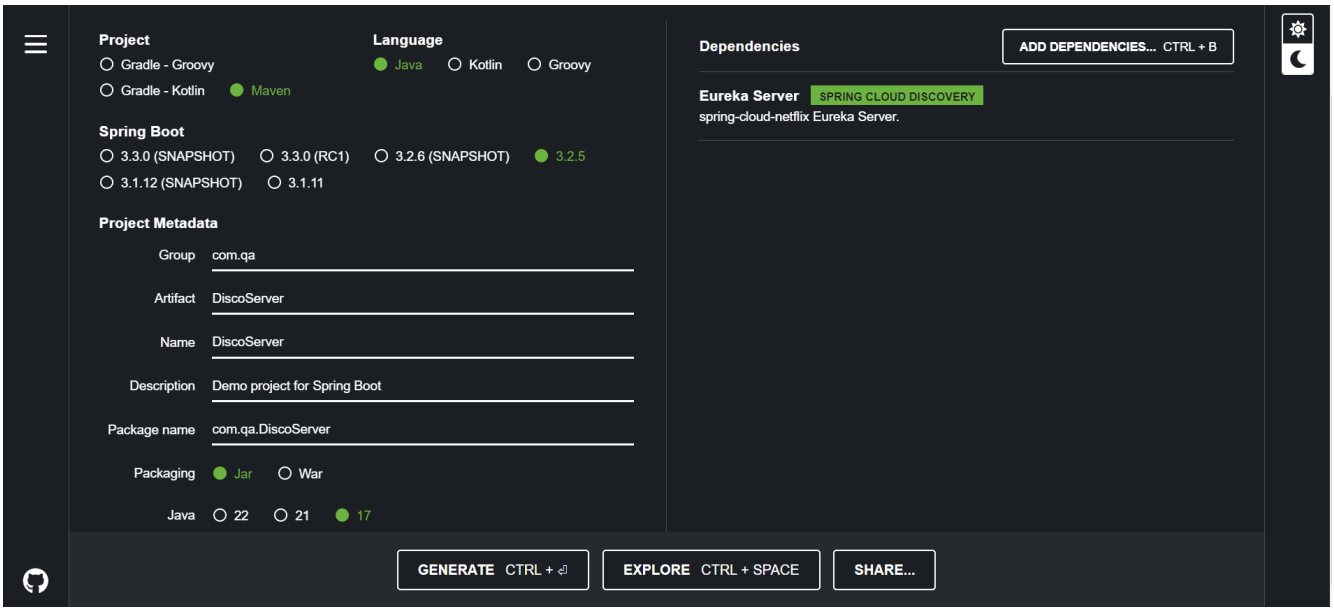
Lab 6 – Microservices

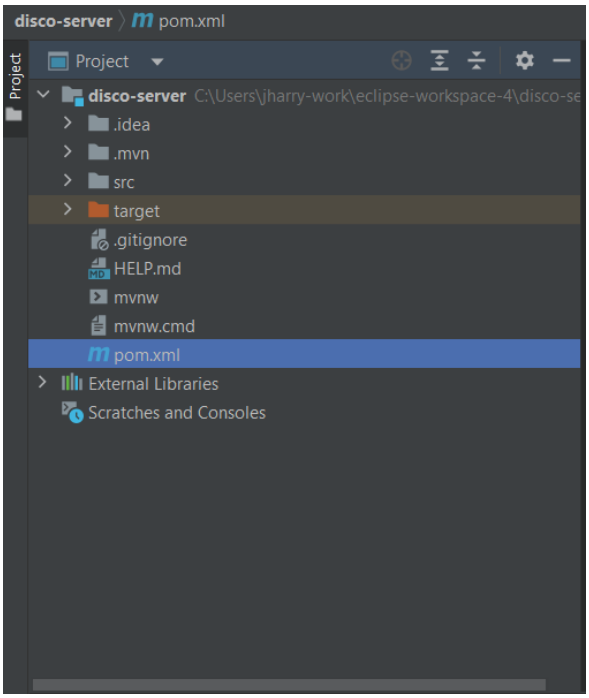
Task 1

1. Go to https://start.spring.io/
2. Generate a new Spring project with this configuration:



(Spring Boot should be the newest non-snapshot version)

