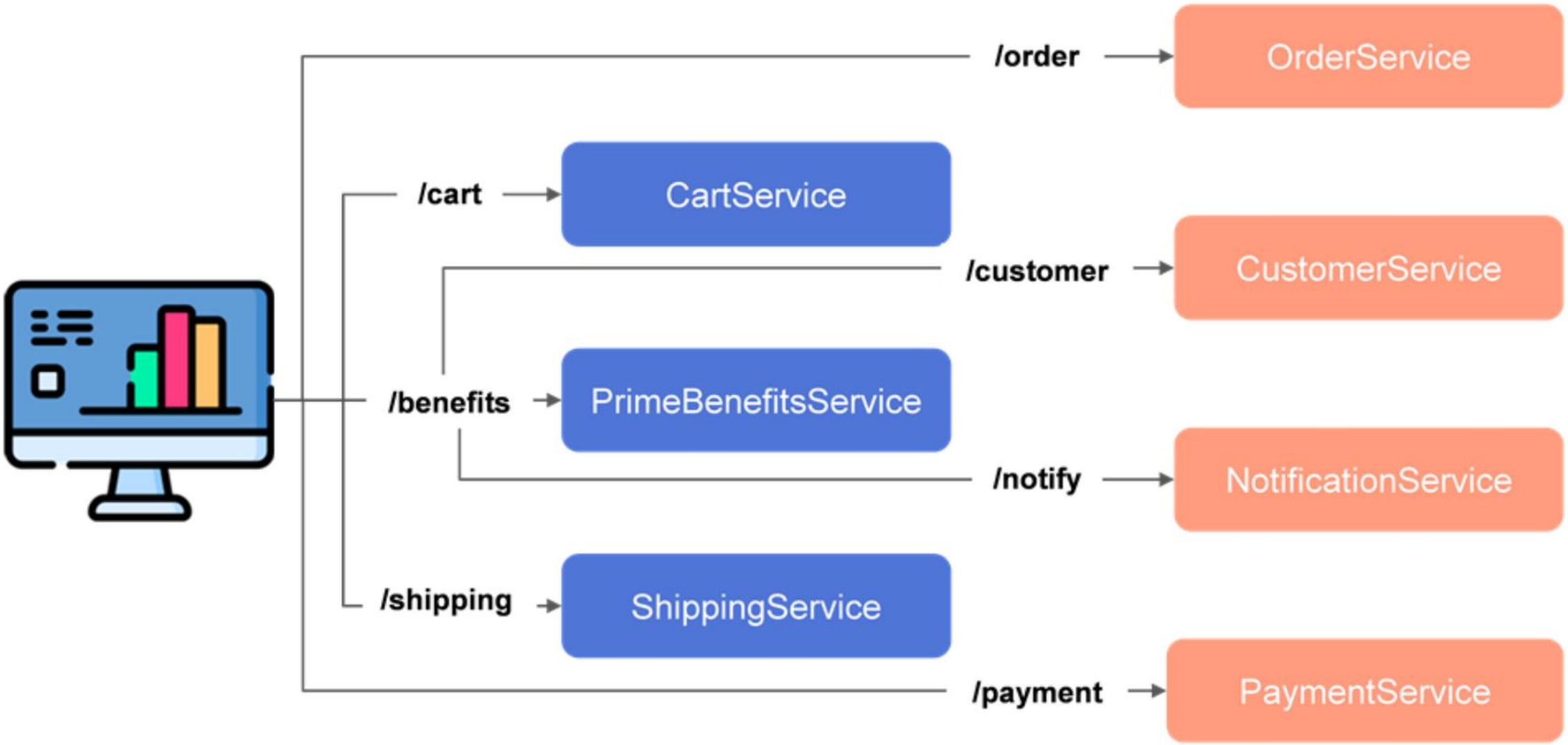
E-commerce Workflow – Multiple Services



Slide Note

- In a large application with multiple microservices, how will the client know which service to call?
- Will the client know all the URIs or path to the service?
- How do you notify the client of a change in the URI of a service?
- 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?

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?
- Should you give all the information to the client for the service name and the port number on which the service runs?
- If a port number to a service changes, how will the client know about the change?

