secunet

The PIT

Partielle Integrationstests in Spring Boot Anwendungen mit Spring Data JPA

Inhaltsverzeichnis

01

Tests

Tests – Zielbild

Tests – Zielbild

- schnell in der Ausführung

Tests – Zielbild

- schnell in der Ausführung
- schnelle Erstellung

Tests – Zielbild

- schnell in der Ausführung
- schnelle Erstellung
- einfache Wartung

Tests – Zielbild

- schnell in der Ausführung
- schnelle Erstellung
- einfache Wartung
- hohe Aussagekraft

Tests – Zielbild

- schnell in der Ausführung
- schnelle Erstellung
- einfache Wartung
- hohe Aussagekraft
- leicht verständlich

unit Tests

unit Tests

- schnell in Ausführung

unit Tests

- schnell in Ausführung
- Mocking: Erstellung teilweise langsam, Wartung aufwändig, Verständlichkeit gering

unit Tests

- schnell in Ausführung
- Mocking: Erstellung teilweise langsam, Wartung aufwändig, Verständlichkeit gering
- testen nicht, ob die Integration von Code korrekt ist

unit Tests

- schnell in Ausführung
- Mocking: Erstellung teilweise langsam, Wartung aufwändig, Verständlichkeit gering
- testen nicht, ob die Integration von Code korrekt ist
- verstecken insb. durch Mocking Fehler versehentlich

unit Tests

- schnell in Ausführung
- Mocking: Erstellung teilweise langsam, Wartung aufwändig, Verständlichkeit gering
- testen nicht, ob die Integration von Code korrekt ist
- verstecken insb. durch Mocking Fehler versehentlich

Beispiel

- 100% branch coverage, aber Persistieren von Objekten mit Fremdschlüsselbeziehung in falscher Reihenfolge

E2E-Tests

E2E-Tests

- sehr langsam

E2E-Tests

- sehr langsam
- aufwändig in Erstellung und Pflege, weil Code geschrieben werden muss, der dazu dient den zu testenden Code überhaupt zu erreichen

E2E-Tests

- sehr langsam
- aufwändig in Erstellung und Pflege, weil Code geschrieben werden muss, der dazu dient den zu testenden Code überhaupt zu erreichen

Beispiel

- Setup in datengetriebenen Anwendungen

Integrationstests mit SpringBoot mit Slices

Tests mit `@SpringBootTest`, `@DataJpaTest`, `@WebMvcTest`

Integrationstests mit SpringBoot mit Slices

Tests mit `@SpringBootTest`, `@DataJpaTest`, `@WebMvcTest`

- langsam durch Erstellung des Spring-Kontexts

Integrationstests mit SpringBoot mit Slices

Tests mit `@SpringBootTest`, `@DataJpaTest`, `@WebMvcTest`

- langsam durch Erstellung des Spring-Kontexts
- langsam durch Einspielen von Migrationsskripten für Datenbank

Und nun

Und nun

Ebene zwischen unit Tests und Spring Integrationstests

Und nun

Ebene zwischen unit Tests und Spring Integrationstests

- partielle Integrationstests ohne Spring Kontext

02

Unsere Bausteine

Sequenzen als Generator für Testwerte

Sequenzen als Generator für Testwerte

- simulieren DB-Sequenzen

Sequenzen als Generator für Testwerte

- simulieren DB-Sequenzen
- eignen sich gut für beliebige Initialisierungen von Datenobjekten

Sequenzen als Generator für Testwerte

- simulieren DB-Sequenzen
- eignen sich gut für beliebige Initialisierungen von Datenobjekten

Initialisierung ohne Sequenz

Sequenzen als Generator für Testwerte

- simulieren DB-Sequenzen
- eignen sich gut für beliebige Initialisierungen von Datenobjekten

Initialisierung ohne Sequenz

```
int id = 1, version = 2;
var name = "name3";
new DomainModel(id, version, name, 4);
```

