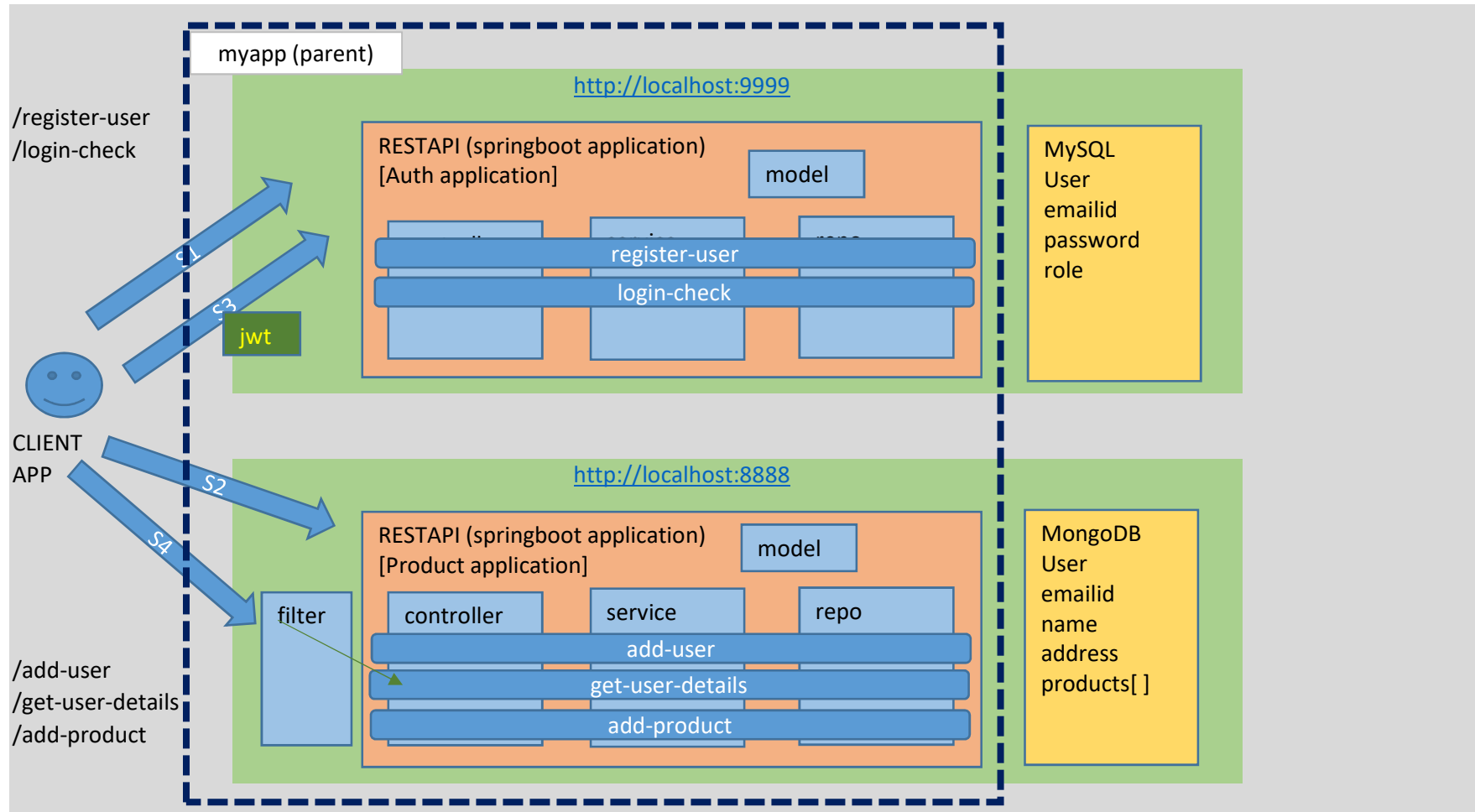
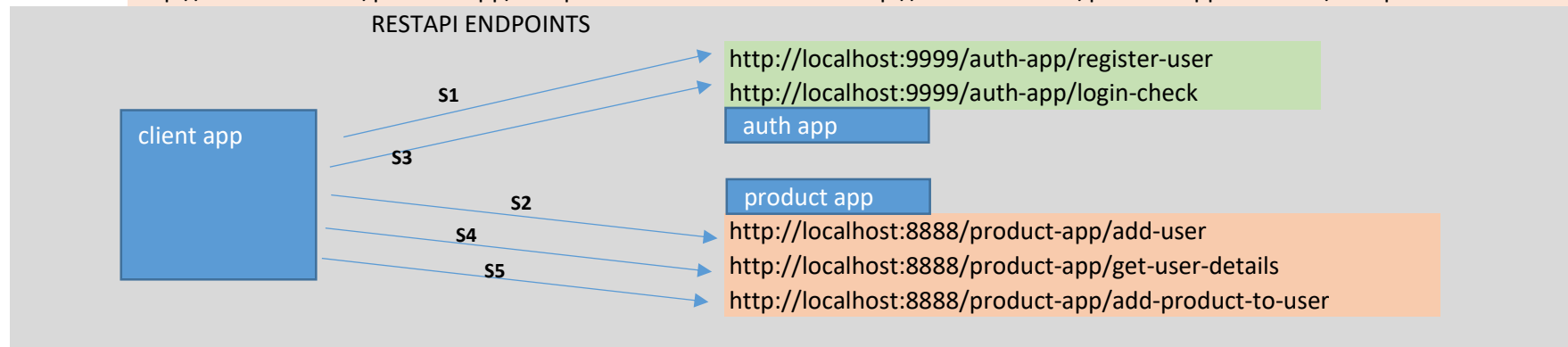


