МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни
«АРХІТЕКТУРА І ТЕХНОЛОГІЇ ВЕБСЕРВІСІВ»

для магістрів спеціальності F7 «Комп’ютерна інженерія»
всіх форм навчання

Частина I

2025

Методичні вказівки до виконання лабораторних робіт з дисципліни «Архітектура і технології вебсервісів» для магістрів спеціальності F7 «Комп'ютерна інженерія», всіх форм навчання. Частина I / Укл. М.Б. Ільяшенко – Запоріжжя: НУ «Запорізька політехніка», 2025. – 70с.

Укладачі : М.Б. Ільяшенко, доцент, к.т.н.

Рецензент : Г.Г. Киричек, доцент, к.т.н.

Відповідальний
за випуск : М.Б. Ільяшенко, доцент, к.т.н.

Затверджено:
На засіданні кафедри
«Комп'ютерні системи та мережі»
Протокол № 2
від 29 серпня 2025 р.

Рекомендовано до видання
НМК факультету КНТ
Протокол № 2
від 10 вересня 2025 р.

ЗМІСТ

С.

Вступ 4

1 Лабораторна робота № 1 – Простий застосунок Spring Boot 5

1.1 Теоретичні відомості 5

1.2 Хід роботи 12

1.3 Завдання до лабораторної роботи 22

1.4 Зміст звіту 22

1.5 Контрольні питання 22

2 Лабораторна робота № 2 – Розробка SOAP вебсервісу 24

2.1 Теоретичні відомості 24

2.2 Хід роботи 36

2.3 Завдання до лабораторної роботи 68

2.4 Зміст звіту 68

2.5 Контрольні питання 68

Перелік джерел посилання 70

ВСТУП

Методичні матеріали з дисципліни «Архітектура і технології вебсервісів» присвячені вивченню основ Spring Boot – сучасного фреймворку на базі Java, який значно спрощує створення вебзастосунків та сервісів. Перша частина покриває питання побудови мікросервісів, що взаємодіють між собою за допомогою архітектури SOAP. SOAP (Simple Object Access Protocol) є одним із найстаріших і водночас фундаментальних протоколів для обміну повідомленнями в розподілених системах. Незважаючи на поширення REST, SOAP усе ще активно використовується в корпоративних рішеннях, зокрема там, де критично важливі стандартизованість, безпека та транзакційність.

Робота з SOAP із фреймворком Spring Boot дає змогу студентам зрозуміти, як проєктуються клієнтські та серверні застосунки, які обмінюються даними через XML–повідомлення, використовуючи строго визначені контракти (WSDL). У межах лабораторних завдань студенти навчяться створювати серверну частину вебсервісу, описувати її контракт, а також реалізовувати клієнт, здатний звертатися до цього сервісу й отримувати коректні відповіді.

Особливу увагу буде приділено налаштуванню середовища, основам конфігурації Spring Boot, генерації класів із WSDL, роботі з маршалінгом та анмаршалінгом XML–даних, а також перевірці працездатності клієнтських і серверних компонентів. Це забезпечить практичне розуміння того, як SOAP–сервіси інтегруються в більші системи, і сформує ґрунтовні знання для подальшого вивчення більш гнучких архітектур.

У наступних частинах методичних вказівок з дисципліни «Архітектура і технології вебсервісів» будуть розглянуті питання побудови REST API на основі фреймворку Spring Boot, що дозволить студентам поступово ознайомитися як із класичними підходами до вебсервісів, так і з сучасними практиками, які сьогодні є більш поширеними в архітектурі мікросервісів.

1 ЛАБОРАТОРНА РОБОТА № 1 – ПРОСТИЙ ЗАСТОСУНОК SPRING BOOT

Мета роботи: одержати знання та навички написання веб додатків із використання фреймворку Spring Boot [1–3].

1.1 Теоретичні відомості

Spring Boot – це сучасний фреймворк, покликаний зробити розробку Java–застосунків простішою, швидшою та ефективнішою. Він не замінює класичний Spring, а навпаки – розширює його можливості, пропонуючи набір «розумних» налаштувань і готових інструментів. Завдяки цьому розробнику не потрібно витрачати час на громіздкі конфігурації та шаблонний код. Основний принцип, на якому базується Spring Boot, – «конвенція понад конфігурацію»: застосунок відразу працює «з коробки» з оптимальними налаштуваннями, але їх завжди можна адаптувати під конкретні потреби.

Ключові переваги Spring Boot:

– автоконфігурація – автоматичне підлаштування середовища залежно від доданих бібліотек (наприклад, при підключенні Spring Data JPA одразу налаштовується DataSource та Hibernate);

– starter–залежності – готові набори залежностей (spring–boot–starter–*), які спрощують підключення типових технологій (Web, JPA, Security, Test тощо);

– вбудовані веб–сервери – підтримка Tomcat, Jetty та Undertow без додаткових налаштувань; застосунок можна запускати як самостійний JAR із сервером усередині;

– єдина точка входу – анотація @SpringBootApplication поєднує автоконфігурацію, сканування компонентів і підключення конфігураційних класів;

– зовнішня конфігурація – підтримка файлів application.properties чи application.yml, профілів середовища (dev, prod), змінних оточення та параметрів запуску;

– actuator – набір готових endpoints для моніторингу, збору метрик і адміністрування застосунку;

– devTools – інструменти для зручної розробки: автоматичний перезапуск, «гаряче» оновлення ресурсів і спрощене кешування під час розробки;

– інтеграція зі Spring Data та Spring Security – готові рішення для роботи з базами даних, створення REST API та побудови безпечних застосунків;

– спрощене тестування – стартер spring–boot–starter–test, підтримка JUnit 5, Mockito, а також анотацій @SpringBootTest, @DataJpaTest, @WebMvcTest та інших;

– spring Initializr – вебсервіс, що дозволяє за лічені хвилини створити готовий шаблон проєкту з усіма потрібними залежностями.

1.1.1 Автоконфігурація (Auto–configuration)

Однією з ключових переваг Spring Boot є механізм автоконфігурації. Саме він робить роботу з фреймворком максимально зручною, адже дозволяє запускати застосунок практично без додаткових налаштувань.

Автоконфігурація аналізує classpath (класи, доступні у вашому проєкті та його залежностях) і, залежно від цього, автоматично створює необхідні біни та підключає конфігурацію. Наприклад:

– якщо у classpath є залежність spring–boot–starter–data–jpa і налаштовано джерело даних, Spring Boot автоматично підніме EntityManagerFactory, DataSource і створить потрібні репозиторії;

– якщо додано spring–boot–starter–web, буде налаштований DispatcherServlet, механізми маршрутизації запитів (RequestMappingHandlerMapping), серіалізація JSON через Jackson, а також вбудований вебсервер.

Технічно автоконфігурація реалізована через набір класів із анотацією @Configuration, які підключаються лише за певних умов. Це забезпечується спеціальними умовними анотаціями на кшталт @ConditionalOnClass, @ConditionalOnProperty та інших. Завдяки цьому вдається поєднати два підходи: «все працює відразу» та «налаштування можна гнучко змінювати під потреби».

1.1.2 Starter–залежності

Однією з важливих особливостей Spring Boot є концепція starter–залежностей. У класичному підході розробнику доводилося самостійно підбирати та підключати численні бібліотеки, стежити за їх сумісністю та версіями. Наприклад, для створення веб–застосунку потрібно було окремо додавати Spring MVC, Jackson для роботи з JSON, вбудований сервер і ще низку компонентів.

Spring Boot значно спрощує цей процес, пропонуючи готові набори бібліотек – стартери. Кожен стартер – це попередньо згрупована колекція залежностей, орієнтована на певну функціональність, що дозволяє швидко підключити потрібний стек технологій без зайвої рутинної роботи.

Наприклад, щоб створити веб–застосунок на Spring Boot, достатньо підключити стартер spring–boot–starter–web. Він автоматично підключає необхідні компоненти: Spring MVC, Jackson для обробки JSON, вбудований сервер Tomcat (за замовчуванням) та інші потрібні бібліотеки. Таким чином розробник звільняється від необхідності самостійно підбирати і координувати численні залежності – Spring Boot робить це автоматично:

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

1.1.3 Вбудовані веб–сервери

Однією з найзручніших можливостей Spring Boot є підтримка вбудованих веб–серверів. У традиційній Java–розробці для запуску веб–застосунку потрібно було розгортати його на окремому сервлет–контейнері, наприклад, Tomcat або Jetty. Це вимагало додаткових дій: встановлення та налаштування сервера, вибору порту, створення WAR–файлу та його деплою.

У Spring Boot усе змінено: сервер інтегрований безпосередньо в застосунок і запускається разом із ним. Веб–застосунок перетворюється на самодостатній JAR–файл**,** який можна виконати однією командою:

```
java –jar myapp.jar
```

За замовчуванням використовується Tomcat, проте за потреби його можна замінити на Jetty або Undertow, просто змінивши залежності у `pom.xml`.

Такий підхід значно спрощує розробку та розгортання: більше не потрібно налаштовувати зовнішній сервер, достатньо зібрати застосунок і перенести його у будь–яке середовище з встановленою Java. Крім того, це забезпечує стабільну та передбачувану поведінку у різних середовищах (розробка, тестування, продакшн) і зменшує кількість помилок, пов’язаних з несумісністю серверів.

Основна анотація: @SpringBootApplication.

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Технічно це зручне поєднання трьох анотацій:

– @Configuration – дозволяє визначати біні;

– @EnableAutoConfiguration – увімкнення автоконфігурації;

– @ComponentScan – сканування компонентів у пакеті і підпакетах.

1.1.4 Запуск та життєвий цикл застосунку.

Метод SpringApplication.run(...) ініціалізує контекст Spring (ApplicationContext), запускає автоконфігурацію, реєструє біні, піднімає вбудований сервер (якщо це веб–застосунок), а також розгортає можливість підключення різних життєвих хуків: ApplicationRunner, CommandLineRunner – інтерфейси, які дозволяють виконувати код під час старту програми. Середовище (environment) додає джерела властивостей (property sources), які можна розширити: application.properties/application.yml, змінні оточення, аргументи командного рядка, та інші конфігураційні джерела.

1.1.5 Конфігурація і зовнішні властивості

Spring Boot робить акцент на externalized configuration: параметри, що впливають на поведінку додатка, зберігаються не в коді, а в зовнішніх файлах або джерелах. Найпоширеніші формати – application.properties або application.yml. Властивості можуть задаватися на різних рівнях: global, profile–specific (application–dev.yml), через змінні оточення або аргументи командного рядка. Spring Boot визначає пріоритет джерел (command–line > environment variables > application.properties).

Приклад application.yml:

```
server:
  port: 8081
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mydb
    username: user
    password: secret
```

Еквівалентний файл application.properties буде виглядати так:

```
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=user
spring.datasource.password=secret
```

1.1.6 Інтеграція зі Spring Data

Однією з ключових переваг Spring Boot є його тісна інтеграція зі Spring Data, що значно спрощує роботу з базами даних. Це дозволяє розробнику зосередитися на бізнес–логіці, не витрачаючи час на рутинну роботу з SQL–запитами або управлінням підключеннями.

За допомогою Spring Data JPA можна описувати сутності (entity) через анотації та працювати з ними через репозиторії. Більшість CRUD–операцій (створення, читання, оновлення, видалення) реалізовані автоматично. Тобто для отримання всіх записів із таблиці або пошуку за конкретним полем достатньо викликати методи стандартного інтерфейсу JpaRepository, без написання SQL–коду вручну.

Крім того, Spring Boot автоматично підхоплює налаштування бази даних із файлів application.properties або application.yml і конфігурує підключення через Hibernate. Для початку роботи з MySQL або іншою СКБД достатньо вказати URL, логін та пароль – усе інше фреймворк налаштує сам.

Також доступні механізми автоматичної генерації таблиць на основі сутностей, підтримка складних зв’язків між ними (OneToMany**,** ManyToMany тощо) і можливість формувати запити як через іменовані методи репозиторіїв, так і за допомогою JPQL або нативного SQL. Завдяки такій інтеграції робота з базою даних стає зрозумілою, ефективною та зручною для розробника.

1.1.7 Спрощене тестування додатків

У Spring Boot процес тестування значно спрощено завдяки вбудованим інструментам та інтеграціям. Розробникам доступний стартер spring–boot–starter–test, який «з коробки» підключає найпопулярніші бібліотеки для тестування, такі як JUnit 5, AssertJ, Mockito, Hamcrest та інші. Це дозволяє одразу почати створювати тести, без необхідності вручну додавати всі залежності чи налаштовувати середовище.

Фреймворк повністю підтримує модульне, інтеграційне **та** end–to–end тестування, що дає змогу перевіряти як окремі компоненти, так і роботу всього застосунку в цілому.

Велику роль відіграють спеціальні анотації: @SpringBootTest, @DataJpaTest, @WebMvcTest. Вони дозволяють завантажувати лише ті частини контексту застосунку, які потрібні для конкретного тесту. Наприклад, @DataJpaTest запускає середовище лише для перевірки роботи з базою даних, а @WebMvcTest – для тестування веб–рівня без підняття всього контексту.

Такий підхід значно економить час на виконання тестів і підвищує продуктивність розробки, роблячи тестування невід’ємною та зручною складовою життєвого циклу створення програмних продуктів.

1.1.8 Сервіс конфігурації Spring Initializr

Spring Initializr пропонує розробникам низку суттєвих переваг. По–перше, він значно прискорює підготовку нового проєкту: немає потреби вручну створювати структуру каталогів чи налаштовувати pom.xml або build.gradle – сервіс одразу формує готовий до роботи каркас застосунку.

По–друге, Initializr дозволяє обирати потрібні залежності та версії Spring Boot, що суттєво зменшує ризик конфліктів між бібліотеками. Крім того, сервіс підтримує різні мови програмування – Java, Kotlin, Groovy, а також різні системи

збірки (Maven, Gradle), що робить його універсальним для будь–яких команд розробників.

