Backend Program: Course 3: Structure

Program

Build job-ready skills

Enterprise Application Development By Using Spring Framework

Courses

Build competencies

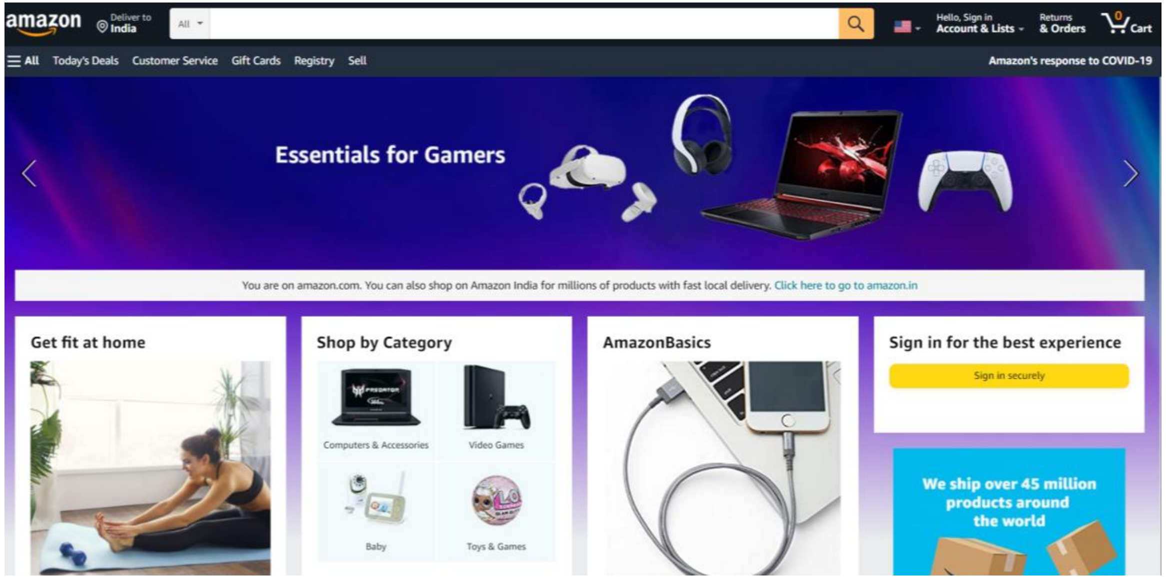
- CRS1 : Spring Framework Foundation
- CRS 2 : Creating Restful Services Using SpringBoot
- **CRS 3 : Creating and Managing Microservices**
- CRS 4 : Establishing Communication Among Microservices

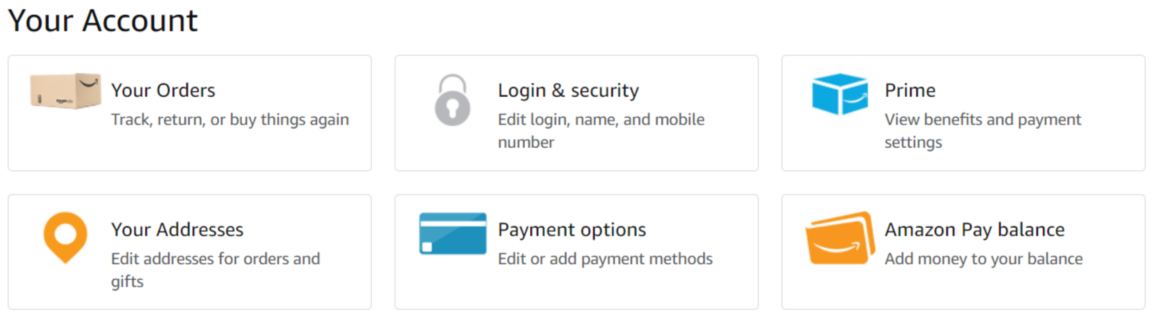
Learning Sprints

Perform specific tasks

- Work with Virtual Machine
- Enable Continuous Integration within a Backend Application to Support a Collaborative Development Environment
- Containerize RESTful services and Database by Using Docker
- Containerize RESTful Services and Database by using Docker Compose
- Create Microservices by Using Spring Boot
- **Create a Single-Entry Point to Route the Request Coming for Different Microservices Using Spring Cloud**
- Register Microservices on a Netflix Eureka Discovery Server
- Handle the Failure of Microservices by Implementing the Circuit Breaker Pattern Using Hystrix

