

# Übung Meteo Service Unit Test Spring Boot

Diese Übung zeigt die meisten verwendeten Unit Test Varianten von Spring Boot. So testen wir durch diese Variantenstudie den gleichen Use Case teilweise doppelt. In der Praxis wählt man die richtige Variante nach den aktuellen Gegebenheiten und Anforderungen. Programmieren Sie mindestens einen der nachfolgend beschriebenen Unit Tests. Wir empfehlen Ihnen mit dem Test Setup Daten fix zu definieren und zu laden ohne die Daten aus der Datei data.sql zu verwenden. Erstellen Sie hierzu die Datei application-unittest.properties und schalten Sie das Laden der data.sql aus. Das folgende Listing zeigt eine mögliche unittest Profile Properties Datei

```
application-unittest.properties: spring.sql.init.mode=never; spring.datasource.url=jdbc:h2:mem:
testdb; spring.datasource.driver-class-name=org.h2.Driver; spring.jpa.properties.hibernate.
dialect=org.hibernate.dialect.H2Dialect; spring.jpa.database-platform=org.hibernate.dialect.H2D
ialectDie Datei sollte im src/test/resources Verzeichnis gespeichert werden. Einige Tests arbeiten mit
Mock Daten und beziehen keine Daten aus der Datenbank. Die anderen Tests arbeiten mit Daten
aus der H2 Datenbank. Das folgende Beispiel zeigt eine mögliche Setup Klasse für die
Bereitstellung der Testdaten. Die nachfolgenden Vorlagen für die Unit Tests arbeiten mit dieser
Klasse: package ch.std.meteo.test.setup; import java.util.Arrays; import
java.util.stream.Collectors; import java.util.stream.DoubleStream; import
javax.persistence.EntityManager; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Component; import
ch.std.meteo.jpa.MeteoData; import
ch.std.meteo.repositories.MeteoDataRepository; @Component public class
MeteoDataTestSetup { public static Double MIN = 4.5; public static Double MAX =
16.1; public static Double[] TEMPERATURES = { MIN, 6.2, 10.3, 14.2, MAX, 13.3, 9.8,
7.2 }; public static Double MED = 0.0; static { MED =
Arrays.stream(TEMPERATURES).collect(Collectors.averagingDouble(x -> x)); }
private MeteoDataRepository meteoDataRepository; private EntityManager
entityManager; @Autowired public MeteoDataTestSetup(MeteoDataRepository
meteoDataRepository, EntityManager entityManager) { this.meteoDataRepository =
meteoDataRepository; this.entityManager = entityManager; } public void
setup() { for (Double t : TEMPERATURES) { MeteoData meteoData = new
MeteoData(t); this.meteoDataRepository.save(meteoData); }
this.entityManager.clear(); } public void teardown() {
this.meteoDataRepository.deleteAllInBatch(); } Mit dem Aufruf der Methode
entityManager.clear() löschen wir die JPA Session und entkoppeln den Setup von den Unit Tests.
```

## JPA Repository Unit Test (@DataJpaTest)

Das folgende Listing zeigt einen möglichen @DataJpaTest:package  
ch.std.meteo.test.repositories;...;@ExtendWith(SpringExtension.class);@DataJpaTest  
test;@ComponentScan(basePackages =  
{&#34;ch.std.meteo.test&#34;})&#xA;@ActiveProfiles(&#34;unittest&#34;)&#xA;public class  
MeteoDataRepositoryJPATest {&#xA;&#xA; @Autowired&#xA; private MeteoDataRepository  
meteoDataRepository;&#xA; @Autowired&#xA; private MeteoDataTestSetup  
meteoDataTestSetup;&#xA;&#xA; @BeforeEach&#xA; public void setup() {&#xA;  
this.meteoDataTestSetup.setup();&#xA; }&#xA;&#xA; @AfterEach&#xA; public void teardown()  
{&#xA; this.meteoDataTestSetup.teardown();&#xA; }&#xA;&#xA; @Test&#xA; public void  
testMeteoDataFindAll() {&#xA; // TODO&#xA; }&#xA;&#xA; }Arbeiten Sie mit der Vorlage und  
programmieren Sie die TODO Sequenz aus oder beginnen Sie komplett neu mit einer eigenen  
Testklasse.

## DTO Mapping Unit Test

Bei diesem reinen Java Unit Test testen wir das korrekte Mapping zwischen Instanzen der Klasse `ch.std.meteo.ipa.MeteoData` -&qt; `ch.std.meteo.dto.MeteoDataDTO`.

## Meteo Service Unit Test

Bei diesem Test testen wir die `ch.std.meteo.service.MeteoService` Klasse. Wir arbeiten mit dem

Kontext der H2 Datenbank und verwenden damit die @DataJpaTest Umgebung:package ch.std.meteo.test.service;...&#xA;@ExtendWith(SpringExtension.class)&#xA;@DataJpaTest&#xA;@ComponentScan(basePackages = {&#34;ch.std.meteo.test&#34;}, basePackageClasses = {MeteoService.class})&#xA;@ActiveProfiles(&#34;unittest&#34;)&#xA;public class MeteoServiceTest {&#xA; @Autowired&#xA; private MeteoDataTestSetup meteoDataTestSetup;&#xA;&#xA; @Autowired&#xA; private MeteoService meteoService;&#xA;&#xA; @BeforeEach&#xA; public void setup() {&#xA; this.meteoDataTestSetup.setup();&#xA; }&#xA;&#xA; @AfterEach&#xA; public void teardown() {&#xA; this.meteoDataTestSetup.teardown();&#xA; }&#xA;&#xA;&#xA; @Test&#xA; public void test\_getMeteoDaten() {&#xA; // TODO&#xA; }&#xA; &#xA; @Test&#xA; public void test\_getMeteoMetrics() {&#xA; // TODO&#xA; }&#xA; &#xA; }Arbeiten Sie mit der Vorlage und programmieren Sie die TODO Sequenzen aus oder beginnen Sie komplett neu mit einer eigenen Testklasse.