Сервіс доступний за посиланням: https://start.spring.io

1.2 Хід роботи

Створення Spring Boot програми

У цій лабораторній роботі розглядається процес створення та запуску базового додатку на Spring у середовищі IntelliJ IDEA. Проєкт реалізується як Spring Boot із використанням Gradle для керування залежностями і формується за допомогою Spring Initializr – це найшвидший спосіб розгорнути робочу структуру програми Spring. IntelliJ IDEA пропонує зручний майстер для створення такого проєкту, який значно спрощує старт роботи.

У рамках лабораторної роботи ви навчитеся налаштовувати HTTP–endpoint та зв'язувати його з методом контролера, який повертатиме привітання користувачеві під час звернення через веб–браузер.

Створення нового проєкту Spring Boot з використанням майстра у складі IntelliJ IDEA Ultimate.

Для виконання цієї лабораторної роботи рекомендується використовувати IntelliJ IDEA Ultimate. Щоб скористатися академічною ліцензією, потрібно зареєструвати обліковий запис, використовуючи електронну пошту університету з доменом @zp.edu.ua. Компанія JetBrains надає безкоштовну академічну ліцензію студентам і викладачам, якщо обліковий запис зареєстрований на корпоративний або університетський email, і наш університет бере участь у цій програмі.

Якщо немає можливості отримати Ultimate–версію, можна скористатися альтернативним способом створення проєкту без ліцензії Ultimate:

– у головному меню IntelliJ IDEA оберіть File → New → Project;

– на лівій панелі майстра створення проєкту виберіть Spring Initializr;

– вкажіть ім'я проєкту, наприклад, spring–boot;

– у списку JDK оберіть опцію Download JDK та завантажте останню версію Oracle OpenJDK. Рекомендовано використовувати Java 17, оскільки Gradle на даний момент ще не повністю сумісний з Java 19.

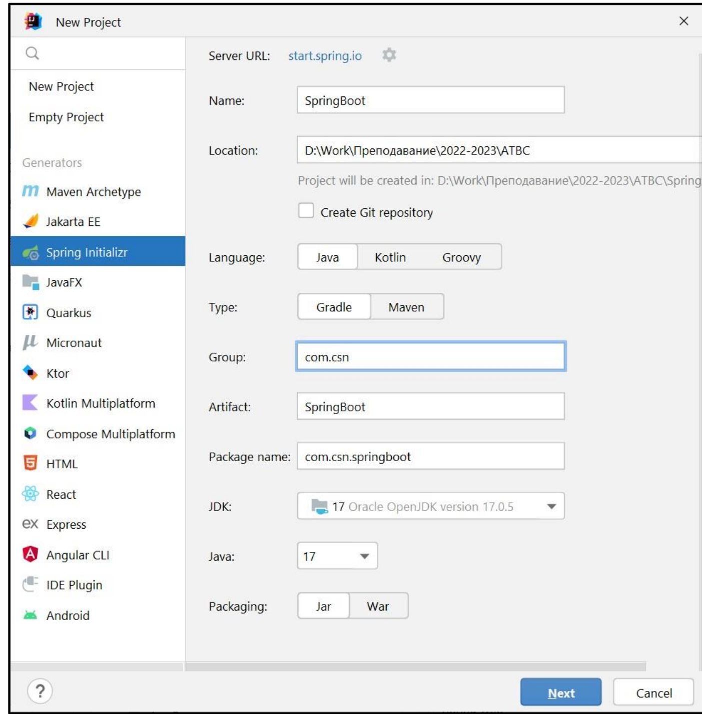

Рисунок 1.1 – Вікно налаштування нового проєкту із Spring Initializr

Натисніть Next, щоб продовжити.

Виберіть залежність Spring Web у розділі Web. Ця залежність потрібна для будь–якої веб–програми, що використовує Spring MVC.

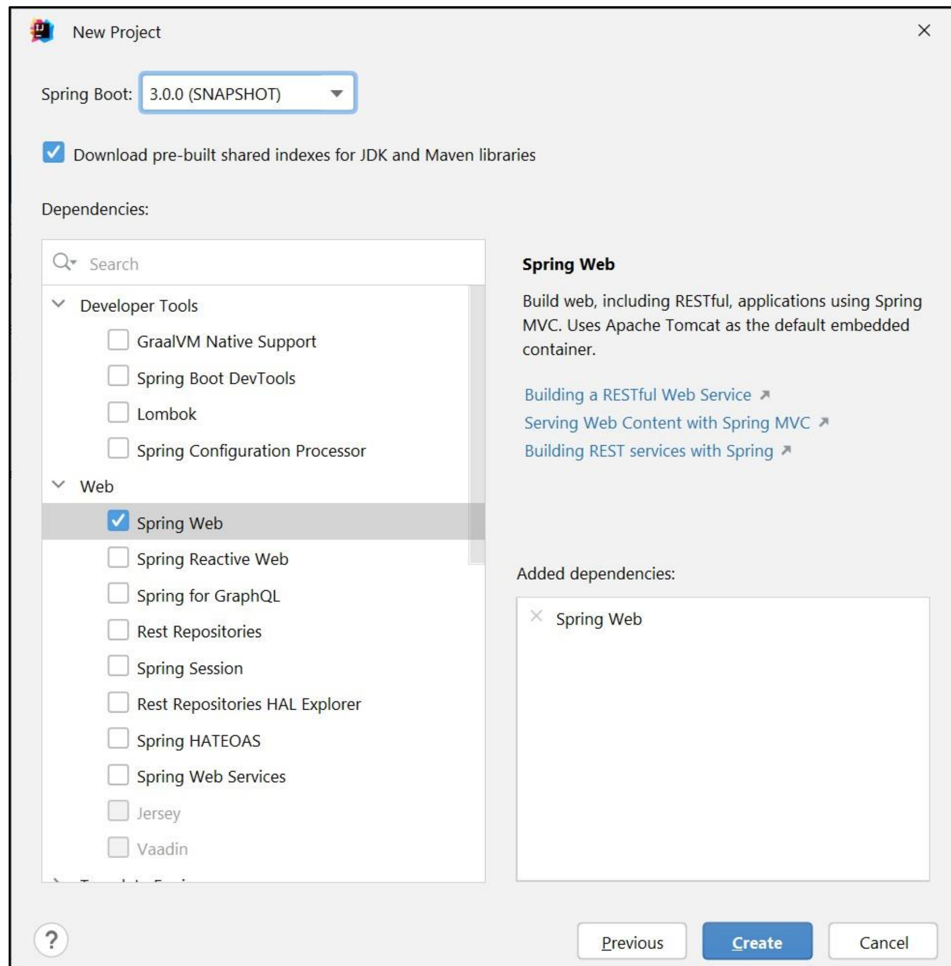

Рисунок 1.2 – Вікно налаштування залежностей проєкту

Натисніть Create, щоб згенерувати та налаштувати проєкт.

1.2.1 Використання генератора Spring Initializr

У цьому випадку цілком підійде IntelliJ IDEA Community Edition (безкоштовна версія). Відкрийте Spring Initializr за адресою https://start.spring.io та оберіть необхідні параметри для створення проєкту.

У нашому прикладі ми створюємо веб–застосунок, тому потрібно поставити прапорець Web і натиснути Generate. Після цього розпочнеться завантаження zip–архіву з проєктом. Розпакуйте його, і ви отримаєте базову структуру каталогів Maven або Gradle, яка вже містить відповідні файли конфігурації: pom.xml для Maven або build.gradle для Gradle. У подальшому прикладі буде використано Maven.

Рисунок 1.3 – Параметри налаштування проєкту сервісом Spring Initializr

Щоб отримати архів із конфігурацією проєкту, натисніть Generate (або скористайтеся комбінацією Ctrl+Enter). Також є можливість заздалегідь переглянути вміст архіву, що буде завантажено, натиснувши Explore (або Ctrl+Space).

Рисунок 1.4 – Приклад коду файла pom.xml налаштувать проєкту за допомогою системи збірки Maven

Після розпакування архіву створіть новий проєкт у IntelliJ IDEA Community Edition, оберіть в меню File → New → Project from Existing Sources, а потім вкажіть папку з розпакованими файлами. Під час імпорту вкажіть систему збірки

Maven (зазвичай вона визначається автоматично на основі наявних файлів проєкту).

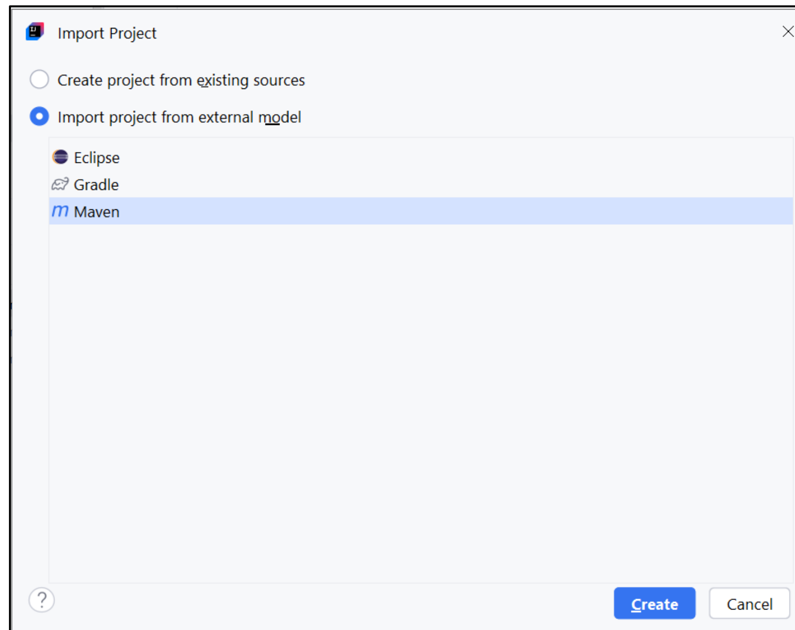

Рисунок 1.5 – Вікно вибору системи збірки

Імпортований проєкт повинен виглядати так:

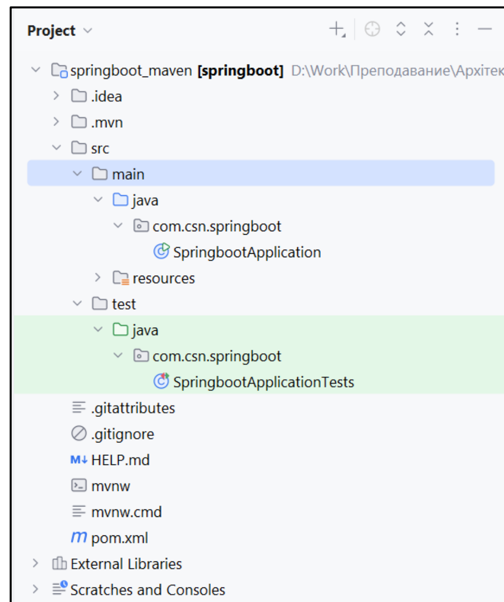

Рисунок 1.6 – Структура створеного проєкту

1.2.2 Додавання методу, який надсилає привітання

Spring Initializr створює клас з методом main() для початкового завантаження вашої програми Spring. Ми додамо метод sayHello() безпосередньо до цього класу.

Відкрийте файл SpringBootApplication.java у папці src/main/java/com/csn/springboot. У IntelliJ IDEA є дія «Перейти до файлу» для швидкого пошуку та відкриття файлів. У головному меню виберіть Навігація | File або натисніть Ctrl+Shift+N, почніть вводити ім’я файлу та виберіть його зі списку.

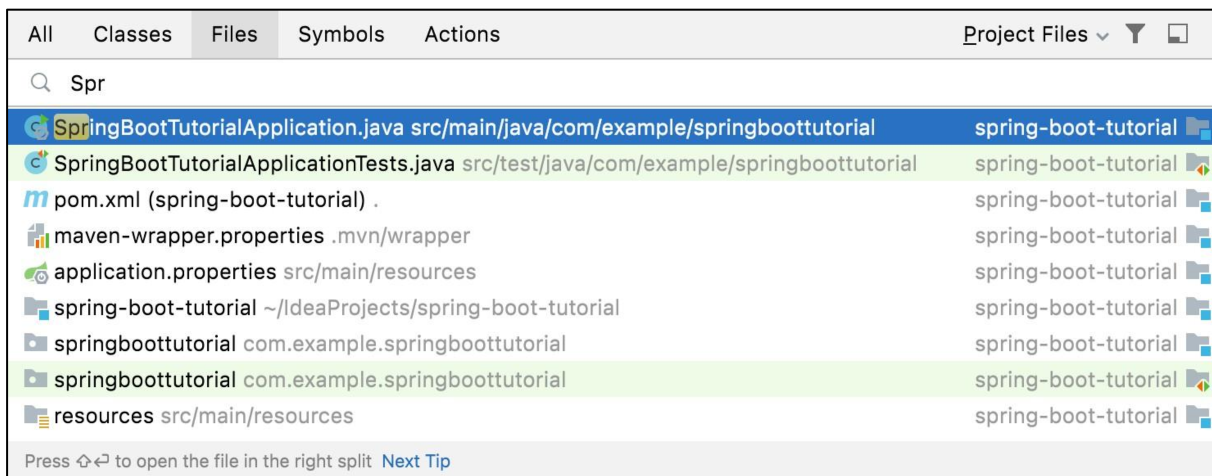

Рисунок 1.7 – Спливаюче вікно пошуку файлів та класів в проєкті

Додайте метод sayHello() з усіма необхідними анотаціями та імпортами, щоб файл виглядав як на лістингу 1.1:

Лістинг 1.1 – Код основного класу SpringbootApplication

```
package com.csn.springboot;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
```

