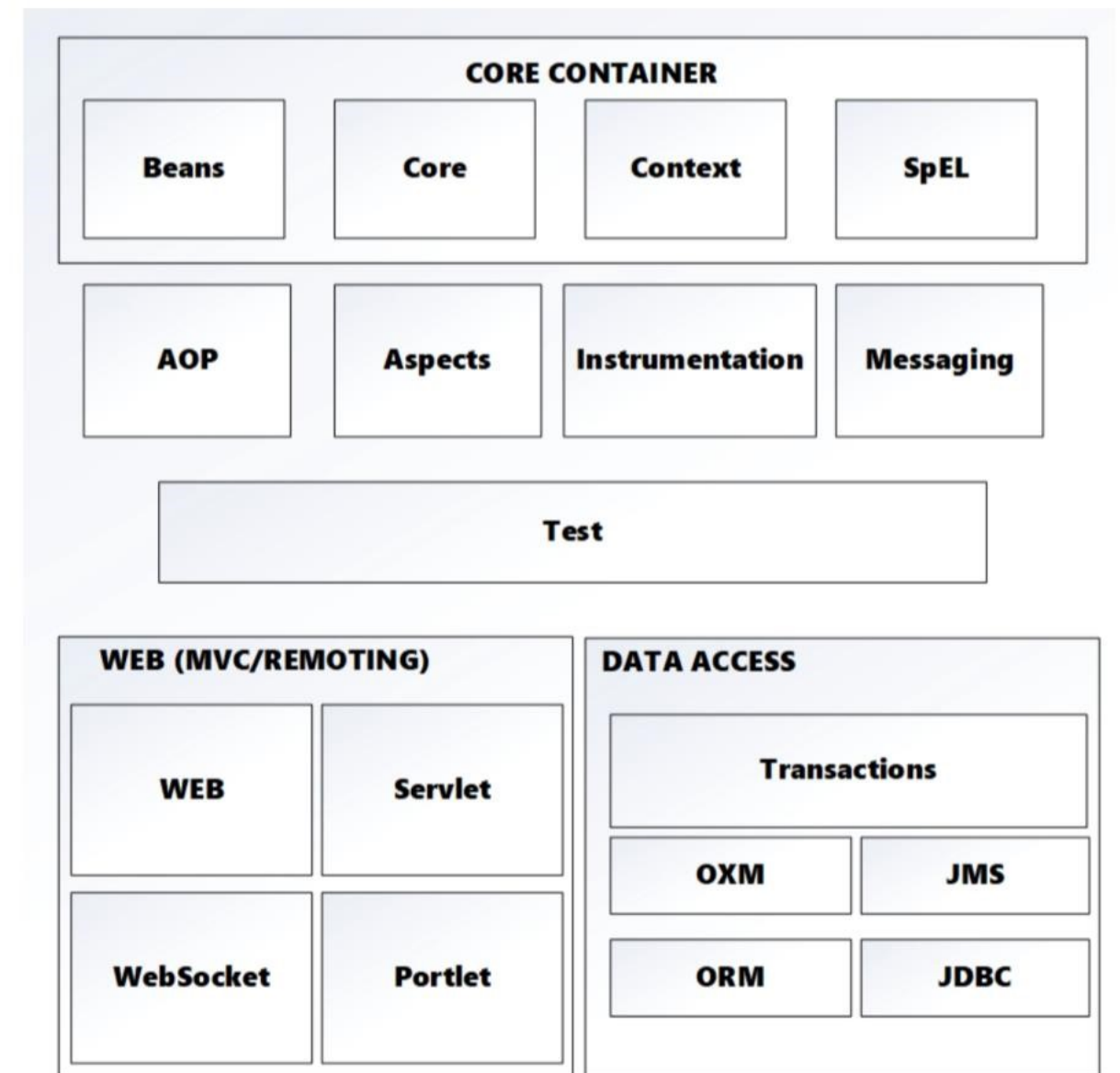


TESTIRANJE SOFTVERA - VEŽBE 09

TESTIRANJE SPRING APLIKACIJA

SPRING FRAMEWORK

- ▶ Spring je jedan od najpopularnijih okvira za izgradnju java EE aplikacija
- ▶ Osnovni koncepti na kojima počiva popularnost Spring radnog okvira su:
 - ▶ Inverzija kontrole (Inversion of Control IoC)
 - ▶ Injekcija zavisnosti (Dependency Injection)
 - ▶ Aspektno orijentisano programiranje (Aspect oriented programming)



MAVEN DEPENDENCIES

- ▶ *spring-boot-starter-test* je osnovni dependency koji sadrži sve neophodne elemente za testiranje Spring aplikacije

