ЗАТВЕРДЖУЮ
Начальник кафедри інформаційних технологій та систем електронних комунікацій, к.т.н., доцент
Назарій БУРАК
«____» _____ 20___ р.

МЕТОДИЧНА РОЗРОБКА

для проведення лабораторного заняття із здобувачами освіти першого (бакалаврського) рівня вищої освіти спеціальності F3 «Комп'ютерні науки»
з дисципліни «WEB-технології та WEB-програмування»

Лабораторна робота № 5: «Створення проєкту Spring Boot за допомогою Spring Initializr. Впровадження залежностей, робота з бінами» (тема 2.1)

Мета: навчитися створювати проєкт Spring Boot у Spring Initializr, переносити в нього шари застосунку, оголошувати біни і впроваджувати залежності через конструктор, працювати з БД через `DataSource` і `JdbcClient`, використовувати декларативні транзакції і створювати перші REST-контролери.

Час: 4 години.

Місце заняття: комп'ютерна лабораторія.

Матеріальне забезпечення: ПК з доступом до мережі Internet; JDK 21; IntelliJ IDEA; Maven; PostgreSQL; Postman або curl; браузер Chrome; Git; демо-проєкт https://github.com/SidiromUA/shelter-registry (тег `lab-05`).

Після лабораторного заняття здобувачі освіти повинні:

вміти: генерувати проєкт у Spring Initializr і запускати його; позначати класи стереотипними анотаціями; оголошувати біни методами `@Bean`; впроваджувати залежності через конструктор і розв'язувати неоднозначність бінів; налаштовувати підключення до PostgreSQL; виконувати запити через `JdbcClient`; позначати транзакційні методи `@Transactional`; створювати REST-контролери з `@GetMapping`, `@PathVariable`, `@RequestParam`; збирати виконуваний JAR;

знати: принципи IoC і DI, поняття біна, область видимості singleton і етапи життєвого циклу біна, призначення стартерів і автоконфігурації, структуру проєкту Spring Boot, шлях запиту через `DispatcherServlet`.

I. Підготовка до заняття

Викладач перевіряє на ПК лабораторії наявність JDK 21, Maven, запущеної БД PostgreSQL і доступу до start.spring.io. Здобувачі освіти до заняття опрацьовують лекцію 5 і мають власний проєкт з ЛР 4 (шари model, dto, repository, service, БД з ЛР 2).

II. Структура заняття

Короткі теоретичні відомості

Spring Boot – надбудова над Spring Framework, що дозволяє створювати готові до запуску застосунки з мінімумом налаштувань. Об'єкти застосунку (**біни**) створює і зв'язує **IoC-контейнер** (`ApplicationContext`): клас, позначений `@Component`, `@Service`, `@Repository`, `@RestController`, стає біном під час сканування пакета головного класу; бін з класу JDK чи сторонньої бібліотеки оголошують методом `@Bean` у класі `@Configuration`. Залежності передаються через **конструктор**; якщо конструктор один, `@Autowired` не потрібен. За замовчуванням біни – одинаки (singleton), тому їх поля містять лише залежності. **Стартери** підключають узгоджені набори бібліотек, **автоконфігурація** створює `DataSource`, `JdbcClient`, `DispatcherServlet` і вбудований Tomcat. `@Transactional` на методі сервісу замінює ручні `commit()`/`rollback()`. `@RestController` повертає дані, які Jackson перетворює на JSON.

2.1. Огляд демо-проєкту

```
git checkout lab-05
git diff lab-04 lab-05 --stat
mvn spring-boot:run                # http://localhost:8080/hello
```

Розгляньте, що змінилося порівняно з `lab-04`: у `pom.xml` з'явився батьківський POM `spring-boot-starter-parent` і стартери `webmvc`, `jdbc`; видалено пакет `web` (сервлети, фільтри, слухач), каталог `webapp`, класи `Database` і `DevServer`; репозиторії переписано на `JdbcClient`; додано пакети `api` і `config`, файл `application.properties` і тест.

