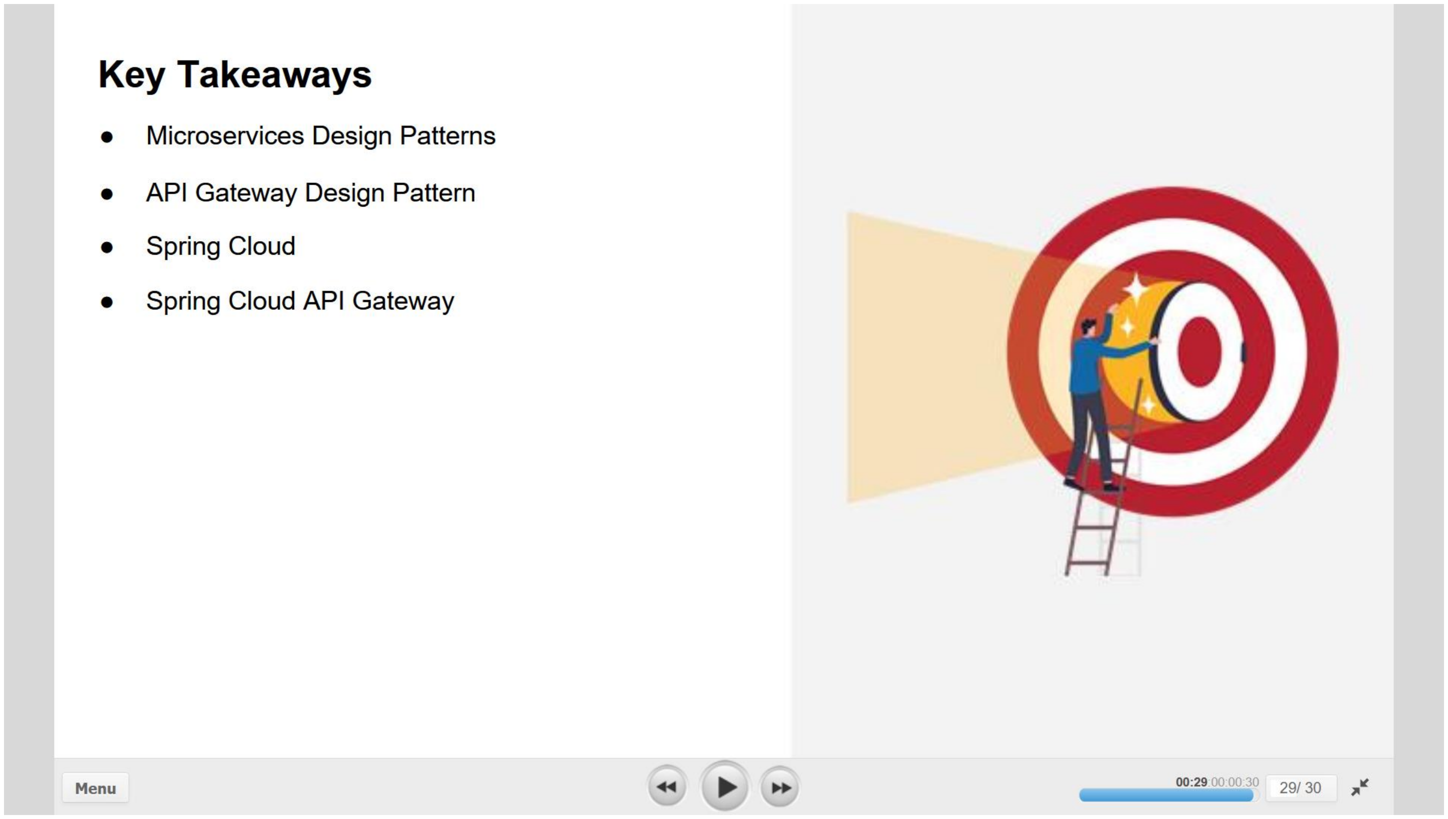
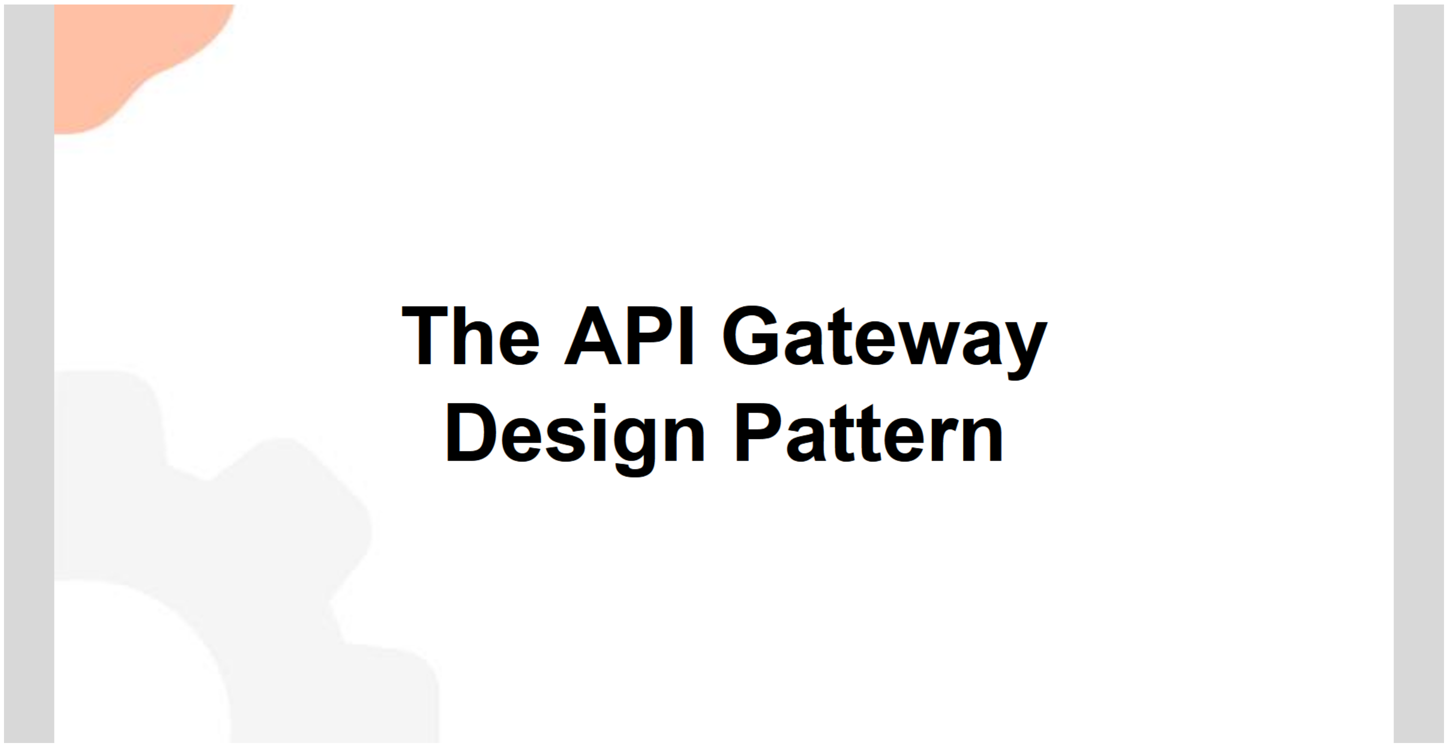


## Key Takeaways

- Microservices Design Patterns
- API Gateway Design Pattern
- Spring Cloud
- Spring Cloud API Gateway