Microservices

Multi module



		Base URL	reqverb
auth app	http://localhost:9999/register-user	http://localhost:9999	/register-user
	http://localhost:9999/login-check	http://localhost:9999	/login-check
	http://localhost:9999/auth-app/register-user	http://localhost:9999/auth-app	/register-user
	http://localhost:9999/auth-app/login-check	http://localhost:9999/auth-app	/login-check
product app	http://localhost:8888/add-user	http://localhost:8888	/add-user
	http://localhost:8888/get-user-details	http://localhost:8888	/get-user-details
	http://localhost:8888/add-product-to-user	http://localhost:8888	/add-product-to-user
	http://localhost:8888/product-app/add-user	http://localhost:8888/product-app	/add-user
	http://localhost:8888/product-app/get-user-details	http://localhost:8888/product-app	/get-user-details
	http://localhost:8888/product-app/add-product-to-user	http://localhost:8888/product-app	/add-product-to-user



Client application uses RESTAPI Endpoints to interact with RESTAPI

Client application needs to identify which port based URL connecting to RESTAPI

http://....:9999/....

auth-app

http://....:8888/....

product-app

S1


POST ⌵ http://localhost:9999/auth-app/register-user

Params Authorization Headers (8) **Body** ● Pre-request Script

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

```
1 {
2   "emailId": "newuser@gmail.com",
3   "password": "12345"
4 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ 

```
1 {
2   "emailId": "newuser@gmail.com",
3   "password": "12345",
4   "role": "ROLE_USER"
5 }
```

S2


POST ⌵ http://localhost:8888/product-app/add-user

Params Authorization Headers (8) **Body** ● Pre-request Script

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

```
1 {
2   "emailId": "newuser@gmail.com",
3   "name": "New User",
4   "address": "Pune"
5 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ 

```
1 {
2   "emailId": "newuser@gmail.com",
3   "name": "New User",
4   "address": "Pune",
5   "products": []
6 }
```

S3

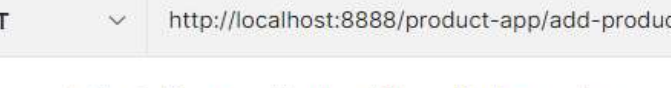
The screenshot shows a Postman interface with a POST request to `http://localhost:9999/auth-app/login-check`. The "Body" tab is selected, showing raw JSON data:

```
{  
  "emailId": "newuser@gmail.com",  
  "password": "12345"  
}
```

