Noroff
School of technology
and digital media

Tuesday - Week 3

SQL to REST

What is a REST service?

- Representational State Transfer
- Communications occur without the knowledge of state
- A resource is just a URL

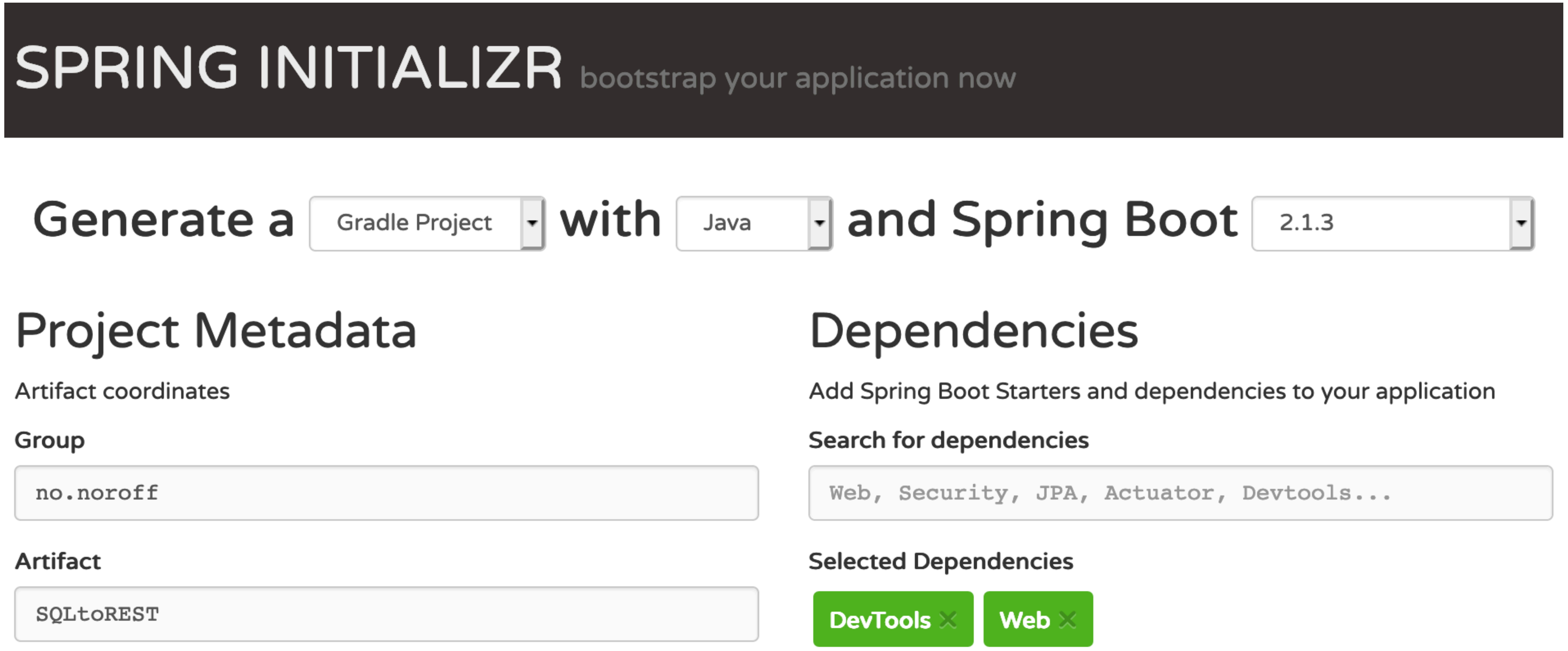
Why is it useful?

- All responses can be simple HTTP status codes
- It’s language independent, out client can ***easily*** be a different language to the server
- Data is just Json …
- Security is handled with layers not technologies
- REST is everywhere