Кінець лістингу 1.1

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import
org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class SpringbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringbootApplication.class,
args);
    }

    @GetMapping("/hello")
    public String sayHello(@RequestParam(value = "myName",
defaultValue = "World") String name) {
        return String.format("Hello %s!", name);
    }
}
```

Метод sayHello() приймає параметр name та повертає слово Hello у поєднанні зі значенням параметра. Все інше обробляється додаванням анотацій Spring:

– анотація @RestController позначає клас SpringBootApplication як обробник запитів (контролер REST);

– анотація @GetMapping("/hello") зіставляє метод sayHello() із запитами GET для /hello;

– анотація @RequestParam зіставляє параметр методу імені з параметром веб–запиту myName. Якщо ви не вкажете параметр myName у своєму веб–запиті, за замовчуванням він дорівнює World.

1.2.3 Запуск SpringBoot програми

IntelliJ IDEA створює конфігурацію запуску Spring Boot, яку можна використовувати для запуску нової програми Spring.

– якщо вибрано конфігурацію запуску, натисніть Shift+F10;

– ви також можете використовувати піктограму «Виконати» у нижній частині файлу SpringBootApplication.java поряд з оголошенням класу або оголошенням методу main().

За замовчуванням IntelliJ IDEA показує запущену програму Spring Boot у вікні інструмента «Виконати».

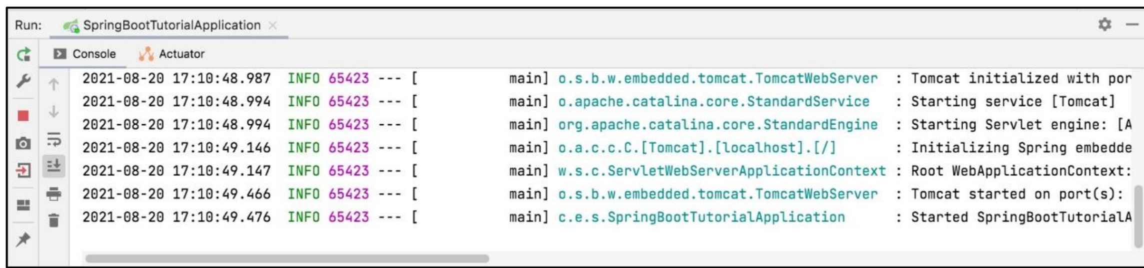

Рисунок 1.8 – Приклад консольного виводу після запуску проєкта

На вкладці «Консоль» відображаються повідомлення журналу Spring. За замовчуванням вбудований сервер Apache Tomcat прослуховує порт 8080. Відкрийте веб–браузер і перейдіть за адресою http://localhost:8080/hello. Якщо ви все зробили правильно, ви повинні побачити, що ваш застосунок виведе повідомлення Hello World!(рис. 1.9).

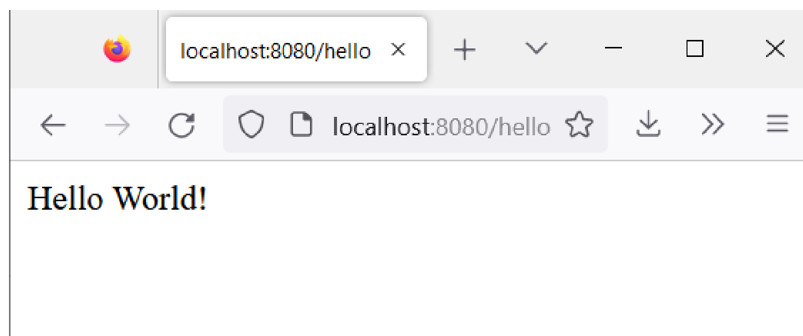

Рисунок 1.9 – Приклад виводу програми в браузері

Це стандартна відповідь за замовчуванням. Ви можете вказати параметр у своєму веб–запиті, щоб програма знала, як правильно вітати вас. Наприклад, спробуйте http://localhost:8080/hello?myName=Human (рис. 1.10).

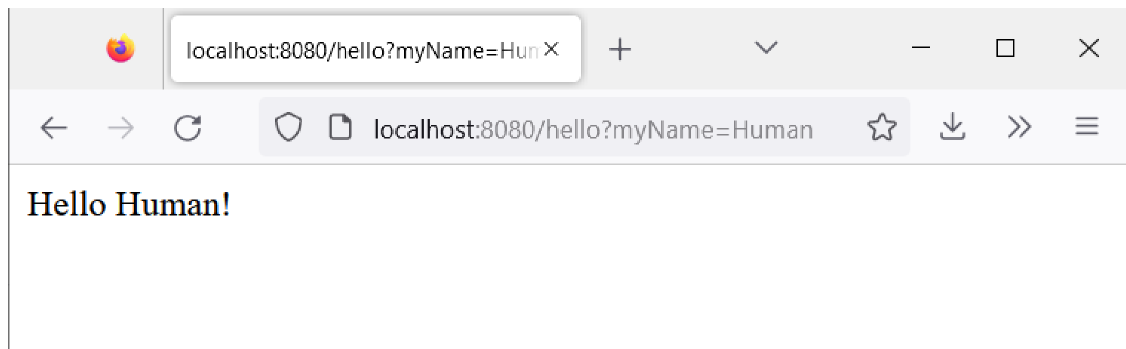

Рисунок 1.10 – Приклад використання параметра, переданого методом GET

1.2.4 Додавання домашньої сторінки

Створена програма Spring Boot має одну точку доступу, доступну за адресою /hello. Однак, якщо ви відкриєте кореневий контекст своєї програми за адресою http://localhost:8080/, ви отримаєте повідомлення про помилку, оскільки кореневий ресурс не визначено. Давайте додамо статичну домашню сторінку HTML з посиланнями на кінцеву точку.

Створіть файл index.html у папці /src/main/resources/static/.

У вікні інструмента «Проєкт» клацніть правою кнопкою миші каталог /src/main/resources/static/ та оберіть «Створити | HTML–файл, вкажіть ім’я index.html та натисніть Enter.

Змініть шаблон за замовчуванням або замініть його наступним кодом HTML як показано на лістингу 1.2:

Лістинг 1.2 – HTML код домашньої сторінки в файлі infex.html

```
<!DOCTYPEHTML>
<html>
<head>
<title>Spring Boot application</title>
```

Кінець лістингу 1.2

```
<meta http-equiv="Content-Type"content="text/html;
charset=UTF-8"
/>
</head>
<body>
<p><a href="/hello">Greet the world!</a></p>