Below the request, the "Response" section shows the status code `200 OK` and the response body in the "Pretty" view:

```
{  
  "message": "Login success, Token generated",  
  "token": "eyJhbGciOiJIUzUxMiJ9.  
          eyJyb2xlIjoiaUk9MRV9VU0VSIIiwiaXNzIjoiaWludDZ1Y  
          GM52qMmVQeq4TcmVleHBACenBb4UnEBi-yJVhEbcn2xe"
```

S5



POST ▼ http://localhost:8888/product-app/add-product-to-use

Params Authorization Headers (9) Body Pre-request Script

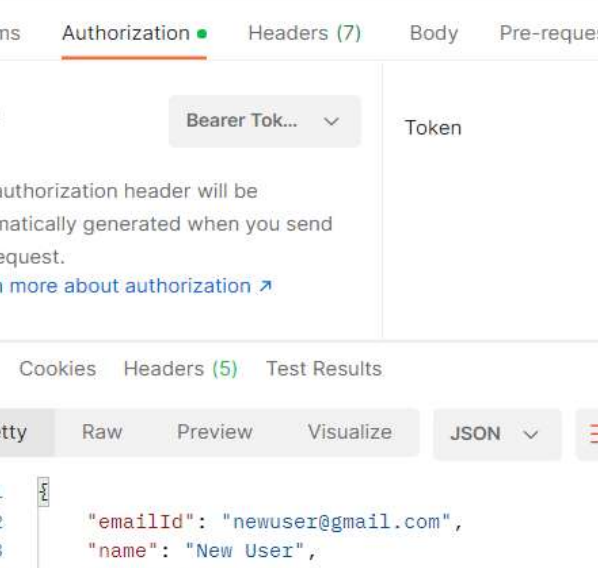
☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary

```

1 {
2   ... "productName": "Notebook",
3   ... "desc": "stationary",
4   ... "price": 45
5 }

```

S4



The screenshot displays the Swagger UI interface for a REST API. The top section shows the HTTP method as **GET** and the endpoint as `http://localhost:8888/product-app/get-user-details`. Below this, the **Authorization** tab is selected, showing the **Type** as **Bearer Token** and the **Token** field is empty. A note explains that the authorization header will be automatically generated when the request is sent, with a link to [Learn more about authorization](#). The **Body** tab is also visible, showing a JSON response in the **Pretty** format:

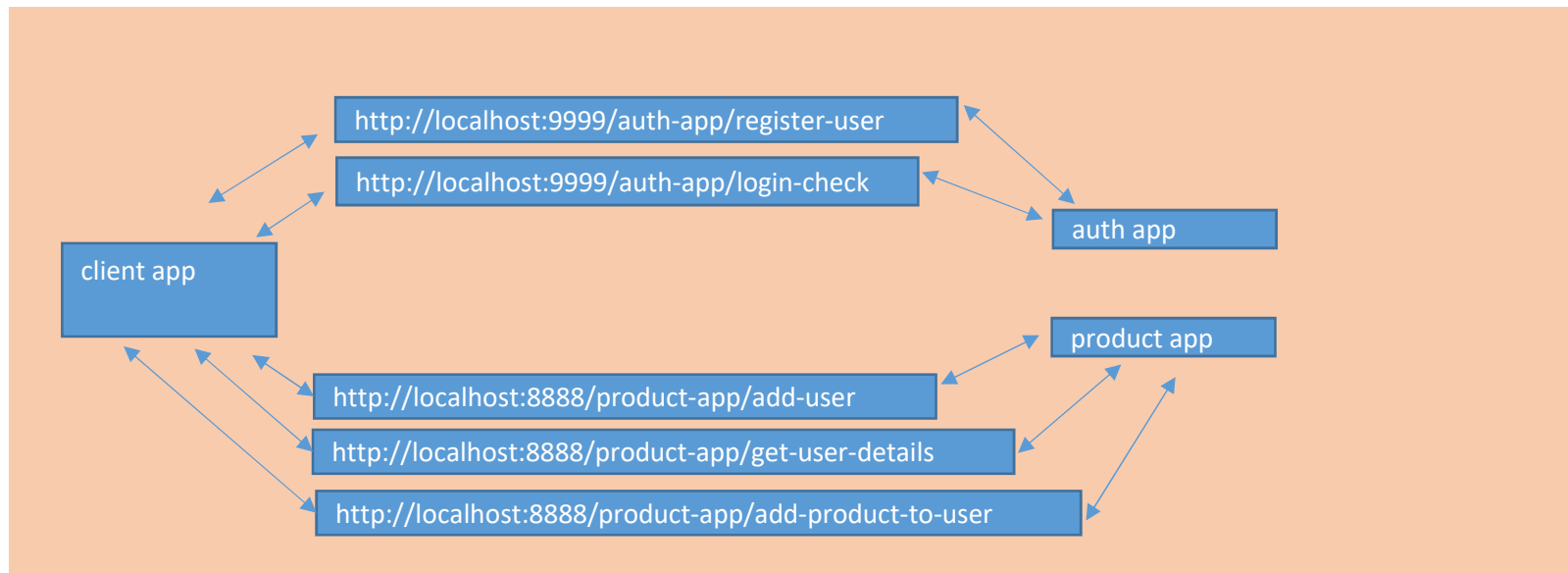
```
1 {
2   "emailId": "newuser@gmail.com",
3   "name": "New User",
4   "address": "Pune",
5   "products": []
6 }
```

