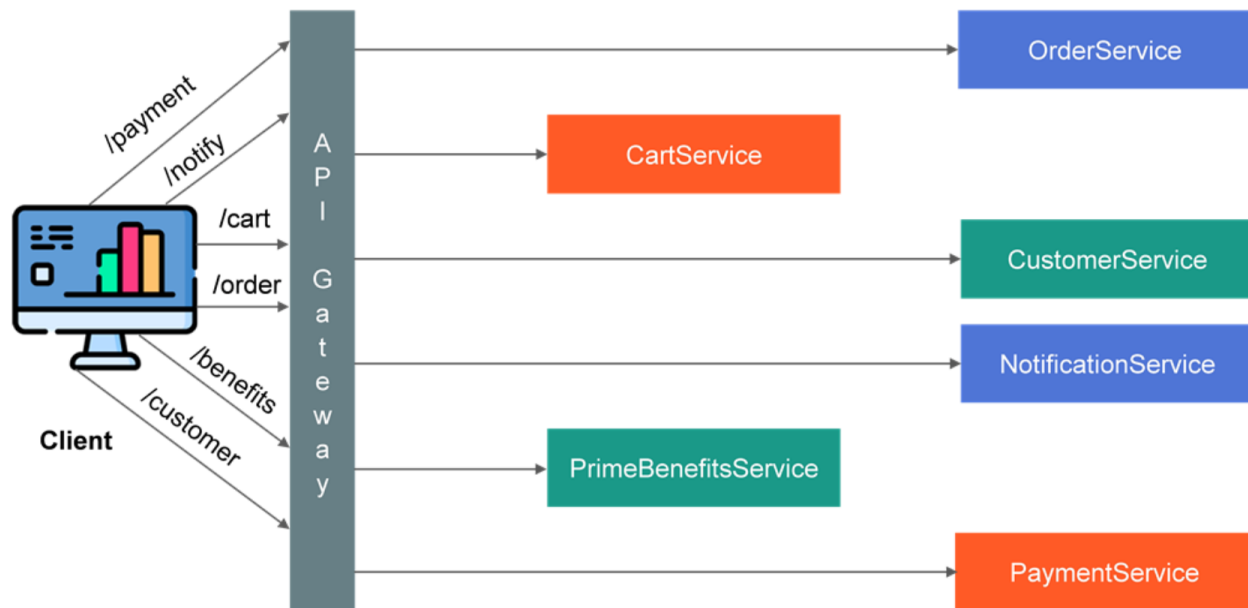


Application Workflow – Multiple Services



Think and Tell

In an application with multiple microservices, routing happens through an API gateway.

- Will the API gateway maintain the details of all the services in the application?
- How will the API gateway know the health of a particular service?
- Will the API gateway still route the request to a service that is down?

Register Microservices on a Netflix Eureka Discovery Server



Learning Objectives

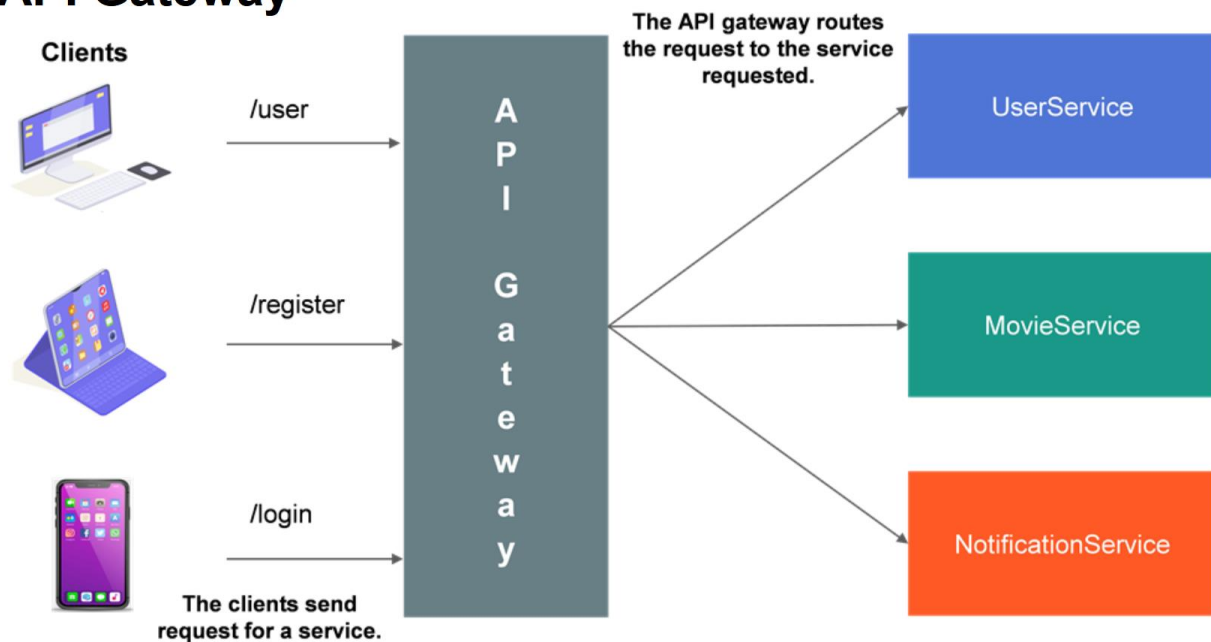
- Define the service discovery design pattern
- Implement the service discovery server using Eureka
- Register the services on the Eureka server



