

DevOps

TD-01 - Création d'un projet – Intégration et Déploiement manuelle

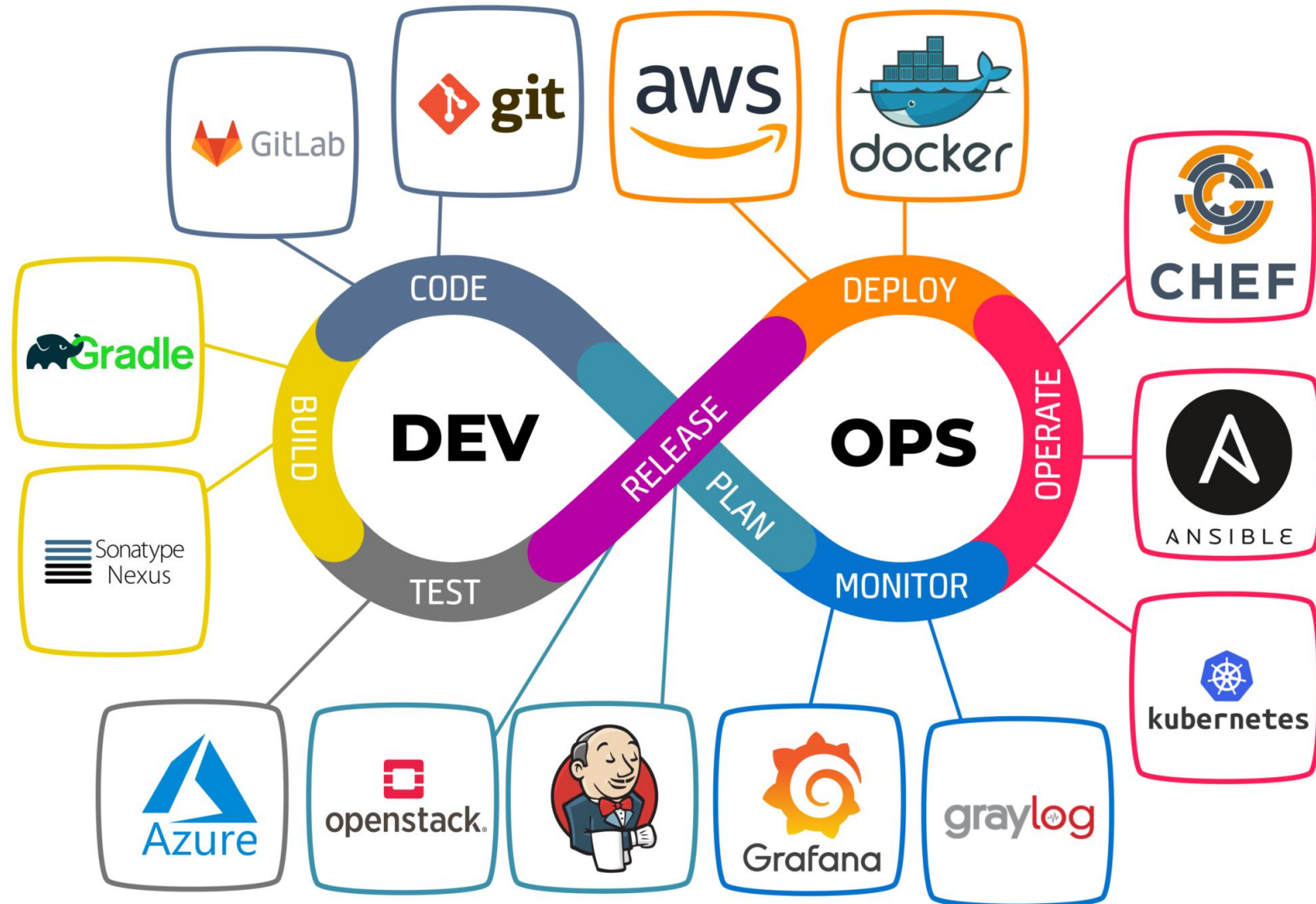
Jamel ESSOUSSI: Architecte logiciel

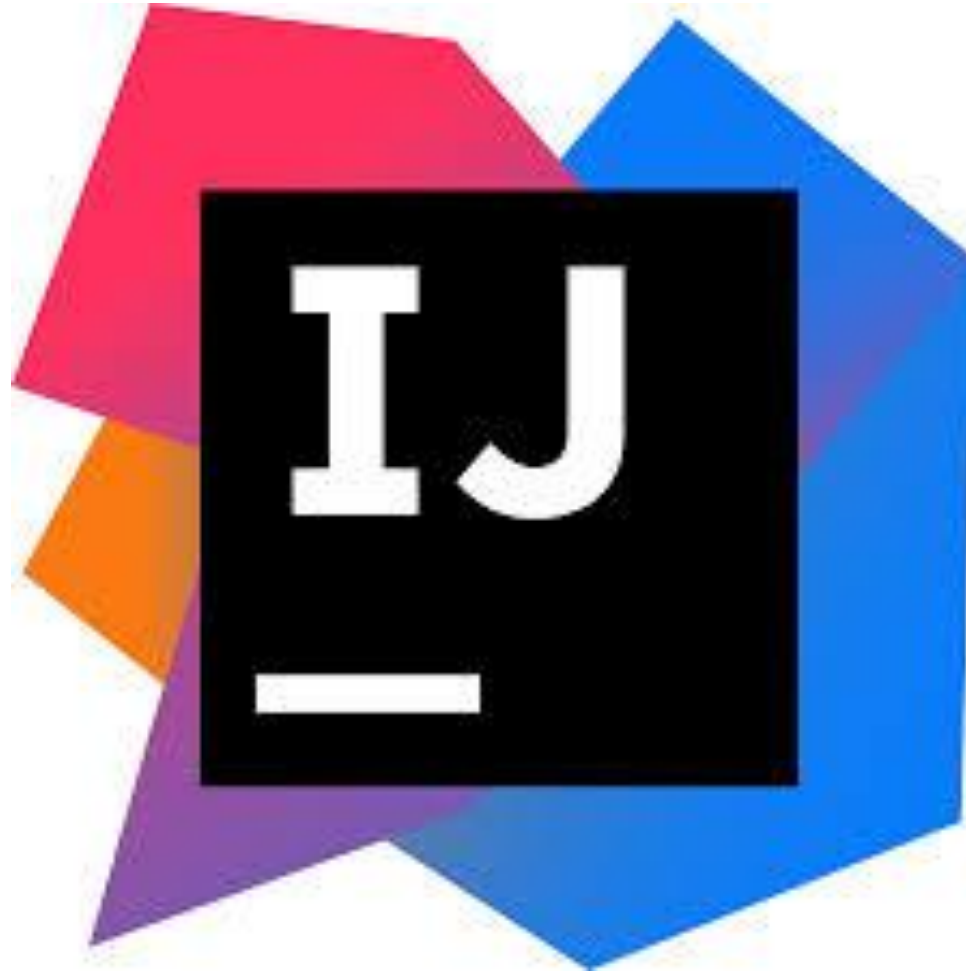
jamel.essoussi@gmail.com

<https://github.com/jessoussi>

2025

Outils du DevOps



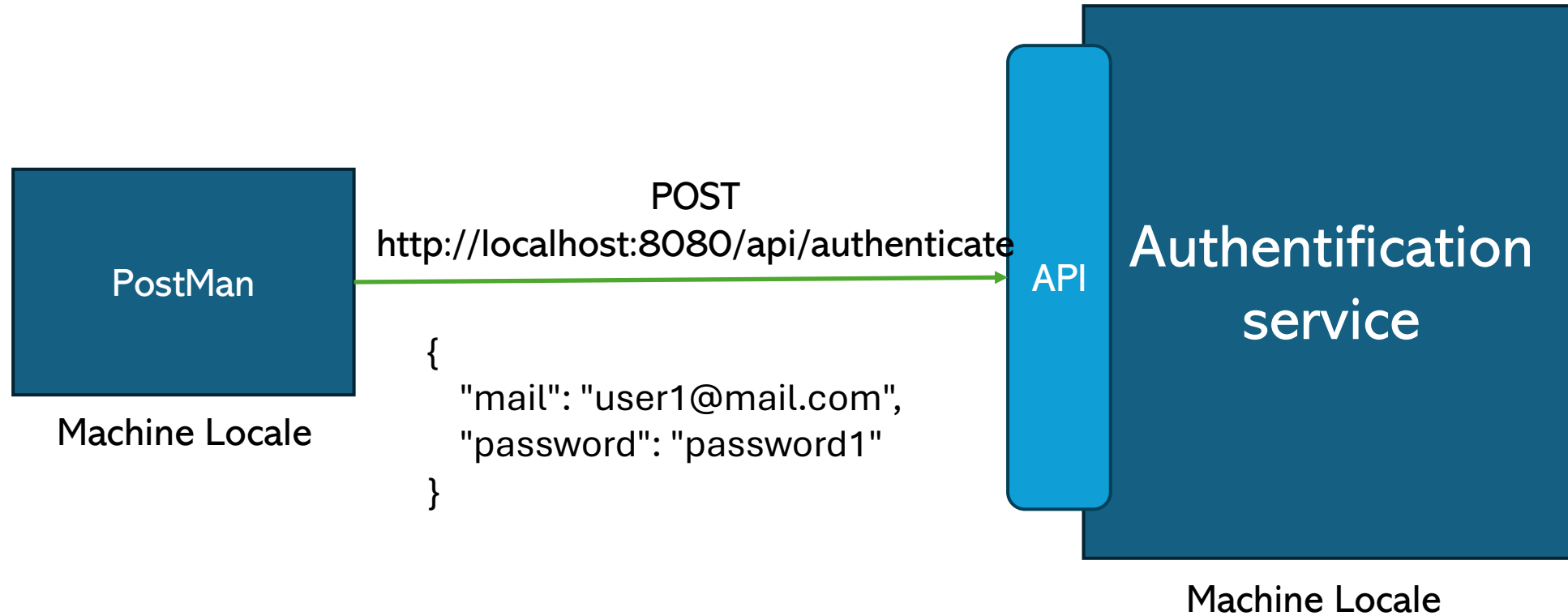


IntelliJ
IDE
PostMan

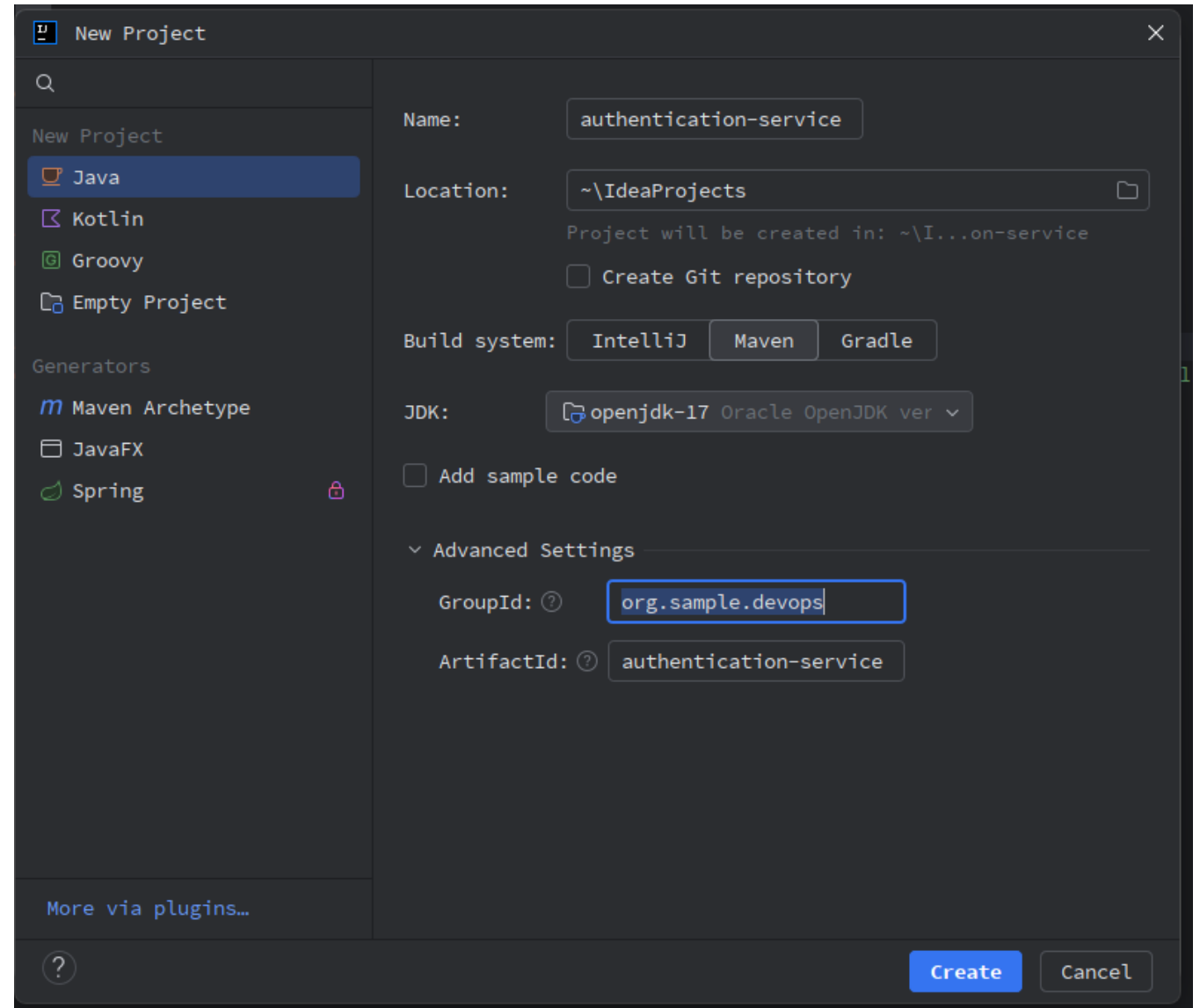
Objectif du TD

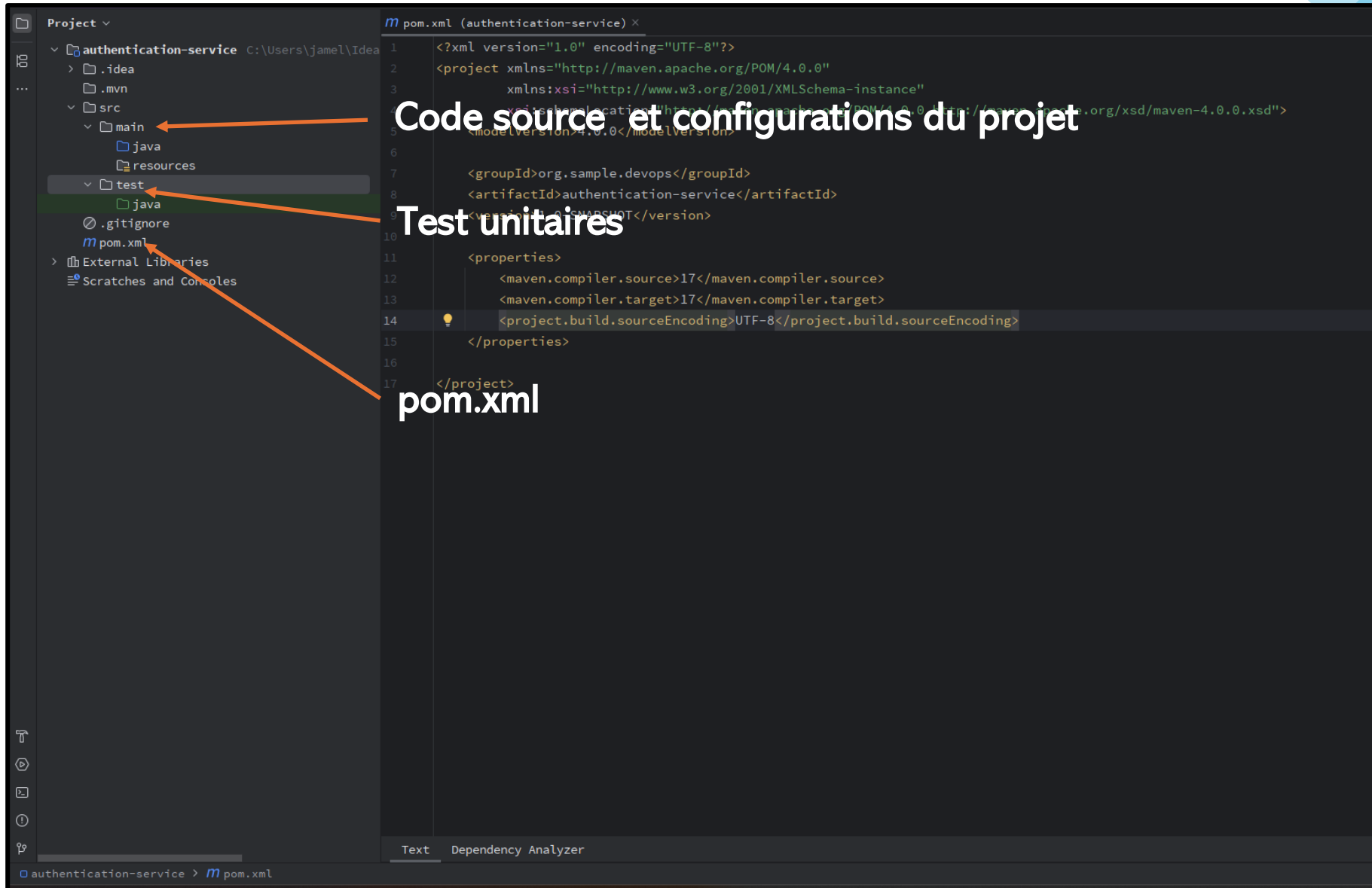
- ☐ Développer un service d'authentification basique en springboot
- ☐ Exposer un Service en API REST qui permet d'authentifier des utilisateurs,
- ☐ Un utilisateur est identifié par son mail et son mot de passe,
- ☐ La liste des utilisateurs est gérée dans une structure en mémoire,

Architecture du projet

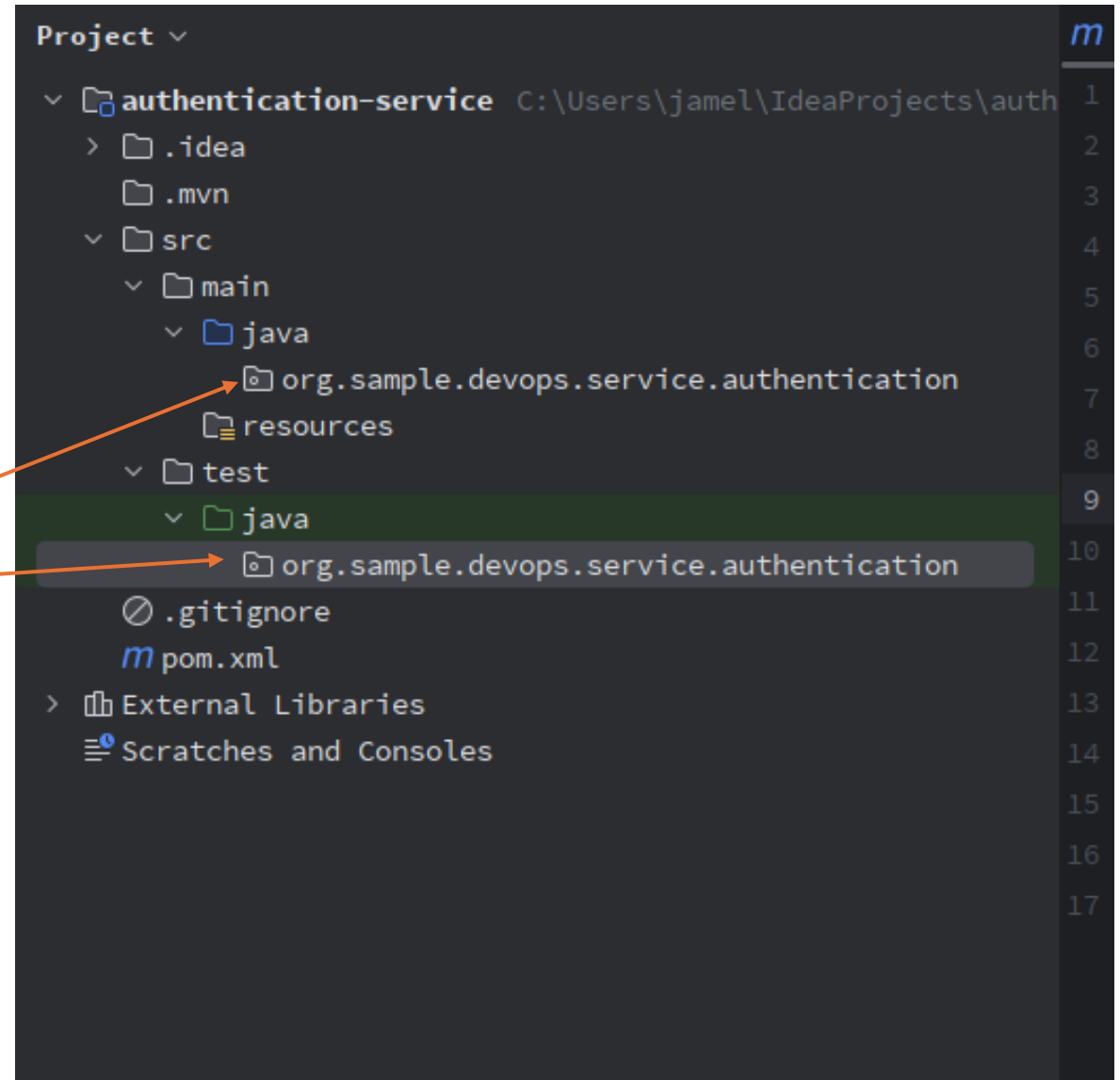


Création du projet





Création des packages



Ajoutez les dépendances SpringBoot

Spring initializr

<https://start.spring.io/>

Caractéristique
s du projet

The screenshot shows the Spring Initializr web application interface. It is divided into several sections:

- Project:** Includes options for **Language** (Java, Kotlin, Groovy) and **Spring Boot** versions (4.0.0 (SNAPSHOT), 4.0.0 (M3), 3.5.7 (SNAPSHOT), 3.5.6, 3.4.11 (SNAPSHOT), 3.4.10). The **Project Metadata** section contains fields for Group, Artifact, Name, Description, and Package name, along with **Packaging** options (Jar, War) and **Java** versions (25, 21, 17).
- Dependencies:** Includes a button **ADD DEPENDENCIES... CTRL + B** and a list of dependencies, currently showing **Spring Web** with a **WEB** tag and a description: "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container."
- Footer:** Contains buttons **GENERATE CTRL + G**, **EXPLORE CTRL + SPACE**, and a **...** button.

Annotations with orange arrows point to specific parts of the interface:

- An arrow points from the text "Caractéristique s du projet" to the **Project** section.
- An arrow points from the text "Ajoutez Spring web" to the **Spring Web** dependency entry.
- An arrow points from the text "Générez les dépendances" to the **GENERATE** button.