3. Import this project into IntelliJ

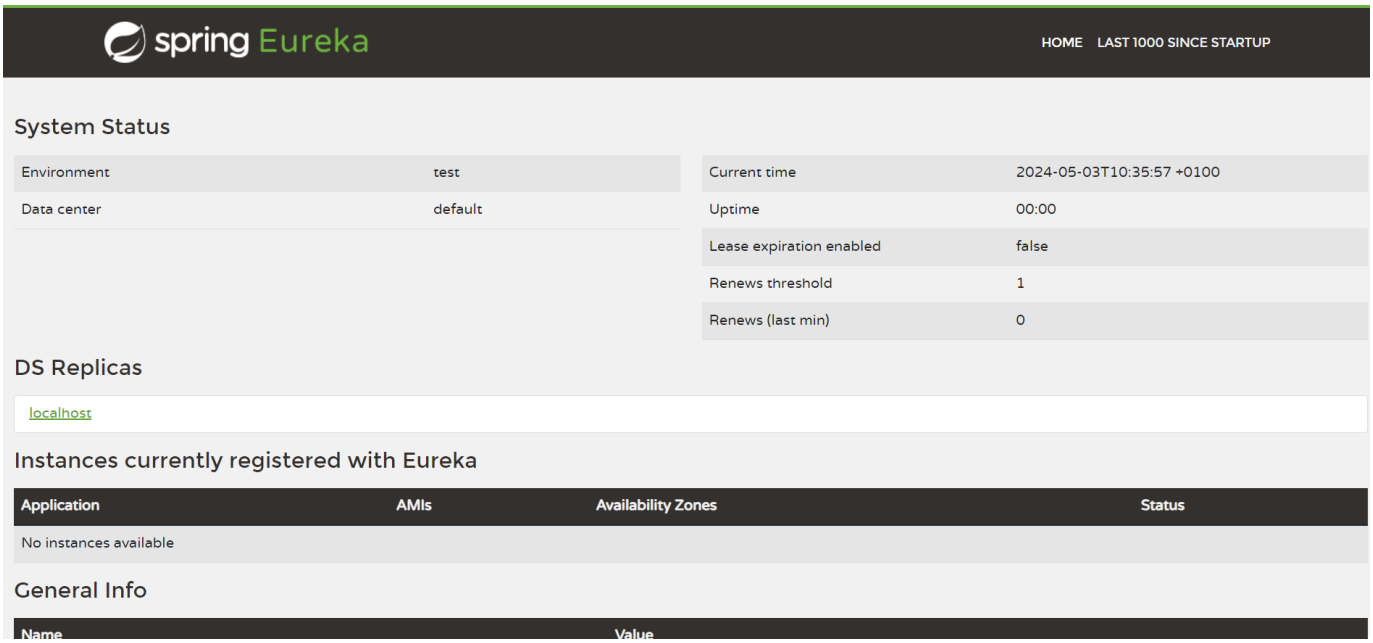


4. Open the **application.properties** file and insert these properties:

```
spring.application.name=disco-server

server.port=9999

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF
```

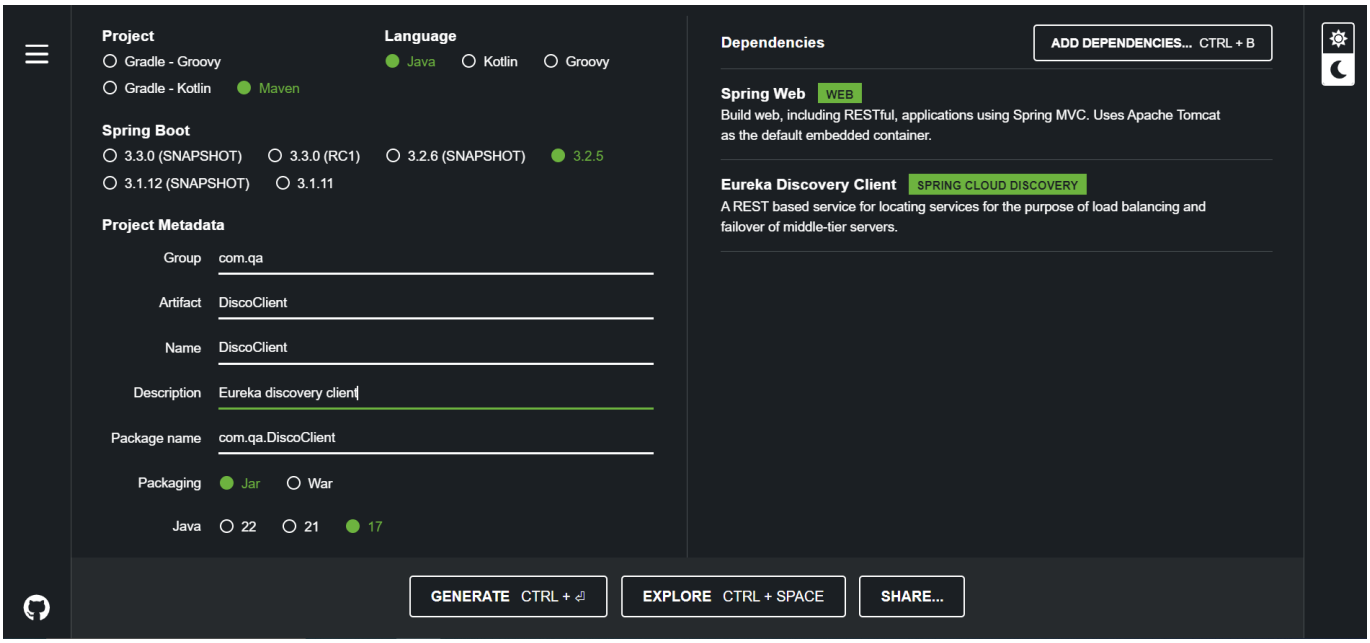
This will instruct Spring to setup the discovery server on an arbitrary port (9999). It will also prevent the discovery server searching for backups, which you would have in a real-world example but are omitted here for simplicity.