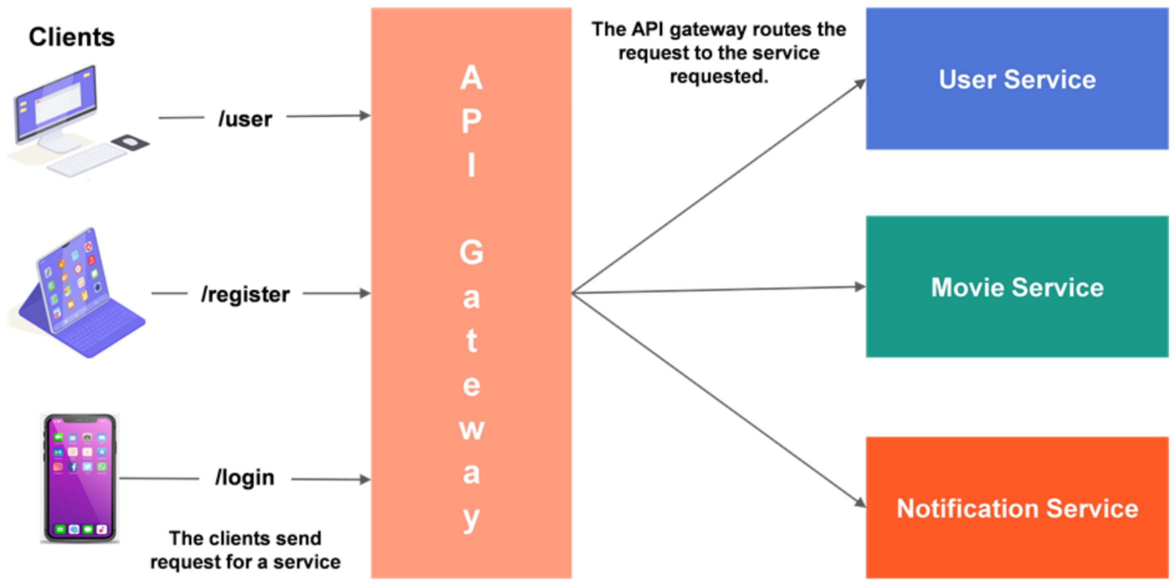


## The API Gateway Design Pattern

## API Gateway

- An API Gateway is a server that is the single-entry point into the system.
- It is a tool that sits between a client and a collection of backend services.
- An API gateway acts as a reverse proxy to:
  - Accept all application programming interface (API) calls
  - Aggregate the various services required to fulfill them
  - Return the appropriate result back to the client
- Most enterprise APIs are deployed via API Gateways.

## API Gateway


Clients
/user
/register
/login
The clients send request for a service
API Gateway
The API gateway routes the request to the service requested.
User Service
Movie Service
Notification Service

## Need for API Gateway

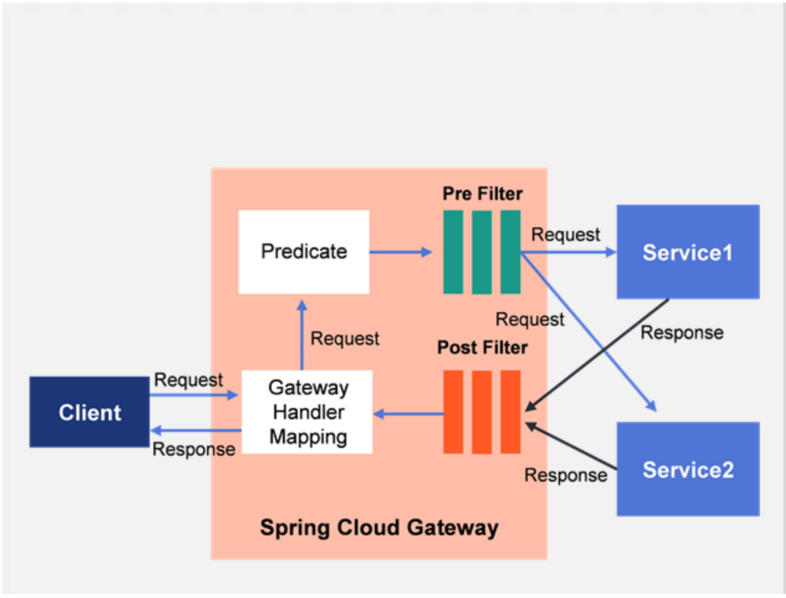
An API Gateway:

- Insulates the clients from how the application is partitioned into microservices.
- Insulates the clients from the problem of determining the locations of service instances.
- Provides the optimal API for each client.
- Reduces the number of requests/roundtrips.
- Translates from a "standard" public web-friendly API protocol to whichever protocols are used internally.

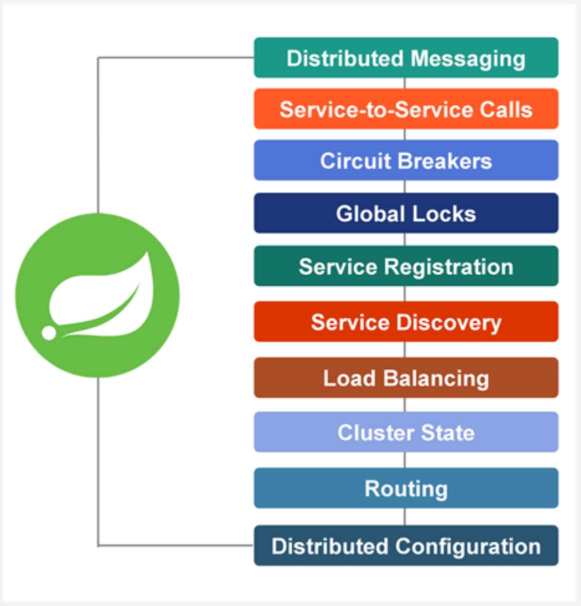
## Spring Cloud API Gateway Architecture

Spring Cloud API Gateway:

- Is built on top of the Spring ecosystem.
- Aims to provide a simple, yet effective way to route to the APIs.
- Consists of the following:
  - Route
  - Predicate
  - Filter