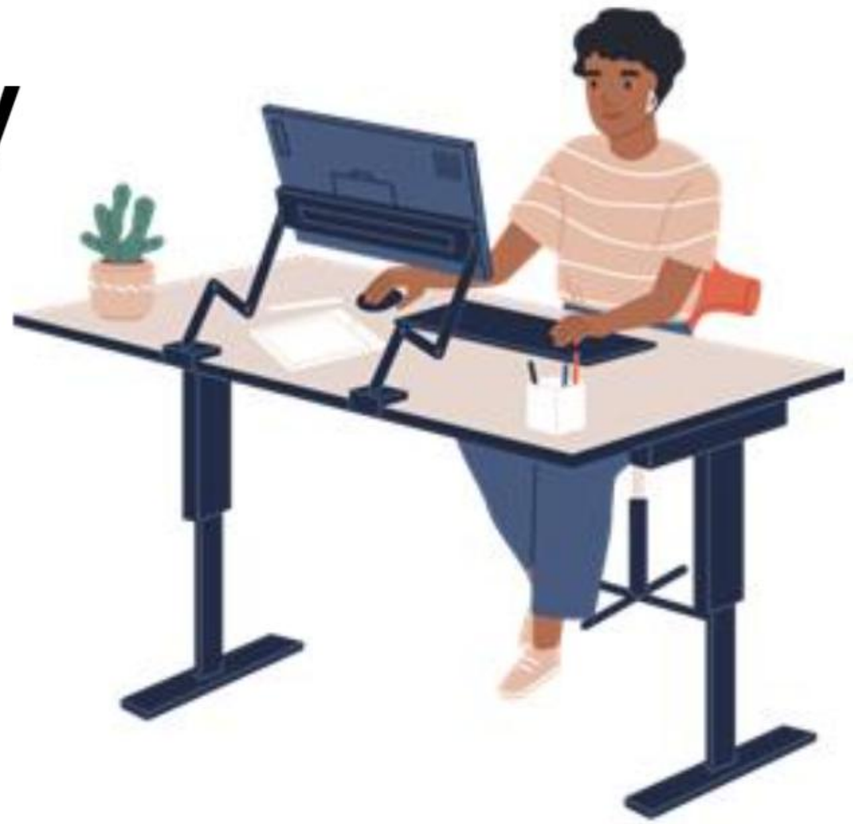
Slide Note

Think and Tell

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

Slide Note

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



Slide Note

Learning Objectives

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

Slide Note

Microservices Design Patterns

Slide Note

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 when developing individual microservices.

Slide Note

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 - The Circuit Breaker Pattern helps handle the failure of the services invoked.

Slide Note

The API Gateway Design Pattern

Slide Note

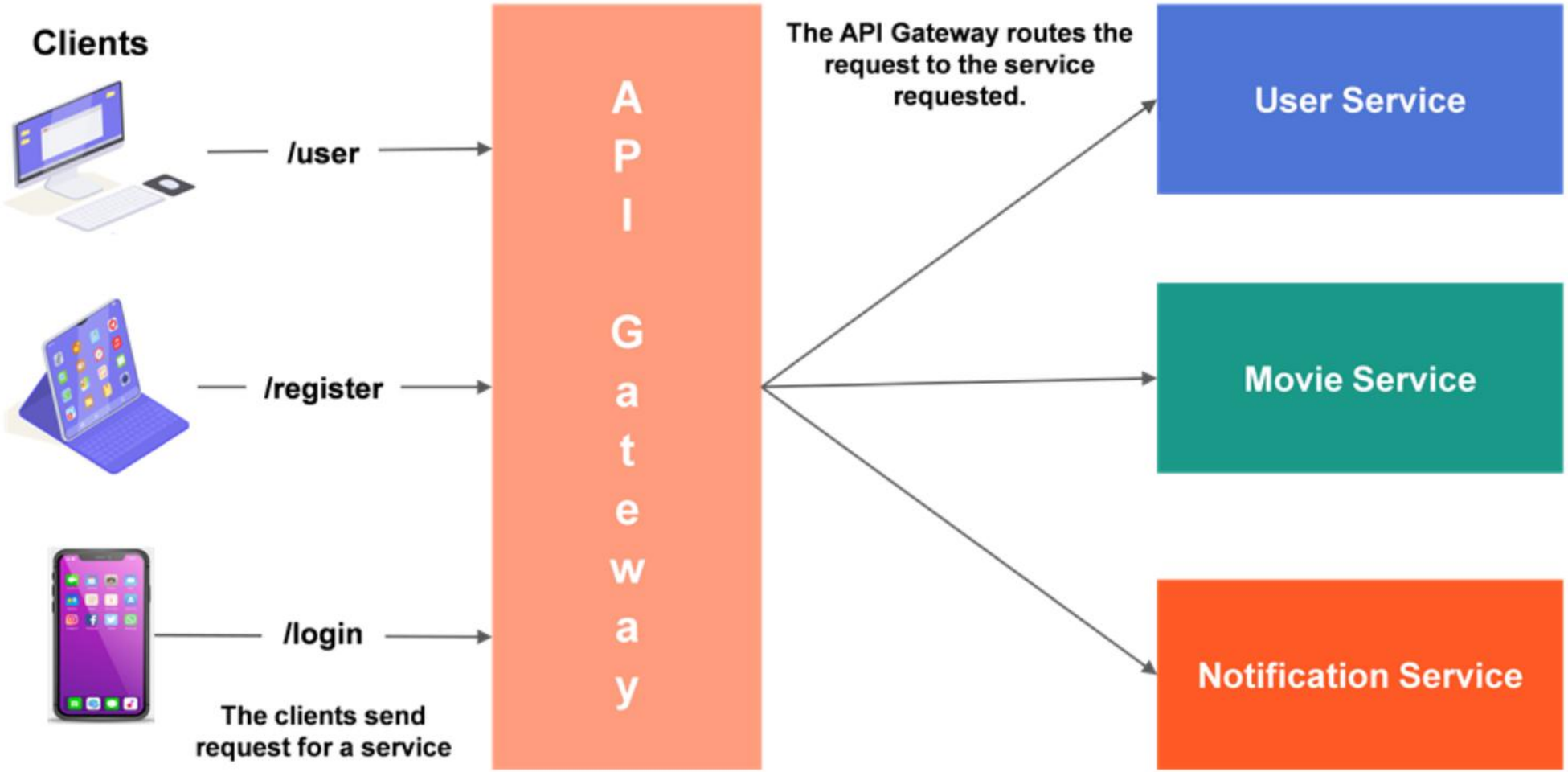
- It might have other responsibilities such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling.
- An API Gateway is an API management tool that sits between a client and a collection of backend services.
- The API Gateway encapsulates the internal system architecture and provides an API that is tailored to each client.
- It's common for API gateways to handle common tasks that are used across a system of API services, such as user authentication, rate limiting, and statistics.