Noroff
School of technology
and digital media

Building a RESTful service

https://start.spring.io/



SPRING INITIALIZR bootstrap your application now

Generate a Gradle Project with Java and Spring Boot 2.1.3

Project Metadata

Artifact coordinates

Group

no.noroff

Artifact

SQLtoREST

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

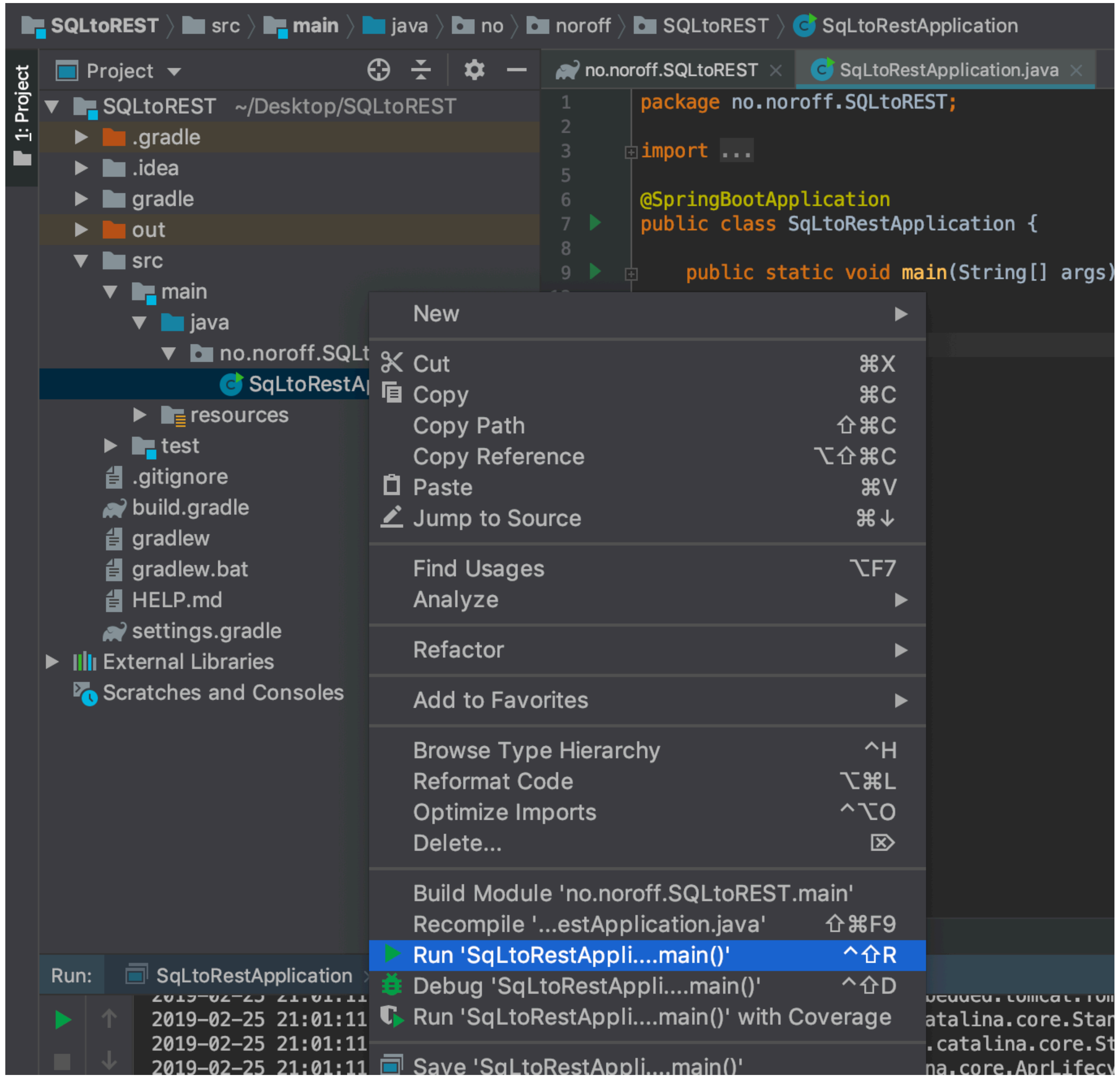
Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

DevTools ✕ Web ✕

Launch it

- Download
- Open
- And Launch!

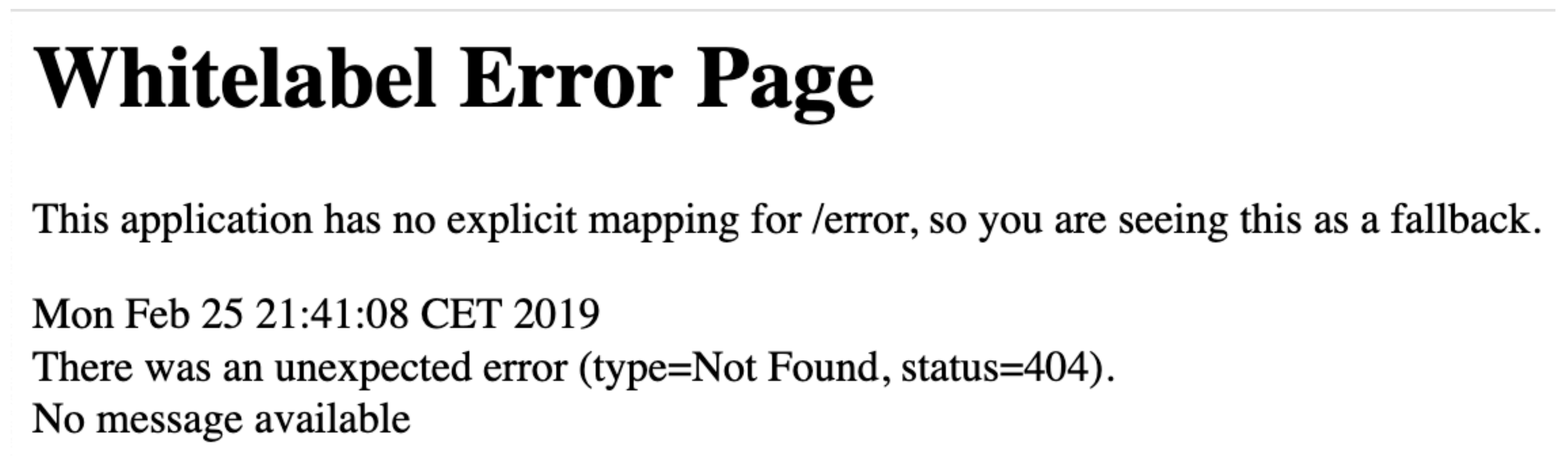


```
package no.noroff.SQLtoREST;

import ...

@SpringBootApplication
public class SqLtoRestApplication {

    public static void main(String[] args)
```

```
2019-02-25 21:01:11
2019-02-25 21:01:11
2019-02-25 21:01:11
2019-02-25 21:01:11
```

localhost:8080

- Whitelabel Error Page
- This means it's working but there is nothing there
- (not even a real error page)



