NNPIA LAB 02: Dependencies, beany a Lombok

Cílem tohoto cvičení je pochopit, jak Spring Boot funguje pod povrchem. Seznámíme se s pojmem bean, aplikační kontext a
principem Inversion of Control. Ukážeme si, jak Spring vytváří a propojuje objekty bez nutnosti jejich ručního instancování. Představíme si i možnosti pro odstranění zbytečného boilerplate kódu pomocí Lomboku.

Předpoklady

- JDK 21 nebo novější
- IntelliJ IDEA
    - Doporučena je verze **IntelliJ IDEA Ultimate**
    - Studenti mohou využít **bezplatnou studentskou licenci**
- Verzovací systém Git
- **Webový prohlížeč** ideálně postavený na jádru Chromium
    - Google Chrome, Microsoft Edge, Brave...
- **HTTP klient pro testování API**
    - Postman
    - Insomnia

2.1. Přidání závislostí

Jako vývojář nikdy netvoříte projekt od základů, ale pracujete s množstvím knihoven a frameworků, které vám usnadňují
vývoj. Tyto knihovny a frameworky jsou přidávány do projektu pomocí závislostí. Build nástroje jako `Gradle` nebo
`Maven` nám umožňují snadno spravovat tyto závislosti.

1. Soubor `LAB02.md` vložte do složky `doc` a zahrňte do verzování.
2. Otevřete soubor `build.gradle` a přidejte do projektu následující závislosti v sekci `dependencies`:

```
implementation 'org.springframework.boot:spring-boot-starter-actuator'
developmentOnly 'org.springframework.boot:spring-boot-devtools'
testImplementation 'org.springframework.boot:spring-boot-starter-actuator-test'
```

💡 *Pro rychlé otevření souboru `build.gradle` můžete v IntelliJ využít dvojité stisknutí **Shift** a začít psát název souboru.*

💡 `testImplementation` slouží k přidání závislostí, které jsou dostupné pouze při běhu testů. Tyto knihovny nejsou
součástí produkční aplikace a nebudou součástí výsledného artefaktu.

❗ Závisloti přidejte, nenahrazujte již existující z minulého cvičení.

2. Synchronizujte Gradle projekt pro stáhnutí nových závislostí. V pravém menu vyberte ikonku Gradle a klikněte na
   tlačítko "Sync all Gradle projects".
3. Restartujte Spring Boot aplikaci, tentokrát však přes hlavní třídu:
   - Třída je označena anotací `@SpringBootApplication`:
   - Naleznete jí ve složce `src/main/java/com/cz/upce/fei/backend/BackendApplication.java`

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

4. Spusťte aplikaci přes tuto třídu (pravým tlačítkem na třídu -> Run 'BackendApplication.main()').

💡 *Aplikaci můžete spustit i přes zelenou ikonku vedle čísla řádku s hlavní třídou.*

💡 *To, že jste aplikaci spustili přes hlavní třídu, poznáte tak, že run konfigurace obsahuje ikonku listu a název třídy.*

5. Všimněte si, že v run konzoli IntelliJ je více panelů, konkrétně `Console`, `Beans`, `Health`, `Mappings` a další.
   - Tyto panely jsou dostupné díky přidané závislosti `spring-boot-starter-actuator`.
   - Prozkoumejte jednotlivé panely a jejich funkce.
   - Všimněte si, že v `Mappings` jsou vypsány všechny endpointy vaší aplikace.

💡 *Jeden z endpointů znáte z předchozího cvičení.*

6. Založte nový controller `AppUserController.java` v balíčku `com.cz.upce.fei.backend.controller`.
   - Třídu a metody opatřete všemi potřebnými anotacemi tak, aby byla mapována na `/api/v1/app-users`.
   - Vytvořte v ní metodu `getAllAppUsers()`, která zatím vrátí prázdnou kolekci.
   - Jako návratový typ použijte zatím `ResponseEntity<?>`.
   - GET HTTP metoda.

