

DevOps

TD-01 - Creation d'un projet – Integration et Deploiement manuelle

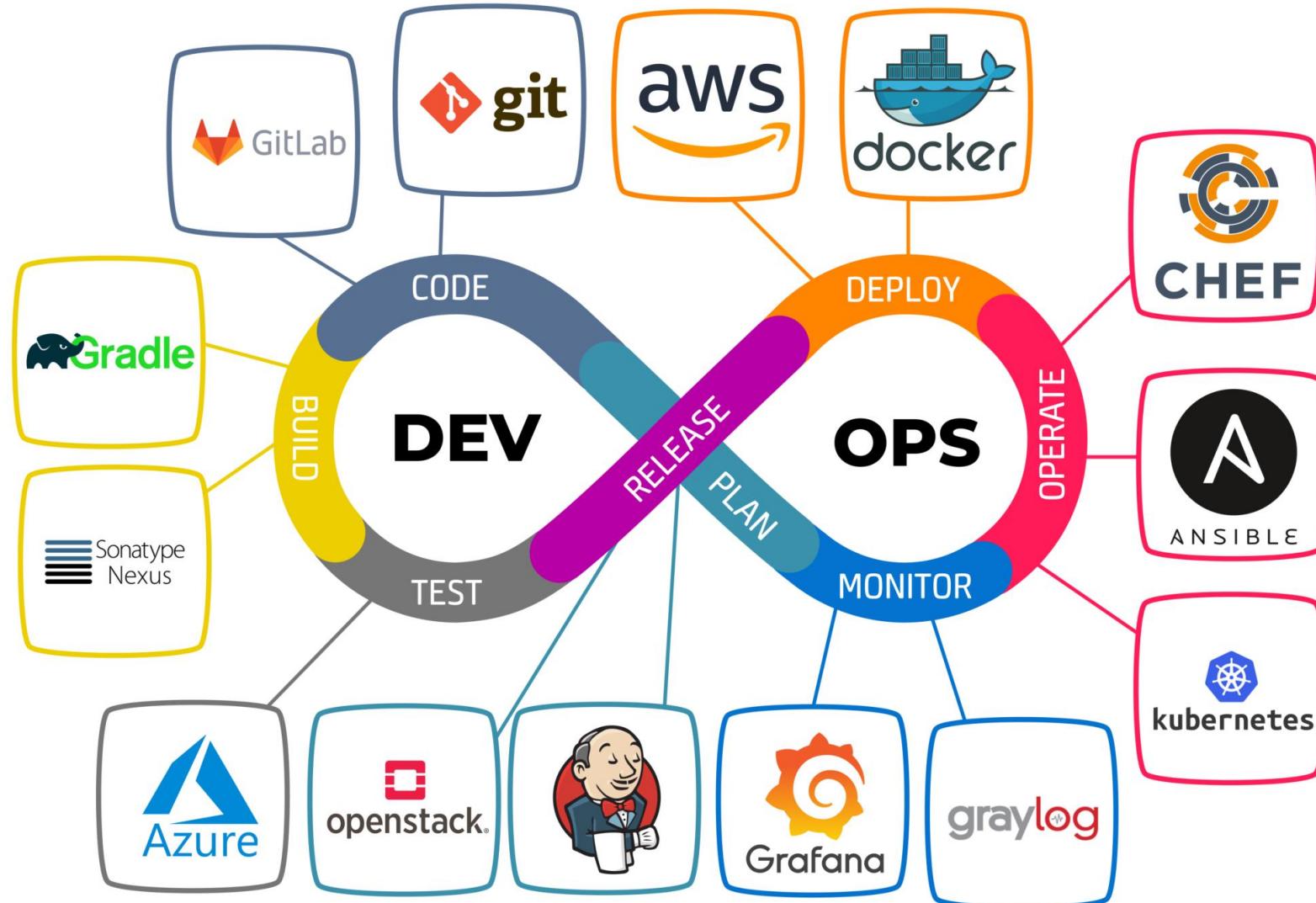
Jamel ESSOUSSI: Architecte logiciel

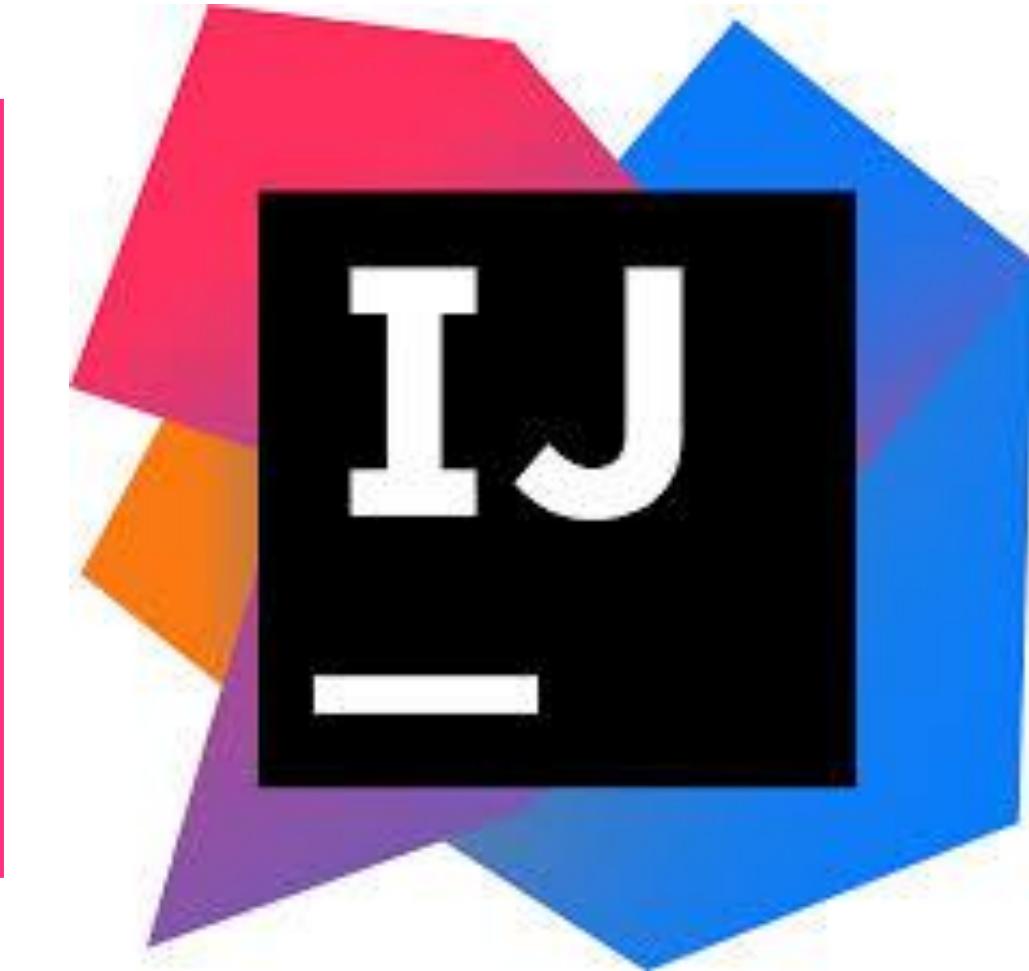
jamel.essoussi@gmail.com

<https://github.com/jessoussi>

2025

Outils du DevOps



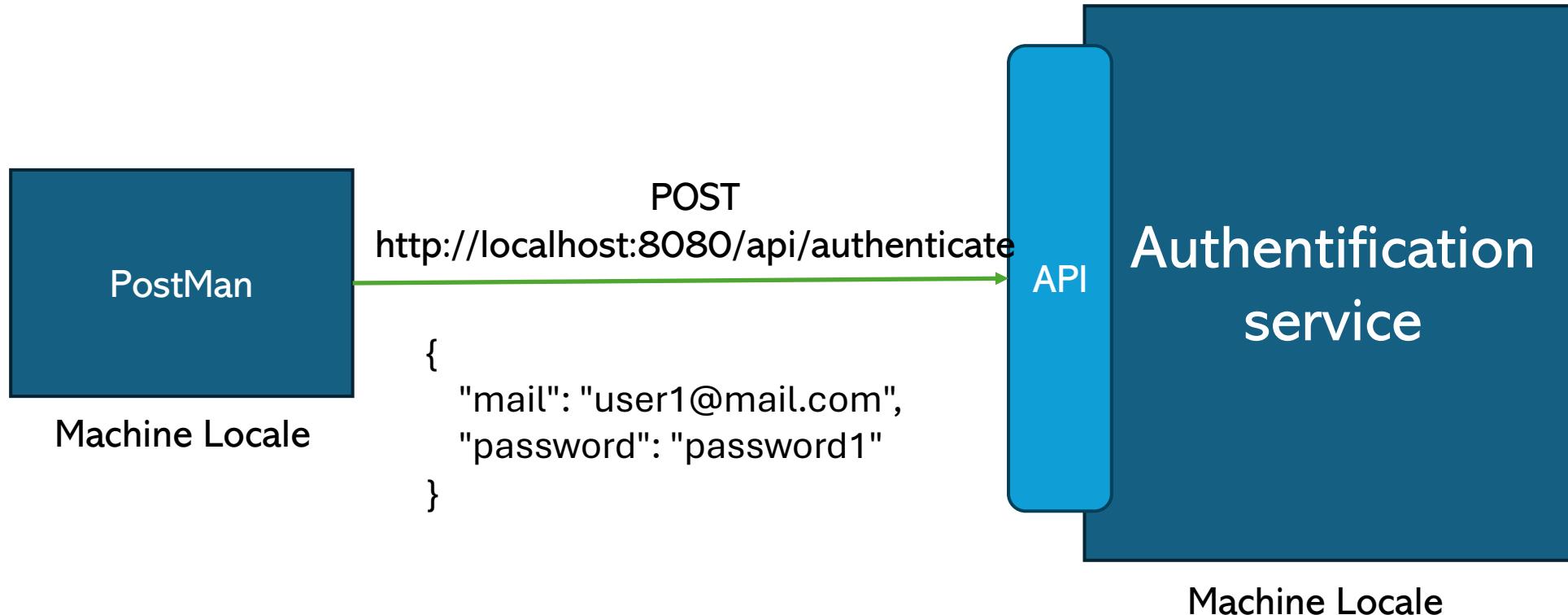


**IntelliJ
IDE
PostMan**

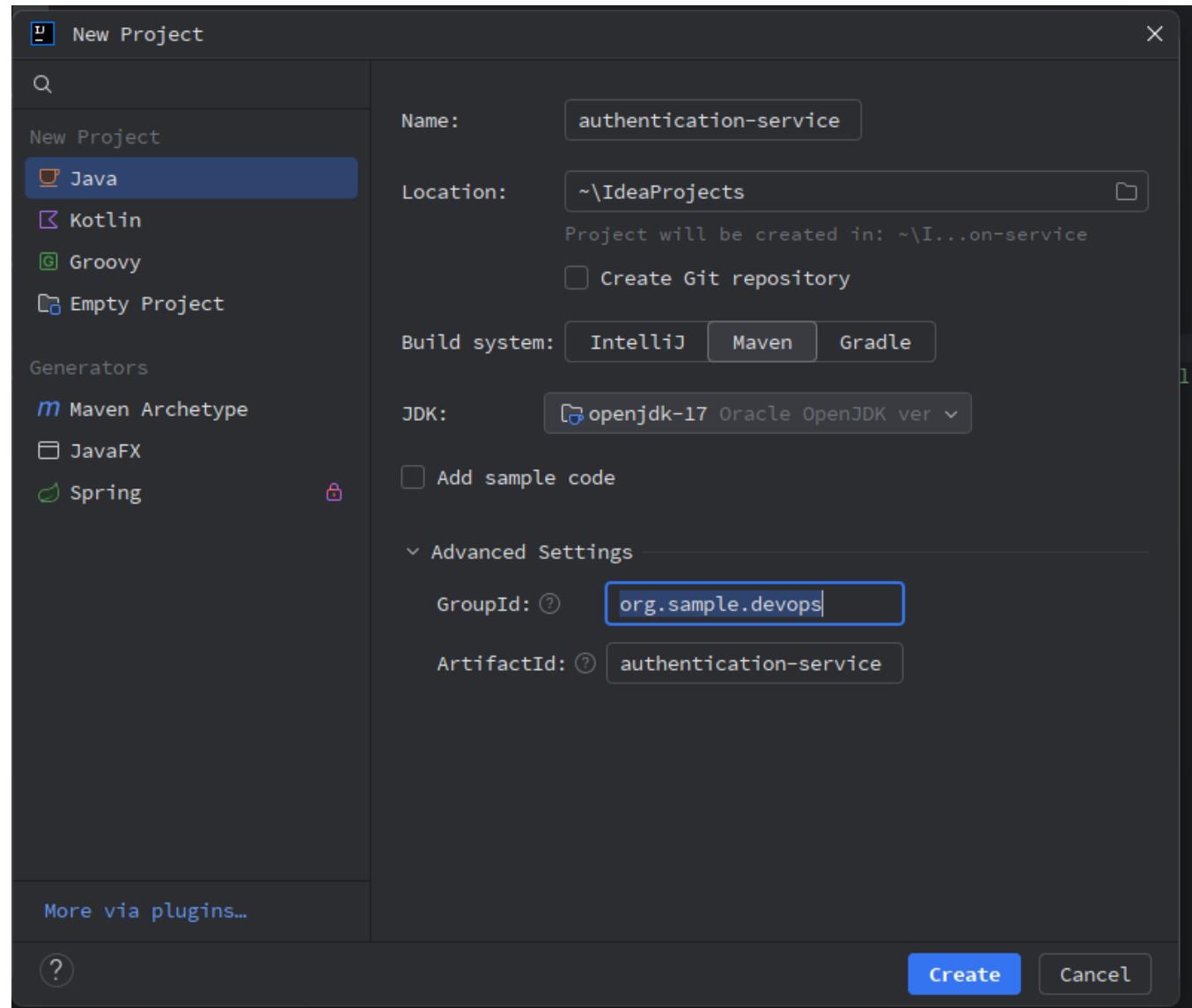
Objectif du TD

- 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



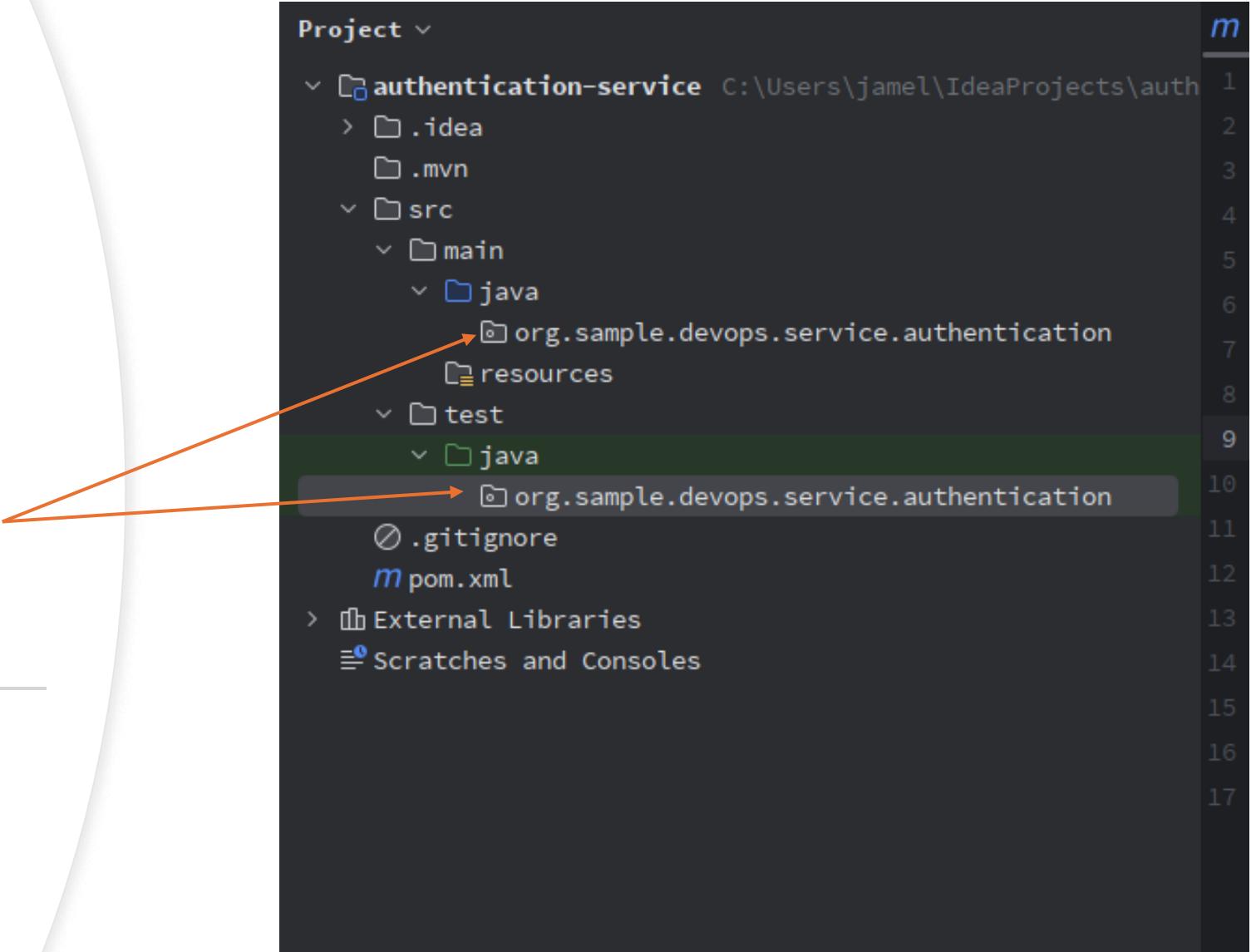
The screenshot shows the IntelliJ IDEA interface with the following elements:

- Project View:** On the left, it shows the project structure under "authentication-service". It includes ".idea", ".mvn", "src" (with "main" and "test" subfolders), "resources", ".gitignore", and "pom.xml".
- pom.xml Content:** The right pane displays the XML configuration for the Maven project. The code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.sample.devops</groupId>
    <artifactId>authentication-service</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

Annotations: Three orange arrows point from the text "Code source et configurations du projet", "Test unitaires", and "pom.xml" to the corresponding parts of the interface: the project tree, the test folder in the tree, and the pom.xml file in the editor.

Création des packages



Ajoutez les dépendances SpringBoot

Spring initializr

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

The screenshot shows the Spring Initializr web interface at <https://start.spring.io/>. The left side displays project metadata and build settings, while the right side shows dependency selection. A red box highlights the 'Project Metadata' section, and an arrow points from the text 'Caractéristiques du projet' to it. Another red box highlights the 'Dependencies' section, and an arrow points from the text 'Ajoutez Spring web' to it. A third arrow points from the text 'Générez les dépendances' to the 'GENERATE' button at the bottom.

Caractéristiques du projet

spring initializr

Project

Language

Spring Boot

Project Metadata

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

GENERATE CTRL + G

EXPLORE CTRL + SPACE

...