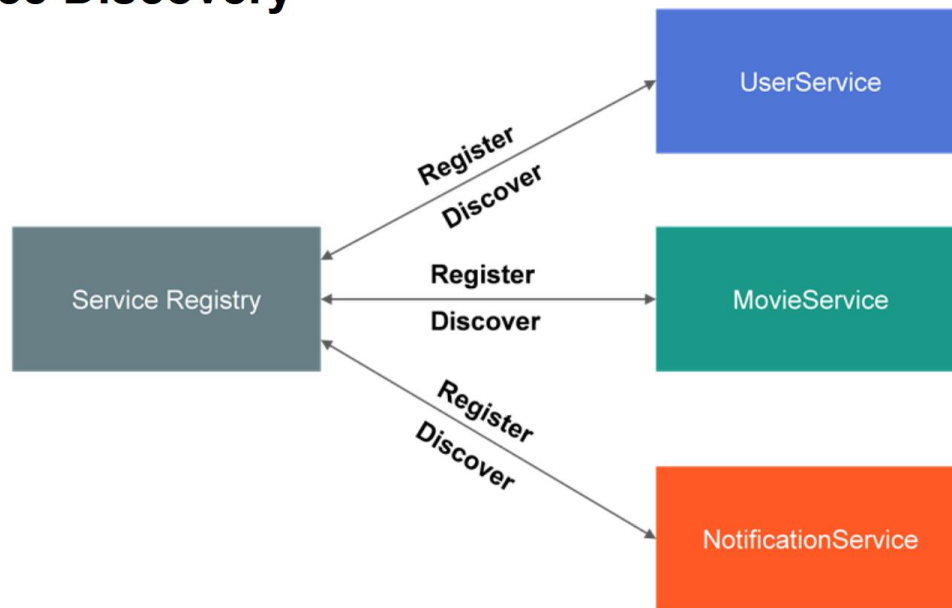
Service Discovery

- Services typically need to call one another for effective working of an application.
- This is how microservices locate each other on a network.
- Multiple instances of the same microservice can be executed at any given point in time.
- Service discovery makes it easy for clients to be serviced depending on the availability of a service.
- Service discovery is the first step towards granular scaling.
- Implementation includes a server that maintains a list of services registered on it.
- Clients connect to the server to update and retrieve the service addressed.

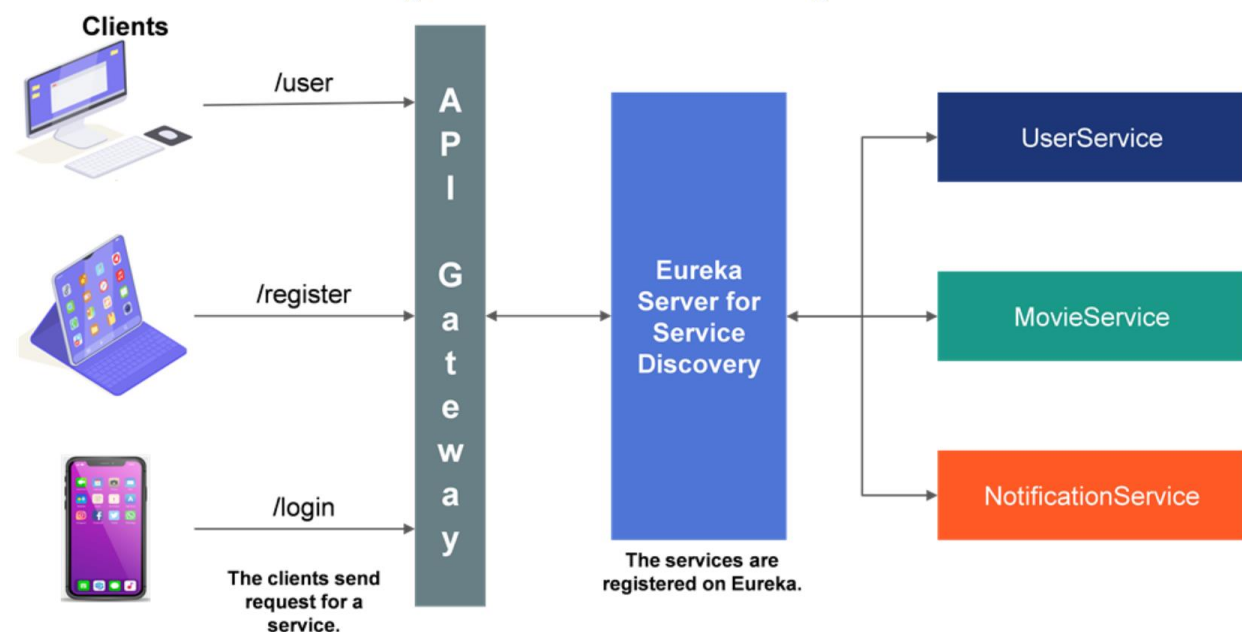
API Gateway



Service Discovery



Service Discovery and API Gateway



Service Discovery Design Pattern - How Does This Work?