5. Go to http://localhost:9999 and view the Eureka landing page

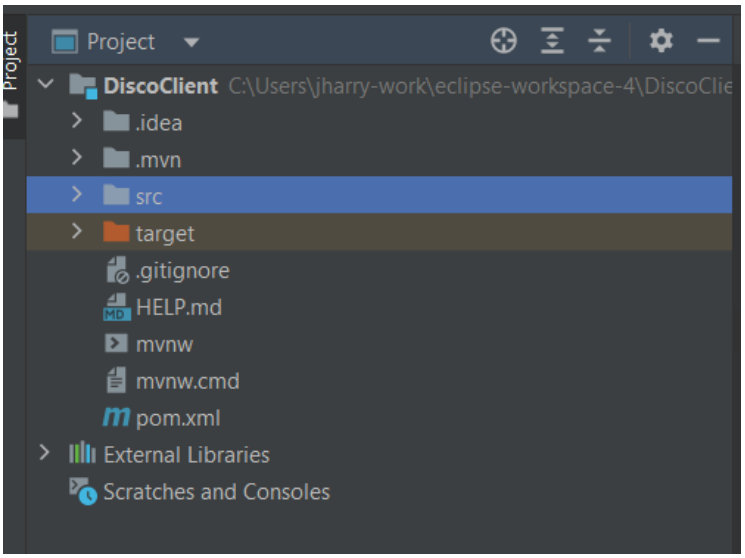


From here it is possible to view the status of any client connected to the discovery server. As of yet there are no clients to view so the next step is to set one up.