Générez les dépendances SpringBoot

Spring initializr

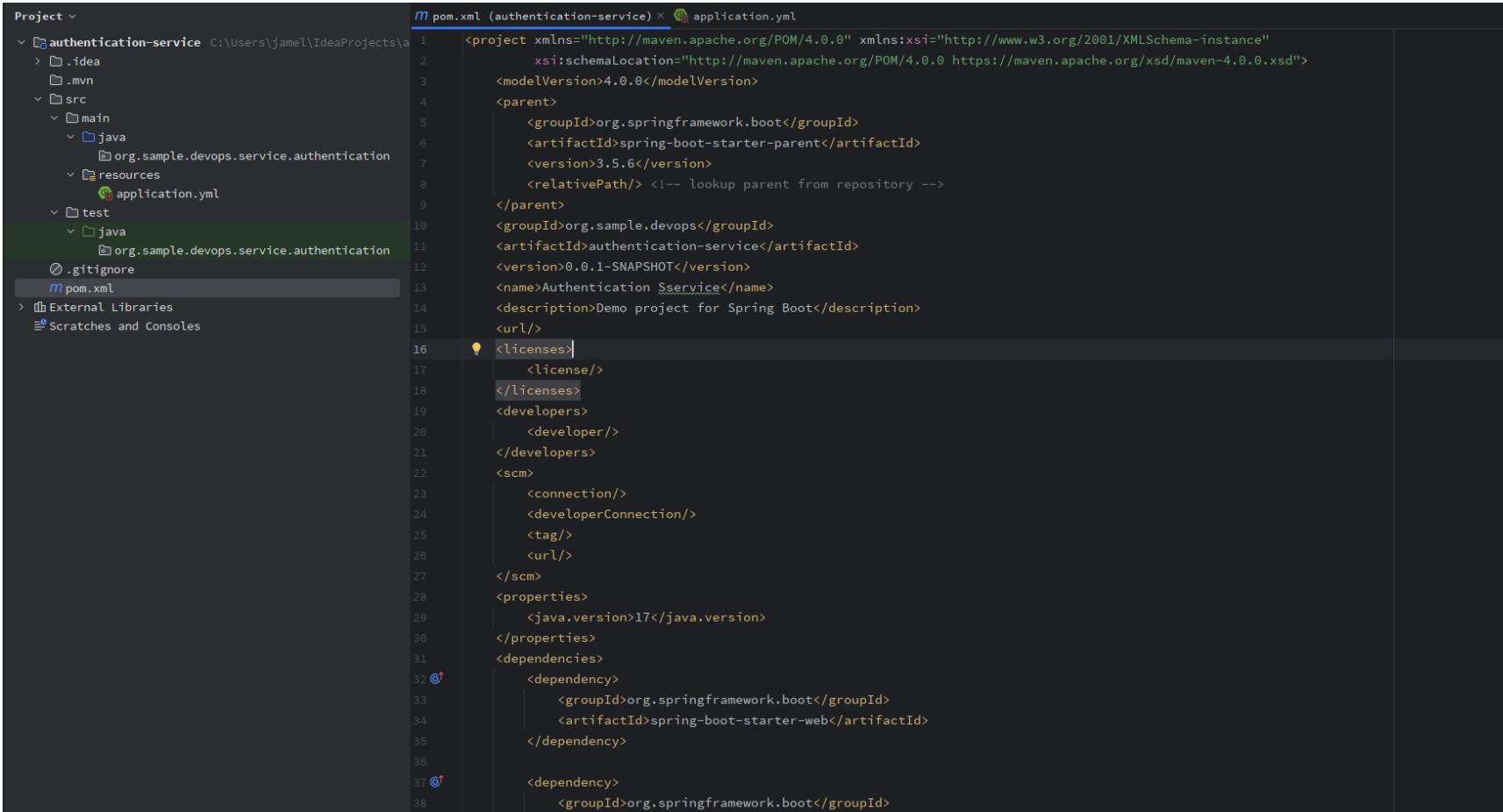
<https://start.spring.io/>

The screenshot shows the Spring Initializr interface. On the left, a sidebar displays the contents of a zip file named "authentication-service.zip", which includes ".gitattributes", ".gitignore", ".mvn", "HELP.md", "mvnw", "mvnw.cmd", and "pom.xml". The "pom.xml" file is highlighted. On the right, the main configuration area shows the XML content of the pom.xml file. At the top right of this area are two buttons: "DOWNLOAD" and "COPY". A blue arrow points from the "COPY" button to the text "Copiez dans le pom.xml de votre projet" located on the right side of the slide. At the bottom right of the configuration area are two icons: a sun and a moon. At the very bottom of the page are two buttons: "DOWNLOAD CTRL + ⌘" and "CLOSE ESC".

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.5.6</version>
    <relativePath/> <!-- Lookup parent from repository -->
  </parent>
  <groupId>org.sample.devops</groupId>
  <artifactId>authentication-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Authentication Sservice</name>
  <description>Demo project for Spring Boot</description>
  <url/>
  <licenses>
    <license/>
  </licenses>
  <developers>
    <developer/>
  </developers>
  <scm>
    <connection/>
    <developerConnection/>
    <tag/>
    <url/>
  </scm>
  <properties>
```

Copiez les dépendances dans le pom.xml

IntelliJ



The screenshot shows the IntelliJ IDEA interface with the project navigation bar on the left and the code editor on the right. The project structure on the left includes a folder named 'authentication-service' containing '.idea', '.mvn', 'src' (with 'main' and 'test' subfolders), 'application.yml', 'pom.xml', '.gitignore', and 'External Libraries'. The code editor on the right displays the 'pom.xml' file for the 'authentication-service' project. The XML code defines the Maven project with its group ID, artifact ID, version, and parent. It also specifies the Java version (17) and includes dependencies for Spring Boot Starter Web and Spring Boot Starter Parent.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.5.6</version>
    <relativePath/> 
  </parent>
  <groupId>org.sample.devops</groupId>
  <artifactId>authentication-service</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Authentication Service</name>
  <description>Demo project for Spring Boot</description>
  <url/>
  <licenses>
    <license/>
  </licenses>
  <developers>
    <developer/>
  </developers>
  <scm>
    <connection/>
    <developerConnection/>
    <tag/>
    <url/>
  </scm>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
```

Ajouter un fichier de config application.yml

IntelliJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** On the left, the project structure for "authentication-service" is shown. It includes .idea, .mvn, src (with main and resources), test (with java), target, .gitignore, pom.xml, External Libraries, and Scratches and Consoles.
- Editor:** The right side shows the contents of the application.yml file. The code is as follows:

```
spring:
  application:
    name: Authentication Service
  server:
    port: 8080
```
- Annotations:** Two large white arrows point from the text "Application name" and "port" to the corresponding lines in the application.yml file.

Ajouter la lanceur Main

IntelliJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** On the left, the project structure for "authentication-service" is shown. It includes a .idea folder, a .mvn folder, a src directory with main and test Java sub-directories, resources containing an application.yml file, and a target directory.
- Code Editor:** The main window displays the `MainRunnerApplication.java` file. The code is as follows:

```
1 package org.sample.devops.service.authentication;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class MainRunnerApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MainRunnerApplication.class, args);
11     }
12 }
```

- Annotations:** Two orange arrows point from the text "Main" at the bottom left to the `main` method in the code editor.

