DefineX Java Spring Boot Bootcamp
HW-3

İlker KUŞ

İÇİNDEKİLER

1) Aşağıdaki kavramları örneklerle açıklayın. (10 PUAN)

1) **Unit Test:**

Birim testi, yazılım programlamasında bir tasarım ve geliştirme yöntemidir. Bu yöntemde yazılımcı yazılım kodunu oluşturan birimlerin kullanıma hazır olduğuna iknâ olur. Birim, bir bilgisayar uygulamasında test edilebilecek en küçük bölüme denir.

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculatorTest {

    @Test
    public void testAddition() {
        Calculator calculator = new Calculator();
        int result = calculator.add(3, 5);
        assertEquals(8, result);
    }
}
```

2) **Integration Test:**

Entegrasyon testleri, farklı yazılım bileşenlerinin bir ortamda bir araya getirildiğinde sorunsuz bir şekilde çalışıp çalışmadığını doğrulayan bir test türüdür. Bu testler, uygulamanın gerçek dünya senaryolarında nasıl davrandığını anlamamıza ve bileşenlerin uyum içinde olup olmadığını doğrulamamıza yardımcı olur.

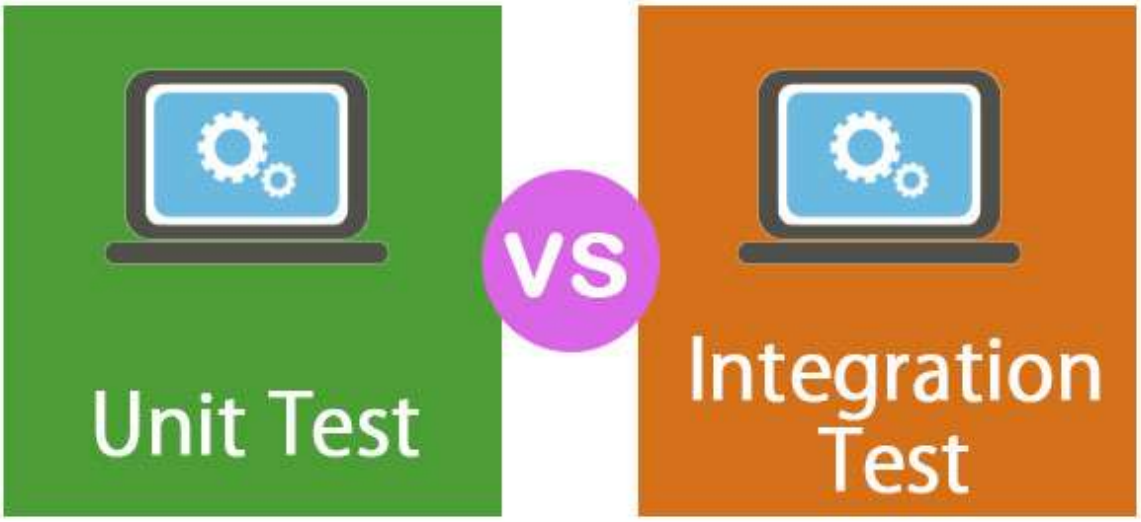
Integration testler genellikle Spring Boot'un sağladığı **@SpringBootTest** ve **@AutoConfigureMockMvc** gibi annotasyonlarla yazılabilir.

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testGetUserById() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/api/users/{id}", 1))
               .andExpect(MockMvcResultMatchers.status().isOk())
               .andExpect(MockMvcResultMatchers.jsonPath("$.id").value(1));
    }
}
```



Peki, neden test yazmalıyız ?

1. **Kalite Güvencesi**: Testler, kodunuzun beklenen şekilde çalıştığını doğrulamanıza yardımcı olur. Böylece yazılımınızın kalitesini artırarak hataları ve sorunları erken aşamalarda tespit edebilirsiniz.
2. **Refactoring Desteği**: Kodunuzu yeniden düzenleme (refactoring) ihtiyacı duyduğunuzda, mevcut testlerinizin varlığı, bu değişikliklerin kodunuzun işlevselliğini etkilemediğinden emin olmanıza yardımcı olur.
3. **Dökümantasyon ve Anlaşılabilirlik**: İyi yazılmış testler, kodunuzun nasıl kullanılması gerektiği hakkında açık bir belge sağlar. Ayrıca, kodun ne yapması gerektiği hakkında bir anlayış sağlar.
4. **Sürdürülebilirlik**: Testler, yazılımınızın daha uzun süreli sürdürülebilirliğini sağlar. Özellikle yeni eklenen özellikler veya kodun güncellenmesi gerektiğinde, testler değişikliklerin beklenen sonuçları ürettiğini doğrulamanıza yardımcı olur.

KAYNAKÇA

- https://betulsahinn.medium.com/spring-boot-ile-unit-test-yazmak-f1e4fc1f3df
- https://medium.com/devopsturkiye/unit-test-mi-integration-test-mi-34ddea054696
- https://mbahardogan.medium.com/the-integration-test-with-spring-boot-4893d8151278