<formaction="/hello"метод= "GET"id="nameForm">
<div>
<label for="nameField">How should the app call you?</label>
<input name="myName"id="nameField">
<button>Greet me!</button>
</div>
</form>
</body>
</html>
```

У вікні інструмента «Виконати» натисніть кнопку «Повторити запуск» або натисніть Shift+F10, щоб перезапустити програму Spring.

Тепер ваш застосунок буде використовувати index.html як кореневий ресурс за адресою http://localhost:8080/ (див. рис. 1.11).

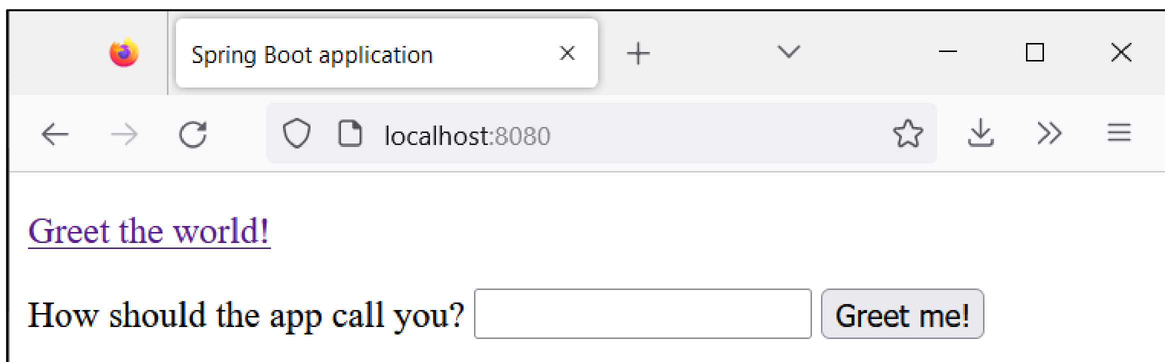

Рисунок 1.11 – Результат виводу домашньої сторінки в браузері

1.3 Завдання до лабораторної роботи

1. Створіть endpoint /list, який виведе список всіх користувачів, які повідомляли серверу своє ім'я за час його роботи.
2. Відформатуйте висновок у вигляді таблиці HTML із заголовком.
3. Створіть endpoint /find який прийматиме єдиний рядковий параметр і виведе список всіх користувачів, які повідомляли серверу своє ім'я, якщо їхнє ім'я починалося з введеного рядка.
4. Створіть HTML сторінку з формою запиту та HTML сторінку з виведенням результатів

1.4 Зміст звіту

1. Формулювання мети й задачі лабораторної роботи.
2. Власний програмний код, розроблений під час виконання лабораторної роботи із коментарями.
3. Короткі відповіді на контрольні запитання.
4. Висновки за результатами роботи.

1.5 Контрольні питання

1. Що таке Spring Boot і які його основні переваги у порівнянні з класичним використанням Spring Framework?
2. Яку роль виконує анотація @SpringBootApplication і які функції вона об'єднує?
3. Як у Spring Boot реалізується відображення HTML–сторінок за допомогою шаблонізатора (наприклад, Thymeleaf)?
4. Як можна передати параметри з контролера до HTML–сторінки у Spring Boot?

5. Для чого використовується клас Model або ModelMap у методах контролера?

6. Як налаштовується порт та інші параметри застосунку у файлі application.properties або application.yml?

7. Що відбувається після запуску Spring Boot застосунку і яким чином можна отримати доступ до створеної HTML–сторінки у браузері?

8. Як можна організувати просту форму на HTML–сторінці і передати дані з неї до контролера у Spring Boot?

2 ЛАБОРАТОРНА РОБОТА № 2 – РОЗРОБКА SOAP ВЕБСЕРВІСУ

Мета роботи: одержати знання та навички розробки серверної та клієнтської частини SOAP вебсервісу із використання фреймворку Spring Boot [4–7].

2.1 Теоретичні відомості

SOAP (англ. Simple Object Access Protocol) – це протокол обміну структурованими повідомленнями між комп'ютерними системами через мережу, який широко використовується для створення веб–служб. На відміну від REST, який орієнтований на ресурси та легкі формати обміну, SOAP базується на стандартизованому XML, що дозволяє строго описувати структуру повідомлень, їх типи та правила взаємодії між клієнтом і сервером. Основна ідея SOAP полягає в тому, щоб забезпечити незалежний від платформи і мови програмування спосіб передачі даних між віддаленими компонентами, гарантуючи при цьому сумісність і надійність.

Повідомлення SOAP складається з трьох основних елементів: Envelope (конверт), який визначає початок і кінець повідомлення; Header (заголовок), де можуть передаватися додаткові метадані, наприклад, інформація про безпеку або маршрутизацію; та Body (тіло), яке містить фактичні дані запиту або відповіді. Крім того, SOAP передбачає спеціальний елемент Fault для обробки помилок, що дозволяє клієнту отримувати детальну інформацію про проблеми, які виникли під час обробки запиту.

Однією з головних переваг SOAP є його стандартизованість і підтримка розширених функцій, таких як безпечна передача даних (WS–Security), транзакції та надійна доставка повідомлень (WS–ReliableMessaging). Це робить його особливо корисним для корпоративних систем і інтеграцій між різнорідними платформами, де критично важлива строгість структури повідомлень і надійність комунікації. Завдяки цим властивостям SOAP

продовжує залишатися популярним у сценаріях, де потрібно забезпечити формальну специфікацію взаємодії між сервісами та гарантовану обробку помилок.

2.1.1 Структура SOAP повідомлень

Повідомлення SOAP має строго визначену структуру, яка будується на основі XML і складається з декількох ключових елементів: Envelope, Header, Body та опційно Fault. Кожен із цих елементів виконує конкретну функцію і забезпечує стандартизовану передачу даних між клієнтом і сервером.

Envelope (конверт) – це кореневий елемент SOAP–повідомлення, який обов’язково присутній у кожному запиті або відповіді. Він визначає, що дане повідомлення є SOAP–повідомленням і може містити неймспейси, необхідні для розпізнавання структурованих даних. Envelope містить усі інші елементи повідомлення.

Header (заголовок) – необов’язковий елемент, призначений для передачі метаданих або службової інформації. У ньому можна передавати, наприклад, дані про автентифікацію, токени безпеки, інформацію про маршрутизацію або спеціальні параметри обробки повідомлення. Header дозволяє серверу або проміжним вузлам обробляти повідомлення до того, як буде прочитане його тіло.

Body (тіло) – обов’язковий елемент, який містить фактичні дані запиту або відповіді веб–служби. Саме тут розміщується виклик операції сервісу та передані параметри, а також результати обробки на сервері. Body може містити складні вкладені структури, визначені у WSDL або XSD.

Fault (помилка) – опційний елемент, який використовується для повідомлення про помилки під час обробки запиту. Він містить код помилки (faultcode), пояснення (faultstring) і додаткову інформацію (detail). Використання Fault дозволяє стандартизовано передавати інформацію про проблеми клієнту, що значно полегшує відладку і обробку помилок.

Лістинг 2.1 – Приклад SOAP–запиту

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:gs="http://example.com/countries">
    <soapenv:Header/>
    <soapenv:Body>
        <gs:getCountryRequest>
            <gs:name>Spain</gs:name>
        </gs:getCountryRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

У цьому прикладі Envelope визначає повідомлення як SOAP, Header порожній (немає додаткової інформації), а Body містить виклик операції getCountryRequest із параметром name.

Лістинг 2.2 – Приклад SOAP–відповіді

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:gs="http://example.com/countries">
    <soapenv:Header/>
    <soapenv:Body>
        <gs:getCountryResponse>
            <gs:country>
                <gs:name>Spain</gs:name>
                <gs:population>46704314</gs:population>
                <gs:capital>Madrid</gs:capital>
            </gs:country>
        </gs:getCountryResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Тут у Body розміщено результат обробки запиту – інформацію про країну Spain.

Лістинг 2.3 – Приклад SOAP Fault

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header/>
    <soapenv:Body>
        <soapenv:Fault>
            <faultcode>SOAP-ENV:Client</faultcode>
            <faultstring>Country not found</faultstring>
            <detail>
                <errorCode>404</errorCode>
            </detail>
        </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

У цьому прикладі сервер повертає помилку, повідомляючи клієнту, що країна не знайдена.

2.1.2 Використання WSDL для опису сервісів.

Для роботи з SOAP веб–службами ключовим елементом є WSDL (Web Services Description Language) – мова опису веб–сервісів, яка дозволяє формально визначити, як клієнт і сервер повинні взаємодіяти. WSDL описує структуру сервісу, доступні операції, типи даних, що передаються, а також протоколи та адреси доступу, і це робить його основою для автоматизованого обміну повідомленнями між системами, незалежно від мови програмування чи платформи.

Документ WSDL має чітку ієрархічну структуру, яка включає кілька основних блоків. Types описує всі складні типи даних, які можуть використовуватися у запитах і відповідях (часто через XSD). Message визначає, які дані надсилаються та отримуються для конкретної операції. PortType описує самі операції веб–сервісу і зв’язує їх із повідомленнями. Binding встановлює, який протокол використовується (наприклад, SOAP 1.1 або 1.2) та як повідомлення серіалізуються у XML. І, нарешті, Service вказує фактичну адресу сервера (endpoint), за якою клієнт може відправляти запити.

Використання WSDL дозволяє автоматично генерувати клієнтські та серверні класи за допомогою інструментів на кшталт wsimport або плагінів Maven (jaxws–maven–plugin). Це означає, що розробник отримує готові Java–класи для запитів і відповідей, не займаючись ручним написанням XML–структур. Таким чином, WSDL забезпечує сувору типізацію, стандартизовану структуру повідомлень та автоматизацію інтеграції між системами, що значно спрощує розробку та підтримку SOAP веб–служб.

2.1.3 Підтримка SOAP в Spring Boot

Spring Boot значно спрощує створення та розгортання SOAP веб–служб завдяки готовим механізмам і залежностям, що дозволяють мінімізувати ручне налаштування. Фреймворк надає підтримку SOAP через модуль Spring Web Services (Spring–WS), який інтегрується зі Spring Boot «із коробки» і дозволяє швидко створювати серверні кінцеві точки (endpoints) для обробки SOAP–запитів.

Для реалізації SOAP сервісу в Spring Boot достатньо підключити стартер spring–boot–starter–web–services, що автоматично додає необхідні бібліотеки для роботи з XML, маршалінгом/анмаршалінгом через JAXB, а також підтримку транспортного рівня через HTTP. Сервер можна запускати як вбудований веб–сервер (Tomcat, Jetty або Undertow), що дозволяє не налаштовувати окремий контейнер, а мати самодостатній JAR–застосунок із SOAP сервісом всередині.

В Spring Boot для реалізації SOAP кінцевих точок використовуються анотації: @Endpoint, яка позначає клас як обробник запитів; @PayloadRoot**,** яка зв'язує метод із конкретним XML–елементом запиту; @RequestPayload та @ResponsePayload, що забезпечують автоматичний маршалінг об'єктів Java у XML і навпаки. Крім того, Spring Boot інтегрується з JAXB для конвертації повідомлень у Java–класи, що дозволяє працювати з об'єктами замість ручного формування XML.

2.1.4 Конфігурація SOAP сервера у Spring Boot

Конфігурація SOAP сервера у Spring Boot здійснюється досить просто завдяки готовим механізмам Spring Web Services. Головна мета конфігурації – забезпечити коректну обробку запитів, маршалінг/анмаршалінг XML–повідомлень у Java–об'єкти та визначити, за яким URL клієнти можуть звертатися до сервісу.

Перш за все, у проєкті підключається залежність spring–boot–starter–web–services, яка додає усі необхідні бібліотеки для роботи з SOAP, включно з підтримкою JAXB. Далі створюється конфігураційний клас, який замість застарілого WsConfigurerAdapter використовує сучасні підходи Spring Boot: визначає @Bean для ServletRegistrationBean, який реєструє MessageDispatcherServlet і встановлює базовий URL для сервісу, наприклад /ws/*. Цей сервлет відповідає за прийом SOAP–запитів, їх маршрутизацію до відповідних endpoint–ів та обробку повідомлень.

Наступний крок – визначення XSD–схем (XML Schema Definition), які описують структуру повідомлень запиту та відповіді. На основі цих схем Spring Boot генерує WSDL, доступний для клієнтів, і забезпечує строгий контроль за типами даних. У конфігурації реєструється DefaultWsdl11Definition, де вказується ім'я сервісу, namespace і посилання на XSD–схему.

Для обробки конкретних запитів створюються endpoint и – класи, позначені анотацією @Endpoint. Методи цих класів позначаються @PayloadRoot, яка

вказує на кореневий XML–елемент запиту, та @ResponsePayload, яка забезпечує маршалінг об'єкта Java у XML–відповідь. Параметр @RequestPayload дозволяє автоматично отримати Java–об'єкт, який відповідає даним запиту.

2.1.5 Розміщення WSDL та XSD файлів

У Spring Boot важливим аспектом побудови SOAP сервісу є розміщення WSDL та XSD файлів, адже вони визначають структуру повідомлень і контракт сервісу. Зазвичай ці файли розташовуються у ресурсах проєкту, щоб їх можна було легко підключити до конфігурації сервера та зробити доступними для клієнтів.

XSD–файли описують схеми XML, які визначають структуру запитів і відповідей сервісу: типи елементів, обов'язкові та опціональні поля, вкладені структури тощо. У Spring Boot їх зазвичай розміщують у папці src/main/resources або у підпапці, наприклад xsd/. На основі цих схем Spring WS автоматично виконує маршалінг/анмаршалінг XML у Java–об'єкти за допомогою JAXB.

WSDL–файл у Spring Boot може бути згенерований автоматично на основі XSD та конфігурації сервісу, що задається через DefaultWsdl11Definition. Це дозволяє клієнтам отримувати актуальний опис сервісу і знати, які операції доступні, які параметри потрібно передавати і яку відповідь очікувати. Для генерації WSDL важливо вказати namespace, ім'я сервісу та шлях до XSD–схеми, після чого Spring Boot робить файл доступним за URL, наприклад http://localhost:8080/ws/countries.wsdl.

Розміщення файлів у ресурсах проєкту забезпечує їх інкапсуляцію всередині JAR–файлу при розгортанні, що спрощує перенесення та розгортання сервісу на різних середовищах, а також гарантує, що клієнти завжди працюватимуть із актуальною схемою і WSDL.

2.1.6 Налаштування Endpoint для обробки запитів.

У Spring Boot налаштування Endpoint для обробки SOAP–запитів є ключовим етапом реалізації сервісу. Endpoint – це клас, який відповідає за прийом запитів від клієнта, обробку даних і формування відповіді у форматі SOAP. У Spring Boot він реалізується досить просто завдяки інтеграції з Spring Web Services.

Клас, який виконує роль Endpoint, позначається анотацією @Endpoint. Це дає зрозуміти Spring Boot, що цей клас слід реєструвати як обробник SOAP–повідомлень. Методи класу позначаються анотацією @PayloadRoot, де вказується простір імен (namespace) та локальна назва кореневого XML–елемента запиту. Цей зв'язок дозволяє Spring точно визначати, який метод обробляє конкретний SOAP–запит.

Для отримання даних запиту в методі використовується анотація @RequestPayload, яка автоматично перетворює XML–повідомлення на Java–об'єкт. Відповідь сервісу повертається через @ResponsePayload, що забезпечує маршалінг Java–об'єкта назад у XML. Такий підхід дозволяє працювати з об'єктами замість ручного розбору XML, значно спрощуючи код і підвищуючи надійність обробки даних.

Наприклад, для сервісу отримання інформації про країну метод може виглядати так:

Лістинг 2.4 – Код класу ендпоінту CountryEndpoint

```
@Endpoint
public class CountryEndpoint {

    private static final String NAMESPACE_URI =
"http://example.com/countries";

    private final CountryRepository countryRepository;
```

Кінець лістингу 2.4

```
    public CountryEndpoint(CountryRepository
countryRepository) {
        this.countryRepository = countryRepository;
    }

    @PayloadRoot(namespace = NAMESPACE_URI, localPart =
"getCountryRequest")
    @ResponsePayload
    public GetCountryResponse getCountry(@RequestPayload
GetCountryRequest request) {
        GetCountryResponse response = new
GetCountryResponse();

response.setCountry(countryRepository.findByName(request.getN
ame()));
        return response;
    }
}
```

У цьому прикладі Spring Boot автоматично підбирає метод для обробки запиту з локальним елементом `getCountryRequest` у зазначеному namespace, маршалить вхідні дані у `GetCountryRequest`, а відповідь перетворює у SOAP XML для відправки клієнту.

Таким чином, налаштування Endpoint у Spring Boot дозволяє створювати чітко структуровані, типізовані та безпечні обробники SOAP–запитів, з мінімальною ручною роботою над XML і зручною інтеграцією з бізнес–логікою.

2.1.7 Маршалінг та анмаршалінг XML

У контексті SOAP сервісів у Spring Boot важливо розуміти поняття маршалінгу та анмаршалінгу XML, оскільки це основний механізм перетворення даних між XML–повідомленнями та об’єктами Java. SOAP–повідомлення завжди передаються у форматі XML, а для зручної роботи всередині сервісу дані перетворюються у відповідні класи, що дозволяє працювати з ними як зі звичайними об’єктами без ручного парсингу.

Маршалінг (marshalling) – це процес перетворення Java–об’єкта у XML–повідомлення. У Spring Boot він використовується для формування відповіді сервісу, коли результат обробки бізнес–логіки потрібно відправити клієнту у форматі SOAP. Наприклад, об’єкт GetCountryResponse, що містить інформацію про країну, автоматично конвертується у відповідний XML–елемент у тілі SOAP.

Анмаршалінг (unmarshalling) – зворотній процес, коли вхідне SOAP–повідомлення XML перетворюється на Java–об’єкт. Це дозволяє endpoint–у отримати дані запиту у зручному для обробки вигляді. У Spring Boot для цього використовується анотація @RequestPayload, яка автоматично виконує анмаршалінг, перетворюючи XML на відповідний клас, наприклад GetCountryRequest.

У Spring Boot ці процеси реалізуються за допомогою Marshaller/Unmarshaller, найчастіше через JAXB2Marshaller. Конфігураційно задається шлях до пакету з генерованими класами на основі XSD–схеми (contextPath), що дозволяє mapshaller–у знати, які класи використовувати для перетворення.

Лістинг 2.5 – Приклад конфігурації JAXB2Marshaller

```
    @Bean
public Jaxb2Marshaller marshaller() {
    Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
```

Кінець лістингу 2.5

```
    marshaller.setContextPath("com.example.countries");
    return marshaller;
}
```

Лістинг 2.6 – Використання у клієнті чи ендпоінті

```
@Autowired
private Jaxb2Marshaller marshaller;

@ResponsePayload
public GetCountryResponse getCountry(@RequestPayload
GetCountryRequest request) {
    // Spring автоматично анмаршалить GetCountryRequest та
маршалить GetCountryResponse
}
```

Завдяки цьому механізму розробник може повністю працювати з Java–об’єктами, не займаючись ручним формуванням або розбором XML, що значно підвищує зручність, надійність та безпеку обробки SOAP–повідомлень.

2.1.8 Створення SOAP клієнту із Spring Boot

Створення SOAP клієнта у Spring Boot дозволяє споживати веб–служби, описані через WSDL, і автоматично обмінюватися даними у форматі XML без ручного формування SOAP–повідомлень. Spring Boot у поєднанні з Spring Web Services забезпечує готові механізми для налаштування клієнта, маршалінгу/анмаршалінгу повідомлень та відправки запитів на сервер.

Перш за все, у проєкті підключають стартер spring–boot–starter–web–services, який додає всі необхідні бібліотеки для роботи з SOAP, включно з JAXB для перетворення XML у Java–об’єкти та навпаки. Далі створюється клас клієнта, який зазвичай містить методи для виклику операцій сервісу. У Spring

Boot для спрощення використовується WebServiceTemplate, що надає методи marshalSendAndReceive, які виконують повний цикл SOAP–виклику: маршалінг об’єкта запиту у XML, відправку повідомлення на сервер, отримання відповіді та її анмаршалінг у Java–об’єкт.

Конфігураційно для клієнта налаштовують Jaxb2Marshaller, де задають contextPath – пакет з класами, згенерованими на основі WSDL/XSD. Це дозволяє WebServiceTemplate автоматично конвертувати Java–об’єкти у SOAP–запити і назад. Також у конфігурації задається default URI сервісу, за яким клієнт надсилає запити.

Лістинг 2.7 – Приклад класу клієнта

