Project: E-Medicare

FUNCTIONAL SPECIFICATION

Project Code:	
Project Name:	E-Medicare

FUNCTIONAL SPECIFICATION

Table of Contents

FUNCTIONAL SPECIFICATION

1 Introduction

Mphasis Ltd is a company which builds a software system which is responsible for adding and processing of a product.

Mphasis Ltd plans to develop "E-Medicare" - web application [J2EE Batches - Web Application], where users can register, login, purchase various products medicines.

Scope and Overview:

The scope of the “E-Medicare” will be to provide the functionality as described below. The system will be developed on a Windows operating system using Java/J2EE, Mysql, Spring,Postman,Tomcat.

2 System Overview

The “E-Medicare” should support basic functionalities (explained in section 2.1) for all below listed users.

- Administrator (A)
- Customer (C)

2.1 Authentication & Authorization

2.1.1 Authentication:

Any end-user should be authenticated using a unique userid and password.

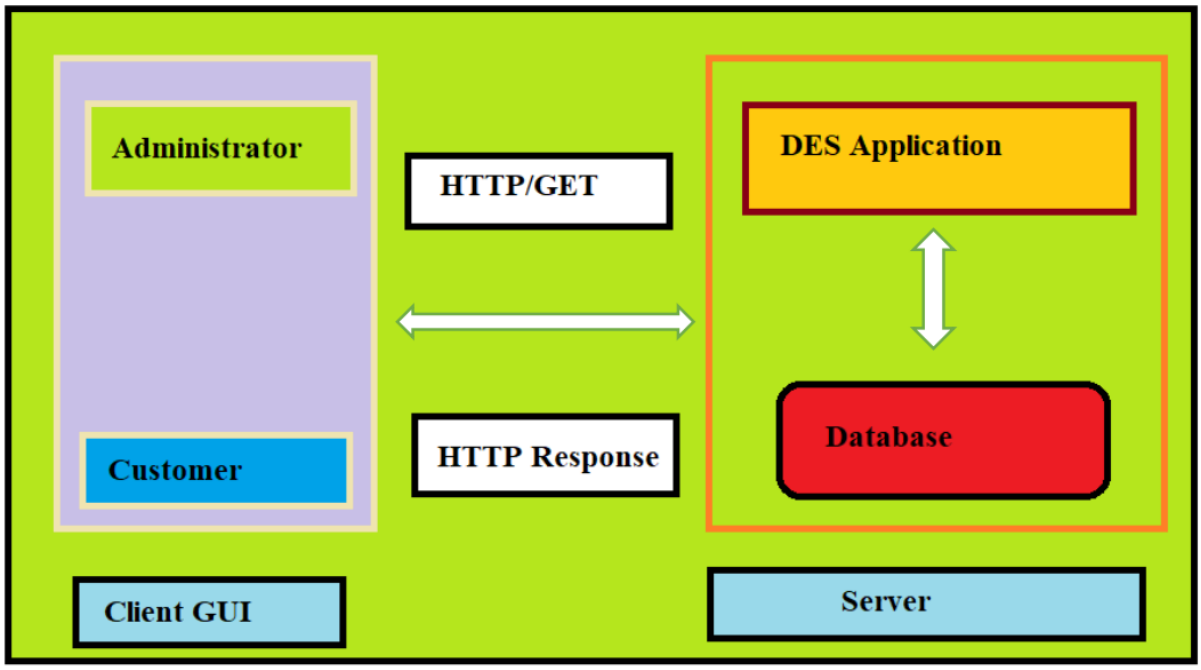
2.1.2 Authorization

The operations supported and allowed would be based on the user type. For example, Administrator has the rights to add product information and view customer details. He can also view order details and purchase details of a medicine.

Whereas Customer/Buyer has a right to Add, Remove and Clear all the medicines from cart.

2.2 Functional Flow

The functional flow of the messages across different application components is shown below.
Ex. - Web Application.



2.3 Environment:

The system will be developed on any Windows OS machine using J2EE, Mysql,Spring.

- Intel hardware machine (PC P4-2.26 GHz, 512 MB RAM, 40 GB HDD)
- Server – Apache Tomcat 8 or higher
- Database – Mysql
- JRE 11
- Eclipse IDE or Spring Tool Suite

FUNCTIONAL SPECIFICATION

3 Sub-system Details

The E-medicare is defined, wherein all users need to login successfully before performing any of their respective operations.

Find below (section 3.1 & 3.2) tables that provides functionality descriptions for each type of user / sub-system. Against each requirement, indicative data is listed in column ‘Data to include’. Further, suggested to add/modify more details wherever required with an approval from customer/faculty.

3.1 Administrator

The administrator as a user is defined to perform below listed operations after successful login.

