Health Care Managaement System

Objective:

Health care management system is an online application to store all the health information.

Users of the System:

1. Admin
2. Doctors
3. Patients

Functional Requirements:

- Patients should be able to check doctor availability and book the appointment.
- Doctor should be able to accept or reject appointment.
- Doctor prescription storage and viewable to all departments.
- Pharmacists to provide stock details and delivery of medicines..
- Billing user to bill based on all the transactions done at each department and keep a track of the same.
- **Maximum 1 Patient can book an appointment per slot**.

While the above ones are the basic functional features expected, the below ones can be nice to have add-on features:

- Multi-factor authentication for the sign-in process
- Payment Gateway

Output/ Post Condition:

- Weekly based patient report file
- Standalone application / Deployed in an app Container

Non-Functional Requirements:

Security	• App Platform –UserName/Password-Based Credentials • Sensitive data has to be categorized and stored in a secure manner • Secure connection for transmission of any data
Performance	• Peak Load Performance • Health Care Management System -< 3 Sec • Admin application < 2 Sec • Non Peak Load Performance • Health Care Management System < 2 Sec • Admin Application < 2 Sec
Availability	• 99.99 % Availability
Standard Features	• Scalability • Maintainability • Usability • Availability • Failover
Logging &	• The system should support logging(app/web/DB) & auditing at

Auditing	all levels
Monitoring	• Should be able to monitor via as-is enterprise monitoring tools
Cloud	• The Solution should be made Cloud-ready and should have a minimum impact when moving away to Cloud infrastructure
Browser Compatible	• IE 7+ • Mozilla Firefox Latest – 15 • Google Chrome Latest – 20 • Mobile Ready

Technology Stack

Front End	React Google Material Design Bootstrap / Bulma
Server Side	Spring Boot Spring Web (Rest Controller) Spring Security Spring AOP Spring Hibernate
Core Platform	OpenJDK 11
Database	MySQL or H2

Platform Pre-requisites (Do's and Don'ts):

1. The React app should run in port 8081. Do not run the React app in the port: 3000.
2. Spring boot app should run in port 8080.

Key points to remember:

1. The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.
2. Remember to check the screenshots provided with the SRS. Strictly adhere to id mapping and attribute mapping. Failing to do may fail test cases.
3. Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.
4. Adhere strictly to the endpoints given below.

Application assumptions:

1. The login page should be the first page rendered when the application loads.

2. Manual routing should be restricted by using AuthGaurd by implementing the canActivate interface. For example, if the user enters as http://localhost:3000/signup or http://localhost:3000/home the page should not navigate to the corresponding page instead it should redirect to the login page.
3. Unless logged into the system, the user cannot navigate to any other pages.
4. Logging out must again redirect to the login page.
5. To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.
6. Use admin/admin as the username and password to navigate to the admin dashboard.

Validations:

1. Basic email validation should be performed.
2. Basic mobile validation should be performed.

Project Tasks:

API Endpoints:

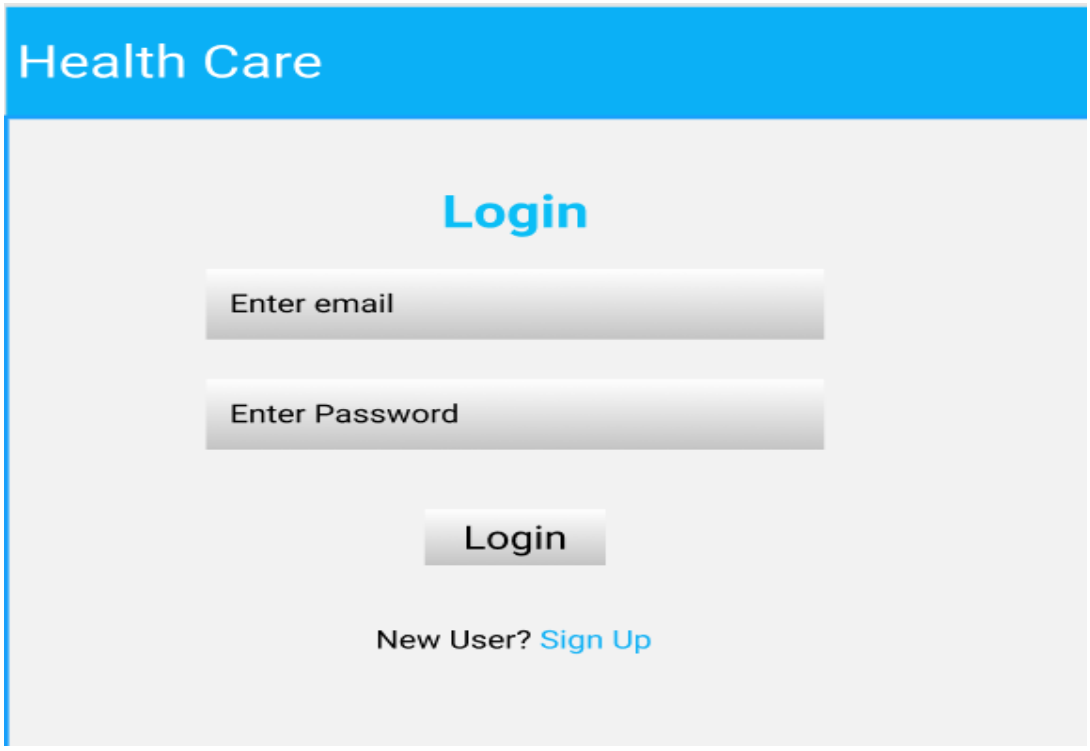
PATIENTS			
Action	URL	Method	Response
Login	/login	POST	true/false
Signup	/signup	POST	true/false
Get All Doctor	/doctor	GET	Array of Doctors
Add Booking	/booking	POST	Booking Created
Remove Booking	/booking/{id}	DELETE	Booking Removed
Get Prescription	/prescription/{id}	GET	Return the prescription based on id
Get Checkup Report	/checkupReport/{id}	GET	Return the resport based on user id
DOCTOR			
Action	URL	Method	Response
Get All Booking	/doctor/booking	GET	Array of Booking
Approve Booking	/ doctor/booking	POST	Booking Appproved
Reject Booking	/ doctor/booking/{id}	DELETE	Booking Deleted
Add Prescription	/doctor/prescription	POST	Prescription Created
Update Prescription	/doctor/prescription/{id}	PUT	Prescription Updated
Delete Prescription	/doctor/prescription/{id}	DELETE	Prescription Deleted
Add Checkup	/doctor/checkup	POST	Checkup Report Added
Update Checkup	/doctor/checkup/{id}	PUT	Checkup Report Updated
Get Checkup	/doctor/checkup/{id}	GET	Return the checkup details
Delete Checkup	/doctor/checkup/{id}	DELETE	Deleted Successfully

Frontend:

Patient:

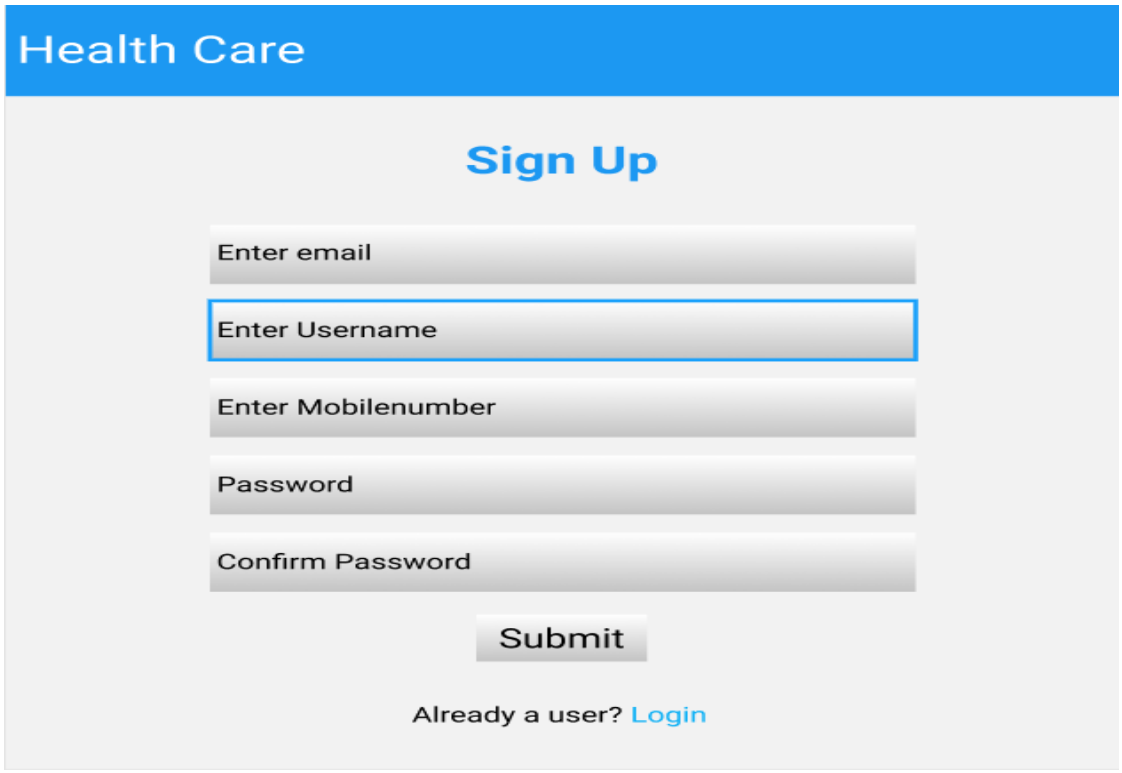
Login:

Output Screenshot:



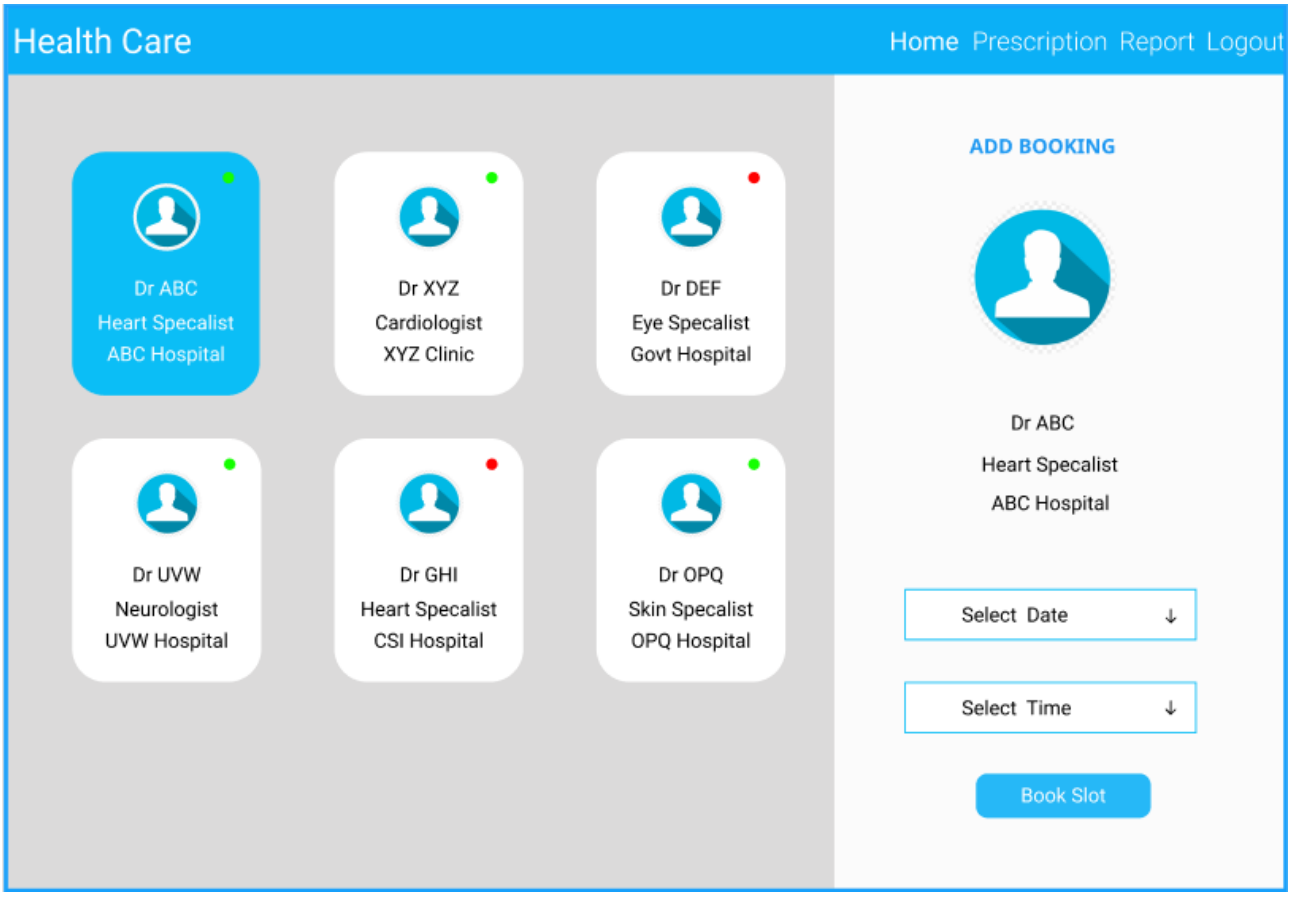
Signup:

Output Screenshot:



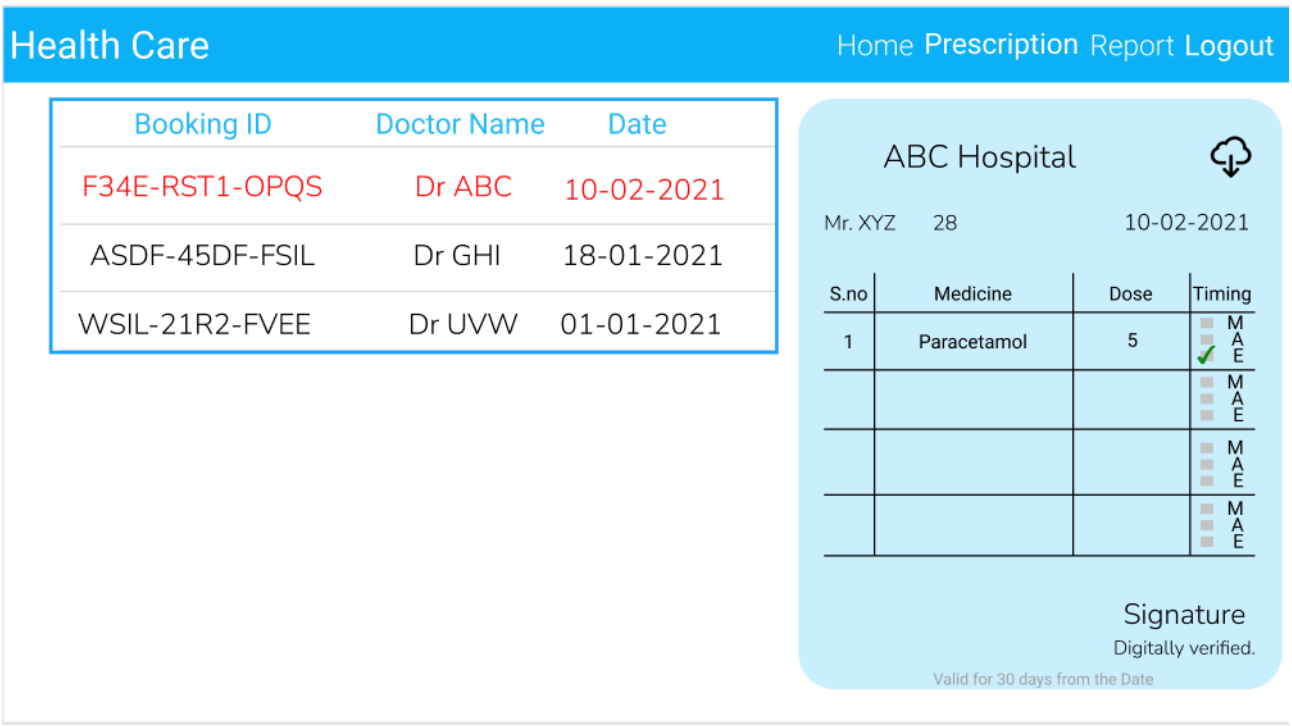
Home:

Output Screenshot:



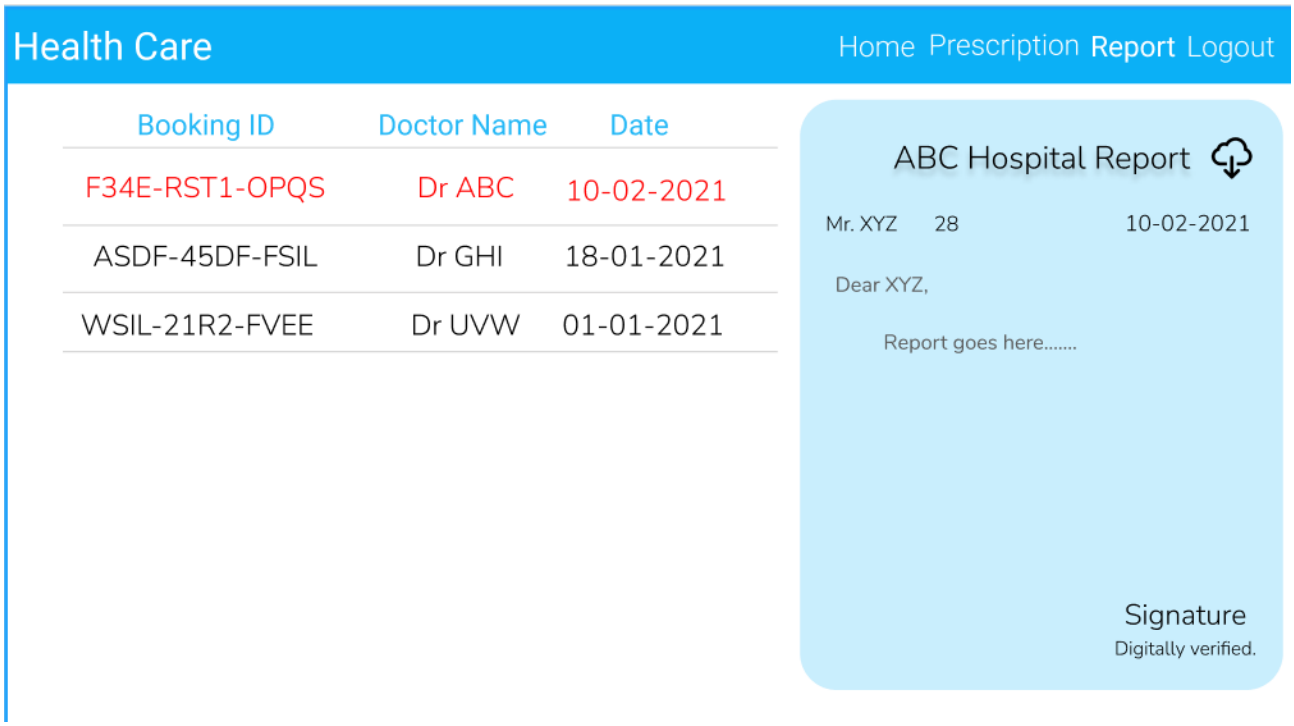
Prescription:

Output Screenshot:



Report:

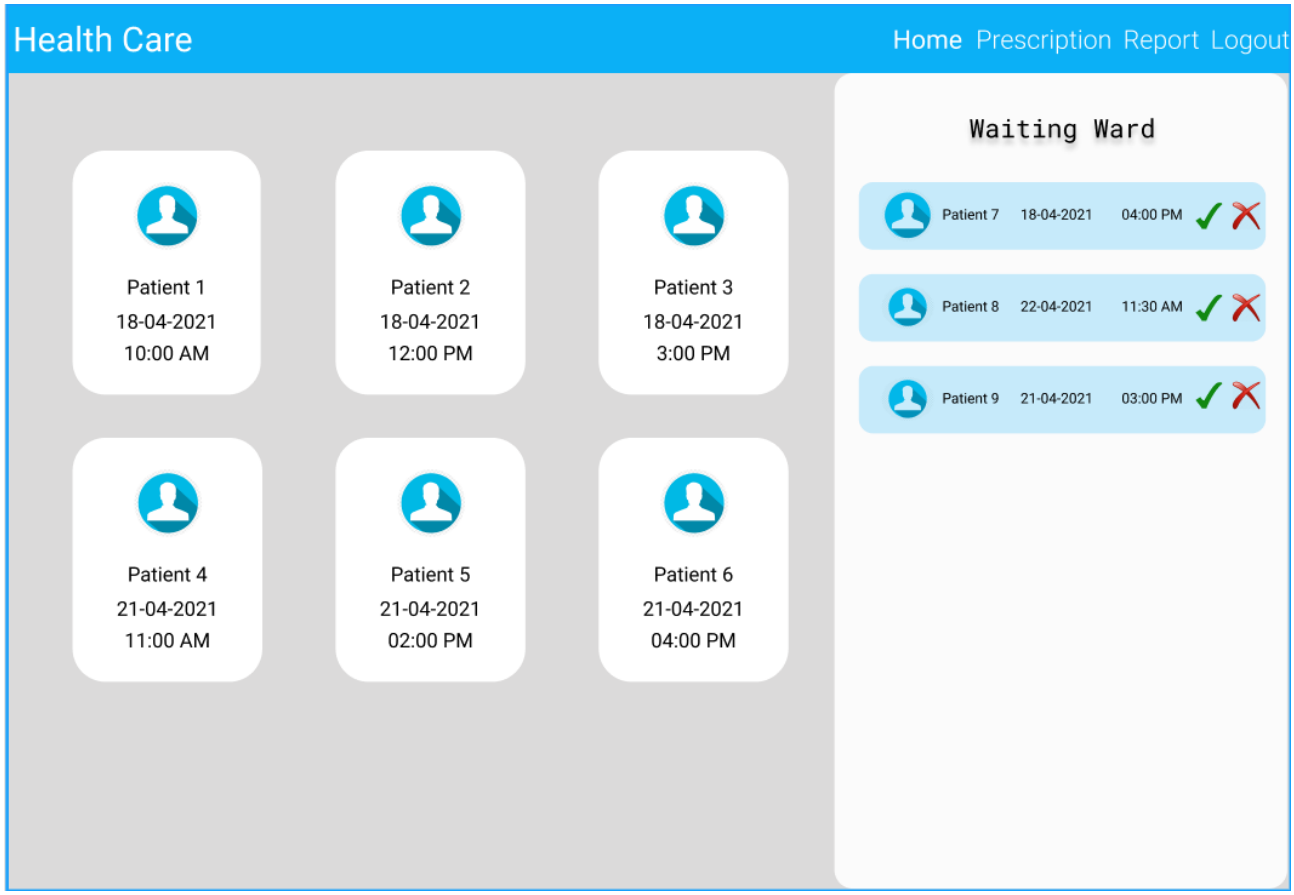
Output Screenshot:



Doctor:

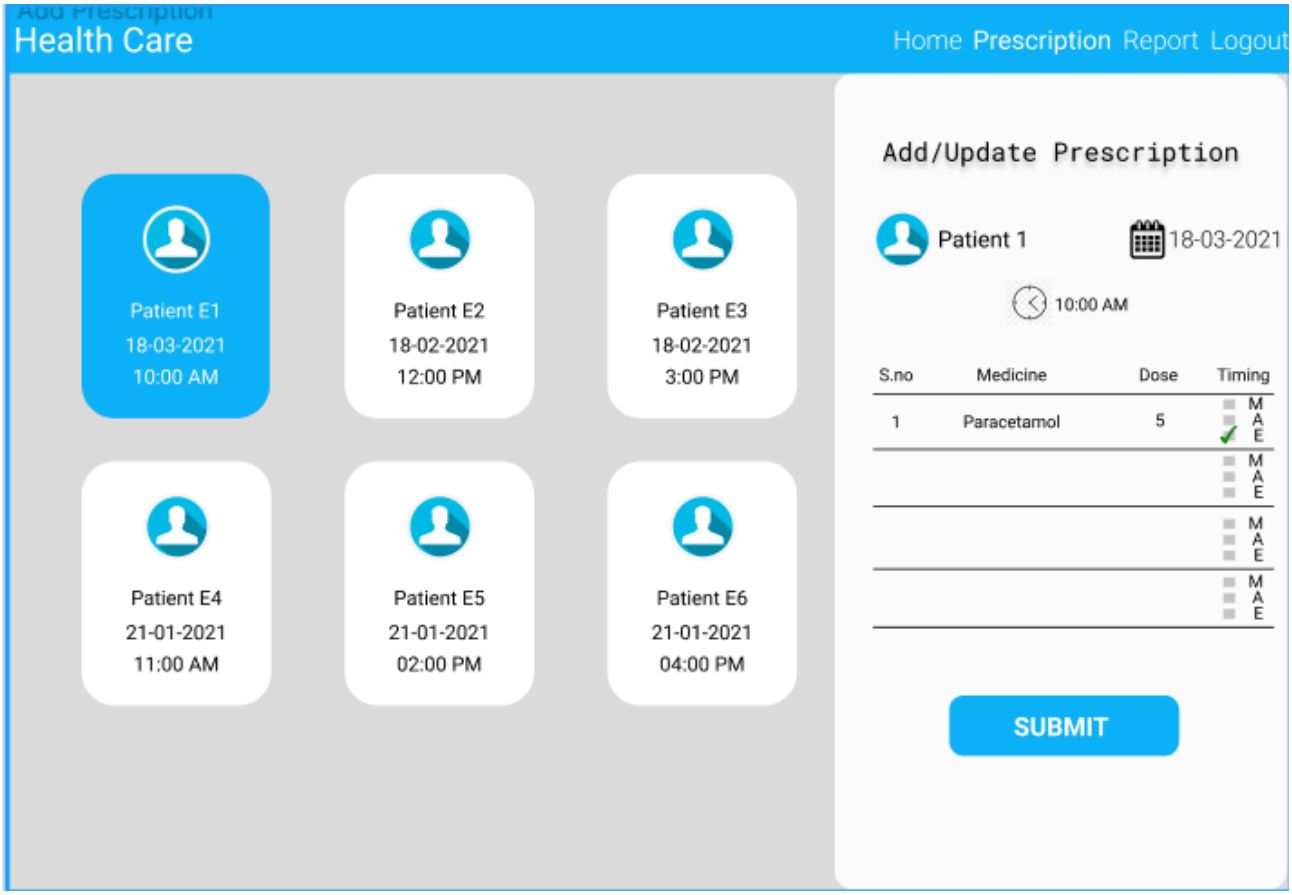
Home:

Output Screenshot:



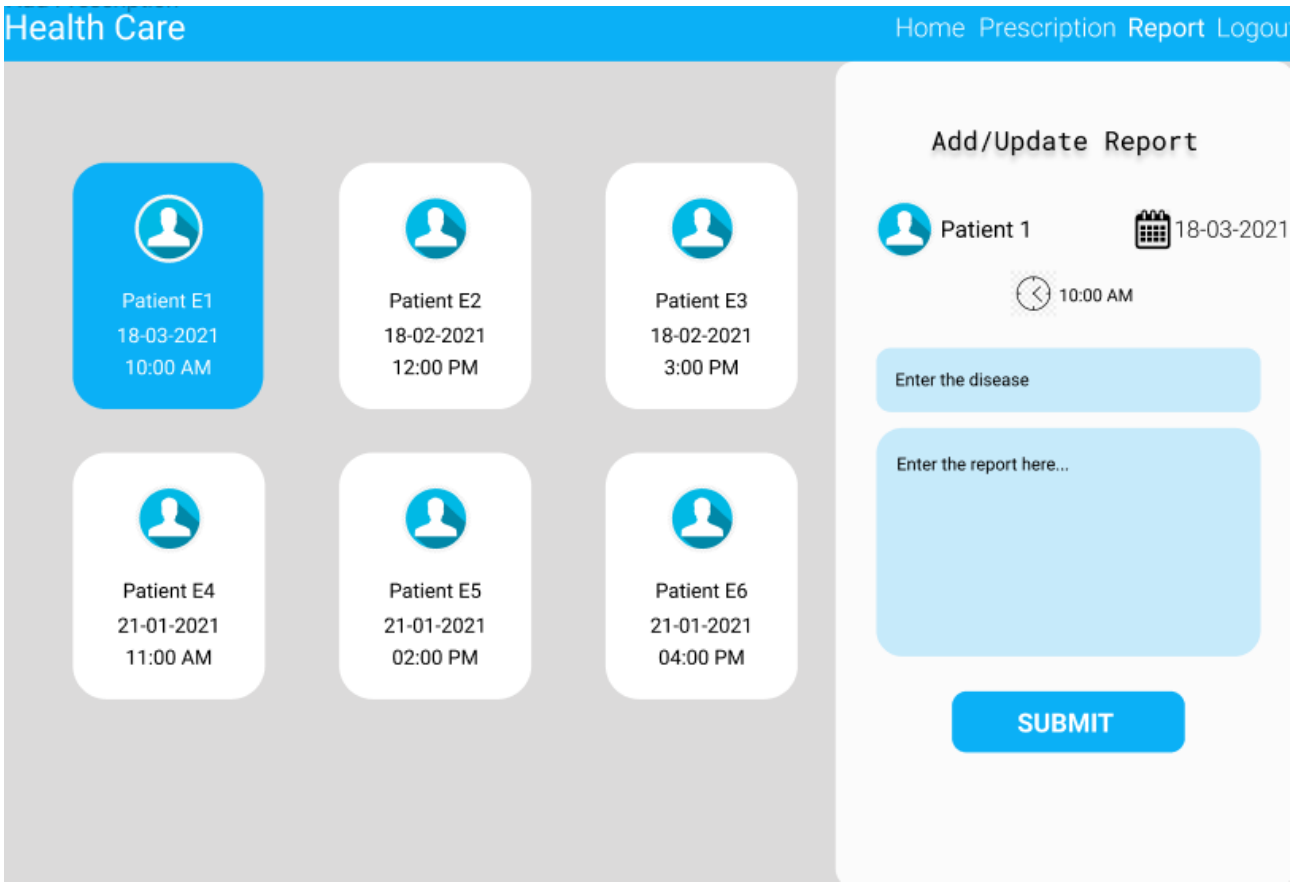
Prescription:

Output Screenshot:



Report:

Output Screenshot:



Backend:

Class and Method description:

Model Layer:

1. UserModel: This class stores the user type (admin or the customer) and all user information.
    a. Attributes:
        i. email: String
        ii. password: String
        iii. username: String
        iv. mobileNumber: String
        v. active: Boolean
        vi. role: String
    b. Methods: -
2. LoginModel: This class contains the email and password of the user.
    a. Attributes:

        i. email: String
        ii. password: String
    b. Methods: -
3. BookingModel: This class stores the appointment details.
    a. Attributes:
        i. bookingId: String
        ii. patientDetail: UserModel
        iii. doctorDetail: DoctorModel
        iv. hospitalName: String
        v. date: Date
        vi. time: Date
        vii. bookingStatus: Boolean
    b. Methods: -
4. PrescriptionModel: This class stores the prescription details for the users.
    a. Attributes:
        i. prescriptionID: String
        ii. userId: UserModel
        iii. date: Date
        iv. description: List <String>
        v. issuedBy: UserModel
    b. Methods: -
5. CheckupModel: This class stores the.
    a. Attributes:
        i. checkupID: String
        ii. appointmentDetail: BookingModel
        iii. prescriptionDetail: PrescriptionModel
        iv. date: Date
        v. diseases: String
        vi. report: String
        vii. issuedBy: UserModel
    b. Methods: -

Controller Layer:

6. SignupController: This class control the user signup

a. Attributes: -

b. Methods:

i. saveUser(UserModel user): This method helps to store users in the database and return true or false based on the database transaction.

7. LoginController: This class controls the user login.

a. Attributes: -

b. Methods:

i. checkUser(LoginModel data): This method helps the user to sign up for the application and must return true or false

8. BookingController: This class controls the adding, upding, removing the booking details.

a. Attributes: -

b. Methods:

i. List<BookingModel> getBooking(): This method helps the admin to fetch all Booking from the database.

ii. List< BookingModel > getBookingByDoctor(): This method helps the doctor to retrieve their all the booking from the database.

iii. BookingModel bookingById(String id): This method helps to retrieve a booking from the database based on the bookingId.

iv. statusModifier(BookingModel data): This method helps the doctor to edit a booking and save the status as Aprrove or Reject.

v. addBooking(BookingModel data): This method helps the patient to add a new booking to the database.

vi. removeBooking(String id): This method helps the doctor to delete a booking from the database.

9. PrescriptionController: This class helps in adding the prescription, deleting the prescription from the cart, updating the prescription.

a. Attributes: -

b. Methods:

i. addPrescription(PrescriptionModel data): This method helps the doctor to add the Prescription to the user.

ii. updatePrescription(PrescriptionModel data): This method helps to update the prescription.

iii. deletePrescription(String id): This method helps the Pharmacists to delete a prescription from the user.

iv. viewPrescription(String id): This method helps the Pharmacists to view the Prescription.

10. CheckupController: This class helps with the doctor to create/read/update the details about the patients.
    a. Attributes: -
    b. Methods:
        i. List<CheckupModel> getCheckupDetails(String id): This method helps to list the details based on the userI id.
        ii. addCheckup(CheckupModel data): This method helps to save the checkup details in the database.
        iii. updateCheckup(CheckupModel data): This method helps to update the checkup details and store it in the database.