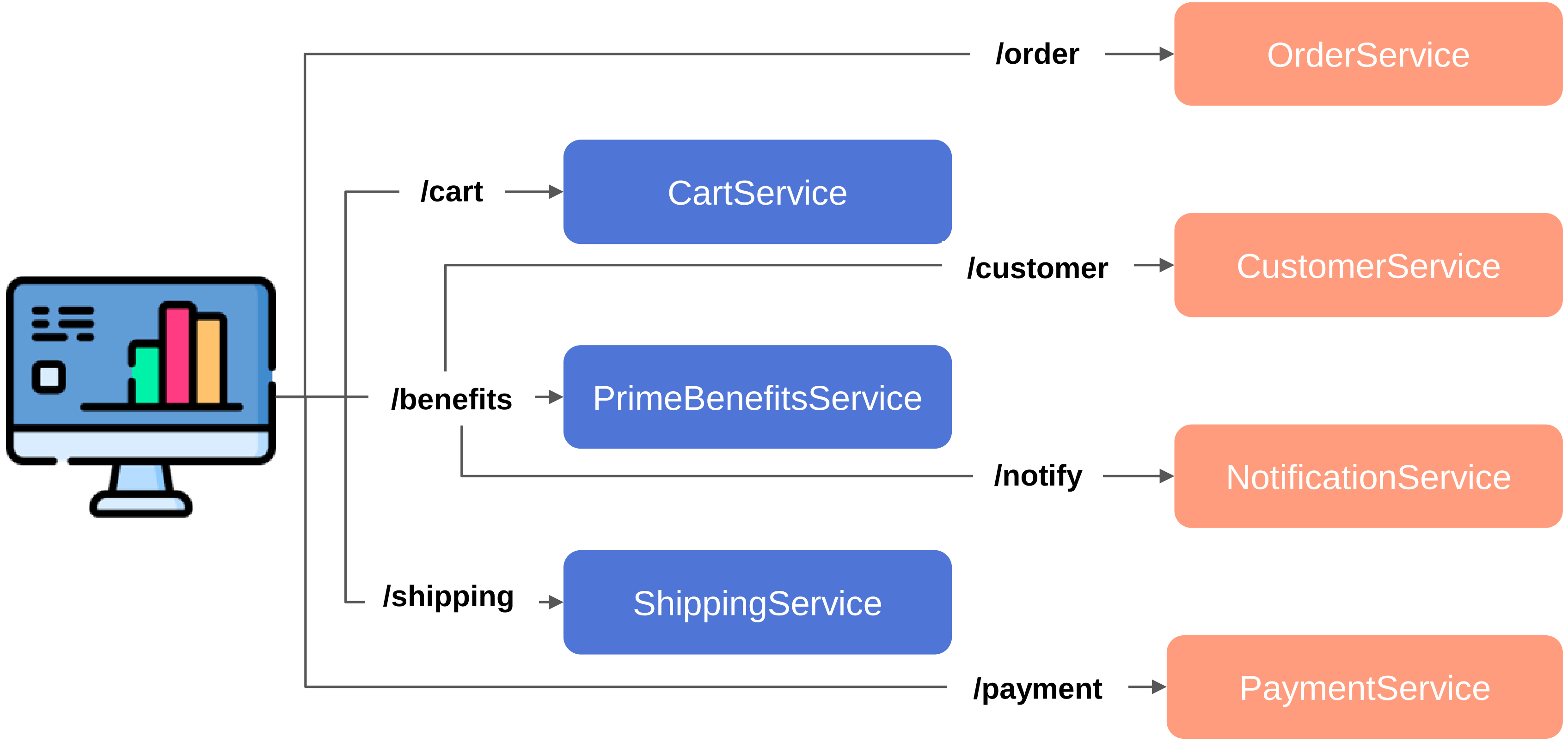
Amazon


amazon
Deliver to India
All
Hello, Sign in Account & Lists
Returns & Orders
0 Cart
All
Today's Deals
Customer Service
Gift Cards
Registry
Sell
Amazon's response to COVID-19
Essentials for Gamers
You are on amazon.com. You can also shop on Amazon India for millions of products with fast local delivery. Click here to go to amazon.in
Get fit at home
Shop by Category
Computers & Accessories
Video Games
Baby
Toys & Games
AmazonBasics
Sign in for the best experience
Sign in securely
We ship over 45 million products around the world


Your Account

Your Orders	Track, return, or buy things again
Login & security	Edit login, name, and mobile number
Prime	View benefits and payment settings
Your Addresses	Edit addresses for orders and gifts
Payment options	Edit or add payment methods
Amazon Pay balance	Add money to your balance

Amazon Workflow – Multiple Services



Think and Tell

- In an application with multiple microservices, how can the client know which service to call?
- Should the client know all the paths to the services? Is this a safe approach?
- The service name, the port number on which the service runs should all information be given to the client, is this a secure way?
- If a port number to a service changes how will the client know about the change in port?

Think and Tell

- If a new service is added how will the client know that there is a new service?
- If multiple services have common cross-cutting functionality can this be grouped in a common service?
- Do we need to create a common service for this purpose?

Create a Single-Entry Point to Route the Request Coming for Different Microservices Using Spring Cloud

Learning Objectives

- Explore the Microservices Design Pattern
- Define the API Gateway Pattern
- Implement the API Gateway using Spring Cloud

Microservices Design Patterns

Microservices Design Patterns

- Microservices design patterns are software design patterns that generate reusable autonomous services.
- The goal for developers using microservices is to accelerate application releases.
- By using microservices, developers can deploy each individual microservice independently, if desired.
- The design pattern helps developers with certain principles at the time of developing individual microservices.

