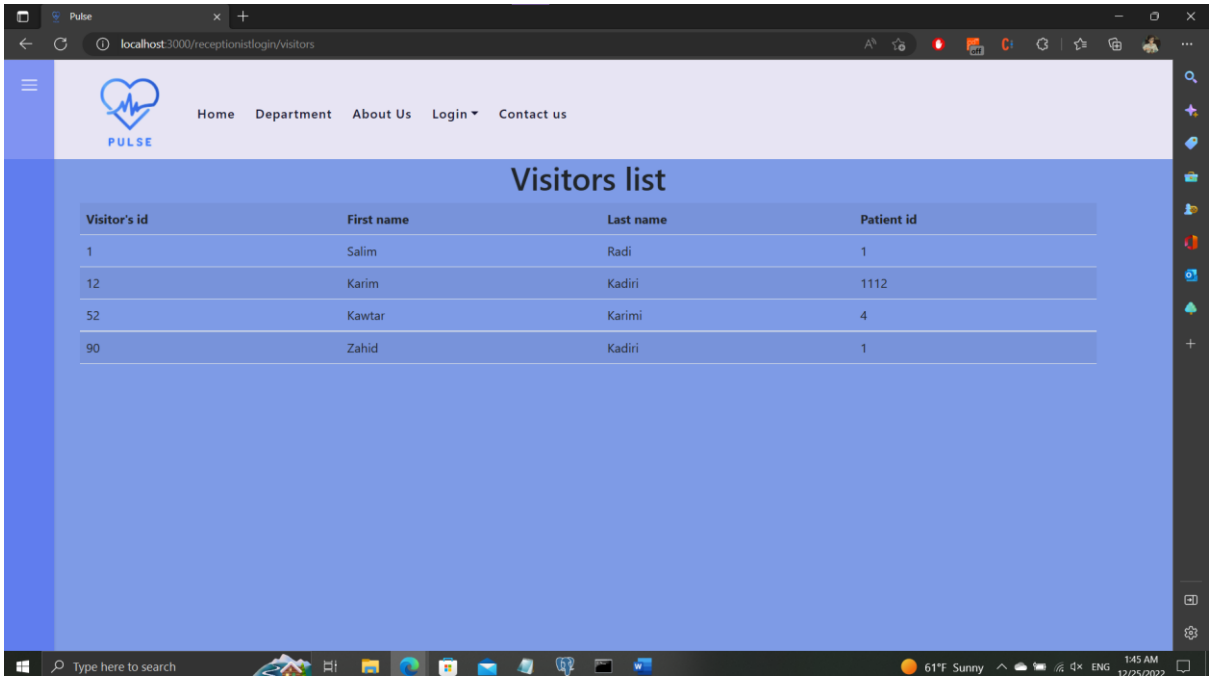




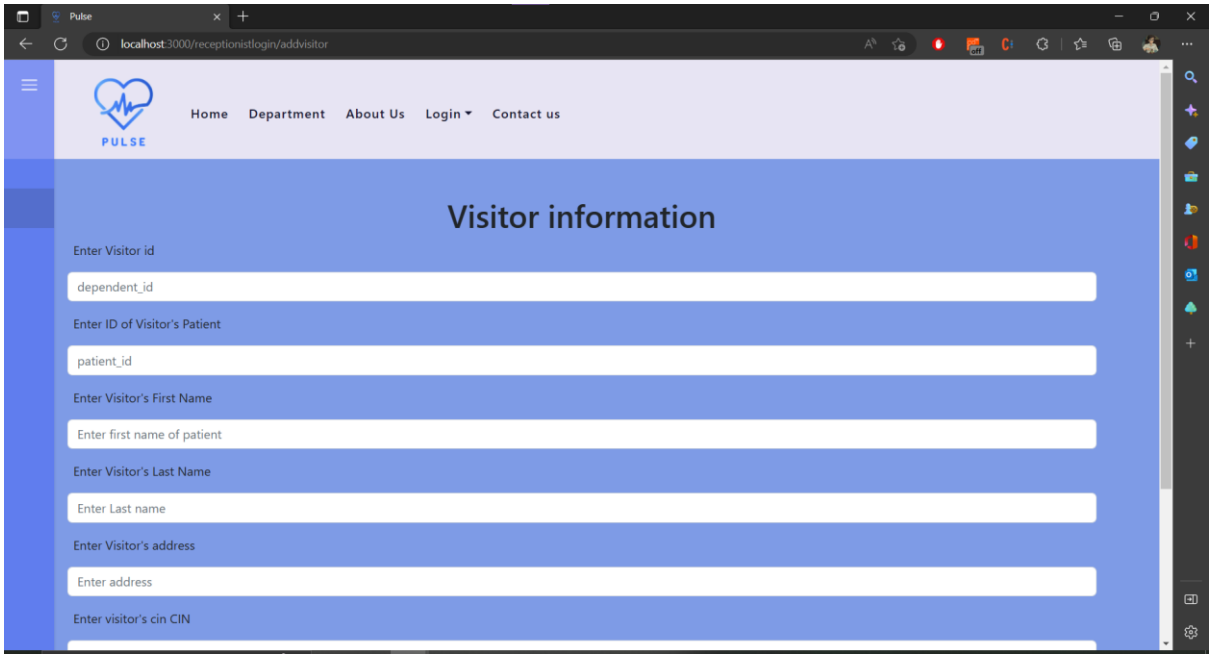
- To view patients click on view patients

