ЛЕКЦІЯ №2.1. ВВЕДЕННЯ В SPRING BOOT

Кафедра інформаційних технологій та систем електронних комунікацій
Спеціальність 122 «Комп'ютерні науки», ступінь вищої освіти «бакалавр»
VI семестр
Дисципліна «Web-технології та web-дизайн»
ЗМІСТОВИЙ МОДУЛЬ 2. РОЗРОБКА ВЕБ-ДОДАТКІВ НА SPRING BOOT

Вступ

Сучасна backend-розробка характеризується високим рівнем складності програмних систем, великою кількістю інтеграцій, потребою у масштабованості, безпеці та швидкості розробки. У процесі створення серверних застосунків програміст повинен вирішувати широкий спектр завдань: обробку HTTP-запитів, взаємодію з базами даних, авторизацію користувачів, логування, конфігурацію середовища, тестування, обробку помилок та інтеграцію з іншими сервісами.

У класичній Java-розробці створення веб-додатка вимагало значної кількості конфігурації, ручного налаштування серверів, XML-файлів та великої кількості шаблонного коду. Це ускладнювало процес розробки та збільшувало час створення програмного забезпечення.

Для вирішення цих проблем було створено Spring Framework — один із найпотужніших та найпоширеніших Java-фреймворків у світі. На основі Spring Framework пізніше було розроблено Spring Boot — платформу, яка значно спрощує створення сучасних веб-додатків.

Spring Boot дозволяє розробнику швидко створювати production-ready застосунки з мінімальною кількістю конфігурації. Саме завдяки цьому Spring Boot став стандартом де-факто для enterprise Java-розробки.

На сьогодні Spring Boot використовується:

- у банківських системах;
- CRM та ERP-платформах;
- державних інформаційних системах;
- мікросервісній архітектурі;
- хмарних рішеннях;
- REST API;
- SaaS-платформах.

У межах лекції розглядаються основи Spring Boot, архітектура фреймворку, принципи його роботи, структура проєкту, автоматична конфігурація, dependency injection, створення першого веб-додатка та особливості використання Spring Boot у сучасній backend-розробці.

Поняття Spring Framework та Spring Boot

Spring Framework є комплексною платформою для розробки Java-застосунків. Основною метою Spring є спрощення створення масштабованих enterprise-систем.

Spring Framework складається з великої кількості модулів:

- Spring Core;
- Spring MVC;
- Spring Data;
- Spring Security;
- Spring Boot;
- Spring Cloud;
- Spring Batch.

Класичний Spring Framework надає надзвичайно широкі можливості, однак ранні версії вимагали складної конфігурації.

Наприклад, для створення простого веб-додатка необхідно було:

- вручну налаштовувати DispatcherServlet;
- конфігурувати XML-файли;
- встановлювати Tomcat;
- створювати WAR-архів;
- виконувати ручний deployment.

Spring Boot був створений як надбудова над Spring Framework із метою автоматизації конфігурації та пришвидшення розробки.

Основна ідея Spring Boot полягає у принципі:

«Convention over configuration».

Це означає, що система автоматично виконує типові налаштування без необхідності ручної конфігурації.

Spring Boot забезпечує:

- автоматичну конфігурацію;
- embedded server;
- starter-залежності;
- швидкий запуск проєкту;
- production-ready середовище;
- моніторинг;
- логування.

Архітектурна модель Spring Boot

Spring Boot базується на багаторівневій архітектурі, яка дозволяє розділити відповідальність між компонентами системи.

Типовий Spring Boot-проєкт містить:

- Controller Layer;
- Service Layer;
- Repository Layer;
- Database Layer.

Controller відповідає за взаємодію з HTTP-запитами.

Service реалізує бізнес-логіку.

Repository працює з базою даних.

Database Layer забезпечує фізичне зберігання інформації.

Подібна архітектура дозволяє:

- спрощувати підтримку коду;
- тестувати компоненти окремо;
- масштабувати систему;
- мінімізувати залежності.

