**ЗАХІДНОУКРАЇНСКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**Факультет комп'ютерних інформаційних технологій**

**Кафедра економічної кібернетики та інформатики**

Курсова робота

на тему

# Основи роботи із фреймворком Spring

Виконав:

студент групи СА-41

Грицайко П.Р.

Тернопіль

2021

# ЗМІСТ

# Вступ

В сучасному світі саме розробка веб-застосунків є досить актуальною та дуже важливою галуззю інформаційних технологій. Застосунки, котрі працюють у веб-середовищі, забезпечують користувачам доступ до величезної кількості різноманітних послуг, ресурсів та додатків через Інтернет. Задля успішних розробок таких застосунків потрібно мати ефективні інструменти та фреймворки, з якими би покращився процес розробки та підвищилась продуктивність розробників.

Spring на даний момент являється популярним фреймворком для розробки корпоративних додатків на Java. Він надає набір бібліотек та інструментів для створення масштабних, ефективних і досить легких для тестування програмних рішень. Основною метою фреймворка є можливість спростити процес розробки, знижуючи складність інтеграції багатьох різних технологій у рамках однієї програми.

У даній курсовій роботі ми зможемо розглянути основи роботи з фреймворком Spring, зокрема його основні компоненти, механізми ін'єкції залежностей, конфігурацію додатків, а також роботу з базами даних і реалізацію RESTful-сервісів. Також розглянемо основи роботи із Spring Framework, його архітектуру, принципи ін'єкції залежностей та базові практики використання.

# Огляд фреймворку Spring

## 1. Що таке Spring Framework?

Spring Framework , або просто Spring - один із найпопулярніших фреймворків для створення веб-додатків на Java. Фреймворк— це щось схоже на бібліотеку (можливо цей термін вам знайоміший), але є один момент. Грубо кажучи, використовуючи бібліотеку, ви просто створюєте об'єкти класів, які в ній є, викликаєте потрібні вам методи і таким чином отримуєте потрібний вам результат. Тобто, тут більш імперативний підхід: ви чітко вказуєте у своїй програмі, у який конкретний момент треба створити якийсь об'єкт, у який момент викликати конкретний метод, ітд.

З фреймворками справи трохи інакше. Ви просто пишете якісь свої класи, прописуєте там якусь частину логіки, а створює об'єкти ваших класів та викликає методи за вас уже сам фреймворк. Найчастіше ваші класи імплементують якісь інтерфейси з фреймворку або успадковують якісь класи з нього, таким чином отримуючи частину вже написаної за вас функціональності. Але не обов'язково саме так. У Spring наприклад намагаються по максимуму відійти від такої жорсткої зв'язності (коли ваші класи безпосередньо залежать від якихось класів/інтерфейсів з цього фреймворку), і використовують для цієї мети інструкції. Ще хочу відразу відзначити, що Spring можна використовувати не тільки для веб-додатків, але і для так знайомих всім нам звичайних консольних програмок. [1]

## 2. Історія створення

Вперше запущений у 2002 році, Spring та різні фреймворки, побудовані на ньому, такі як Spring Boot, стали домінувати в тому, як розробники Java пишуть код. Розробник Род Джонсон (Rod Johnson) придумав ідею Spring в 2002 році. У наступному році Джонсон, разом з Юргеном Хьоллером і Янном Кароффом, розробив Spring як фрамворк з відкритим вихідним

кодом. Вони створили Spring для того, щоб спростити розробку на стороні сервера Java і дозволити командам розробників швидше створювати свої додатки. [2]

Spring з'явися як відповідь на складність ранніх специфікацій J2EE. Хоча дехто вважає, що Java EE та її сучасний наступник Jakarta EE конкурують із Spring, насправді вони доповнюють один одного. Модель програмування Spring не охоплює специфікацію платформи Jakarta EE; скоріше, він інтегрується з ретельно підібраними індивідуальними специфікаціями з традиційної парасольки EE:

Spring Framework також підтримує специфікації Dependency Injection (JSR 330) і Common Annotations (JSR 250), які розробники додатків можуть використовувати замість специфічних для Spring механізмів, наданих Spring Framework. Спочатку вони були засновані на поширених пакетах javax. Починаючи з Spring Framework 6.0, Spring був оновлений до рівня Jakarta EE 9 (наприклад, Servlet 5.0+, JPA 3.0+), заснований на просторі імен jakarta замість традиційних пакетів javax. З EE 9 як мінімумом і EE 10 вже підтримується, Spring готова надати готову підтримку для подальшої еволюції API Jakarta EE. Spring Framework 6.0 повністю сумісний з Tomcat 10.1, Jetty 11 і Undertow 2.3 в якості веб-серверів, а також з Hibernate ORM 6.1.

З часом роль Java/Jakarta EE у розробці додатків еволюціонувала. На зорі J2EE і Spring були створені додатки для розгортання на сервері додатків. Сьогодні, за допомогою Spring Boot, додатки створюються зручним для devops і cloud способом, з вбудованим контейнером Servlet, який тривіально піддається змінам. Починаючи з Spring Framework 5, додаток WebFlux навіть не використовує API Servlet безпосередньо і може працювати на серверах (таких як Netty), які не є контейнерами Servlet.

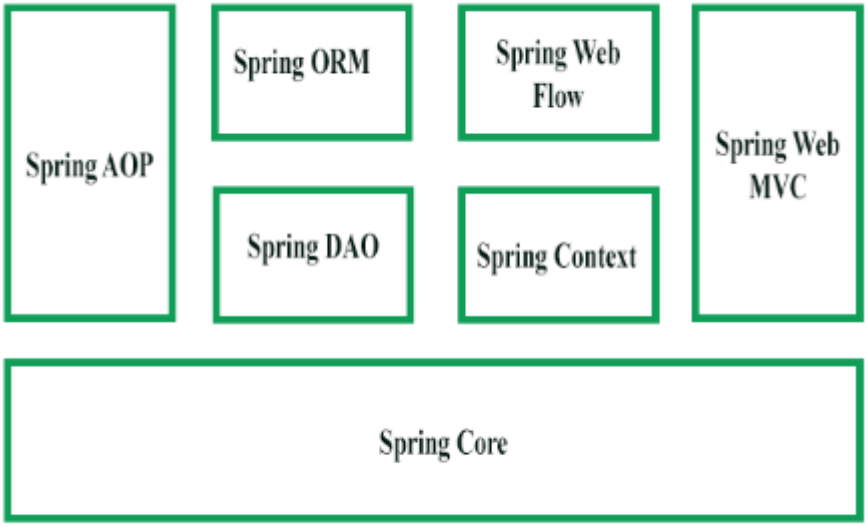
Spring Framework продовжує впроваджувати інновації та розвиватися. Крім Spring Framework, є й інші проекти, такі як Spring Boot, Spring Security, Spring Data, Spring Cloud, Spring Batch та інші. Важливо пам'ятати, що кожен проєкт має власний репозиторій вихідного коду, трекер проблем та періодичність випуску. [3]

## 3. Основні принципи та концепції

Spring базується на кількох ключових концепціях:

- **Інверсія управління (IoC)** — контейнер Spring відповідає за управління життєвим циклом об'єктів, дозволяючи розробникам зосередитися на бізнес-логіці.
- **Аспектно-орієнтоване програмування (AOP)** — підтримка розділення кросс-функціональних задач (наприклад, логування, транзакційне управління) від основної логіки додатка.
- **Модульність** — Spring складається з кількох незалежних, але взаємодіючих компонентів, таких як Spring Core, Spring Data, Spring MVC, Spring Boot, що дозволяє використовувати тільки потрібні частини фреймворку. [3]

## 4.Архітектура Spring




Конструкція Spring складається з семи модулів, які показані на малюнку вище. Такими модулями є Spring Core, Spring AOP, Spring Web MVC, Spring DAO, Spring ORM, Spring context та Spring Web flow. Ці модулі надають різні платформи для створення різних корпоративних додатків; наприклад, ви можете використовувати модуль Spring Web MVC для створення додатків на основі MVC.