6. Go back to https://start.spring.io/ and generate a project with this configuration:



7. Import this project into IntelliJ and open it in a new window

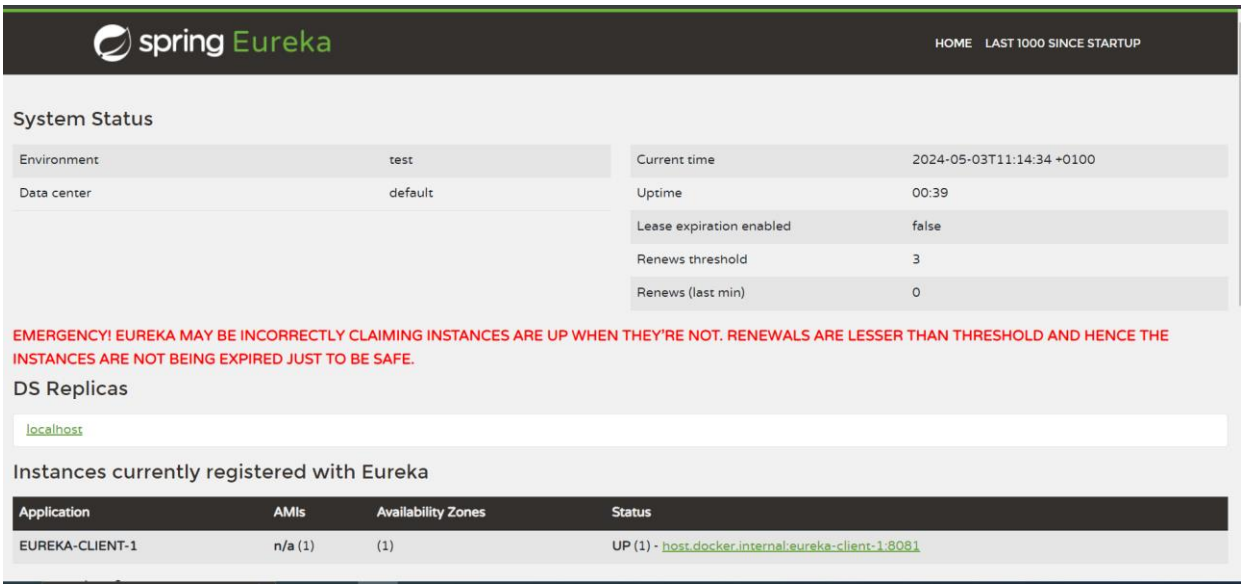


8. Add these properties to the **application.properties** file:

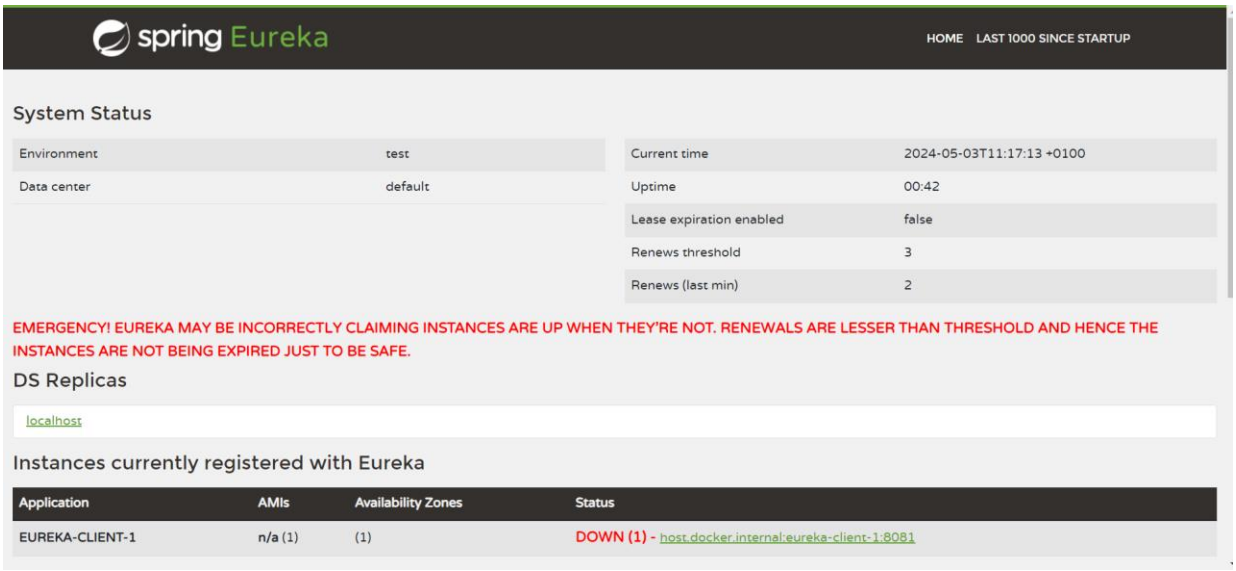
```
spring.application.name=eureka-client-1
server.port=8081

eureka.client.serviceUrl.defaultZone=http://localhost:9999/eureka/
```

This will set the application name (which will be used to register the application with the discovery server) and the location of the discovery server (defaults to localhost:8761)