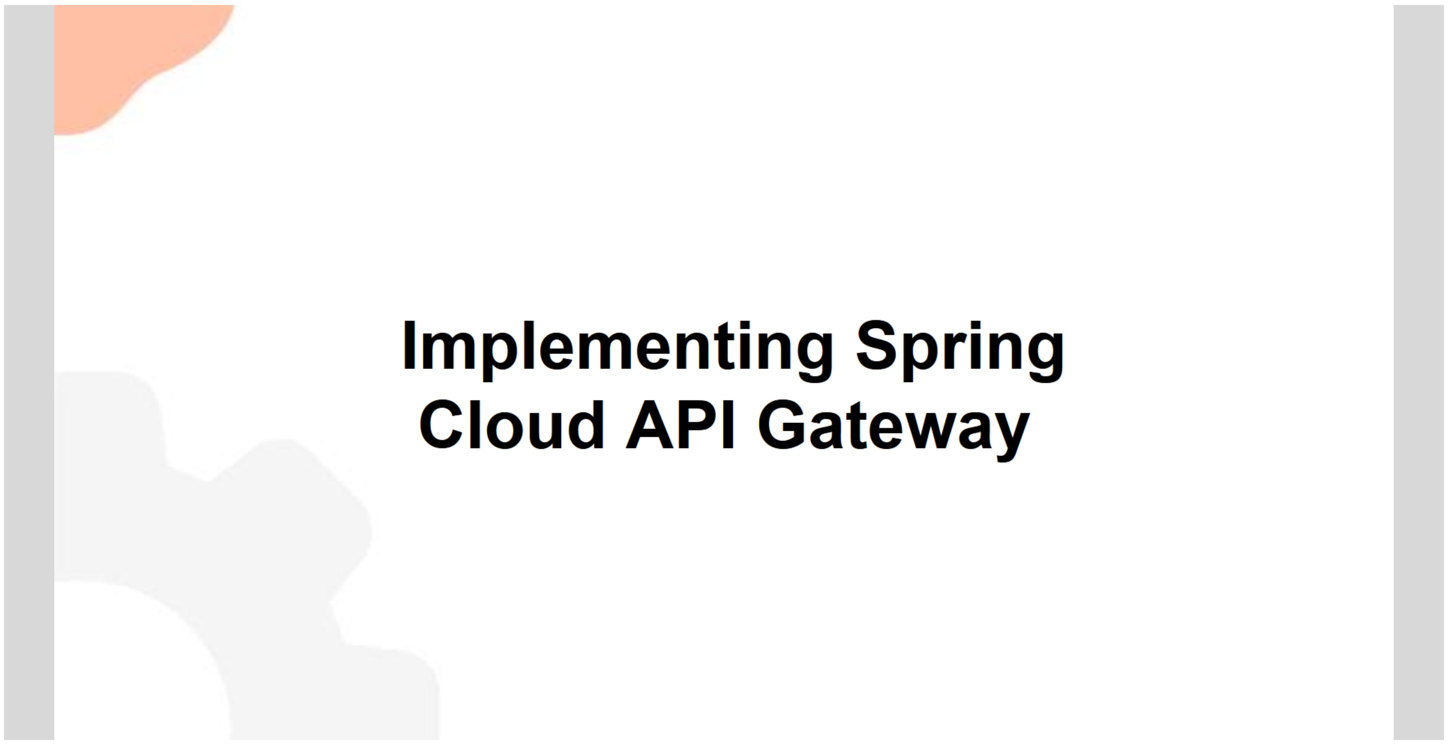
# Spring Cloud API Gateway



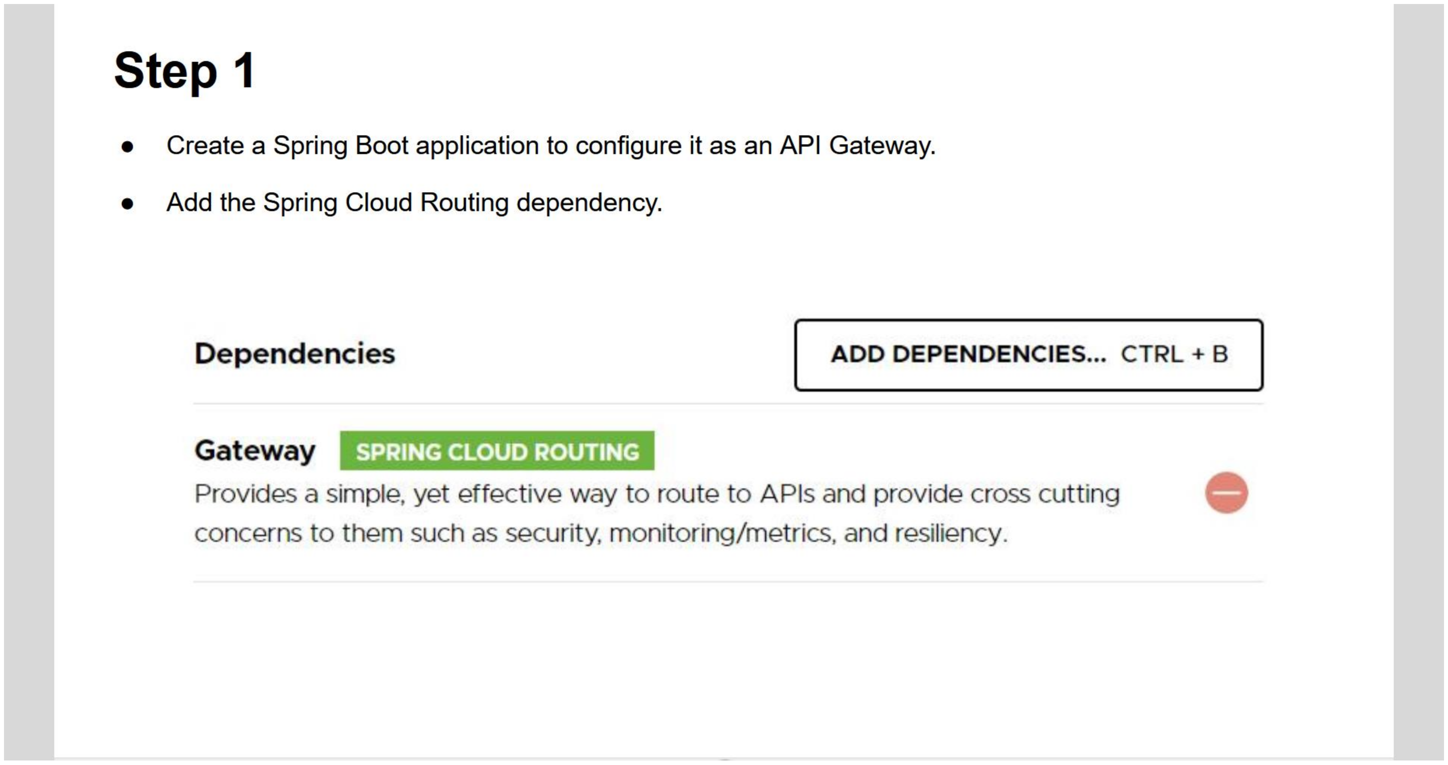
## Spring Cloud

Spring Cloud:

- Is an open-source library that makes it easy to develop applications for the cloud or a distributed environment.
- Provides tools for developers to quickly build some of the common patterns in the distributed systems involving microservices.
- Focuses on providing a good out-of-box experience for typical use cases and extensibility mechanism.



## Implementing Spring Cloud API Gateway



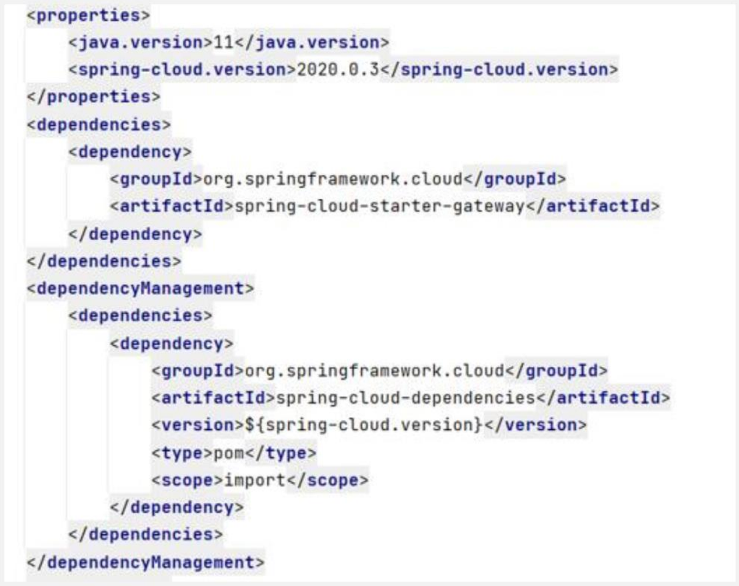
### Step 1

- Create a Spring Boot application to configure it as an API Gateway.
- Add the Spring Cloud Routing dependency.