Lancez l'application

IntelliJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** Shows the project name "authentication-service" and branch "master".
- Project Tree:** Displays the project structure under "authentication-service". Key components include ".idea", ".mvn", "src" (containing "main" and "java" packages), "resources" (with "application.yml" and "start_my_application.sh"), and "target" (containing generated artifacts like "authentication-service-0.0.1-SNAPSHOT.jar").
- Code Editor:** The "MainRunnerApplication.java" file is open. The code is as follows:

```
1 package org.sample.devops.service.authentication;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class MainRunnerApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(MainRunnerApplication.class, args);
11     }
12 }
```

An orange arrow originates from the "Run application" button in the top right corner of the IDE and points to the first line of the main method (`public class MainRunnerApplication {`).

Run application

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.

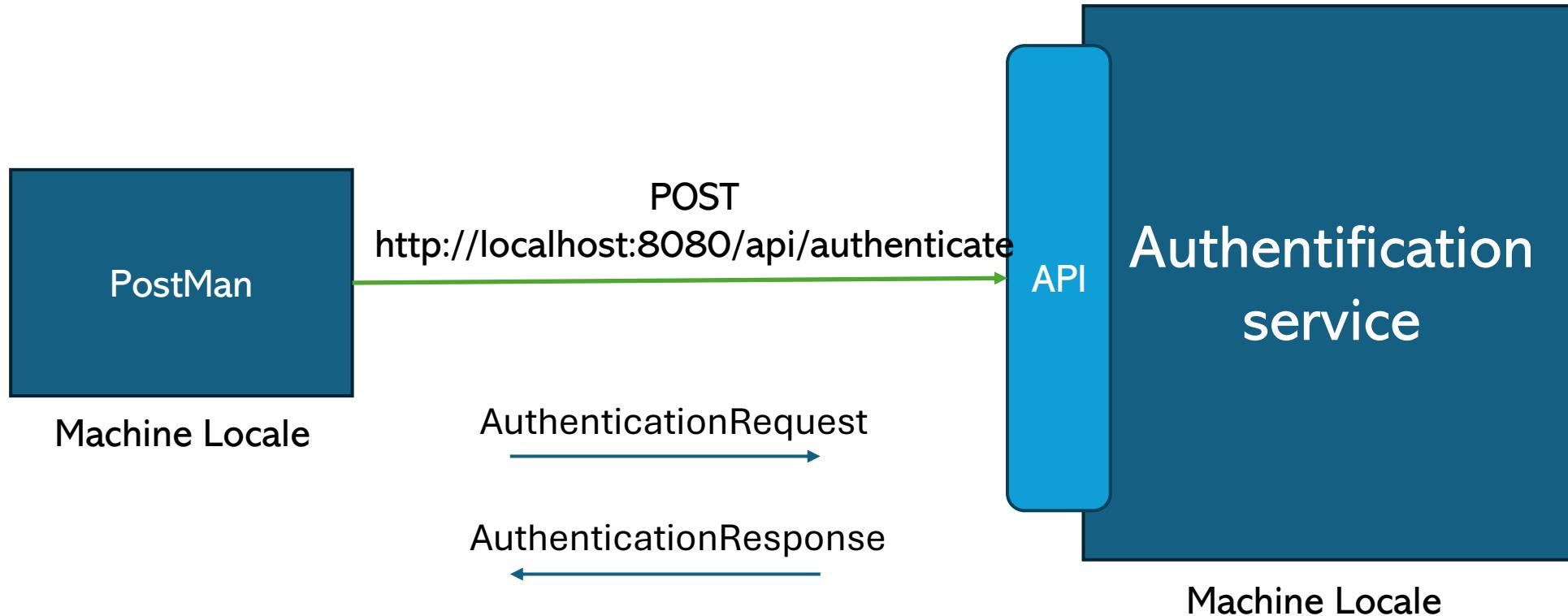
Sun Sep 28 21:15:31 CEST 2025

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

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "authentication-service". The "src/main/java/org.sample.devops.service.authentication.dto" package is expanded, and the "AuthenticationRequest.java" file is selected.
- Code Editor:** Displays the code for the "AuthenticationRequest" class:

```
1 package org.sample.devops.service.authentication.dto;
2
3 public record AuthenticationRequest (String mail, String password){}
```
- Toolbars and Status:** The top bar includes tabs for pom.xml, application.yml, MainRunnerApplication.java, and AuthenticationRequest.java. The status bar at the bottom shows "File | Open | Help | Exit".

Création du DTO AuthenticationResponse

IntelliJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure for "authentication-service" located at C:\Users\jamel\IdeaProjects\authentication-service. It includes .idea, .mvn, src (with main and test), resources (application.yml), and target.
- Code Editor:** Displays the file AuthenticationResponse.java with the following code:

```
package org.sample.devops.service.authentication.dto;
public record AuthenticationResponse(String mail, String firstname, String lastname) {
```
- Toolbars and Status Bar:** The top bar shows icons for pom.xml, application.yml, MainRunnerApplication.java, AuthenticationRequest.java, and AuthenticationResponse.java. The status bar at the bottom indicates "File | Open Recent | Help".

Création du Bean User

IntelliJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** On the left, the project structure for "authentication-service" is shown. It includes a .idea folder, a .mvn folder, a src directory containing main and test Java packages, resources, and a target folder.
- User.java File:** The file "User.java" is open in the editor. The code defines a class "User" with fields for mail, firstname, lastname, and password, and corresponding getters and setters.
- Editor:** The code editor shows the following Java code:

```
package org.sample.devops.service.authentication.db;

public class User {

    private final String mail;
    private final String firstname;
    private final String lastname;
    private final String password;

    public User(String mail, String firstname, String lastname, String password) {
        this.mail = mail;
        this.firstname = firstname;
        this.lastname = lastname;
        this.password = password;
    }

    public String getMail() {
        return mail;
    }

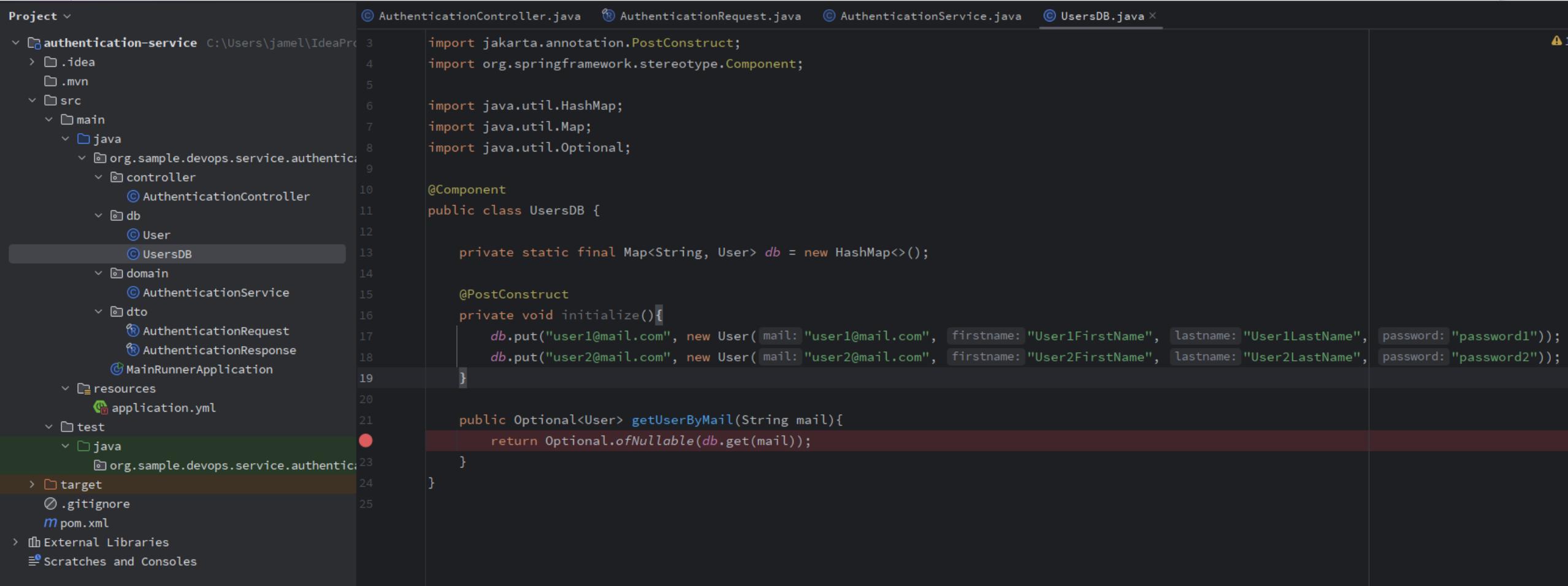
    public String getFirstname() {
        return firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public String getPassword() {
        return password;
    }
}
```