- Services are registered to the discovery server.
- Service discovery server listens for the registered service when they start up.
- The service discovery server sends a heartbeat continuously to check for services.
- There is a timeout period to assume that the service is offline.
- Once the time out threshold is reached, the server assumes that the service is down and does not route the client request to that service until it is back up.
- The service discovery pattern is called non-invasive, as it does not alter the code in any of the microservices.

Create the Discovery Server

- Spring Cloud Netflix provides Netflix OSS integrations for Spring Boot apps through autoconfiguration. With a few simple annotations, you can quickly enable and configure the common patterns inside the application.
- The patterns provided include Service Discovery (Eureka), Circuit Breaker (Hystrix), Intelligent Routing (Zuul) and Client-Side Load Balancing (Ribbon).
- As the first step, create a Eureka Discovery Server. Use Spring Initializr and bootstrap the dependencies.

Dependencies

ADD DEPENDENCIES... CTRL + B

Eureka Server

SPRING CLOUD DISCOVERY

spring-cloud-netflix Eureka Server.

Enable the Server in the Application

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

application.yml

```
spring:
  application:
    name: eureka-service
server:
  port: 8761
```

- Use the `@EnableEurekaServer` annotation in the main class.
- Mention the service name and the server port where the server will run in the `application.properties` or `application.yml` file.

Eureka Server

- Start the server and access the service running at <http://localhost:8761/eureka>.

The screenshot displays the Spring Eureka Server web interface. At the top, there is a navigation bar with the 'spring Eureka' logo and links for 'HOME' and 'LAST 1000 SINCE STARTUP'. The main content area is divided into three sections:

- System Status:** A table showing environment and data center details, and a summary of system metrics.
- DS Replicas:** A section showing the local replica status.
- Instances currently registered with Eureka:** A table listing registered instances.

System Status	
Environment	N/A
Data center	N/A
Current time	2021-07-14T22:34:51 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	3
Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
EUREKA-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:eureka-service:8761

Register the Services on the Eureka Server

- To register a microservice that is also called a Eureka client, on the Eureka server, follow the given steps.
- **Step 1:** Add the below dependencies in the pom.xml of the service.

```
<properties>
  <java.version>11</java.version>
  <spring-cloud.version>2020.0.3</spring-cloud.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-dependencies</artifactId>
        <version>${spring-cloud.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
</dependencies>
```

- **Step 2:** Enable the microservice with `@EnableEurekaClient` annotation.

```
@SpringBootApplication
@EnableEurekaClient
public class UserAuthenticationServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(UserAuthenticationServiceApplication.class, args);
    }
}
```

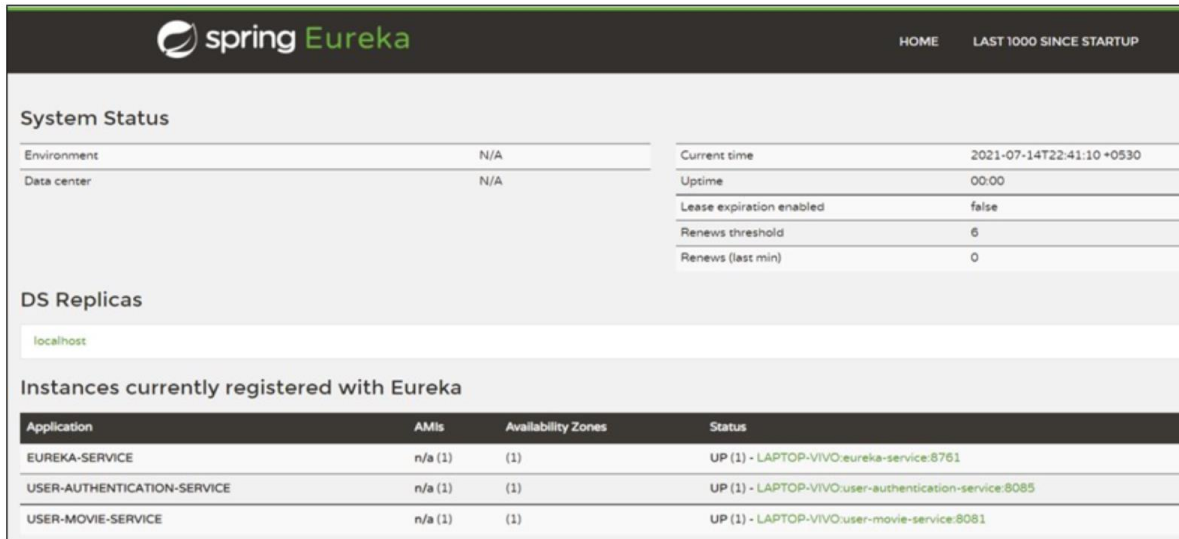
- **Step 3:** Modify the application.yml file.