Клас	Вид біна	Залежності (через конструктор)
`ShelterApiController`, `HelloController`	`@RestController`	`ShelterService`
`ShelterService`	`@Service`	три репозиторії, `Clock`
`JdbcShelterRepository` та ін.	`@Repository`	`JdbcClient` (автоконфігурація)
`AppConfig.clock()`	`@Bean`	–
`StartupReport`	`@Component`, `CommandLineRunner`	`ShelterService`

Таблиця 1 – Біни демо-проєкту (тег lab-05)

Під час запуску знайдіть у журналі повідомлення 1–3 від `StartupReport`, рядок ініціалізації пулу HikariCP і «Tomcat started on port 8080». Зупиніть застосунок (Ctrl+C) і знайдіть повідомлення 4 (`@PreDestroy`).

2.2. Дослідження бінів і REST API

1. У Postman (або curl) виконайте запити з таблиці 2 і перевірте коди стану та формат відповіді.
2. Тимчасово видаліть анотацію `@Service` з `ShelterService` і запустіть застосунок. Прочитайте повідомлення про помилку («required a bean of type … that could not be found»). Поверніть анотацію.
3. Створіть другу реалізацію `ShelterRepository` (наприклад, порожній клас `InMemoryShelterRepository` з `@Repository`) і запустіть застосунок – отримайте помилку неоднозначності. Розв’яжіть її анотацією `@Primary`, потім видаліть тестовий клас.
4. Запустіть застосунок з параметром `--debug` і знайдіть у звіті «CONDITIONS EVALUATION REPORT» автоконфігурації `DataSourceAutoConfiguration` і `JdbcClientAutoConfiguration`.
5. Зберіть JAR (`mvn package`) і запустіть його на порту 9090: `java -jar target/shelter-registry-0.5.0.jar --server.port=9090`.

Запит	Очікувана відповідь
`GET /hello?name=Олена`	200, текст привітання
`GET /api/shelters`	200, масив JSON усіх укриттів
`GET /api/shelters?districtId=6`	200, укриття району
`GET /api/shelters/3`, `/api/shelters/3/inspections`	200, об’єкт укриття; перевірки укриття
`GET /api/shelters/999`	404, `{"error": "…"}`
`GET /api/shelters/abc`	400 (неможливо перетворити на число)

Таблиця 2 – Запити для перевірки

2.3. Індивідуальне завдання

Перенесіть власний проєкт з ЛР 4 на Spring Boot:

1. Створіть у Spring Initializr (start.spring.io або IntelliJ IDEA) проєкт Maven: Spring Boot 4.1.x, Java 21, Packaging Jar, Group – `ua.edu.ldubgd`, Artifact – назва вашого проєкту; залежності Spring Web, JDBC API, PostgreSQL Driver. Розпакуйте його в репозиторій з ЛР 1–4 (новою гілкою або замість старого коду).
2. Перенесіть пакети `model`, `dto`, `exception`, `repository`, `service` у підпакети кореневого пакета. Видаліть сервлети, фільтри, слухач, JSP і клас ручного підключення до БД.
3. У `application.properties` задайте параметри підключення до своєї БД з ЛР 2 і назву застосунку.
4. Позначте класи стереотипними анотаціями; передавайте всі залежності через конструктор з полями `final` (без `@Autowired` на полях).
5. Перепишіть репозиторії на `JdbcClient` з іменованими параметрами і `RowMapper`; для вставлення отримуйте згенерований ключ.
6. Позначте `@Transactional` щонайменше один метод сервісу, що змінює дві таблиці. Переконайтеся, що в разі винятку зміни відкочуються (наприклад, тимчасово згенеруйте виняток між двома операціями).
7. Оголосіть щонайменше один бін методом `@Bean` у класі `@Configuration` (`Clock` або інший) і використайте його в сервісі.
8. Створіть клас з `@PostConstruct`, `@PreDestroy` і `CommandLineRunner`, що виводить у журнал етапи життєвого циклу і кількість записів головної сутності.