**Dependencies** ADD DEPENDENCIES... CTRL + B

**Gateway** SPRING CLOUD ROUTING

Provides a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as security, monitoring/metrics, and resiliency.



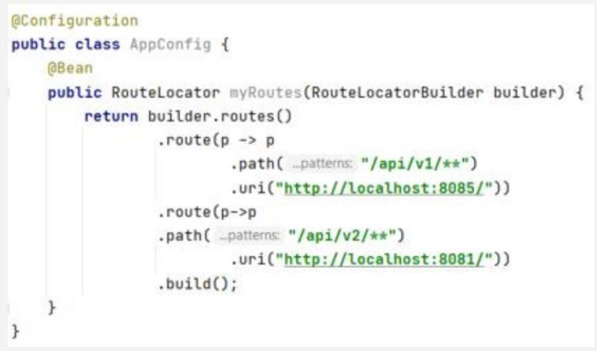
```xml
<properties>
    <java.version>11</java.version>
    <spring-cloud.version>2020.0.3</spring-cloud.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
</dependencies>
<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>
```

## pom.xml

- The spring cloud dependencies are added in the pom.xml file.
- The cloud dependencies of the version 2020.0.3 are added under the dependency management tag.

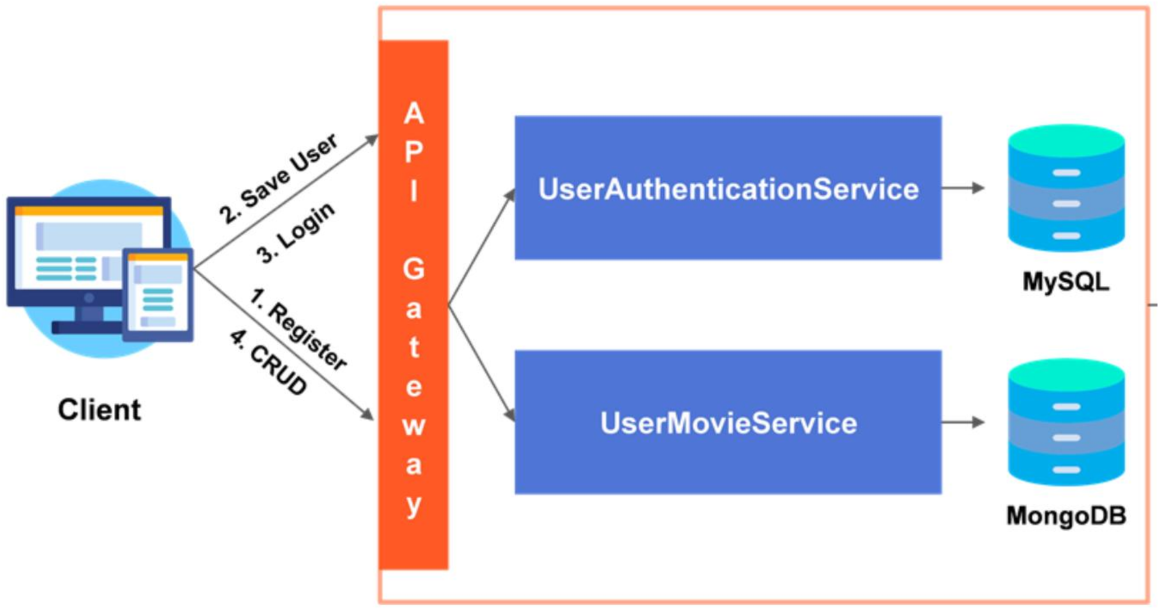
## Step 2 – Configure the Routes

- Create a Java class as a Configuration file for configuring the routes to the APIs in the application.
- Build the routes using the below classes:
  - `RouteLocator` – To obtain route information.
    - `path` – the rest end point patterns
    - `uri` – the uri at which the service is currently running
  - `RouteLocatorBuilder` – Used to create routes.

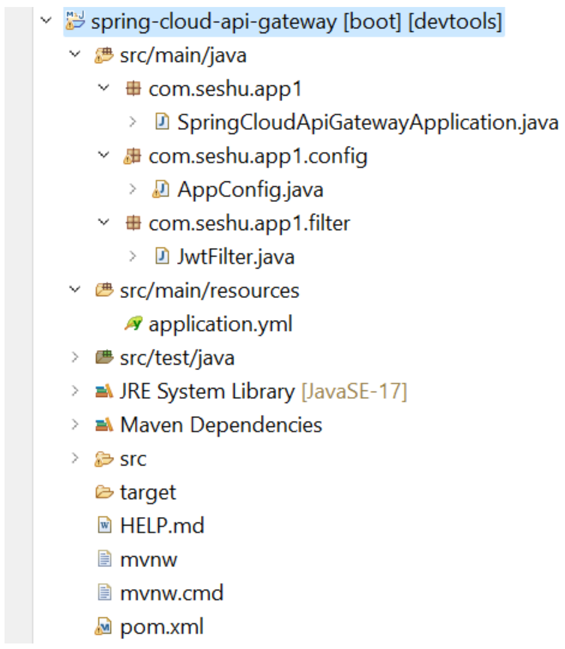


```java
@Configuration
public class AppConfig {
    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(p -> p
                        .path( …patterns: "/api/v1/**")
                        .uri("http://localhost:8085/"))
                .route(p->p
                .path( …patterns: "/api/v2/**")
                        .uri("http://localhost:8081/"))
                .build();
    }
}
```

## How Does The Application Work?



Example:



```
spring-cloud-api-gateway [boot] [devtools]
  src/main/java
    com.seshu.app1
      SpringCloudApiGatewayApplication.java
    com.seshu.app1.config
      AppConfig.java
    com.seshu.app1.filter
      JwtFilter.java
  src/main/resources
    application.yml
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Develop a new spring starter project as ***spring-cloud-api-gateway*** with fallowing dependency.

**Gateway**

**Spring web**

**Devtools**

**Update pom.xml file with fallowing dependency;**

```xml
		<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>
```

pom.xml

```
<?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>
	</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
```

AppConfig.java

```
package com.seshu.app1.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.app1.filter.JwtFilter;

@Configuration
public class AppConfig {
    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(p -> p
                        .path("/api/v1/**")
                        .uri("http://localhost:8085/"))
                .route(p->p
                .path("/api/v2/**")
                        .uri("http://localhost:8081/"))
                .build();
    }
    @Bean
    public FilterRegistrationBean jwtFilterBean(){
        FilterRegistrationBean filterRegistrationBean = new
FilterRegistrationBean();
        filterRegistrationBean.setFilter(new JwtFilter());
        filterRegistrationBean.addUrlPatterns("/api/v2/user/*");
        return filterRegistrationBean;
    }
}
```

JwtFilter.java

```
package com.seshu.app1.filter;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import org.springframework.web.filter.GenericFilterBean;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class JwtFilter extends GenericFilterBean {
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain) throws IOException,
ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpServletResponse response = (HttpServletResponse)
servletResponse;

        //expects the token to come from the header
        final String authHeader = request.getHeader("Authorization");
        if(request.getMethod().equals("OPTIONS")){
            //if the method is options the request can pass through not
validation of token is required
            response.setStatus(HttpServletResponse.SC_OK);
            filterChain.doFilter(request,response);
        }
        else if(authHeader == null || !authHeader.startsWith("Bearer "))
        {
            throw new ServletException("Missing or Invalid Token");
        }
        //extract token from the header
        String token = authHeader.substring(7);//Bearer => 6+1 = 7, since
token begins with Bearer