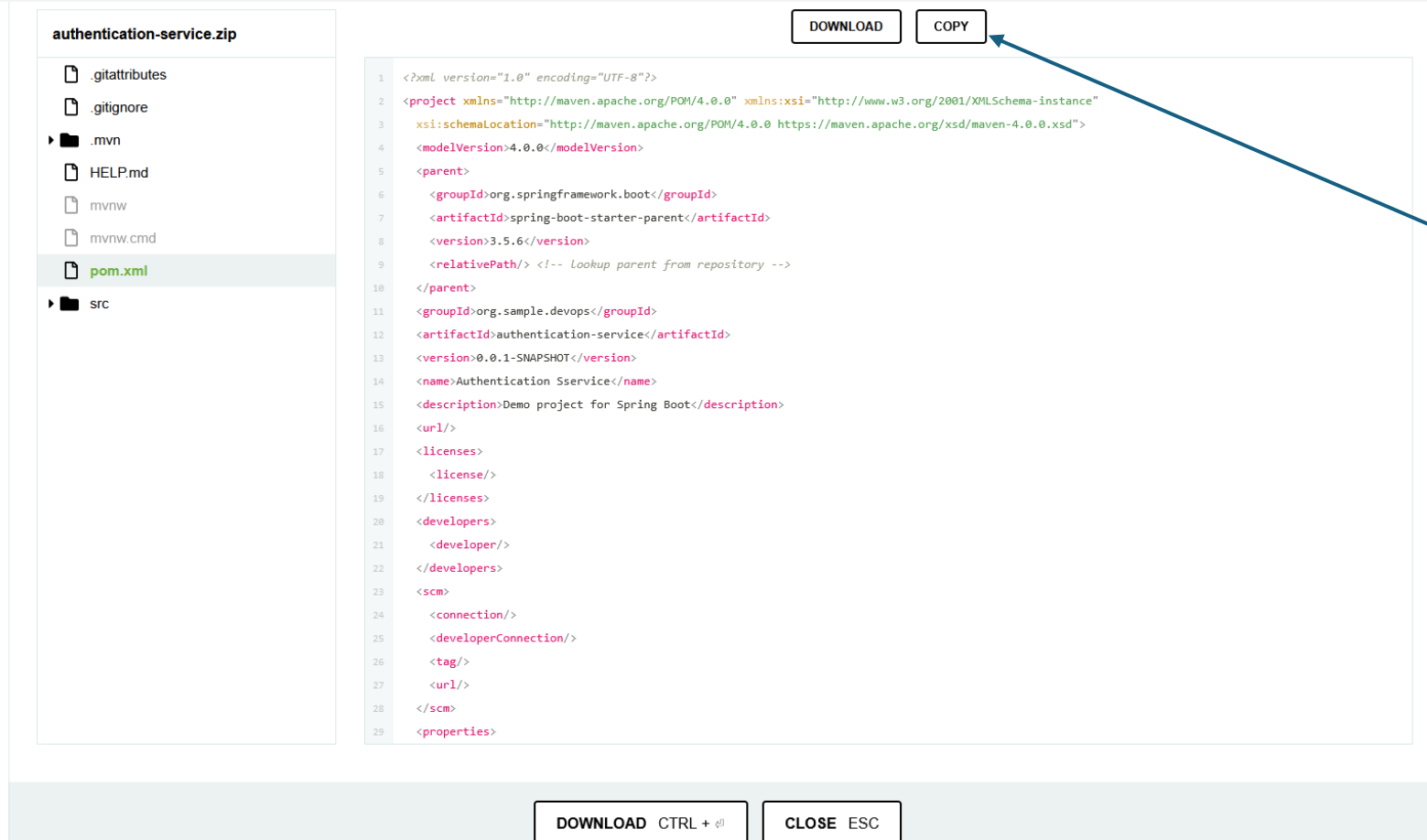
Ajoutez Spring
web

Générez les
dépendances

Générez les dépendances SpringBoot

Spring initializr

<https://start.spring.io/>



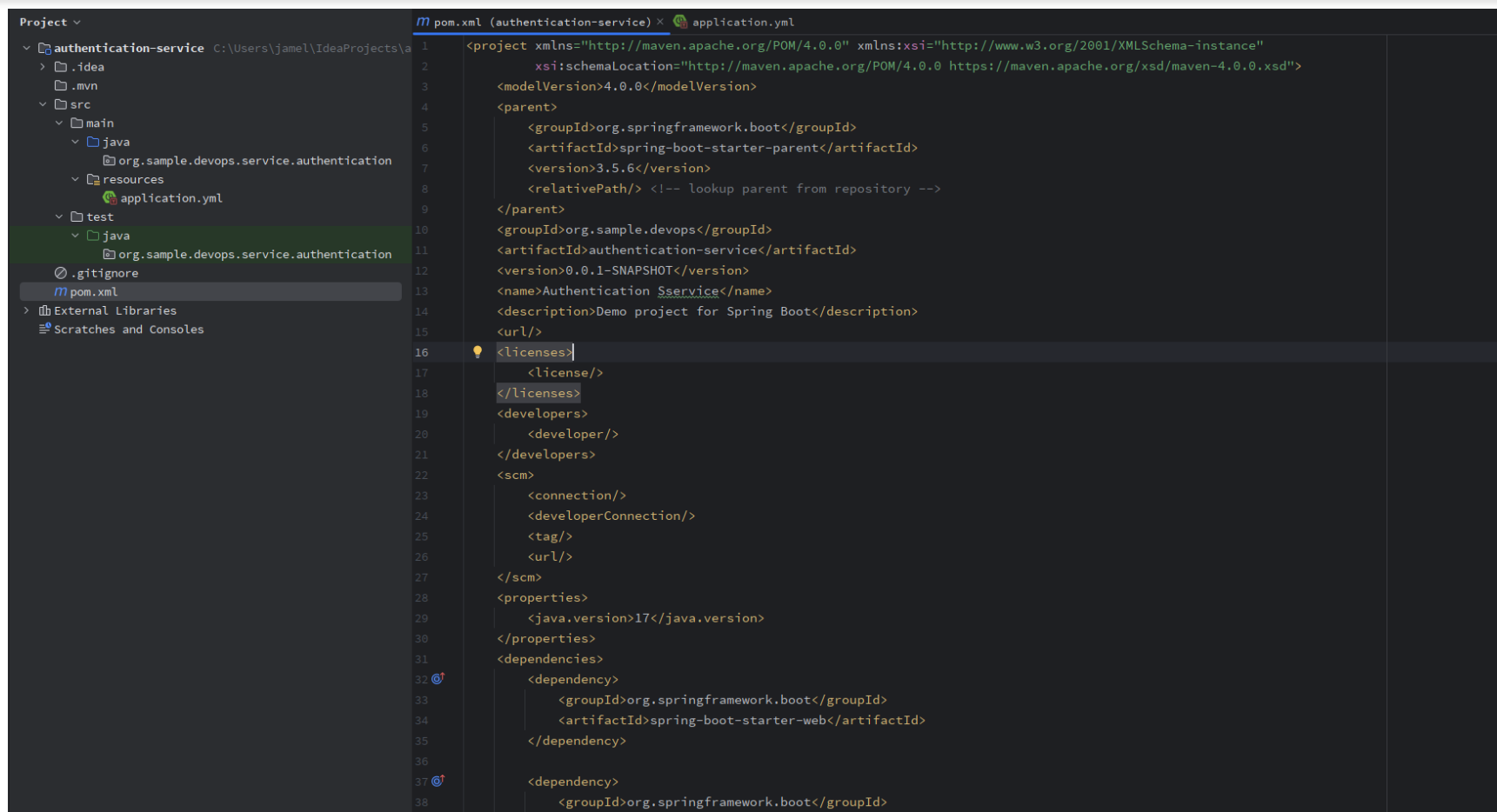
The screenshot displays the Spring Initializr interface. On the left, a file explorer shows the project structure for 'authentication-service.zip', including files like .gitattributes, .gitignore, .mvn, HELP.md, mvnw, mvnw.cmd, and pom.xml (highlighted), and a src directory. The main area shows the generated pom.xml code, which includes dependencies for Spring Boot starter parent and authentication-service. At the top right, there are 'DOWNLOAD' and 'COPY' buttons. A blue arrow points from the 'COPY' button to the text 'Copiez dans le pom.xml de votre projet'. At the bottom, there are 'DOWNLOAD CTRL + D' and 'CLOSE ESC' buttons.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.5.6</version>
9     <relativePath/> <!-- Lookup parent from repository -->
10  </parent>
11  <groupId>org.sample.devops</groupId>
12  <artifactId>authentication-service</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>Authentication Sservice</name>
15  <description>Demo project for Spring Boot</description>
16  <url/>
17  <licenses>
18    <license/>
19  </licenses>
20  <developers>
21    <developer/>
22  </developers>
23  <scm>
24    <connection/>
25    <developerConnection/>
26    <tag/>
27    <url/>
28  </scm>
29  <properties>
```

Copiez dans le
pom.xml de
votre projet

Copiez les dépendances dans le pom.xml

IntelliJ

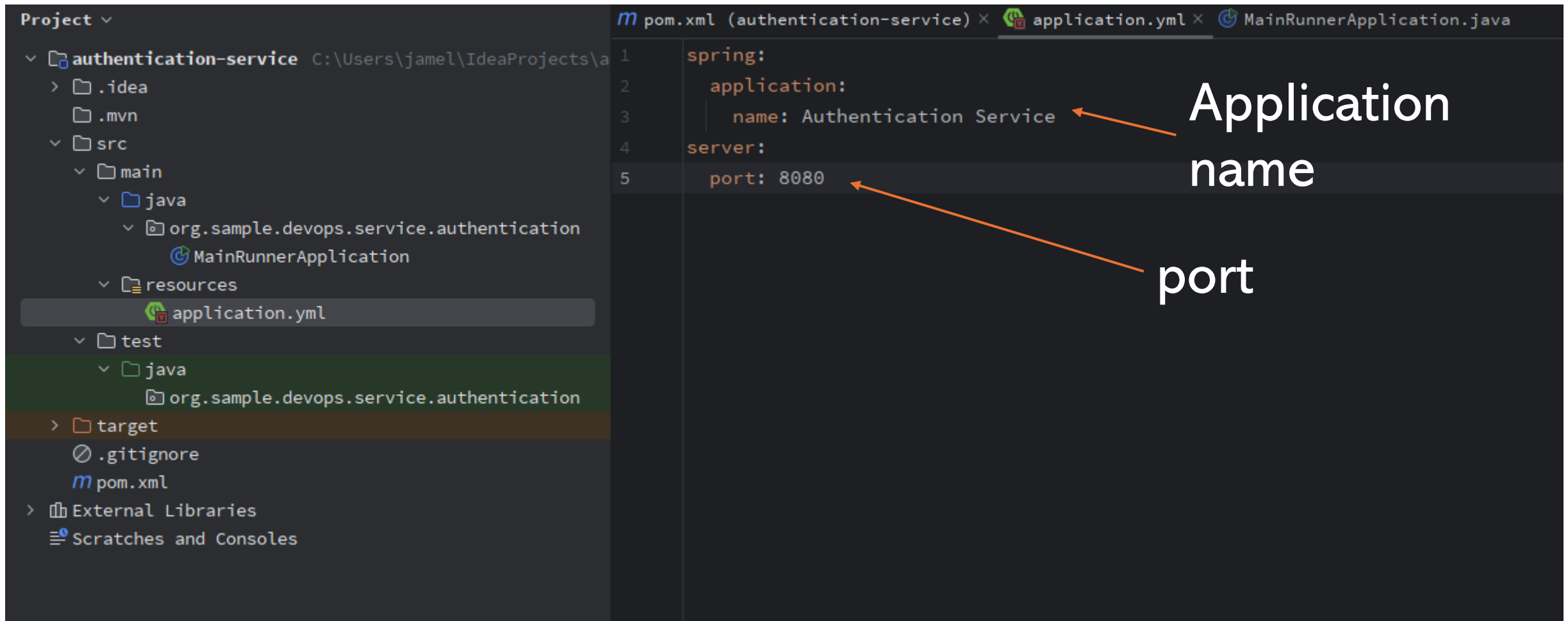


The screenshot shows the IntelliJ IDEA interface. On the left, the 'Project' view displays the directory structure of a Maven project named 'authentication-service'. The structure includes a 'src' directory with 'main' and 'test' subdirectories, each containing 'java' and 'resources' folders. The 'pom.xml' file is highlighted in the 'Project' view. On the right, the 'pom.xml' file is open in the editor. The XML content is as follows:

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.5.6</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>org.sample.devops</groupId>
12  <artifactId>authentication-service</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>Authentication Service</name>
15  <description>Demo project for Spring Boot</description>
16  <url/>
17  <licenses>
18    <license/>
19  </licenses>
20  <developers>
21    <developer/>
22  </developers>
23  <scm>
24    <connection/>
25    <developerConnection/>
26    <tag/>
27    <url/>
28  </scm>
29  <properties>
30    <java.version>17</java.version>
31  </properties>
32  <dependencies>
33    <dependency>
34      <groupId>org.springframework.boot</groupId>
35      <artifactId>spring-boot-starter-web</artifactId>
36    </dependency>
37    <dependency>
38      <groupId>org.springframework.boot</groupId>
```

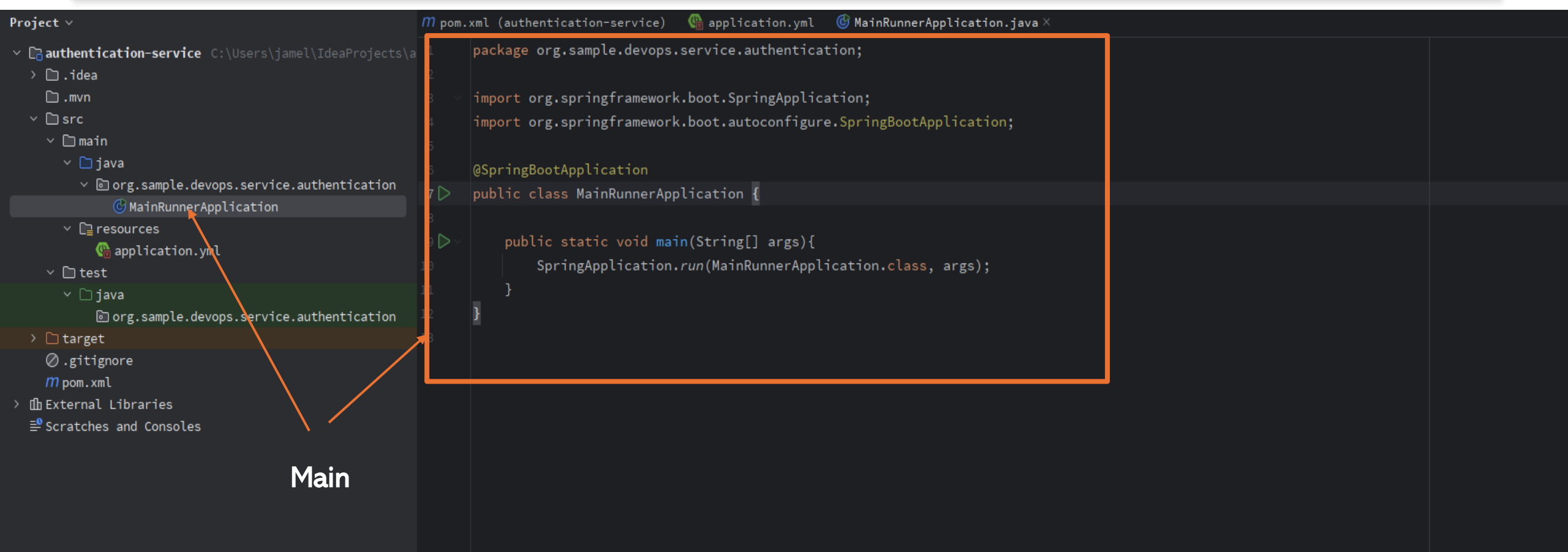
Ajouter un fichier de config application.yml