💡 *Použijte `HealthController.java` jako vzor pokud si nejste jisti implementací endpointu.*

7. Restartujte aplikaci a Ověřte, že nový endpoint funguje:
   - Ověřte že nový endpoint je vidět v panelu `Mappings` v run konzoli IntelliJ.
   - Použijte Postman nebo webový prohlížeč a zavolejte `http://localhost:8080/api/v1/app-`

`users`.

- Měli byste obdržet prázdnou odpověď `[]`.

💡 *URL adresu můžete otevřít i z panelu `Mappings`. Stačí kliknout na URI endpointu v prvním sloupčku.*

❗ Využívat knihovny třetích stran je skvělé. Nebo ne? Je dobré sledovat zranitelnosti v knihovnách, které používáme. Může
se snadno stát, že si do projektu zaneseme chyby nebo škodlivý kód:

- What We Know About the NPM Supply Chain Attack
- What is Log4Shell? The Log4j vulnerability explained (and what to do about it)

2.2. Beany

Beany jsou základním stavebním kamenem Spring frameworku. Jsou to instance tříd, které jsou spravovány Springem a mohou
být vkládány do jiných tříd podle návrhového vzoru `Dependency injection`. Jednotlivé beany jsou obvykle označeny pomocí
anotací jako `@Component`, `@Service`, `@Repository` , `@Bean` nebo `@Controller`. To zajišťuje, že se programátor nemusí
starat o správu instancí, ale pouze o jejich vytvoření a používání.

Spring Boot Cvičení

8. Vytvořte nový package `com.cz.upce.fei.backend.service` a v něm třídu `AppUserService.java`.
   - Třídu označte pomocí anotace `@Service`.
   - Vytvořte v ní metodu `getAllAppUsers()`, která vrátí kolekci s náhodnými řetězci.
   - Třídě nedefinujte žádný konstruktor ani atributy.
9. Restartujte aplikaci a ověřte, že se naše nově přidaná beana objevila v panelu `Beans` v run konzoli IntelliJ.
   - V panelu `Beans` jsou zobrazeny všechny beany, které jsou součástí aplikačního kontextu.

💡 *Můžete použít vyhledávání v horní části. Nebo začít psát název třídy pro průběžné filtrování.*

💡 *To, že se nově přidaná třída stala beanou, lze indikovat i zelenou ikonkou fazolky u deklarace třídy nebo v logách
pokud máte nastavený root level na DEBUG:*

```
2026-02-12T21:23:17.324+01:00 DEBUG 21101 --- [backend] [  restartedMain]
o.s.b.f.s.DefaultListableBeanFactory     : Creating shared instance of singleton bean
'appUserService'
```

10. Nyní, když víme, že je beana součástí aplikačního kontextu,, použijeme ji v našem controlleru.
    - V `AppUserController.java` přidejte atribut `AppUserService appUserService`.
    - Nově přidaný atribut opatřete anotací `@Autowired`.
    - V metodě `getAllAppUsers()` zavolejte metodu `appUserService.getAllAppUsers()` a její

výsledek vraťte v těle
odpovědi.

```
@Autowired
private AppUserService appUserService;
```

11. Restartujte aplikaci a ověřte že endpoint nyní vrací náhodná jména.

💡 *Anotace `@Autowired` zajistí vložení instance (beany) z aplikačního kontextu do atributu. To že do atributu bude*
vložena beana z aplikačního kontextu lze indikovat i zelenou ikonkou fazolky s šipkou vedke deklarace atributu a nebo v
logách pokud máte nastavený root level na DEBUG:

```
2026-02-12T21:23:17.325+01:00 DEBUG 21101 --- [backend] [  restartedMain]
o.s.b.f.s.DefaultListableBeanFactory     : Autowiring by type from bean name
'appUserController' via constructor to bean named 'appUserService'
```