```
@Component
public class CountryClient {

    private final WebServiceTemplate webServiceTemplate;

    public CountryClient(Jaxb2Marshaller marshaller) {
        this.webServiceTemplate = new
WebServiceTemplate(marshaller);

this.webServiceTemplate.setDefaultUri("http://localhost:8080/
ws");
    }
    public GetCountryResponse getCountryInfoByName(String
name) {
        GetCountryRequest request = new GetCountryRequest();
        request.setName(name);
        return (GetCountryResponse)
webServiceTemplate.marshalSendAndReceive(request);
    }
}
```

У цьому прикладі метод `getCountryInfoByName` створює запит `GetCountryRequest`, який `WebServiceTemplate` автоматично перетворює на SOAP XML, відправляє на сервер і отримує `GetCountryResponse`.

Таким чином, Spring Boot дозволяє швидко і зручно створювати SOAP клієнти, зосередившись на бізнес–логіці, а не на ручному формуванні та обробці XML. Це значно спрощує інтеграцію з існуючими SOAP сервісами і робить клієнтський код чистим та типізованим.

2.2 Хід роботи

Згенеруйте шаблон проєкту за допомогою сайту https://start.spring.io/ з наступними налаштуваннями (додайте Spring Web до списку залежностей):

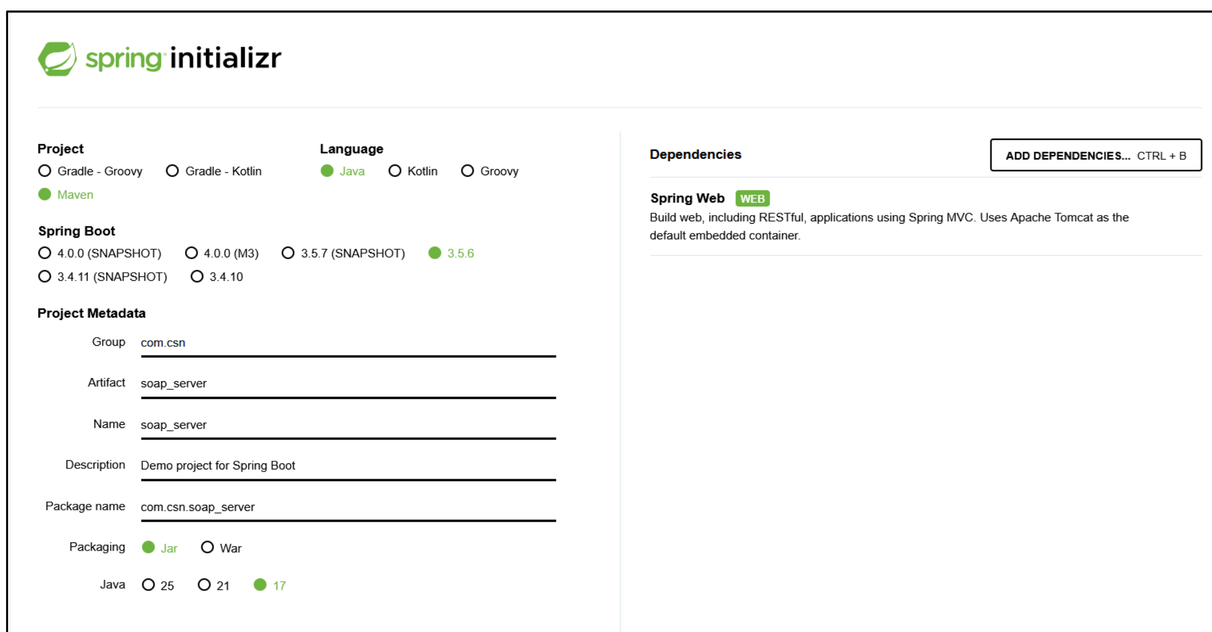

Рисунок 2.1 – Параметри проєкту в сервісі Spring Initializr

Створіть новий проєкт у IntelliJ IDEA Community Edition, вибравши в меню File → New → Project from Existed Sources та вкажіть шлях до розпакованого архіву з шаблоном проєкту.

При імпорті проєкту вкажіть системи збірки Maven (хоча IDE повинна підставляти саме правильне значення):

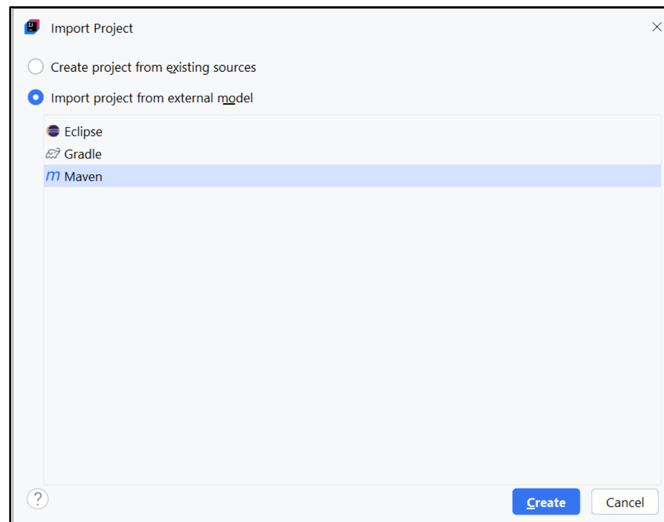

Рисунок 2.2 – Вікно вибору системи зборки

Додайте наступні залежності до файлу pom.xml в корені проєкту:

Лістинг 2.8 – Додаткові залежності в файлі pom.xml

```
<dependency>
    <groupId>org.springframework.ws</groupId>
    <artifactId>spring-ws-core</artifactId>
    <version>5.0.0-M1</version>
</dependency>

<dependency>
    <groupId>wsdl4j</groupId>
    <artifactId>wsdl4j</artifactId>
    <version>1.6.3</version>
</dependency>
```

Використовуйте Google, щоб знайти найновіші версії пакетів на момент проведення лабораторної роботи. Зазвичай першим сайтом у списку має бути той https://mvnrepository.com/, з якого можна відразу скопіювати правильні налаштування для кожного з пакетів:

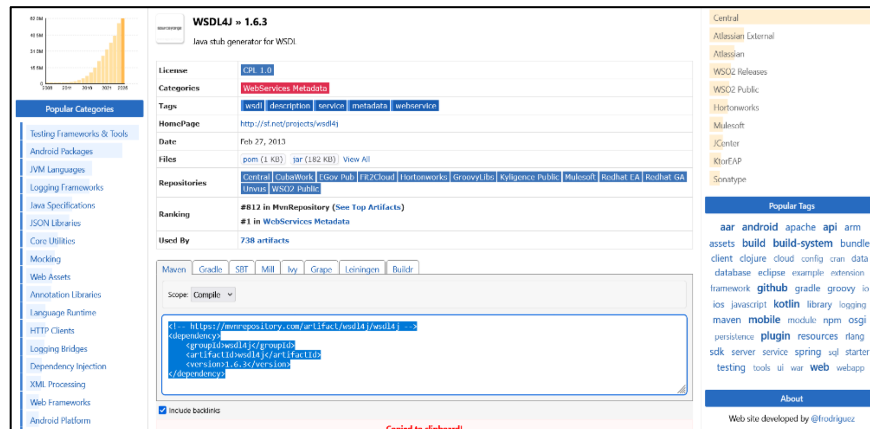

Рисунок 2.3 – Приклад Maven залежності на сайті репозиторію

2.2.1 Створення схеми XML для визначення домену

Домен web сервісу описаний у файлі XML схеми (XSD), який Spring–WS експортуватиме автоматично як WSDL.

Створіть XSD файл з операціями для повернення name, population, capital та currency: src/main/resources/countries.xsd

Лістинг 2.9 – Файл схеми даних countries.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://soap_server.csn.com/countries"

targetNamespace="http://soap_server.csn.com/countries"
elementFormDefault="qualified">
    <xs:element name="getCountryRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

Кінець лістингу 2.9

```
    <xs:element name="getCountryResponse">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="country"
type="tns:country"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="country">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="population" type="xs:int"/>
            <xs:element name="capital" type="xs:string"/>
            <xs:element name="currency" type="tns:currency"/>
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="currency">
        <xs:restriction base="xs:string">
            <xs:enumeration value="GBP"/>
            <xs:enumeration value="EUR"/>
            <xs:enumeration value="PLN"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

У цьому файлі надзвичайно важливо правильно визначити верхній тег xs:schema, вказавши простір імен та атрибути tns. Саме цей простір імен визначає пакет, у якому будуть згенеровані класи зі схеми даних. Обов’язково

перевірте, яке саме значення ви вказали, і за потреби спробуйте кілька варіантів. У нашому прикладі ми використали значення:

http://soap_server.csn.com/countries

Це означає, що згенеровані класи будуть розташовані у пакеті com.csn.soap_server.countries.

2.2.2 Генерація доменних класів на основі XML–схеми

Наступний крок – створення Java–класів із XSD–файлу. Найкращий підхід полягає в автоматичній генерації під час складання проєкту за допомогою плагіна Maven або Gradle.

У нашому випадку ми використаємо плагін для Maven. Для цього потрібно додати залежність у файл pom.xml до плагіна, який відповідає за генерацію Java–класів зі схем даних (`.xsd`).

Додайте залежність у файл pom.xml до плагіна для генерації Java класів з .xsd зі схемою даних. Знайдіть розділ <build> <plugins> у pom.xml файлі... </plugins> </build> та додайте до нього:

Лістинг 2.10 – Додаткова залежність для плагіна jaxb2–maven–plugin

```
<plugin>
 <groupId>org.codehaus.mojo</groupId>
 <artifactId>jaxb2-maven-plugin</artifactId>
 <version>3.1.0</version>
 <executions>
     <execution>
         <id>xjc</id>
         <goals>
             <goal>xjc</goal>
         </goals>
     </execution>
```

Кінець лістингу 2.10

```
    </executions>
    <configuration>
        <sources>
            <source>src/main/resources/countries.xsd</source>
        </sources>
        <outputDirectory>src/main/java</outputDirectory>
        <clearOutputDir>false</clearOutputDir>
    </configuration>
</plugin>
```

Зверніть увагу: у полі `<source>` необхідно вказати шлях до файлу зі схемою даних, а в полі `<outputDirectory>` – директорію, куди буде збережено згенеровані класи.

Після внесення змін потрібно оновити конфігурацію Maven. Це можна зробити, натиснувши відповідну піктограму у верхньому правому куті IDE, або ж через панель Maven ліворуч, обравши першу кнопку Sync/Reload Maven Project.

Для генерації класів зі схеми запустіть фазу Compile у розділі Lifecycle панелі Maven. Альтернативний варіант – виконати команду: mvn compile в консолі.

В результаті будуть згенеровані наступні файли:

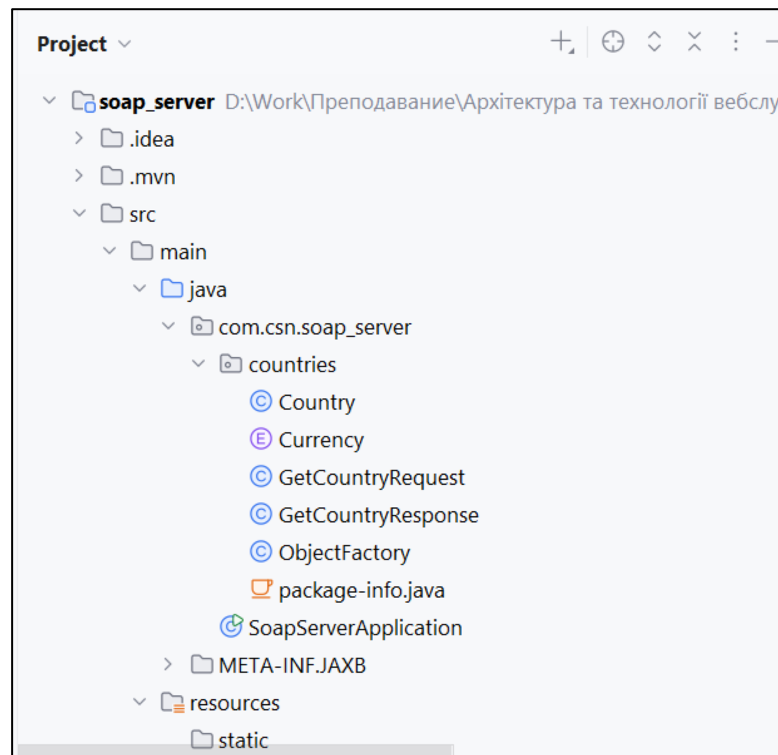

Рисунок 2.4 – Структура проєкта, що було створено

Відкрийте, подивіться ці класи. Це POJO об’єкти, що містять тільки властивості та getters/setters для цих властивостей.