What Is an 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.

Slide Note

How Does an API Gateway Work?



Slide Note

The Need for an 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.

Slide Note

Spring Cloud API Gateway

Slide Note

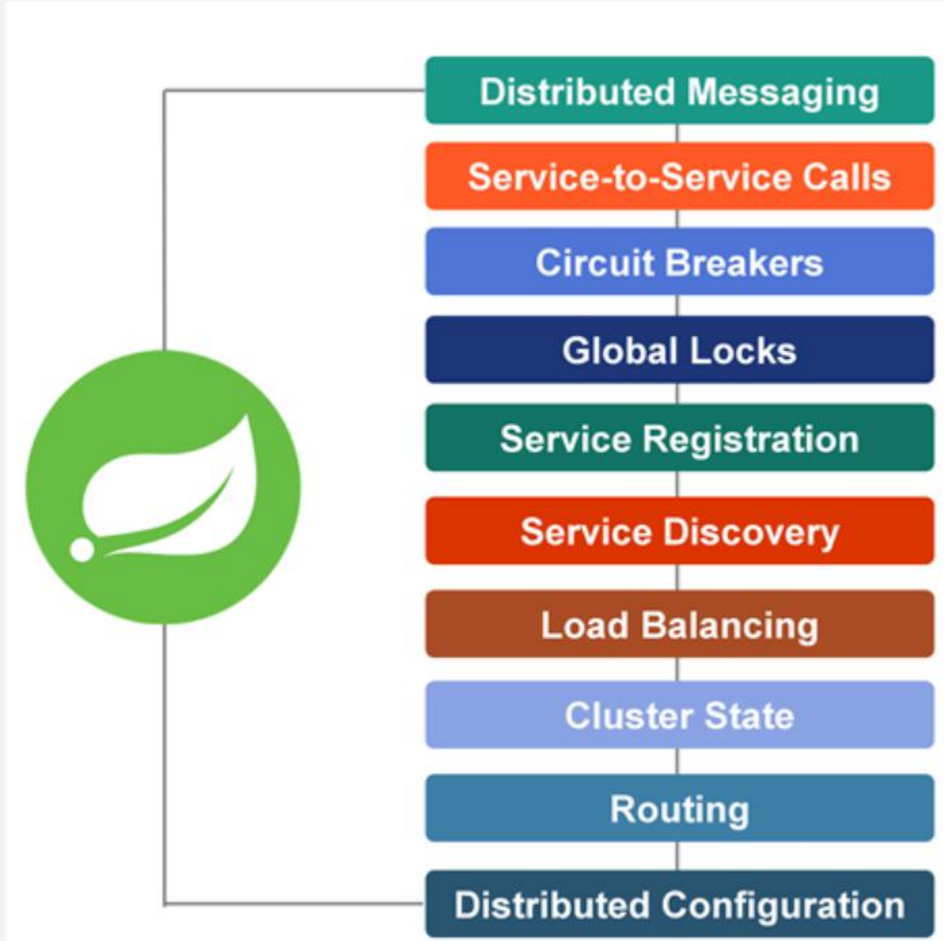
• Spring Cloud is a framework for building robust cloud applications. The framework facilitates the development of applications by providing solutions to many of the common problems faced when moving to a distributed environment.

• Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g., configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state).

• Coordination of distributed systems leads to boilerplate patterns, and using Spring Cloud, developers can quickly stand up services

and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centers, and managed platforms such as Cloud Foundry.

• One of the many advantages of running an application in the cloud is easy availability of a variety of services. Instead of managing hardware, installation, operation, backups, etc., you simply create and bind services with a click of a button or a shell command.