## Meteo Service Unit Test using Repository MockBean

Bei diesem Test testen wir die ch.std.meteo.service.MeteoService Klasse. Wir arbeiten aber alternativ mit einer Mock Umgebung anstelle der H2 Datenbank:&#xA;package ch.std.meteo.test.service;...&#xA;@ExtendWith(SpringExtension.class)&#xA;@DataJpaTest&#xA;@ComponentScan(basePackages = {&#34;ch.std.meteo.test&#34;}, basePackageClasses = {MeteoService.class})&#xA;@ActiveProfiles(&#34;unittest&#34;)&#xA;public class MeteoServiceTestUsingMockBean {&#xA; &#xA; @MockBean&#xA; private MeteoDataRepository meteoDataRepository;&#xA;&#xA; @Autowired&#xA; private MeteoService meteoService;&#xA;&#xA; @BeforeEach&#xA; public void setup() {&#xA; when(meteoDataRepository.findAll()).thenReturn(Arrays.stream(MeteoDataTestSetup.TEMPERATURES).map(t -&gt; new MeteoData(t)).collect(Collectors.toList()));&#xA; }&#xA;&#xA; @Test&#xA; public void test\_getMeteoDaten() {&#xA; // TODO&#xA; }&#xA; &#xA; @Test&#xA; public void test\_getMeteoMetrics() {&#xA; MeteoMetrics meteoMetrics = this.meteoService.getMeteoMetrics();&#xA; // TODO&#xA; }&#xA; &#xA; }Arbeiten Sie mit der Vorlage und programmieren Sie die TODO Sequenzen aus oder beginnen Sie komplett neu mit einer eigenen Testklasse.

## Meteo Service Integration Unit Test (@SpringBootTest)

Bei diesem Integratoinstest testen wir den REST Endpoint der Klasse ch.std.meteo.rest.MeteoController. Wir verwenden die @SpringBootTest Annotation mit dem Random Port. Als Test Client arbeitet die Vorlage mit der TestRestTemplate-Klasse:&#xA;package ch.std.meteo.test.integration;...&#xA;@SpringBootTest(webEnvironment = WebEnvironment.RANDOM\_PORT)&#xA;@ComponentScan(basePackages = {&#34;ch.std.meteo&#34;})&#xA;@ActiveProfiles(&#34;unittest&#34;)&#xA;public class MeteoServiceIntegrationTest {&#xA; &#xA; @Autowired&#xA; private TestRestTemplate restTemplate;&#xA;&#xA; @Autowired&#xA; private MeteoDataTestSetup meteoDataTestSetup;&#xA;&#xA; @BeforeEach&#xA; public void setup() {&#xA; this.meteoDataTestSetup.setup();&#xA; }&#xA;&#xA; @AfterEach&#xA; public void teardown() {&#xA; this.meteoDataTestSetup.teardown();&#xA; }&#xA;&#xA; @Test&#xA; public void test\_MeteoController\_getMeteoDaten() throws Exception {&#xA; // TODO&#xA; }&#xA; &#xA; @Test&#xA; public void test\_MeteoController\_getMeteoMetrics() {&#xA; // TODO&#xA; }&#xA; &#xA; }Arbeiten Sie mit der Vorlage und programmieren Sie die TODO Sequenzen aus oder beginnen Sie komplett neu mit einer eigenen Testklasse.

## Meteo Service WebMvc Unit Test (@WebMvc)

Abschliessen testen wir den REST Endpoint MeteoController mit der gemockten Umgebung @WebMvc, und damit wird nur der Controller geladen aber keine weiteren Beans oder Components. Die ch.std.meteo.service.MeteoService Klassen mocken wir, siehe Vorlage:&#xA;package ch.std.meteo.test.webmvc;...&#xA;@ExtendWith(SpringExtension.class)&#xA;@WebMvcTest(MeteoController.class)&#xA;public class MeteoServiceWebLayerTest {&#xA; @Autowired&#xA; private MockMvc mvc;&#xA;&#xA; @MockBean&#xA; private MeteoService meteoService;&#xA;&#xA; @BeforeEach&#xA; public void setup() {&#xA; List meteoDataList = Arrays.stream(MeteoDataTestSetup.TEMPERATURES).map(t -&gt; new

```
MeteoData(t)).collect(Collectors.toList());&#xA;
when(meteoService.getMeteoDaten()).thenReturn(meteoDataList);&#xA; MeteoMetrics
meteoMetrics = new MeteoMetrics(meteoDataList);&#xA;
when(meteoService.getMeteoMetrics()).thenReturn(meteoMetrics);&#xA; }&#xA;&#xA; @Test&#xA;
public void test_MeteoController_getMeteoDaten() throws Exception {&#xA; // TODO&#xA;&#xA;
}&#xA; &#xA; @Test&#xA; public void test_getMeteoMetrics() throws Exception {&#xA; //
TODO&#xA; }&#xA;&#xA;}Arbeiten Sie mit der Vorlage und programmieren Sie die TODO
Sequenzen aus oder beginnen Sie komplett neu mit einer eigenen Testklasse.
```

## Lösung

Eine mögliche Lösung finden Sie hier

### Kontakt

Simtech AG  
Finkenweg 23  
3110 Münsingen  
Schweiz

### Impressum

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.  
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsausschluss.  
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

### Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

[https://www.simtech-ag.ch/test\\_meteocontroller\\_getmeteometrics](https://www.simtech-ag.ch/test_meteocontroller_getmeteometrics)