Common Microservices Design Patterns

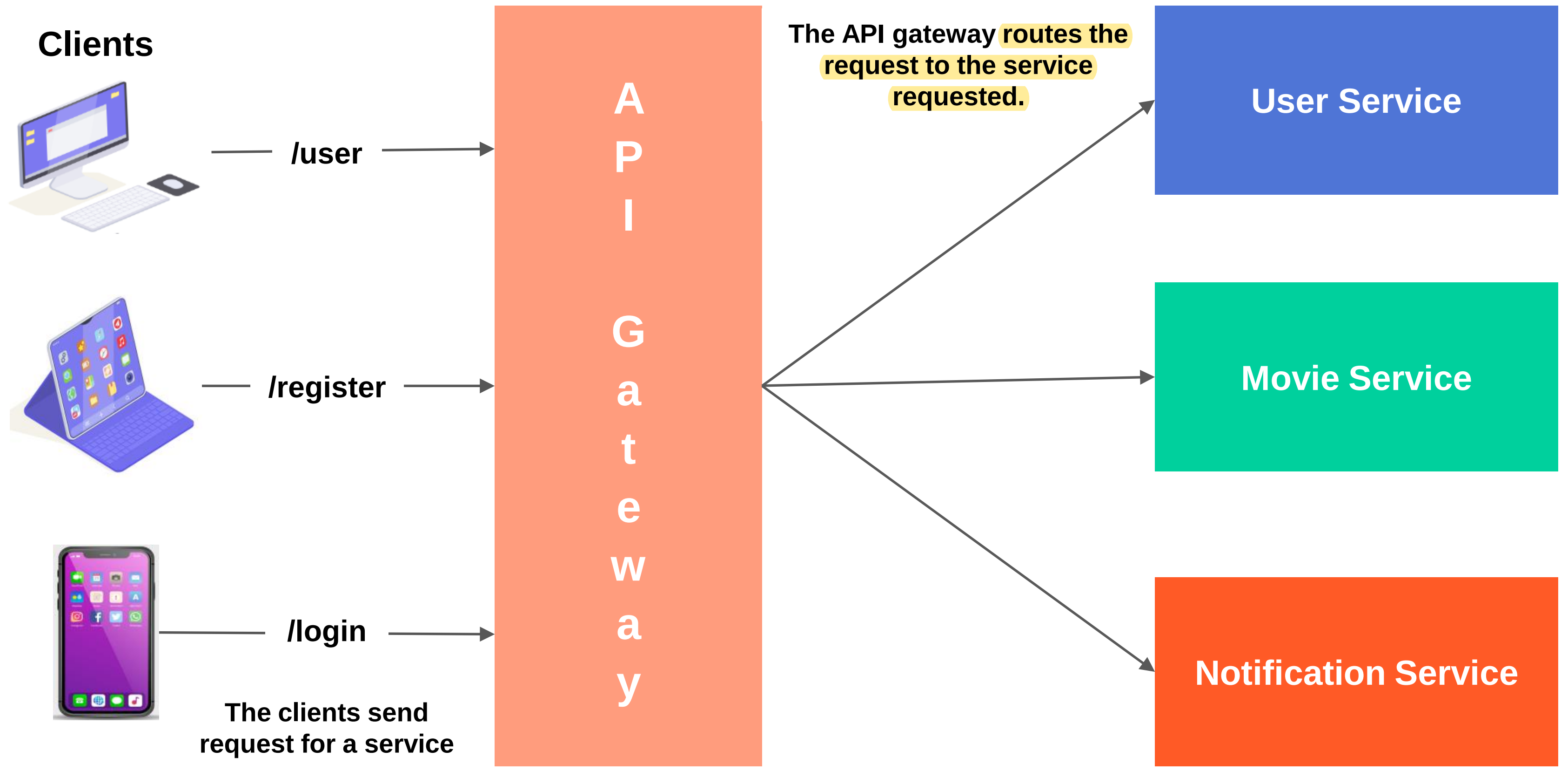
- API Gateway Pattern - The API Gateway pattern defines how clients access the services in a microservice architecture.
- Service Discovery Pattern - The Service Discovery patterns are used to route requests for a client to an available service instance in a microservice architecture.
- Circuit Breaker Pattern - This Circuit Breaker Pattern helps handle the failure of the services invoked.

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

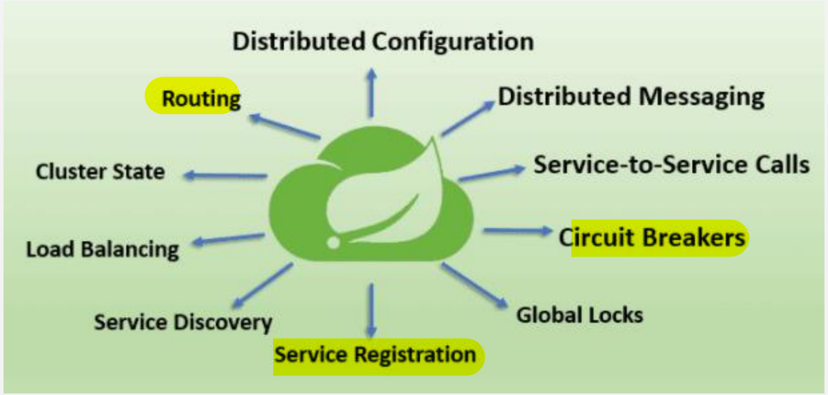

Clients
/user
/register
/login
The clients send request for a service
A P I Gateway
The API gateway routes the request to the service requested.
User Service
Movie Service
Notification Service

Need for 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 whatever protocols are used internally.