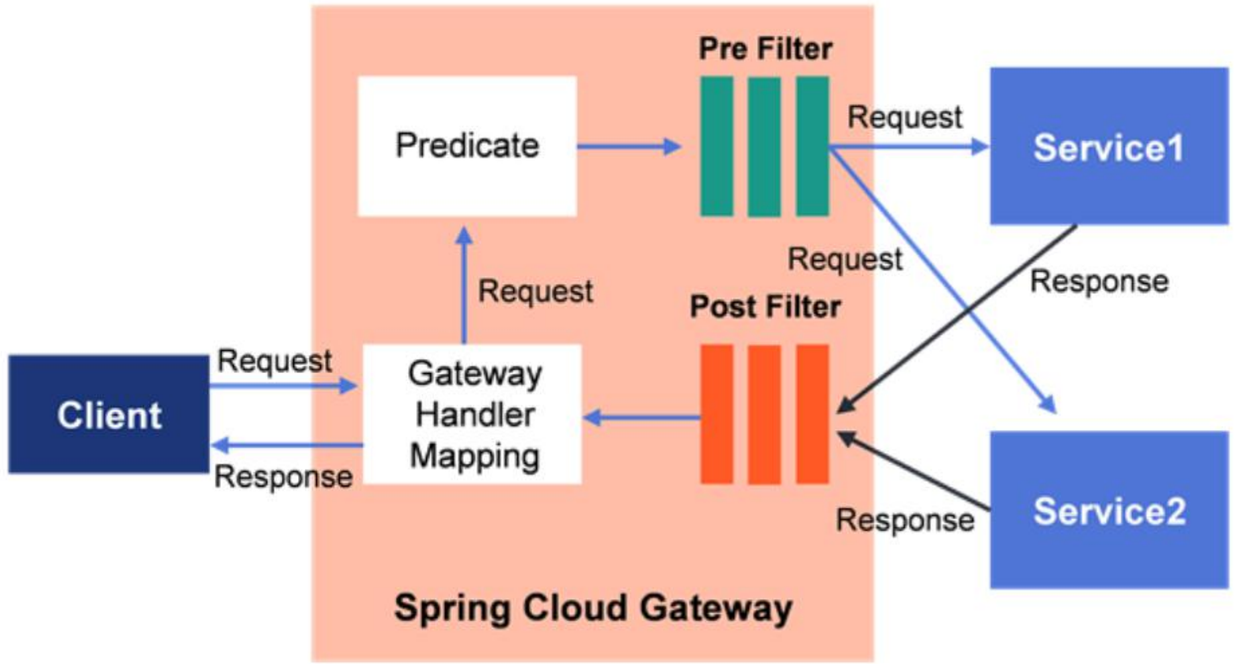
Features of Spring Cloud

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.

Slide Note

Features of Spring Cloud



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

Slide Note

Spring Cloud API Gateway Architecture

- Route: The route is the basic building block of the Gateway. It is defined by an ID, a destination URI, a collection of predicates, and a collection of filters. A route is matched if the aggregate predicate is true.
  - Predicate: This is a Java 8 Function Predicate. It is an object in Spring Cloud gateway that tests whether the given request fulfills a given condition. For each route, you can define one or more predicates that, if satisfied, will accept requests for the configured backend.
  - Filter: These are instances of Spring Framework GatewayFilter that have been constructed with a specific factory. Here, you can modify requests and responses before or after sending the downstream request.

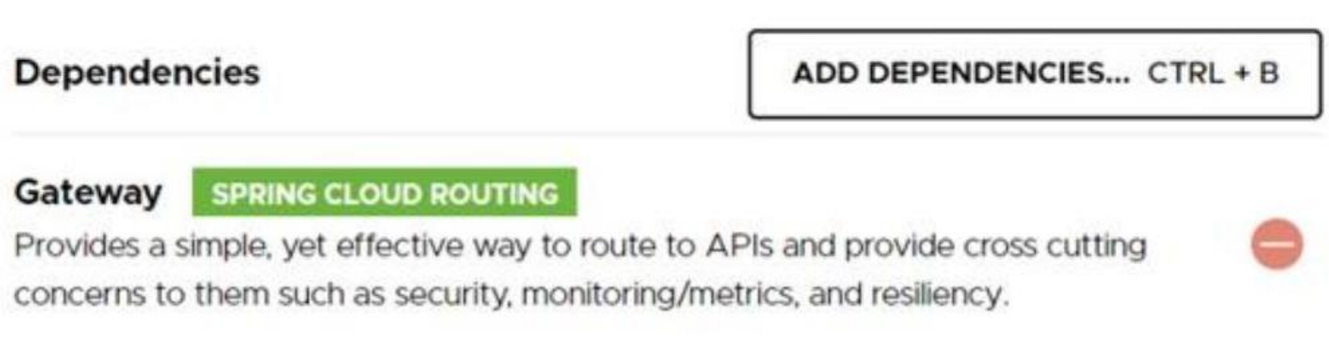
Slide Note

Implementing Spring Cloud API Gateway

Slide Note

Step 1: Adding Spring Cloud Dependencies

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



Slide Note

pom.xml

```
<properties>
    <java.version>11</java.version>
    <spring-cloud.version>2021.0.4</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>
```

- The Spring Cloud dependencies are added in the pom.xml file.
- The cloud dependencies of version 2021.0.4 are added under the dependency management tag.