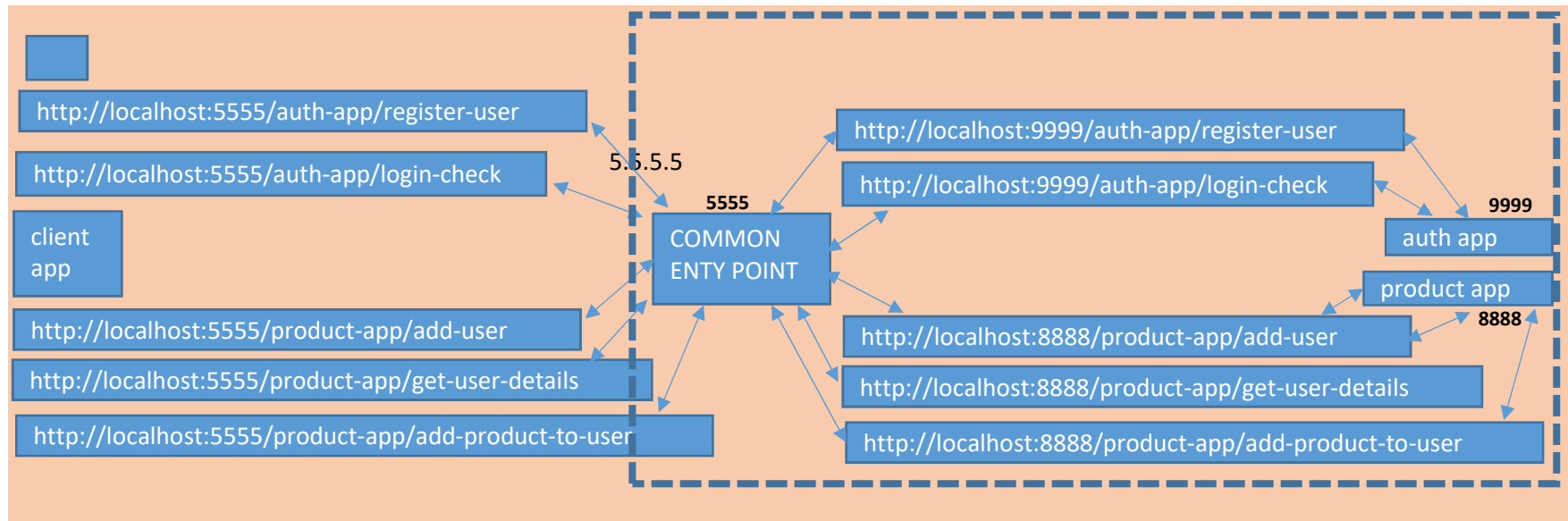
⌕ 5

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "emailId": "newuser@gmail.com",
3   "name": "New User",
4   "address": "Pune",
5   "products": [
6     {
7       "productName": "Notebook",
8       "desc": "stationary",
9       "price": 45
10    }
11  ]
12 }
```





in common/single entry point

if any request coming for '/auth-app/ANY_REQUEST_VERB OR ANY_PATH_VARIABLE'

then route to 'http://localhost:9999/*'

if any request coming for '/product-app/ANY_REQUEST_VERB OR ANY_PATH_VARIABLE'

then route to 'http://localhost:8888/*'

If product app and auth app hosted on different computers with ips as 1.1.1.1 and 2.2.2.2

if any request coming for '/auth-app/ANY_REQUEST_VERB OR ANY_PATH_VARIABLE'

then route to 'http://1.1.1.1:9999/*'

if any request coming for '/product-app/ANY_REQUEST_VERB OR ANY_PATH_VARIABLE'

then route to 'http://2.2.2.2:8888/*'

if any request coming for '/auth-app/ANY_REQUEST_VERB OR ANY_PATH_VARIABLE'

then route to 'http://auth.com/*'

if any request coming for '/product-app/ANY_REQUEST_VERB OR ANY_PATH_VARIABLE'

then route to 'http://product.com/*'

Steps to create and configure cloud gateway in existing application

Step 1 Make sure product application and auth application are working with all functionalities