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
    fetchRegistry: true
    registerWithEureka: true
```

- **Step 4:** Run the individual services.

Eureka Server With the Services Registered

- Refresh the browser at <http://localhost:8761/eureka>.



The screenshot shows the Spring Eureka Server web interface. At the top, there's a header with the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'. Below the header, the 'System Status' section displays a table with system metrics. To the right of this table, another table shows configuration details like 'Current time', 'Uptime', 'Lease expiration enabled', 'Renews threshold', and 'Renews (last min)'. Below the system status, the 'DS Replicas' section shows 'localhost'. The main section, 'Instances currently registered with Eureka', contains a table listing three services: EUREKA-SERVICE, USER-AUTHENTICATION-SERVICE, and USER-MOVIE-SERVICE, each with details on AMIs, Availability Zones, and their status.

System Status	
Environment	N/A
Data center	N/A

Current time	2021-07-14T22:41:10 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	6
Renews (last min)	0

DS Replicas

localhost

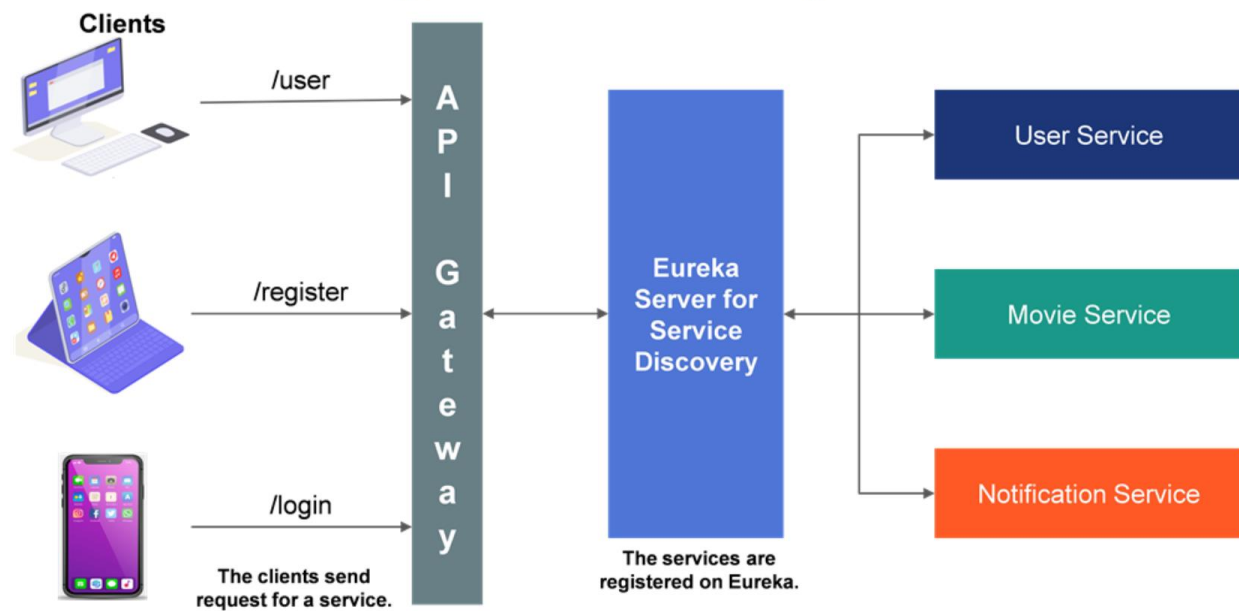
Application	AMIs	Availability Zones	Status
EUREKA-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:eureka-service:8761
USER-AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:user-authentication-service:8085
USER-MOVIE-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:user-movie-service:8081

Register the API Gateway onto the Eureka Server

- The Spring Cloud API Gateway must be registered on the Eureka server. As a client, we need to add these dependencies.
- As shown in the image, the route can also be written using the application name we configured in the application.yml file, instead of the URI of the application.

```
@Configuration
public class AppConfig {
    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
            .route(p -> p
                .path( ...patterns: "/api/v1/**")
                .uri("lb://user-authentication-service"))
            .route(p->p
                .path( ...patterns: "/api/v2/user/**", "/api/v2/register")
                .uri("lb://user-movie-service"))
            .build();
    }
}
```


Service Discovery and API Gateway



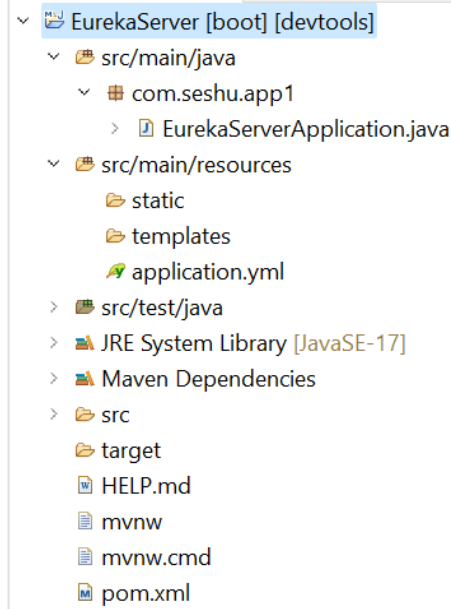
Example:

Create a spring starter project as **EurekaServer** with following dependencies.

Spring web

Spring Devtools

Eureka Server



EurekaServerApplication.java

```
package com.seshu.app1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

application.yml

```
server:
  port: 8761

spring:
  application:
    name: eureka-service

eureka:
  client:
    fetchRegistry: false
    registerWithEureka: false
```

Update User Authentication Service

Update pom.xml with new Eureka Client dependency.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.1</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.seshu</groupId>
  <artifactId>user-authentication-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>user-authentication-service</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
    <spring-cloud.version>2021.0.3</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
```