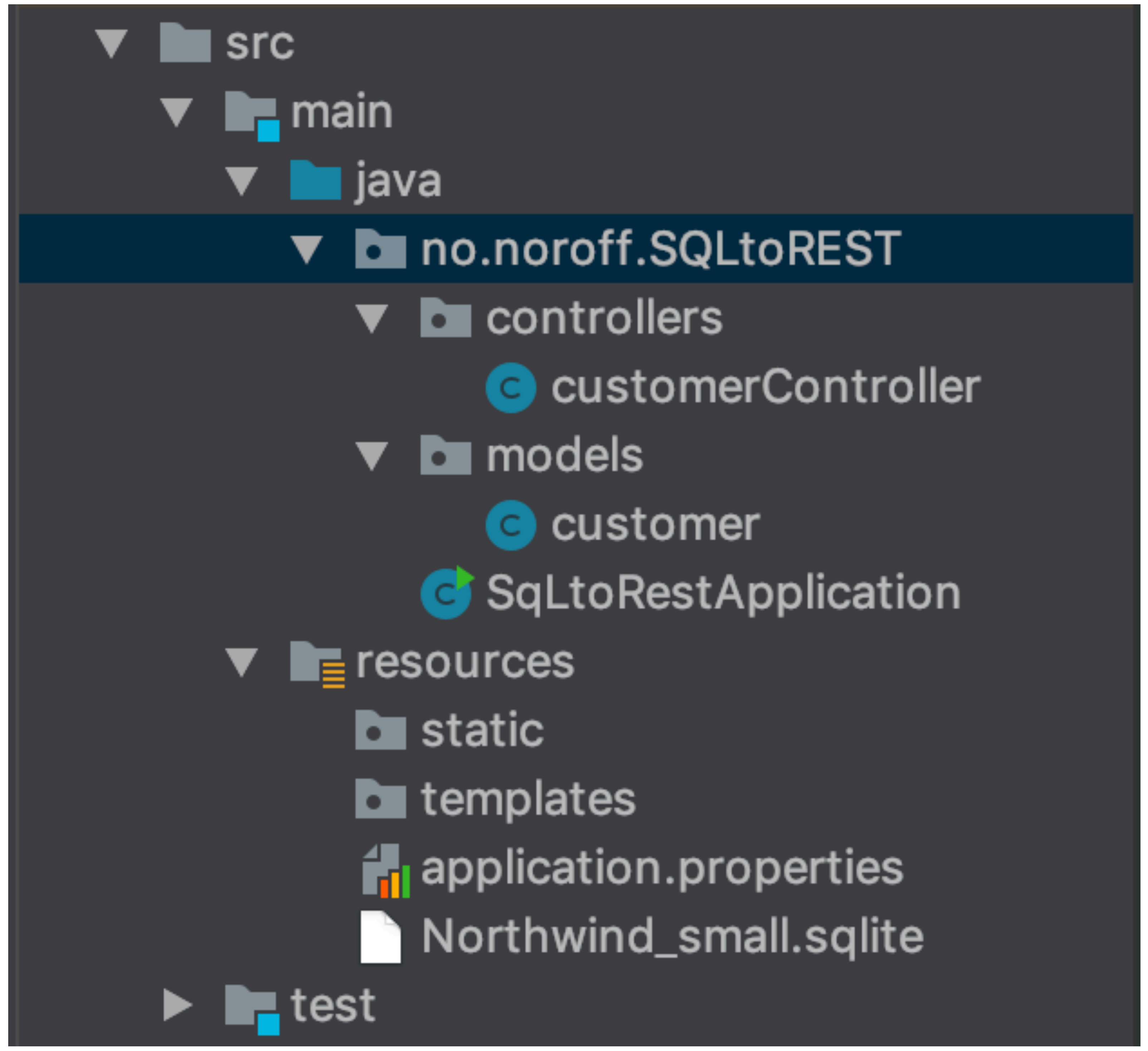
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

```
Mon Feb 25 21:41:08 CET 2019
There was an unexpected error (type=Not Found, status=404).
No message available
```