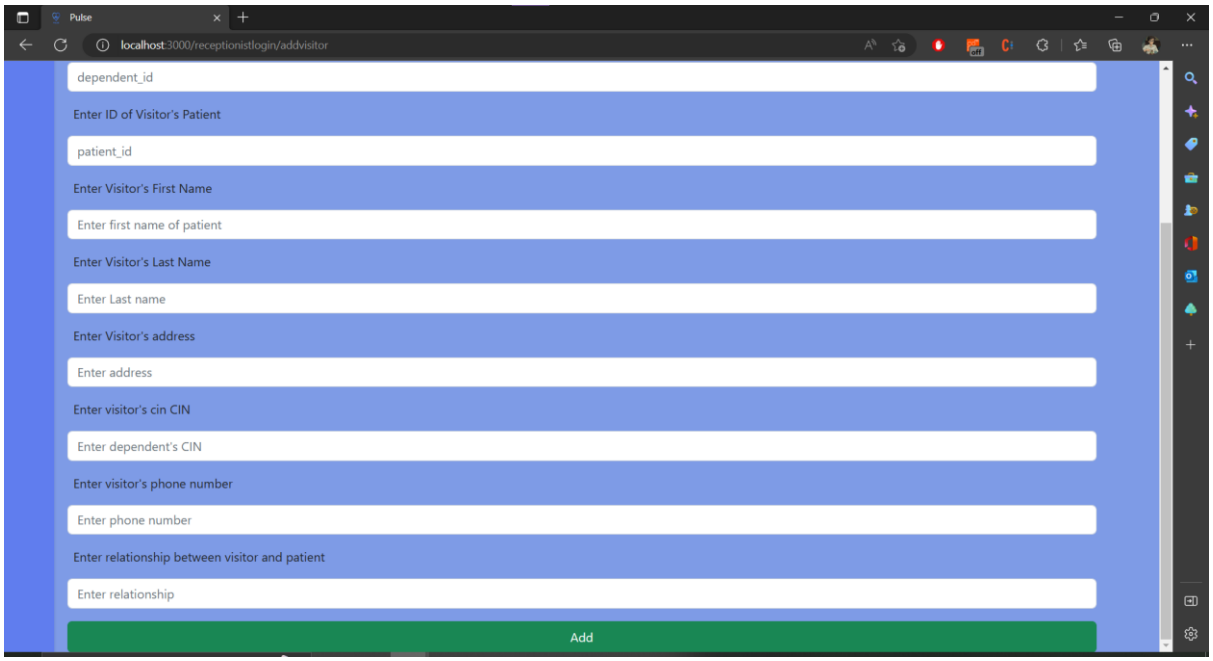


- To view visitors click on view visitors

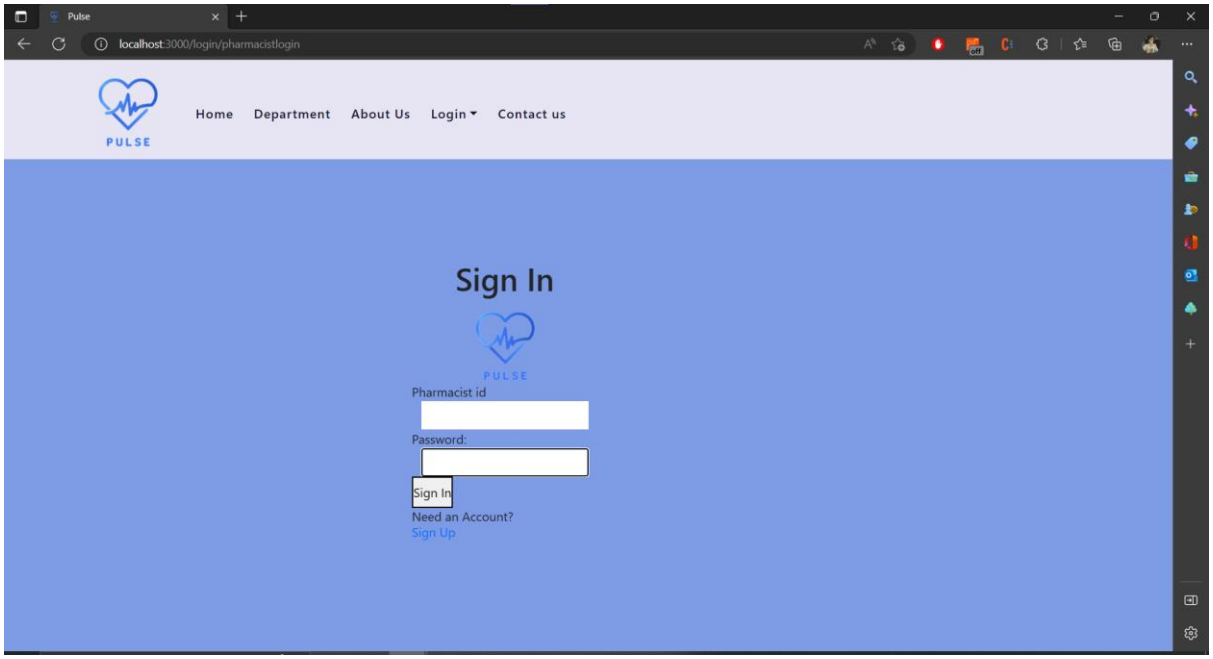