Приклад структури Spring Boot-проєкту:

```
src
└── main
    ├── java
    │   └── com.example.demo
    │       ├── controller
    │       ├── service
    │       ├── repository
    │       ├── entity
    │       └── DemoApplication.java
    │
    └── resources
        ├── static
        ├── templates
        └── application.properties
```

Подібна структура використовується у більшості сучасних Java-проєктів.

Embedded Server у Spring Boot

Однією з найважливіших особливостей Spring Boot є використання embedded server.

У класичних Java EE-системах програміст повинен був окремо встановлювати Tomcat або інший сервер застосунків.

Spring Boot автоматично вбудовує сервер у застосунок.

Найчастіше використовується:

- Tomcat;
- Jetty;
- Undertow.

Після запуску проєкту сервер стартує автоматично.

Це значно спрощує розробку та deployment.

Приклад запуску Spring Boot:

```
@SpringBootApplication
public class DemoApplication {

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

Після запуску:

- створюється ApplicationContext;
- запускається embedded Tomcat;
- виконується автоматична конфігурація;
- застосунок починає приймати HTTP-запити.

Starter-залежності у Spring Boot

Spring Boot використовує starter-залежності.

Starter — це набір готових бібліотек для певного типу функціоналу.

Наприклад:

- spring-boot-starter-web;
- spring-boot-starter-data-jpa;
- spring-boot-starter-security.

Starter дозволяє уникати ручного підключення десятків залежностей.

Приклад Maven-конфігурації:

```
<dependencies>

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

   <dependency>
```

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

  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
  </dependency>

</dependencies>
```

Spring Boot автоматично визначає необхідні компоненти та конфігурує їх.

Автоматична конфігурація

Одним із ключових механізмів Spring Boot є Auto Configuration.

Spring Boot аналізує:

- наявні бібліотеки;
- конфігурацію;
- властивості application.properties.

На основі цього система автоматично створює необхідні Bean-компоненти.

Наприклад, якщо у проєкті присутній:

spring-boot-starter-data-jpa

Spring Boot автоматично:

- створює EntityManager;
- налаштовує Hibernate;
- підключає DataSource.

Це значно зменшує кількість конфігураційного коду.

Поняття Bean та IoC Container

Однією з фундаментальних концепцій Spring є Inversion of Control (IoC).

У традиційному програмуванні програміст сам створює об’єкти:

```
UserService service = new UserService();
```

У Spring Boot об’єкти створює контейнер Spring.

Такі об’єкти називаються Bean.

Приклад Bean:

```
@Service
public class UserService {
}
```

Spring автоматично:

- створює об’єкт;
- керує життєвим циклом;
- виконує dependency injection.

Dependency Injection

Dependency Injection є механізмом автоматичної передачі залежностей між компонентами.

Приклад:

```
@Service
public class UserService {

   private final UserRepository repository;

   public UserService(UserRepository repository) {
      this.repository = repository;
   }
}
```

Spring автоматично знаходить UserRepository та передає його у конструктор.

Dependency Injection забезпечує:

- слабку зв’язність;
- простіше тестування;
- масштабованість;
- кращу підтримуваність.

Створення першого REST API

Spring Boot широко використовується для створення REST API.

REST API дозволяє frontend та backend працювати незалежно один від одного.

Приклад простого контролера:

```
@RestController
@RequestMapping("/api")
public class HelloController {

   @GetMapping("/hello")
   public String hello() {
      return "Hello Spring Boot";
```

```
    }
}
```

Після запуску застосунку:

http://localhost:8080/api/hello

сервер поверне:

Hello Spring Boot

Конфігурація application.properties

Основна конфігурація Spring Boot міститься у файлі:

src/main/resources/application.properties

Приклад:

```
server.port=8080

spring.datasource.url=jdbc:mysql://localhost:3306/testdb
spring.datasource.username=root
spring.datasource.password=1234

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Через application.properties можна налаштовувати:

- сервер;
- базу даних;
- логування;
- безпеку;
- JWT;
- email;
- профілі середовищ.

Entity та робота з базою даних

Spring Boot активно інтегрується з JPA та Hibernate.

Entity-клас відповідає таблиці бази даних.

Приклад:

```
@Entity
@Table(name = "users")
public class User {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    private String email;
}
```

Repository дозволяє працювати з базою даних.