```
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>
    <dependency>
        <groupId>javax.xml.bind</groupId>
        <artifactId>jaxb-api</artifactId>
        <version>2.4.0-b180830.0359</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
    </dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

application.yml

```
server:
  port: 8085
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/adidb
    username: root
    password: seshu
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL57Dialect

  application:
    name: user-authentication-service

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
    fetchRegistry: true
    registerWithEureka: true
```

Update starter class

UserAuthenticationServiceApplication.java

```
package com.seshu.app1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class UserAuthenticationServiceApplication {
    public static void main(String[] args) {

        SpringApplication.run(UserAuthenticationServiceApplication.class,
args);
    }
}
```

Update User Movie Service

Update pom.xml with Eureka Client dependencies;

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.1</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.seshu</groupId>
  <artifactId>user-movie-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>user-movie-service</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
    <spring-cloud.version>2021.0.3</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt</artifactId>
```



```
        <version>0.9.1</version>
    </dependency>
    <dependency>
        <groupId>javax.xml.bind</groupId>
        <artifactId>jaxb-api</artifactId>
        <version>2.4.0-b180830.0359</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
    </dependency>

</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

</project>
```

application.yml

```
server:
  port: 8081
spring:
  data:
    mongodb:
      uri: mongodb://localhost:27017/
      database: moviesdb
  application:
    name: user-movie-service

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
    fetchRegistry: true
    registerWithEureka: true
```

Update starter class

UserMovieServiceApplication.java

```
package com.seshu.app1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class UserMovieServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(UserMovieServiceApplication.class, args);
    }
}
```

Update Spring-cloud-api-gateway application

Update pom.xml with Eureka Client dependencies;

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.1</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.seshu</groupId>
  <artifactId>spring-cloud-api-gateway</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>spring-cloud-api-gateway</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
    <spring-cloud.version>2021.0.3</spring-cloud.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>io.jsonwebtoken</groupId>
      <artifactId>jjwt</artifactId>
      <version>0.9.1</version>
    </dependency>

    <dependency>
      <groupId>javax.xml.bind</groupId>
      <artifactId>jaxb-api</artifactId>
      <version>2.4.0-b180830.0359</version>
    </dependency>

    <dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
    </dependency>

</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

</project>
```

application.yml

```
server:
  port: 9000
spring:
  application:
    name: spring-cloud-api-gateway
  main:
    web-application-type: reactive

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
    fetchRegistry: true
    registerWithEureka: true
```

AppConfig.java

```
package com.seshu.appl.config;

import org.springframework.boot.web.servlet.FilterRegistrationBean;
import org.springframework.cloud.gateway.route.RouteLocator;
import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.seshu.appl.filter.JwtFilter;

@Configuration
public class AppConfig {
    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
            .route(p -> p
                .path("/api/v1/**")
                .uri("lb://user-authentication-service"))
            .route(p -> p
                .path("/api/v2/user/**", "/api/v2/register")
                .uri("lb://user-movie-service"))
            .build();
    }
    @Bean
    public FilterRegistrationBean jwtFilterBean(){
        FilterRegistrationBean filterRegistrationBean = new
        FilterRegistrationBean();
        filterRegistrationBean.setFilter(new JwtFilter());
        filterRegistrationBean.addUrlPatterns("/api/v2/user/*");
        return filterRegistrationBean;
    }
}
```

Update starter class

SpringCloudApiGatewayApplication.java

```
package com.seshu.app1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class SpringCloudApiGatewayApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringCloudApiGatewayApplication.class,
args);
    }
}
```


Execution Steps:

1. Eureka Server
2. Spring Cloud Api Gateway
3. User Authentication Service
4. User Movie Service

Services are registered in Eureka Server

<http://localhost:8761/>

The screenshot shows the Eureka Server web interface in a browser window. The address bar shows `localhost:8761`. The interface includes a header with navigation links, a main content area with a sidebar, and a footer. The sidebar contains links for Home, Instances, Applications, Clusters, and Metrics. The main content area displays the following information:

- Renews threshold:** 6
- Renews (last min):** 12
- DS Replicas:** localhost
- Instances currently registered with Eureka:**

Application	AMIs	Availability Zones	Status
SPRING-CLOUD-API-GATEWAY	n/a (1)	(1)	UP (1) - host.docker.internal:9000
USER-AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:8085
USER-MOVIE-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal:8081

