



E-Medicare

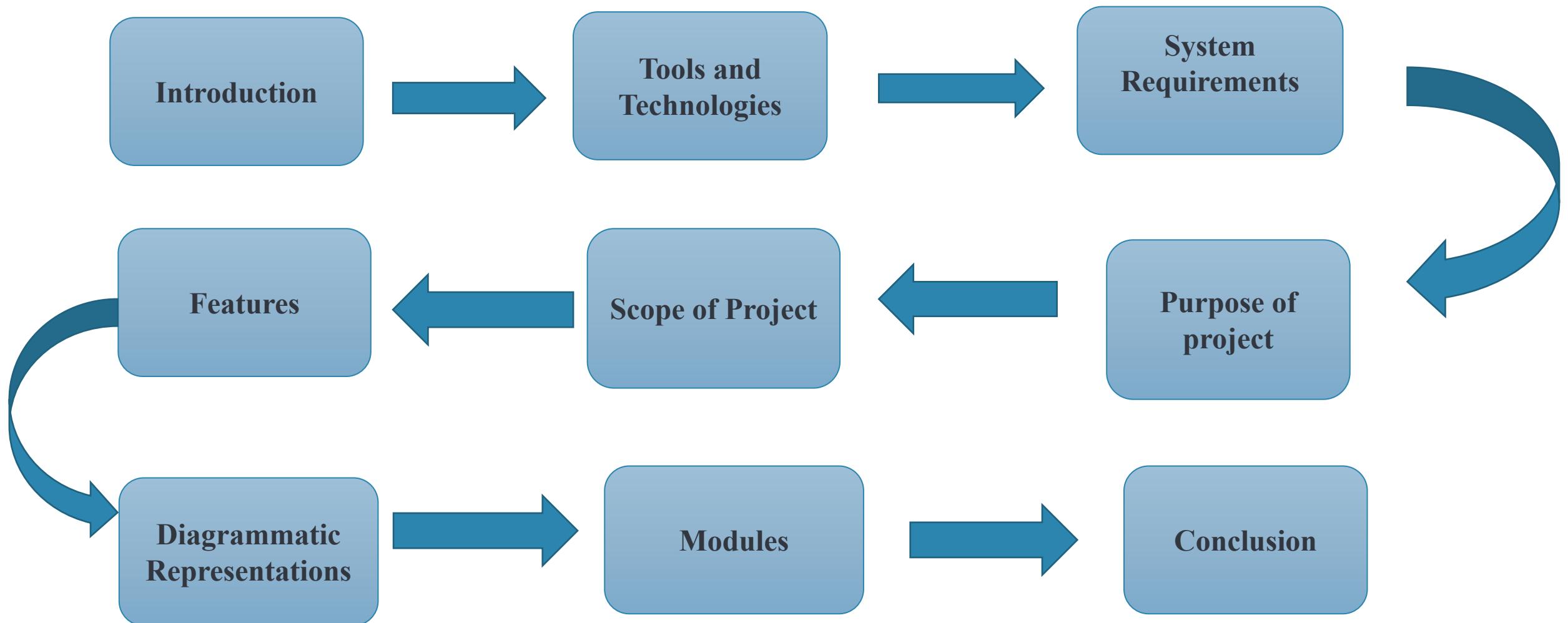
-GROUP 5

Group-5



- Harsh Rathod -2563088
- Swaroop A V -2562956
- Sulagna Chatterjee -2563990
- Priya Singha -2564017
- Sinchana Rao -2564007
- Pavithra S -2564183
- Rajalakshmi H -2564069
- Vallepu Sree Lakshmi Kavya -2563318
- Varshita D C -2563588
- Yendru Pavan Kumar -2564326

Contents



Tools And Technologies

Back End

- ❑ MySQL
- ❑ Spring Boot
- ❑ Postman
- ❑ Rapid API
- ❑ Docker
- ❑ RabbitMq

Cloud

- ❑ Cloud Config
- ❑ Netflix Eureka
- ❑ Open Feign
- ❑ Resilience4j
- ❑ Retry(Circuit Breaker)
- ❑ Zipkin

System Requirements

The Basic System Requirements for Running this project are listed below:

- Visual studio/Eclipse, Postman , Docker and MySQL are to be installed to the system
- 1 GB Random Access Memory
- 200 MB of Free Space on Hard Disk
- Microsoft Windows 8,10,11 or Linux or Equivalent OS
- Web Browser (Microsoft Edge, Google Chrome, Firefox)





Introduction

INTRODUCTION

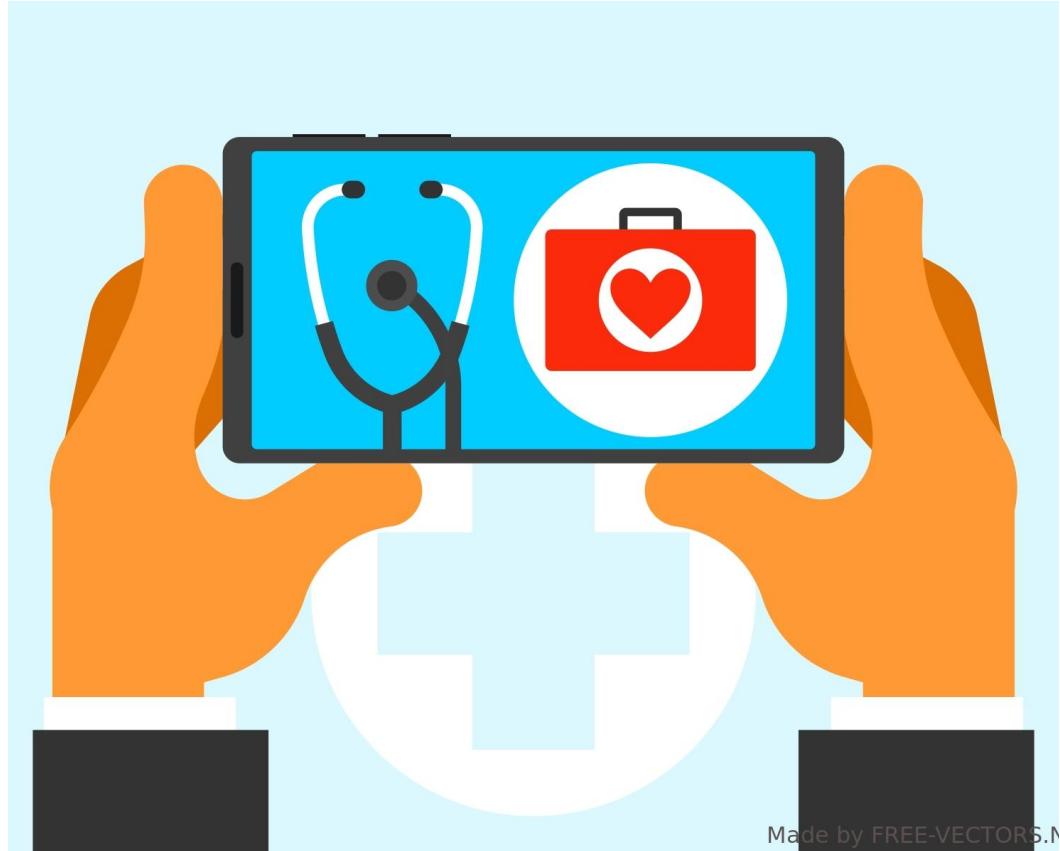
- ❑ E-Medicare is a modular and distributed system built upon the principles of microservices architecture, a software development approach that breaks down complex applications into smaller, interconnected components or services.
- ❑ Each microservice is designed to perform a specific function, such as register service, login service, cart service, payment service and more.
- ❑ The key advantages of Scalability, Interoperability, Data Security, Telemedicine Integration, Cost-Efficiency and more.

PURPOSE:

- E-Medicare microservices break down complex healthcare systems into smaller, specialized components.
- They facilitate seamless integration with existing healthcare systems, ensuring that E-Medicare services can work together with other healthcare applications, databases, and technologies.
- By optimizing resource allocation and reducing operational overhead, E-Medicare microservices can help healthcare organizations reduce costs while maintaining or even improving the quality of care delivered to Medicare beneficiaries.
- E-Medicare microservices generate valuable healthcare data that can be leveraged for data analytics and insights.