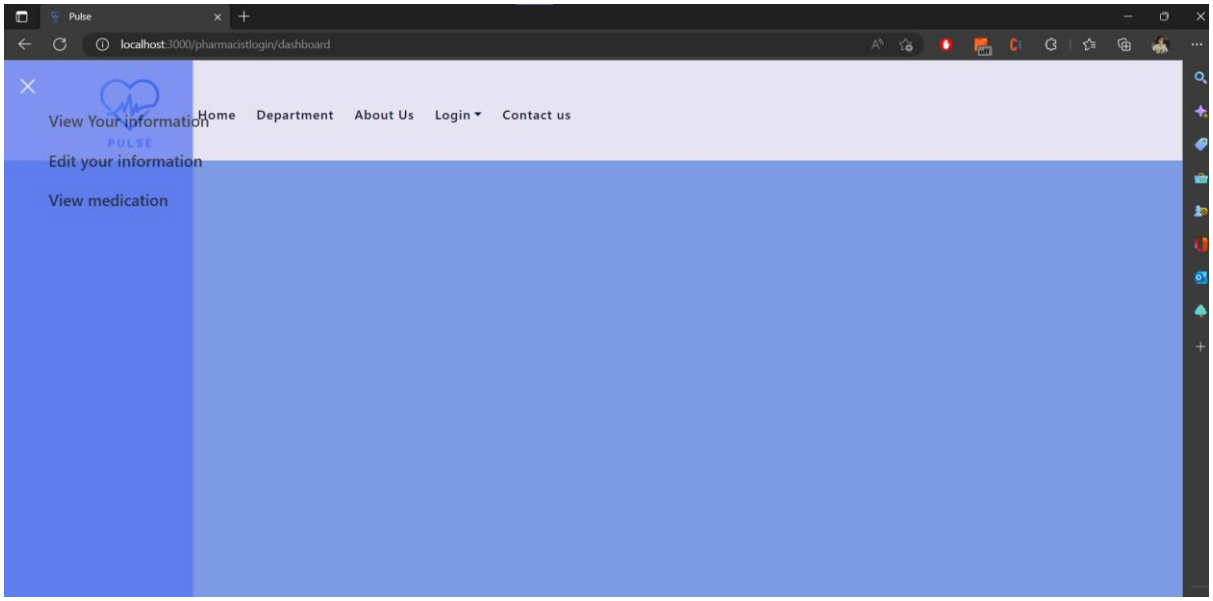
- To add visitors click on add visitors:



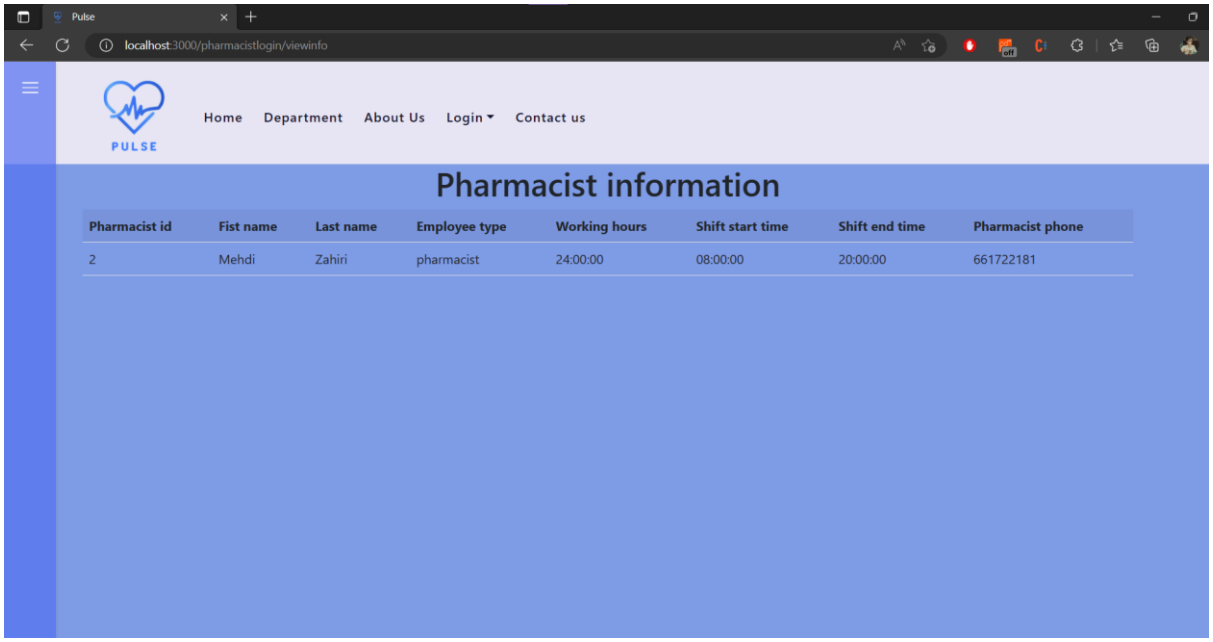


- Now login as a pharmacist
- Enter login details and go to dashboard