General Info

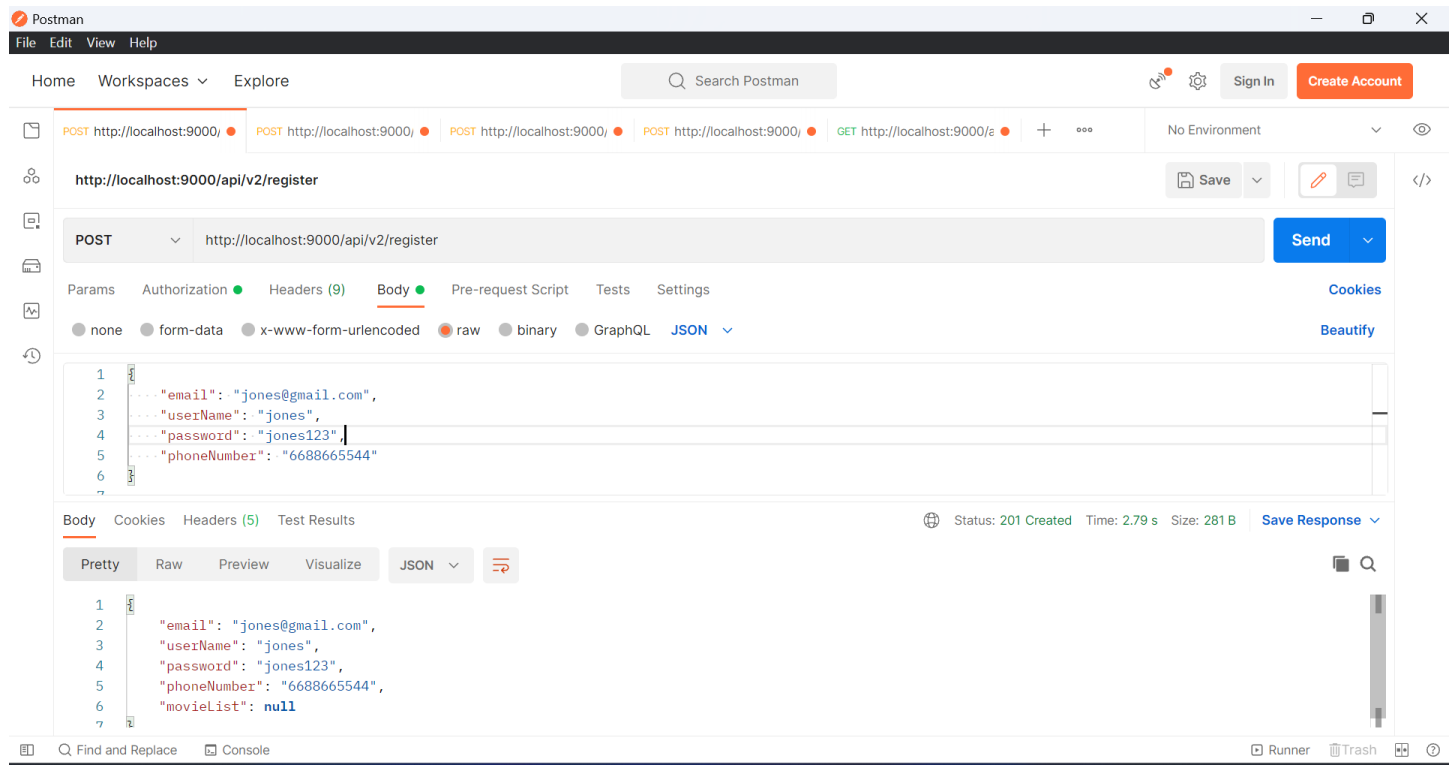
Name	Value
total-avail-memory	60mb
num-of-cpus	4
CURRENT-MEMORY-USAGE	36mb (60%)

Test the application using Postman client tool

Register a new user;

<http://localhost:9000/api/v2/register>

```
{
  "email": "jones@gmail.com",
  "userName": "jones",
  "password": "jones123",
  "phoneNumber": "6688665544"
}
```



Save user in the Movie Service;

<http://localhost:9000/api/v1/user>

The screenshot shows the Postman application interface. At the top, there's a menu bar with 'File', 'Edit', 'View', and 'Help'. Below it is a toolbar with 'Home', 'Workspaces', 'Explore', and a search bar. The main workspace shows a collection of requests. The selected request is a POST to 'http://localhost:9000/api/v1/user'. The 'Body' tab is active, showing a JSON payload:

```
{  "email": "jones@gmail.com",  "password": "jones123"}
```

. The 'Send' button is visible. Below the request editor, the 'Body' tab of the response is shown, displaying the same JSON payload in a pretty-printed format. The status bar at the bottom indicates 'Status: 201 Created', 'Time: 808 ms', and 'Size: 218 B'.

Postman

File Edit View Help

Home Workspaces Explore

Search Postman

Sign In Create Account

No Environment

http://localhost:9000/api/v1/user

Save

POST http://localhost:9000/api/v1/user

Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

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

```
1 {
2   "email": "jones@gmail.com",
3   "password": "jones123"
4 }
```