IntelliJ



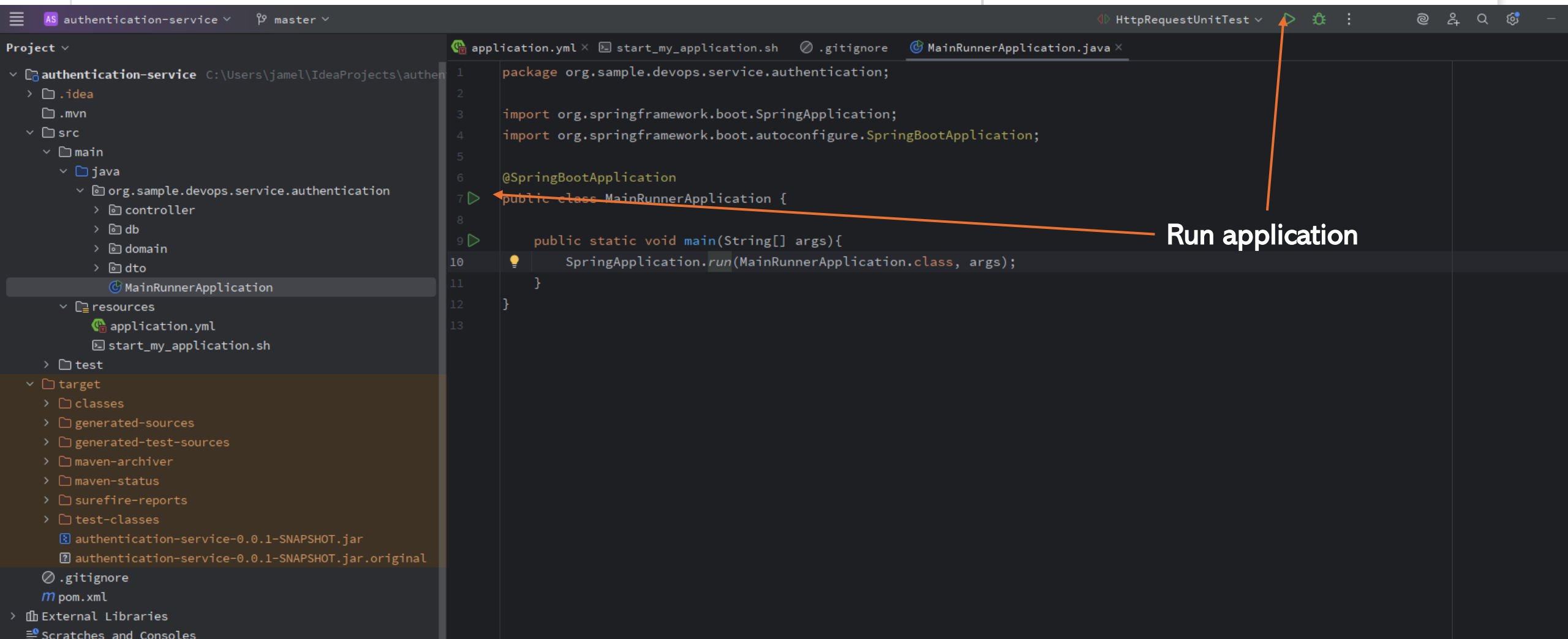
Ajouter la lanceur Main

IntelliJ



Lancez l'application

IntelliJ



Tester l'application

Browser: Firefox
http://localhost:8080



Whitelabel Error Page

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

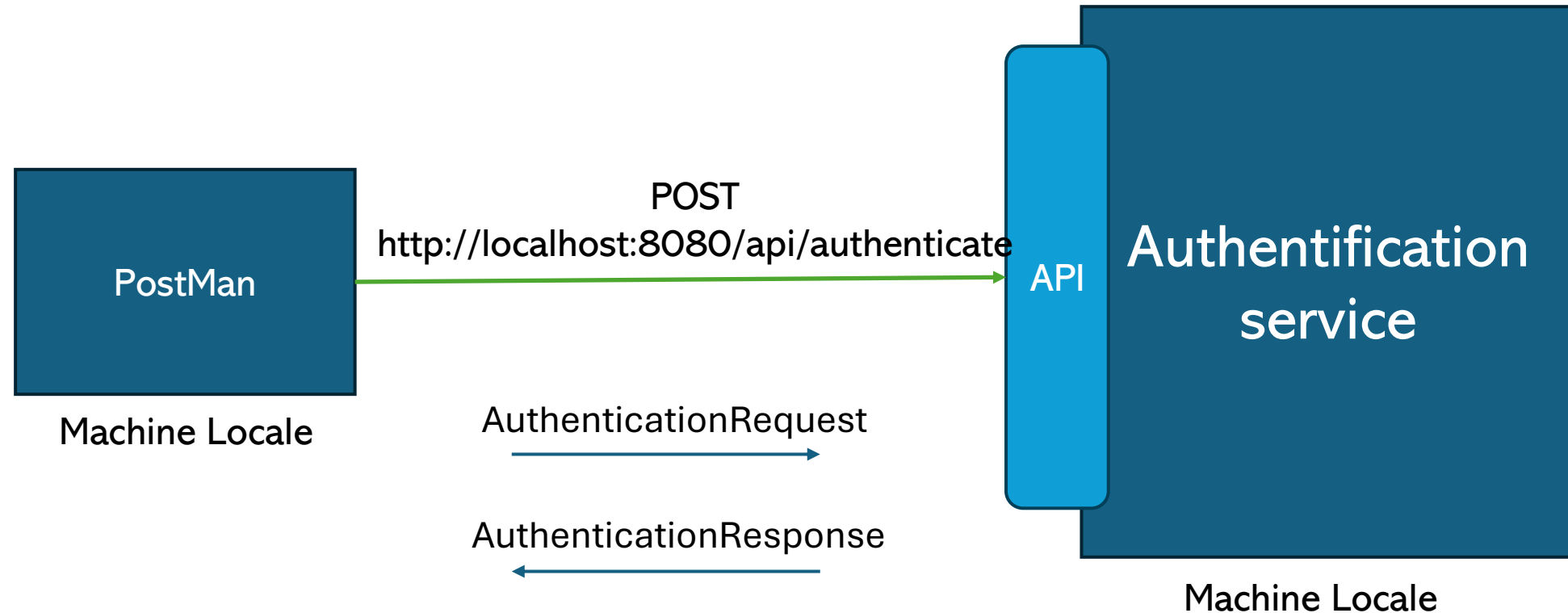
Sun Sep 28 21:15:31 CEST 2025

There was an unexpected error (type=Not Found, status=404).

URL port 8080

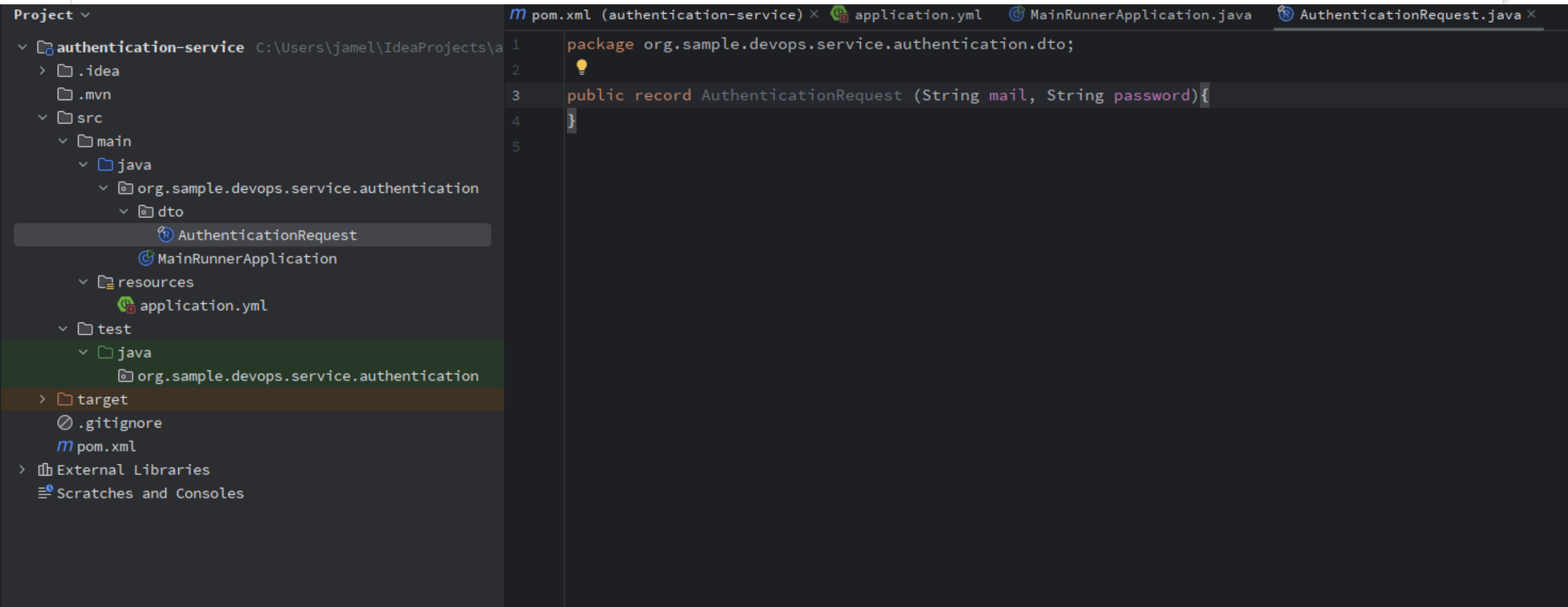
Erreur 404: Pas d'API exposée

Ajoutez un Controller (API Endpoint /authenticate)