Scope

- Online Healthcare product delivery
- Telemedicine Integration
- Billing and Claims Processing
- User Management
- Scalability and Flexibility



Made by FREE-VECTORS.N

SERVICES

- Eureka Server
- Config Server
- Register Service
- Login Service
- Dashboard Service
- Product Service
- Management Service
- Change password Service
- Cart Service
- Address Service
- Payment Service
- Shipping Server



Features Available

Admin

- Register
- Login
- Manage Product
- Manage User Login creds

User

- Register
- Login
- Search,view,
purchaseProduct
- Add Address

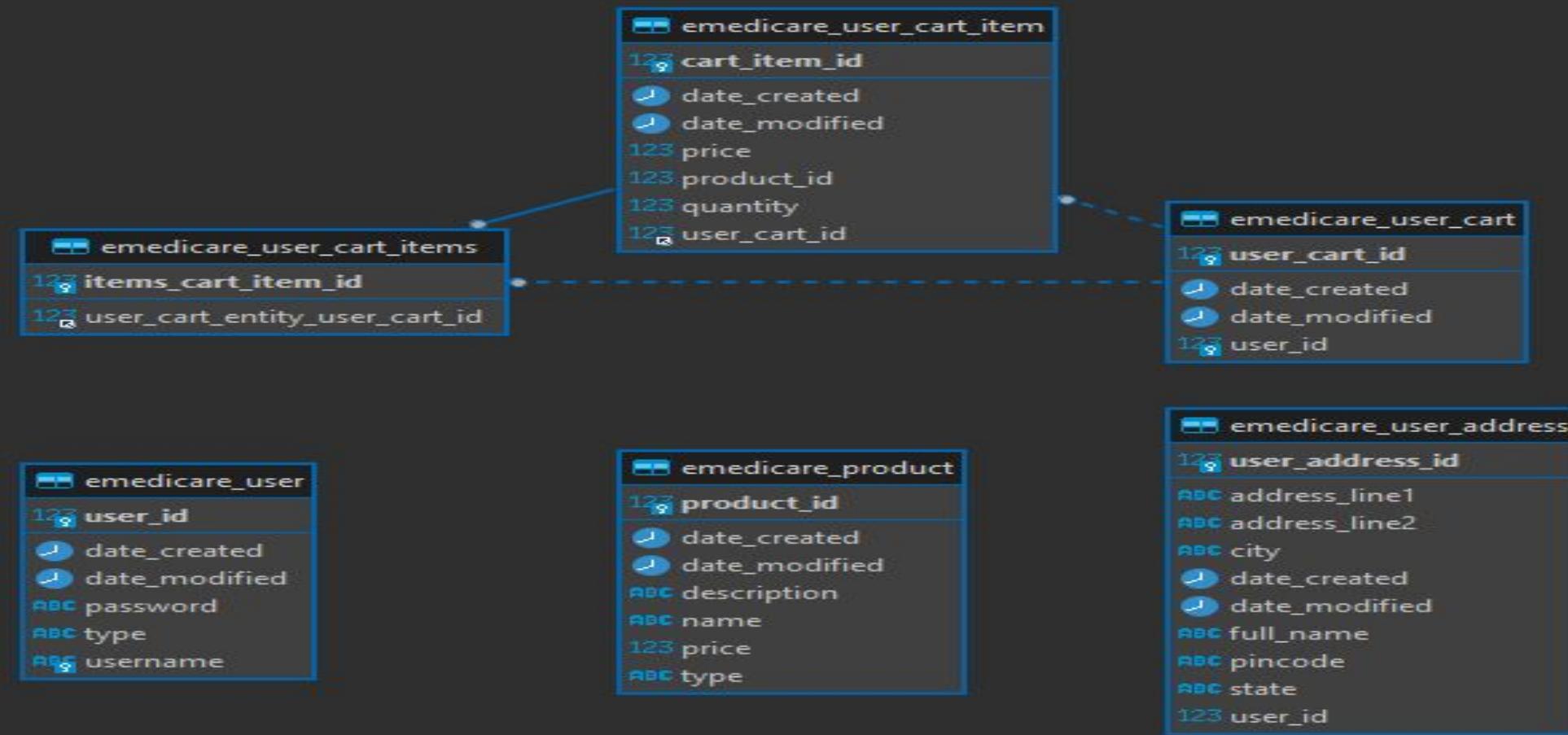
Product

- Add Product
- Update Product
- Delete Product

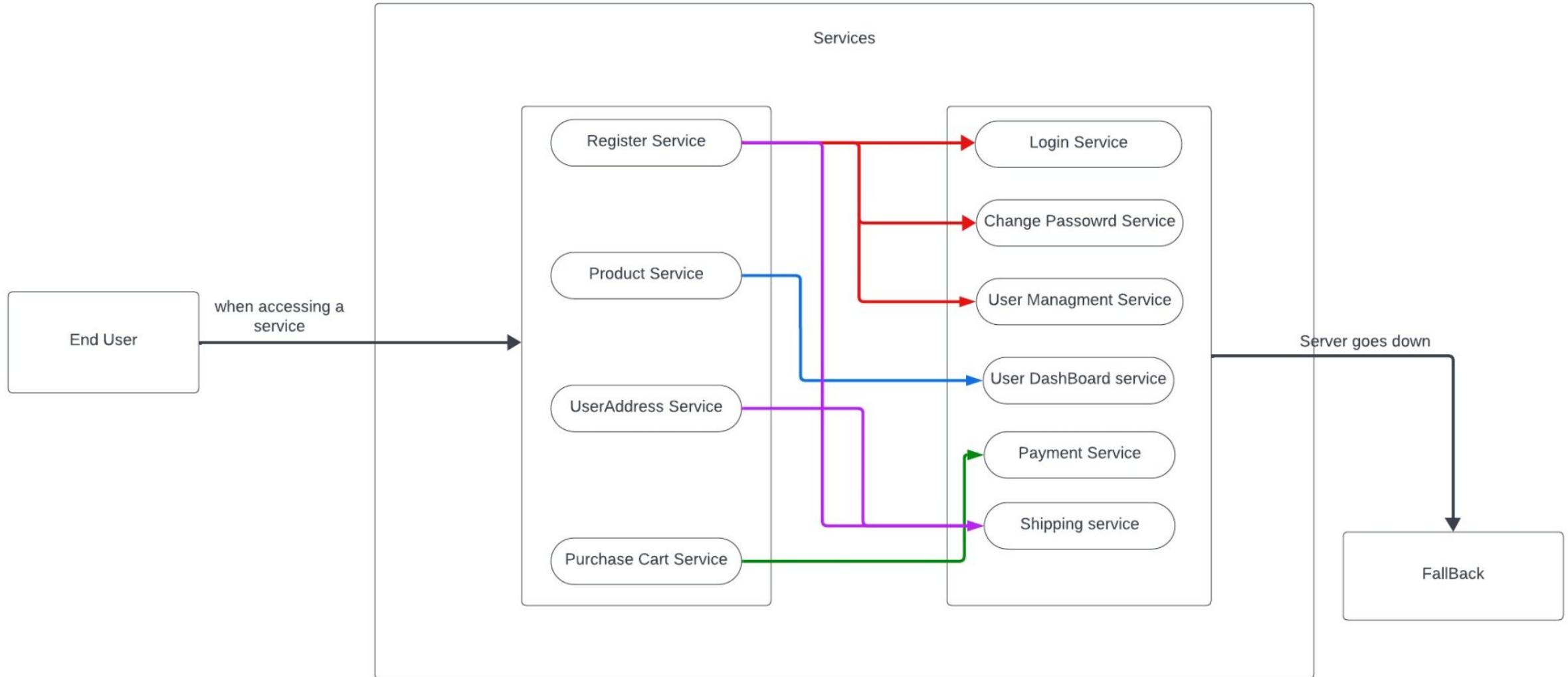
Payment

- Place Order
- Cash on Delivery
- Add Card
- Delete Card
- Payment

Database Schema



FlowChart



Eureka Server

The screenshot shows a Java IDE interface with the following details:

- Code Editor:** The main window displays the `EMedicareEurekaServerApplication.java` file under the package `com.emedicare.eMedicareEurekaServer`. The code implements a Spring Boot application with an Eureka server.
- Terminal Output:** The terminal tab shows the application's startup logs:

```
0718.springCloudBusInput.errors' has 2 subscriber(s).
2023-09-12T11:26:06.742+05:30  INFO 6304 --- [    main] o.s.i.a.i.AmqpInboundChannelAdapter      : started bean 'inbound.
springCloudBus.anonymous.IG_eidiES1Wspt1zgXhi2g'
2023-09-12T11:26:06.816+05:30  INFO 6304 --- [    main] o.s.i.monitor.IntegrationMBeanExporter   : Registering MessageCha
nnel rabbit-1543760718.springCloudBusInput.errors
2023-09-12T11:26:06.879+05:30  INFO 6304 --- [    main] .e.EMedicareCloudConfigServerApplication : Started EMedicareCloud
ConfigServerApplication in 14.54 seconds (process running for 15.566)
2023-09-12T11:29:05.879+05:30  INFO 6304 --- [nio-8888-exec-1] o.s.c.c.s.e.NativeEnvironmentRepository : Adding property source
: Config resource 'file [C:\Users\KIIT\AppData\Local\Temp\config-repo-14762878018193285407\product-service-prod.yaml]' via locat
ion 'file:/C:/Users/KIIT/AppData/Local/Temp/config-repo-14762878018193285407/'
```
- Run Configuration:** A dropdown menu in the top right shows two run configurations: "Run: EMed..." and "Run: EMedi...".
- Bottom Navigation:** The footer includes links for "serviceApplication (rl)", "Git Graph", and "Harsh Rathod, 7 days ago".