Project Overview

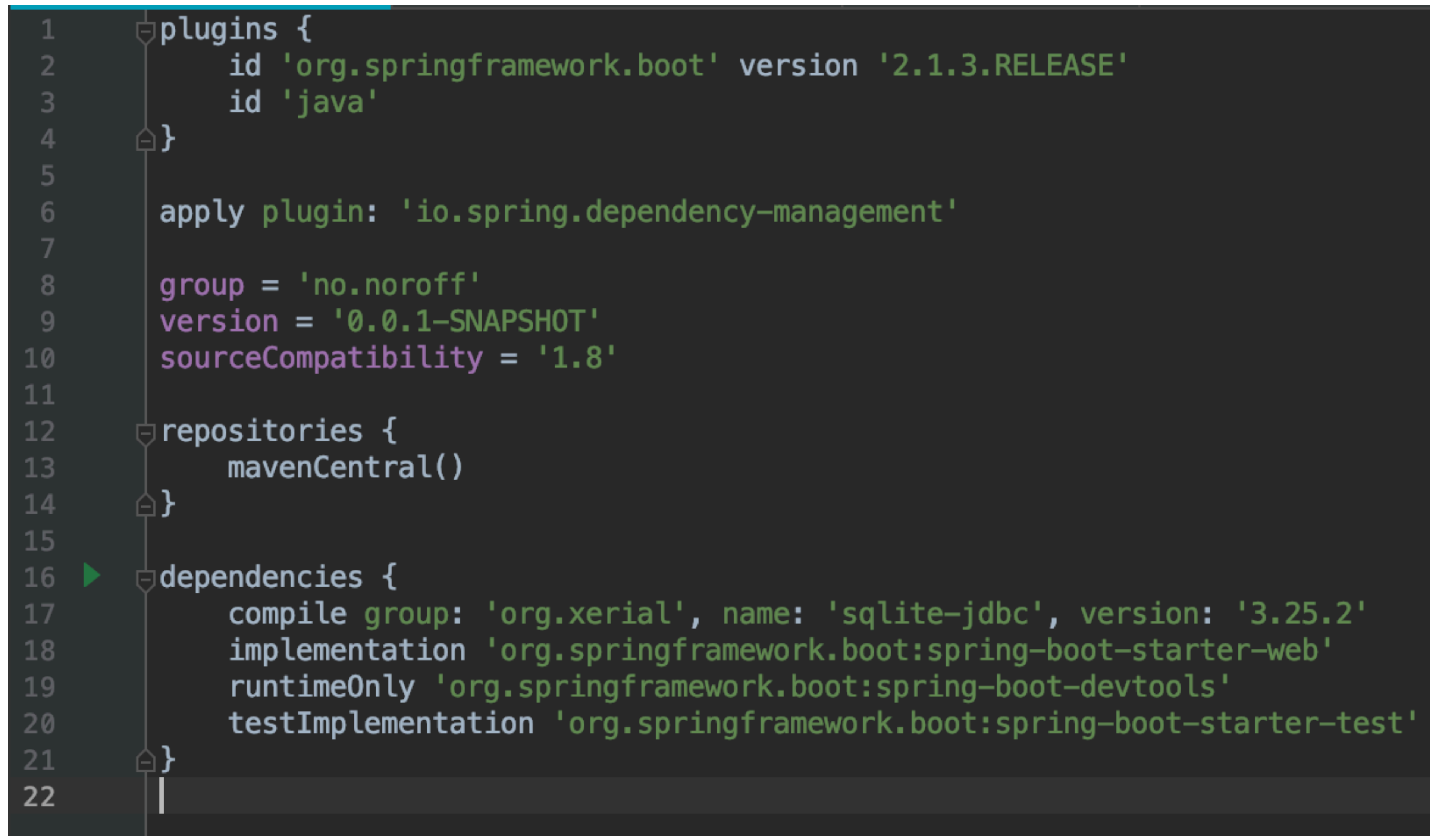
- Customer Controller
- Customer
- Main Application
- The Database



```
▼ src
  ▼ main
    ▼ java
      ▼ no.noroff.SQLtoREST
        ▼ controllers
            customerController
        ▼ models
            customer
          SqLtoRestApplication
    ▼ resources
        static
        templates
        application.properties
        Northwind_small.sqlite
  ▶ test
```

