Test-Konzept

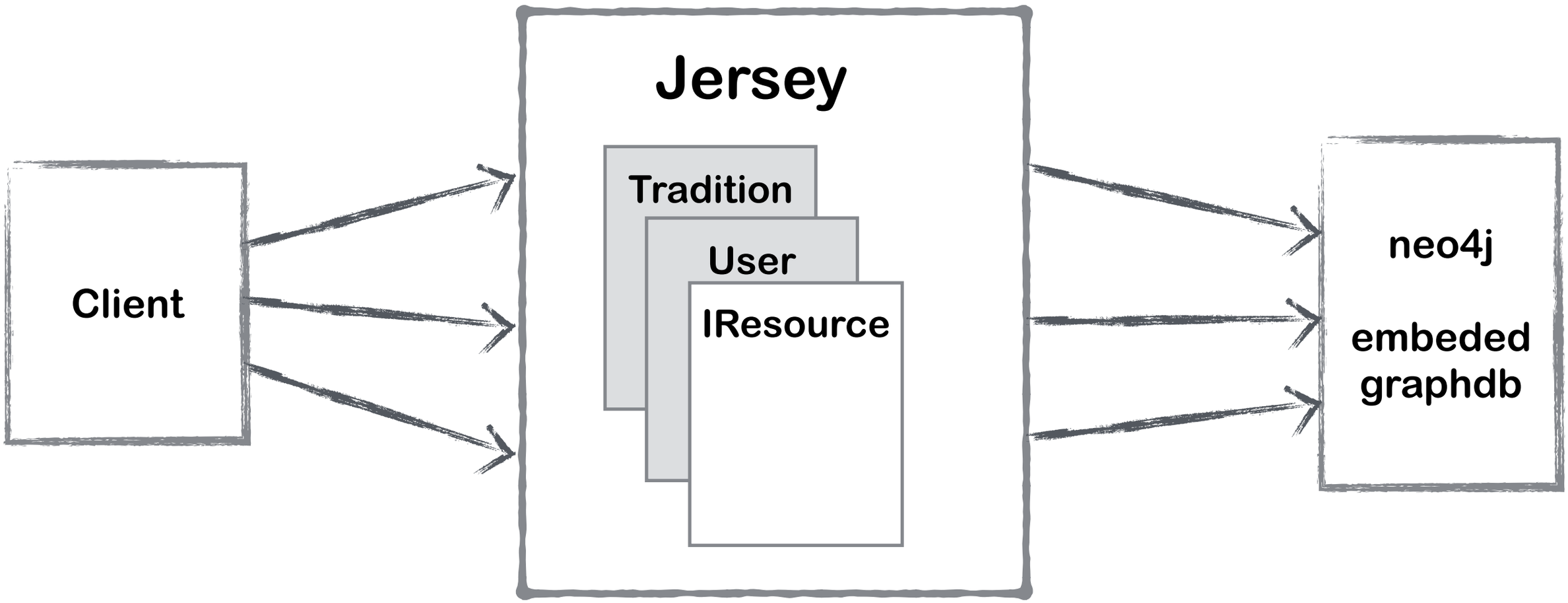
Konzept

- Unit Tests für testdriven Development
  - So viele Tests wie zum Entwickeln nötig sind

- Integration Tests zum testen von Userstories
  - Alle Userstories werden getestet
  - Für die Qualitätssicherung relevant