Spring Cloud API Gateway

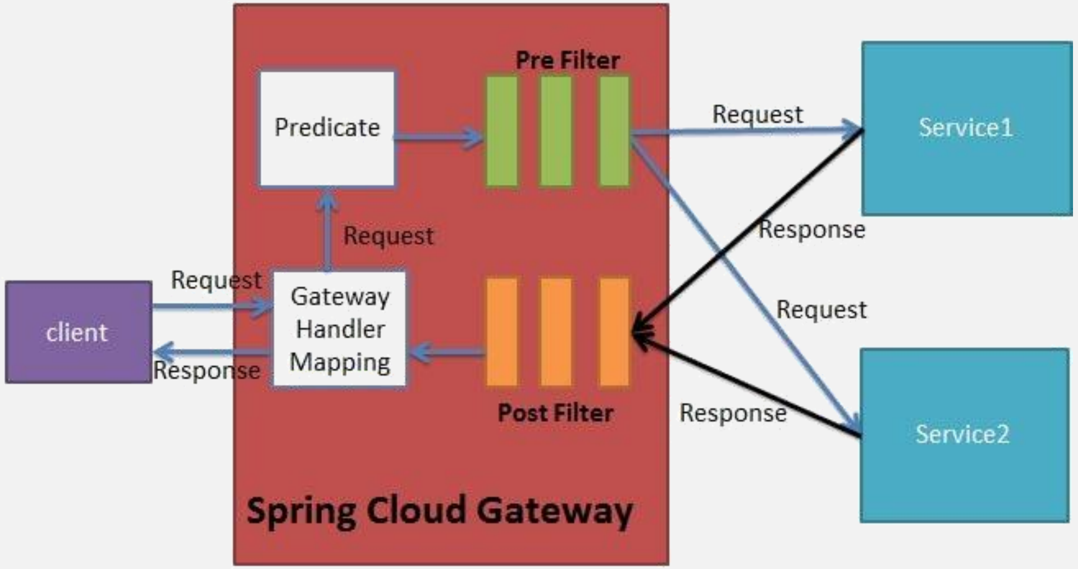
Spring Cloud



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

Spring Cloud API Gateway Architecture

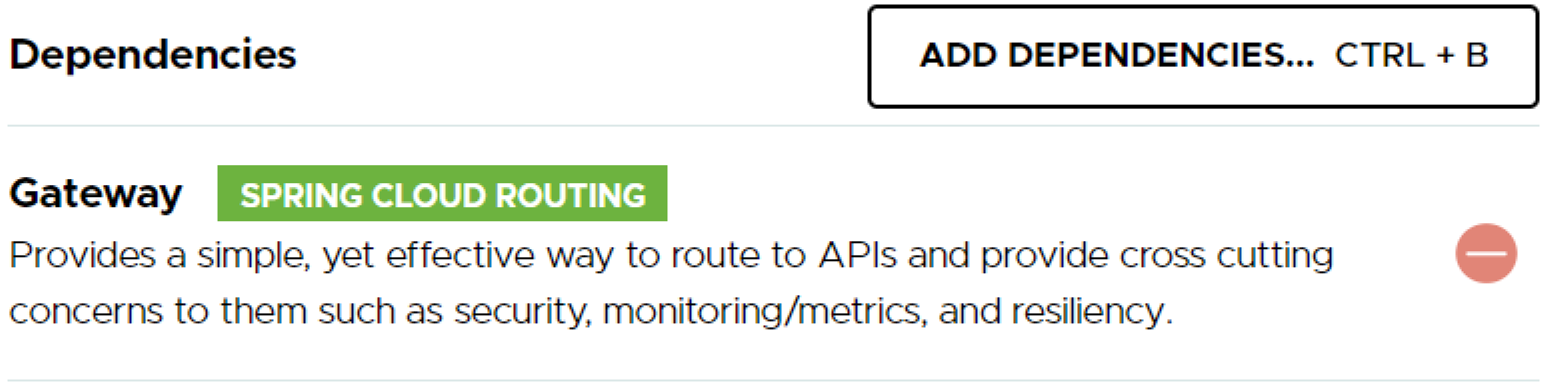
- Spring Cloud API Gateway is built on top of the Spring ecosystem.
- Spring Cloud Gateway aims to provide a simple, yet effective way to route to the APIs.
- It consists of the following:
  - Route
  - Predicate
  - Filter



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.

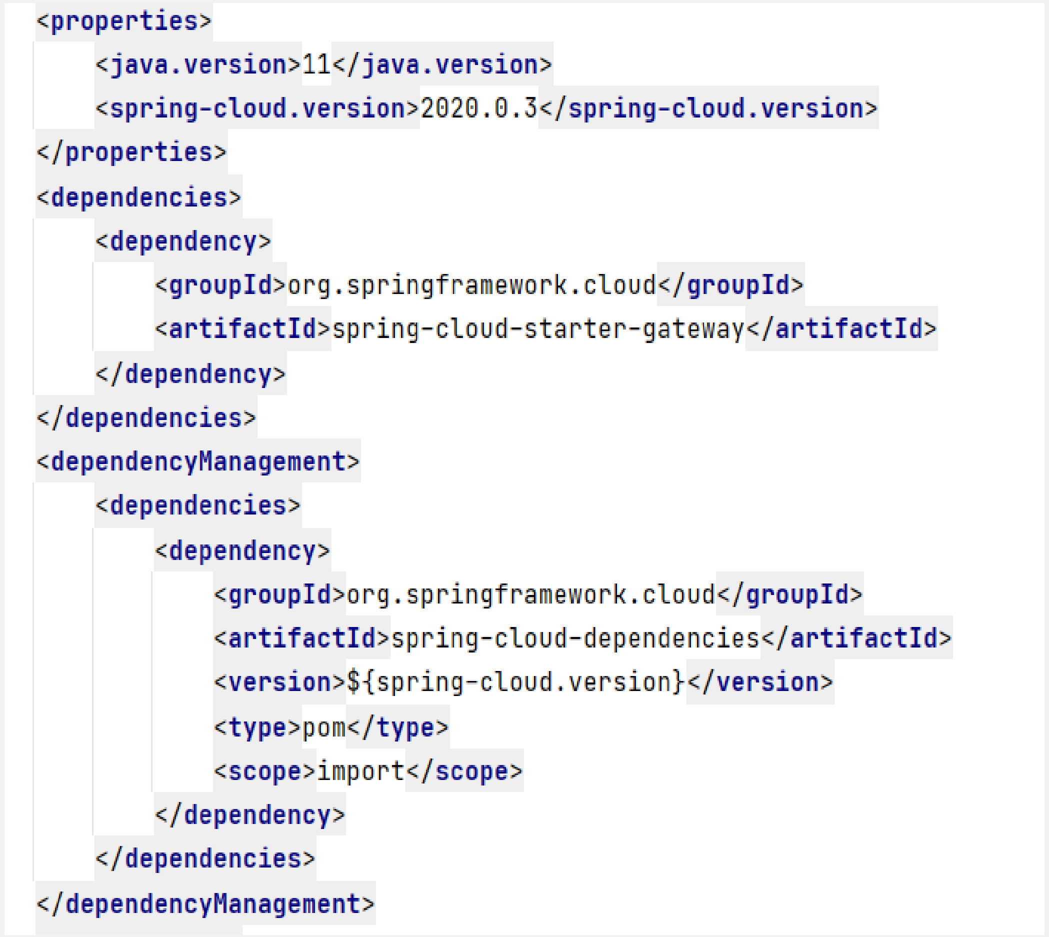

Dependencies

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.