Slide Note

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 classes below:
  - `RouteLocator` – To obtain route information:
    - `path` – the rest endpoint patterns
    - `uri` – the uri at which the service is currently running
  - `RouteLocatorBuilder` – Used to create routes
- Here, `UserAuthenticationService` runs on port `8086`.
- And `UserMovieService` runs on port `8081`.

```
@Bean
public RouteLocator myRoutes(RouteLocatorBuilder builder) {
    return builder.routes()
            .route(p -> p
                    .path( ...patterns: "/api/v1/**")
                    .uri("http://localhost:8086/"))

            .route(p->p
                    .path( ...patterns: "/api/v2/**")
                    .uri("http://localhost:8081/"))
            .build();
}
```

Slide Note

1. REST call is made to the API Gateway.

2. The API Gateway routes the request to the appropriate service.

3. The call is routed to the UserMovieService to register the new user.

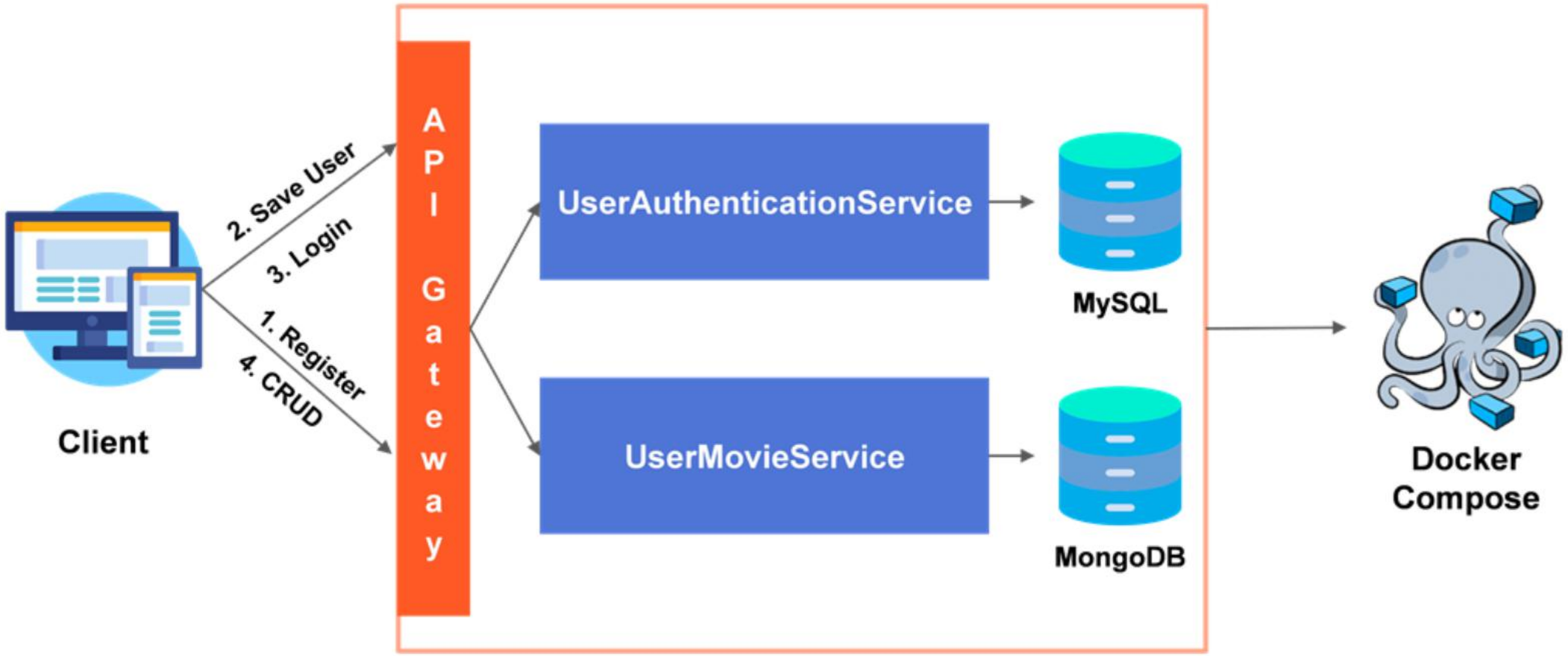
4. Once the user is registered, he/she will be directed to login, thus we need to save the user in the UserAuthenticationService.