build.gradle

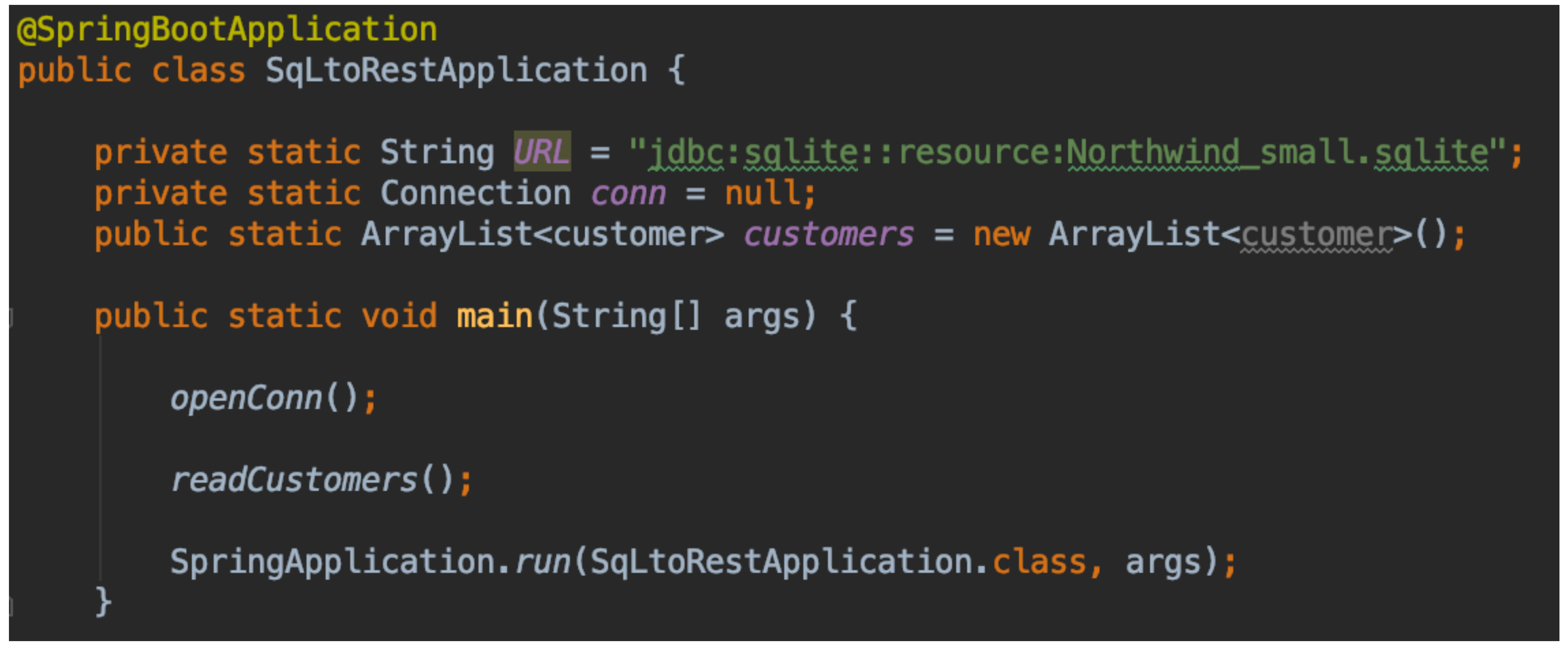
- The only thing I need to add is the “sqlite-jdbc” dependency



```
plugins {
    id 'org.springframework.boot' version '2.1.3.RELEASE'
    id 'java'
}

apply plugin: 'io.spring.dependency-management'

group = 'no.noroff'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

repositories {
    mavenCentral()
}

dependencies {
    compile group: 'org.xerial', name: 'sqlite-jdbc', version: '3.25.2'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    runtimeOnly 'org.springframework.boot:spring-boot-devtools'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

The main application

- I read my database on starting the server (bad!)
- This is just for demo purposes
- You should use custom queries when the user accesses the API

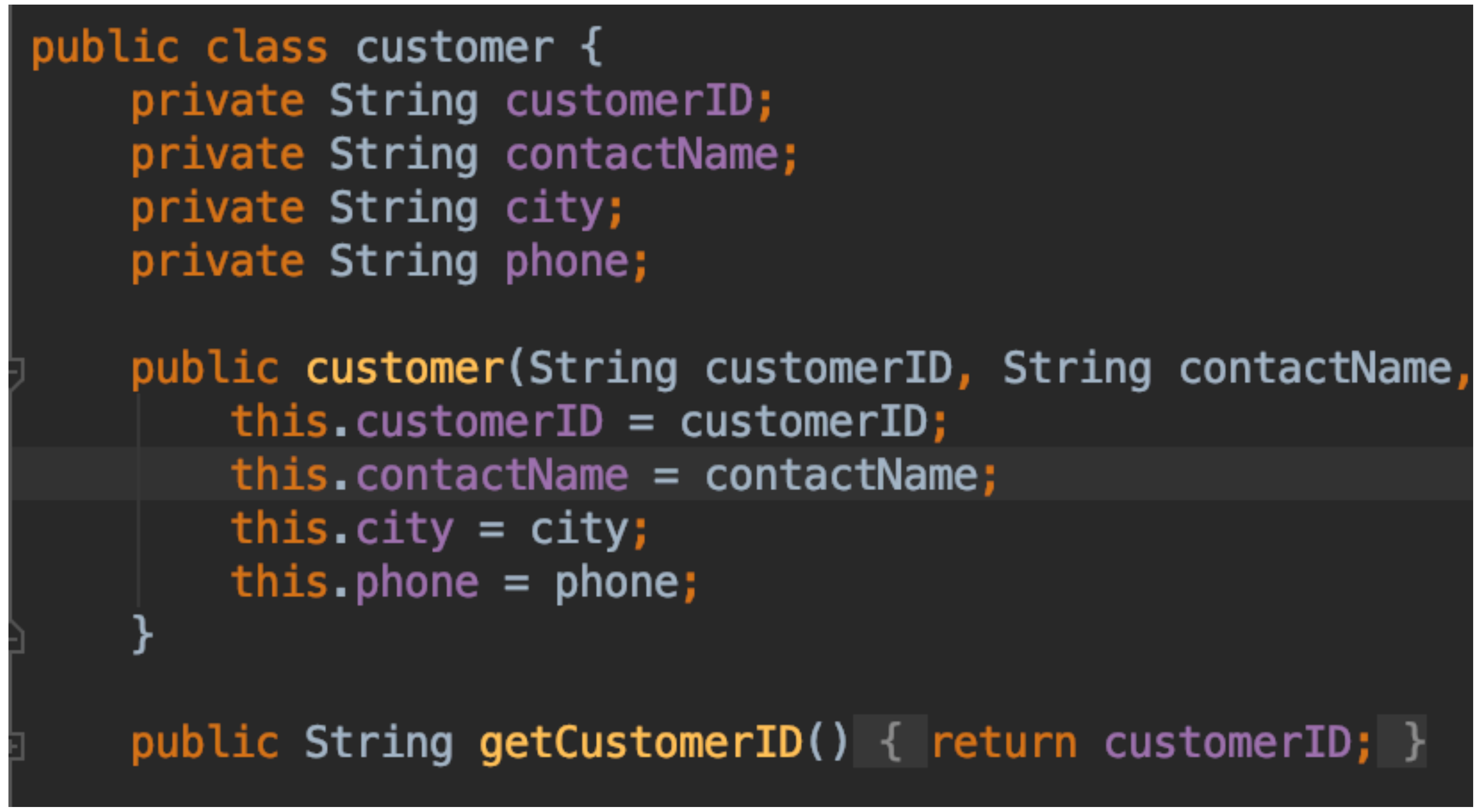