Config Server

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows the file name `EMedicareCloudConfigServerApplication.java`, a toolbar with various icons, and the project name `Spring Boot-EMedicare`.
- Code Editor:** Displays the `EMedicareCloudConfigServerApplication.java` file content. The code is annotated with comments from Harsh Rathod, 7 days ago, indicating the implementation of the EMedicare Cloud Config Server.
- Output Terminal:** Shows the application's log output:

```
2023-09-12T11:25:28.959+05:30 INFO 14176 --- [ restartedMain] c.e.e.EMedicareEurekaServerApplication : Started EMedicareEure  
kaServerApplication in 11.05 seconds (process running for 12.422)  
2023-09-12T11:26:28.055+05:30 INFO 14176 --- [a-EvictionTimer] c.n.e.registry.AbstractInstanceRegistry : Running the evict tas  
k with compensationTime 0ms  
2023-09-12T11:26:37.479+05:30 INFO 14176 --- [nio-8761-exec-2] c.n.e.registry.AbstractInstanceRegistry : Registered instance U  
SER-DASHBOARD-SERVICE/host.docker.internal:user-dashboard-service:0 with status UP (replication=false)  
2023-09-12T11:26:38.615+05:30 INFO 14176 --- [nio-8761-exec-3] c.n.e.registry.AbstractInstanceRegistry : Registered instance U  
SER-DASHBOARD-SERVICE/host.docker.internal:user-dashboard-service:0 with status UP (replication=true)  
2023-09-12T11:27:28.054+05:30 INFO 14176 --- [a-EvictionTimer] c.n.e.registry.AbstractInstanceRegistry : Running the evict tas  
k with compensationTime 0ms
```
- Bottom Status Bar:** Shows the author `Harsh Rathod, 7 days ago`, line count `Ln 1, Col 1`, spaces used `Spaces: 2`, encoding `UTF-8`, line endings `CRLF`, and file type `{ } Java`.
- Run Configuration:** A dropdown menu on the right lists two run configurations: "Run: EMedicar..." and "Run: EMedicar...".

Register

sulagna

Logic

Main part:

- 1.Create microService
- 2.Connect with Config and Eureka Server

Secondary part:

1. Feign Client
2. Reselience4j
3. Retry

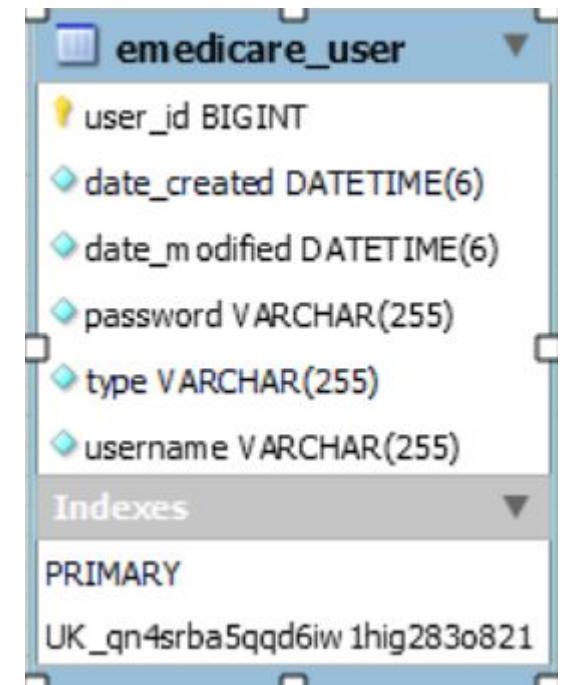
Table

Table: emedicare_user

Columns:

user_id	bigint AI PK
date_created	datetime(6)
date_modified	datetime(6)
password	varchar(255)
type	varchar(255)
username	varchar(255)

ER Diagram



MicroService

The screenshot shows a Java IDE interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, ...
- Search Bar:** rll
- Toolbars:** Standard icons for file operations.
- Left Sidebar (EXPLORER):**
 - OPEN EDITORS: UserController.java
 - RLL
 - configuration
 - controller
 - UserController.java
 - dto
 - entity
 - repository
 - service
 - UserService.java
 - UserRegisterServiceApplication.java
 - resources
 - test
 - target
 - .gitignore
 - .prettierignore
 - .prettierrc.json
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - emedicare-user-register-service-feign
 - emedicare-user-shipping-service
 - Purchase-Cart
 - .gitignore
 - .prettierignore
 - .prettierrc.json
- Central Area (Code Editor):** UserController.java content.
- Right Sidebar:** Language Support for Java(TM) by Red Hat, UserController, various status indicators.
- Bottom Status Bar:** You, 7 days ago | 1 author (You), Ln 80, Col 21, Spaces: 2, UTF-8, CRLF, Java.

```
You, 7 days ago | 1 author (You)
21  @RestController(value = "UserController")
22  @Scope(value = "request")
23  @RequestMapping(value = "/user")
24  public class UserController {
25      @Autowired
26      @Qualifier(value = "UserService")
27      private UserService userService;
28
29      @PostMapping(
29       value = "/register",
30       produces = MediaType.APPLICATION_JSON_VALUE,
31       consumes = MediaType.APPLICATION_JSON_VALUE
32     )
33      public ResponseEntity<UserDTO> register(@RequestBody UserDTO userDTO) {
34          return new ResponseEntity<UserDTO>(
35              this.userService.addOne(userDTO),
36              HttpStatus.CREATED
37          );
38      }
39
40      @PostMapping(
41       value = "/login",
42       consumes = MediaType.APPLICATION_JSON_VALUE,
43       produces = MediaType.APPLICATION_JSON_VALUE
44     )
45      public ResponseEntity<Object> login(@RequestBody UserDTO userDTO) {
46          userDTO = this.userService.login(userDTO);
47          if (userDTO != null) {
48              return new ResponseEntity<Object>(userDTO, HttpStatus.OK);
49          }
50          return new ResponseEntity<Object>(headers: null, HttpStatus.NOT_FOUND);
51      }
52  }
```

Register as Admin

The screenshot shows the Postman application interface. On the left, the History panel displays several previous requests, mostly to 'localhost:52530/user/register'. The main workspace shows a POST request to 'localhost:52530/user/register' with the following details:

- Method:** POST
- URL:** localhost:52530/user/register
- Body (JSON):**

```
1 {  
2   "username": "admin123@gmail.com",  
3   "type": "admin",  
4   "password": "admin"  
5 }
```
- Response Headers:** 201 Created, 583 ms, 247 B
- Response Body (Pretty JSON):**

```
1 {  
2   "userId": 9,  
3   "username": "admin123@gmail.com",  
4   "type": "admin",  
5   "password": "admin"  
6 }
```

The bottom navigation bar includes icons for Cloud, Windows, Search, Microsoft Edge, Google Chrome, Mail, File, Folder, VS Code, Google Sheets, and others.

Register as User

The screenshot shows the Postman application interface. The top navigation bar includes Home, Workspaces, Explore, a search bar, and account options (Sign In, Create Account). The left sidebar displays a history of API requests, including various PUT and POST operations on localhost ports 57743, 8082, and 51854. The main workspace shows a POST request to `localhost:51854/user/register`. The request details tab shows the method as POST, the URL as `localhost:51854/user/register`, and the Body tab selected with JSON type. The raw JSON payload is:

```
1 {  
2   "username": "user345@gmail.com",  
3   "type": "user",  
4   "password": "user12"  
5 }
```

The response tab shows a successful `201 Created` status with a response time of 1996 ms and a response size of 247 B. The response body is identical to the raw payload. The bottom taskbar shows system icons for weather, battery, and connectivity, along with the date and time (12-09-2023, 11:04).

Fallback

The screenshot shows a POST request to `localhost:63156/user/register`. The request body is a JSON object:

```
1 {  
2   "username": "sonikudi@gmail.com",  
3   "type": "admin",  
4   "password": "12345"  
5 }  
6
```

The response status is `503 Service Unavailable`, with a time of `2.06 s` and a size of `211 B`.