💡 Jak by to vypadalo bez Springu? Bez Dependency Injection by controller mohl vytvářet instanci služby ručně tak jak to
znáte ze základů programování. Při větším množství navzájem provázaných tříd by to ale vedlo k těžko udržitelnému
kódu:

```
private AppUserService appUserService = new AppUserService();
```

12. **Anotace `@Autowired` není v dnešní době doporučeným způsobem vložení závislostí**.
    - Doporučeným způsobem je vložení přes konstruktor. Této technice se říká `Constructor injection`.
    - Odeberte anotaci `@Autowired` z atributu `appUserService`.
    - Vytvořte konstruktor, který přijímá `AppUserService jako parametr.

```
private final AppUserService appUserService;

public AppUserController(AppUserService appUserService) {
    this.appUserService = appUserService;
}
```

💡 *Tato technika se nazývá `Inversion of Control`. Třída definuje své závislosti (v tomto případě `AppUserService`)*
ale neřídí jejich vytvoření. O to se stará Spring.

13. Restartujte aplikaci a ověřte že funkčnost endpointu zůstala zachována.

💡 Spring ve výchozím nastavení vytváří beany jako `singleton`.

2.3. Lombok

Lombok je knihovna, která pomáhá eliminovat boilerplate kód v Javě. Pomocí jednoduchých anotací lze generovat konstruktory, gettery, settery, `toString()`, `equals()` a `hashCode()` metody a další. Všechny tyto anotace
jsou nahrazeny deklaracemi před procesem kompilace. Takže nejsou viditelné v samotném zdrojovém kódu, ale jsou přítomny
v bytecode a dostupné během běhu programu.

💡 *Starší verze IntelliJ neumí bez pluginů s Lombokem pracovat a mohou hlásit kompilační chyby v kódu.*

1. Otevřete soubor `build.gradle` a přidejte do projektu následující závislosti v sekci `dependencies`:

```
compileOnly 'org.projectlombok:lombok'
annotationProcessor 'org.projectlombok:lombok'
```

2. Synchronizujte Gradle projekt pro stáhnutí nových závislostí. V pravém menu vyberte ikonku Gradle a klikněte na
   tlačítko "Sync all Gradle projects".
3. Vytvořte nový package `com.cz.upce.fei.backend.domain` a v něm třídu `AppUser.java`.
   - Třída bude představovat model uživatele a bude mít následující atributy:
     - `private Long id;`
     - `private String email;`
     - `private String password;`
     - `private Boolean active;`
   - Pro všechny atributy vygenerujte parametrický konstruktor, gettery, settery, `toString()`, `equals()` a
     `hashCode()` metody.
   - Nejdříve po staru pomocí IntelliJ IDEA (pravý klik na třídu -> Generate -> Constructor / Getters and Setters /
     toString() / equals() and hashCode()).
4. Ve třídě `AppUserService` přidejte nový atribut `private List<AppUser> appUsers;`.
   - Tento atribut bude představovat seznam uživatelů, který bude naše služba spravovat.
   - Inicializujte tento atribut na alespoň 3 náhodné hodnoty typu `AppUser`.
   - Ujistěte se, že kolekci bude možné upravovat (tzv. bude `mutable`), protože v budoucnu do ní budeme přidávat nové
     uživatele.

💡 *Použijte volně dostupné AI k vytvoření vzorových dat.*

5. Ve třídě `AppUserService` vytvořte logger a přidejte logovací výstup do metody `getAllAppUsers()`, který vypíše
   uživatele v kolekci do INFO levelu v seznamu `appUsers`.

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

```
logger.info("About to retrieve: {}",appUsers);
```

💡 *V případě třídy Logger máme více kandidátů na import. Ujistěte se že importujete třídu `org.slf4j.Logger`*

6. Upravte metodu `getAllAppUsers()` tak aby místo prázdné kolekce vracela seznam uživatelů z atributu `appUsers`.
7. Restartujte aplikaci a ověřte že endpoint nyní vrací seznam uživatelů místo prázdného pole.