Sequenzen als Generator für Testwerte

- simulieren DB-Sequenzen
- eignen sich gut für beliebige Initialisierungen von Datenobjekten

Initialisierung ohne Sequenz

```
int id = 1, version = 2;
var name = "name3";
new DomainModel(id, version, name, 4);
```

Initialisierung mit Sequenz

Sequenzen als Generator für Testwerte

- simulieren DB-Sequenzen
- eignen sich gut für beliebige Initialisierungen von Datenobjekten

Initialisierung ohne Sequenz

```
int id = 1, version = 2;
var name = "name3";
new DomainModel(id, version, name, 4);
```

Initialisierung mit Sequenz

```
int id = s.next(), version = s.next();
var name = s.nextString("name");
new DomainModel(id, version, name, s.nextLong());
```

Builder für Datenklassen

Builder für Datenklassen

- Builder für Datenklassen in Produktivcode oft Overkill/unpassend

Builder für Datenklassen

- Builder für Datenklassen in Produktivcode oft Overkill/unpassend
- in Tests oft sehr flexible/einfache Initialisierung von Datenklassen ausreichend und vorteilhaft

Builder für Datenklassen

- Builder für Datenklassen in Produktivcode oft Overkill/unpassend
- in Tests oft sehr flexible/einfache Initialisierung von Datenklassen ausreichend und vorteilhaft
- Builder dann in korrespondierender Testklasse

Beispiel Builder in Testklasse

```
1   public class DomainModelTest {
2     public static final class Builder {
3       private final Sequence s;
4       private Integer id;
5       private Integer version;
6       private String name;
7       private Long value;
8       ...
9       public DomainModel build() {
10        if (id == null) { id = s.next(); }
11        ...
12        return new DomainModel(...);
13      }
```

Transitive Builder für Datenklassen

Transitive Builder für Datenklassen

- rufen transitiv die Builder referenzierter Objekte auf

Transitive Builder für Datenklassen

- rufen transitiv die Builder referenzierter Objekte auf
- sorgen für konsistente Datenlage

Transitive Builder für Datenklassen

- rufen transitiv die Builder referenzierter Objekte auf
- sorgen für konsistente Datenlage
- erfüllen Fremdschlüsselbeziehungen

Beispiel transitiver Builder

```
1   public class RepoModelTest {
2     public static final class Builder {
3       private final Sequence s;
4       private RefEntity refEntity;
5       private boolean shouldHaveRefEntity;
6       private RefEntityTest.Builder refEntityBuilder =
7           new RefEntityTest.Builder(s());
8       ...
9       public RepoModel build() {
10        if (shouldHaveRefEntity && refEntity == null) {
11          refEntity = refEntityBuilder.build();
12        }
13        ...
14        return new RepoModel(...,
15            shouldHaveRefEntity ? refEntity.id() : null);
16      }
```

Spezialfall @DataJpaTest

Transitive Builder in @DataJpaTest:

Spezialfall @DataJpaTest

Transitive Builder in @DataJpaTest:

- einfache Setups von Tests

Spezialfall @DataJpaTest

Transitive Builder in @DataJpaTest:

- einfache Setups von Tests
- kompakte und dadurch verständliche Tests

Spezialfall @DataJpaTest

Transitive Builder in @DataJpaTest:

- einfache Setups von Tests
- kompakte und dadurch verständliche Tests
- geht auch, wenn Fremdschlüsselbeziehungen in JPA Entity nur als numerischer Wert abgebildet sind

Spezialfall @DataJpaTest

Transitive Builder in @DataJpaTest:

- einfache Setups von Tests
- kompakte und dadurch verständliche Tests
- geht auch, wenn Fremdschlüsselbeziehungen in JPA Entity nur als numerischer Wert abgebildet sind
- auch für Nicht-@DataJpaTest, die ähnliche Anforderungen an Konsistenz der Daten stellen