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

- ▶ Sve zavisnosti čiji je scope **test** će prilikom deploymenta aplikacije biti zanemarene (takve zavisnosti su prisutne u projektu samo prilikom faze razvoja i testiranja Maven aplikacije)
- ▶ pom.xml biblioteke *spring-boot-starter-test* uključuje JUnit 4 kao tranzitivnu zavisnost, da bismo izbegli koliziju u slučaju da želimo da testiramo koristeći JUnit 5, ovu predefinisanu zavisnost možemo isključiti

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

TESTIRANJE SPRING BOOT APLIKACIJE

- **@SpringBootTest** anotacija

- Kreira `ApplicationContext` koji se koristi za testiranje i po defaultu ne startuje server
- Dodavanjem `webEnvironment` atributa definišemo ponašanje testova:
 - MOCK (default)
 - učitava se `ApplicationContext` i mokuje se web okruženje
 - embedded servisi nisu startovani
 - RANDOM_PORT
 - učitava `WebServerApplicationContext` i omogućava stvarno web okruženje
 - embedded servisi su startovani i slušaju na random portu
 - korisno za izbegavanje konflikata prilikom izvršavanja testova
 - ukoliko želimo da znamo na kom portu je pokrenuta aplikacija možemo iskoristiti `@LocalServerPort`
 - DEFINED_PORT
 - isto ponašanje kao i za RANDOM_PORT, s tim da aplikacija sluša na unapred definisanom portu (`application.properties`) ili defaultnom 8080
 - NONE
 - učitava se `ApplicationContext`, ali ne postoji web okruženje

TESTIRANJE SPRING BOOT APLIKACIJE

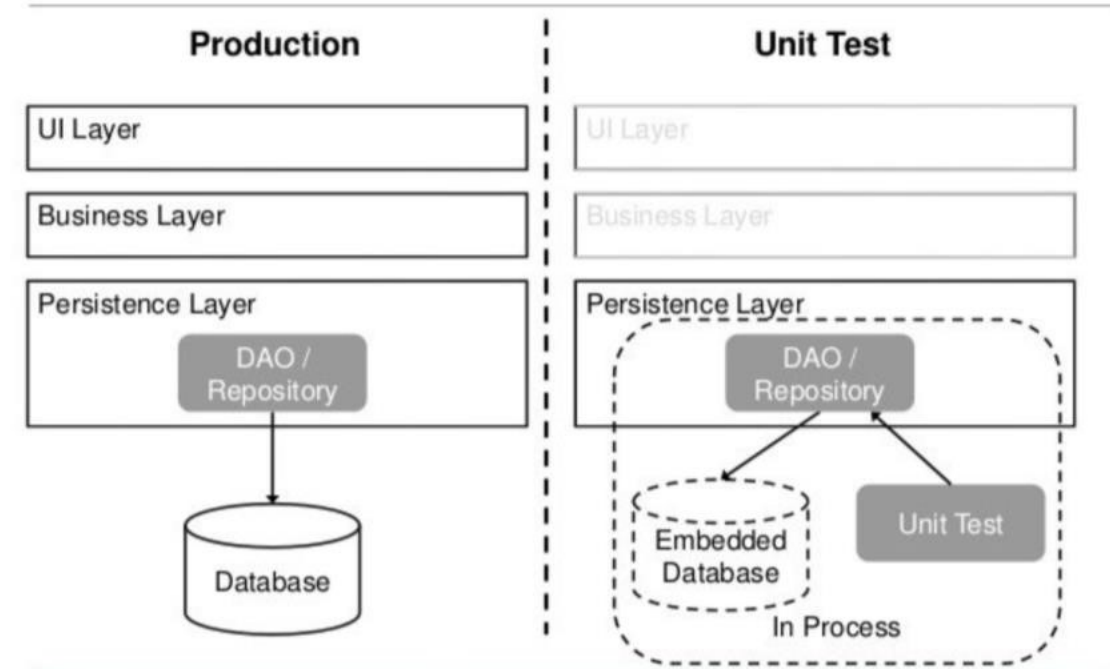
- ▶ **@SpringBootTest** omogućava kreiranje konteksta veoma sličnom onom koji ćemo imati u produkciji
- ▶ Koristi se za integracione testove, kada je potrebno prolaziti kroz sve slojeve aplikacije
- ▶ Osnovna mana ovog pristupa jeste njegova brzina, te stoga ovu anotaciju, u zavisnosti od sloja Spring aplikacije koju testiramo, možemo zameniti nekom od sledećih:
 - ▶ `@DataJpaTest` - za testiranje Spring Data Jpa repozitorijuma
 - ▶ `@WebMvcTest` - za testiranje REST sloja