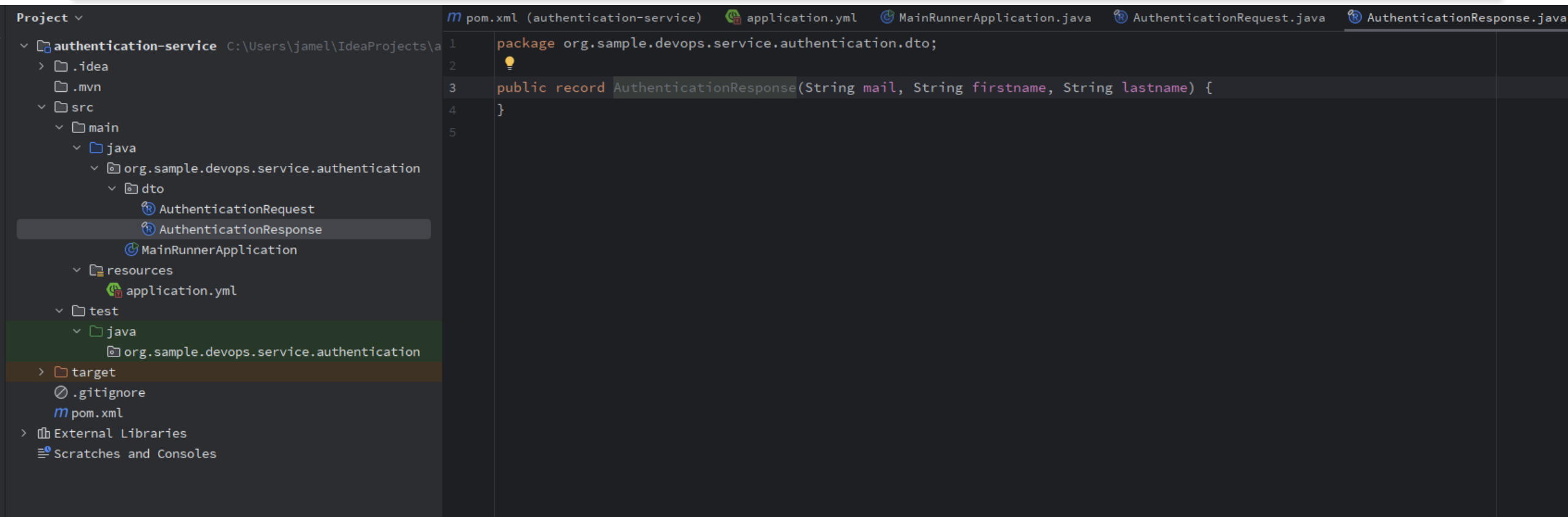
Création du DTO AuthenticationRequest

IntelliJ



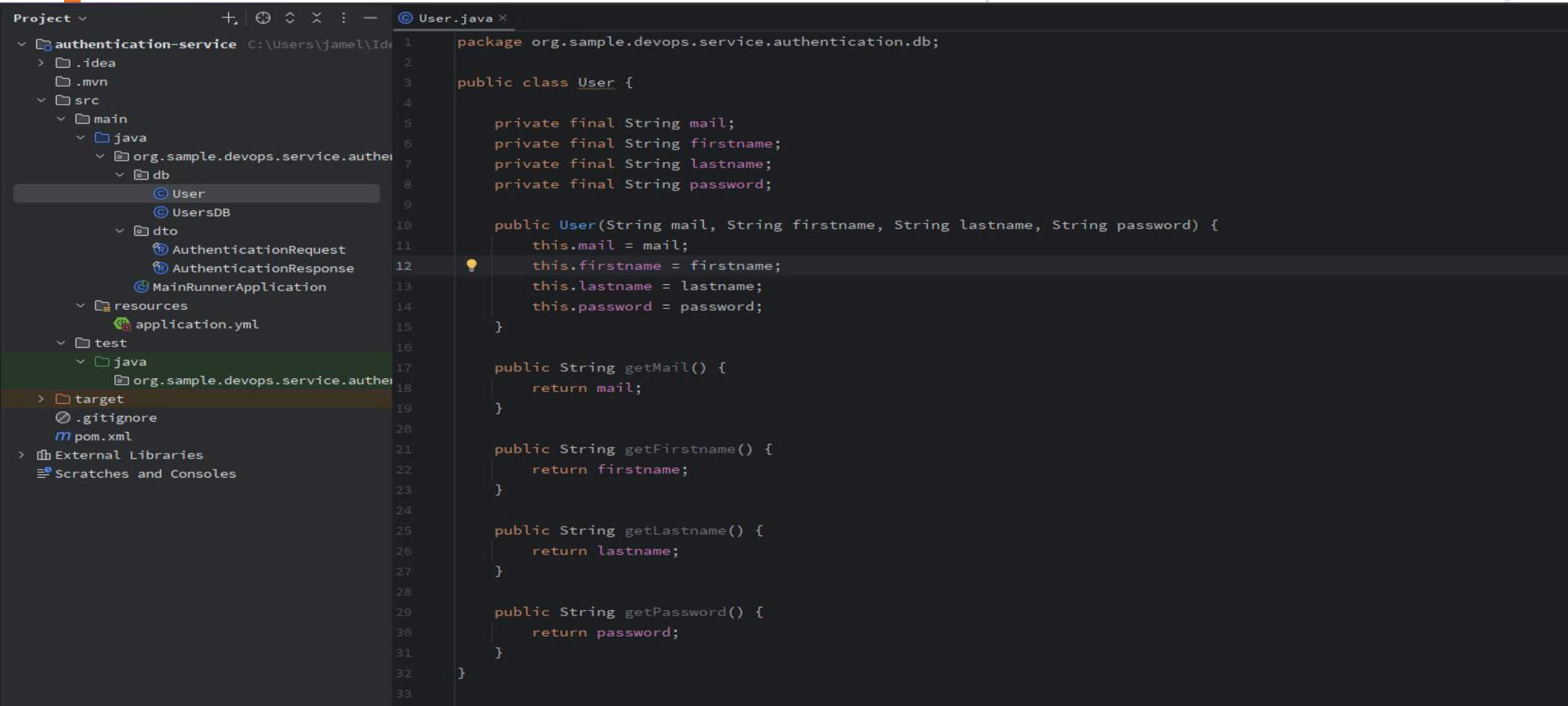
Création du DTO AuthenticationResponse

IntelliJ



Création du Bean User

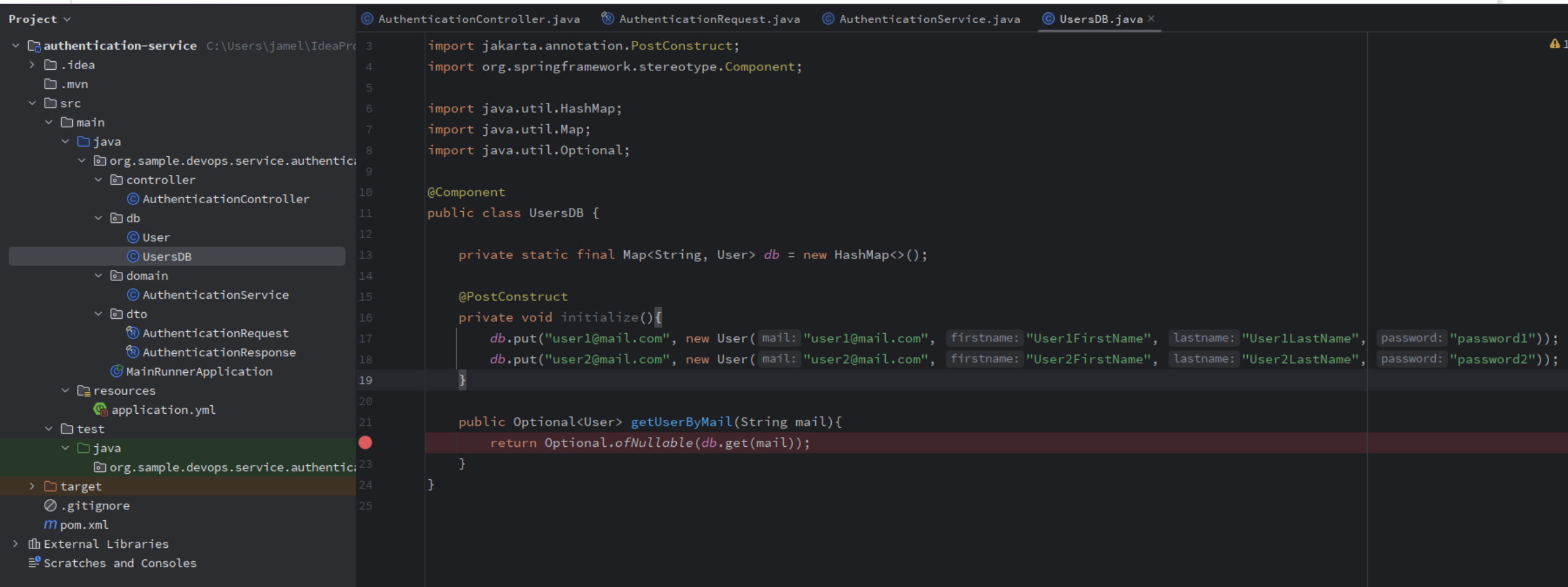
IntelliJ



```
1 package org.sample.devops.service.authentication.db;
2
3 public class User {
4
5     private final String mail;
6     private final String firstname;
7     private final String lastname;
8     private final String password;
9
10    public User(String mail, String firstname, String lastname, String password) {
11        this.mail = mail;
12        this.firstname = firstname;
13        this.lastname = lastname;
14        this.password = password;
15    }
16
17    public String getMail() {
18        return mail;
19    }
20
21    public String getFirstname() {
22        return firstname;
23    }
24
25    public String getLastname() {
26        return lastname;
27    }
28
29    public String getPassword() {
30        return password;
31    }
32 }
33
```