Exkurs: Struktur Spring Boot Anwendungen

Exkurs: Struktur Spring Boot Anwendungen

- Objektgraph, in dem es für jede Komponente genau eine Instanz gibt

Exkurs: Struktur Spring Boot Anwendungen

- Objektgraph, in dem es für jede Komponente genau eine Instanz gibt
- Komponenten i.d.R. zustandslose Klassen für Verhalten

Exkurs: Struktur Spring Boot Anwendungen

- Objektgraph, in dem es für jede Komponente genau eine Instanz gibt
- Komponenten i.d.R. zustandslose Klassen für Verhalten
- Initialisierung benötigt Instanz/Implementierung für jedes Interface

Exkurs: Struktur Spring Boot Anwendungen

- Objektgraph, in dem es für jede Komponente genau eine Instanz gibt
- Komponenten i.d.R. zustandslose Klassen für Verhalten
- Initialisierung benötigt Instanz/Implementierung für jedes Interface

Verzögerte Initialisierung für Tests

Exkurs: Struktur Spring Boot Anwendungen

- Objektgraph, in dem es für jede Komponente genau eine Instanz gibt
- Komponenten i.d.R. zustandslose Klassen für Verhalten
- Initialisierung benötigt Instanz/Implementierung für jedes Interface

Verzögerte Initialisierung für Tests

- sehr vorteilhaft, um gezielt entsprechende Teile der Anwendung zu überschreiben

Spring Data JPA

Spring Data JPA

- im eigenen Quellcode nur Java `interface` für JPA repositories

Spring Data JPA

- im eigenen Quellcode nur Java `interface` für JPA repositories
- auch bei hexagonaler Architektur Implementierungen für Instanziierung benötigt

Fake-Implementierungen für JPA repositories

Fake-Implementierungen für JPA repositories

- zusätzlicher Aufwand

Fake-Implementierungen für JPA repositories

- zusätzlicher Aufwand
- kann unbemerkt von echter Implementierung abweichen

Fake-Implementierungen für JPA repositories

- zusätzlicher Aufwand
- kann unbemerkt von echter Implementierung abweichen
- als in memory DB sehr schnell

Fake-Implementierungen für JPA repositories

- zusätzlicher Aufwand
- kann unbemerkt von echter Implementierung abweichen
- als in memory DB sehr schnell
- bei Neuinitialisierung pro Test echte Unabhängigkeit der Tests

Fake-Implementierungen für JPA repositories

- zusätzlicher Aufwand
- kann unbemerkt von echter Implementierung abweichen
- als in memory DB sehr schnell
- bei Neuinitialisierung pro Test echte Unabhängigkeit der Tests
- verwendet unsere Sequenz für IDs

Hifsklassen

Hifsklassen

- Testinitializer für verzögerte Initialisierung

Hifsklassen

- Testinitializer für verzögerte Initialisierung
- Registry, die von einem Typ (Klasse/Interface) auf eine Instanz bzw. einen entsprechenden TestInitializer abbildet

Hifsklassen

- Testinitializer für verzögerte Initialisierung
- Registry, die von einem Typ (Klasse/Interface) auf eine Instanz bzw. einen entsprechenden TestInitializer abbildet
- Defaultimplementierung Testinitializer für jede Spring Komponente

03

Was haben wir erreicht

Setup für Tests

Setup für Tests

- einfach: Aufruf des entsprechenden Testinitializer in @BeforeEach

Setup für Tests

- einfach: Aufruf des entsprechenden Testinitializer in @BeforeEach
- vollständig: Vollständiger Objektgraph ab der initialisierten Klasse

Setup für Tests

- einfach: Aufruf des entsprechenden Testinitializer in @BeforeEach
- vollständig: Vollständiger Objektgraph ab der initialisierten Klasse
- modifizierbar: TestInitializer können gezielt überschrieben werden und werden konsistent verwendet