- **Модулі Spring Framework**

  **Модуль Spring Core**: модуль Spring Core, який є основним компонентом фреймворку Spring, забезпечує контейнер IoC. Існує два типи реалізації весняних контейнерів, а саме фабрика бобів і контекст застосування. Фабрика бобів визначається за допомогою інтерфейсу org.springframework.beans.factory.BeanFactory і діє як контейнер для бобів. Заводський контейнер bean дозволяє відокремити конфігурацію та специфікацію залежностей від логіки програми. У фреймворку Spring фабрика бобів виступає в ролі центрального контейнера IoC, який відповідає за створення екземплярів об'єктів застосування. Він також налаштовує та збирає залежності між цими об'єктами. Існує

безліч реалізацій інтерфейсу BeanFactory. Клас XmlBeanFactory є найбільш поширеною реалізацією інтерфейсу BeanFactory. Це дозволяє виразити об'єкт для створення додатку та видалити залежності між об'єктами програми.

- **Модуль Spring AOP**: подібно до об'єктно-орієнтованого програмування (ООП), яке розділяє програми на ієрархію об'єктів, АОП ділить програми на аспекти або проблеми. Модуль Spring AOP дозволяє реалізувати проблеми або грані у вашому весняному застосуванні у весняному AOP, грані - це звичайні зерна Spring або звичайні класи з анотацією @Aspect. Ці аспекти допомагають керувати транзакціями, реєструвати та відстежувати збої програм. Наприклад, управління транзакціями потрібне при банківських операціях, таких як переказ суми з одного рахунку на інший. Модуль Spring AOP надає рівень абстракції управління транзакціями, який можна застосувати до API транзакцій.
- **Модуль Spring ORM**: Модуль Spring ORM використовується для доступу до даних з баз даних у програмі. Він надає API для маніпулювання базами даних з JDO, Hibernate та iBatis. Spring ORM підтримує DAO, що забезпечує зручний спосіб створення наступних ORM-рішень на основі DAO:
  - Просте декларативне управління транзакціями
  - Прозора обробка винятків
  - Легкі шаблонні класи, безпечні для ниток
  - Класи підтримки DAO
  - Управління ресурсами
- **Модуль Spring Web MVC**: Модуль Spring Web MVC реалізує архітектуру MVC для створення веб-додатків. Відокремлює код для компонентів представлення моделі та веб-застосунку. У Spring MVC, коли запит генерується з браузера, він спочатку переходить до класу

DispatcherServlet (Front Controller), який відправляє запит контролеру (клас SimpleFormController або клас AbstractWizardformController) за допомогою набору відображень обробників. Контролер витягує та обробляє інформацію, закладену в запит, та надсилає результат класу DispatcherServlet у вигляді об'єкту моделі. Нарешті, клас DispatcherServlet використовує класи ViewResolver для надсилання результатів у представлення, яке відображає ці результати користувачам.

- **Модуль Spring Web Flow**: Модуль Spring Web Flow є розширенням модуля Spring Web MVC. Фреймворк Spring Web MVC надає контролери форм, такі як клас SimpleFormController і клас AbstractWizardFormController, для реалізації попередньо визначеного робочого процесу. Spring Web Flow допоможе вам визначити XML-файл або клас Java, який керує робочим процесом між різними сторінками веб-додатку. Spring Web Flow розповсюджується окремо і може бути скачаний з сайту http://www.springframework.org. Ось переваги Spring Web Flow:
  - Потік між різними користувацькими інтерфейсами програми чітко визначається шляхом визначення веб-потоку в XML-файлі.
  - Визначення мережевих потоків дозволяють практично розділити додатки на різні модулі і повторно використовувати ці модулі в різних ситуаціях.
- **Модуль Spring Web DAO**: пакет DAO в рамках Spring забезпечує підтримку DAO за рахунок використання технологій доступу до даних, таких як JDBC, Hibernate або JDO. Цей модуль вводить рівень абстракції JDBC, усуваючи необхідність забезпечувати виснажливе кодування JDBC. Він також надає програмні, а також декларативні класи управління транзакціями. Пакет Spring DAO підтримує гетерогенне підключення до баз даних Java та відображення O/R, що