5. Then the user logs into the system and a JSON web token is generated.

6. Using the JWT the user can perform all the CRUD operations.

7. Finally, the entire application is dockerized.

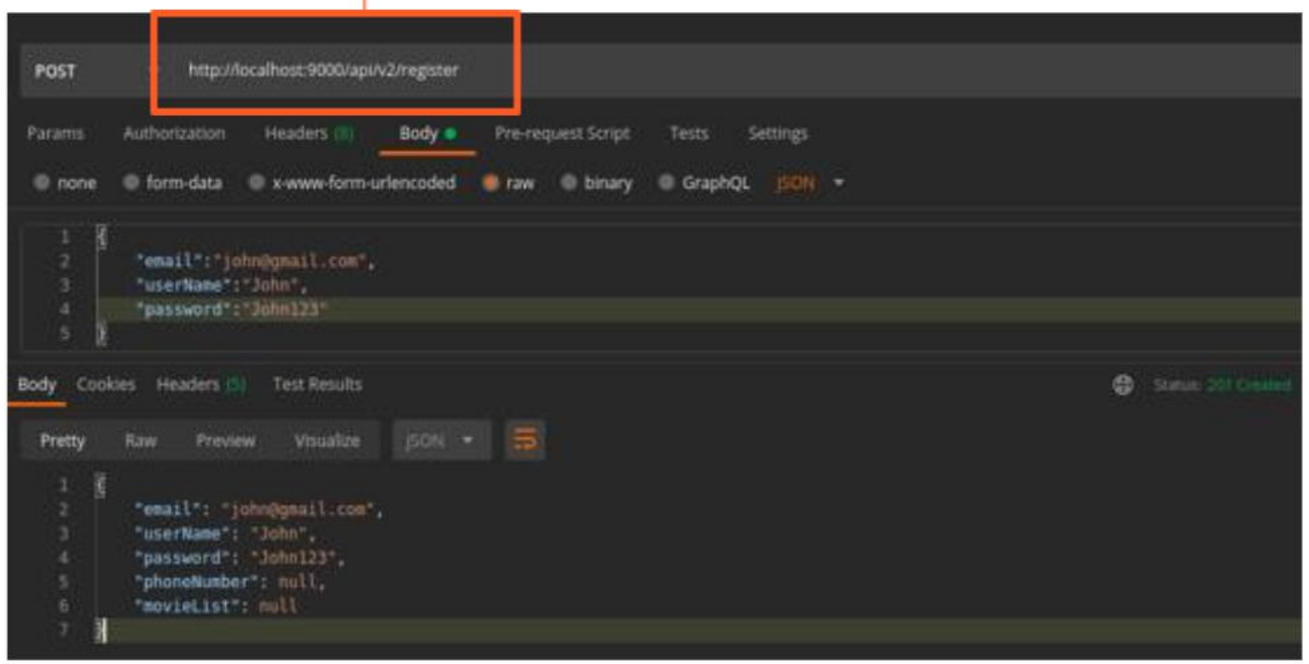
How Does This Application Work?



Slide Note

Postman Output – Register a New User

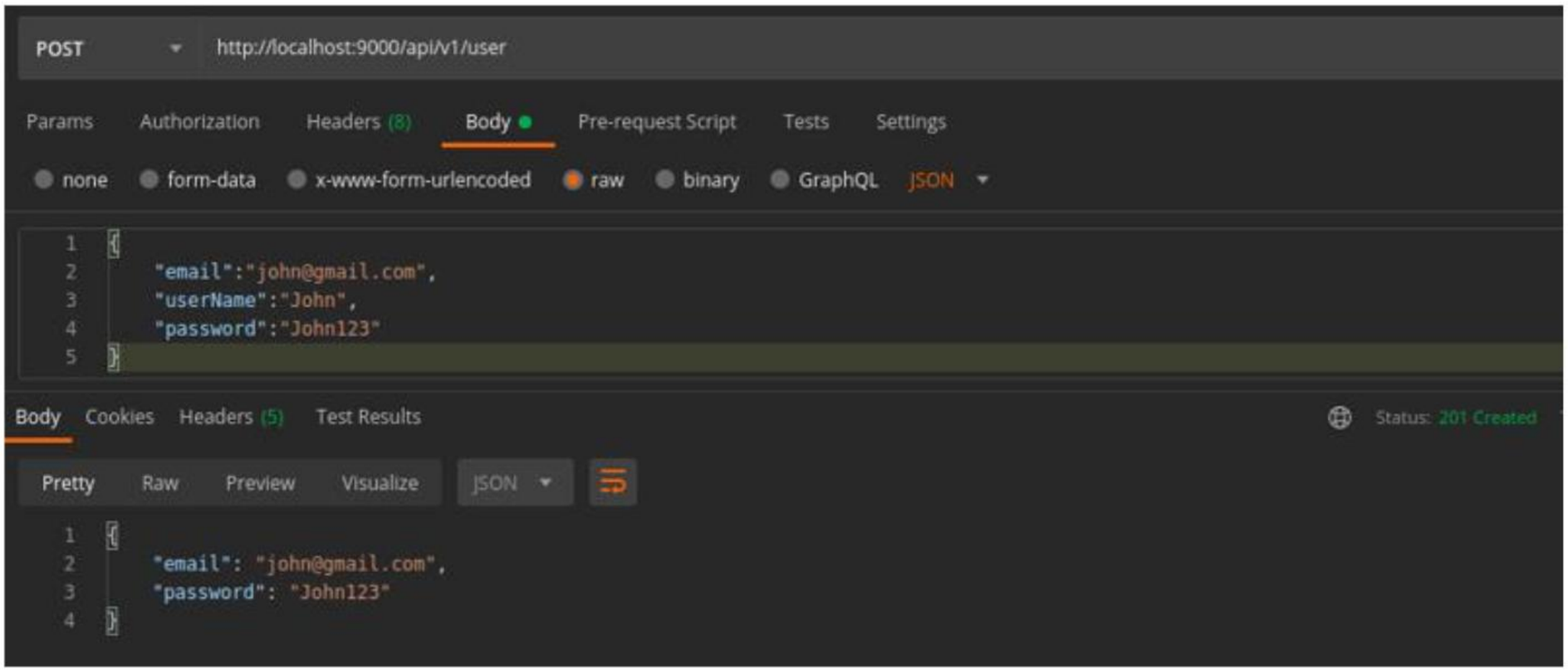
The path to the requested resource is through the API Gateway that runs at port 9000.