        //token validation
        Claims claims =
Jwts.parser().setSigningKey("mysecret").parseClaimsJws(token).getBody();
        request.setAttribute("claims",claims);

        //pass the claims in the request, anyone wanting to

        filterChain.doFilter(request,response);
```

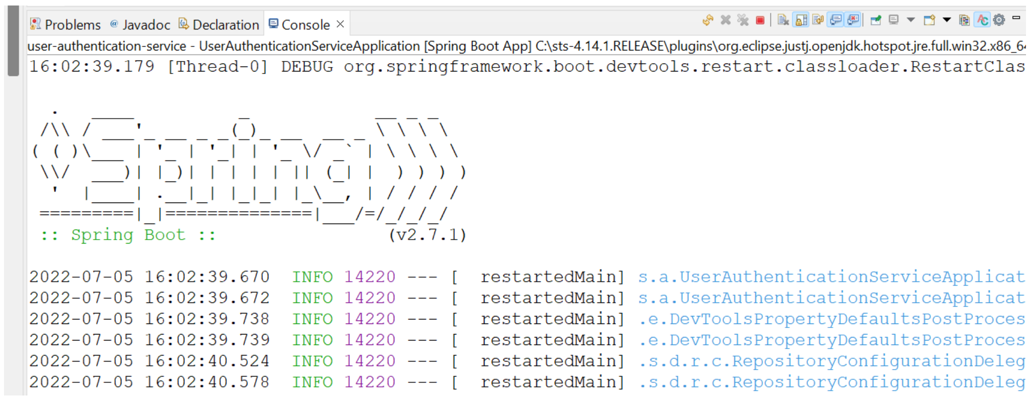
```
     }
}
```

SpringCloudApiGatewayApplication.java

```
package com.seshu.app1;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringCloudApiGatewayApplication {

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

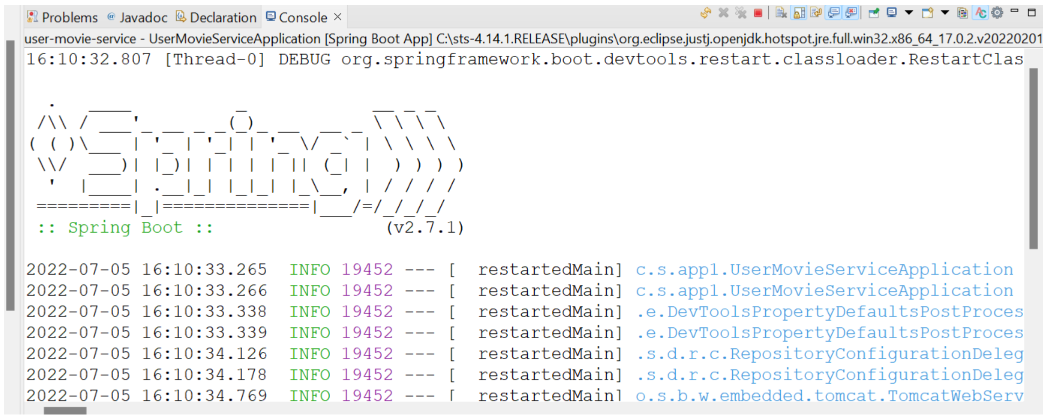
## Execution steps:

### First, run user-authentication-service starter



```
user-authentication-service - UserAuthenticationServiceApplication [Spring Boot App] C:\sts-4.14.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
16:02:39.179 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClas

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.1)

2022-07-05 16:02:39.670  INFO 14220 --- [  restartedMain] s.a.UserAuthenticationServiceApplicat
2022-07-05 16:02:39.672  INFO 14220 --- [  restartedMain] s.a.UserAuthenticationServiceApplicat
2022-07-05 16:02:39.738  INFO 14220 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProces
2022-07-05 16:02:39.739  INFO 14220 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProces
2022-07-05 16:02:40.524  INFO 14220 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDeleg
2022-07-05 16:02:40.578  INFO 14220 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDeleg
```

### Second, run user-movie-service starter.



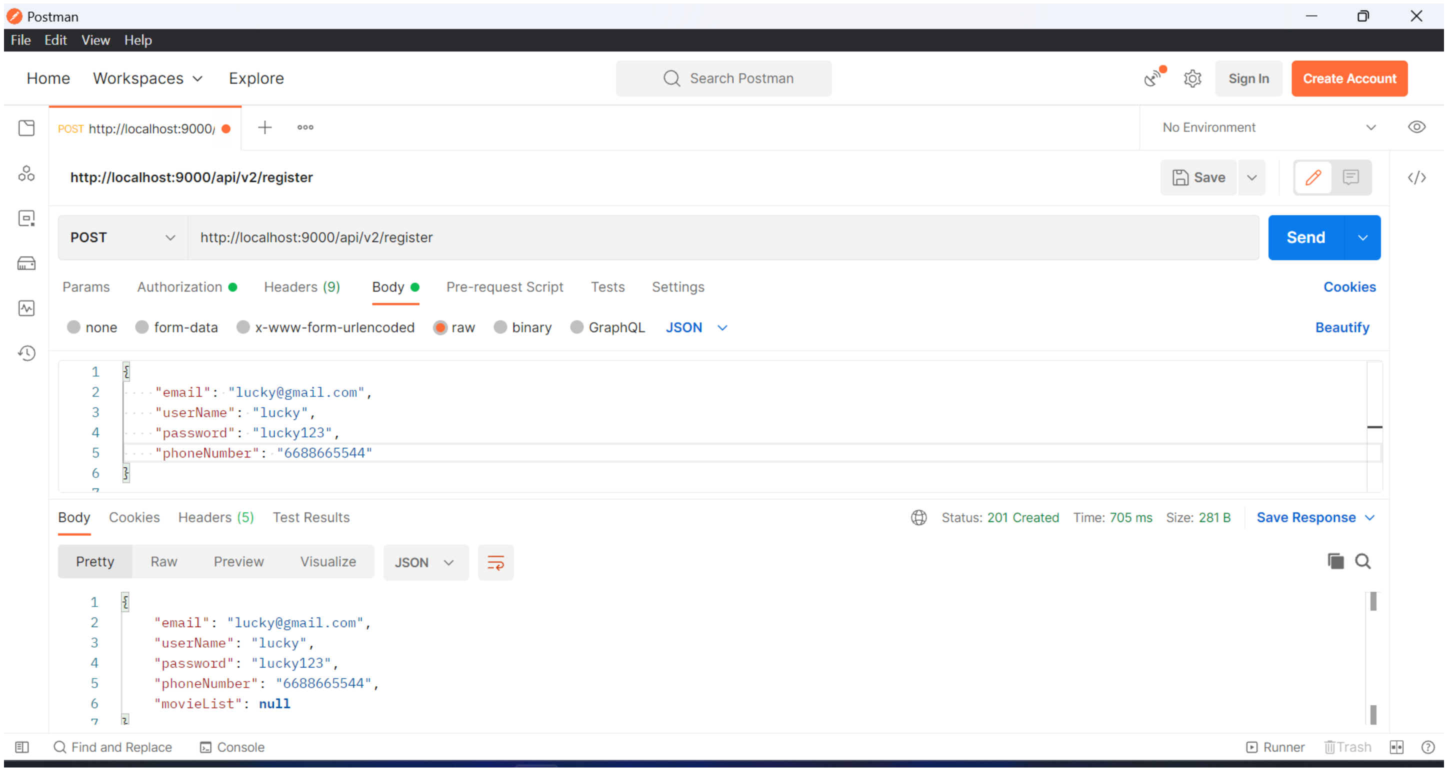
```
user-movie-service - UserMovieServiceApplication [Spring Boot App] C:\sts-4.14.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201
16:10:32.807 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClas

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.1)

2022-07-05 16:10:33.265  INFO 19452 --- [  restartedMain] c.s.app1.UserMovieServiceApplication
2022-07-05 16:10:33.266  INFO 19452 --- [  restartedMain] c.s.app1.UserMovieServiceApplication
2022-07-05 16:10:33.338  INFO 19452 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProces
2022-07-05 16:10:33.339  INFO 19452 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProces
2022-07-05 16:10:34.126  INFO 19452 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDeleg
2022-07-05 16:10:34.178  INFO 19452 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDeleg
2022-07-05 16:10:34.769  INFO 19452 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServ
```