Below the response, the raw JSON body is displayed:

```
1 {  
2   "userId": null,  
3   "username": null,  
4   "type": null,  
5   "password": null  
6 }
```

Git

```
commit d863cace9f0994a297b6b21c745a41d39e6864f6 (HEAD -> main, origin/main, origin/HEAD)
Author: sulagna <sulagna.chatterjee27@gmail.com>
Date:   Tue Sep 12 16:15:12 2023 +0530

    Register Feign

commit 828f8b39c37b6f505b87ecf9c39db6fa99b9a7c8
Author: sulagna <sulagna.chatterjee27@gmail.com>
Date:   Wed Sep 6 14:14:01 2023 +0530

....skipping...
commit d863cace9f0994a297b6b21c745a41d39e6864f6 (HEAD -> main, origin/main, origin/HEAD)
Author: sulagna <sulagna.chatterjee27@gmail.com>
Date:   Tue Sep 12 16:15:12 2023 +0530

    Register Feign

commit 828f8b39c37b6f505b87ecf9c39db6fa99b9a7c8
Author: sulagna <sulagna.chatterjee27@gmail.com>
Date:   Wed Sep 6 14:14:01 2023 +0530

    Implemented user-register-service

commit bc6927a4136ce98d431710aca77d5fa26bdbe2c3
Author: sulagna <sulagna.chatterjee27@gmail.com>
Date:   Sat Aug 12 10:46:16 2023 +0530

    update test

commit f6fb64f6e5d00b414a6dfa1da33f0d13652162c
Author: sulagna <sulagna.chatterjee27@gmail.com>
Date:   Thu Aug 10 15:06:37 2023 +0530

    adminregistration testing

commit 3c9631cb0e4c76e521166161d3864a4497e30307
Author: sulagna <sulagna.chatterjee27@gmail.com>
Date:   Thu Aug 10 12:25:12 2023 +0530

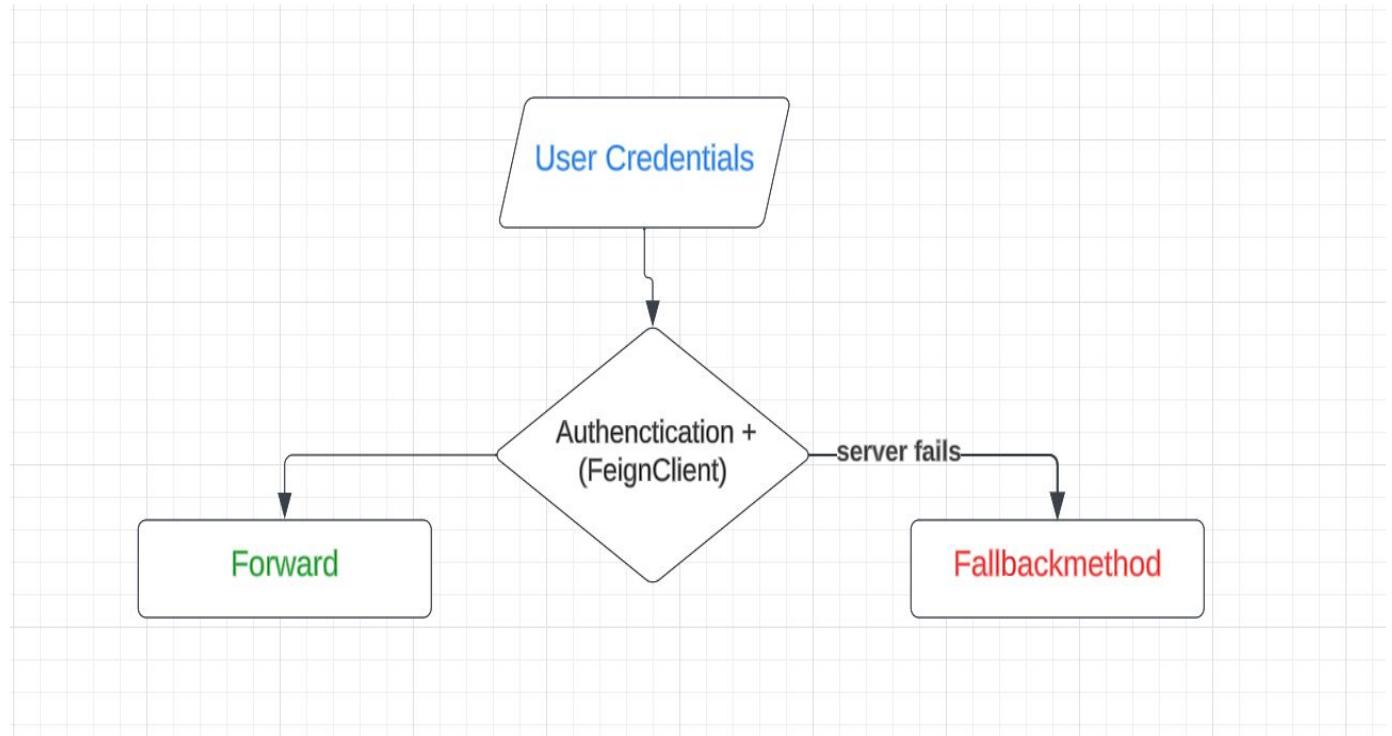
    new update

commit 56f803a086e3c5210d61d9fd131c8c6f8f2b9f0f
Author: sulagna <sulagna.chatterjee27@gmail.com>
```

Login

Swaroop

- Logic
 - Feign Client
 - Resilience4j
 - Retry



Proxy Service

```
12 You, last week | 1 author (You)
13 @FeignClient(value = "user-register-service")
14 public interface UserServiceProxy {
15     You, last week | 1 author (You)
16     @PostMapping(
17         value = "/user/login",
18         consumes = MediaType.APPLICATION_JSON_VALUE,
19         produces = MediaType.APPLICATION_JSON_VALUE
20     )
21     @CircuitBreaker(name = "login", fallbackMethod = "fallbackLogin")
22     @Retry(name = "user-register-service")
23     public ResponseEntity<Object> login(@RequestBody UserDTO userDTO);
24
25     public default ResponseEntity<Object> fallbackLogin(Long userId, Exception e) {
26         try {
27             return new ResponseEntity<Object>(
28                 new UserDTO(userId:null, username:null, type:null, password:null),
29                 HttpStatus.OK
30             );
31         } catch (Exception exception) {
32             return new ResponseEntity<Object>(
33                 new UserDTO(userId:null, username:null, type:null, password:null),
34                 HttpStatus.NOT_FOUND
35             );
36         }
37     }
38 }
```

Login

The screenshot shows the Postman application interface. The top navigation bar includes 'Home', 'Workspaces', 'Explore', a search bar 'Search Postman', and account options 'Sign In' and 'Create Account'. The left sidebar displays a 'History' section with a list of recent API requests, categorized by date ('Today' and 'Yesterday'). The main workspace shows a 'POST' request to 'localhost:8082/userLogin/login'. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "username": "admin345@gmail.com",  
3   "type": "admin",  
4   "password": "admin12"  
5 }
```

The 'Headers' tab shows 8 headers. Below the request, the response status is 200 OK, with a duration of 1258 ms and a size of 245 B. The response body is also displayed in JSON format:

```
1 {  
2   "userId": 10,  
3   "username": "admin345@gmail.com",  
4   "type": "admin",  
5   "password": "admin12"  
6 }
```

At the bottom, there are tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'. The bottom right corner features a 'Save Response' button.

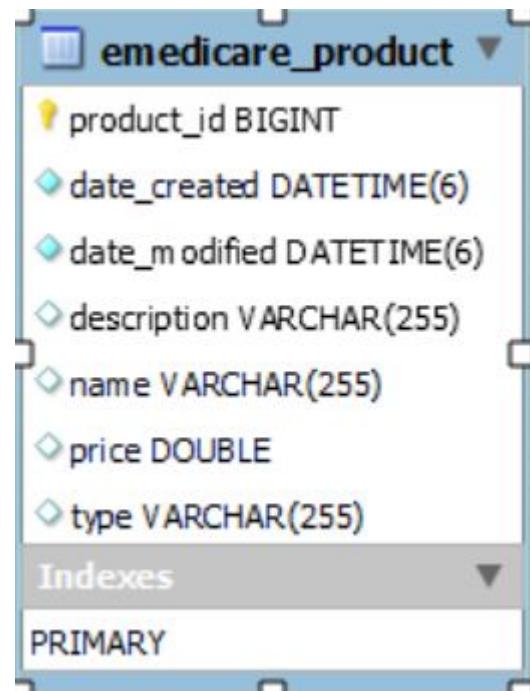
Table

Table: emedicare_product

Columns:

<u>product_id</u>	bigint AI PK
date_created	datetime(6)
date_modified	datetime(6)
description	varchar(255)
name	varchar(255)
price	double
type	varchar(255)

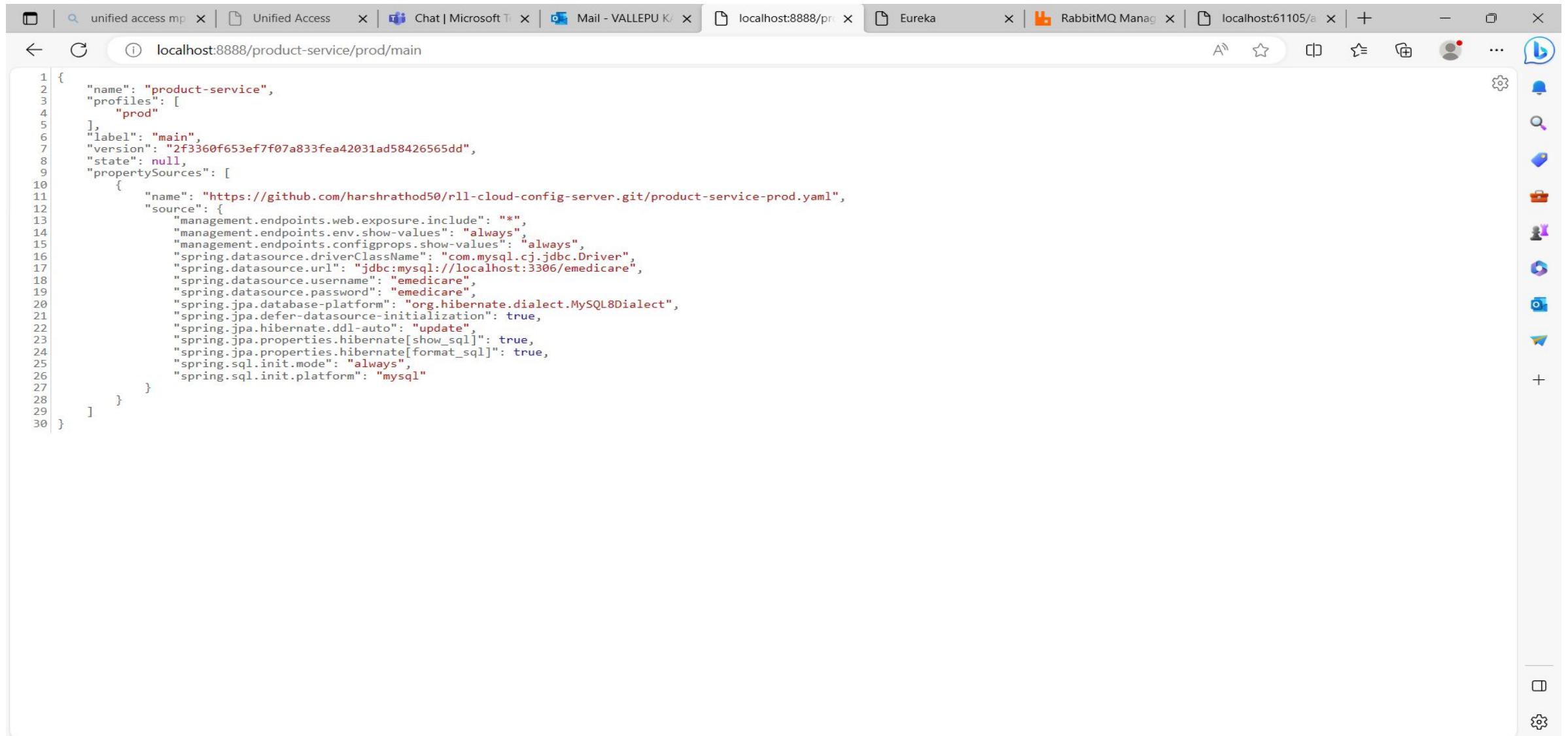
ER Diagram



Logic

- CRUD Implementation
- Cloud Config Server
- Eureka Server

Cloud Config Server



The screenshot shows a browser window with multiple tabs open. The active tab is 'localhost:8888/product-service/prod/main'. The page content displays a JSON configuration file with the following code:

```
1 {  
2     "name": "product-service",  
3     "profiles": [  
4         "prod"  
5     ],  
6     "label": "main",  
7     "version": "2f3360f653ef7f07a833fea42031ad58426565dd",  
8     "state": null,  
9     "propertySources": [  
10        {  
11            "name": "https://github.com/harshratod50/r11-cloud-config-server.git/product-service-prod.yaml",  
12            "source": {  
13                "management.endpoints.web.exposure.include": "*",
14                "management.endpoints.env.show-values": "always",
15                "management.endpoints.configprops.show-values": "always",
16                "spring.datasource.driverClassName": "com.mysql.cj.jdbc.Driver",
17                "spring.datasource.url": "jdbc:mysql://localhost:3306/emedicare",
18                "spring.datasource.username": "emedicare",
19                "spring.datasource.password": "emedicare",
20                "spring.jpa.database-platform": "org.hibernate.dialect.MySQL8Dialect",
21                "spring.jpa.defer-datasource-initialization": true,
22                "spring.jpa.hibernate.ddl-auto": "update",
23                "spring.jpa.properties.hibernate[show_sql]": true,
24                "spring.jpa.properties.hibernate[format_sql]": true,
25                "spring.sql.init.mode": "always",
26                "spring.sql.init.platform": "mysql"
27            }
28        }
29    ]
30}
```

Add Product

The screenshot shows the Postman application interface. The top navigation bar includes Home, Workspaces, API Network, Explore, a search bar, and various toolbars. The left sidebar displays 'My Workspace' with sections for Collections, Environments, History, and a pinned item. The main workspace shows a collection named 'Today' containing several POST and GET requests. A specific POST request is selected for the URL `http://localhost:62404/product/addOne`. The request details show it's a POST method with the URL `http://localhost:62404/product/addOne`. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   ...  
3   "name": "Cold Tablets",  
4   "type": "Tablet",  
5   "description": null,  
6   "price": 50.67  
7 }
```

The response section shows a status of 201 Created, a time of 350 ms, and a size of 345 B. Below the response, there are tabs for Body, Cookies, Headers (8), and Test Results. The Body tab is currently selected, showing the same JSON payload. The bottom navigation bar includes links for Runner, Capture requests, Cookies, Trash, and Help.

Read Product

The screenshot shows the Postman application interface. The left sidebar has tabs for 'My Workspace' (selected), 'Collections', 'Environments', and 'History'. The main area shows a list of recent requests under 'Today' and 'Yesterday'. A specific request for 'http://localhost:62404/product/readOne/1' is selected. The request details show a 'GET' method and the URL. The 'Body' tab displays the response in JSON format:

```
1  {
2   "productId": 1,
3   "name": "Dolo-650",
4   "type": "Tablet",
5   "description": null,
6   "price": 35.67
7 }
```

The status bar at the bottom indicates the response was 200 OK with a time of 45 ms and size of 335 B.

Update Product

The screenshot shows the Postman application interface. The top navigation bar includes Home, Workspaces, API Network, Explore, a search bar, and various user and settings icons. The main workspace is titled "My Workspace". On the left sidebar, there are sections for Collections, Environments, History, and a recent items section with a circular icon.

The central area displays a collection named "HTTP" with one item: "PUT http://localhost:62404/product/updateOne". The request details show a PUT method, URL, and headers. The "Body" tab is selected, showing a JSON payload:

```
1 {  
2   "productId": 14,  
3   "name": "Cough Tablets",  
4   "type": "Tablet",  
5   "description": null,  
6   "price": 50.67  
7 }
```

The response details show a status of 200 OK, time 185 ms, size 341 B. Below the response, there are tabs for Body, Cookies, Headers (8), and Test Results. The Body tab displays the same JSON response as the request body.

At the bottom, there are footer links for Online, Find and replace, Console, Runner, Capture requests, Cookies, Trash, and a help icon. The page number 29 is located in the bottom right corner.

Remove Product

The screenshot shows the Postman application interface. The top navigation bar includes Home, Workspaces, API Network, Explore, a search bar, and various toolbars. The left sidebar features sections for Collections, Environments, History, and a workspace overview. The main workspace displays a list of recent requests under 'Today' and 'Yesterday'. A specific DELETE request is selected, targeting the URL `http://localhost:62404/product/removeOne/10`. The request details panel shows the method as `DELETE`, the URL, and tabs for Params, Authorization, Headers (8), Body (green dot), Pre-request Script, Tests, and Settings. The Headers tab lists several key-value pairs. The Body tab is currently active, showing a table for Query Params with one row. Below the table, there are tabs for Body, Cookies, Headers (7), and Test Results. The Test Results section shows a status of 200 OK with a response time of 142 ms and a size of 212 B. The response body is displayed as a single character: '1'. At the bottom, there are links for Runner, Capture requests, Cookies, Trash, and Help.

Users Management

Pavan Kumar

- ❑ User management module handles user list related operations.
- ❑ User management services relies on user register services for user data.

Cloud Services

- config server
- eureka server
- OpenFeign
- Reselience4j
- Retry

View User

The screenshot shows the Postman application interface with the following details:

- Header Bar:** Home, Workspaces, API Network, Explore, Search Postman, Invite, Settings, Notifications, Upgrade.
- Left Sidebar:** My Workspace (Collections, Environments, History), New Collection (selected), GET get user.
- Request Section:** Method: GET, URL: http://localhost:8085/userManagement/readOne/4. Params tab selected. Query Params table:

	Key	Value	Description	Bulk Edit
	Key	Value	Description	
- Response Section:** Status: 200 OK, Time: 36 ms, Size: 238 B. Body tab selected. Response body (Pretty):

```
1 "userId": 4,
2   "username": "user@example.com",
3   "type": "user",
4   "password": "user"
```
- Bottom Navigation:** Online, Find and replace, Console, Runner, Capture requests, Cookies, Trash, 07:54 AM, 12-09-2023, 32.