допомагає Spring працювати з кількома технологіями доступу до даних. Щоб забезпечити легкий і швидкий доступ до ресурсів бази даних, фреймворк Spring надає абстрактні базові класи DAO. Для кожної технології доступу до даних доступно кілька реалізацій, які підтримує Spring. Наприклад, у JDBC клас JdbcDaoSupport та його методи використовуються для доступу до екземпляру DataSource та попередньо налаштованого екземпляру JdbcTemplate. Вам просто потрібно розширити клас JdbcDaoSupport і надати відображення фактичного екземпляра DataSource у контекстній конфігурації програми, щоб отримати доступ до програми на основі DAO.

- **Контекстний модуль Spring** : Контекстний модуль додатка базується на модулі Core. Контекст програми org.springframework.context.ApplicationContext — це інтерфейс BeanFactory. Цей модуль черпає свою функціональність з пакета org.springframework.beans, а також підтримує такі функції, як інтернаціоналізація (I18N), валідація, поширення подій та завантаження ресурсів. Контекст додатку реалізує інтерфейс MessageSource і надає функціонал для відправки повідомлень додатку. [4]

## Основи роботи з Spring

### 1. Конфігурація Spring

Існує кілька способів конфігурації додатків у Spring:

#### a. XML-базована конфігурація

Це традиційний спосіб конфігурації Beans у Spring. Всі Beans та їхні залежності описуються в XML-файлі.

Приклад:

```xml
<beans>
    <bean id="myBean" class="com.example.MyBean">
        <!-- Конфігурація властивостей -->
    </bean>
</beans>
```

Переваги:

- Чітке відокремлення конфігурації від Java-коду.
- Гнучкість у зміні конфігурації без зміни коду.

Недоліки:

- Велика кількість XML-коду може бути складною для розуміння та підтримки.
- Високий ризик помилок при ручному редагуванні.
- Вважається застарілим способом

**b. Анотаційна конфігурація**

В анотаційній конфігурації, Beans визначаються безпосередньо в Java-коді за допомогою спеціальних анотацій.

Приклад:

```java
@Component
class MyBean {
}
```

Переваги:

- Зручність та інтуїтивність роботи з Java-кодом.
- Легше відстежувати залежності та конфігурації.

Недоліки:

- Потребує рекомпіляції коду при зміні конфігурації.
- Може призвести до “розпилення” конфігурації по багатьом класам.

### c. Java-базована конфігурація

Цей підхід дозволяє повністю уникнути використання XML, замість цього використовуючи Java-класи для конфігурації.

Приклад:


Переваги:

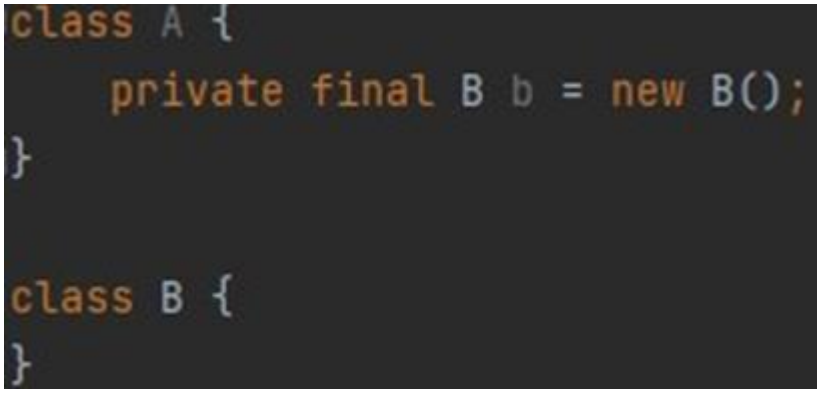
- Повна інтеграція з Java-кодом та його можливостями.
- Зручність у рефакторингу та читабельності.