### Third, run spring-cloud-api-gateway

## Test the application using Postman client tool

**Register a new user;**

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


Postman
File Edit View Help
Home Workspaces Explore
Search Postman
Sign In Create Account
No Environment
Save
Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console Runner Trash

POST http://localhost:9000/api/v2/register

```
{
    "email": "lucky@gmail.com",
    "userName": "lucky",
    "password": "lucky123",
    "phoneNumber": "6688665544"
}
```

Status: 201 Created Time: 705 ms Size: 281 B

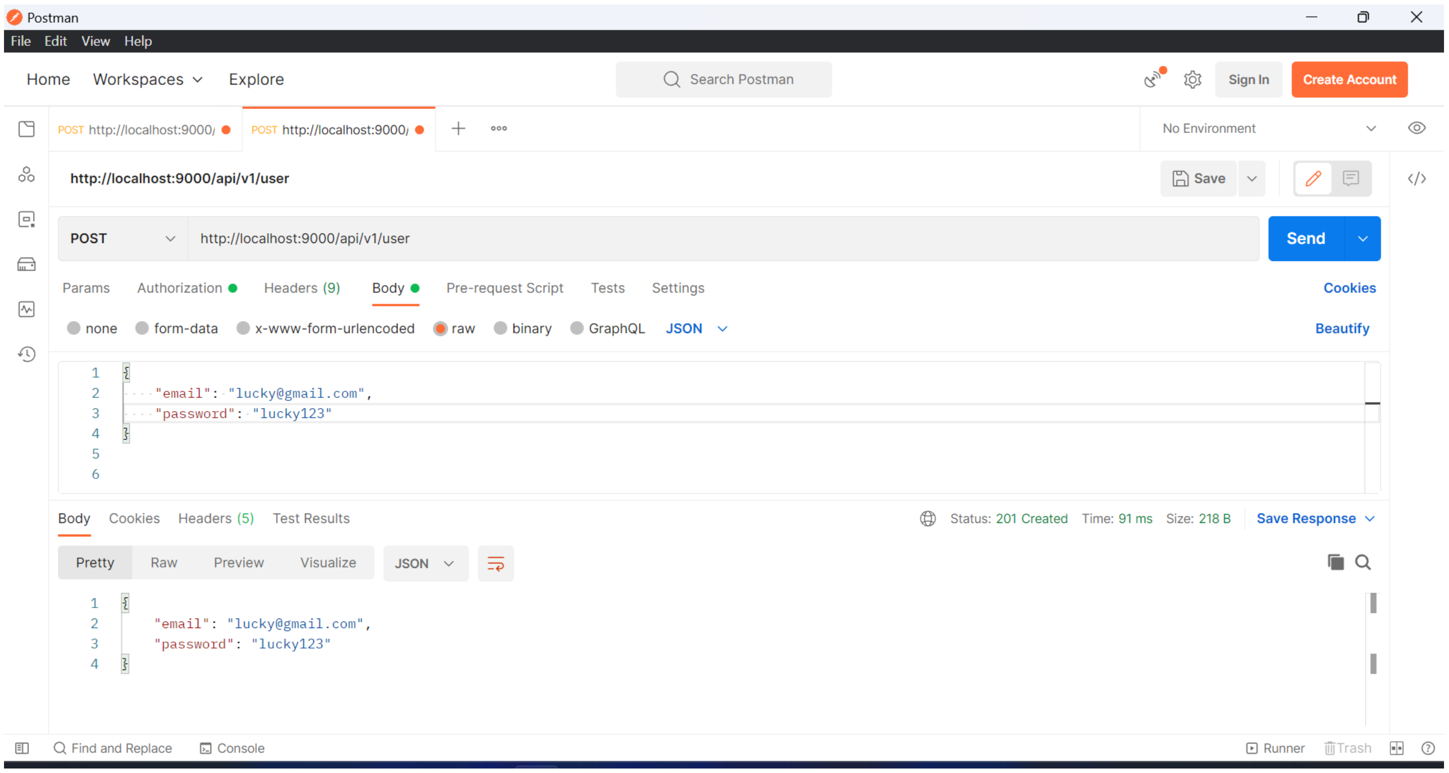
```
{
    "email": "lucky@gmail.com",
    "userName": "lucky",
    "password": "lucky123",
    "phoneNumber": "6688665544",
    "movieList": null
}
```

**Note:**

- **UserAuthenticationService** is running on port **8081** and **UserMovieService** is running on port **8085**, but as we can see here the request from the client is not routed directly to those services.
- The API Gateway intercepts the request and passes the request to the service.
- The client is not aware of the details of the service like path, URI, etc.

## Save user in the Movie Service;

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



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

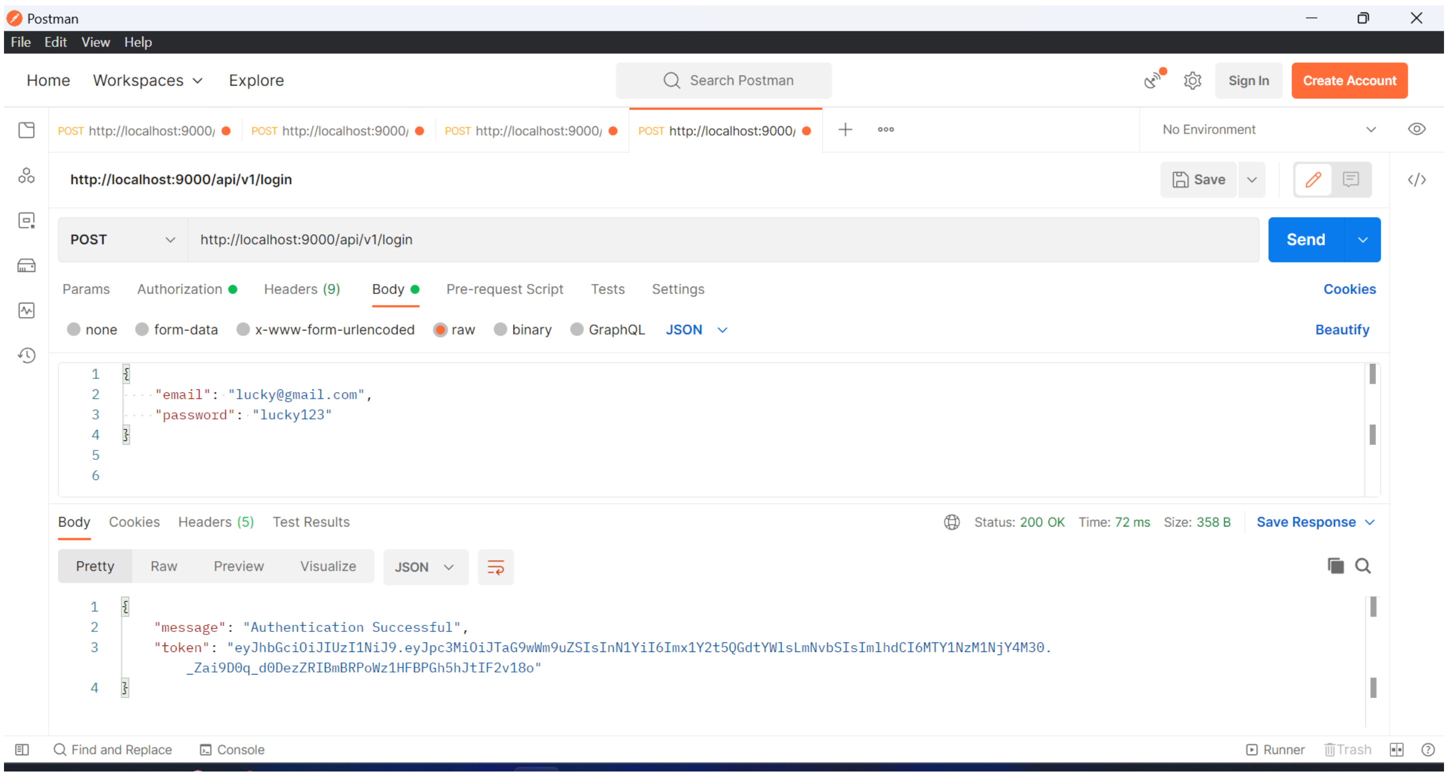
```
{
    "email": "lucky@gmail.com",
    "password": "lucky123"
}
```