```
@Repository
public interface UserRepository
        extends JpaRepository<User, Long> {
}
```

Spring Boot автоматично реалізує CRUD-операції.

Життєвий цикл HTTP-запиту у Spring Boot

Коли клієнт надсилає HTTP-запит:

1. Запит приймає embedded Tomcat.
2. DispatcherServlet аналізує маршрут.
3. Викликається Controller.
4. Controller звертається до Service.
5. Service працює з Repository.
6. Repository взаємодіє з базою даних.
7. Відповідь повертається клієнту.

Схематично:

```
Client
  ↓
Tomcat
  ↓
DispatcherServlet
  ↓
Controller
  ↓
Service
  ↓
Repository
  ↓
Database
```

Переваги Spring Boot

Spring Boot став надзвичайно популярним завдяки:

- швидкому старту проєкту;
- мінімальній конфігурації;
- великій екосистемі;
- інтеграції з ORM;
- підтримці мікросервісів;
- високій масштабованості.

Велика кількість enterprise-компаній використовує саме Spring Boot.

Особливо популярним він є у:

- банківському секторі;
- fintech;
- e-commerce;
- cloud computing;
- державних системах.

Недоліки Spring Boot

Попри значну популярність, Spring Boot має певні недоліки.

Через велику кількість автоматизації початківці часто не розуміють внутрішніх механізмів роботи системи.

Також можливими є:

- надлишкове використання пам’яті;
- складність дебагу;
- приховані автоматичні конфігурації.

Саме тому backend-розробник повинен розуміти:

- Spring Core;
- Servlet;
- HTTP;
- JDBC;
- принципи роботи контейнера.

Висновки

Spring Boot є одним із найважливіших інструментів сучасної Java-розробки та фактичним стандартом створення backend-систем enterprise-рівня. Його використання дозволяє значно скоротити час розробки, автоматизувати конфігурацію та спростити побудову масштабованих веб-додатків.

У межах лекції було розглянуто архітектуру Spring Boot, embedded server, dependency injection, starter-залежності, автоматичну конфігурацію, роботу з REST API та інтеграцію з базами даних.

Особливу увагу приділено принципам IoC та Dependency Injection, які є фундаментом Spring Framework. Саме ці механізми дозволяють створювати гнучкі, підтримувані та масштабовані системи.

Розуміння Spring Boot є необхідною умовою для сучасного Java backend-розробника, оскільки більшість сучасних enterprise-проєктів використовують саме цей фреймворк.

Отримані знання є основою для подальшого вивчення:

- Spring MVC;
- Spring Security;
- Spring Data JPA;
- Hibernate;
- JWT;
- Docker;
- мікросервісної архітектури.

Завдання на самопідготовку

1. Встановити IntelliJ IDEA Community Edition.
2. Створити Spring Boot-проєкт через Spring Initializr.
3. Підключити spring-boot-starter-web.
4. Створити HelloController.
5. Реалізувати GET-запит.
6. Змінити порт сервера через application.properties.
7. Підключити MySQL.
8. Створити Entity та Repository.
9. Запустити проєкт.

Література

1. Craig Walls. Spring in Action.
2. Herbert Schildt. Java: The Complete Reference.
3. Spring Boot Documentation.
4. Spring Framework Documentation.
5. Hibernate Documentation.
6. Oracle Java Documentation.
7. Martin Fowler. Patterns of Enterprise Application Architecture.