Недоліки:

- Подібно до анотаційної конфігурації, вимагає рекомпіляції при зміні. [5]

**2. Інверсія управління (IoC)**

Інверсія контролю (IoC) – це фундаментальний принцип в архітектурі Spring, який перевертає традиційне управління об’єктами. В традиційному програмуванні, компоненти програми безпосередньо створюють і управляють залежностями. Наприклад, якщо клас A потребує доступу до класу B, він самостійно створює екземпляр B.




В такому підході клас **A** має високу ступінь залежності від класу **B**, що вважається анти-патерном та ускладнює тестування та зміни.

Інверсія контролю в Spring перекладає відповідальність за створення та управління залежностями на фреймворк. Замість того, щоб клас A створював екземпляр B, Spring контейнер сам створює обидва об’єкти та інжектує (вводить) **B** у **A**.

Роль IoC у Spring: Spring використовує IoC для керування життєвим циклом об’єктів і залежностями між ними. Це забезпечує більшу модульність, гнучкість і легкість у тестуванні. [6]

### 3.Контейнер Spring

Контейнер Spring відповідає за створення, конфігурування та управління біном. Найпростішим способом є використання **SpringApplication** у Spring Boot:

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

Цей клас ініціалізує контейнер Spring і запускає додаток.

## Spring MVC

### 1. Основи Spring MVC

Spring MVC — це платформа на основі Java, яка в основному використовується для розробки веб-додатків. Він відповідає шаблону проектування MVC (Model-View-Controller). Цей шаблон проектування вказує, що програма складається з моделі даних, презентаційної інформації та керуючої інформації.

Ця структура розроблена навколо **DispatcherServlet** , який надсилає запити обробникам. У поточній галузі багато з них використовують Spring Boot Microservices, але є багато проектів, які все ще працюють у Spring MVC. Тож варто вивчити Spring MVC останнім часом. Ось чому ми збираємося впорядковано охопити всі речі, які є частиною Spring MVC framework.

**Spring Model-View-Controller:**

- **Модель** – модель містить дані програми. Набір даних може бути окремим об’єктом або групою речей.

- **Контролер** – контролер містить бізнес-логіку програми. Анотація @Controller використовується тут для визначення класу як контролера.
- **Перегляд** – це представлення наданої інформації в певному форматі. У більшості випадків для створення сторінки перегляду використовується JSP+JSTL. Однак Spring підтримує додаткові технології перегляду, такі як Apache Velocity, Thymeleaf і FreeMarker.
- **Передній контролер** – клас DispatcherServlet служить переднім контролером у Spring Web MVC. Він відповідає за керування потоком програми Spring MVC. [7]

**2. Робота з контролерами**

Для створення контролера використовуються анотації:

```
@Controller
public class MyController {
    @RequestMapping("/hello")
    public String sayHello(Model model) {
        model.addAttribute("message", "Hello, Spring MVC!");
        return "hello"; // Назва шаблону
    }
}
```

Контролер обробляє запит за адресою /hello і передає модель з повідомленням, яке буде відображене в шаблоні hello.jsp або hello.html.

**3. Інтеграція з шаблонами**

Spring MVC підтримує різні технології для створення представлень, зокрема Thymeleaf і JSP.

Для інтеграції потрібна бібліотека Thymeleaf-spring.

Ми додамо наступні залежності до нашого файлу Maven POM:

```xml
<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf</artifactId>
    <version>3.1.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf-spring5</artifactId>
    <version>3.1.2.RELEASE</version>
</dependency>
```

Зверніть увагу, що для проекту Spring 4 ми повинні використовувати бібліотеку thymeleaf-spring4 замість thymeleaf-spring5.