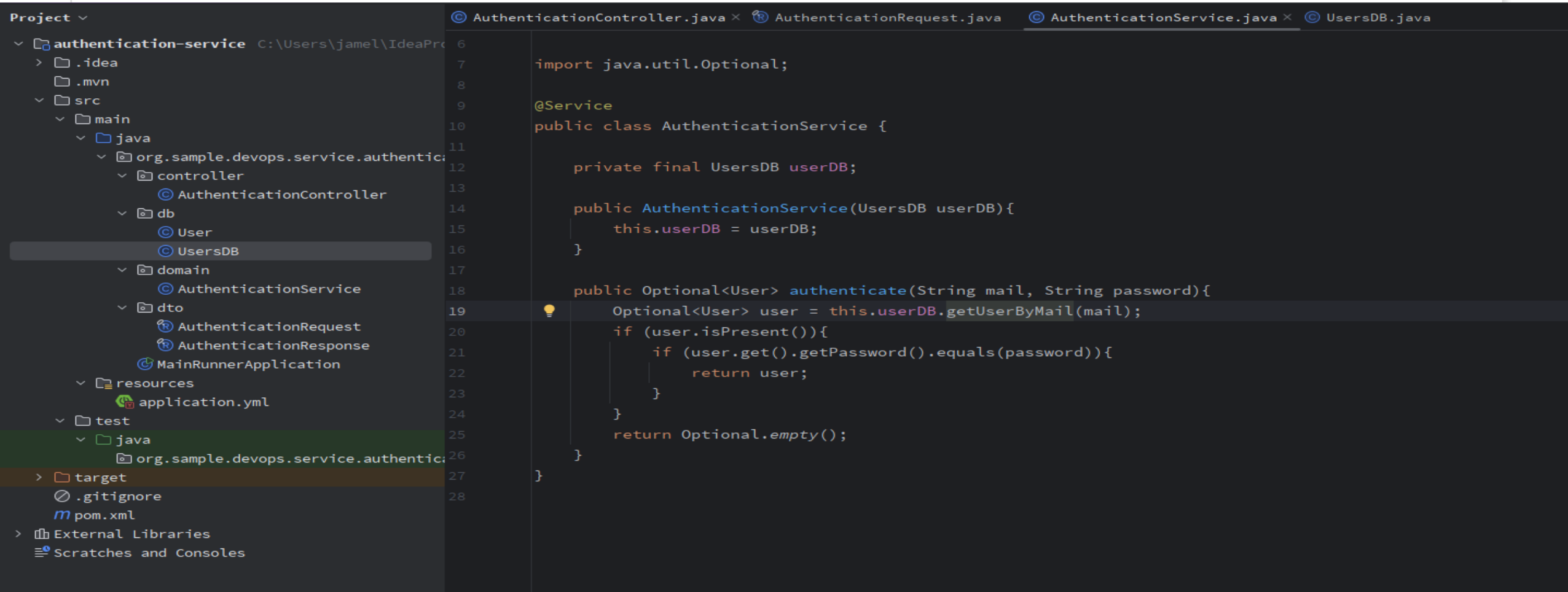
Création de la DB User

IntelliJ



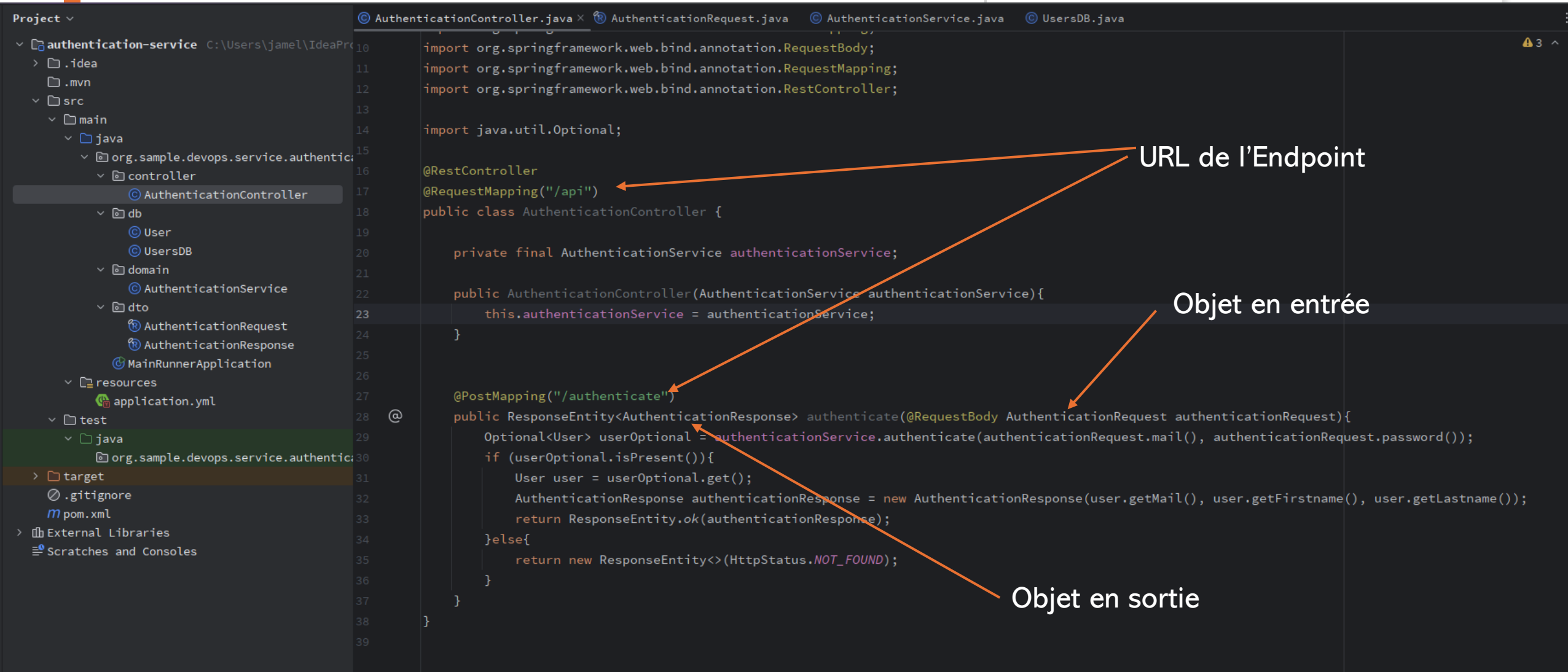
Création du service domaine

IntelliJ



Création du Controller Rest

IntelliJ



Lancement du Projet

IntelliJ

The screenshot displays the IntelliJ IDEA IDE interface. On the left, the 'Project' view shows the directory structure of the 'authentication-service' project, including folders for 'main' (java, resources) and 'test'. The 'AuthenticationController' class is selected in the 'controller' folder. The main editor window shows the code for 'AuthenticationController.java', which includes imports for Spring Web MVC, a REST controller annotation, and a constructor that injects 'AuthenticationService'. The 'authenticate' method is also visible, taking an 'AuthenticationRequest' and returning an 'AuthenticationResponse'. The bottom panel shows the 'Run' configuration for 'MainRunnerApplication' and a console output window displaying the application's startup logs, including Spring Boot version (v3.5.6) and Tomcat initialization details.

```
Project
├── authentication-service
│   ├── .idea
│   ├── .mvn
│   └── src
│       ├── main
│       │   ├── java
│       │   │   ├── org.sample.devops.service.authentication
│       │   │   │   ├── controller
│       │   │   │   │   ├── AuthenticationController
│       │   │   │   │   ├── db
│       │   │   │   │   │   ├── User
│       │   │   │   │   │   ├── UsersDB
│       │   │   │   │   ├── domain
│       │   │   │   │   │   ├── AuthenticationService
│       │   │   │   │   ├── dto
│       │   │   │   │   │   ├── AuthenticationRequest
│       │   │   │   │   │   ├── AuthenticationResponse
│       │   │   │   │   ├── MainRunnerApplication
│       │   │   ├── resources
│       │   │   │   ├── application.yml
│       │   └── test
│       └── test
└── test
```

```
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Optional;

@RestController
@RequestMapping("/api")
public class AuthenticationController {

    private final AuthenticationService authenticationService;

    public AuthenticationController(AuthenticationService authenticationService) {
        this.authenticationService = authenticationService;
    }

    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponse> authenticate(@RequestBody AuthenticationRequest authenticationRequest) {
        Optional<User> userOptional = authenticationService.authenticate(authenticationRequest.mail(), authenticationRequest.password());
    }
}
```

```

\W/  _ _ _ | | _ | | | | | | ( _ | ) ) ) )
' | _ _ _ | . _ _ | | _ | | _ _ _ | / / / /
=====|_|=====| _ _ _ / _ / _ / _ /

:: Spring Boot ::                (v3.5.6)

2025-09-29T09:25:29.134+02:00 INFO 29780 --- [Authentication Service] [main] o.s.d.s.a.MainRunnerApplication : Starting MainRunnerApplication using Java 17.0.2 with PID 29780 (C:\U
2025-09-29T09:25:29.142+02:00 INFO 29780 --- [Authentication Service] [main] o.s.d.s.a.MainRunnerApplication : No active profile set, falling back to 1 default profile: "default"
2025-09-29T09:25:31.380+02:00 INFO 29780 --- [Authentication Service] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-09-29T09:25:31.400+02:00 INFO 29780 --- [Authentication Service] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-09-29T09:25:31.400+02:00 INFO 29780 --- [Authentication Service] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]
2025-09-29T09:25:31.509+02:00 INFO 29780 --- [Authentication Service] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-09-29T09:25:31.511+02:00 INFO 29780 --- [Authentication Service] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2239 ms
2025-09-29T09:25:32.268+02:00 INFO 29780 --- [Authentication Service] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-09-29T09:25:32.285+02:00 INFO 29780 --- [Authentication Service] [main] o.s.d.s.a.MainRunnerApplication : Started MainRunnerApplication in 4.185 seconds (process running for 5
```

Testez l'Endpoint

Postman

http://localhost:8080

POST

Request

Réponse "User"

URL de l'Endpoint

JSON

The screenshot shows the Postman interface for a POST request. The URL bar at the top shows 'http://localhost:8080/api/authenticate'. The 'Body' tab is selected, and the request body is a JSON object: { "mail": "user1@mail.com", "password": "password1" }. The 'JSON' tab is selected in the body editor. The response section at the bottom shows a status of '200 OK' and a JSON response: { "mail": "user1@mail.com", "firstname": "User1FirstName", "lastname": "User1LastName" }. Orange arrows point from the labels to the corresponding parts of the interface: 'POST' to the method dropdown, 'Request' to the body editor, 'Réponse "User"' to the response body, 'URL de l'Endpoint' to the URL bar, and 'JSON' to the JSON tab in the body editor.

```
POST http://localhost:8080/api/authenticate
```

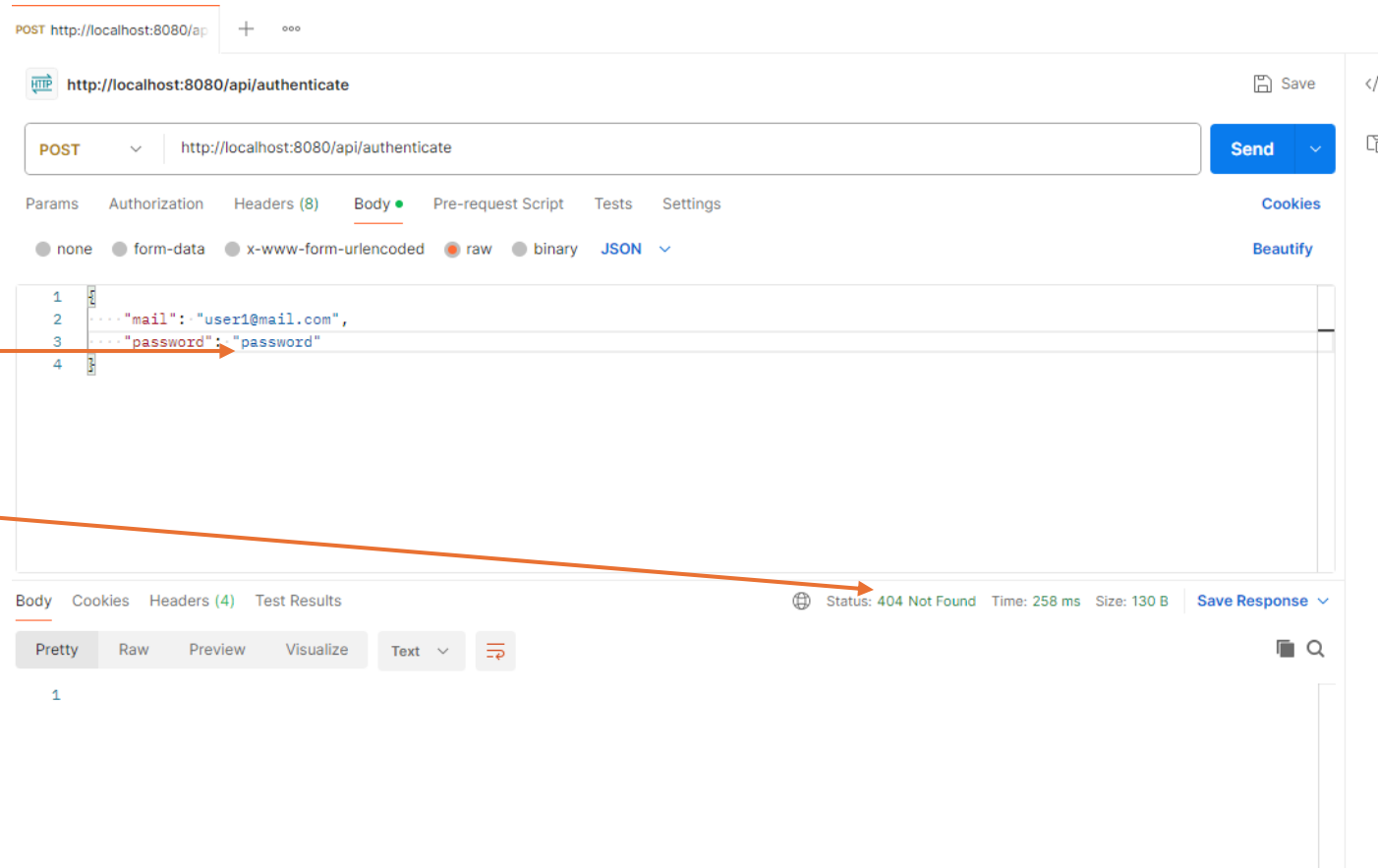
```
{ 1  "mail": "user1@mail.com", 2  "password": "password1" 3 }
```

```
{ 1  "mail": "user1@mail.com", 2  "firstname": "User1FirstName", 3  "lastname": "User1LastName" 4 }
```