Body Cookies Headers (5) Test Results

Status: 201 Created Time: 808 ms Size: 218 B Save Response

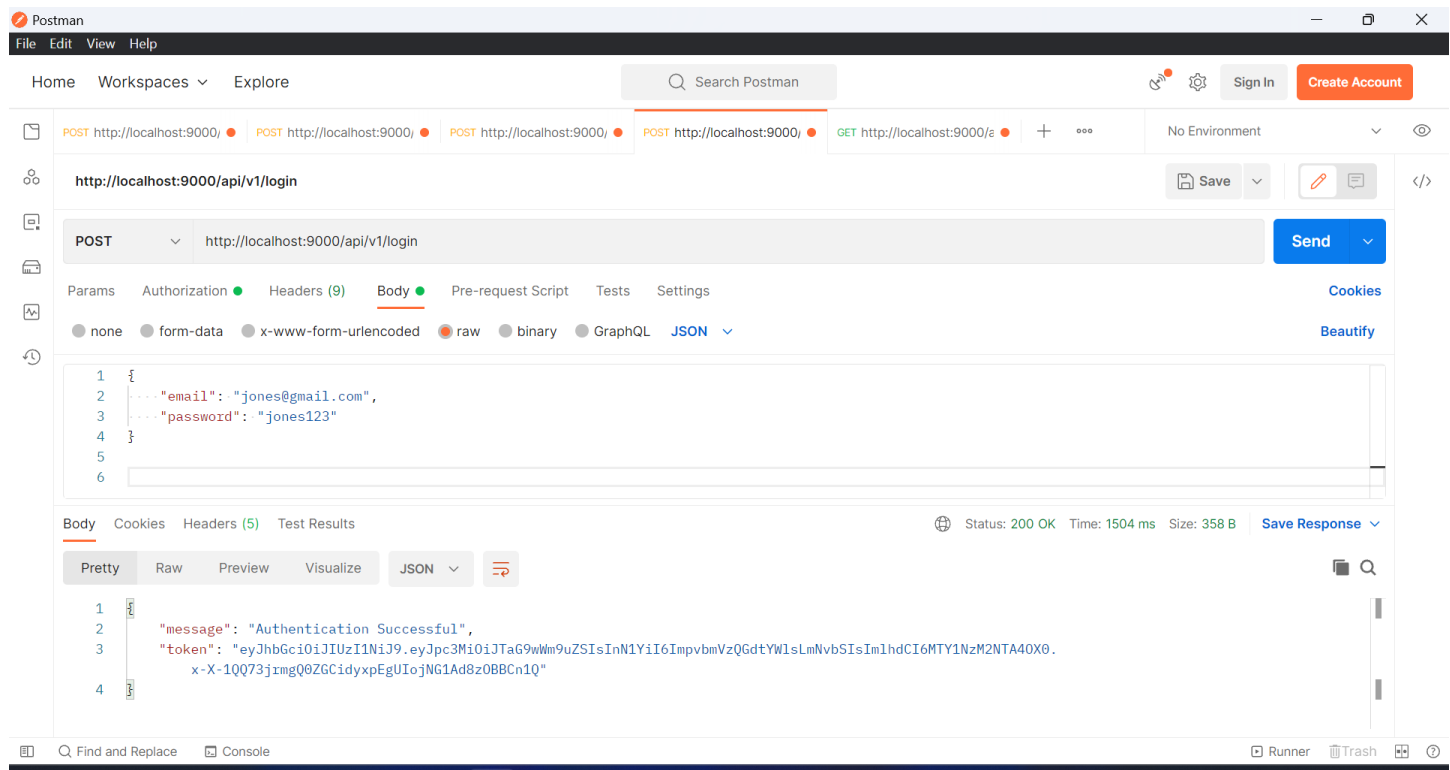
Pretty Raw Preview Visualize JSON

```
1 {
2   "email": "jones@gmail.com",
3   "password": "jones123"
4 }
```

Find and Replace Console Runner Trash

Login to the Movie Service;