Клас SpringTemplateEngine виконує всі кроки конфігурації.

Ми можемо налаштувати цей клас як bean у файлі конфігурації Java:

```java
@Bean
@Description("Thymeleaf Template Resolver")
public ServletContextTemplateResolver templateResolver() {
    ServletContextTemplateResolver templateResolver = new ServletContextTemplateResolver();
    templateResolver.setPrefix("/WEB-INF/views/");
    templateResolver.setSuffix(".html");
    templateResolver.setTemplateMode("HTML5");

    return templateResolver;
}

@Bean
@Description("Thymeleaf Template Engine")
public SpringTemplateEngine templateEngine() {
    SpringTemplateEngine templateEngine = new SpringTemplateEngine();
    templateEngine.setTemplateResolver(templateResolver());
    templateEngine.setTemplateEngineMessageSource(messageSource());
    return templateEngine;
}
```

Префікс і суфікс властивостей бінів templateResolver вказують на розташування сторінок представлення в каталозі webapp і розширення їх імені файлу, відповідно.

Інтерфейс ViewResolver у Spring MVC відображає імена представлень, повернуті контролером, на реальні об'єкти представлення. ThymeleafViewResolver реалізує інтерфейс ViewResolver, і він використовується для визначення того, які представлення Thymeleaf відображає, отримавши ім'я представлення. [8]

Останнім кроком в інтеграції є додавання ThymeleafViewResolver як bean:

```java
@Bean
@Description("Thymeleaf View Resolver")
public ThymeleafViewResolver viewResolver() {
    ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
    viewResolver.setTemplateEngine(templateEngine());
    viewResolver.setOrder(1);
    return viewResolver;
}
```

# Spring Boot

## 1. Що таке Spring Boot?

Spring Boot — це інструмент, що значно спрощує процес налаштування та розгортання Spring-додатків. Він надає можливість використовувати автоматичну конфігурацію, що дозволяє зекономити час на налаштування.

Spring Boot є фреймворком відкритого джерела, який спрощує розробку веб застосунків на платформі Java. Він побудований на основі Spring Framework та надає розробникам зручний спосіб створення потужних і масштабованих додатків. Основними принципами роботи Spring Boot є зручність конфігурації, автоматична налаштування та зменшення зусиль для створення проектів.

Spring Boot пропонує конфігурацію "за замовчуванням" (convention over configuration), що означає, що багато основних налаштувань веб-додатку вже налаштовані за вас, дозволяючи швидко створювати проекти без необхідності у глибокій конфігурації. Фреймворк також надає багато автоматичних налаштувань, які дозволяють автоматично інтегрувати різні компоненти та бібліотеки без додаткової конфігурації [9].

## 2. Приклад реалізації простого продукту

Ось приклад реалізації простого продукту в Spring Boot, який включає основні елементи, такі як створення моделі продукту, контролера для обробки HTTP запитів, а також використання простого списку для

зберігання продуктів. Ми створимо невелике REST API для роботи з продуктами.

1. Структура проекту:

2. Кроки для створення проєкту:

## 2.1. Створення pom.xml для Maven

```xml
<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>com.example</groupId>
    <artifactId>spring-boot-product-demo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>spring-boot-product-demo</name>
    <description>Spring Boot product demo project</description>
    <properties>
        <java.version>11</java.version>
        <spring.boot.version>2.7.4</spring.boot.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starter Web for creating RESTful API -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- Spring Boot Starter Test for testing purposes -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

## 2.2. Створення основного класу програми ProductApplication.java

```java
package com.example.product;

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

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

2.3. Створення моделі продукту Product.java

Це клас, який буде представляти продукт.

```java
package com.example.product.model;

public class Product {
    private Long id;
    private String name;
    private Double price;

    public Product(Long id, String name, Double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }
}
```