Також зверніть увагу на те, в якому пакеті розташовані згенеровані класи:

```
package com.csn.soap_server.countries;
```

Це саме той пакет (простір імен), який було вказано в заголовку .xsd файлу схеми даних:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://soap_server.csn.com/countries"
targetNamespace="http://soap_server.csn.com/countries"
elementFormDefault="qualified">
```

Розуміння того, в якому просторі імен розташовані ті чи інші дані та методи доступу до них важливо при побудові та зверненні до SOAP сервісів, кожен запит супроводжується інформацією про те, до якого простору імен він адресований.

2.2.3 Створення репозиторію

Для надання даних web сервісом створіть репозиторій. У цьому прикладі ви створюєте простий репозиторій із встановленими даними.

Лістинг 2.11 – Код класу репозиторію CountryRepository

```
package com.csn.soap_server;

import com.csn.soap_server.countries.Country;
import com.csn.soap_server.countries.Currency;
import jakarta.annotation.PostConstruct;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;

@Component
public class CountryRepository {
    private static final List<Country> countries = new
ArrayList<Country>();

    @PostConstruct
    public void initData() {
        Country spain = new Country();
        spain.setName("Spain");
        spain.setCapital("Madrid");
        spain.setCurrency(Currency.EUR);
        spain.setPopulation(46704314);

        countries.add(spain);

        Country poland = new Country();
```

Кінець лістингу 2.11

```
        poland.setName("Poland");
        poland.setCapital("Warsaw");
        poland.setCurrency(Currency.PLN);
        poland.setPopulation(38186860);

        countries.add(poland);

        Country uk = new Country();
        uk.setName("United Kingdom");
        uk.setCapital("London");
        uk.setCurrency(Currency.GBP);
        uk.setPopulation(63705000);

        countries.add(uk);
    }

    public Country findCountry(String name) {
        Country result = null;

        for (Country country : countries) {
            if (name.equals(country.getName())) {
                result = country;
            }
        }

        return result;
    }
}
```

2.2.4 Створення точки виходу сервісу

Для створення точки виходу сервісу, вам необхідний лише POJO з кількома Spring WS анотаціями для обробки вхідних запитів SOAP.

Лістинг 2.12 – Код класу точки входу CountryEndpoint

```
package com.csn.soap_server;

import com.csn.soap_server.countries.GetCountryRequest;
import com.csn.soap_server.countries.GetCountryResponse;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.ws.server.endpoint.annotation.Endpoint;
import
org.springframework.ws.server.endpoint.annotation.PayloadRoot
;
import
org.springframework.ws.server.endpoint.annotation.RequestPayl
oad;
import
org.springframework.ws.server.endpoint.annotation.ResponsePay
load;

@Endpoint
public class CountryEndpoint {
    public static final String NAMESPACE_URI =
"http://soap_server.csn.com/countries";

    private final CountryRepository countryRepository;

    @Autowired
```

Кінець лістингу 2.12

```
    public CountryEndpoint(CountryRepository
countryRepository) {
        this.countryRepository = countryRepository;
    }

    @PayloadRoot(namespace = NAMESPACE_URI, localPart =
"getCountryRequest")
    @ResponsePayload
    public GetCountryResponse getCountry(@RequestPayload
GetCountryRequest request) {
        GetCountryResponse response = new
GetCountryResponse();

response.setCountry(countryRepository.findCountry(request.get
Name()));

        return response;
    }
}
```

@Endpoint – реєструє клас Spring WS як потенційний кандидат для обробки вхідних SOAP–повідомлень.

@PayloadRoot – використовується Spring WS для вибору методу–обробника на основі namespace і localPart повідомлення.

@RequestPayload – вказує, що вхідне повідомлення буде зіставлено з параметром request методу.

@ResponsePayload – визначає, що значення, яке повертає метод, буде сформоване як корисна частина SOAP–відповіді.

У всіх цих фрагментах коду класи com.csn.soap_server.countries будуть викидати помилку компіляції у вашій IDE доти, доки ви не запустите завдання

генерації доменних класів на основі WSDL із попереднього пункту. Тобто POJO об'єкти повинні бути вже згенерованими в цей момент.

2.2.5 Конфігурування бінів web сервісу

Створіть новий клас, пов'язаний із конфігурацією Spring WS бінів:

Лістинг 2.13 – Код класу конфігурації SOAP серверу

```
package com.csn.soap_server;

import
org.springframework.boot.web.servlet.ServletRegistrationBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.ws.config.annotation.EnableWs;
import org.springframework.ws.config.annotation.WsConfigurer;
import
org.springframework.ws.transport.http.MessageDispatcherServle
t;
import
org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;
import org.springframework.xml.xsd.SimpleXsdSchema;
import org.springframework.xml.xsd.XsdSchema;

@EnableWs
@Configuration
public class SoapServerConfig implements WsConfigurer {
    @Bean
    public ServletRegistrationBean<MessageDispatcherServlet>
messageDispatcherServlet(ApplicationContext
```

Кінець лістингу 2.13

```
applicationContext) {
        MessageDispatcherServlet servlet = new
MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean<>(servlet,
"/ws/*");
    }

    @Bean(name = "countries")
    public DefaultWsdl11Definition
defaultWsdl11Definition(XsdSchema countriesSchema) {
        DefaultWsdl11Definition wsdl11Definition = new
DefaultWsdl11Definition();
        wsdl11Definition.setPortTypeName("CountriesPort");
        wsdl11Definition.setLocationUri("/ws");

wsdl11Definition.setTargetNamespace(CountryEndpoint.NAMESPACE
_URI);
        wsdl11Definition.setSchema(countriesSchema);
        return wsdl11Definition;
    }

    @Bean
    public XsdSchema countriesSchema() {
        return new SimpleXsdSchema(new
ClassPathResource("countries.xsd"));
    }
}
```

Spring WS використовує спеціальний сервлет MessageDispatcherServlet для обробки SOAP–повідомлень. Його ключова роль – інтеграція з ApplicationContext. Без цього механізму Spring WS не зможе автоматично виявити та підключити Spring–біни.

Якщо назвати цей бін `dispatcherServlet`, він замінить стандартний DispatcherServlet, який Spring Boot створює за замовчуванням.

Бін DefaultMethodEndpointAdapter забезпечує підтримку програмної моделі Spring WS, що дозволяє використовувати анотації (зокрема, @Endpoint) для оголошення кінцевих точок сервісу.

За допомогою біна DefaultWsdl11Definition можна автоматично згенерувати WSDL 1.1 на основі наданої XsdSchema.

Важливо правильно вказати імена бінів MessageDispatcherServlet та DefaultWsdl11Definition, адже саме вони визначають URL–адреси доступу до веб–сервісу та WSDL–файлу. У прикладі WSDL буде доступний за адресою: http://<host>:<port>/ws/countries.wsdl

2.2.6 Створення окремої виконуваної програми

Незважаючи на те, що пакет цього сервісу може бути у складі web–програми та WAR файлів, більш простий підхід, продемонстрований нижче, створює окремий самостійний застосунок. Ви упаковуєте все в єдиний JAR–файл, який запускається через стандартний метод main() Java–метод. В цьому випадку ви використовуєте підтримку Spring для вбудованого Tomcat контейнера сервлетів як середовище HTTP виконання замість розгортання на сторонній екземпляр.

Лістинг 2.14 – Код основного виконуваного класу додатку

```
package com.csn.soap_server;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
```

Кінець лістингу 2.14

```
@SpringBootApplication
public class SoapServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(SoapServerApplication.class,
args);
    }
}
```

main() метод передає керування допоміжним класом SpringApplication, де SoapServerApplication.class – аргумент його run() методу. Це повідомляє Spring про читання метаданих анотації з SoapServerApplication і управління нею як компонента в Spring application context.

2.2.7 Запуск програми

Для запуску програми необхідно в контекстному меню вибрати пункт Run SoapServerApplication на закладці файлу SoapServerApplication.java або вибравши цей файл у списку файлів проєкту на панелі зліва або методу main класу SoapServerApplication.java.

У разі успішного запуску ви побачите у консолі напис: Tomcat started on port(s): 8080 (http):

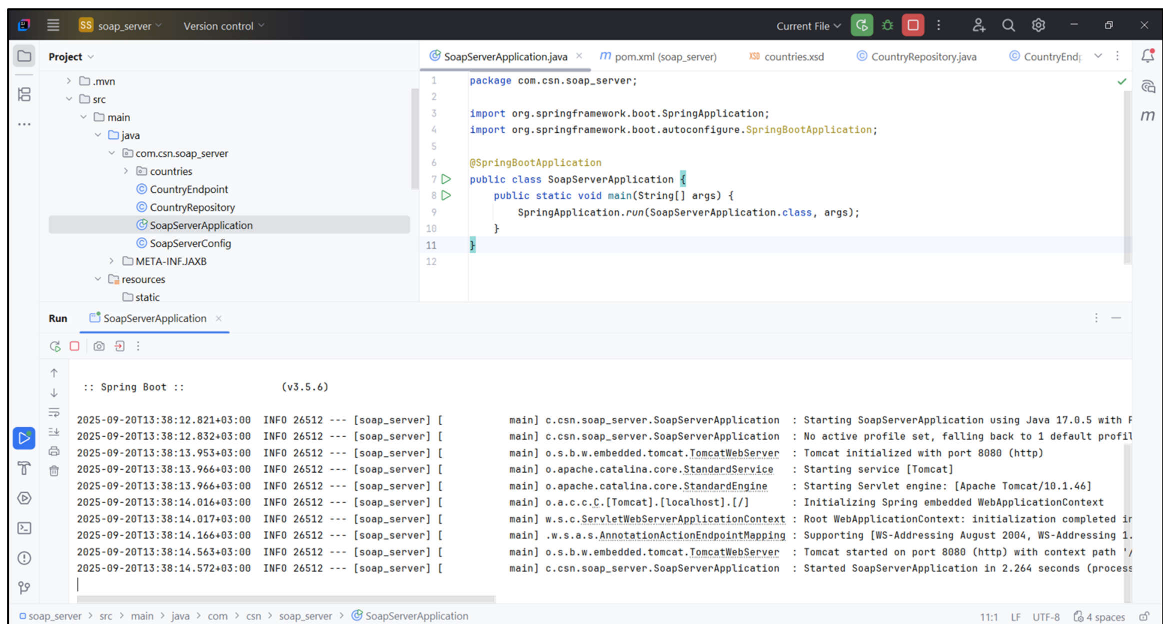

Рисунок 2.5 – Результат успішного запуску застосунку

2.2.8 Тестування програми з використанням утиліти curl у командному рядку

Коли програма запуститься, ви можете протестувати її. Створіть файл request.xml, що містить наступний запит SOAP:

Лістинг 2.15 – Вміст файлу із запитом request.xml

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:gs="http://soap_server.csn.com/countries">
    <soapenv:Header/>
    <soapenv:Body>
        <gs:getCountryRequest>
            <gs:name>Spain</gs:name>
        </gs:getCountryRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

Щоб виконати виклик сервісу, сформуємо запит утилітою curl за допомогою команди:

$ curl —header "content–type: text/xml"–d

@request.xml http://localhost:8080/ws

В результаті ви повинні побачити наступну відповідь:

Лістинг 2.16 – Приклад відповіді SOAP сервера

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getCountryResponse
xmlns:ns2="http://soap_server.csn.com/countries">
            <ns2:country>
                <ns2:name>Spain</ns2:name>
                <ns2:population>46704314</ns2:population>
                <ns2:capital>Madrid</ns2:capital>
                <ns2:currency>EUR</ns2:currency>
            </ns2:country>
        </ns2:getCountryResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Також можна зробити запит (або просто відкрити в браузері) адресу http://localhost:8080/ws/countries.wsdl щоб побачити автоматично згенерований WSDL для вашого сервісу.

2.2.9 Використання плагіна для генерації REST–запитів у браузері

Спершу необхідно завантажити та встановити плагін для виконання REST–запитів у вашому браузері. У цій роботі ми застосуємо RESTClient

(https://addons.mozilla.org/en–US/firefox/addon/restclient/) для FireFox але ви можете знайти інші варіанти. Для цього достатньо ввести в пошуку фрази на кшталт:

– `rest plugin firefox`

– `rest plugin chrome`

Зазвичай функціональність таких розширень майже ідентична, тому ви можете обрати будь–яке з них.

У налаштуваннях плагіна важливо зазначити, що тіло запиту містить дані у форматі XML. Для цього додайте заголовок:

Content–Type: text/xml

У RESTClient плагіні це можна зробити через меню Headers–>Custom Header і вписати ім’я та значення атрибуту:

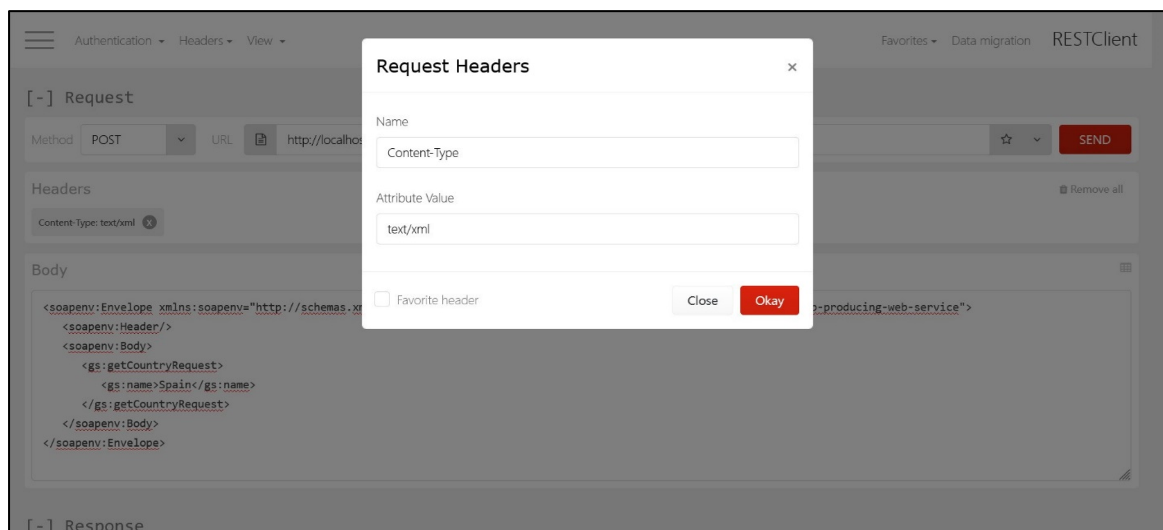

Рисунок 2.6 – Налаштування заголовків запиту в RESTClient

Далі вибрати метод запиту POST та вказати адресу для запиту: http://localhost:8080/ws

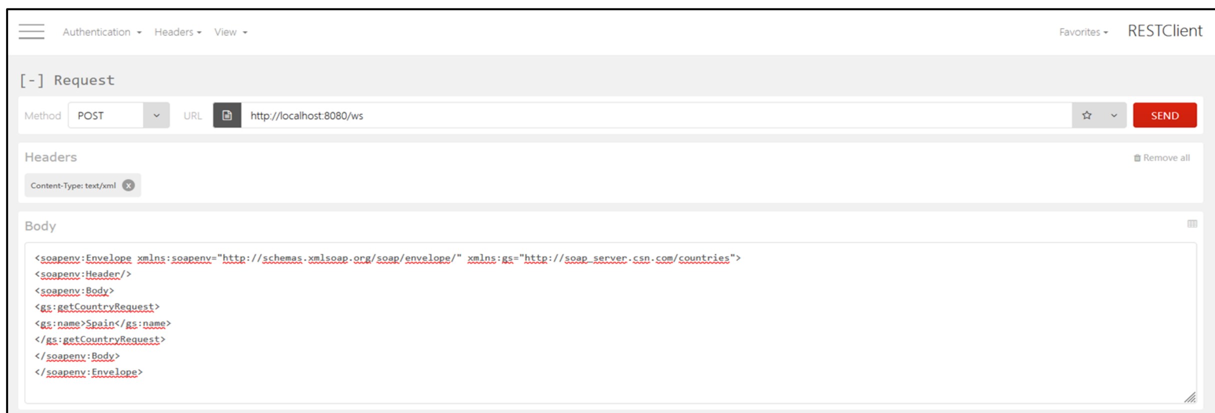

Рисунок 2.7 – Тіло POST запиту до SOAP сервера

Після надсилання запиту, якщо не було помилок на сервері, ви повинні побачити у відповіді інформацію про Іспанію:

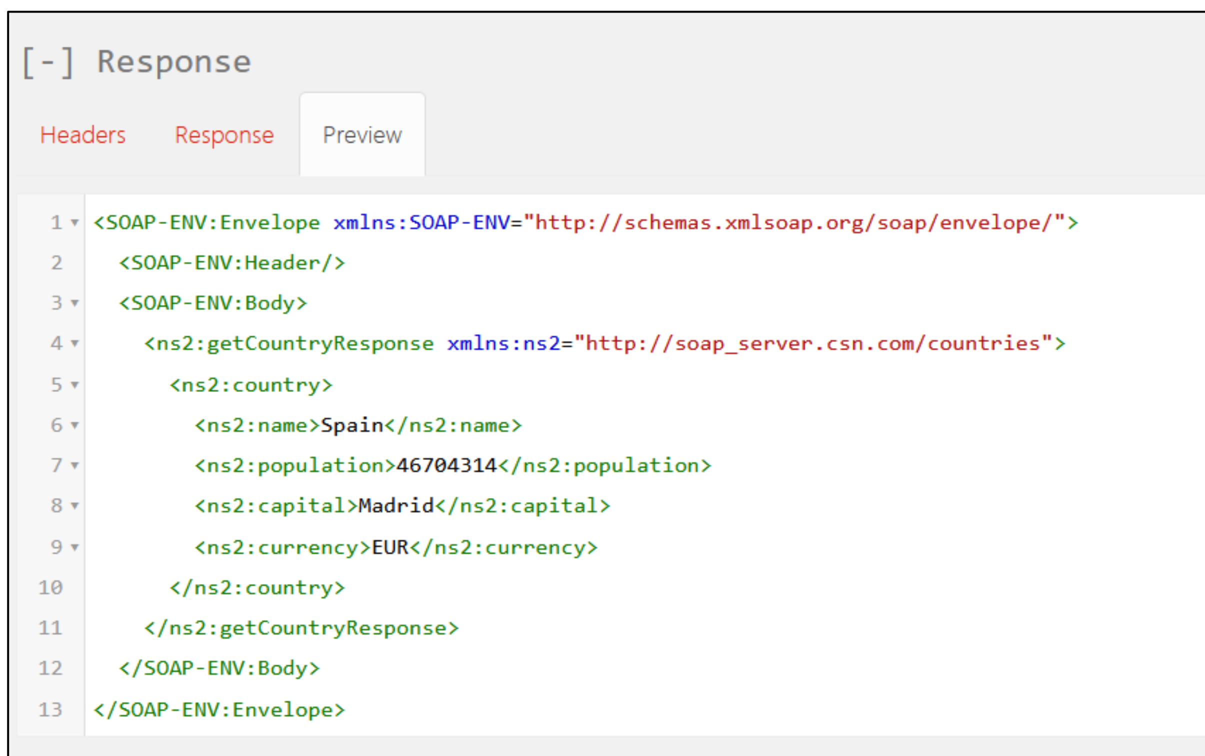

Рисунок 2.8 – Приклад відповіді сервера із даними про Іспанію

Зверніть увагу на розділ Curl унизу. Це те, як насправді плагін робив запит засобами curl, просто всі ці виклики для вас опинилися за кадром. Фактично використання такого плагіна або прямий запит засобами curl у консолі це той самий запит.

[-] Curl

Command