ID	Objects	Operations	Data to include	Remarks
01 To 04	Medicine	Add View Delete Modify	Medicine Id, Medicine Name,Medicine Company, Medicine Price,picbyte.	
05 To 10	User	View	User Id,User Name,Password,Type	
11 To 13	Order and purchase	View	Userid,Medicine Id,Medicine price,quantity,Status	

3.2 Customer

The customer as a user is defined to perform below listed operations after successful login.

ID	Objects	Operations	Data to include	Remarks
US-001	User_Login	Register	UserId, Username, Password, Email, Phone Number,Address.	
US-002	Medicine	Add to Cart. Delete from Cart. Delete all products from cart.	Medicine Id, Medicine Name, Price, Quantity, Status	
US-003	Checkout	Add User Details and Price	CartId, UserId and Total Price	

FUNCTIONAL SPECIFICATION

3.3 Login | Logout

[Web Application - J2EE, Mysql, Spring]

- Go to Registration screen when you click on Register link.
- Go to Success screen when you login successfully after entering valid username & password fetched from the database.
- Redirect back to same login screen if username & password are not matching.
- Implement Session tracking for all logged in users before allowing access to application features. Anonymous users should be checked, unless explicitly mentioned.

4 Data Organization

This section explains the data storage requirements of the E-Medicare and **indicative** data description along with suggested table (database) structure. The following section explains few of the tables (fields) with description. However, in similar approach need to be considered for all other tables.

4.1 Table: User_Registration_Details

The user specific details such as username, email, phone etc. Authentication, and authorization / privileges should be kept in one or more tables, as necessary and applicable.

Field Name	Description
UserID	UserID is auto generated after registration and it is used as LoginID.
Username	Username of the Customer
Password	User Password
Email	Customer Email Id
Phone Number	10-digit contact number of users
Adress	Adress

4.2 Table: Medicine_Details

This table contains information related to a product.

Field Name	Description
Medicine Id	Unique medicine Id, Here medicine Id will be Primary Key
Medicine Name	Name of the Medicine e.g., Paracetmol etc.
Medicine Price	Price of the Medicine
Picbyte	Image of the Medicine
Company Name	Name of the Company

4.3 Table: Cart_Details

This table contains information related to cart details.

Field Name	Description
UserId	UserId corresponding to logged in user
Medicine Price	Price of the Medicine
Total Price	Total price of the purchased medicines
Status	Medicine Availability Status, Example: Pending or Placed.

4.4 Table: Checkout_Details

This table contains information related to final checkout details.

Field Name	Description
UserId	Registered User Id, this field should be foreign key
Total Price	Total Price of the purchased medicines

FUNCTIONAL SPECIFICATION

5. REST APIs to be Built.

Create following REST resources which are required in the application,

1. Creating **User** Entity: Create Spring Boot with Microservices Application with Spring Data JPA

Technology stack:

- Spring Boot
- Spring REST
- Spring Data JPA

Here will have multiple layers into the application:

1. Create an Entity: User
2. Create a UserRepository interface and will make use of Spring Data JPA
   a. Will have findByUserName method.
   b. Add the User details
3. Create a UserService class and will expose all these services.
4. Finally, create a UserRestController will have the following Uri's:

URI	METHODS	Description	Format
/api/users	GET	Give all the users	JSON
/api/users/{userid }	GET	Give a single user description searched based on user id	String
/api/users	POST	Add the user details	JSON

2. Creating Medicine Entity:

Build a RESTful resource for **Product** manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Product
2. Create a ProductRepository interface and will make use of Spring Data JPA
   a. Will have findByProductName method.
   b. Add the Product details method.
   c. Will have deleteProductById method.
   d. Will have findAllProducts method.
3. Create a ProductService class and will expose all these services.
4. Finally, create a ProductRestController will have the following Uri's:

URI	METHODS	Description	Format
/api/medicines	GET	Get all the medicines	JSON
/api/medicine	GET	Add a single medicine	JSON
/api/medicine	POST	Update medicine	JSON
/api/{id}	DELETE	Delete a medicine based on product id	JSON

3. Creating **Adminlogin** Entity:

Build a RESTful resource for **Adminlogin** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Adminlogin
2. Create a adminlogin tRepository interface and will make use of Spring Data JPA
   a. Add the adminlogin details
3. Create a Adminlogin Service class and will expose all these services.
4. Finally, create a Adminlogin RestController will have the following Uri's:

URI	METHODS	Description	Format
/api/admin	POST	Add a single admin	JSON
/api/admins	GET	Get all the admins	JSON
/api/admin/{id}	GET	Get a single admin description searched based on admin id	JSON

4. Creating **Checkout** Entity:

Build a RESTful resource for **Checkout** manipulations, where following operations to be carried out. Here will have multiple layers into the application:

5. Create an Entity: Checkout
6. Create a CheckoutRepository interface and will make use of Spring Data JPA
   a. Add the checkout details.
7. Create a CheckoutService class and will expose all these services.
8. Finally, create a CheckoutRestController will have the following Uri's:

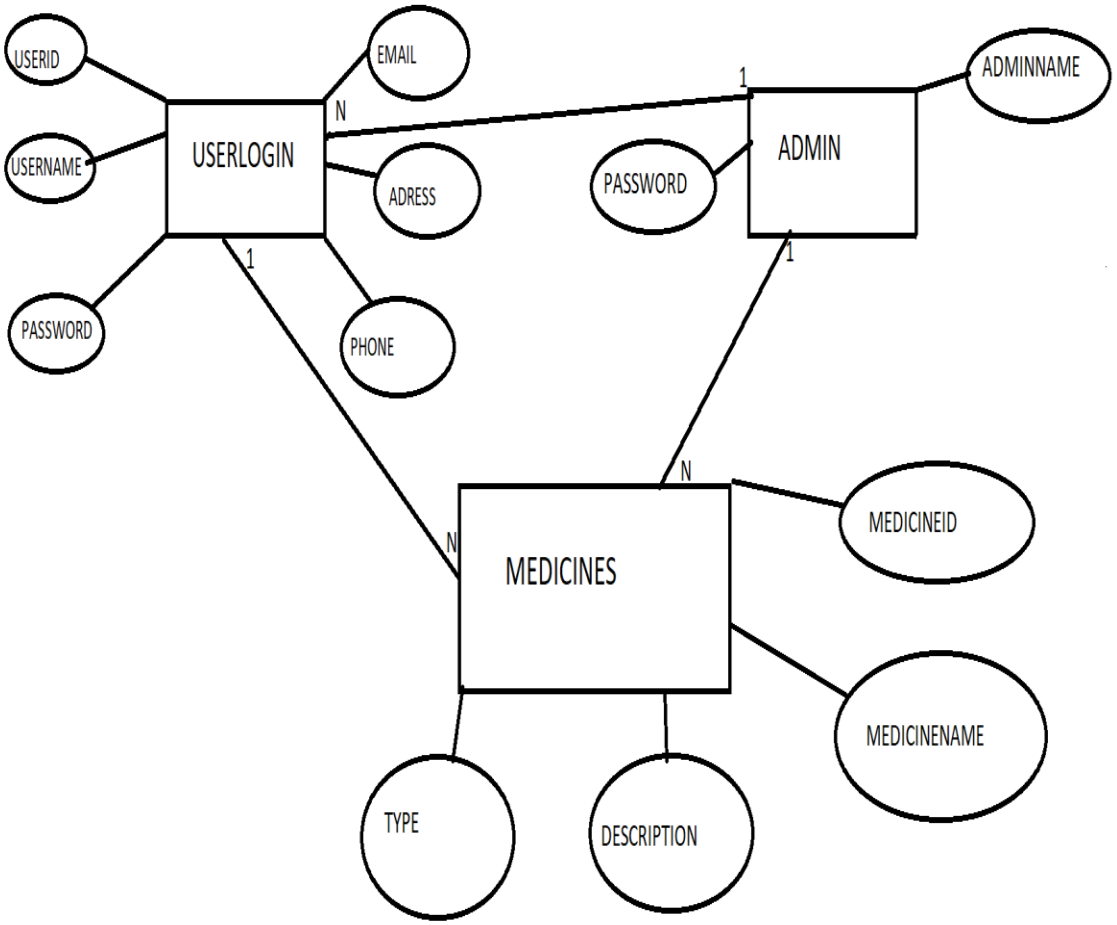
URI	METHODS	Description	Format
/checkout	POST	Add the user details with total price.	JSON