Status: 201 Created Time: 91 ms Size: 218 B

```
{
    "email": "lucky@gmail.com",
    "password": "lucky123"
}
```

## Login to the Movie Service;

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


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

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

```
{
    "email": "lucky@gmail.com",
    "password": "lucky123"
}
```

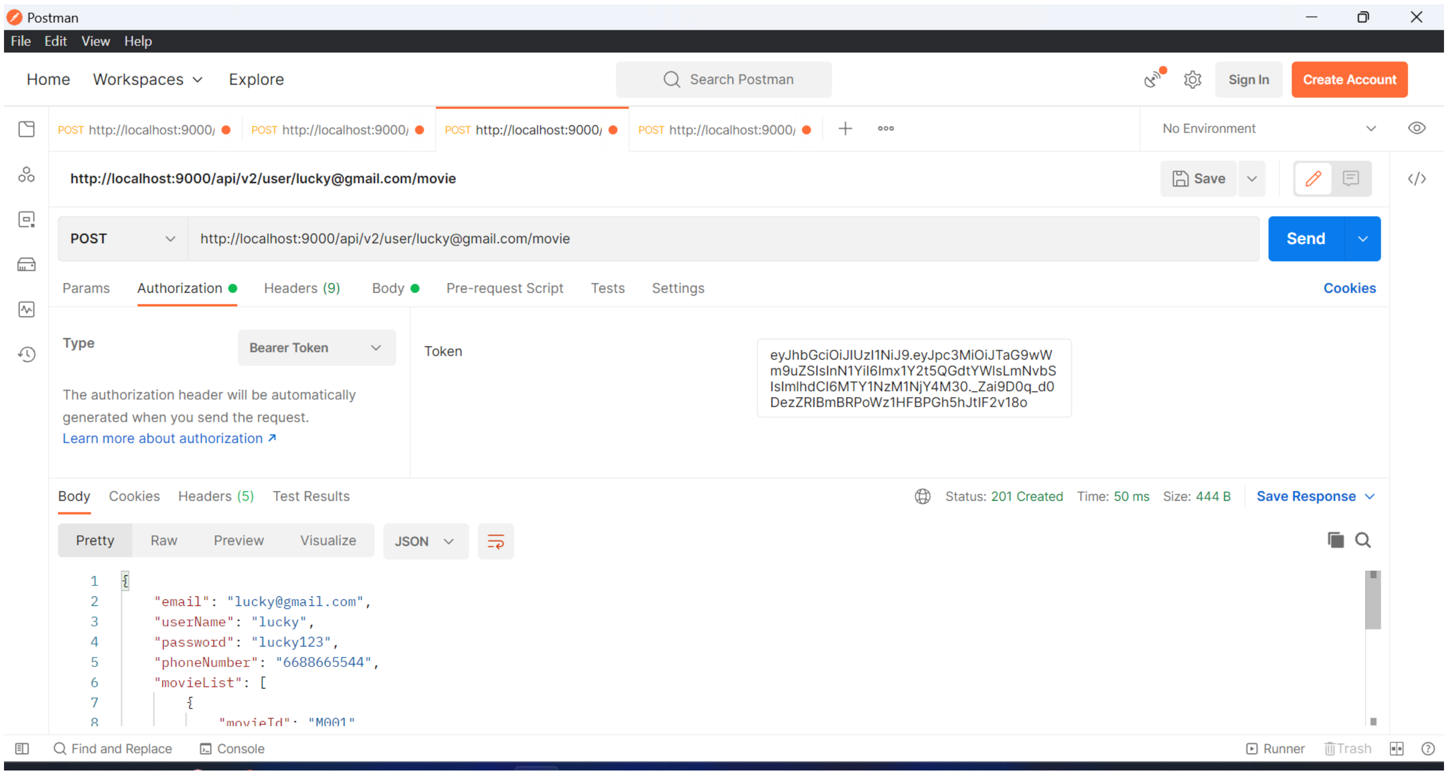
Status: 200 OK Time: 72 ms Size: 358 B

```
{
    "message": "Authentication Successful",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJTaG9wWm9uZSIsInN1YiI6Imx1Y2t5QGdtYWlsLmNvbSIsImlhdCI6MTY1NzM1NjY4M30.
        _Zai9D0q_d0DezZRIBmBRPoWz1HFBPGh5hJtIF2v18o"
}
```