Status: 200 OK Time: 15 ms Size: 245 B

Testez l'Endpoint

Postman

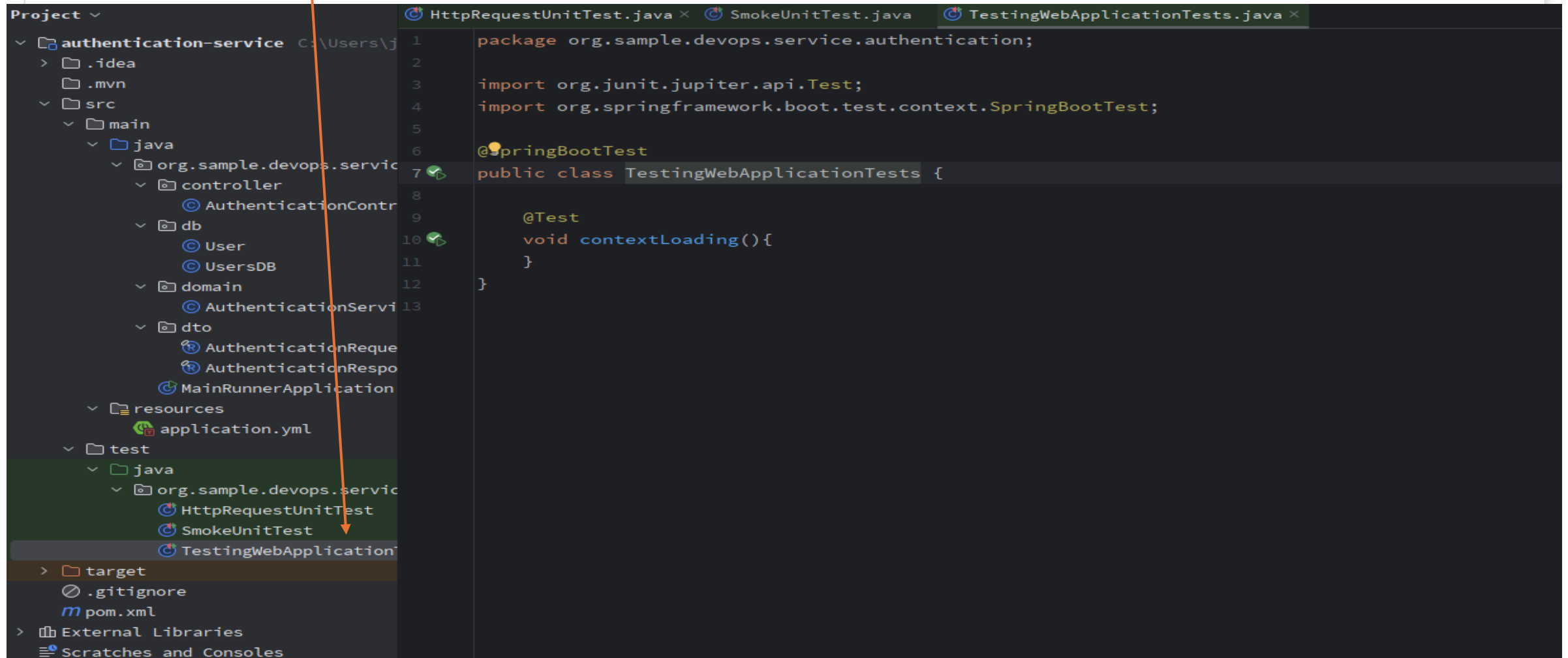


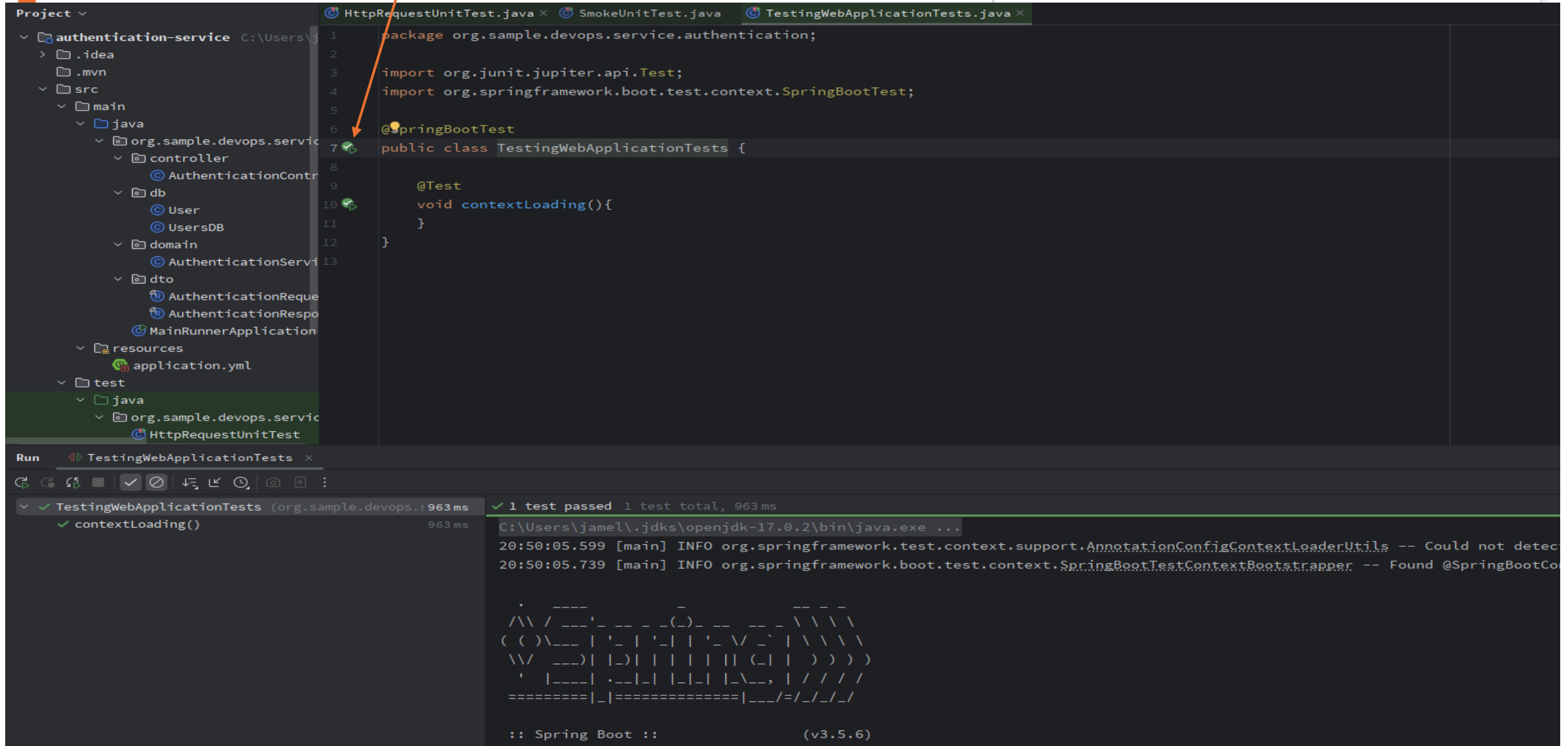
Mot de passe incorrect

Erreur 404

Ajoutez un test Unitaire

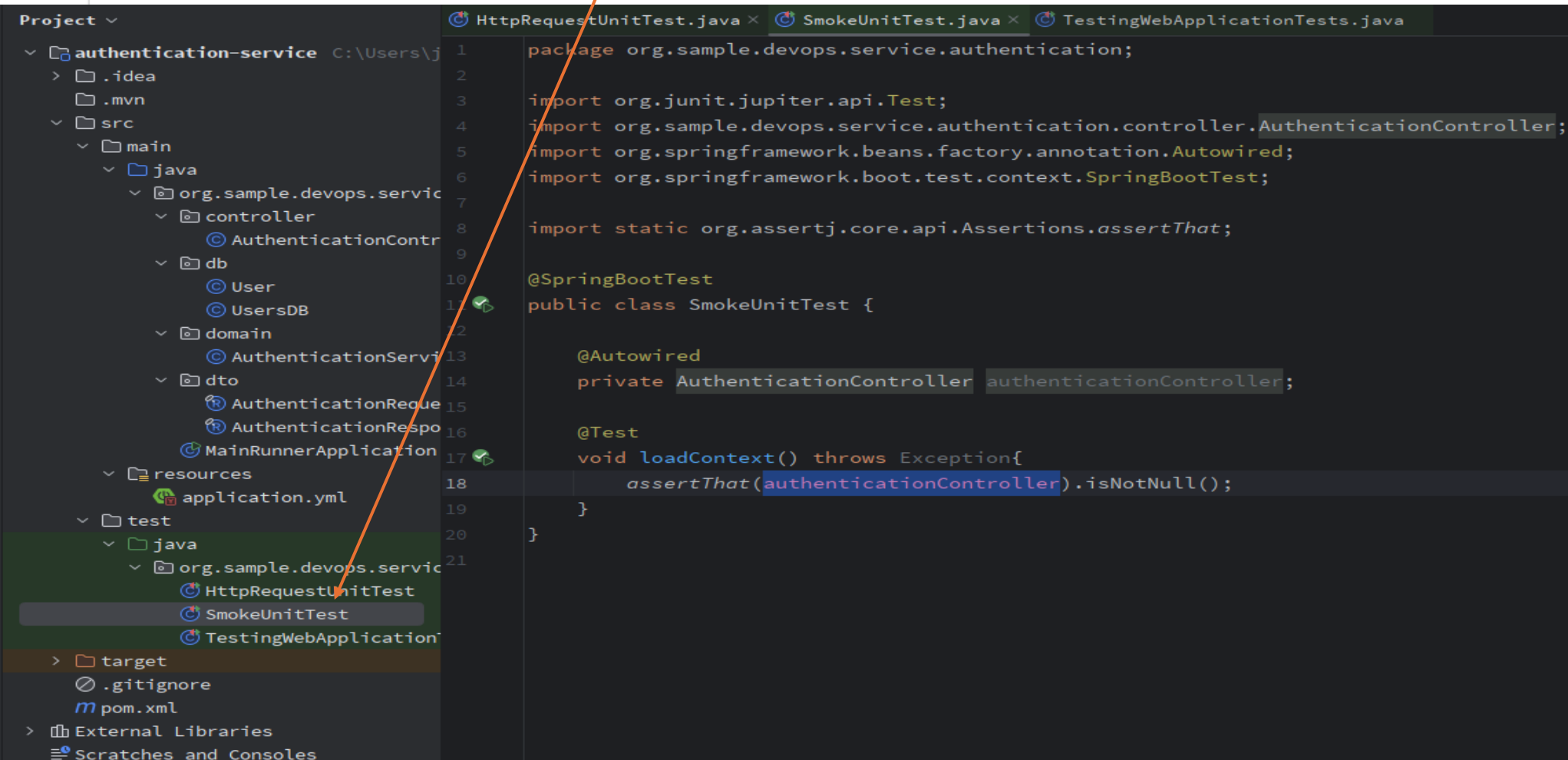
IntelliJ





Ajoutez un test Unitaire

IntelliJ

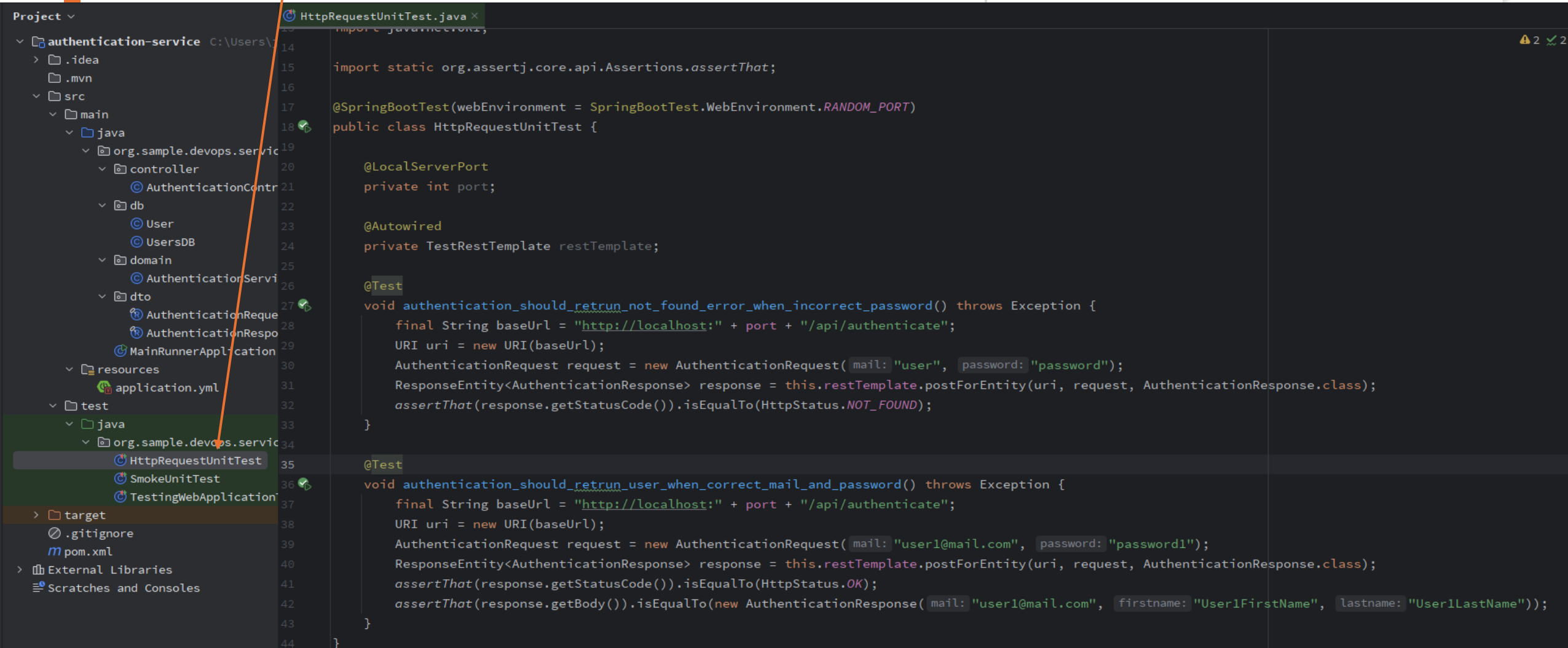


The screenshot shows the IntelliJ IDEA interface. On the left, the 'Project' view displays the file structure of a project named 'authentication-service'. The 'test' directory is expanded, showing a 'java' subdirectory with three test classes: 'HttpRequestUnitTest', 'SmokeUnitTest', and 'TestingWebApplicationTests'. An orange arrow points from the title 'Ajoutez un test Unitaire' to the 'SmokeUnitTest' file. The main editor area shows the code for 'SmokeUnitTest.java'. The code includes imports for JUnit 5, Spring Boot Test, and AssertJ. It defines a class 'SmokeUnitTest' with an '@Autowired' field 'authenticationController' and a '@Test' method 'loadContext()' that asserts the controller is not null.

```
1 package org.sample.devops.service.authentication;
2
3 import org.junit.jupiter.api.Test;
4 import org.sample.devops.service.authentication.controller.AuthenticationController;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.test.context.SpringBootTest;
7
8 import static org.assertj.core.api.Assertions.assertThat;
9
10 @SpringBootTest
11 public class SmokeUnitTest {
12
13     @Autowired
14     private AuthenticationController authenticationController;
15
16     @Test
17     void loadContext() throws Exception{
18         assertThat(authenticationController).isNotNull();
19     }
20 }
21
```