FUNCTIONAL SPECIFICATION

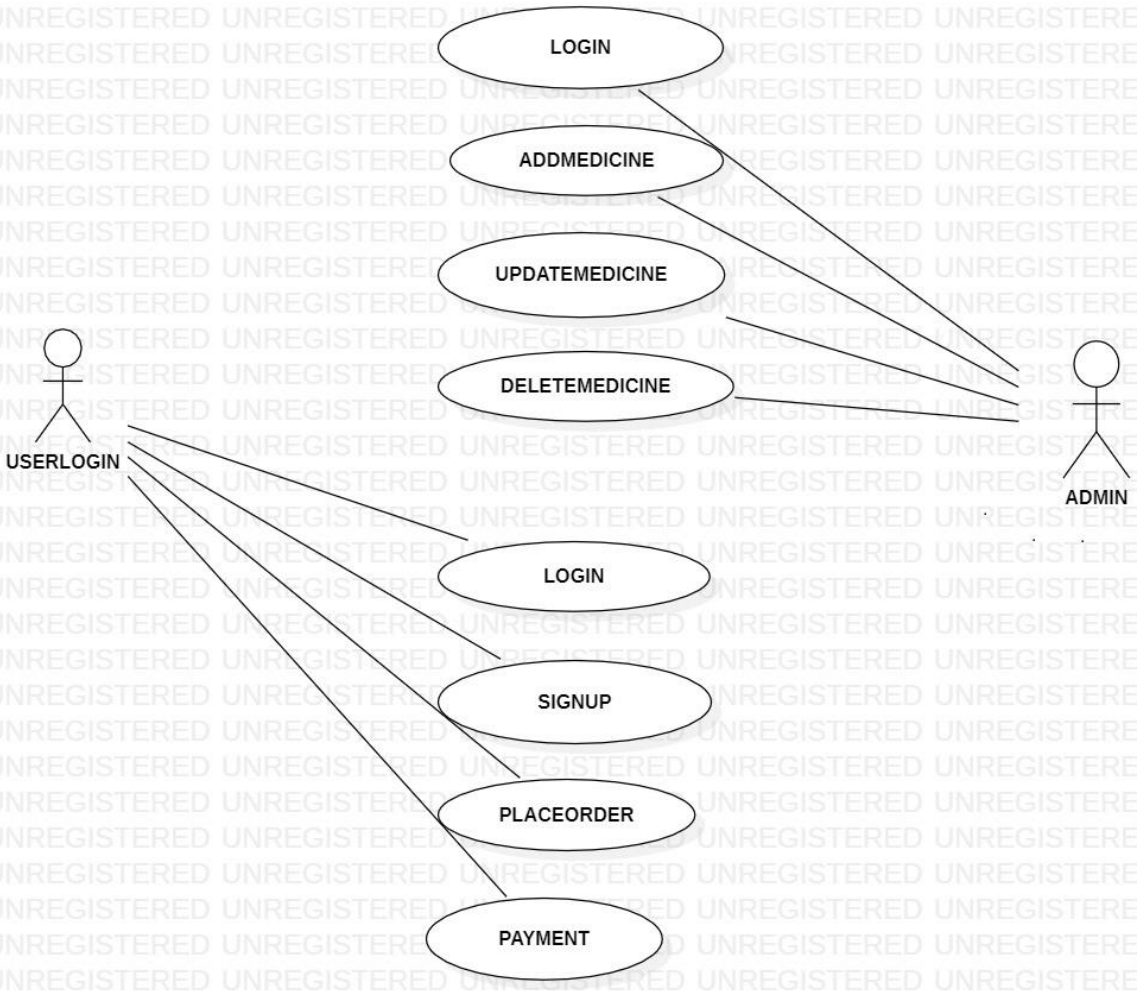
6. Assumptions

- User Interface: The type of client interface (front-end) to be supported - Angular based
- The administrator can add and remove medicines into the database on a weekly basis.
- You must not allow user to add same medicine twice.
- When you add medicines into cart the No. Of medicines selected will be incremented.
- If you remove the medicine from the cart, the counter will be decremented.
- The clear will remove all the medicines so that the No. of medicines will be zero
- The total amount will be calculated based on the medicine, accordingly, change the medicine counter & total amount.