## Add the favourite Movie for a user

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

Add Authorization token



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

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

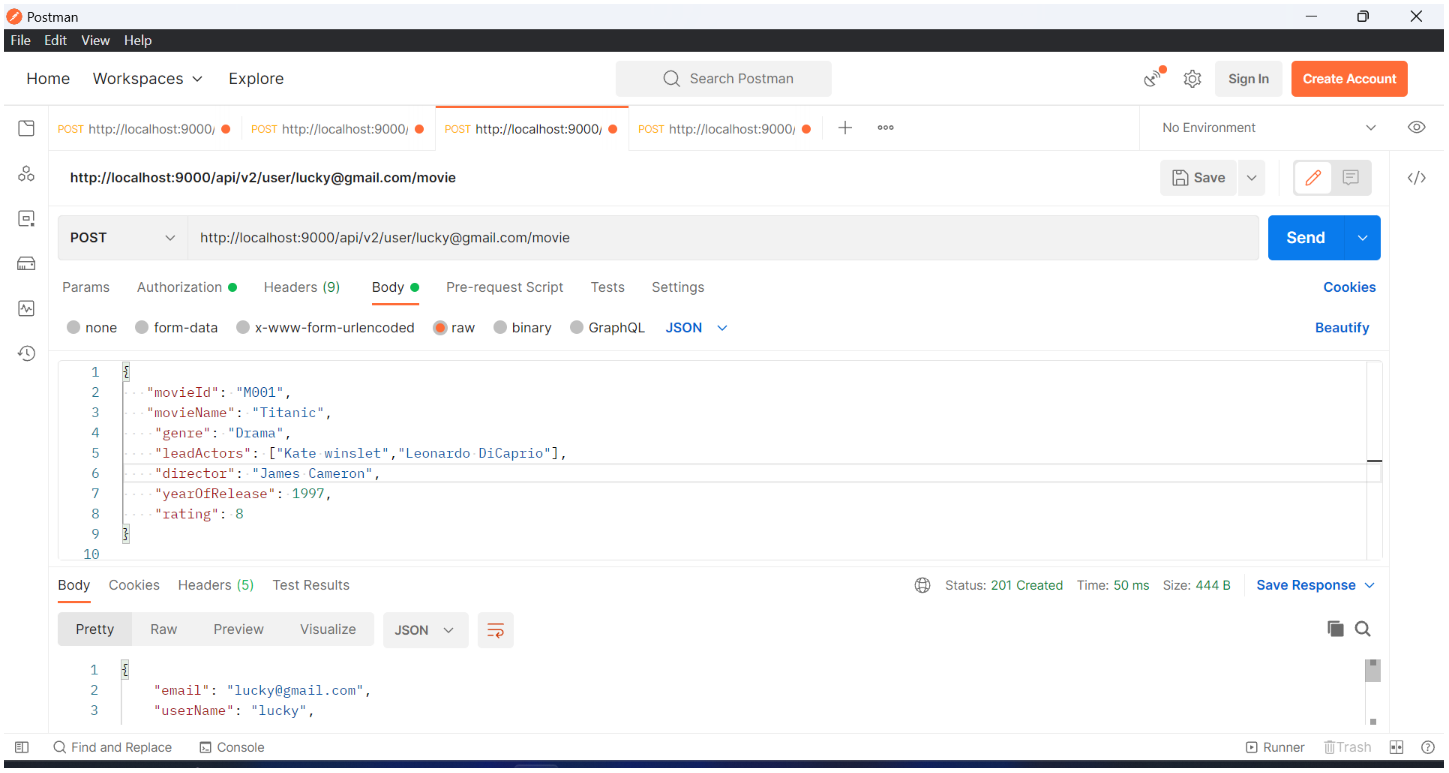
Type: Bearer Token

The authorization header will be automatically generated when you send the request.
Learn more about authorization ↗

Token: eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJTaG9wW m9uZSIsInN1YiI6Imx1Y2t5QGdtYWlsLmNvbS IsImlhdCI6MTY1NzM1NjY4M30._Zai9D0q_d0 DezZRlBmBRPoWz1HFBPGh5hJtIF2v18o

Status: 201 Created Time: 50 ms Size: 444 B

```
{
    "email": "lucky@gmail.com",
    "userName": "lucky",
    "password": "lucky123",
    "phoneNumber": "6688665544",
    "movieList": [
        {
            "movieId": "M001",
```


Postman
File Edit View Help
Home Workspaces Explore
Search Postman
Sign In
Create Account
No Environment
Save
Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console
Runner Trash

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

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

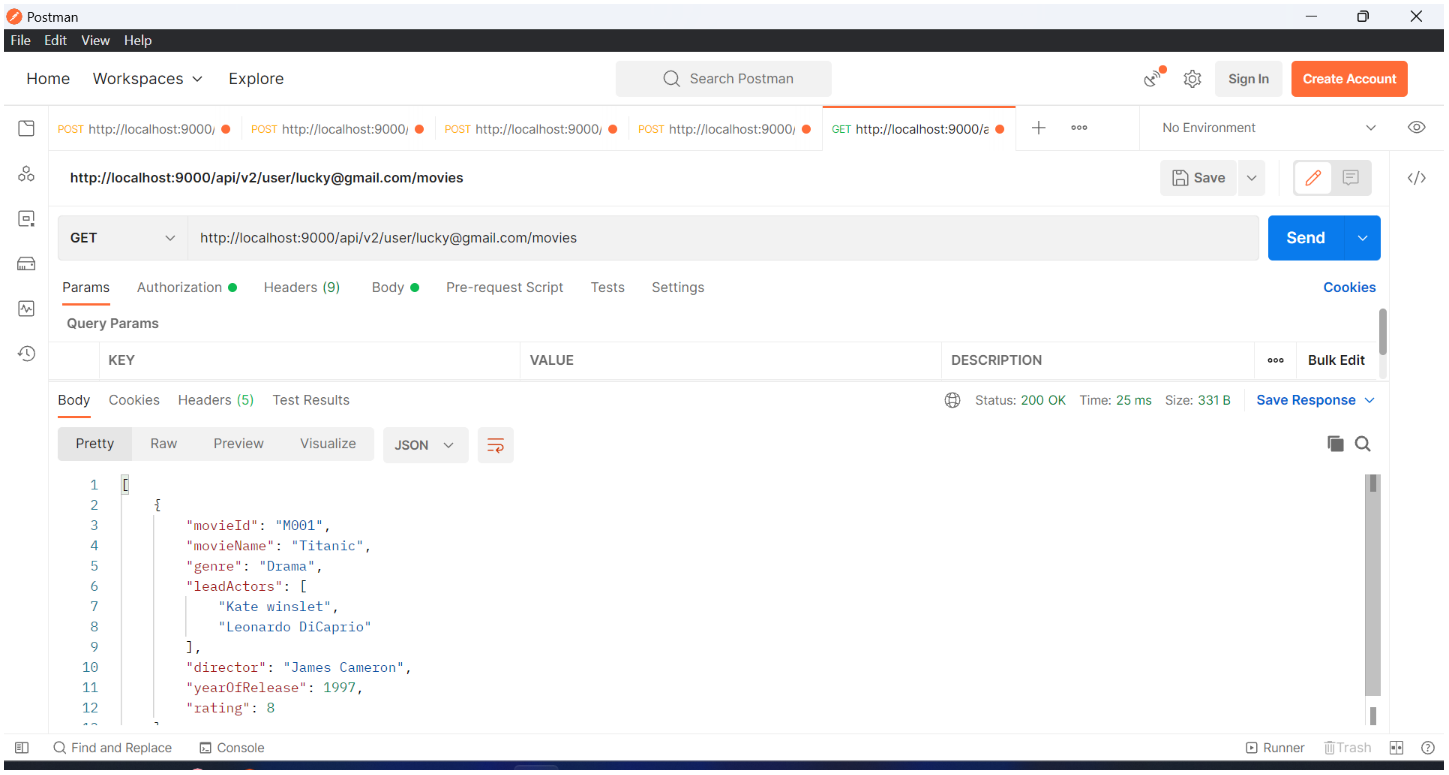
```
{
  "movieId": "M001",
  "movieName": "Titanic",
   "genre": "Drama",
   "leadActors": ["Kate winslet","Leonardo DiCaprio"],
   "director": "James Cameron",
   "yearOfRelease": 1997,
   "rating": 8
}
```

Status: 201 Created Time: 50 ms Size: 444 B

```
{
    "email": "lucky@gmail.com",
    "userName": "lucky",
```

## Display all favorite movies list of specific user

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


Postman
File Edit View Help
Home Workspaces Explore
Search Postman
Sign In
Create Account
No Environment
Save
Send
Params Authorization Headers (9) Body Pre-request Script Tests Settings
Cookies
Query Params
KEY VALUE DESCRIPTION Bulk Edit
Body Cookies Headers (5) Test Results
Save Response
Pretty Raw Preview Visualize JSON
Find and Replace Console Runner Trash

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

Status: 200 OK Time: 25 ms Size: 331 B

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