Ajoutez un test Unitaire

IntelliJ



The screenshot shows the IntelliJ IDEA IDE. On the left, the Project Explorer displays the structure of the 'authentication-service' project. The 'test' directory is expanded, showing the 'java' subdirectory. The 'HttpRequestUnitTest' class is highlighted. An orange arrow points from the title 'Ajoutez un test Unitaire' to this class. The main editor shows the code for 'HttpRequestUnitTest.java'. The code includes imports for 'org.springframework.test.web.servlet.MockMvc' and 'org.assertj.core.api.Assertions.assertThat'. It uses '@SpringBootTest' with 'RANDOM_PORT' and '@LocalServerPort' to define a test environment. Two '@Test' methods are present: one for an incorrect password and one for a correct password. The first test expects a 'NOT_FOUND' status, while the second expects a '200 OK' status and checks the response body for specific user details.

```
import org.springframework.test.web.servlet.MockMvc;
import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HttpRequestUnitTest {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    void authentication_should_retrun_not_found_error_when_incorrect_password() throws Exception {
        final String baseUrl = "http://localhost:" + port + "/api/authenticate";
        URI uri = new URI(baseUrl);
        AuthenticationRequest request = new AuthenticationRequest( mail: "user", password: "password");
        ResponseEntity<AuthenticationResponse> response = this.restTemplate.postForEntity(uri, request, AuthenticationResponse.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
    }

    @Test
    void authentication_should_retrun_user_when_correct_mail_and_password() throws Exception {
        final String baseUrl = "http://localhost:" + port + "/api/authenticate";
        URI uri = new URI(baseUrl);
        AuthenticationRequest request = new AuthenticationRequest( mail: "user1@mail.com", password: "password1");
        ResponseEntity<AuthenticationResponse> response = this.restTemplate.postForEntity(uri, request, AuthenticationResponse.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody()).isEqualTo(new AuthenticationResponse( mail: "user1@mail.com", firstname: "User1FirstName", lastname: "User1LastName"));
    }
}
```

Lancez le test Unitaire

IntelliJ

The screenshot displays the IntelliJ IDEA IDE interface. On the left, the Project view shows the file structure of the project, with the 'test' directory expanded and 'HttpRequestUnitTest.java' selected. The main editor window shows the code for 'HttpRequestUnitTest.java'. The code includes imports for 'org.springframework.test.web.servlet.MockMvc' and 'org.assertj.core.api.Assertions.assertThat'. It is annotated with '@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)', '@LocalServerPort', '@Autowired', and '@Test'. The test method 'authentication_should_retrun_not_found_error_when_incorrect_password()' is shown, which uses 'MockMvc' to perform a POST request and asserts that the status code is 'NOT_FOUND'. The Run view at the bottom shows the test results: '2 tests passed 2 tests total, 2 sec 942 ms'.

```
import org.springframework.test.web.servlet.MockMvc;
import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HttpRequestUnitTest {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    void authentication_should_retrun_not_found_error_when_incorrect_password() throws Exception {
        final String baseUrl = "http://localhost:" + port + "/api/authenticate";
        URI uri = new URI(baseUrl);
        AuthenticationRequest request = new AuthenticationRequest(mail: "user", password: "password");
        ResponseEntity<AuthenticationResponse> response = this.restTemplate.postForEntity(uri, request, AuthenticationResponse.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
    }

    @Test
    void authentication_should_retrun_user_when_correct_password() throws Exception {
        final String baseUrl = "http://localhost:" + port + "/api/authenticate";
        URI uri = new URI(baseUrl);
        AuthenticationRequest request = new AuthenticationRequest(mail: "user", password: "password");
        ResponseEntity<AuthenticationResponse> response = this.restTemplate.postForEntity(uri, request, AuthenticationResponse.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
    }
}
```

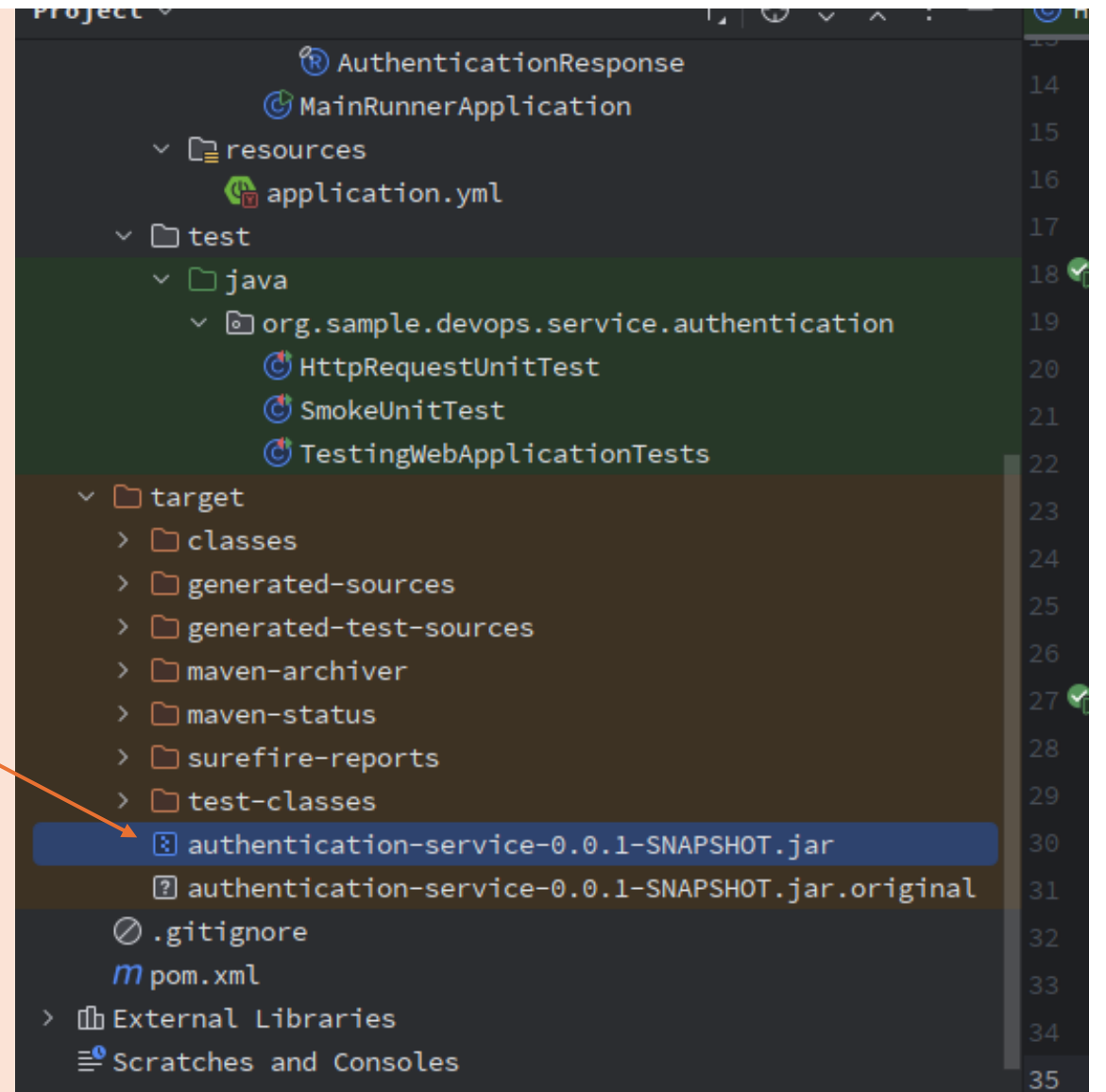
Run: HttpRequestUnitTest

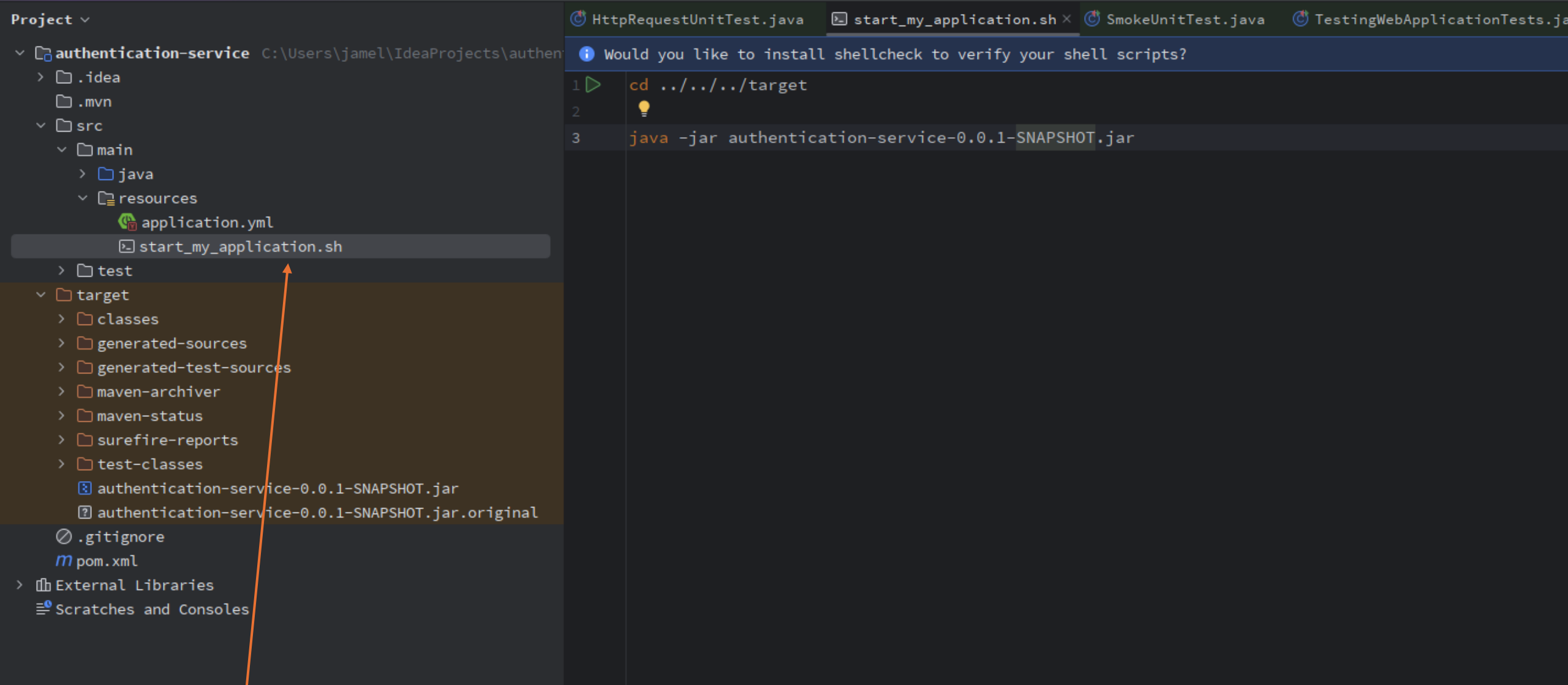
2 tests passed 2 tests total, 2 sec 942 ms

- ✓ authentication_should_retrun_user_when_correct_password 2 sec 805 ms
- ✓ authentication_should_retrun_not_found_error 137 ms

Livrable

IntelliJ





Script de lancement du livrable

IntelliJ

Testez l'Endpoint

Postman

http://localhost:8080

POST

Request

Réponse "User"

URL de l'Endpoint

JSON

The screenshot shows the Postman interface for a POST request. The URL bar at the top shows 'http://localhost:8080/api/authenticate'. The 'Body' tab is selected, and the request body is a JSON object: { "mail": "user1@mail.com", "password": "password1" }. The 'JSON' tab is selected in the body editor. The response section at the bottom shows a status of '200 OK' and a JSON response: { "mail": "user1@mail.com", "firstname": "User1FirstName", "lastname": "User1LastName" }. Orange arrows point from the labels to the corresponding elements in the interface.

```
POST http://localhost:8080/api/authenticate
```

```
{ 1  "mail": "user1@mail.com", 2  "password": "password1" 3 }
```

```
{ 1  "mail": "user1@mail.com", 2  "firstname": "User1FirstName", 3  "lastname": "User1LastName" 4 }
```

Status: 200 OK Time: 15 ms Size: 245 B

Deployment manuelle sur chaque serveur

Scripting