TESTIRANJE DAO SLOJA

TESTIRANJE DAO SLOJA

- ▶ Testiranje DAO sloja zahteva pristup bazi podataka, međutim testiranja nad development bazom najčešće nisu dobra ideja jer sam proces testiranja može oštetiti ili narušiti validnost podatka
- ▶ Rešenje ovog problema može biti korišćenje testne baze podataka
- ▶ Ovaj pristup zahteva kreiranje baze sa istom konfiguracijom kao i u fazama razvoja i produkcije i override onih parametara koji treba da budu drugačiji specifično za fazu testiranja
- ▶ Iako je ovo najčešća praksa, često se i za fazu testiranja koriste baze IM tipa (in-memory)

Unit Testing Your Persistence Layer



TESTIRANJE DAO SLOJA

- ▶ Potrebno je prvo uvesti zavisnosti za H2 in-memory bazu podataka

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.4.200</version>
  <scope>test</scope>
</dependency>
```

- ▶ Zatim je potrebno konfigurisati testnu bazu i kreirati test profil (application-test.properties)

```
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
```

- ▶ Konkretni konfiguracioni profil je kasnije moguće aktivirati anotiranjem klase sa
@ActiveProfiles("test") ili sa
@TestPropertySource("classpath:application-test.properties")

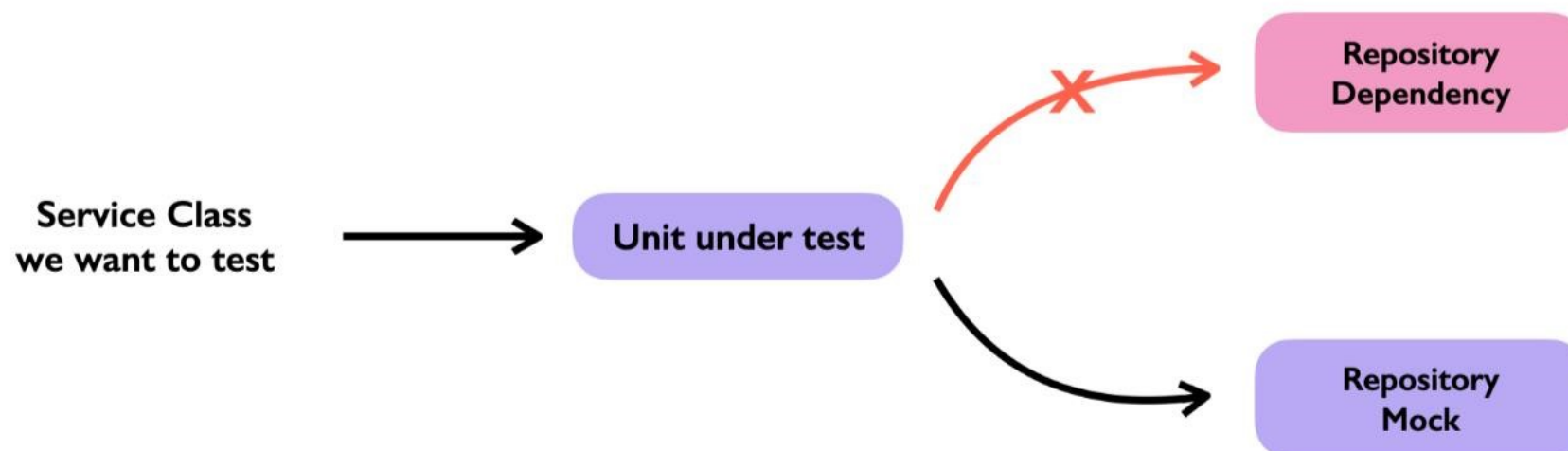
TESTIRANJE DAO SLOJA

- ▶ `@DataJpaTest` anotacija služi za testiranje JPA sloja, tako što omogućava:
 - ▶ Konfigurisanje H2, in-memory baze podataka
 - ▶ Auto-konfiguraciju Hibernate-a, Spring Data i DataSource-a
 - ▶ Izvođenje `@EntityScan`-a
 - ▶ SQL logging
 - ▶ Po default-u JPA testovi su transakcioni i rade roll-back nakon završetka svakog testa
- ▶ `@DataJpaTest` neće kreirati ceo kontekst Spring Boot aplikacije, već samo deo neophodan za inicijalizaciju repozitorijuma
- ▶ `@DataJpaTest` injektuje `TestEntityManager` bean
 - ▶ Alternativa standardnom `EntityManager`-u

TESTIRANJE SERVISNOG SLOJA

TESTIRANJE SERVISNOG SLOJA – JEDINIČNI TESTOVI

- ▶ Servisni sloj Spring aplikacije zavisi od Repository sloja
- ▶ Kako bi se omogućilo kreiranje unit testa neophodno je zanemariti postojeće zavisnosti (persistence sloj)
- ▶ Rešenje: mockovati repository sloj



TESTIRANJE SERVISNOG SLOJA – JEDINIČNI TESTOVI

- ▶ `@Mock`
 - ▶ Deo Mockito framework-a
 - ▶ Za injektovanje lažnih objekata, potrebno je test klasu anotirati sa `@InjectMocks`
- ▶ `@MockBean`
 - ▶ Deo Spring Boot framework-a (wrapper oko Mockito framework-a)
 - ▶ Kreira mock objekat unutar `Spring ApplicationContext`-a
 - ▶ Ukoliko bean definisan anotacijom postoji u `ApplicationContext`-u, dolazi do njegove zamene mock objektom, u suprotnom se dodaje novi mock objekat
 - ▶ Mockovani beanovi se automatski resetuju nakon izvršenja svakog testa

TESTIRANJE REST SLOJA

TESTIRANJE REST SLOJA – JEDINIČNI TESTOVI

- ▶ Rest sloj Spring aplikacije zavisi od servisnog sloja
- ▶ Kako bi se omogućilo kreiranje unit testa neophodno je zanemariti postojeće zavisnosti (persistence sloj)
- ▶ Rešenje: mokovati servisni sloj
- ▶ `@WebMvcTest`
 - ▶ Koristi se umesto `@SpringBootTest` anotacije
 - ▶ Ne instancira se ceo `ApplicationContext`, već samo web layer
 - ▶ Moguće je navesti kontrolere za koje želimo da budu instancirani `@WebMvcTest(NazivKontrolera.class)`
 - ▶ Ukoliko je potrebno instancirati ceo `ApplicationContext` koristiti anotacije `@SpringBootTest` i `@AutoConfigureMockMvc`
- ▶ `@MockMvc`
 - ▶ Pokretanje testova bez startovanja servera
 - ▶ Omogućava SpringMVC testiranje

TESTIRANJE REST SLOJA – JEDINIČNI TESTOVI

- ▶ MockMvc

- ▶ **perform()**

- ▶ Izvršava get metodu nad odgovarajućom rutom i vraća rezultat u okviru **ResultActions** objekta nad kojim se rade dalje asertacije

- ▶ **andDo(print())**

- ▶ Ispisuje zahtev i odgovor

- ▶ **andExpect()**

- ▶ Očekuje prosleđeni parametar, koristi se za očekivanje odgovarajućih statusnih kodova, kao i tela samih odgovora

- ▶ **andReturn()**

- ▶ Vraća MvcResult objekat