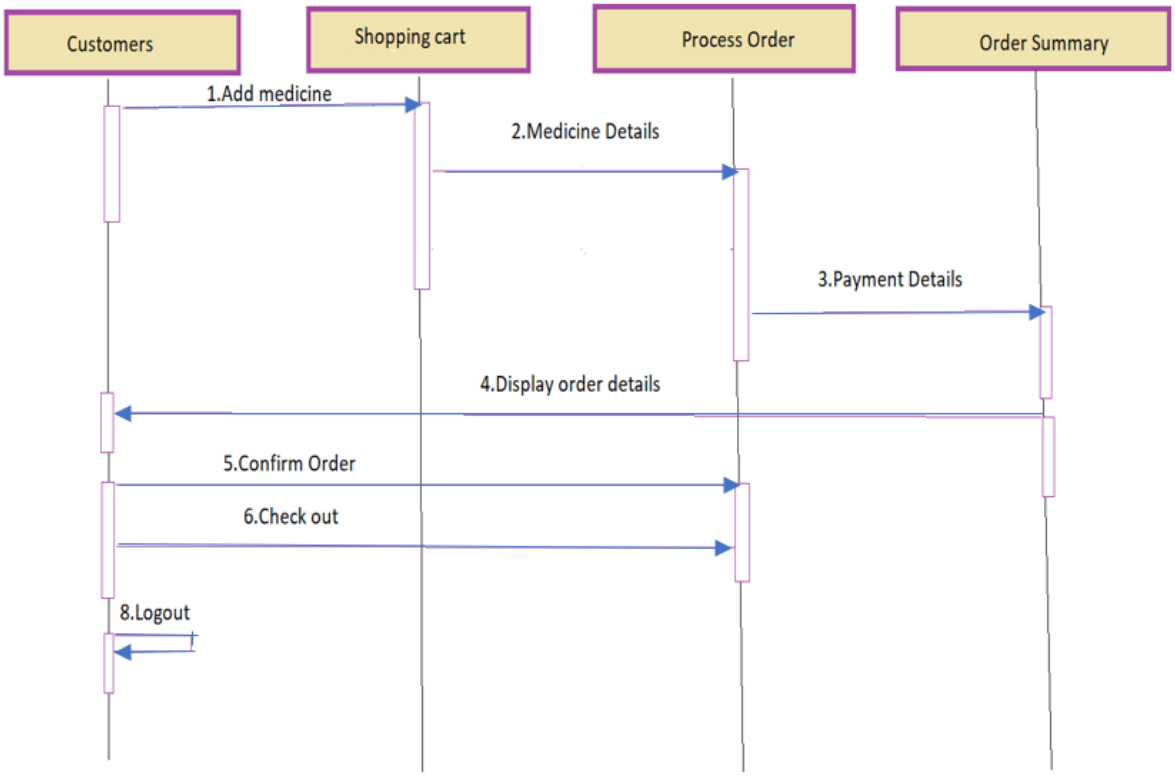
7. ER-Diagram


USERID
EMAIL
USERNAME
USERLOGIN
ADRESS
PASSWORD
PHONE
N
1
1
ADMINNAME
ADMIN
PASSWORD
1
N
N
MEDICINEID
MEDICINES
MEDICINENAME
TYPE
DESCRIPTION