pom.xml

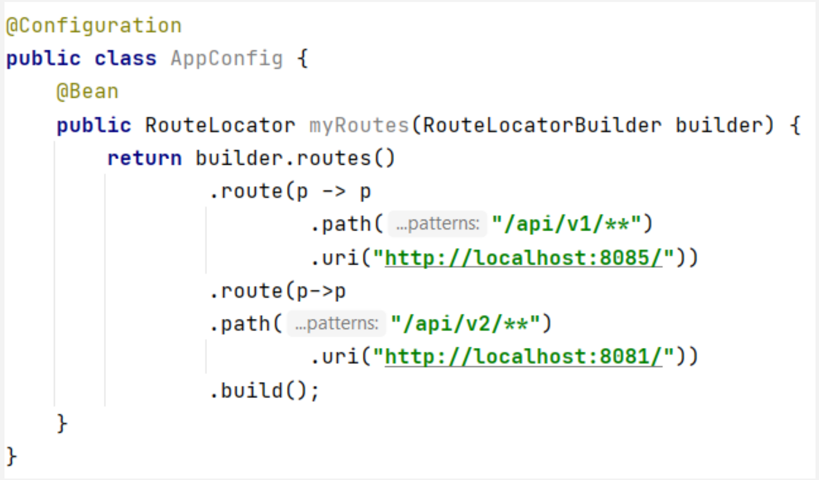


```xml
<properties>
    <java.version>11</java.version>
    <spring-cloud.version>2020.0.3</spring-cloud.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
</dependencies>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

- 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` – It is used to create routes.