Création de la DB User

IntelliJ



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "authentication-service". The "src/main/java/org.sample.devops.service.authentication" package is expanded, showing the "AuthenticationController.java", "UserService.java", and "UsersDB.java" files.
- Code Editor:** The "UsersDB.java" file is open. The code implements a static map to store users by email. It includes a constructor and a method to get a user by email.

```
import jakarta.annotation.PostConstruct;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

@Component
public class UsersDB {

    private static final Map<String, User> db = new HashMap<>();

    @PostConstruct
    private void initialize(){
        db.put("user1@mail.com", new User( mail: "user1@mail.com", firstname: "User1FirstName", lastname: "User1LastName", password: "password1"));
        db.put("user2@mail.com", new User( mail: "user2@mail.com", firstname: "User2FirstName", lastname: "User2LastName", password: "password2"));
    }

    public Optional<User> getUserByMail(String mail){
        return Optional.ofNullable(db.get(mail));
    }
}
```

- Status Bar:** Shows a warning icon with the number 1, indicating one unresolved reference or error in the code.

Création du service domaine

IntelliJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "authentication-service". Key components include:
 - src/main/java/org.sample.devops.service.authentication/controller:** Contains `AuthenticationController`.
 - src/main/java/org.sample.devops.service.authentication/db:** Contains `User` and `UsersDB`.
 - src/main/java/org.sample.devops.service.authentication/domain:** Contains `AuthenticationService`.
 - src/main/java/org.sample.devops.service.authentication/dto:** Contains `AuthenticationRequest` and `AuthenticationResponse`.
 - src/main/java/org.sample.devops.service.authentication/MainRunnerApplication**
 - resources/application.yml**
- Code Editor:** Displays the `AuthenticationService.java` file. The code implements a service that authenticates users based on their email and password, using a `UsersDB` dependency.

```
import java.util.Optional;
@Service
public class AuthenticationService {
    private final UsersDB userDB;
    public AuthenticationService(UsersDB userDB){
        this.userDB = userDB;
    }
    public Optional<User> authenticate(String mail, String password){
        Optional<User> user = this.userDB.getUserByMail(mail);
        if (user.isPresent()){
            if (user.get().getPassword().equals(password)){
                return user;
            }
        }
        return Optional.empty();
    }
}
```

Création du Controller Rest

IntelliJ

```
Project authentication-service C:\Users\jamel\IdeaProjects\authentication-service AuthenticationController.java AuthenticationRequest.java AuthenticationService.java UsersDB.java
src/main/java/org.sample.devops.service.authentication.controller.AuthenticationController.java
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RestController;
13
14 import java.util.Optional;
15
16 @RestController
17 @RequestMapping("/api")
18 public class AuthenticationController {
19
20     private final AuthenticationService authenticationService;
21
22     public AuthenticationController(AuthenticationService authenticationService) {
23         this.authenticationService = authenticationService;
24     }
25
26
27     @PostMapping("/authenticate")
28     public ResponseEntity<AuthenticationResponse> authenticate(@RequestBody AuthenticationRequest authenticationRequest) {
29         Optional<User> userOptional = authenticationService.authenticate(authenticationRequest.mail(), authenticationRequest.password());
30         if (userOptional.isPresent()) {
31             User user = userOptional.get();
32             AuthenticationResponse authenticationResponse = new AuthenticationResponse(user.getMail(), user.getFirstname(), user.getLastname());
33             return ResponseEntity.ok(authenticationResponse);
34         } else {
35             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
36         }
37     }
38 }
39
```

URL de l'Endpoint

Objet en entrée

Objet en sortie

Lancement du Projet

IntelliJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "authentication-service". Key files include `AuthenticationController.java`, `AuthenticationRequest.java`, `AuthenticationService.java`, and `UsersDB.java`.
- Code Editor:** Displays the `AuthenticationController.java` file, which contains Java code for a REST controller using Spring framework annotations like `@RestController`, `@RequestMapping`, and `@PostMapping`.
- Run Tab:** Set to run the `MainRunnerApplication`.
- Terminal:** Shows the command line interface with Spring Boot startup logs. The logs indicate the application is starting using Java 17.0.2, no active profile is set, Tomcat is initialized with port 8080, and the application is started in 4.185 seconds.

Testez l'Endpoint

Postman
http://localhost:8080

POST

Request

Réponse “User”

URL de l'Endpoint

JSON

The screenshot shows the Postman interface with a POST request to `http://localhost:8080/api/authenticate`. The request body is set to `JSON` and contains the following data:

```
1 "mail": "user1@mail.com",
2 ...
3 "password": "password1"
4 }
```

The response status is `200 OK`.

The screenshot shows the Postman interface with the response body displayed in `Pretty` format. The response contains the following data:

```
1 {
2   "mail": "user1@mail.com",
3   "firstname": "User1FirstName",
4   "lastname": "User1LastName"
5 }
```

Testez l'Endpoint

Postman

The screenshot shows the Postman interface with a failed API request and its response.