```java
@SpringBootApplication
public class SqLtoRestApplication {

    private static String URL = "jdbc:sqlite::resource:Northwind_small.sqlite";
    private static Connection conn = null;
    public static ArrayList<customer> customers = new ArrayList<customer>();

    public static void main(String[] args) {

        openConn();

        readCustomers();

        SpringApplication.run(SqLtoRestApplication.class, args);
    }
```

customer.java

- I create a customer class to match what I want the suer to be able to access

- I autogenerate getters and setters



```
public class customer {
    private String customerID;
    private String contactName;
    private String city;
    private String phone;

    public customer(String customerID, String contactName,
        this.customerID = customerID;
        this.contactName = contactName;
        this.city = city;
        this.phone = phone;
    }

    public String getCustomerID() { return customerID; }
```

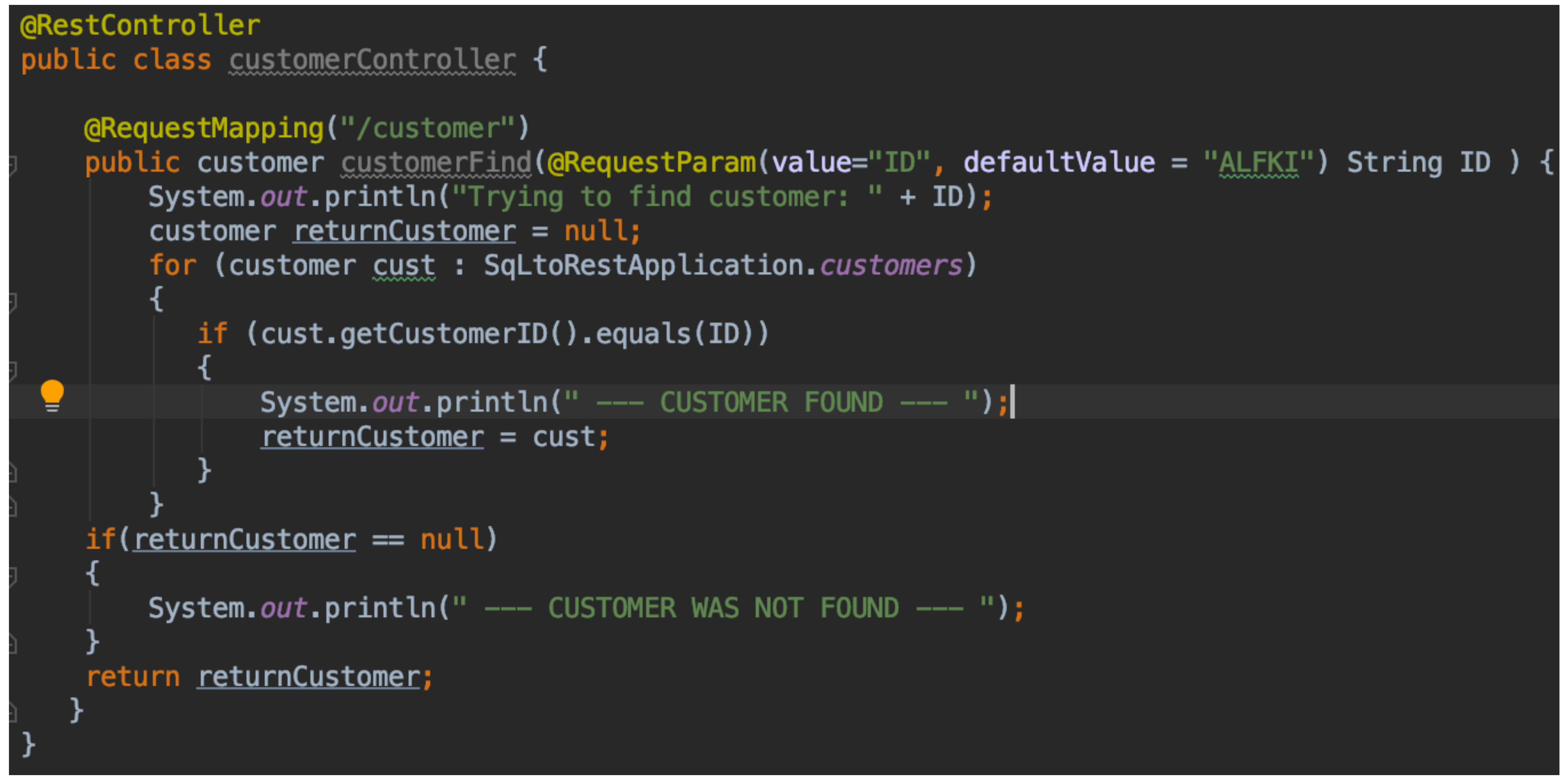
customerController.java

- Lots going on here!

- Rest Controller

- Request Mapping

- Request Parameter