```
curl -X POST -H 'Content-Type: text/xml' -i http://localhost:8080/ws --data '<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:gs="http://soap_server.csn.com/countries">
<soapenv:Header/>
<soapenv:Body>
<gs:getCountryRequest>
<gs:name>Spain</gs:name>
</gs:getCountryRequest>
</soapenv:Body>
</soapenv:Envelope>'
```

Рисунок 2.9 – Вивод консольної команди curl

2.2.10 Створення клієнта SOAP

Згенеруйте новий проєкт за допомогою https://start.spring.io/ з наступними налаштуваннями:

spring initializr

Project: Gradle - Groovy, Gradle - Kotlin, **Maven** (selected)

Language: **Java** (selected), Kotlin, Groovy

Spring Boot: 4.0.0 (SNAPSHOT), 4.0.0 (M3), 3.5.7 (SNAPSHOT), **3.5.6** (selected), 3.4.11 (SNAPSHOT), 3.4.10

Project Metadata

Group	com.csn
Artifact	soap_client
Name	soap_client
Description	Demo project for Spring Boot
Package name	com.csn.soap_client
Packaging	**Jar** (selected), War
Java	25, 21, **17** (selected)

Dependencies — ADD DEPENDENCIES... CTRL + B

Spring Web WEB

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

Рисунок 2.10 – Налаштування проєкту SOAP клієнту

Створіть новий проєкт у IntelliJ IDEA Community Edition, вибравши в меню File → New → Project from Existed Sources та вкажіть шлях до розпакованого архіву з шаблоном проєкту.

Додайте до залежностей наступні пакети, як наведено у лістингу 2.17.

Лістинг 2.17 – Додаткові залежності SOAP клієнту

```
<dependency>
    <groupId>org.springframework.ws</groupId>
    <artifactId>spring-ws-core</artifactId>
    <version>4.1.1</version>
</dependency>

<dependency>
    <groupId>jakarta.xml.ws</groupId>
    <artifactId>jakarta.xml.ws-api</artifactId>
    <version>4.0.2</version>
</dependency>
```

2.2.11 Генерація класів POJO з WSDL

У файлі pom.xml у розділі <build> <plugins> ... </plugins> </build> додайте плагін:

Лістинг 2.18 – Додаткова залежність на плагін jaxws–maven–plugin

```
< plugin>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId> jaxws-maven-plugin</artifactId>
    <version>4.0.2</version>
    <executions>
        <execution>
            <goals>
                <goal>wsimport</goal>
            </goals>
            <phase>generate-sources</phase>
        </execution>
    </executions>
</plugin>
```

Цей плагин використовує wsimport та генерує портативні артефакти JAX–WS для використання у клієнтах і сервісах JAX–WS. Інструмент зчитує файл WSDL та створює всі необхідні артефакти для розробки, розгортання й виклику вебсервісів.

2.2.12 Налаштування WSDL URLs

Сконфігуруємо плагін так, щоб він використовував URL у якості джерела WSDL:

```
<plugin>
    <configuration>
        <wsdlUrls>
<wsdlUrl>http://localhost:8888/ws/country?wsdl</wsdlUrl>
        </wsdlUrls>
    </configuration>
</plugin>
```

Оскільки файл WSDL доступний лише під час роботи SOAP–сервера, перед виконанням запиту необхідно запустити сервер.

Налаштування папки для сгенерованих класів. Далі в полі `packageName` треба вказати назву пакету де будуть розміщуватися згенеровані класи.

```
<plugin>
  <configuration>
        <packageName>com.csn.soap_client</packageName>
        <sourceDestDir>
            src/main/java/com.csn.soap_client/countries
        </sourceDestDir>
    </configuration>
</plugin>
```

Фінальна конфігурація плагіна jaxws–maven–plugin виглядає так:

Лістинг 2.19 – Фінальна конфігурація плагіну jaxws–maven–plugin

```
<plugin>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>jaxws-maven-plugin</artifactId>
    <version>4.0.2</version>
    <executions>
        <execution>
            <goals>
                <goal>wsimport</goal>
            </goals>
            <phase>generate-sources</phase>
        </execution>
    </executions>
    <configuration>
        <wsdlUrls>
<wsdlUrl>http://localhost:8080/ws/countries.wsdl</wsdlUrl>
        </wsdlUrls>
        <packageName>com.csn.soap_server</packageName>
        <sourceDestDir>
            src/main/java/com.csn.soap_server/countries
        </sourceDestDir>
    </configuration>
</plugin>
```

Для запуску генерації POJO класів необхідно справа на панелі Maven вибрати пункт Lifecycle → Install:

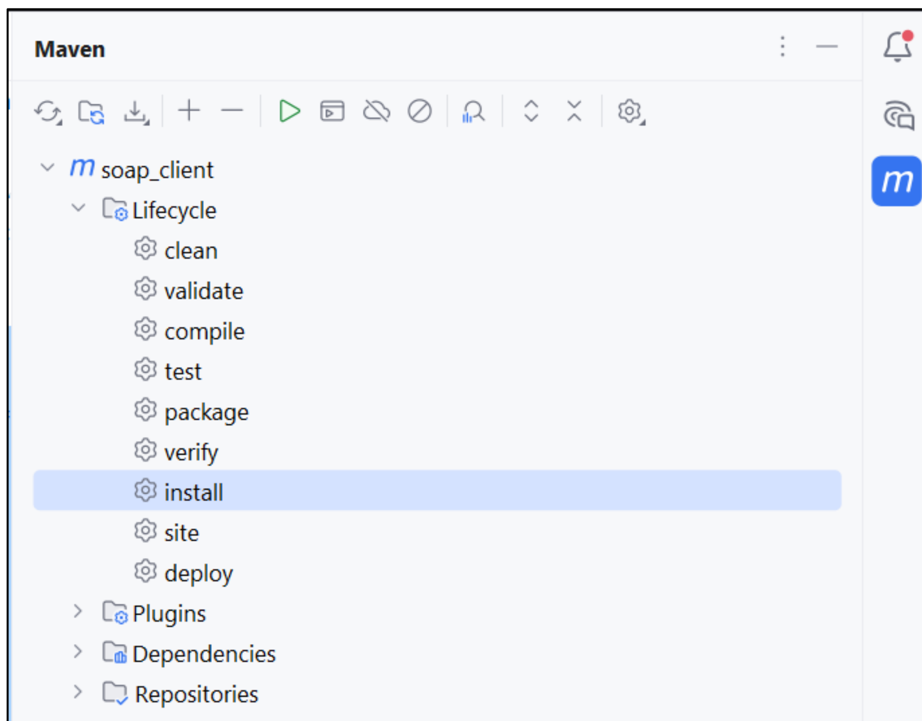

Рисунок 2.11 – Положення команди install на панелі Maven

Або більш специфічний пункт: Plugins → jaxws → jaxws:wsimport:

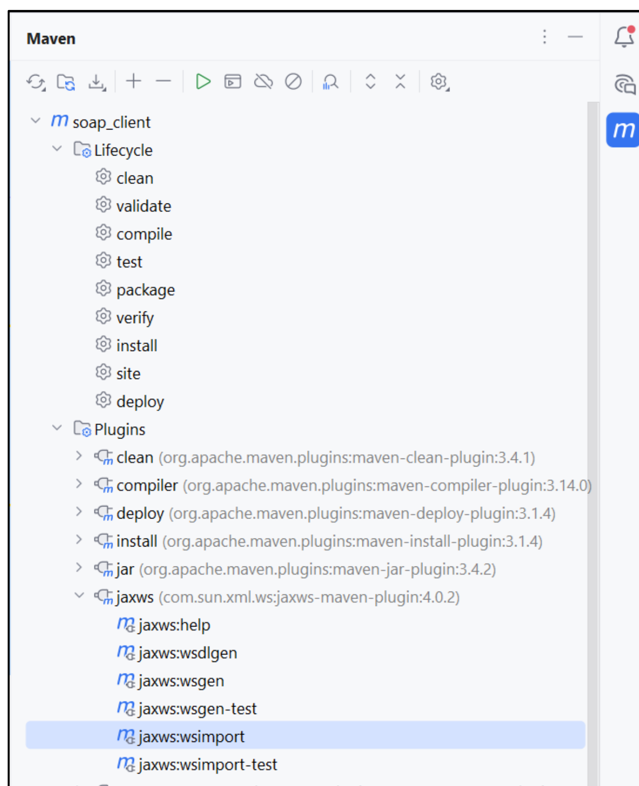

Рисунок 2.12 – Положення команди jaxws:wsimport на панелі Maven

В результаті в директорії target будуть сгенеровані наступні файли:

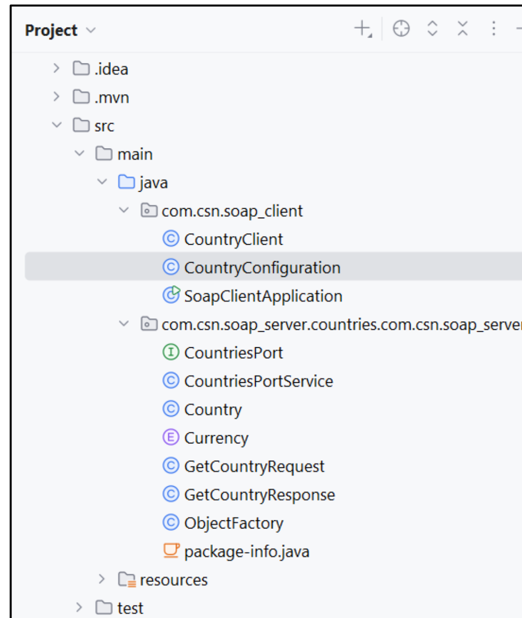

Рисунок 2.13 – Структура сгенерованого проєкту

Ці файли повністю аналогічні тим, що були сгенеровані на сервері із схеми даних countries.xsd.

2.2.13 Створення класу клієнта

Тепер нам необхідно використати функціональність наявного класу WebServiceGatewaySupport для створення власного екземпляру класу SOAP клієнту:

Лістинг 2.20 – Код класу SOAP клієнту CountryClient