Request:

- Method: POST
- URL: <http://localhost:8080/api/authenticate>
- Body (JSON):

```
1 {  
2   "mail": "user1@mail.com",  
3   "password": "password"  
4 }
```

Response:

- Status: 404 Not Found
- Time: 258 ms
- Size: 130 B

A red arrow points from the error message "Mot de passe incorrect" to the "password" field in the JSON body. Another red arrow points from the "Erreur 404" text to the status code in the response header.

Ajoutez un test Unitaire

IntelliJ

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** On the left, the project structure for "authentication-service" is displayed. It includes the following packages:
 - src/main/java:** Contains controller, db, domain, and dto sub-packages. The db package contains User and UsersDB classes; the domain package contains AuthenticationService; the dto package contains AuthenticationRequest and AuthenticationResponse.
 - src/test/java:** Contains org.sample.devops.service sub-package with HttpUnitTest, SmokeUnitTest, and TestingWebApplicationTest classes.
 - resources:** Contains application.yml.
 - target:** Contains .gitignore and pom.xml.
- Code Editor:** The right pane shows the code for **TestingWebApplicationTests.java**. The code is as follows:

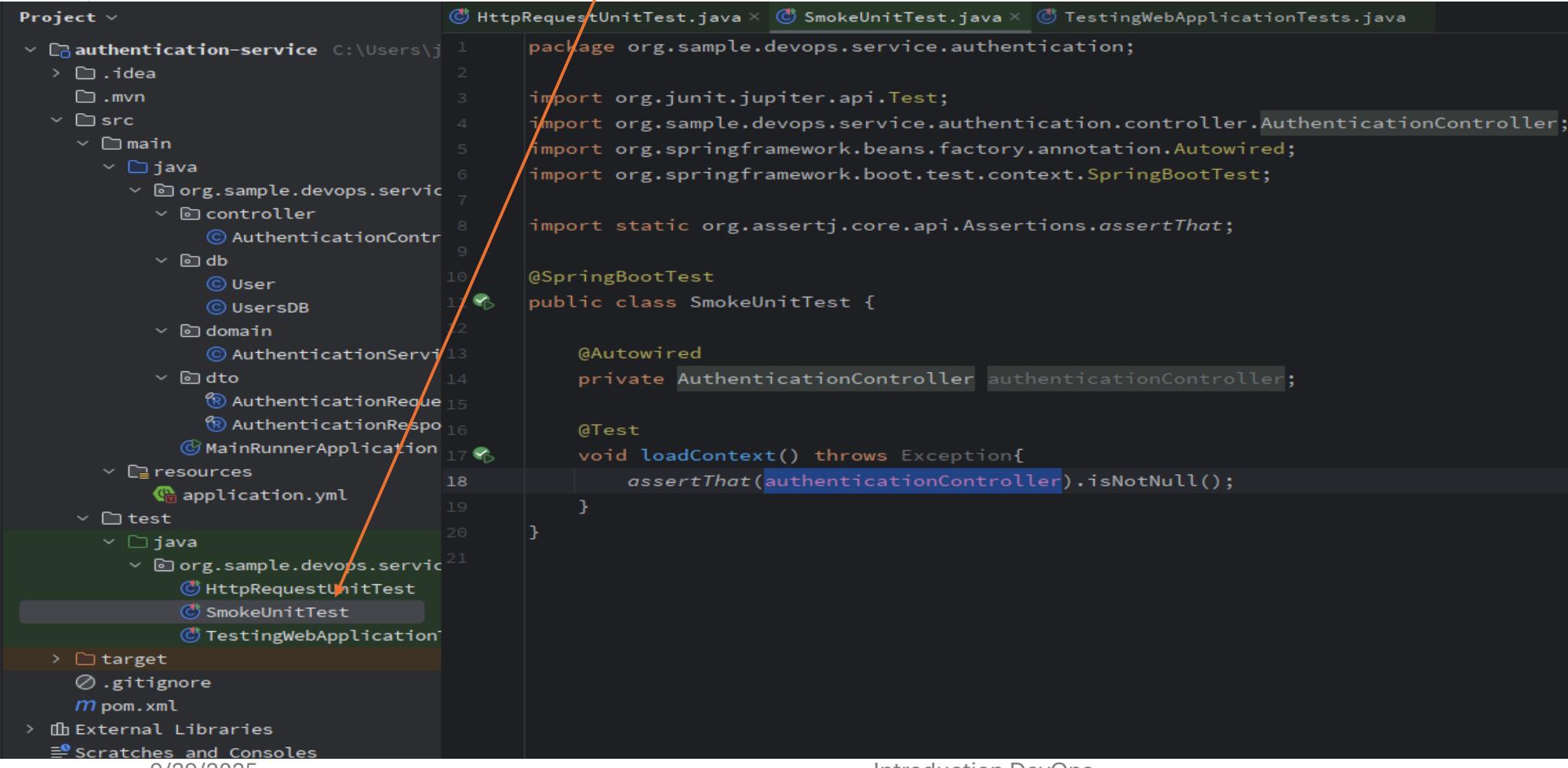
```
package org.sample.devops.service.authentication;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest
public class TestingWebApplicationTests {
    @Test
    void contextLoading(){
    }
}
```
- Toolbars:** Standard IntelliJ toolbars for file operations, search, and navigation are visible at the top.

Lancez le test Unitaire

IntelliJ

Ajoutez un test Unitaire

IntelliJ



Project

- authentication-service C:\Users\j...
 - .idea
 - .mvn
 - src
 - main
 - java
 - org.sample.devops.service.controller
 - AuthenticationController
 - db
 - User
 - UsersDB
 - domain
 - AuthenticationService
 - dto
 - AuthenticationRequest
 - AuthenticationResponse
 - MainRunnerApplication
 - resources
 - application.yml
 - test
 - java
 - org.sample.devops.service
 - HttpRequestUnitTest
 - SmokeUnitTest
 - TestingWebApplication
 - target
 - .gitignore
 - pom.xml
 - External Libraries
 - Scratches and Consoles

HttpRequestUnitTest.java × SmokeUnitTest.java × TestingWebApplicationTests.java

```
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 }
```