Users list

Home Workspaces API Network Explore Search Postman Invite Settings Bell Upgrade No Environment

New Import < user POST add POST add POST add POST tak POST tak GET user GET get GET get > + ...

My Workspace Collections Environments History

HTTP New Collection / **get user list**

GET http://localhost:8085/userManagement/readMany/1/4

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Body Cookies Headers (5) Test Results Status: 200 OK Time: 114 ms Size: 466 B Save as Example

Pretty Raw Preview Visualize JSON

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
{
  "userId": 5,
  "username": "admin@yahoo.com",
  "type": "admin",
  "password": "admin"
},
{
  "userId": 6,
  "username": "user@gmail.com",
  "type": "user",
  "password": "user"
},
{
  "userId": 7,
  "username": "user@outlook.com",
  "type": "user",
  "password": "user"
}
```

Online Find and replace Console Runner Capture requests Cookies Trash

Watchlist ideas ENG 07:56 AM 12-09-2023 33

Delete user

Home Workspaces API Network Explore Search Postman Invite Settings Bell Upgrade New Import < adi ● POST addq ● POST addq ● POST tak ● POST tak ● GET user ● GET get ● GET get ● DEL delete > + ... No Environment

Collections Environments History

New Collection

GET get user
GET get user list
DEL delete user

New Collection

HTTP New Collection / **delete user**

DELETE http://localhost:8085/userManagement/removeOne/2

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies </>

Query Params

	Key	Value	Description	... Bulk Edit
	Key	Value	Description	

Body Cookies Headers (4) Test Results Status: 200 OK Time: 3.03 s Size: 123 B Save as Example

Pretty Raw Preview Visualize Text

1

Online Find and replace Console Runner Capture requests Cookies Trash

Watchlist ideas ENG 07:58 AM 12-09-2023

34

User-Dashboard-Service:

- Logic
 - Feign Client
 - Resilience4j
 - Retry

Search Product

Varshita

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists various collections, environments, and history items. In the center, a request is being viewed for the endpoint `http://localhost:62453/product/readOne/2`. The method is set to `GET`. The 'Params' tab is selected, showing a single query parameter 'Key'. The 'Body' tab displays the response in 'Pretty' format:

```
1 "productId": 2,
2 "name": "Adderall",
3 "type": "Capsule",
4 "description": null,
5 "price": 50.78
```

The status bar at the bottom indicates a `200 OK` response with a time of `96 ms` and a size of `247 B`.

Home Workspaces API Network Explore

Search Postman

Invite Settings Bell Upgrade

My Workspace New Import

Collections Environments History

http://localhost:62453/product/readMany/0/8

GET http://localhost:62453/product/readMany/0/8

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Description	... Bulk Edit
1	productID	6	Product ID	
2	name	EyeDrop	Product Name	
3	type	Drop	Product Type	
4	description	null	Product Description	
5	price	150.0	Product Price	

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 66 ms Size: 831 B Save as Example

Pretty Raw Preview Visualize JSON

```
36:   },
37:   {
38:     "productId": 6,
39:     "name": "EyeDrop",
40:     "type": "Drop",
41:     "description": null,
42:     "price": 150.0
43:   },
44:   {
45:     "productId": 7,
46:     "name": "Rabbies",
47:     "type": "Injection",
48:     "description": null,
49:     "price": 400.0
50:   },
51:   {
52:     "productId": 8,
53:     "name": "EarDrop",
54:     "type": "Drop",
55:     "description": null,
56:     "price": 50.0
57:   }
58: ]
```

Runner Capture requests Cookies Trash

Online Find and replace Console

37

- ❑ Change password services uses Feign for making HTTP requests to “user-register-service” to change a password

- ❑ It also uses the @CircuitBreaker and @Retry annotations for fault tolerance, specifying a fallback method in case of failures.

Cloud Services

- Config
- Eureka
- OpenFeign
- Reselience4j
- Retry

Change password

Admin:

The screenshot shows a POST request in Postman to `http://localhost:53026/user/changePassword`. The request body is JSON, containing the following data:

```
1
2 {
3   "username": "Rajiilax@gmail.com",
4   "type": "admin",
5   "password": "L@x2347"
6 }
```

The response status is 200 OK, with a response body identical to the request body:

```
1
2 {
3   "userId": 11,
4   "username": "Rajiilax@gmail.com",
5   "type": "admin",
6   "password": "L@x2347"
7 }
```

Change password

User:

The screenshot shows a POST request in Postman to `http://localhost:53026/user/changePassword`. The request body is a JSON object with fields `username`, `type`, and `password`. The response is a 200 OK status with a response body containing the user's ID, username, type, and password.

Request URL: `http://localhost:53026/user/changePassword`

Method: `PUT`

Body (JSON):

```
1
2   {
3     "username": "Rajii@gmail.com",
4     "type": "user",
5     "password": "R@jii47"
6   }
```

Response Status: 200 OK

Response Body (Pretty JSON):

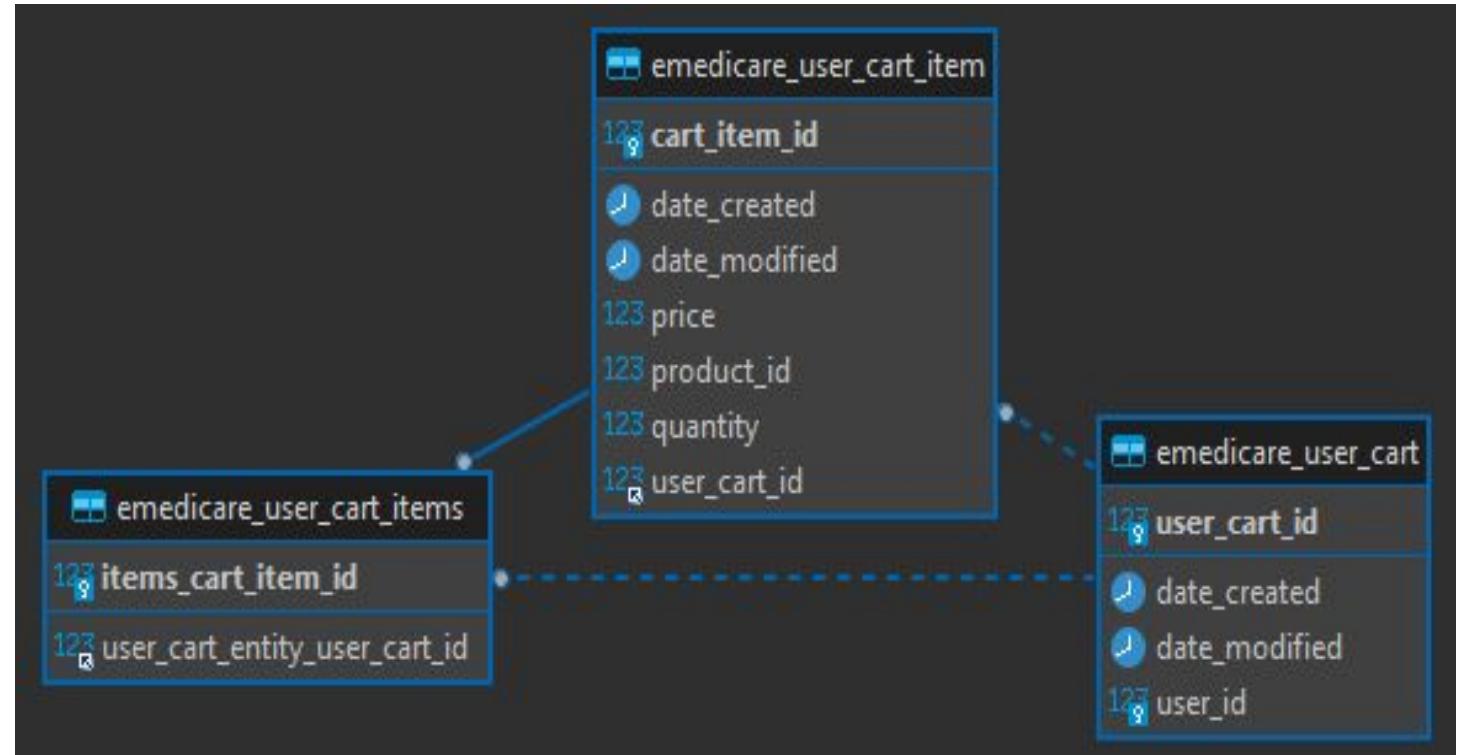
```
1   {
2     "userId": 12,
3     "username": "Rajii@gmail.com",
4     "type": "user",
5     "password": "R@jii47"
6   }
```

Cart

Harsh

Logic:

- Config Server
- Eureka Server
-



Add product to cart api

The screenshot shows a REST API testing interface with the following details:

- Method:** POST
- URL:** <http://localhost:8087/user/cart/addOneProduct>
- Status:** 201 Created (green checkmark icon) with a response time of 206 ms.
- Request Body:** JSON (selected tab), showing the following JSON payload:

```
1 {  
2   "userId": 9,  
3   "userCartId": 9,  
4   "productId": 8,  
5   "quantity": 6,  
6   "price": 300  
7 }
```
- Response Headers:** Headers tab (disabled)
- Response Content:** Text tab (disabled), showing an empty response body: "The response body is empty".
- Other Options:** Description, Headers, Query, Body (selected), Auth, Options, Text, JSON Tree, Form URL-Encoded, Multipart, GraphQL, Advanced Options, and a pretty print JSON button.

USER ADDRESS SERVICE

Logic:

- Config Server
- Eureka Server

emedicare_user_address	
123	user_address_id
abc	address_line1
abc	address_line2
abc	city
⌚	date_created
⌚	date_modified
abc	full_name
abc	pincode
abc	state
123	user_id

Microservice

```
UserAddressController.java ×
emedicare-user-address-service > src > main > java > com > emedicare > userAddressService > controller > UserAddressController.java
22  @RestController(value = "UserAddressController")
23  @Scope(value = "request")
24  @RequestMapping(value = "/user/address")
25  public class UserAddressController {
26      @Autowired
27      @Qualifier(value = "UserAddressService")
28      UserAddressService userAddressService;
29
30      You, 7 days ago | 1 author (You)
31      @PostMapping(
32          value = "/addOne",
33          produces = MediaType.APPLICATION_JSON_VALUE,
34          consumes = MediaType.APPLICATION_JSON_VALUE
35      )
36      public ResponseEntity<UserAddressDTO> addOne(@RequestBody UserAddressDTO userAddress) {
37          return new ResponseEntity<UserAddressDTO>(
38              this.userAddressService.addOne(userAddress),
39              HttpStatus.CREATED
40          );
41      }
42
43      You, 7 days ago | 1 author (You)
44      @GetMapping(
45          value = "/readOne/{userAddressId}",
46          produces = MediaType.APPLICATION_JSON_VALUE
47      )
48      public ResponseEntity<Object> readOne(
49          @PathVariable("userAddressId") Long userAddressId
50      ) {
51          UserAddressDTO userAddressDTO = this.userAddressService.readOne(userAddressId);
52          if (userAddressDTO != null) {
53              return new ResponseEntity<Object>(userAddressDTO, HttpStatus.OK);
54          }
55          return new ResponseEntity<Object>(headers: null, HttpStatus.NOT_FOUND);
56      }
57  }
```

ADD ADDRESS

Pavithra

The screenshot shows the Postman application interface for making a POST request to the endpoint `http://localhost:58611/user/address/addOne`.

Request Headers: There are 9 headers defined.

Request Body: The body is set to `raw JSON`. The JSON payload is:

```
1 {
2   "userId": 5,
3   "fullName": "Pavithra S",
4   "addressLine1": "Avadi",
5   "city": "Chennai",
6   "state": "Tamil Nadu",
7   "pincode": "600072"
8 }
```

Response Headers: The response status is `201 Created`, with a response time of `383 ms` and a size of `323 B`.

Response Body: The response JSON is:

```
1 {
2   "userAddressId": 3,
3   "userId": 5,
4   "fullName": "Pavithra S",
5   "addressLine1": "Avadi",
6   "addressLine2": null,
7   "city": "Chennai",
8   "state": "Tamil Nadu",
9 }
```

READ ADDRESS

The screenshot shows the Postman application interface. At the top, there is a header bar with a blue 'HTTP' icon, the URL 'http://localhost:58611/user/address/readOne/2', and various action buttons like 'Save', 'Edit', and 'Send'. Below the header, the main interface has several tabs: 'GET' (selected), 'Params', 'Authorization', 'Headers (7)', 'Body' (selected), 'Pre-request Script', 'Tests', and 'Settings'. Under the 'Body' tab, there are options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', 'GraphQL', and 'JSON' (selected). A large text area labeled '1' contains the JSON response. At the bottom, there are tabs for 'Body' (selected), 'Cookies', 'Headers (5)', and 'Test Results'. On the right side, there are status indicators: a globe icon, '200 OK', '56 ms', '318 B', and a 'Save as Example' button. The bottom-most section displays the raw JSON data.

```
1
2     "userAddressId": 2,
3     "userId": 3,
4     "fullName": "Pavithra S",
5     "addressLine1": "Avadi",
6     "addressLine2": null,
7     "city": "Chennai",
8     "state": "Tamil Nadu",
```

UPDATE ADDRESS

The screenshot shows the Postman application interface for making a **PUT** request to the endpoint **http://localhost:58611/user/address/updateOne**. The request body is defined as follows:

```
1 {  
2   "userAddressId": 3,  
3   "userId": 5,  
4   "fullName": "Pavithra Singaram",  
5   "addressLine1": "Avadi",  
6   "city": "Chennai",  
7   "state": "Tamil Nadu",  
8   "pincode": "600072"  
}
```

The response received is a **200 OK** status with a response time of **177 ms** and a response size of **325 B**. The response body is identical to the request body.

DELETE ADDRESS

The screenshot shows the Postman application interface for making a DELETE request to remove an address. The URL in the header is `http://localhost:58611/user/address/removeOne/3`. The method is set to `DELETE`. The `Body` tab is selected, showing a single parameter named `1`. The response at the bottom indicates a `200 OK` status with `22 ms` latency and `123 B` size.

HTTP `http://localhost:58611/user/address/removeOne/3`

`DELETE` `http://localhost:58611/user/address/removeOne/3`

Save `Send`

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Beautify

1

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize Text

200 OK 22 ms 123 B Save as Example

48

Logic

Main part:

- 1.Create microService
- 2.Connect with Config and Eureka Server
- 3.Feign Client
- 4.Reselience4j
- 5.Retry
- 5.Zipkin

Table

Table: emedicare_payment

Columns:

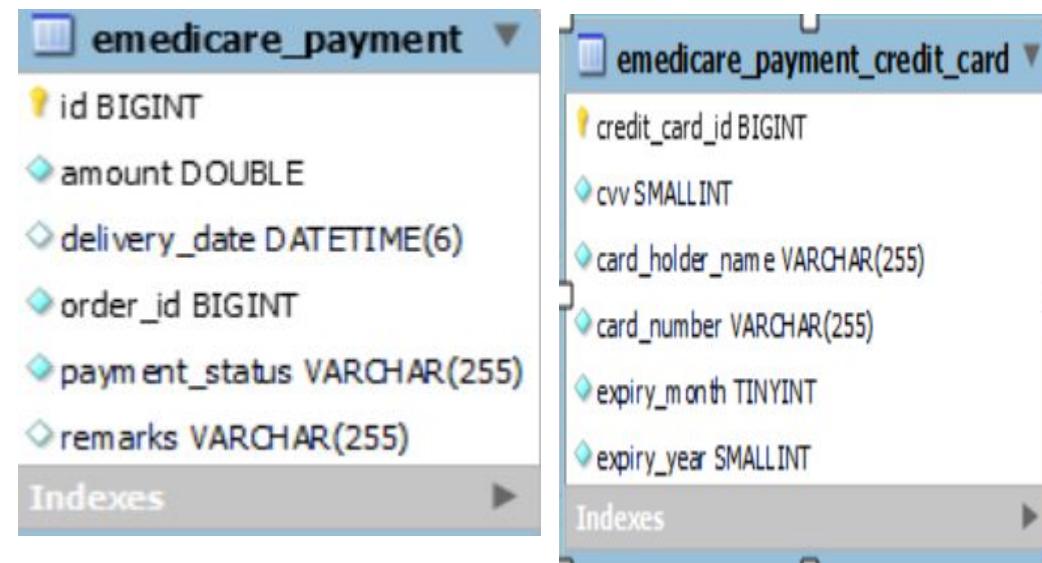
id	bigint AI PK
amount	double
delivery_date	datetime(6)
order_id	bigint
payment_status	varchar(255)
remarks	varchar(255)

Table: emedicare_payment_credit_card

Columns:

credit_card_id	bigint AI PK
cvv	smallint
card_holder_name	varchar(255)
card_number	varchar(255)
expiry_month	tinyint
expiry_year	smallint

ER Diagram



Microservices

The screenshot shows a Java code editor interface with a dark theme. The file being edited is `CreditCardController.java`, located in the package `com.emedicare.userPaymentService.controller`. The code implements a REST controller for managing credit cards.