```
@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();
    }
}
```

Streaming Application

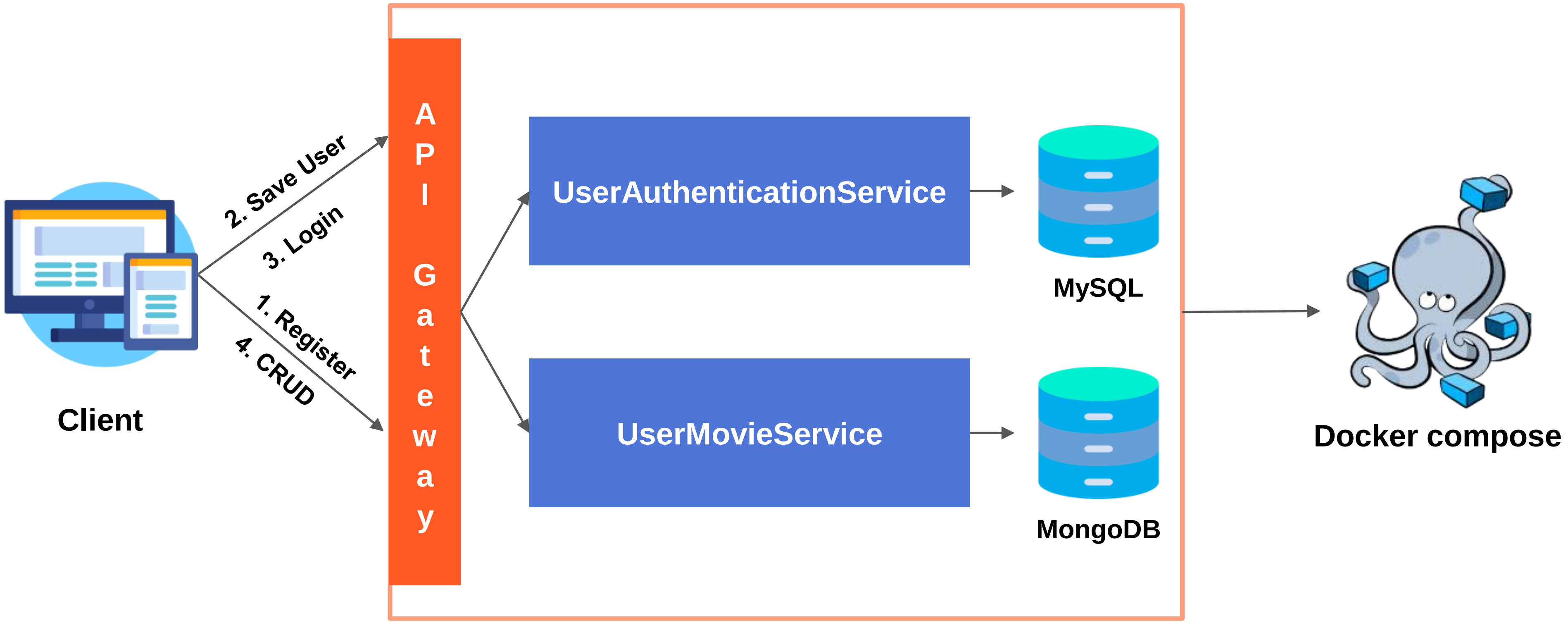
Consider a streaming application that enables users to watch movies on any smart device. The application provides multiple features to all its registered users. A user needs to register with the application in order to access some of its features. Let us create multiple microservices for the streaming application.

1. A user must first register with the application.
2. Use credentials such as id, password to login.
3. Access the features provided by the streaming application, like adding favourites, compiling a watch later list, etc.

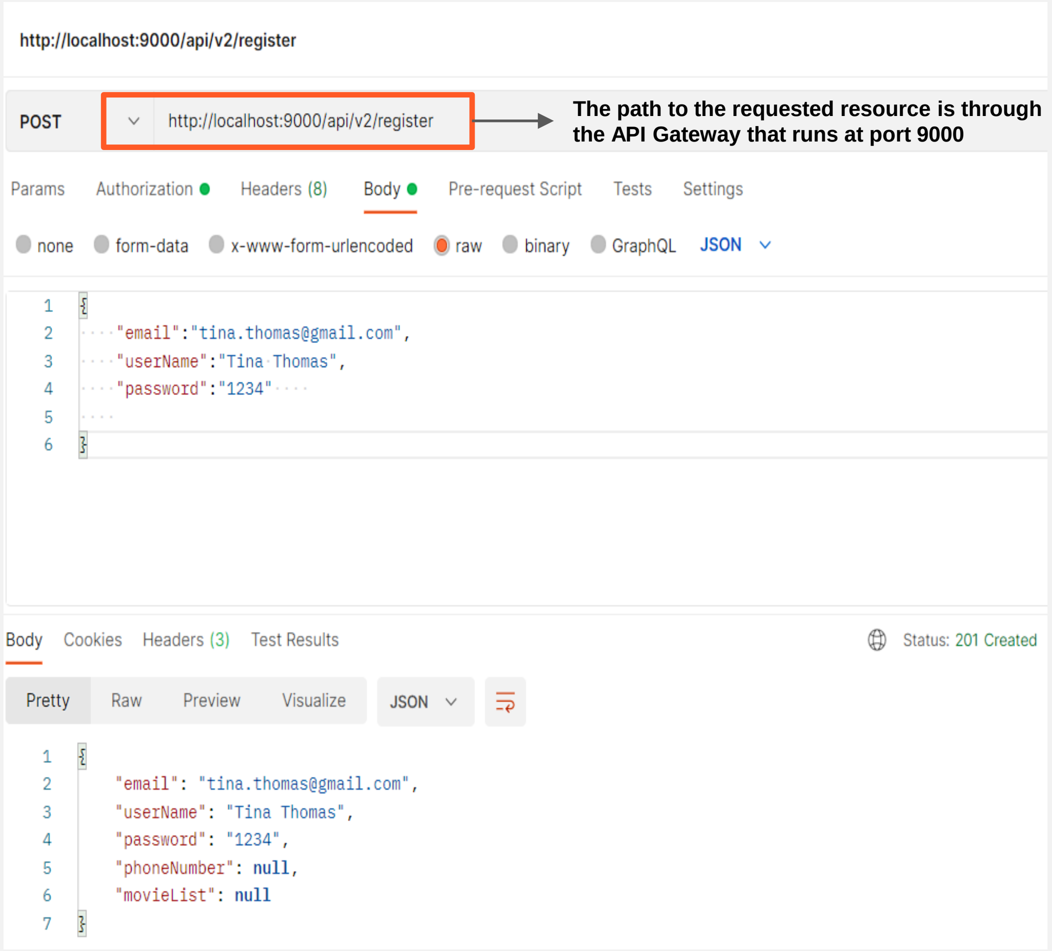
Let us create a parent project called **MovieApplication**.
This will contain the **UserAuthenticationService** and the **UserMovieService** as microservices. Enable single entry point by routing all requests through the spring cloud API.
Dockerize the application.

DEMO

How Does This Application Work?



Postman Output – Register a New User



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

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

The path to the requested resource is through the API Gateway that runs at port 9000

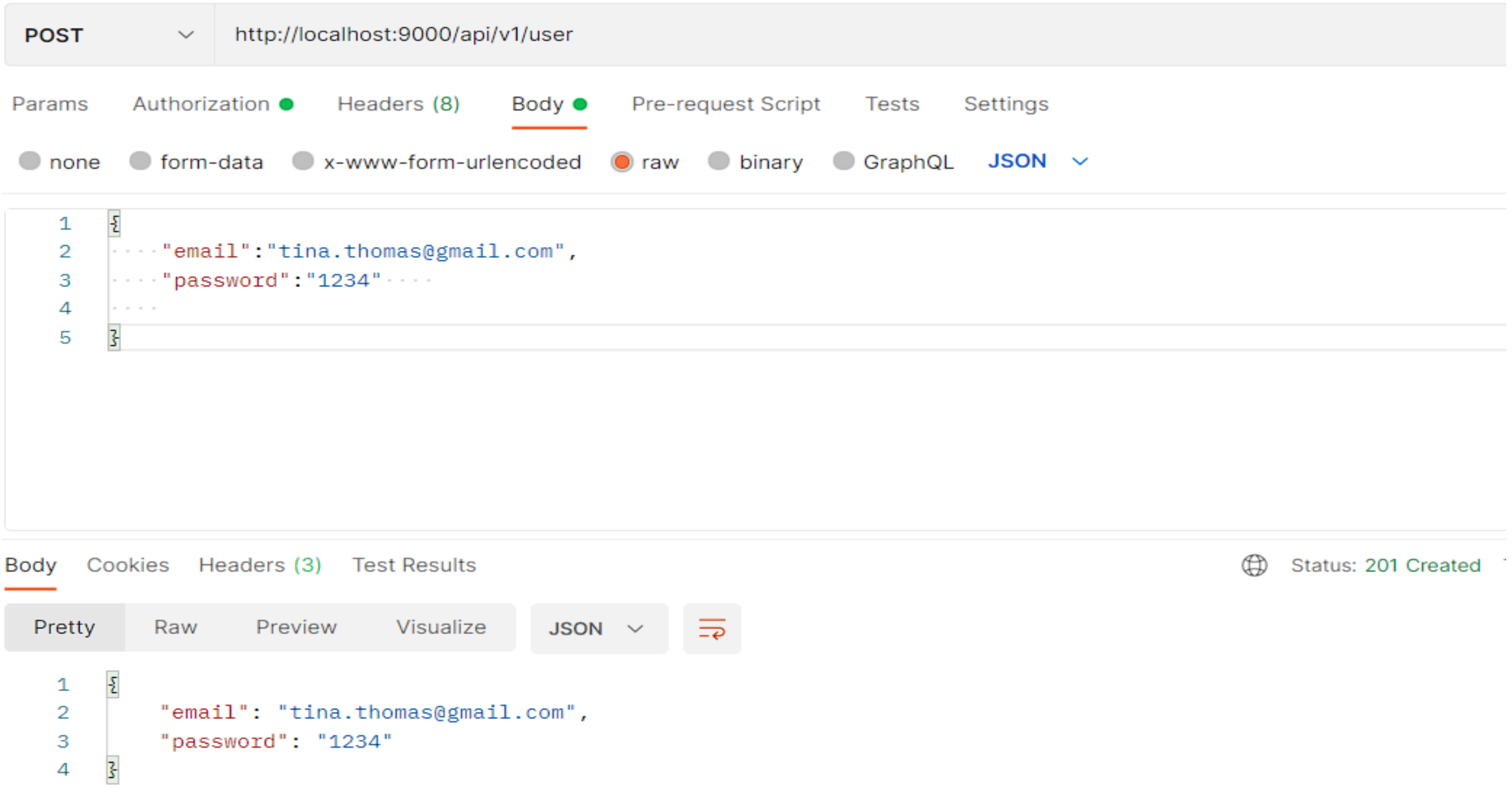
```
{
    "email":"tina.thomas@gmail.com",
    "userName":"Tina Thomas",
    "password":"1234"
    