9/29/2025

Ajoutez un test Unitaire

IntelliJ

The screenshot shows the IntelliJ IDEA interface with a Java unit test file open. The left sidebar displays the project structure for an 'authentication-service' project, including source code in 'src/main/java' and test code in 'src/test/java'. The 'HttpRequestUnitTest.java' file is selected in the test directory. The code implements two tests for an authentication endpoint, using Spring Boot Test annotations like @SpringBootTest and @LocalServerPort. It uses a TestRestTemplate to send POST requests to '/api/authenticate' and assert the response status and body.

```
HttpRequestUnitTest.java
import java.net.URI;
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 shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under `org.sample.devops.service`, including `controller`, `db`, `domain`, `dto`, and `resources` packages, along with `MainRunnerApplication` and configuration files `application.yml`.
- Code Editor:** Displays `HttpRequestUnitTest.java` with the following code:import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Test;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.boot.test.web.server.LocalServerPort;
import org.springframework.test.context.junit.jupiter.SpringExtension;

@ExtendWith(SpringExtension.class)
@LocalServerPort
public class HttpRequestUnitTest {

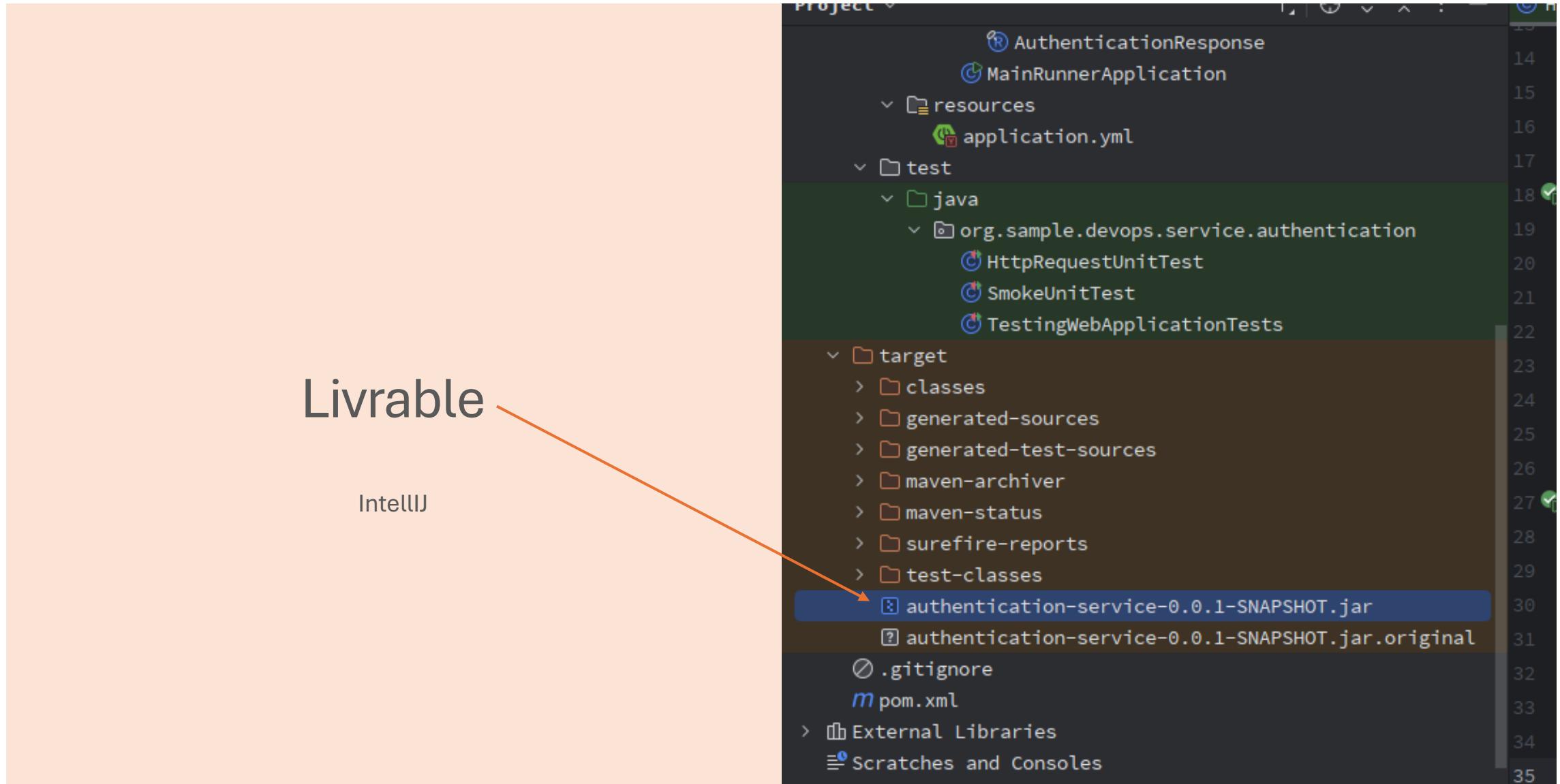
 private TestRestTemplate restTemplate;

 @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);
 }

 @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: "incorrect");
 ResponseEntity<AuthenticationResponse> response = this.restTemplate.postForEntity(uri, request, AuthenticationResponse.class);
 assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
 }
}
- Run Tab:** Shows the test class `HttpRequestUnitTest` selected.
- Output Tab:** Displays the test results:

```
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



The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays the file structure of the 'authentication-service' project. A red arrow points from the 'start_my_application.sh' file in the 'src/main/resources' directory up towards the terminal window. The terminal window at the top right shows a shell script with three commands:

```
1 cd ../../..../target
2
3 java -jar authentication-service-0.0.1-SNAPSHOT.jar
```

Script de lancement du livrable

IntelliJ



:: Spring Boot :: (v3.5.6)

```
2025-09-30T08:47:46.964+02:00 INFO 25796 --- [Authentication Service] [main] o.s.d.s.a.MainRunnerApplication : Starting MainRunnerApplication v0.0.1-SNAPSHOT using Java 17.0.2 with PID 25796 (C:\Users\jamel\IdeaProjects\authentication-service\target\authentication-service-0.0.1-SNAPSHOT.jar started by jamel in C:\Users\jamel\IdeaProjects\authentication-service\target)
2025-09-30T08:47:46.971+02:00 INFO 25796 --- [Authentication Service] [main] o.s.d.s.a.MainRunnerApplication : No active profile set, falling back to 1 default profile: "default"
2025-09-30T08:47:48.089+02:00 INFO 25796 --- [Authentication Service] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-09-30T08:47:48.104+02:00 INFO 25796 --- [Authentication Service] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-09-30T08:47:48.104+02:00 INFO 25796 --- [Authentication Service] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]
2025-09-30T08:47:48.150+02:00 INFO 25796 --- [Authentication Service] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-09-30T08:47:48.151+02:00 INFO 25796 --- [Authentication Service] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1106 ms
```

Lancement du script

IntelliJ

Testez l'Endpoint

Postman
http://localhost:8080

POST

Request

Réponse “User”

URL de l'Endpoint

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

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {  
2   "mail": "user1@mail.com",  
3   "password": "password1"  
4 }
```

Send Save Cookies Beautify

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

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "mail": "user1@mail.com",  
3   "firstname": "User1FirstName",  
4   "lastname": "User1LastName"  
5 }
```

Deployment manuelle sur chaque serveur

Scripting