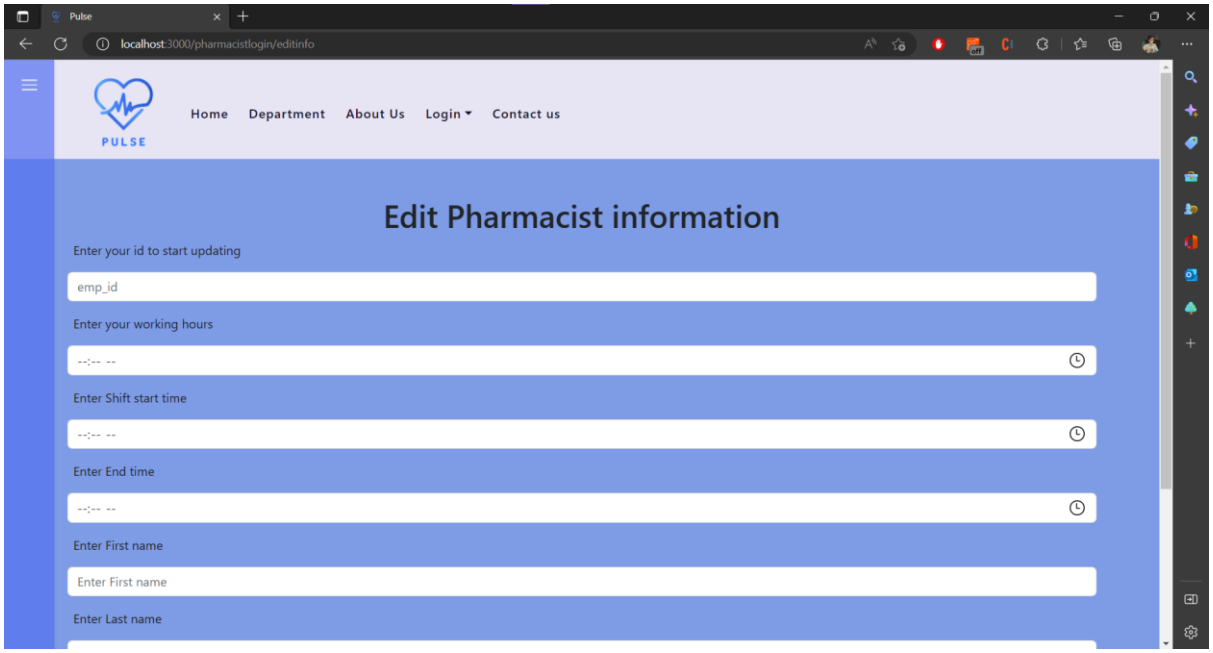


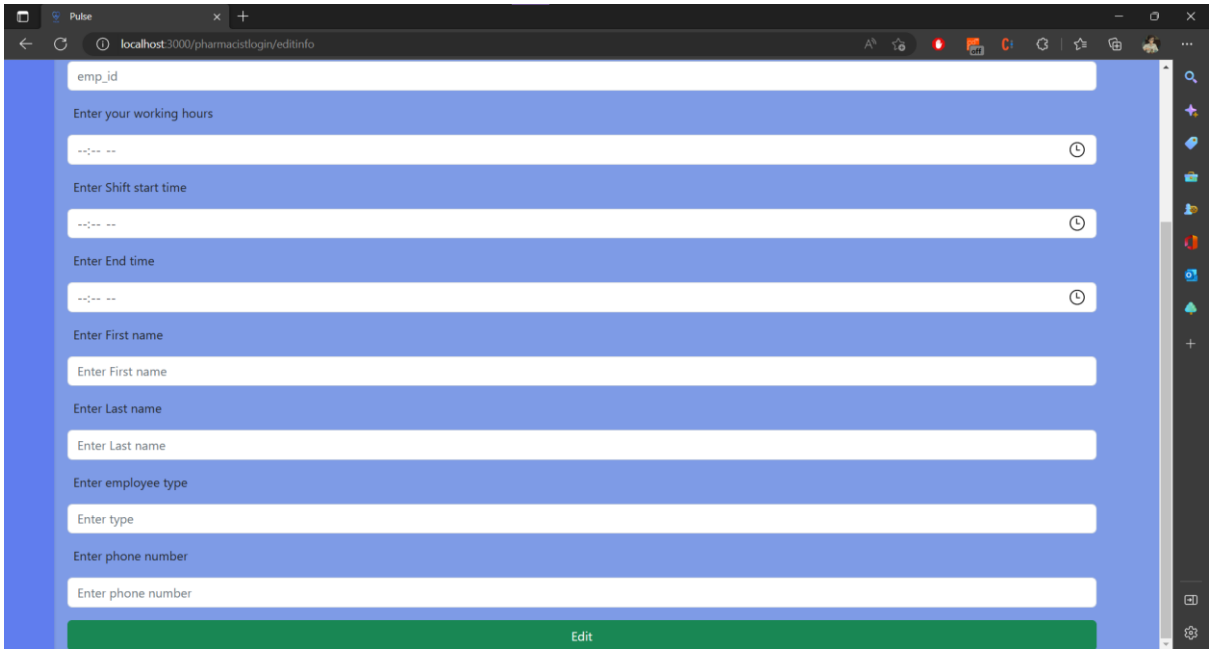


- To view pharmacist information click on view your information:

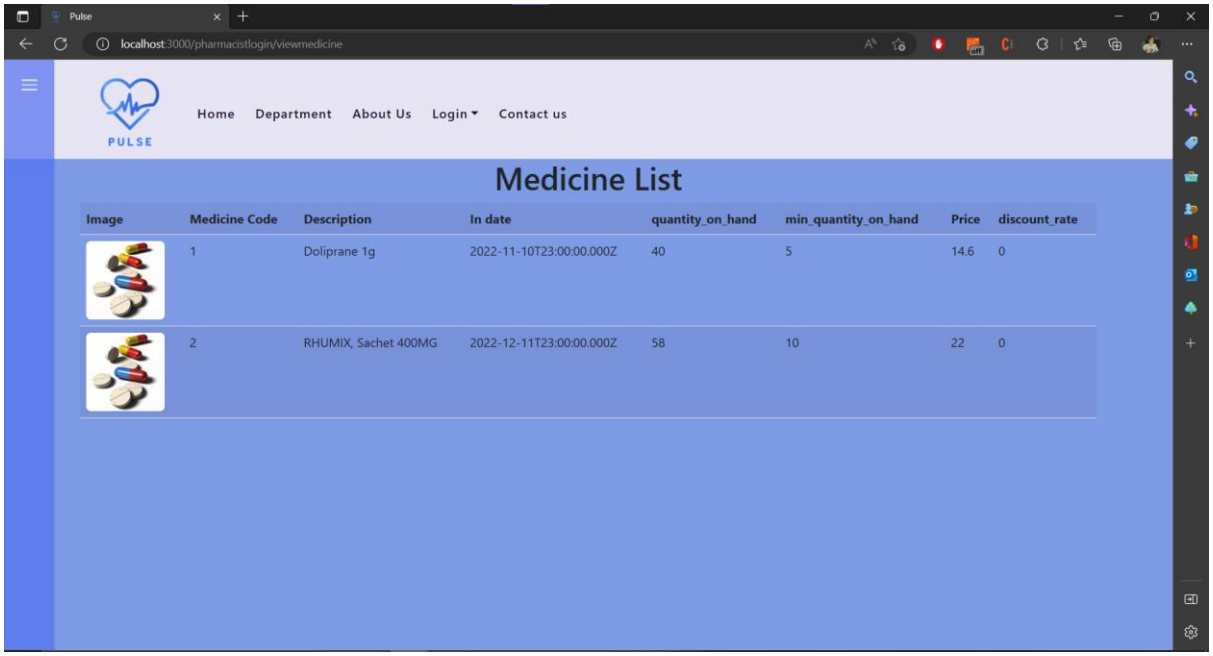


- To edit your information click on edit information





- To see medication click on view medication



That is all thank you.

VIII. Conclusion :

Throughout this project, we've had the chance to learn so much more than what was covered in class. In addition to the fundamentals of the database systems and the pillars of good design, we also got to learn about how to build a front end, and a backend and to link them together, which was also a fundamental aspect of our learning journey. And by fulfilling these tasks we managed to design and implement a medical application 'Pulse' with a user-friendly interface. And we are content with the final version of our project.

IX. Future work (proposed future work to extend/improve your application) :

Future work will include examining the potential for further development of the project. This may include exploring additional features or improvements that can be made to the existing project, as well as the potential for expansion to other areas or applications. Additionally, this project report could include further research of the strategies used, an evaluation of their effectiveness, and the development of a more detailed implementation plan.

Furthermore, we could consider expanding the scope of the database. This could entail integrating more data sources and collecting more data points at a larger scale. Additionally, further research could be conducted on the algorithms and techniques used for query optimization, to increase the efficiency and effectiveness of the system. Furthermore, the system could be extended to facilitate integration with other services and applications. Finally, it would be beneficial to develop a plan for the ongoing monitoring and evaluation of the strategies to ensure that they remain effective.