}
```

Status: 201 Created

```
{
    "email": "tina.thomas@gmail.com",
    "userName": "Tina Thomas",
    "password": "1234",
    "phoneNumber": null,
    "movieList": null
}
```

- 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.

Postman Output – Save User Credentials



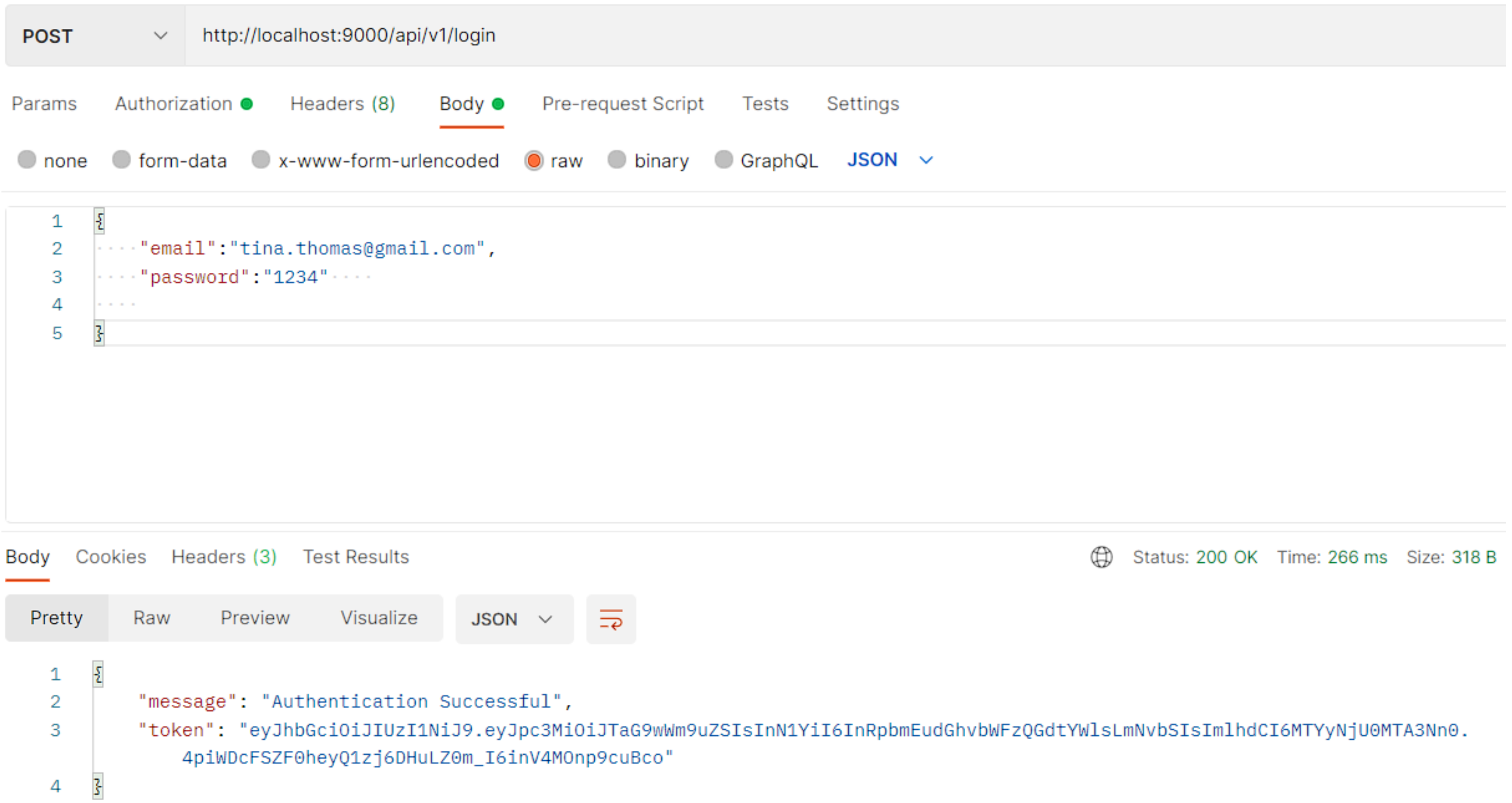
```
http://localhost:9000/api/v1/user
```