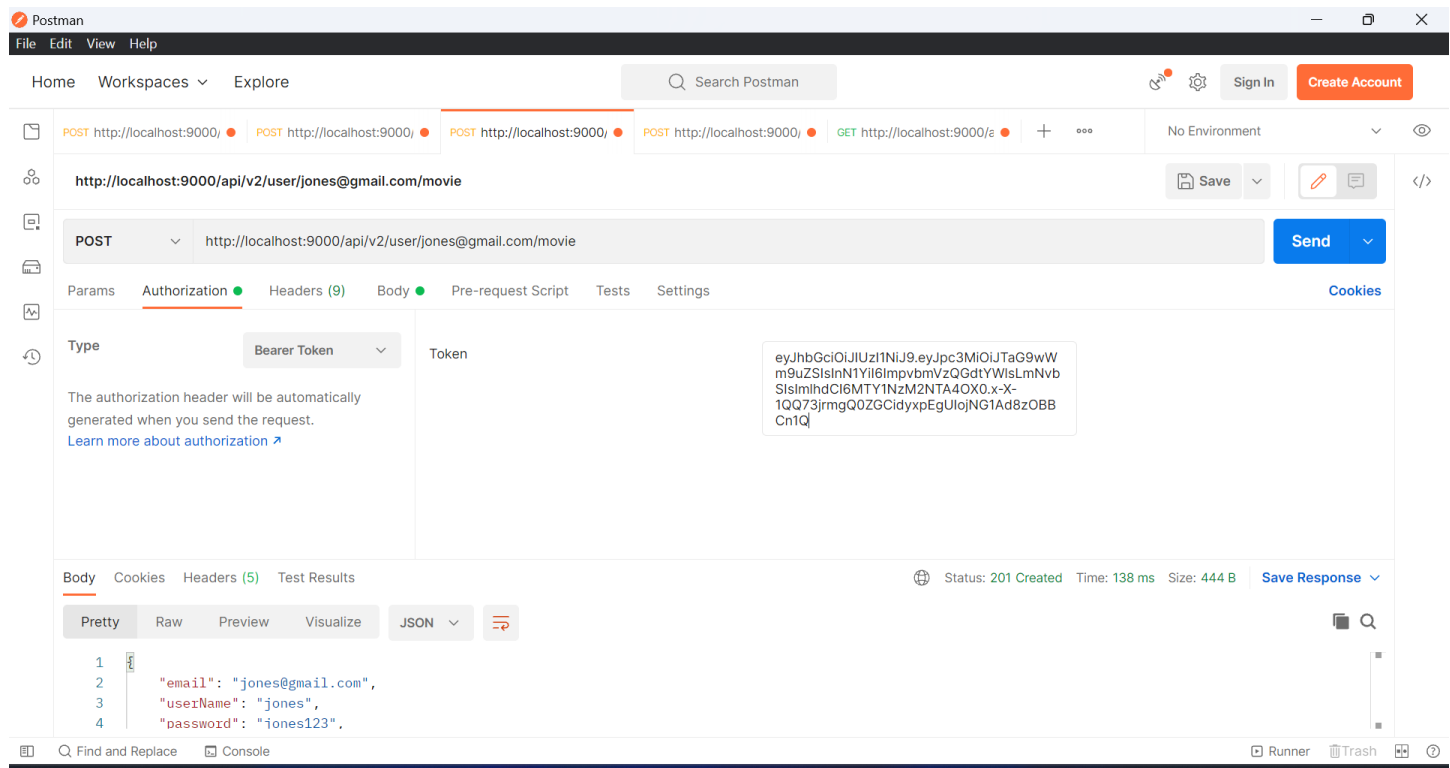
<http://localhost:9000/api/v1/login>



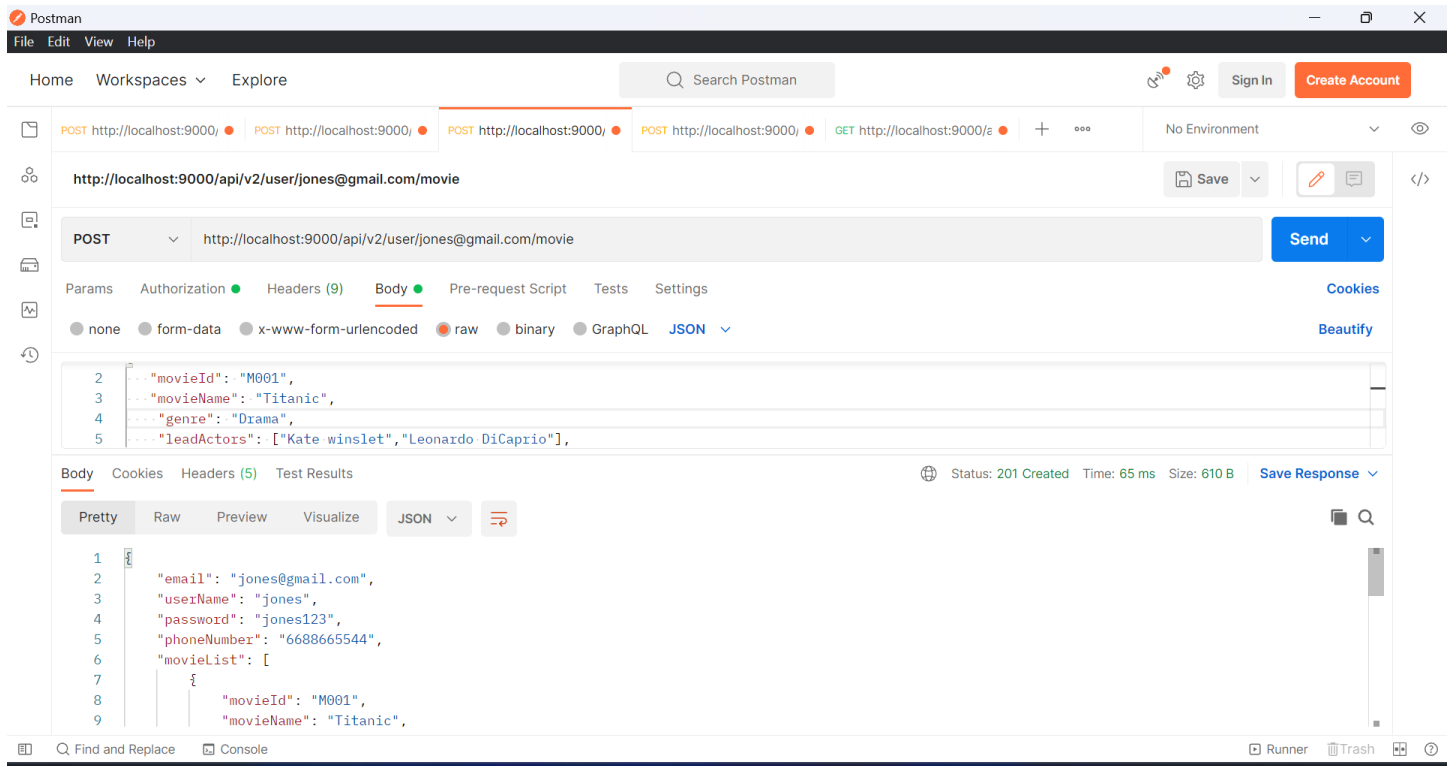
Add the favourite Movie for a user

<http://localhost:9000/api/v2/user/lucky@gmail.com/movie>

Add Authorization token



```
{  "movieId": "M001",  "movieName": "Titanic",  "genre": "Drama",  "leadActors": ["Kate winslet", "Leonardo DiCaprio"],  "director": "James Cameron",  "yearOfRelease": 1997,  "rating": 8}
```



Display all favorite movies list of specific user

<http://localhost:9000/api/v2/user/lucky@gmail.com/movies>