```java
@RestController
public class customerController {

    @RequestMapping("/customer")
    public customer customerFind(@RequestParam(value="ID", defaultValue = "ALFKI") String ID ) {
        System.out.println("Trying to find customer: " + ID);
        customer returnCustomer = null;
        for (customer cust : SqLtoRestApplication.customers)
        {
            if (cust.getCustomerID().equals(ID))
            {
                System.out.println(" --- CUSTOMER FOUND --- ");
                returnCustomer = cust;
            }
        }
    if(returnCustomer == null)
    {
        System.out.println(" --- CUSTOMER WAS NOT FOUND --- ");
    }
    return returnCustomer;
   }
}
```

@RestController

- This marks the class as a whole as a Rest Controller

- Spring will know to search inside the class for mappings

@RequestMapping

- @RequestMapping(“/customer")
- Spring will use this mapping to create a route
  - This tells the web server how things are structured

- We could return anything we wanted at the route, but we want to find a customer so we will need a way to specify one…

@RequestParam

- This is a bit ‘odd’

```
public customer customerFind(@RequestParam(value="ID", defaultValue = "ALFKI") String ID ) {
 // method code
}
```

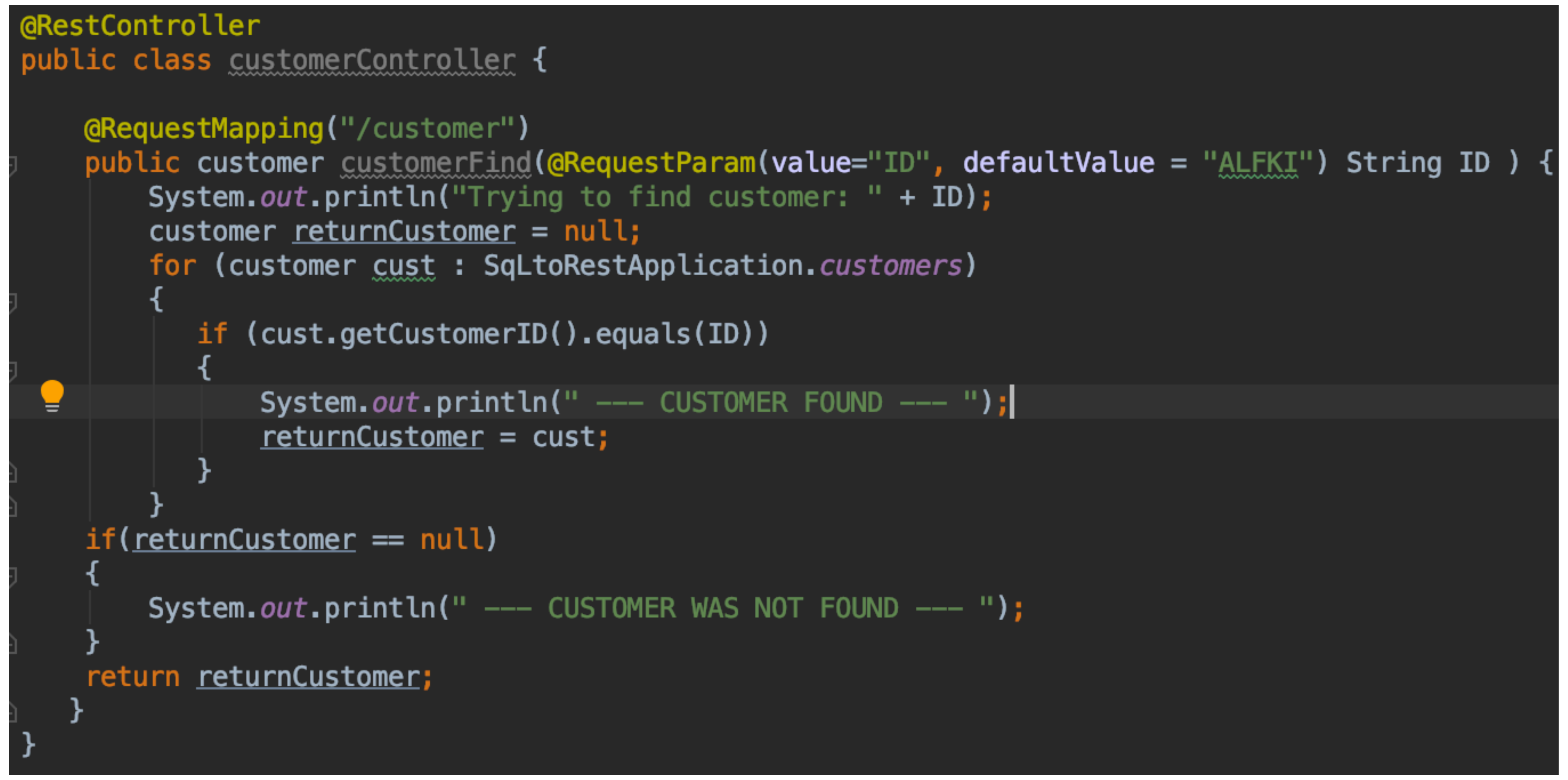
- Each argument is linked to a parameter

- IE:

  http://localhost:8080/customer?ID=BERGS

customerController.java

- The body of the code simply finds the requested customer and returns it