- Getrennte Ordner für Unit und Integration Tests

Produktiv System: Überblick

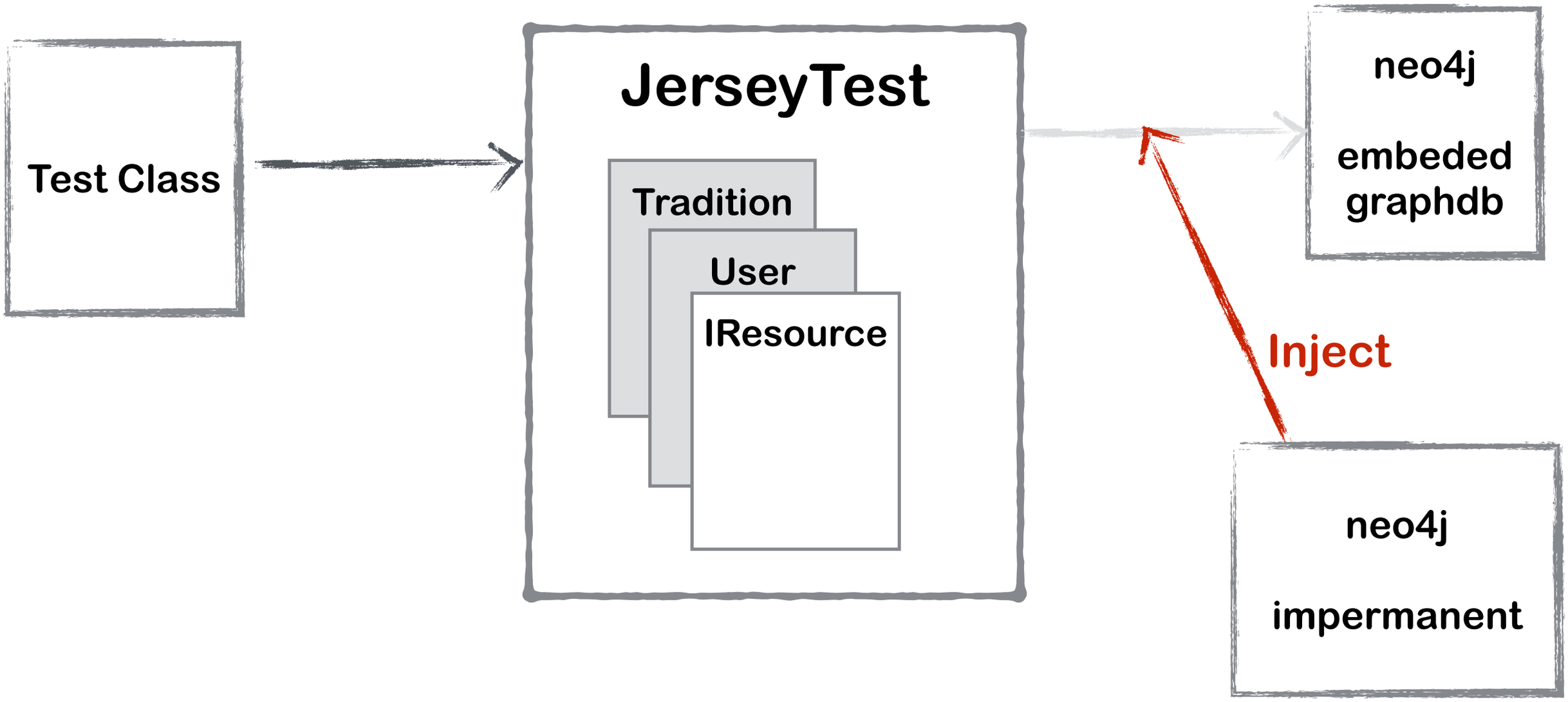


- Für jeden REST call wird eine Resource instanziert
- Jede Instanz hat ihren eigenen DB Service

Test System: Anforderungen

- Unabhängige Datenbank (impermanent)
- Unit Tests
- Integration Tests
- Kein Einfluss auf die Programmarchitektur

Integration Test: Übersicht



- Für jeden REST call wird die selbe Instanz verwendet
- Der Datenbank Pfad wird mit Injection überschrieben

Inject Database Service

```
@RunWith(MockitoJUnitRunner.class)
public class UserUnitTest {

    @Mock
    protected GraphDatabaseFactory mockDbFactory = new GraphDatabaseFactory();

    @Spy
    protected GraphDatabaseService mockDbService =
            new TestGraphDatabaseFactory().newImpermanentDatabase();

    @InjectMocks
    private User userResource;

    ……

    Mockito.when(mockDbFactory.newEmbeddedDatabase(Matchers.anyString()))
            .thenReturn(mockDbService);

    Mockito.doNothing().when(mockDbService).shutdown();
```