```
package com.csn.soap_client;
import
org.springframework.ws.client.core.support.WebServiceGatewayS
upport;
import
org.springframework.ws.soap.client.core.SoapActionCallback;
```

Кінець лістингу 2.20

```
public class CountryClient extends WebServiceGatewaySupport {
    public static final String REQUEST_URL =
"http://localhost:8080/ws";
    public GetCountryResponse getCountryInfoByName(String
countryName) {
        GetCountryRequest request = new GetCountryRequest();
        request.setName(countryName);
        System.out.println();
        System.out.println("Requesting information for " +
countryName);
        return (GetCountryResponse)
getWebServiceTemplate().marshalSendAndReceive(
                request,
                new SoapActionCallback(REQUEST_URL));
    }
    public void printResponse(GetCountryResponse response) {
        Country responseCountry = response.getCountry();
        if (responseCountry != null) {
            System.out.println();
            System.out.println("Country information:");
            System.out.println("Name:" +
responseCountry.getName());
            System.out.println("Population:" +
responseCountry.getPopulation());
            System.out.println("Capital:" +
responseCountry.getCapital());
            System.out.println("Currency:" +
responseCountry.getCurrency());
        } else {
            System.out.println("No information received");
        }}}
```

Тут ми визначили єдиний метод getCountry, що відповідає операції, яку надавав вебсервіс. У цьому методі ми створили екземпляр GetCountryRequest і викликали вебсервіс для отримання GetCountryResponse. Іншими словами, саме тут відбувся обмін повідомленнями SOAP.

Як бачимо, Spring зробив виклик досить простим завдяки своєму WebServiceTemplate. Ми використали метод шаблону marshalSendAndReceive, щоб здійснити SOAP–обмін.

Перетворення XML тут обробляються за допомогою підключеного Marshaller.

Тепер розглянемо конфігурацію, з якої цей Marshaller походить.

2.2.14 Клас CountryClientConfig

Усе, що нам потрібно для налаштування нашого Spring WS клієнта, – це два біни.

По–перше, Jaxb2Marshaller, який перетворюватиме повідомлення з і в XML, а по–друге – наш CountryClient, що буде підключати бін маршаллера.

Лістинг 2.21 – Код класу конфігурації CountryConfiguration

```
package com.csn.soap_client;
import org.springframework.context.annotation.Bean;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;
import static com.csn.soap_client.CountryClient.REQUEST_URL;
public class CountryConfiguration {
    @Bean
    public Jaxb2Marshaller marshaller() {
        Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
        marshaller.setContextPath("com.csn.soap_server");
        return marshaller;
    }
    @Bean
```

Кінець лістингу 2.21

```
    public CountryClient countryClient(Jaxb2Marshaller
marshaller) {
        CountryClient client = new CountryClient();
        client.setDefaultUri(REQUEST_URL);
        client.setMarshaller(marshaller);
        client.setUnmarshaller(marshaller);
        return client;
    }
}
```

Необхідно переконатися, що context path маршаллера збігається з generatePackage, вказаним у конфігурації плагіна в нашому *pom.xml*.

Також зверніть увагу на URI за замовчуванням для клієнта. Він встановлений як значення soap:address location, зазначене у *WSDL*.

2.2.15 Тестування SOAP клієнту

Найпростішим способом протестувати створений SOAP клієнт буде додати прямо в метод main() виклик SOAP серверу, наприклад таким чином:

Лістинг 2.22 – Код основного виконуваного класу застосунку

```
package com.csn.soap_client;

import com.csn.soap_server.GetCountryResponse;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.WebApplicationType;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.context.ApplicationContext;
```

Кінець лістингу 2.22

```
@SpringBootApplication
public class SoapClientApplication {
    public static void main(String[] args) {
        ApplicationContext ctx = new
SpringApplicationBuilder(CountryConfiguration.class)
                .web(WebApplicationType.NONE) // Отключаем
web-сервер
                .run(args);
        CountryClient countryClient =
ctx.getBean(CountryClient.class);
        String countryName = "Spain";
        if (args.length > 0) {
            countryName = args[0];
        }
        GetCountryResponse response =
countryClient.getCountryInfoByName(countryName);
        System.out.println(response);
        countryClient.printResponse(response);
    }

}
```

Якщо все зроблено вірно, при запуску він повинен отримати інформацію із SOAP серверу та надрукувати:

Лістинг 2.23 – Приклад виводу після запуску SOAP клієнту

```
Requesting information for Spain
com.csn.soap_server.GetCountryResponse@5652f555
Country information:
Name:Spain
```

Кінець лістингу 2.23

```
Population:46704314
Capital:Madrid
Currency:EUR
```

2.2.16 Тестування за допомогою Unit–тестів

Відкрийте клас CountryClient, поставте курсор прямо на назву класу та натисніть Ctrl+Shift+T для створення нового JUnit тесту. Оберіть Create New Tests… в меню що відкрилося:

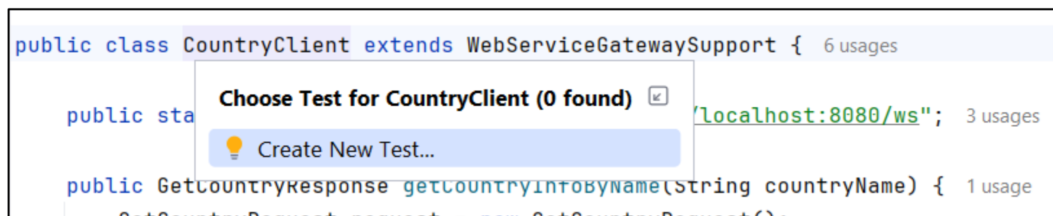

Рисунок 2.14 – Контекстне меню для сворення юніт–тестів

В наступному вікні можна все залишити без змін:

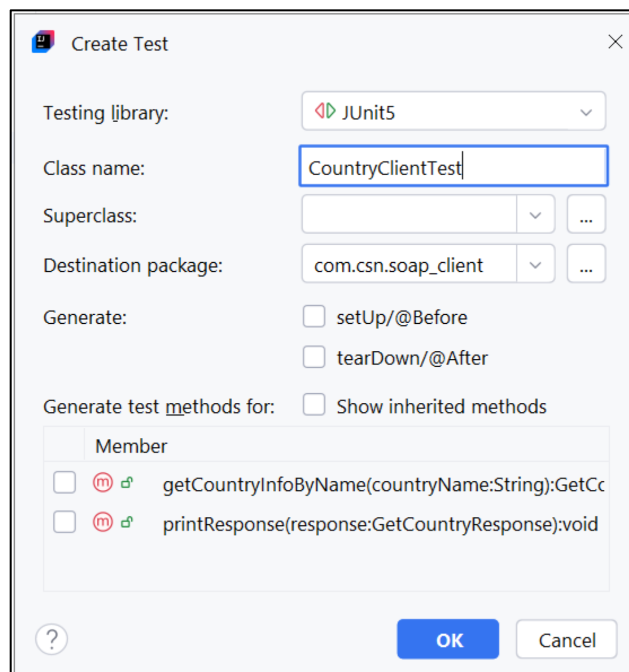

Рисунок 2.15 – Налаштування створюваних юніт–тестів

Буде створено клас CountryClientTest.

Додамо до нього два тести для прикладу:

Лістинг 2.24 – Код класу із юніт–тестами CountryClientTest

```
package com.csn.soap_client;

import com.csn.soap_server.Currency;
import com.csn.soap_server.GetCountryResponse;
import org.junit.jupiter.api.Test;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.ContextConfiguration;
import
org.springframework.test.context.support.AnnotationConfigCont
extLoader;

import static org.junit.jupiter.api.Assertions.*;


@SpringBootTest
@ContextConfiguration(classes = CountryConfiguration.class,
loader = AnnotationConfigContextLoader.class)
public class CountryClientTest {

    @Autowired
    CountryClient client;

    @Test
    public void
givenCountryService_whenCountryPoland_thenCapitalIsWarsaw() {
            GetCountryResponse response =
```

Кінець лістингу 2.24

```
client.getCountryInfoByName("Poland");
        assertEquals("Warsaw",
response.getCountry().getCapital());
    }

    @Test
    public void
givenCountryService_whenCountrySpain_thenCurrencyEUR() {
        GetCountryResponse response =
client.getCountryInfoByName("Spain");
        assertEquals(Currency.EUR,
response.getCountry().getCurrency());
    }
}
```

Запустіть тести, якщо все виконано вірно, вони повинні пройти без помилок:

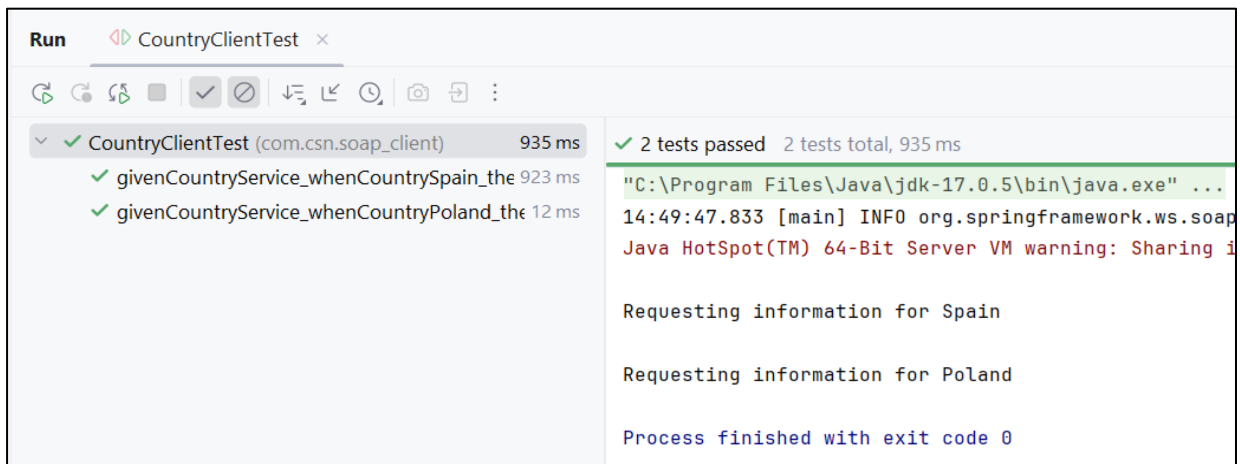

Рисунок 2.16 – Приклад результатів вдалого запуску юніт–тестів

2.3 Завдання до лабораторної роботи

1. Реалізуйте метод addCountryInfo додавання інформації про нову країну та протестуйте його.

2. Реалізуйте метод listCountries, який виведе список усіх країн, які зберігаються в репозиторії CountryRepository, у тому числі додані методом addCountryInfo.

3. Реалізуйте метод usageInfo який поверне інформацію про загальну кількість запитів до сервісу та окремо про кількість звернень до кожного з його реалізованих методів: getCountryRequest, addCountryInfo, listCountries та методу usageInfo.

4. Покрийте нові створені методи JUnit тестами, продемонструйте як позитивні результати тестування, так і випадки тестування заздалегідь помилкових викликів, зробіть перевірки на граничні значення.

2.4 Зміст звіту

1. Власний програмний код, розроблений під час виконання лабораторної роботи із коментарями.

2. Короткі відповіді на контрольні запитання.

3. Висновки за результатами роботи.

2.5 Контрольні питання

1. Що таке SOAP та які основні особливості цього протоколу у порівнянні з REST?

1. Яку роль виконує WSDL–файл у SOAP сервісі і яку інформацію він містить?

2. Як у Spring Boot здійснюється конфігурація SOAP сервера та які основні кроки для його запуску?

3. Для чого потрібні XSD–схеми та як вони використовуються у SOAP сервісі?

4. Що таке Endpoint у Spring Boot SOAP сервері та як його налаштувати для обробки запитів?

5. Як реалізується маршалінг та анмаршалінг XML–повідомлень у Spring Boot?

6. Як створюється SOAP клієнт у Spring Boot та яким чином він взаємодіє із сервером?

7. Які переваги надає використання Spring Web Services у розробці SOAP застосунків?

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mastering Spring Boot 3.0: From Basics to Advanced – Implement Architectural Patterns and Advanced Testing Strategies / Meric, A. // Packt Publishing. 2024 – 256 p.

2. Learning Spring Boot 3.0 – Third Edition: Simplify the Development of Production–Grade Applications Using Java and Spring. / Turnquist, G. L. // Packt Publishing. 2022 – 270 p.

3. Microservices with Spring Boot 3 and Spring Cloud: Build resilient and scalable microservices using Spring Cloud, Istio, and Kubernetes / Magnus Larsson // Packt Publishing, 3rd edition. 2023. – 706 p.

4. SOAP Web Service Tutorials – Herong's Tutorial Examples / Herong Yang // Self–published/independent, 2024 – 374 p.

5. Building Microservices: Designing Fine–Grained Systems. 2nd ed. / Newman S. // Beijing ; Boston : O'Reilly Media, 2021. – 420 p.

6. Spring Microservices in Action / John Carnell, Illary Huaylupo Sánchez // Manning, 2nd edition. 2021 – 448 p.

7. Building Modern Business Applications: Reactive Cloud Architecture for Java, Spring, and PostgreSQL / Peter Royal // Apress, 2022 – 208 p.