9. Run the new application and check the discovery server again. You should now see the new client listed on the page



10. Stop the client and recheck the server – you should see that its status changes to **DOWN**

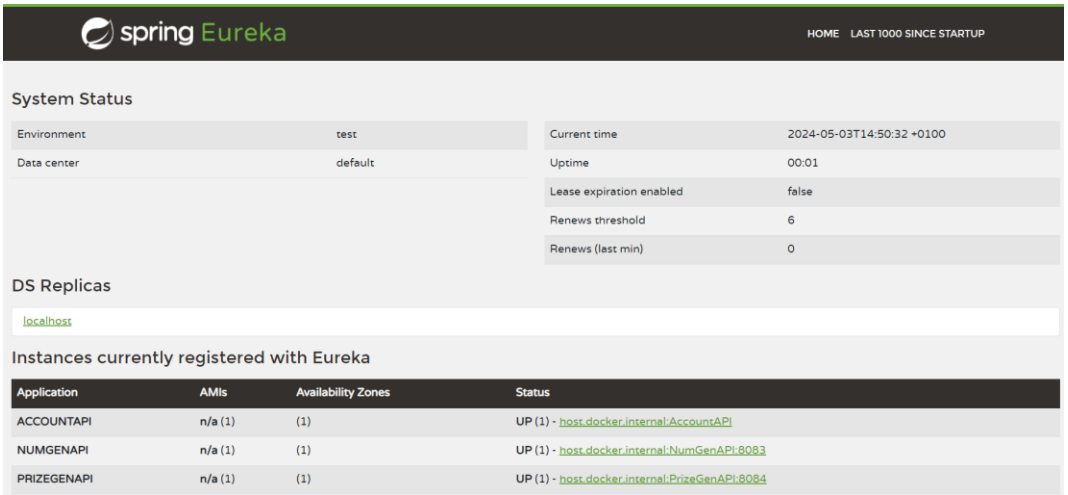


Task 2

Your instructor will provide you with the three starter projects for this task. These projects are:

- **AccountAPI** -> allows for the creation of accounts with a randomly generated account number. Each account is assigned a prize based on the randomly generated number.
- **NumGenAPI** -> responsible for generating the random number used by the **AccountAPI.**
- **PrizeGenAPI** -> responsible for calculating what, if any, prize should be given to each account upon creation.

1. Open all three projects in IntelliJ and run them.
2. You should now be able to see all of them in the Eureka discovery server:



3. Now that all of the microservices are running they need to be able to communicate with each other. In the **AccountAPI** project open the **AccountService**

```java
@Service
public class AccountService {

    @Autowired
    private AccountRepo repo;

    public Account register(Account newAccount) {
        // add call to number generator


        // add call to prize calculator

        return this.repo.save(newAccount);
    }
}
```

This is where the requests to the number and prize generators will go.

4. Add an instance of **RestClient** to the **AccountService**:

```java
2 usages
@Service
public class AccountService {

    1 usage
    @Autowired
    private AccountRepo repo;

    1 usage
    private RestClient restClient = RestClient.create();
```

RestClient allows Spring to send requests to other API's; in this case it will be sending requests to the **NumGenAPI** and the **PrizeGenAPI**.

5. In the **register** method use the **restClient** to send a GET request to the **NumGenAPI** and retrieve the response body as a String:

```
1 usage
public Account register(Account newAccount) {
    // add call to number generator
    String accountNumber = restClient
            .get() RequestHeadersUriSpec<capture of ?>
            .uri( uri: "http://localhost:8083/generateNumber") capture of ?
            .retrieve() ResponseSpec
            .body(String.class);
```

6. Set the account number of **newAccount** to be the account number retrieved from the **NumGenAPI**

```
public Account register(Account newAccount) {
    // add call to number generator
    String accountNumber = restClient
            .get() RequestHeadersUriSpec<capture of ?>
            .uri( uri: "http://localhost:8083/generateNumber")
            .retrieve() ResponseSpec
            .body(String.class);

    newAccount.setAccountNumber(accountNumber);
```