```
{
    "email":"tina.thomas@gmail.com",
    "password":"1234"
    
}
```

Status: 201 Created

```
{
    "email": "tina.thomas@gmail.com",
    "password": "1234"
}
```

Postman Output – Login to the Movie Service



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

Params | Authorization ● | Headers (8) | Body ● | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

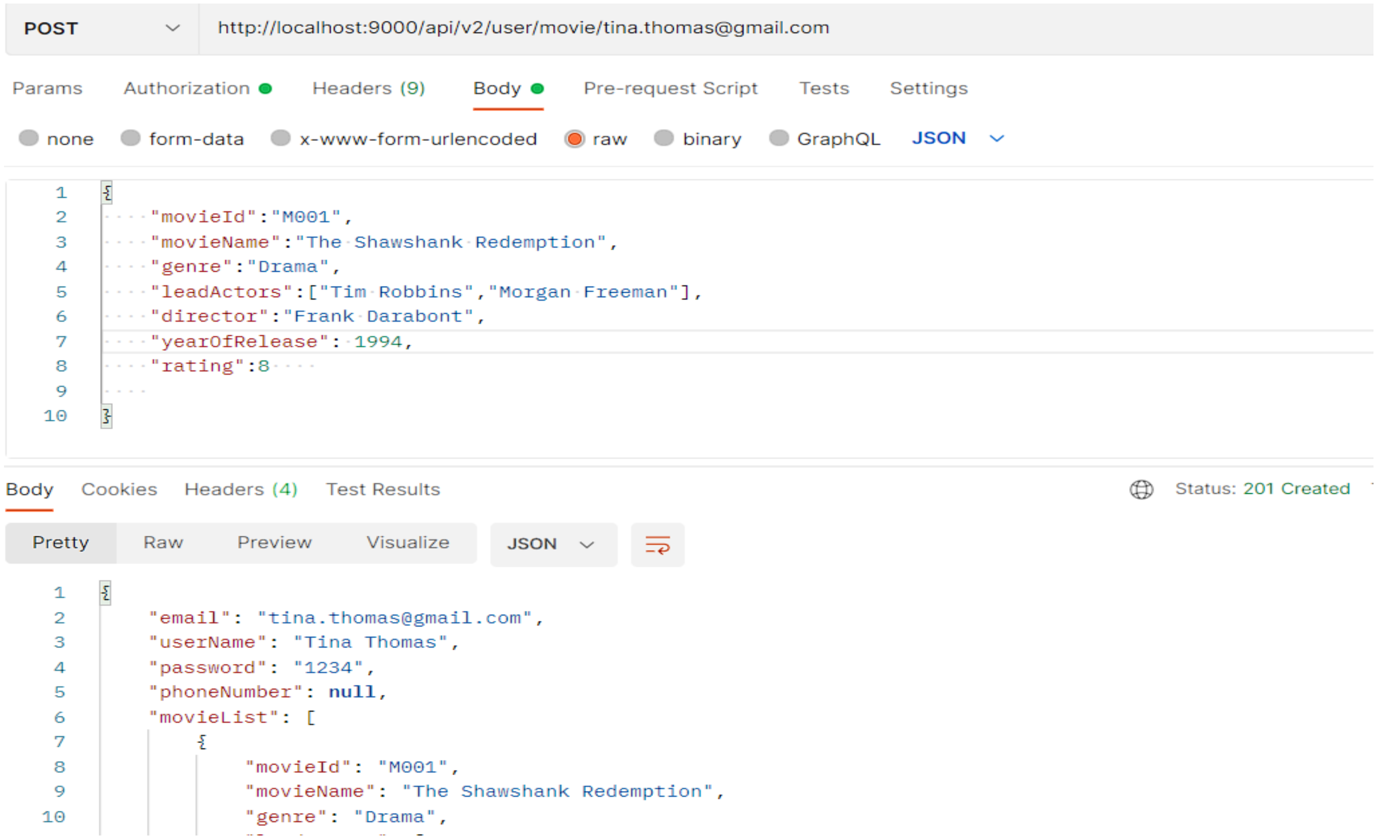
```
{
    "email":"tina.thomas@gmail.com",
    "password":"1234"

}
```

Body | Cookies | Headers (3) | Test Results — Status: 200 OK Time: 266 ms Size: 318 B

Pretty | Raw | Preview | Visualize | JSON

```
{
    "message": "Authentication Successful",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJTaG9wWm9uZSIsInN1YiI6InRpbmEudGhvbWFzQGdtYWlsLmNvbSIsImlhdCI6MTYyNjU0MTA3Nn0.
        4piWDcFSZF0heyQ1zj6DHuLZ0m_I6inV4MOnp9cuBco"
}
```

Postman Output – Add the Favourite Movie for a User



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

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

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```
{
    "movieId":"M001",
    "movieName":"The Shawshank Redemption",
    "genre":"Drama",
    "leadActors":["Tim Robbins","Morgan Freeman"],
    "director":"Frank Darabont",
    "yearOfRelease": 1994,
    "rating":8
    
}
```

Body | Cookies | Headers (4) | Test Results   Status: 201 Created

Pretty | Raw | Preview | Visualize | JSON

```
{
    "email": "tina.thomas@gmail.com",
    "userName": "Tina Thomas",
    "password": "1234",
    "phoneNumber": null,
    "movieList": [
        {
            "movieId": "M001",
            "movieName": "The Shawshank Redemption",
            "genre": "Drama",
```

Quick Check

_______________ is the basic building block of an API Gateway.

1. Route
2. Path
3. Id
4. URI

Quick Check: Solution

_________________ is the basic building block of an API Gateway.

1. **Route**
2. Path
3. Id
4. URI

Key Takeaways

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

Thank you!