ЛЕКЦІЯ №2.3. РОЗРОБКА ВЕБ-ДОДАТКІВ З 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

Розробка сучасних веб-додатків є одним із найважливіших напрямів програмної інженерії. Більшість сучасних інформаційних систем функціонують саме у веб-середовищі та працюють через браузер або REST API. Банківські сервіси, системи електронної комерції, освітні платформи, CRM-системи, державні портали та соціальні мережі побудовані за принципами веб-архітектури.

У Java-екосистемі одним із найпоширеніших інструментів для створення backend-систем є Spring Boot. Саме цей фреймворк дозволяє швидко створювати масштабовані серверні застосунки з мінімальною кількістю конфігурації.

Spring Boot значно спрощує створення:

- REST API;
- MVC-застосунків;
- мікросервісів;
- систем із базами даних;
- захищених backend-рішень.

Перевагою Spring Boot є інтеграція великої кількості технологій у межах єдиної платформи. Програміст може використовувати:

- Spring MVC;
- Spring Data JPA;
- Spring Security;
- Hibernate;
- Thymeleaf;
- REST API;
- JWT-авторизацію.

У межах цієї лекції розглядаються основи створення веб-додатків на Spring Boot, структура проєкту, контролери, шаблонізатори, робота з HTTP-запитами, інтеграція з базою даних, CRUD-операції та побудова багаторівневої backend-архітектури.

Архітектура веб-додатків на Spring Boot

Сучасний веб-додаток складається з кількох взаємопов’язаних компонентів.

Основними компонентами є:

- frontend;
- backend;
- база даних.

Frontend відповідає за взаємодію з користувачем.

Backend реалізує серверну логіку.

База даних забезпечує зберігання інформації.

Spring Boot використовується саме для реалізації backend-рівня.

Типова архітектура Spring Boot-застосунку базується на багаторівневому підході.

У системі зазвичай виділяють:

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

Controller відповідає за HTTP-запити.

Service реалізує бізнес-логіку.

Repository працює з базою даних.

Подібна архітектура дозволяє:

- спрощувати підтримку коду;
- масштабувати систему;
- тестувати компоненти окремо;
- уникати дублювання логіки.

Створення Spring Boot-проєкту

Найпоширенішим способом створення Spring Boot-проєкту є використання Spring Initializr.

Під час створення проєкту задаються:

- назва;
- groupId;
- artifactId;
- версія Java;
- starter-залежності.

Для створення базового веб-додатка зазвичай використовують:

spring-boot-starter-web
spring-boot-starter-thymeleaf
spring-boot-starter-data-jpa
mysql-connector-j

Після генерації проєкту Spring Boot створює готову структуру застосунку.

Основним класом є:

```
@SpringBootApplication
public class DemoApplication {

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

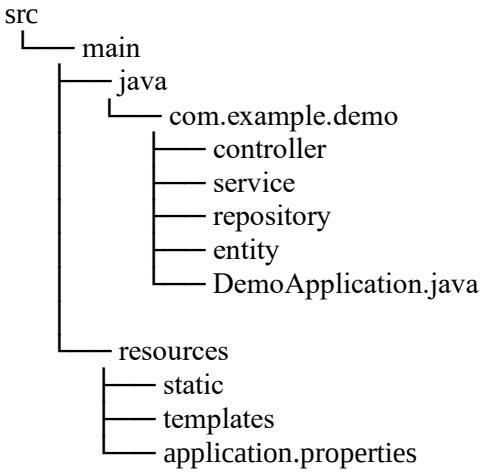
Анотація @SpringBootApplication об’єднує:

- @Configuration;
- @EnableAutoConfiguration;
-

Саме цей клас запускає embedded Tomcat та створює ApplicationContext.

Структура Spring Boot-проєкту

Типовий Spring Boot-проєкт має таку структуру:



Каталог controller містить контролери.

service містить бізнес-логіку.

repository відповідає за доступ до бази даних.

entity містить сутності.

templates використовується для HTML-шаблонів.

static містить CSS, JavaScript та зображення.

Контролери у Spring Boot

Контролер є компонентом, який обробляє HTTP-запити.

У Spring Boot контролери створюються за допомогою анотацій:

- @Controller;
-

2 використовується для REST API.

Приклад:

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

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

У цьому прикладі:

- 2 позначає REST-контролер;
- @RequestMapping задає базовий URL;
- @GetMapping обробляє GET-запити.

Після запуску застосунку запит:

http://localhost:8080/api/hello

поверне:

Hello Spring Boot

Робота з HTTP-запитами

Spring Boot підтримує всі основні HTTP-методи.

GET використовується для отримання інформації.

POST використовується для створення нових даних.

PUT використовується для оновлення.

DELETE використовується для видалення.

Приклад POST-запиту:

```
@PostMapping("/users")
public User createUser(@RequestBody User user) {
   return userService.save(user);
}
```

Анотація @RequestBody автоматично перетворює JSON у Java-об’єкт.

Наприклад:

```
{
  "username":"ivan",
  "email":"ivan@gmail.com"
}
```

буде перетворено у User.

MVC-підхід у Spring Boot

Spring Boot підтримує архітектуру MVC.

MVC означає:

- Model;
- View;
- Controller.

Controller приймає запити.

Model містить дані.

View відповідає за відображення.

Для View у Spring Boot часто використовується Thymeleaf.

Thymeleaf та HTML-шаблони

Thymeleaf є серверним шаблонізатором для Java.

HTML-файли розташовуються у:

src/main/resources/templates

Приклад контролера:

```
@Controller
public class PageController {

   @GetMapping("/home")
   public String home(Model model) {
```

```
        model.addAttribute("message",
                "Hello Thymeleaf");

        return "home";
    }
}
```

HTML-шаблон:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Home</title>
</head>
<body>

<h1 th:text="${message}"></h1>

</body>
</html>
```

Thymeleaf інтегрується зі Spring Boot автоматично.

Робота з базою даних

Spring Boot активно використовується для роботи з реляційними базами даних.

Найчастіше використовуються:

- MySQL;
- PostgreSQL;
- H2;
- MariaDB.

Конфігурація бази даних задається через:

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

spring.jpa.hibernate.ddl-auto=update
```

Entity-класи

Entity є Java-класом, який 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;
}
```

Анотація @Entity повідомляє Hibernate, що клас є сутністю.

Repository Layer

Repository використовується для роботи з базою даних.

Spring Data JPA дозволяє автоматично створювати CRUD-операції.

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

Після цього автоматично стають доступними:

- save();
- findAll();
- findById();
- deleteById().

Service Layer

Service Layer реалізує бізнес-логіку.

```
@Service
public class UserService {

   private final UserRepository repository;

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

   public List<User> findAll() {
      return repository.findAll();
   }
}
```

Service не повинен містити HTTP-логіку.

Його завданням є реалізація правил предметної області.

CRUD-операції

CRUD означає:

- Create;
- Read;
- Update;
- Delete.

Приклад створення користувача:

```
@PostMapping
public User create(@RequestBody User user) {
   return repository.save(user);
}
```

Отримання всіх користувачів:

```
@GetMapping
public List<User> getAll() {
   return repository.findAll();
}
```

Оновлення:

```
@PutMapping("/{id}")
public User update(
       @PathVariable Long id,
       @RequestBody User user
) {
   user.setId(id);
   return repository.save(user);
}
```

Видалення:

```
@DeleteMapping("/{id}")
public void delete(@PathVariable Long id) {
   repository.deleteById(id);
}
```

DTO та передача даних

У production-системах не рекомендується повертати Entity напряму.

Для цього використовуються DTO.

```
public class UserDto {
```

```
    private String username;

    private String email;
}
```

DTO забезпечують:

- безпеку;
- контроль API;
- ізоляцію внутрішньої структури.

Обробка помилок

Spring Boot підтримує централізовану обробку помилок.

Приклад:

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handle(
            Exception ex
    ) {
        return ResponseEntity
                .status(500)
                .body(ex.getMessage());
    }
}
```

Це дозволяє централізовано керувати помилками системи.

Валідація даних

Spring Boot підтримує Bean Validation.

```
@NotBlank
private String username;

@Email
private String email;
```

Контролер:

```
public User create(
        @Valid @RequestBody User user
) {
}
```

Logging у Spring Boot

Логування є важливою частиною backend-систем.

Приклад:

```
private static final Logger logger =
      LoggerFactory.getLogger(UserService.class);

logger.info("User created");
logger.error("Database error");
```

Логування використовується для:

- дебагу;
- моніторингу;
- аналізу помилок.

Запуск та тестування застосунку

Spring Boot запускається через main()-метод.

Після запуску embedded Tomcat починає приймати HTTP-запити.

Тестування REST API можна виконувати через:

- браузер;
- Postman;
- Apidog;
- Swagger.

Висновки

Spring Boot є одним із найважливіших фреймворків сучасної Java-розробки та широко використовується для створення backend-систем enterprise-рівня.

У межах лекції було розглянуто архітектуру веб-додатків на Spring Boot, структуру проєкту, контролери, Thymeleaf, REST API, роботу з базами даних, CRUD-операції, DTO, валідацію та logging.

Особливу увагу приділено багаторівневій архітектурі та розділенню відповідальності між компонентами системи.

Розуміння принципів побудови веб-додатків на Spring Boot є фундаментом для подальшого вивчення:

- Spring Security;
- JWT;
- Hibernate;
- Docker;

- мікросервісної архітектури;
- cloud-native systems.

Завдання на самопідготовку

1. Створити Spring Boot-проєкт.
2. Реалізувати REST Controller.
3. Створити HTML-шаблон через Thymeleaf.
4. Підключити MySQL.
5. Створити Entity.
6. Реалізувати CRUD-операції.
7. Додати DTO.
8. Реалізувати валідацію.
9. Протестувати API через Postman.

Література

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