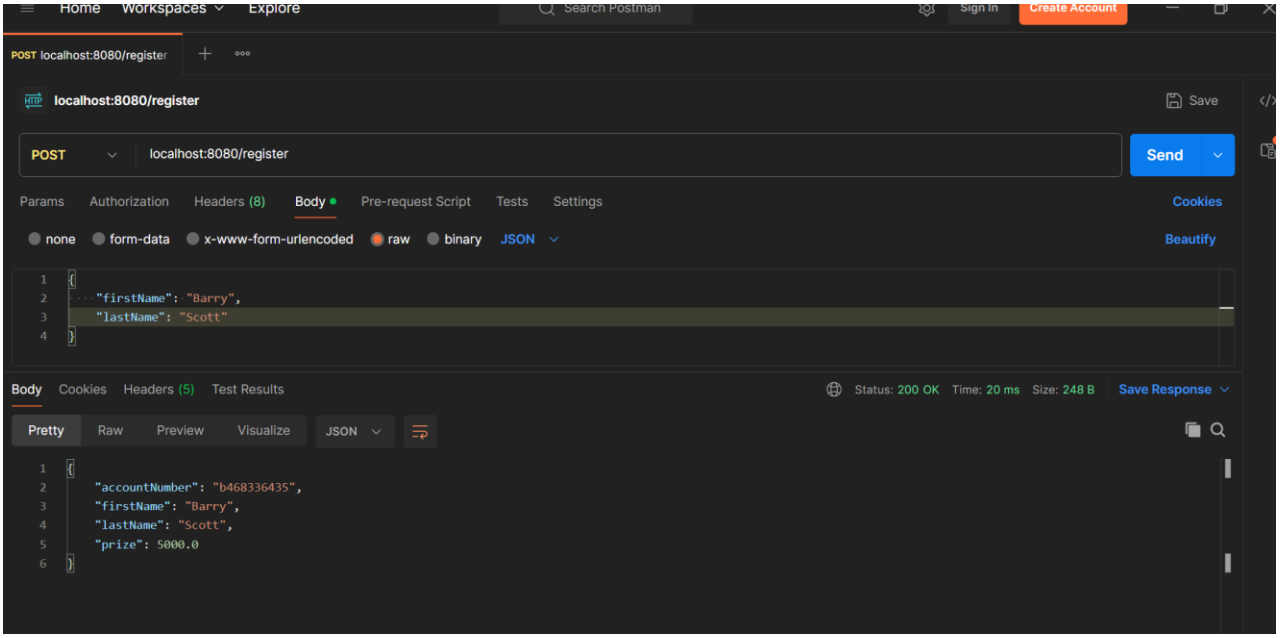
7. Now that you have the account number do the same thing with the **PrizeGenAPI,** except retrieving a double this time:

```
// add call to prize calculator

Double prize = restClient
        .get() RequestHeadersUriSpec<capture of ?>
        .uri( uri: "http://localhost:8084/generatePrize/" + accountNumber)
        .retrieve() ResponseSpec
        .body(Double.class);
```

8. Set the prize of **newAccount** to be the prize retrieved from the **PrizeGenAPI**

```
// add call to prize calculator

Double prize = restClient
        .get() RequestHeadersUriSpec<capture of ?>
        .uri( uri: "http://localhost:8084/generatePrize/" + accountNumber)
        .retrieve() ResponseSpec
        .body(Double.class);

newAccount.setPrize(prize);
```

9. Finally, test it out by sending a POST request to the **AccountAPI** using Postman:



Task 3

In the last task you created a basic microservice application but at the moment it's reliant on hardcoded URI's for the **NumGenAPI** and **PrizeGenAPI**. This makes an application extremely fragile as any instance falling over or being moved would cause the whole thing to fall over. Your application can be made a lot more resilient through the use of the discovery server created in task 1.

1. Inject an instance of **EurekaClient** into the **AccountService**

```
@Service
public class AccountService {

    1 usage
    @Autowired
    private AccountRepo repo;   repo: "org.springframework

    1 usage
    @Autowired
    private EurekaClient eurekaClient;   eurekaClient: "or

    2 usages
    private RestClient restClient = RestClient.create();
```

2. Use the **eurekaClient** to find the next available instance of the **NumGenAPI**:

```
1 usage
public Account register(Account newAccount) {   newAccount: Account@12281
    // add call to number generator
    String numGenUri = this.eurekaClient   numGenUri: "http://host.docker.inte
            .getNextServerFromEureka( virtualHostname: "NumGenAPI",  secure: false)
            .getHomePageUrl();
```

Note: The 'hostname' of a eureka client is the application name set in the **application.properties**

3. Use this URI in the **uri()** method of the **eurekaClient**:

```
public Account register(Account newAccount) {   newAccount: Account@12281
    // add call to number generator
    String numGenUri = this.eurekaClient   numGenUri: "http://host.docker.inte
            .getNextServerFromEureka( virtualHostname: "NumGenAPI",  secure: false)
            .getHomePageUrl();
    String accountNumber = restClient   accountNumber: "b972203842"    restClie
            .get() RequestHeadersUriSpec<capture of ?>
            .uri( uri: numGenUri + "/generateNumber") capture of ?   numGenUri: "htt
            .retrieve() ResponseSpec
            .body(String.class);
```

4. Now do the same thing with the **PrizeGenAPI**:

```
String prizeGenUri = this.eurekaClient
        .getNextServerFromEureka( virtualHostname: "PrizeGenAPI",  secure: false)
        .getHomePageUrl();
Double prize = restClient
        .get() RequestHeadersUriSpec<capture of ?>
        .uri( uri: prizeGenUri + "/generatePrize/" + accountNumber) capture of ?
        .retrieve() ResponseSpec
        .body(Double.class);

newAccount.setPrize(prize);
```

5. Finally, test it still works in Postman.