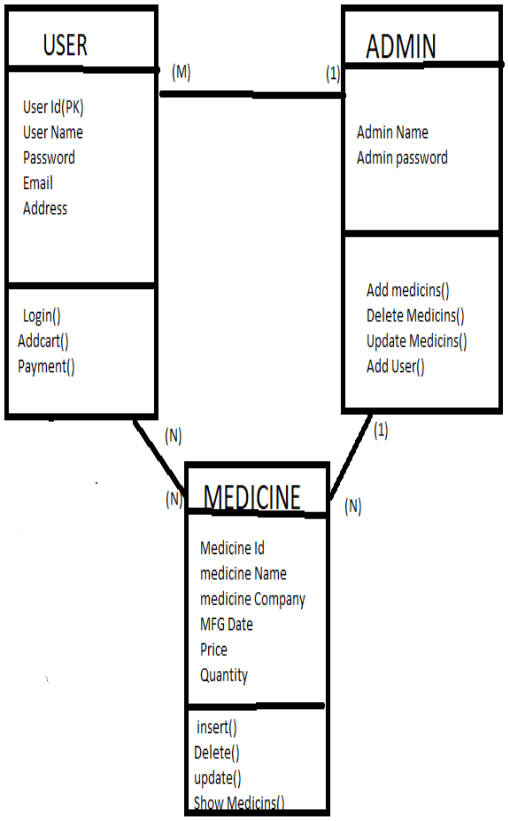
8. UseCase Diagram



9. Sequence Diagram

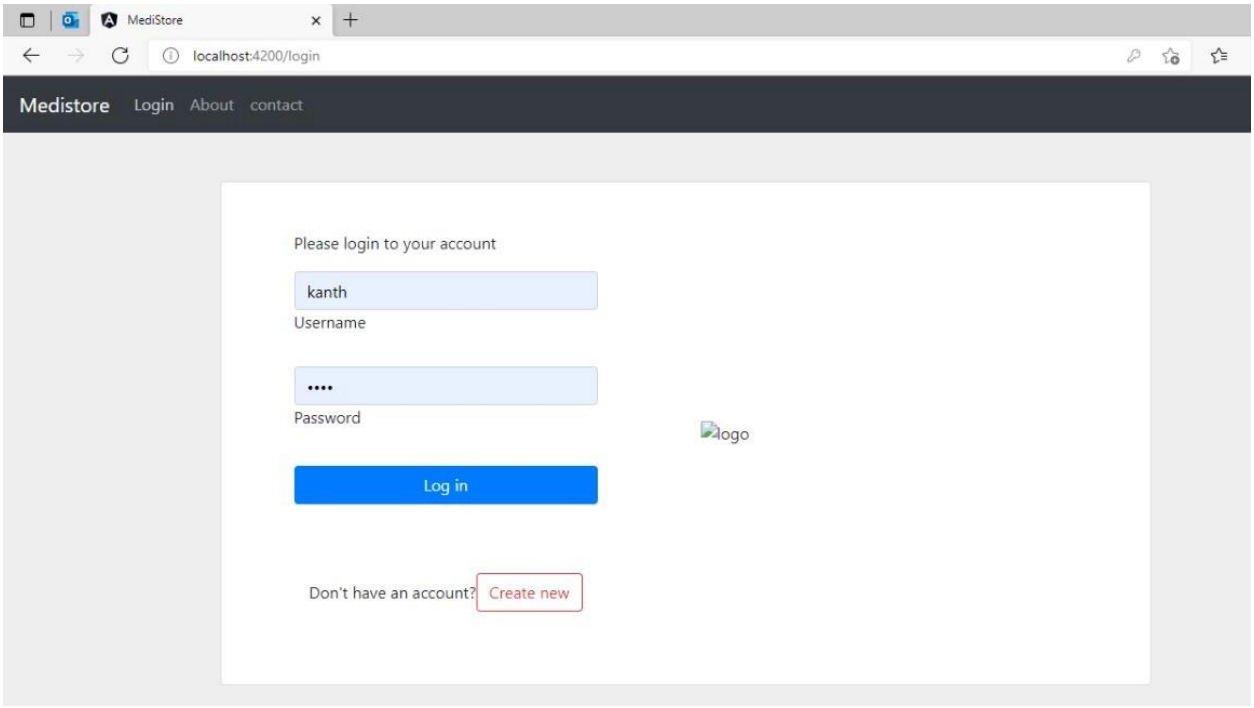


10. DB SCHEMA

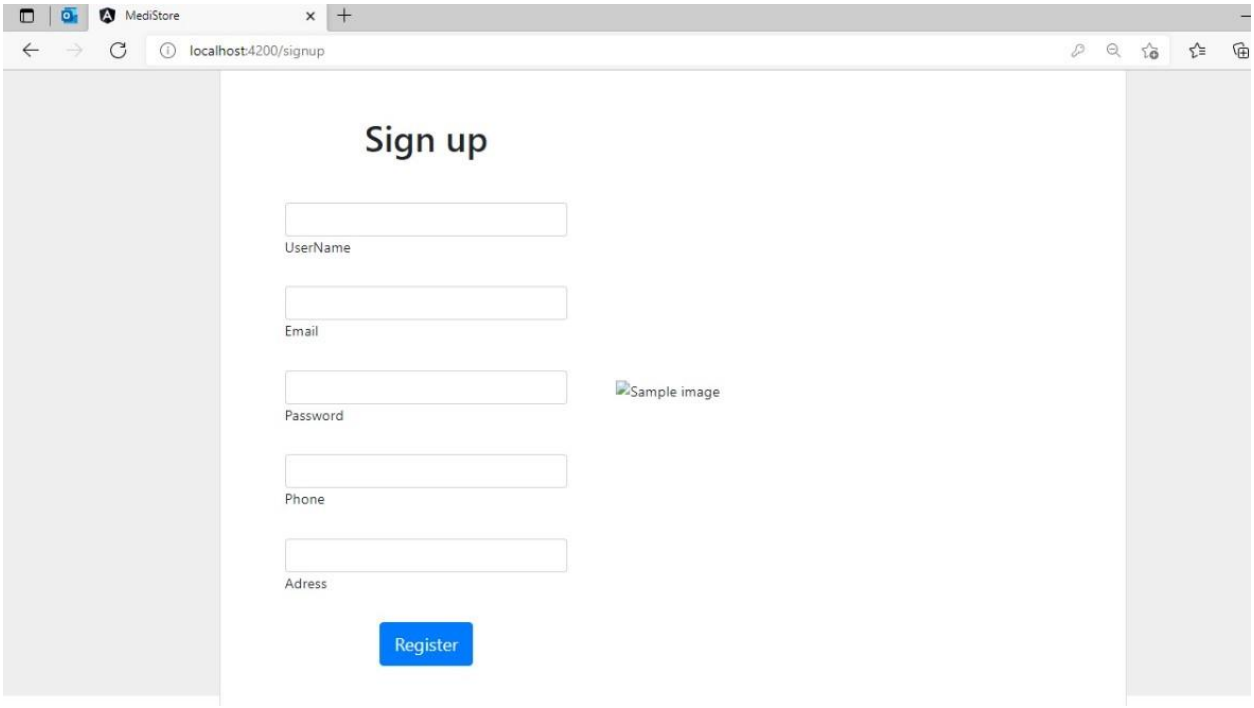

USER
User Id(PK)
User Name
Password
Email
Address
Login()
Addcart()
Payment()
(M)
(1)
ADMIN
Admin Name
Admin password
Add medicins()
Delete Medicins()
Update Medicins()
Add User()
(N)
(1)
(N)
(N)
MEDICINE
Medicine Id
medicine Name
medicine Company
MFG Date
Price
Quantity
insert()
Delete()
update()
Show Medicins()