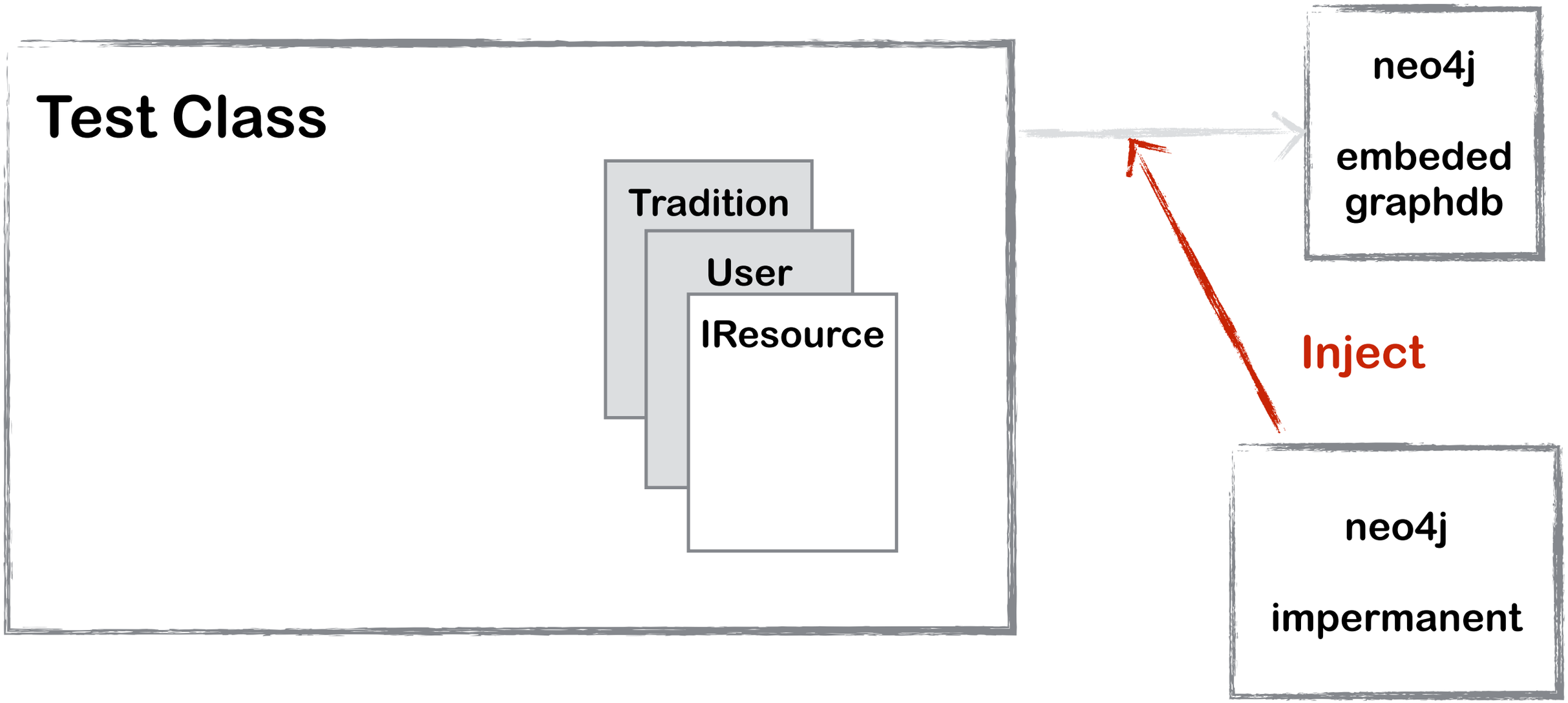
Run JerseyTest

```
private JerseyTest jerseyTest;

……

jerseyTest = JerseyTestServerFactory.newJerseyTestServer()
              .addResource(userResource).create();
jerseyTest.setUp();
```

Unit Test: Übersicht



- Resource wird in der Test Klasse instanziert
- Der Datenbank Pfad wird mit Injection überschrieben