2.4. Створення сервісу для продуктів ProductService.java

Цей клас міститиме логіку для роботи з продуктами. Оскільки це приклад простого проекту, ми будемо використовувати список в пам'яті для зберігання продуктів.

```java
package com.example.product.service;

import com.example.product.model.Product;
import org.springframework.stereotype.Service;

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

@Service
public class ProductService {

    private final List<Product> products = new ArrayList<>();

    public ProductService() {
        // Додаємо декілька продуктів для тестування
        products.add(new Product(1L, "Laptop", 1200.00));
        products.add(new Product(2L, "Smartphone", 800.00));
    }

    public List<Product> getAllProducts() {
        return products;
    }

    public Optional<Product> getProductById(Long id) {
        return products.stream().filter(product -> product.getId().equals(id)).findFirst();
    }

    public void addProduct(Product product) {
        products.add(product);
    }

    public void updateProduct(Long id, Product product) {
        for (int i = 0; i < products.size(); i++) {
            if (products.get(i).getId().equals(id)) {
                products.set(i, product);
                break;
            }
        }
    }

    public void deleteProduct(Long id) {
        products.removeIf(product -> product.getId().equals(id));
    }
}
```

2.5. Створення контролера для роботи з API ProductController.java

Цей клас відповідає за обробку HTTP запитів.

```java
package com.example.product.controller;

import com.example.product.model.Product;
import com.example.product.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    // Отримати список всіх продуктів
    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }

    // Отримати продукт за ID
    @GetMapping("/{id}")
    public ResponseEntity<Product> getProductById(@PathVariable Long id) {
        Optional<Product> product = productService.getProductById(id);
        return product.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
    }

    // Додати новий продукт
    @PostMapping
    public ResponseEntity<Product> addProduct(@RequestBody Product product) {
        productService.addProduct(product);
        return ResponseEntity.status(HttpStatus.CREATED).body(product);
    }

    // Оновити продукт
    @PutMapping("/{id}")
    public ResponseEntity<Product> updateProduct(@PathVariable Long id, @RequestBody Product product) {
        productService.updateProduct(id, product);
        return ResponseEntity.ok(product);
    }

    // Видалити продукт
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteProduct(@PathVariable Long id) {
        productService.deleteProduct(id);
        return ResponseEntity.noContent().build();
    }
}
```

## 3. Запуск і тестування

### 3.1. Запуск програми

Щоб запустити проект, використовуйте одну з команд:

- Для Maven: mvn spring-boot:run

- Для Gradle: gradle bootRun

3.2. Тестування API

- Отримати всі продукти:
  - GET http://localhost:8080/products
  - Відповідь: JSON зі списком продуктів.
- Отримати продукт за ID:
  - GET http://localhost:8080/products/1
  - Відповідь: JSON з продуктом, якщо такий існує, або 404, якщо продукт не знайдений.
- Додати новий продукт:
  - POST http://localhost:8080/products
  - Тіло запиту (JSON):

```
{
  "id": 3,
  "name": "Tablet",
  "price": 500.00
}
```

  - Відповідь: 201 Created з доданим продуктом.
- Оновити продукт:
  - PUT http://localhost:8080/products/1
  - Тіло запиту (JSON):

```
{
  "id": 1,
  "name": "Updated Laptop",
  "price": 1300.00
}
```

  - Відповідь: 200 OK з оновленим продуктом.

- Видалити продукт:
    - DELETE http://localhost:8080/products/1
    - Відповідь: 204 No Content.

Це простий приклад реалізації продукту в Spring Boot, що включає основні функції REST API для CRUD операцій (створення, читання, оновлення, видалення). Ви можете розширити цей проект, додаючи базу даних, валідацію, обробку помилок тощо.

## Переваги та недоліки використання Spring

Переваги використання Spring Framework:

1. Гнучкість та модульність

Spring надає модульну структуру, що дозволяє використовувати лише необхідні компоненти, як-от Spring MVC для веб-додатків, Spring Boot для автоматизації конфігурацій або Spring Data для роботи з базами даних. Це дозволяє створювати оптимізовані рішення без надмірного навантаження від непотрібних залежностей.