💡 *Všiměte si, že odpověď je ve formátu `JSON`. To není náhoda ale jedna ze specifikací RESTu.*

8. Nyní využijeme Lombok pro odstranění zbytečného boilerplate kódu.
    - Postupně odstaňte následující ze třídy `AppUser`:
        - Odstraňte parametrický konstruktor.
            - Třídu místo toho označte anotací `@AllArgsContructor`.
            - Ověřte, že IntelliJ neupozorňuje na žádnou kompilační chybu.
            - Restartujte aplikaci a ověřte že endpoint stále vrací seznam uživatelů.
        - Odstraňte všechny gettery a settery.
            - Třídu místo toho označte anotacemi `@Getter` a `@Setter`.
        - Odstraňte metodu `toString()`:
            - Třídu místo toho označte anotací `@ToString`.
            - Ověřte že v logu se stále vypisuje obsah kolekce `appUsers` i po odstranění metody `toString()`.
9. Nyní, když již máme definovanou třídu `AppUser`, upravte návratový typ metody `getAllAppUsers()` v `AppUserController`.
    - Odstraňte generický návratový typ `ResponseEntity<?>`.
    - Nahraďte jej konkrétním typem `ResponseEntity<List<AppUser>>`.
    - Ověřte, že aplikace nadále funguje a endpoint vrací JSON pole uživatelů.

💡 Používání konkrétních typů zvyšuje čitelnost kódu a typovou bezpečnost.

2.4. Refactoring pomocí Lomboku

I když předchozí cvičení odstranilo velké množství boilerplate kódu ve třídě `AppUser`, stále je zde prostor pro
zlepšení.

Vibe coding Dobrovolný úkol

1. Využijte funkci `Agent` a sepište prompt, který nahradí veškerý boilerplate kód typický pro Javu pomocí anotací
   Lomboku.

    - Při psaní promptu dbejte na následující:
        - Neomezujte své instrukce pouze na třídu `AppUser`, ale zvažte i další třídy ve vašem projektu.
        - Některé anotace Lomboku lze sloučit do jedné.
        - Pomocí Lomboku lze vytvářet i statické atributy.
        - Instruujte agenta aby vám vysvětlil proč daný kód nahradil anotací a jak funguje.

2.5. Logování citlivých údajů

V rámci cvičení logujeme soukromé a citlivé informace o uživatelích, což není nejlepší praxe. V reálném světě mohou být
logy uchovávány po dlouho dobu. V produkčním prostředí navíc už pracujeme s reálnými uživateli a jejich daty.

Vibe coding Dobrovolný úkol

1. Využijte funkci `Agent` a otestujte, zda AI dokáže tuto problematiku identifikovat a navrhnout řešení.
    - Při psaní promptu dbejte na následující:
        - Určete vhodnout roli pro agenta. Nyní už nejedná čistě v roli vývojáře.
        - Pomocí agenta si nechte vysvětlit proč je daný kód problematický.
        - Využijte vlastní kritické myšlení při validaci výstupu.
        - Nechte si navrhnout řešení pro danou problematiku.
2. Poté využijte funkci `Agent` a sepište prompt který implementuje navržené řešení do našeho projektu.

Odevzdání

- Po dokončení všech úkolů vytvořte **commit** se všemi provedenými změnami a **pushněte jej do vzdáleného repozitáře**.
- Název commitu musí **začínat označením LAB02** a obsahovat **stručný popis změn**.

```
LAB02 -  Závislosti, beany a Lombok
```

Užitečné odkazy a zdroje

- What Is a Spring Bean?
- Quick Guide to Spring Bean Scopes
- Lombok Essentials: Enhancing Java Coding Efficiency and Maintainability
- Singleton