- UserAuthenticationService runs on port 8086 and UserMovieService is running on port 8081, 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 API Gateway runs on port 9000.
- The client is not aware of the details of the service like path, URI, etc.

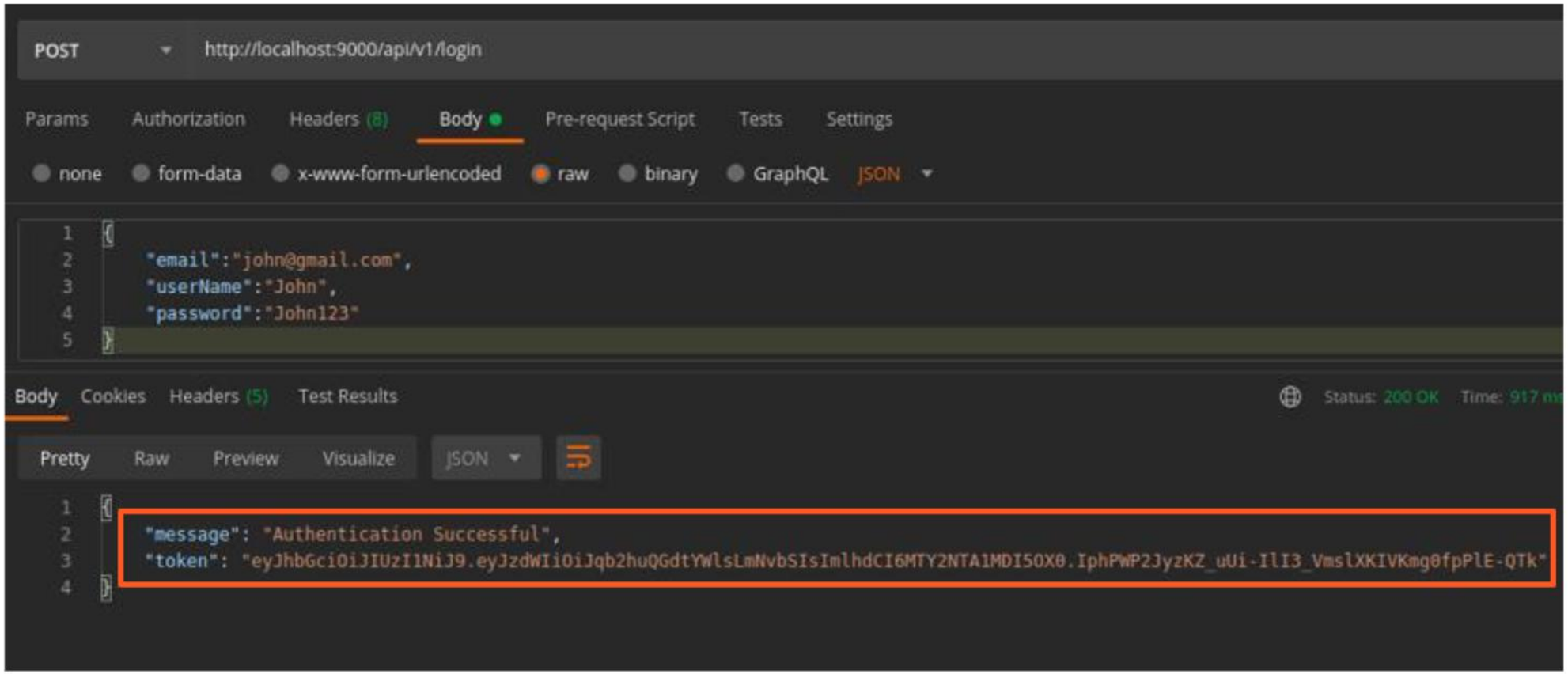
Slide Note

Postman Output – Save User Credentials



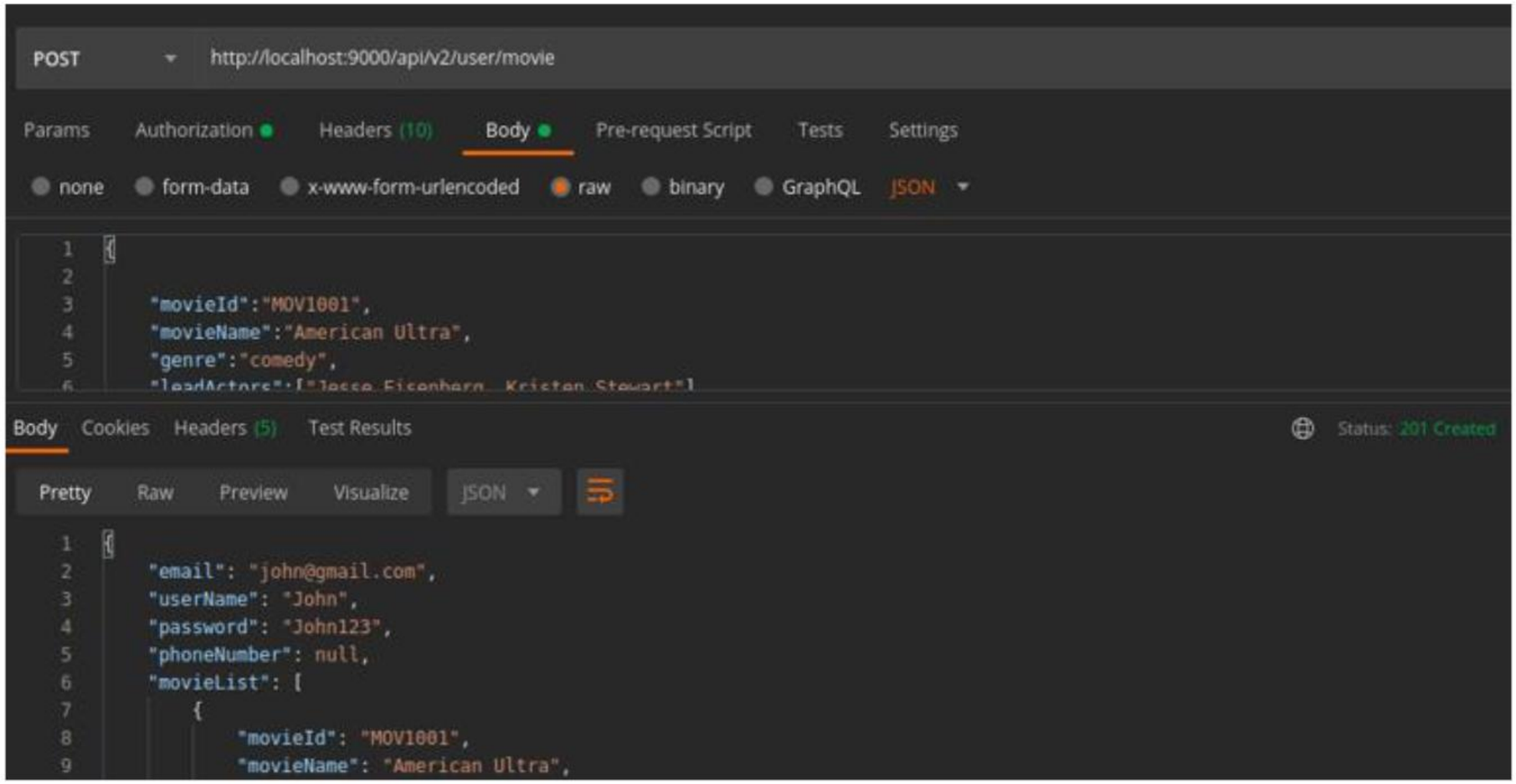
Slide Note

Postman Output – Log in to the Application



Slide Note

Postman Output – Add the Favorite Movie for a User



Menu

00:25/00:00:30

25/ 30

Slide Note

Streaming Application

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

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

DEMO

Slide Note

Streaming Application

Let's 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.
  Check the solution here.

DEMO

Slide Note

Quick Check

_____ is the basic building block of an API Gateway.

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

Slide Note

Quick Check: Solution

_____ is the basic building block of an API Gateway.

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

Menu

00:29/00:00:30

29/ 30