Step 2 Create new springboot application for gw



Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Java ☐ Kotlin ☐ Groovy
☒ Maven

Language

Spring Boot

☐ 3.0.2 (SNAPSHOT) ☐ 3.0.1 ☐ 2.7.8 (SNAPSHOT) ☒ 2.7.7

Project Metadata

Group
Artifact
Name
Description
Package name
Packaging ☒ Jar ☐ War
Java ☐ 19 ☒ 17 ☐ 11 ☐ 8

Dependencies

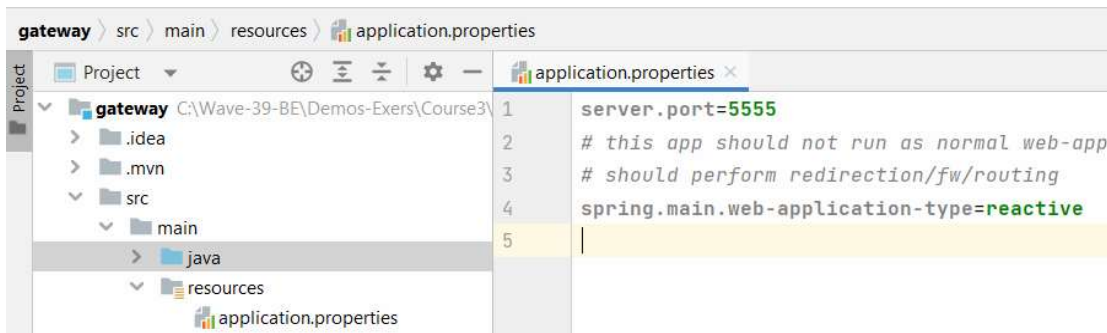
ADD DE

Gateway ☒ SPRING CLOUD ROUTING

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

download, extract, copy to workspace, open in intellij
check settings if required
create any packages if required

Step 2 Edit application properties



Step 3 Define routes in configuration

http://localhost:5555/auth-app/register-user

http://localhost:5555/auth-app/login-check

http://localhost:5555/product-app/add-user

http://localhost:5555/product-app/get-user-details

http://localhost:5555/product-app/add-product-to-user

http://localhost:9999/auth-app/register-user

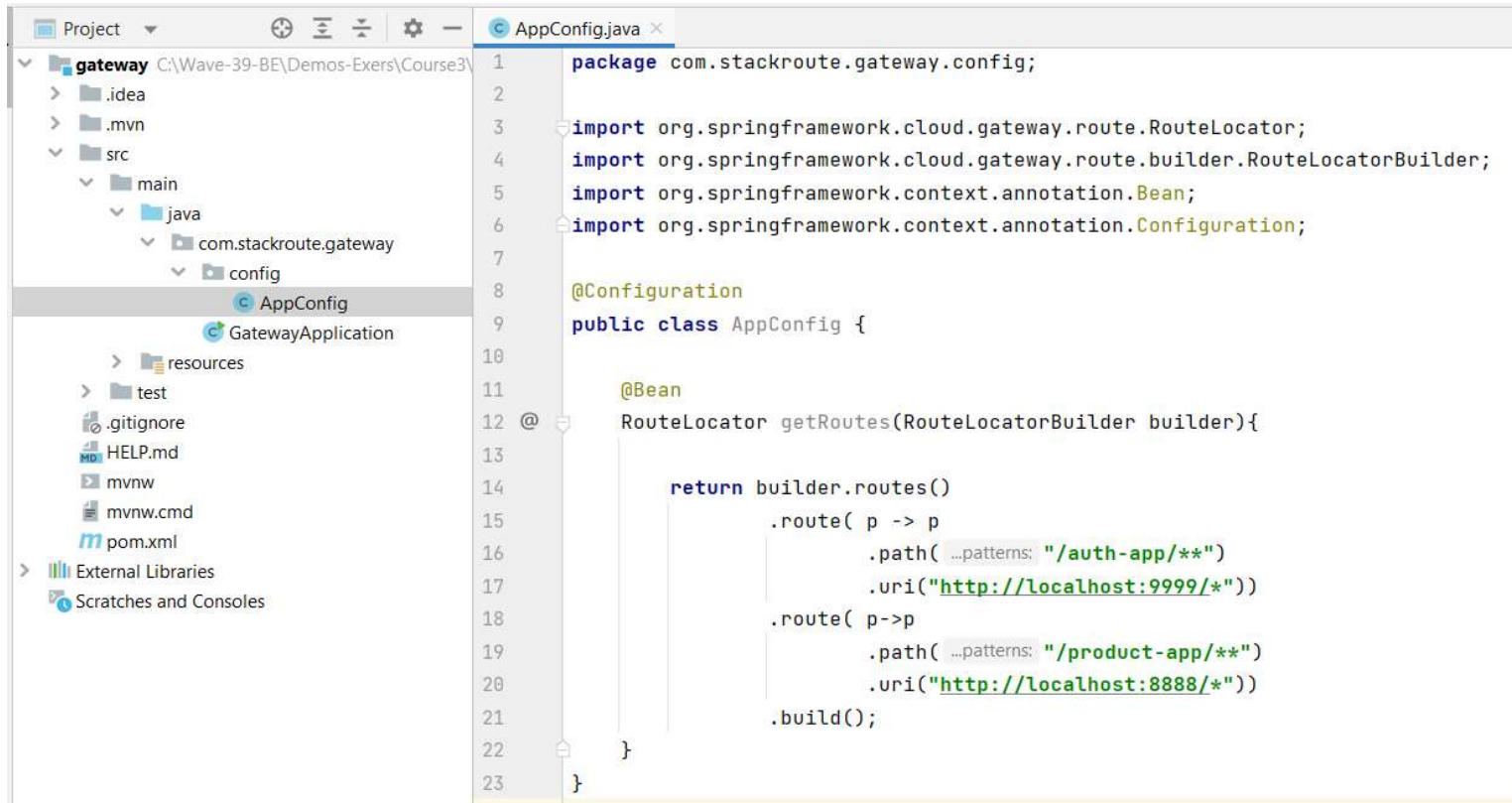
http://localhost:9999/auth-app/login-check

http://localhost:8888/product-app/add-user

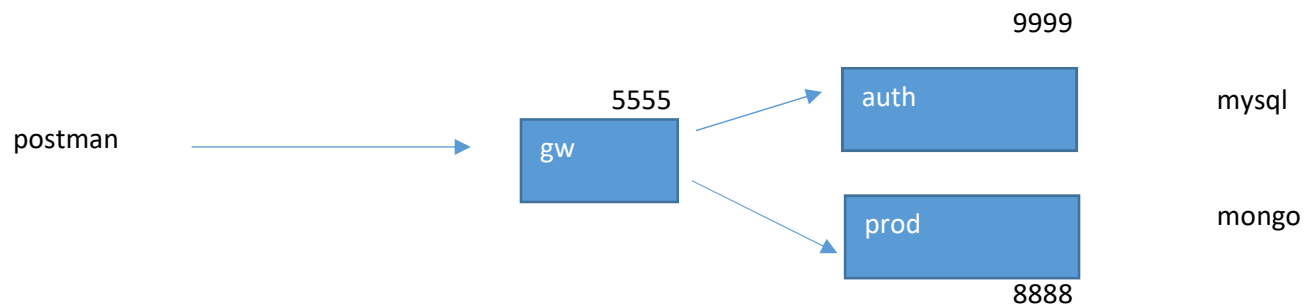
http://localhost:8888/product-app/get-user-details

http://localhost:8888/product-app/add-product-to-user

/auth-app/** ---> http://localhost:9999/* http://localhost:9999/as it is
/product-app/** ---> http://localhost:8888/*