```java
@RestController
public class customerController {

    @RequestMapping("/customer")
    public customer customerFind(@RequestParam(value="ID", defaultValue = "ALFKI") String ID ) {
        System.out.println("Trying to find customer: " + ID);
        customer returnCustomer = null;
        for (customer cust : SqLtoRestApplication.customers)
        {
            if (cust.getCustomerID().equals(ID))
            {
                System.out.println(" --- CUSTOMER FOUND --- ");
                returnCustomer = cust;
            }
        }
    if(returnCustomer == null)
    {
        System.out.println(" --- CUSTOMER WAS NOT FOUND --- ");
    }
    return returnCustomer;
   }
}
```

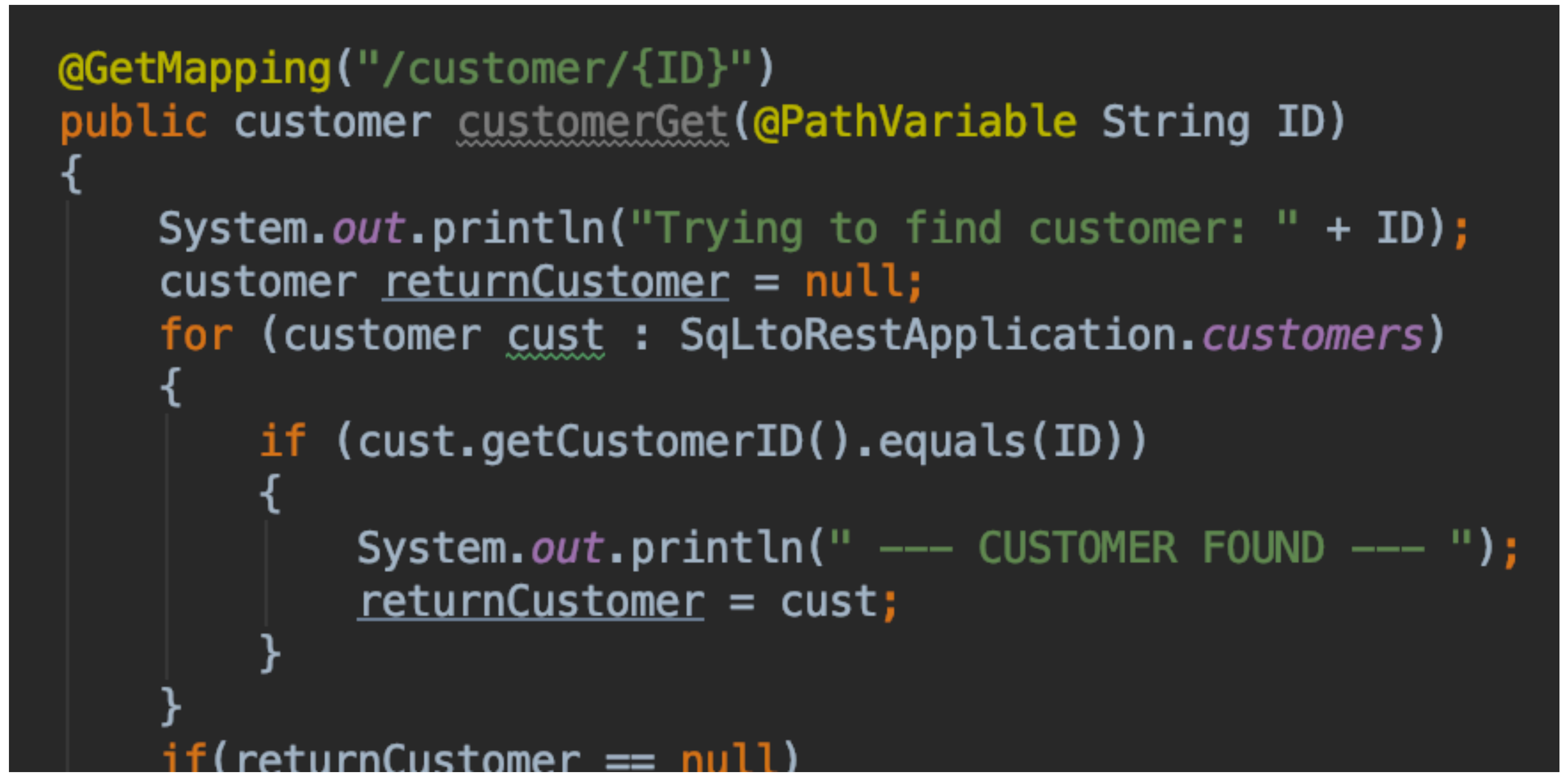
Did you notice the magic?

- Our browser displayed JSON!

- Spring it converting out POJO into JSON before sending it to the browser
- We didn’t have to do anything besides return the object

@GetMapping

- We specify the ID in the path using {ID}
- Then we define it as an @PathVariable
- The rest of the method is the same



```
@GetMapping("/customer/{ID}")
public customer customerGet(@PathVariable String ID)
{
    System.out.println("Trying to find customer: " + ID);
    customer returnCustomer = null;
    for (customer cust : SqLtoRestApplication.customers)
    {
        if (cust.getCustomerID().equals(ID))
        {
            System.out.println(" --- CUSTOMER FOUND --- ");
            returnCustomer = cust;
        }
    }
    if(returnCustomer == null)
```

But why the get mapping?

- The example from before:
  http://localhost:8080/customer?ID=BERGS

- Becomes:
  http://localhost:8080/customer/BERGS

Other mappings

- Obviously we need to be able to do all the CRUD operations

- We will cover these in class later, but they are explained in the links in your Tasks

Noroff
School of technology
and digital media

Tuesday - Week 3

Tasks

Optional Reading

- Some more thorough discussion of REST

  https://www.codecademy.com/articles/what-is-rest

Optional Practise

- A quick guided step-by-step: http://spring.io/guides/gs/rest-service

- A complete tutorial: http://spring.io/guides/tutorials/rest

We will continue this topic …

- For now I simply want you to be able to perform simple queries