```
1 package com.emedicare.userPaymentService.controller;
2
3 import com.emedicare.userPaymentService.entity.CreditCardEntity;
4 import com.emedicare.userPaymentService.service.CreditCardService;
5 import java.util.List;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.context.annotation.Scope;
8 import org.springframework.http.HttpStatus;
9 import org.springframework.http.ResponseEntity;
10 import org.springframework.web.bind.annotation.*;
11
12 You, 7 days ago | 1 author (You)
13 @RestController(value = "CreditCardController")
14 @Scope(value = "request")
15 @RequestMapping("/user/payment/creditCard")
16 public class CreditCardController {
17     @Autowired
18     private CreditCardService creditCardService;
19
20     @PostMapping("/addOne")
21     public ResponseEntity<CreditCardEntity> addOne(
22         @RequestBody CreditCardEntity creditCard
23     ) {
24         return new ResponseEntity<CreditCardEntity>(
25             creditCardService.addOne(creditCard),
26             HttpStatus.CREATED
27         );
28     }
29
30     @GetMapping("/readOne/{creditCardId}")
31     public ResponseEntity<Object> readOne(@PathVariable("creditCardId") Long creditCardId) {
32         CreditCardEntity creditCard = creditCardService.readOne(creditCardId);
33         if (creditCard != null) {
```

PAYMENT SERVICE

The screenshot shows the Postman application interface for a payment service. The top navigation bar includes 'Overview', 'POST localhost:57630/user, ● + ...', 'No Environment', and a dropdown menu. The main header shows 'localhost:57630/user/payment/creditCard/addOne' with an 'HTTP' icon. Below the header, the method is set to 'POST' and the URL is 'localhost:57630/user/payment/creditCard/addOne'. A 'Send' button is visible on the right.

The request configuration section includes tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is selected, showing the following JSON payload:

```
1 {  
2   "cardNumber": "3473-5674-4328-743",  
3   "cardHolderName": "Priya S",  
4   "expiryMonth": "7",  
5   "expiryYear": "2026",  
6   "cvv": "764"  
7 }  
8  
9  
10
```

The 'Body' tab also includes options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw', 'binary', 'GraphQL', and 'JSON'. The 'JSON' option is currently selected. On the right side of the body editor, there are 'Cookies' and 'Beautify' buttons.

At the bottom of the interface, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The 'Body' tab is selected. The status bar at the bottom right shows 'Status: 201 Created', 'Time: 26 ms', 'Size: 293 B', and 'Save as Example'.

The 'Body' section also contains a preview area with tabs for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'. The 'Pretty' tab is selected, displaying the same JSON payload as the editor above.

```
1 {  
2   "creditCardId": 10,  
3   "cardNumber": "3473-5674-4328-743",  
4   "cardHolderName": "Priya S",  
5   "expiryMonth": 7,  
6   "expiryYear": 2026,  
7   "cvv": 764  
8 }
```

PAYMENT FEIGN SERVICE

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Overview', a search bar containing 'POST localhost:50821/userFeign/payment/creditCard/addOne', and a status indicator 'No Environment'. Below the search bar, there are buttons for 'Save' and 'Send'.

The main area is a request builder window. It shows a 'POST' method selected, the URL 'localhost:50821/userFeign/payment/creditCard/addOne', and a 'Body' tab active. Under 'Body', the 'JSON' option is selected, and the following JSON payload is displayed:

```
1
2   - "cardNumber": "3473-5674-4328-374",
3   - "cardHolderName": "Priya S",
4   - "expiryMonth": 9,
5   - "expiryYear": 2024,
6   - "cvv": 453
```

Below the body editor, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Tests', and 'Settings'. On the right side of the interface, there are sections for 'Cookies' and 'Beautify'.

The screenshot shows the Postman interface after sending the request. At the top, the status is 'Status: 200 OK' and the time taken is 'Time: 2.94 s'. There are buttons for 'Save as Example' and a copy icon.

The main area displays the response body under the 'Pretty' tab. The response is identical to the request body:

```
1
2   - "creditCardId": 49,
3   - "cardNumber": "3473-5674-4328-374",
4   - "cardHolderName": "Priya S",
5   - "expiryMonth": 9,
6   - "expiryYear": 2024,
7   - "cvv": 453
```

Below the response, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The 'Pretty' tab is currently selected.

Fallback

The screenshot shows the Postman application interface. At the top, there are three failed requests listed: "POST localhost:52386/user" (status 503), "POST localhost:54888//use" (status 503), and "GET localhost:53444/userF" (status 503). Below these, a new request is being configured:

- Method:** GET
- URL:** localhost:53444/userFeign/payment/creditCard
- Body:** None (selected)
- Headers:** (6) (selected)
- Tests:** None
- Settings:** None

The "Body" tab shows the message: "This request does not have a body".

At the bottom of the request configuration, the status is shown as "Status: 503 Service Unavailable".

The "Body" tab is currently selected, showing the response content:

```
1 []
```

Below the body, there are tabs for "Cookies", "Headers (4)", and "Test Results".

At the very bottom of the interface, there are several global buttons: "Runner", "Capture requests", "Cookies", "Trash", and a help icon.

Zipkin

localhost:9411/zipkin/?lookback=15m&endTs=1694577283168&limit=10

ENGLISH ▾ Search by trace ID

4 Results EXPAND ALL COLLAPSE ALL Service filters ▾

		Start Time	Spans	Duration	
Root					
▼ payment-feign-service: http get /userfeign/payment/creditcard/{id}		an hour ago (09/13 09:21:51:938)	1	1.176s	
▼ payment-feign-service: http get /userfeign/payment/creditcard		an hour ago (09/13 09:23:33:012)	1	1.113s	
▼ payment-feign-service: http delete /userfeign/payment/creditcard/{id}		an hour ago (09/13 09:24:28:075)	1	1.075s	
▼ payment-feign-service: http post		an hour ago (09/13 09:21:44:031)	1	16.110ms	

PAYMENT SERVICES

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

commit d3c56607eb559f4b51011342ad939be0e9838ec0 (HEAD -> main, origin/main, origin/HEAD)
Author: Priya S <priyaa123singha@gmail.com>
Date: Tue Sep 12 03:07:33 2023 +0530

    added Zipkin

commit bd4e1d88c6281f9881cad9e75b4b7ed22886ed30
Author: Priya S <priyaa123singha@gmail.com>
Date: Mon Sep 11 15:55:02 2023 +0530

    Payment Services

commit f21020d969de000f16b9cc868f9af6195c75a8b2
Author: Priya S <priyaa123singha@gmail.com>
Date: Sun Sep 10 16:23:58 2023 +0530

    Feign client minor issues

commit 4690d04261b203a7281271a9be77588011b476ab
Author: Priya Singha <85759510+Priya678438@users.noreply.github.com>
Date: Fri Sep 8 17:56:22 2023 +0530

    Update application.yaml

commit 7a453c15bcf65488eb8a59256aaa737c2cb7715a
Author: Priya S <priyaa123singha@gmail.com>
Date: Fri Sep 8 17:24:22 2023 +0530

    Payment service with feign client

commit 9f7cc4eec2a1cf86c48850839206eeb8d784506a
Author: Priya Singha <85759510+Priya678438@users.noreply.github.com>
Date: Fri Sep 8 09:57:38 2023 +0530

    Update application.properties

commit 3ce7c2660e1ef4834232ee083b0378cedf9b4ab6
Author: Priya S <priyaa123singha@gmail.com>
Date: Wed Sep 6 11:41:24 2023 +0530

:
```

- User shipping service will provide efficient and secure shipping options.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Bus', 'RII-2', and 'SL-PHASE3'. The main area shows a POST request to 'http://localhost:8088/shipping/add'. The 'Body' tab is selected, displaying the following JSON payload:

```
1 {  
2   "shipId": 2,  
3   "userId": 1008,  
4   "AddressId": 1063  
5 }
```

Below the body, the response status is shown as 200 OK with 565 ms and 160 B. The response body is 'Added'.

USER-SHIPPING SERVICE

HTTP New Collection / 2 Save No Environment

GET http://localhost:8088/shipping/shippingdetails Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Code Cookies

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 14 ms Size: 640 B

Pretty Raw Preview JSON

```
1 [  
2 {  
3     "shipId": 1,  
4     "user_Id": 1006,  
5     "userAddressId": 1005,  
6     "dateCreated": "2023-09-11T12:30:30.945+00:00"  
7 },  
8 {  
9     "shipId": 2,  
10    "user_Id": null,  
11    "userAddressId": null,  
12    "dateCreated": "2023-09-11T13:38:19.143+00:00"  
13 },  
14 {  
15     "shipId": 3,  
16     "user_Id": null,  
17     "userAddressId": null,  
18     "dateCreated": "2023-09-11T13:39:26.623+00:00"  
19 },  
20 {  
21     "shipId": 4,
```

Spring Boot-AdminUserLoginApplication<Admin-User-Login> (rl) Git Graph

CONCLUSION

- E-medicare microservices represent a transformative step towards modernizing and improving healthcare delivery.
- By enhancing efficiency, interoperability, patient engagement, and data security, they contribute to better user outcomes and cost-effective healthcare services.
- However, successful implementation requires careful planning, investment, and ongoing maintenance to ensure that the system operates effectively and securely.

THANK YOU