2. Інверсія керування (IoC) та вбудована залежність

Spring реалізує принцип Інверсії Керування (IoC), що дозволяє автоматично керувати об'єктами та їх залежностями. Це спрощує управління великими додатками, підвищує модульність коду та полегшує тестування.

3. Підтримка різних типів програмування

Spring підтримує як традиційне об'єктно-орієнтоване програмування (OOP), так і нові парадигми, такі як реактивне програмування через Spring WebFlux.

4. Актуальність та велика спільнота

Spring — це один із найпопулярніших Java-фреймворків, що забезпечує величезну спільноту розробників та активний розвиток. З нього можна отримати безліч готових рішень і зразків коду.

5. Інтеграція з іншими технологіями

Spring забезпечує зручну інтеграцію з іншими популярними технологіями, такими як Hibernate для роботи з базами даних, Apache Kafka для обробки потокових даних, та різними системами повідомлень і веб-сервісами.

6. Spring Boot і швидкий старт

Spring Boot дозволяє швидко розпочати проект без необхідності складної конфігурації. Це дає змогу швидко створювати застосунки з мінімумом коду та легко налаштовувати середовище.

Недоліки використання Spring Framework:

1. Крута крива навчання

Spring має багатий набір функціональностей, що може бути складним для початківців. Потрібно зрозуміти концепції, такі як інверсія керування, контейнери бінів, а також вивчити складну документацію.

2. Перевантаження функціональності

Завдяки широкому набору можливостей, багато розробників можуть відчувати перевантаження функціональністю та зайвими абстракціями, особливо коли використовуються лише невеликі частини фреймворка.

3. Продуктивність

Хоча Spring має багато оптимізацій, використання великих об'ємів залежностей та контекстів може призвести до певних проблем з продуктивністю в дуже високонавантажених системах.

4. Надмірність конфігурацій

У складних проектах можуть виникнути ситуації, коли необхідно налаштовувати безліч параметрів і конфігурацій, що може стати проблемою для збереження чистоти коду.

5. Складність налагодження

У великих проектах з великою кількістю залежностей і бінів, процес налагодження може стати складним і вимагати багато часу для виявлення проблем. [3]

# Висновок

Фреймворк Spring є потужним інструментом для розробки Java додатків, який дозволяє створювати масштабовані та ефективні рішення. Завдяки своїй модульності, гнучкості та потужним можливостям автоматичної конфігурації, Spring є одним із найпопулярніших фреймворків серед розробників. Вивчення основ роботи з Spring допомагає зрозуміти ключові концепції сучасної розробки на Java та відкриває можливості для використання фреймворку в реальних проектах.

У даній курсовій роботі ми розглянули основи роботи з фреймворком Spring, зокрема його основні компоненти, механізми ін'єкції залежностей, конфігурацію додатків, а також роботу з базами даних і реалізацію RESTful-сервісів. Також освоїли основи роботи із Spring Framework, його архітектуру, принципи ін'єкції залежностей та базові практики використання.

# Список використаних джерел:


1. Spring для лінивих. Основи, базові концепції та приклади з кодом. Частина 1 (javarush.com)

2. Spring and Spring Boot Frameworks: A Brief History (dzone.com)

3. Spring Framework: Офіційна документація фреймворку на сайті Spring: https://spring.io/projects/spring-framework

4. Wprowadzenie do Spring Framework - GeeksforGeeks

5. Стилі конфігурацій в Spring – Про Програмування (abitap.com)

6. 2.1. Spring IoC (Інверсія контролю) – Про Програмування (abitap.com)

7. Навчальний посібник із Spring MVC

8. Introduction to Using Thymeleaf in Spring | Baeldung

9. Microsoft Документація. [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what is-java-spring-boot/