9. Створіть `@RestController` з префіксом `/api` і щонайменше чотирма GET-маршрутами: список, фільтрація через `@RequestParam`, отримання за ідентифікатором через `@PathVariable` з кодом 404 для відсутнього об’єкта, пов’язані дані (сутність 1:N).
10. Переконайтеся, що тест `contextLoads` (`mvn test`) проходить, і зберіть виконуваний JAR.
11. Оновіть `README.md` (інструкція запуску, перелік маршрутів), зафіксуйте зміни, створіть тег `lab-05` і надішліть на GitHub.

Додаткові завдання (за бажанням): додайте `spring-boot-starter-actuator` і перевірте `/actuator/health`; створіть тест сервісу з «зупиненим» годинником `Clock.fixed(...)` без запуску Spring; підключіть Spring Boot DevTools.

2.4. Вимоги до звіту

Звіт (PDF або DOC, до 6 сторінок) містить: мету; параметри проєкту в Spring Initializr (знімок); структуру пакетів; таблицю бінів вашого застосунку з їх залежностями (як табл. 1); фрагменти коду репозиторію на `JdbcClient`, транзакційного методу і REST-контролера; знімки журналу запуску з етапами життєвого циклу; знімки Postman для кожного маршруту, включно з відповіддю 404; посилання на репозиторій з тегом `lab-05`; відповіді на 3 контрольні запитання; висновки. У ВУ в завданні «ЛР 5» у поле «Текстова відповідь» вставте посилання на репозиторій, файлом додайте звіт.

2.5. Контрольні запитання

1. Що таке інверсія керування і впровадження залежностей?
2. Чому рекомендують впроваджувати залежності через конструктор?
3. Як оголосити бін? Коли використовують метод `@Bean`?
4. Що станеться, якщо є два біни одного типу? Як це виправити?
5. Чому бін-одинак не повинен зберігати в полях стан запиту?
6. Назвіть етапи життєвого циклу біна.
7. Що таке стартер і автоконфігурація? Які анотації поєднує `@SpringBootApplication`?
8. Як працює `@Transactional` і чому його розміщують на методах сервісу?
9. Опишіть шлях запиту `GET /api/shelters/3` від клієнта до БД і назад.

2.6. Критерії оцінювання (максимально 5 балів)

Бали	Критерій
5	проєкт створено в Spring Initializr і запускається; DI лише через конструктор; репозиторії на `JdbcClient`; працює транзакція з відкатом; є бін `@Bean` і демонстрація життєвого циклу; REST-контролер з чотирма маршрутами і кодом 404; тест проходить; тег `lab-05` на GitHub; здобувач вільно пояснює IoC, DI і роботу контейнера
4	робота виконана повністю, є незначні недоліки (напр. впровадження через поле, немає демонстрації життєвого циклу)
3	виконано не менше 60 % (напр. немає транзакції або частина репозиторіїв на «чистому» JDBC) або є суттєві помилки
0–2	виконано менше 60 % або здобувач не може пояснити поданий код

Робота, подана після встановленого терміну без поважних причин, оцінюється з пониженням на 1 бал.

III. Підведення підсумків заняття

Викладач перевіряє роботу застосунків, звертає увагу на типові помилки (класи поза пакетом головного класу, `@Autowired` на полях, створення залежностей через `new`, стан у полях сервісу, `@Transactional` на репозиторії або на приватному методі, паролі до БД у публічному репозиторії без пояснення), повідомляє, що в ЛР 6 буде налаштовано профілі, журналювання і контейнеризацію.

Методичну розробку розробив:

Доцент кафедри ІТтаСЕК, к.т.н. Роман ГОЛОВАТИЙ

Методична розробка обговорена на засіданні кафедри ІТтаСЕК
Протокол № ____ від «____» _____ 20___ р.