```
1 package com.stackroute.gateway.config;
2
3 import org.springframework.cloud.gateway.route.RouteLocator;
4 import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.context.annotation.Configuration;
7
8 @Configuration
9 public class AppConfig {
10
11     @Bean
12     @RouteLocator
13     RouteLocator getRoutes(RouteLocatorBuilder builder){
14
15         return builder.routes()
16             .route( p -> p
17                 .path( ...patterns: "/auth-app/**")
18                 .uri("http://localhost:9999/*"))
19             .route( p->p
20                 .path( ...patterns: "/product-app/**")
21                 .uri("http://localhost:8888/*"))
22             .build();
23     }
24 }
```



- S1 create user a/c in auth app
- S2 create user a/c in product app
- S3 login check with auth app, returns JWT if login is success
- S4 access product app intercepted URL resource using JWT (get-user-details)
- S5 access product app intercepted URL resource using JWT (add-product-to-user)

How client application connects with BE REST API

without GW

http://localhost:9999/auth-app/register-user
 http://localhost:9999/auth-app/login-check
 http://localhost:8888/product-app/add-user
 http://localhost:8888/product-app/get-user-details
 http://localhost:8888/product-app/add-product-to-user

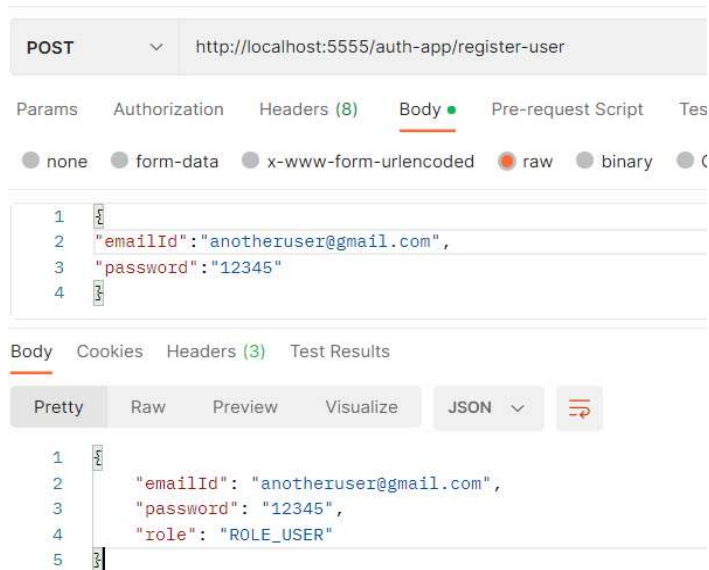
with GW

S1 http://localhost:5555/auth-app/register-user
 S3 http://localhost:5555/auth-app/login-check
 S2 http://localhost:5555/product-app/add-user
 S4 http://localhost:5555/product-app/get-user-details
 S5 http://localhost:5555/product-app/add-product-to-user

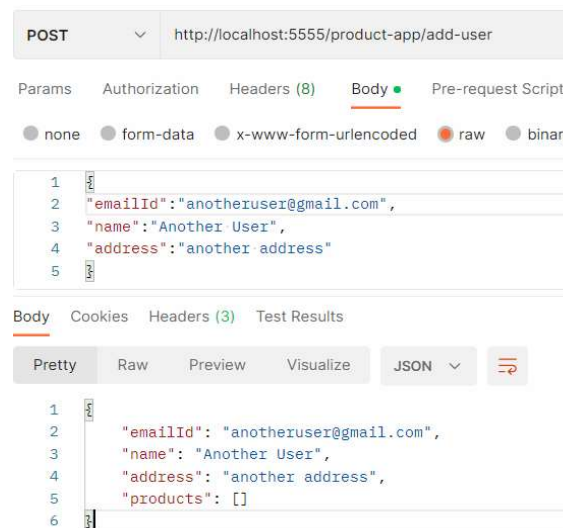
run app

check with postman

S1



S2



S3

POST ▼ http://localhost:5555/auth-app/login-check

Params Authorization Headers (8) **Body** Pre-request Script

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary

```

1  [
2    "emailId": "anotheruser@gmail.com",
3    "password": "12345"
4  ]

```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```

1  [
2    "message": "Login success, Token generated",
3    "token": "eyJhbGciOiJIUzUxMiJ9.eyJyYb2xliIjoiUk9MRV9VU0VSIiwiaXNzIjoiTXlDb21wYW
4    -qHAVI1I2-hX1imVu8Vc7FAhA0eIzzaY7ty5La-3mNwqRA

```

S4

GET ▼ http://localhost:5555/product-app/get-user-details

Params **Authorization** ● Headers (7) Body Pre-request Script

Type Bearer Tok... ▼ Token

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```

1  [
2    "emailId": "anotheruser@gmail.com",
3    "name": "Another User",
4    "address": "another address",
5    "products": []
6  ]

```

S5

POST ▼ http://localhost:5555/product-app/add-product-to-user

Params Authorization ● Headers (9) **Body** Pre-request Script

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ●

```

1  [
2    "productName": "Notebook",
3    "desc": "stationary",
4    "price": 45

```

Body Cookies Headers (3) Test Results

```
1  {
2      "emailId": "anotheruser@gmail.com",
3      "name": "Another User",
4      "address": "another address",
5      "products": [
6          {
7              "productName": "Notebook",
8              "desc": "stationary",
9              "price": 45
10         }
11     ]
12 }
```

Note

If any spring cloud tool dependency injected in pom.xml
makesure below elements also added in pom.xml

```
31  </dependencies>
32
33  <dependencyManagement>
34      <dependencies>
35          <dependency>
36              <groupId>org.springframework.cloud</groupId>
37              <artifactId>spring-cloud-dependencies</artifactId>
38              <version>${spring-cloud.version}</version>
39              <type>pom</type>
40              <scope>import</scope>
41          </dependency>
42      </dependencies>
43  </dependencyManagement>
44
45  <build>
```

```
16  <properties>
17      <java.version>17</java.version>
18      <spring-cloud.version>2021.0.5</spring-cloud.version>
19  </properties>
```