```
@Test
public void givenGreetURI_whenMockMVC_thenVerifyResponse() {
    MvcResult mvcResult = this.mockMvc.perform(get("/greet"))
        .andDo(print()).andExpect(status().isOk())
        .andExpect(jsonPath("$.message").value("Hello World!!!"))
        .andReturn();

    Assert.assertEquals("application/json;charset=UTF-8",
        mvcResult.getResponse().getContentType());
}
```

TESTIRANJE REST SLOJA – JEDINIČNI TESTOVI

```
@Test
public void givenGreetURIWithPathVariable_whenMockMVC_thenResponseOK() {
    this.mockMvc
        .perform(get("/greetWithPathVariable/{name}", "John"))
        .andExpect(status().isOk())
        .andExpect(content().contentType("application/json;charset=UTF-8"))
        .andExpect(jsonPath("$.message").value("Hello World John!!!"));
}
```

TESTIRANJE REST SLOJA – JEDINIČNI TESTOVI

```
@Test
public void givenGreetURIWithQueryParameter_whenMockMVC_thenResponseOK() {
    this.mockMvc.perform(get("/greetWithQueryVariable")
        .param("name", "John Doe"))
        .andDo(print())
        .andExpect(status().isOk())
        .andExpect(content().contentType("application/json;charset=UTF-8"))
        .andExpect(jsonPath("$.message").value("Hello World John Doe!!!"));
}
```

TESTIRANJE REST SLOJA – JEDINIČNI TESTOVI

```
@Value
public class UserResource {

    @NotNull
    private final String name;

    @NotNull
    private final String email;

}
```

```
@Test
public void whenNullValue_thenReturns400() throws Exception {
    UserResource user = new UserResource(null, "zaphod@galaxy.net");

    this.mockMvc.perform(post("/forums/{forumId}/register", 42L)
        ...
        .content(objectMapper.writeValueAsString(user)))
        .andExpect(status().isBadRequest());
}
```