8 Output Screenshots for your Project

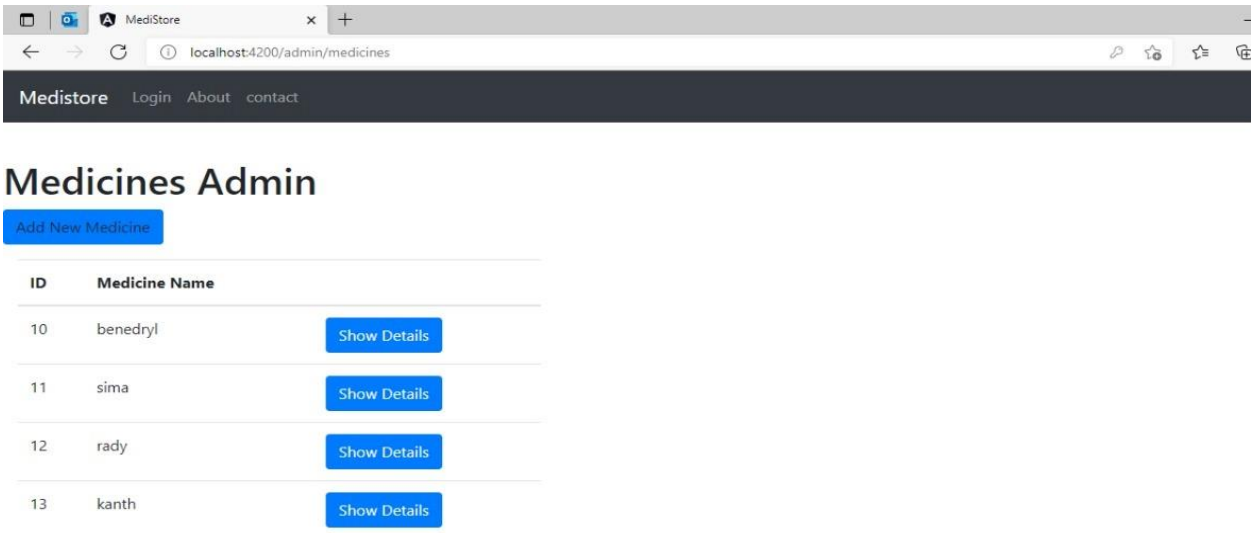
Login page

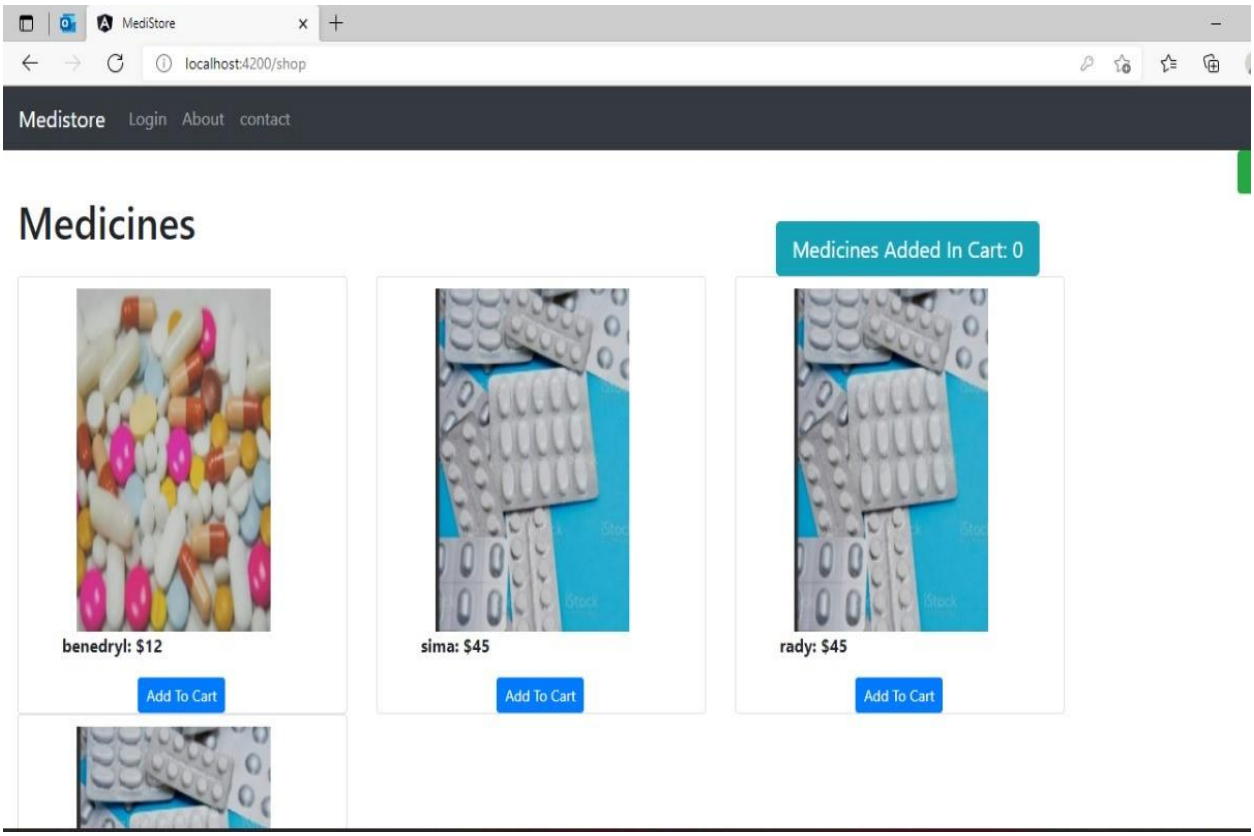


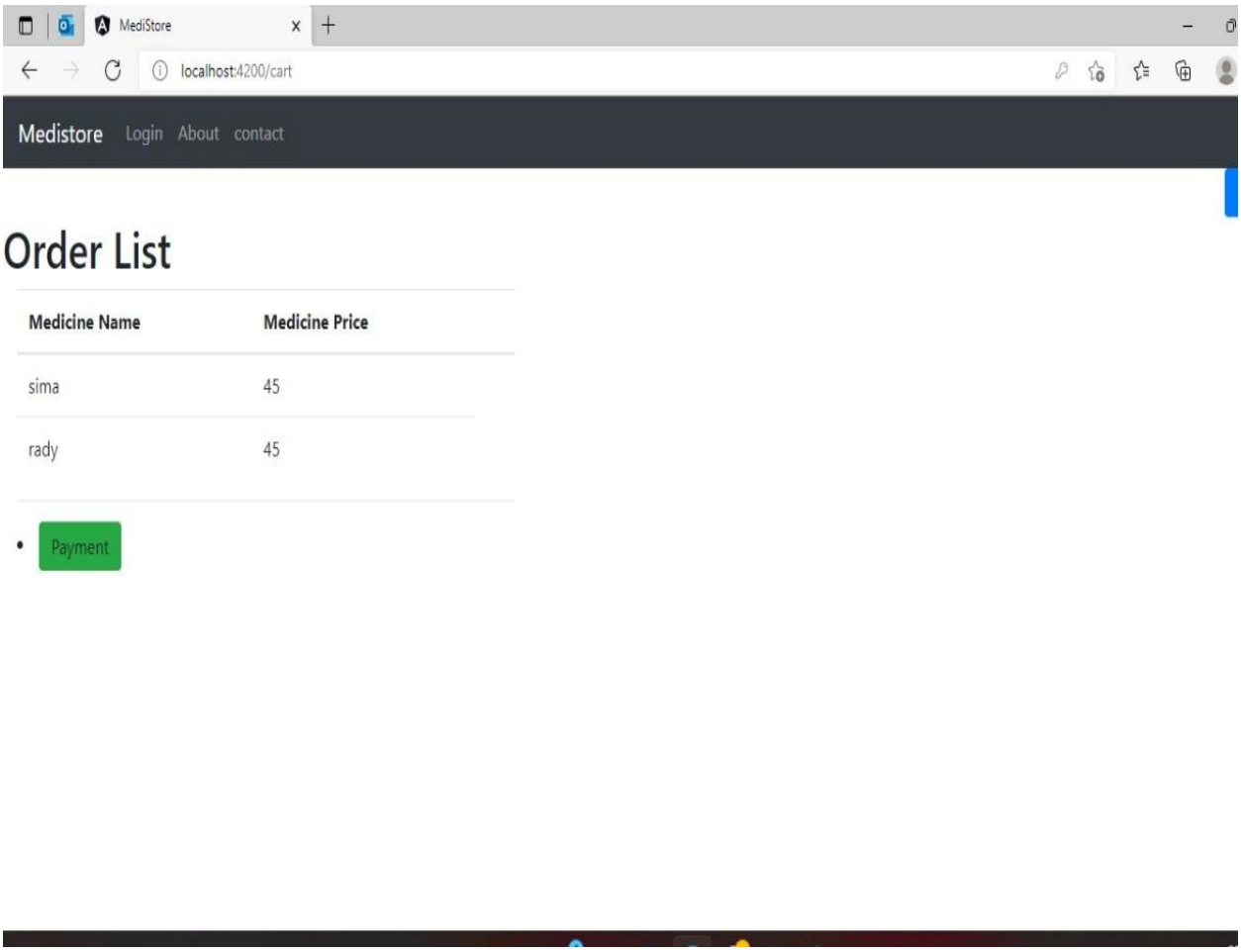
Sign up page



Admin page


MediStore
localhost:4200/admin/medicines
Medistore
Login
About
contact
Medicines Admin
Add New Medicine
ID
Medicine Name
10
benedryl
Show Details
11
sima
Show Details
12
rady
Show Details
13
kanth
Show Details


MediStore
localhost:4200/shop
Medistore
Login
About
contact
Medicines
Medicines Added In Cart: 0
benedryl: $12
Add To Cart
sima: $45
Add To Cart
rady: $45
Add To Cart


MediStore
localhost:4200/cart
Medistore
Login
About
contact
Order List
Medicine Name
Medicine Price
sima
45
rady
45
Payment