Setup für Tests

- einfach: Aufruf des entsprechenden Testinitializer in @BeforeEach
- vollständig: Vollständiger Objektgraph ab der initialisierten Klasse
- modifizierbar: TestInitializer können gezielt überschrieben werden und werden konsistent verwendet
- einschränkbar: Objektgraph kann durch spezielle TestInitializer gezielt gekappt werden

Verständlichkeit von Tests

Verständlichkeit von Tests

- Einfaches Setup der Datenlage

Verständlichkeit von Tests

- Einfaches Setup der Datenlage
- Kein Mockito.mock notwendig (aber möglich), was die Tests aufbläht

Verständlichkeit von Tests

- Einfaches Setup der Datenlage
- Kein Mockito.mock notwendig (aber möglich), was die Tests aufbläht
- stärkerer Fokus auf fachliche Aspekte

Wartbarkeit von Tests

Wartbarkeit von Tests

- Durch Entfall von Mocking wesentlich robuster bei fachlich irrelevanten Änderungen

Wartbarkeit von Tests

- Durch Entfall von Mocking wesentlich robuster bei fachlich irrelevanten Änderungen
- Tests wesentlich empfindlicher bei fachlichen Änderungen (weniger false positives)

Wartbarkeit von Tests

- Durch Entfall von Mocking wesentlich robuster bei fachlich irrelevanten Änderungen
- Tests wesentlich empfindlicher bei fachlichen Änderungen (weniger false positives)
- Seltener rein technisch bedingte Änderungen an Tests

Wartbarkeit von Tests

- Durch Entfall von Mocking wesentlich robuster bei fachlich irrelevanten Änderungen
- Tests wesentlich empfindlicher bei fachlichen Änderungen (weniger false positives)
- Seltener rein technisch bedingte Änderungen an Tests
- Bei Bedarf Änderungen einfacher/nur insoweit wie fachlich benötigt

Sonstiges

Sonstiges

- Tests extrem schnell

Sonstiges

- Tests extrem schnell
- Tests sehr flexibel, aber im Normfall sehr kompakt und verständlich

Sonstiges

- Tests extrem schnell
- Tests sehr flexibel, aber im Normfall sehr kompakt und verständlich
- Vererbung: Testklassen können ggfs. abgeleitet und dann direkt als @SpringBootTest verwendet werden

Sonstiges

- Tests extrem schnell
- Tests sehr flexibel, aber im Normfall sehr kompakt und verständlich
- Vererbung: Testklassen können ggfs. abgeleitet und dann direkt als @SpringBootTest verwendet werden
- Mocking immer noch möglich

Sonstiges

- Tests extrem schnell
- Tests sehr flexibel, aber im Normfall sehr kompakt und verständlich
- Vererbung: Testklassen können ggfs. abgeleitet und dann direkt als @SpringBootTest verwendet werden
- Mocking immer noch möglich
- @SpringBootTest usw. immer noch möglich

Sonstiges

- Tests extrem schnell
- Tests sehr flexibel, aber im Normfall sehr kompakt und verständlich
- Vererbung: Testklassen können ggfs. abgeleitet und dann direkt als @SpringBootTest verwendet werden
- Mocking immer noch möglich
- @SpringBootTest usw. immer noch möglich
- Eingriff auf tiefer Ebene möglich (z.B. Anlage über HTTP-Endpunkt, Prüfung gegen Fake JPA Repo)

Sonstiges

- Tests extrem schnell
- Tests sehr flexibel, aber im Normfall sehr kompakt und verständlich
- Vererbung: Testklassen können ggfs. abgeleitet und dann direkt als @SpringBootTest verwendet werden
- Mocking immer noch möglich
- @SpringBootTest usw. immer noch möglich
- Eingriff auf tiefer Ebene möglich (z.B. Anlage über HTTP-Endpunkt, Prüfung gegen Fake JPA Repo)
- Begrenzung Anzahl Datenbankzugriffe, ”Verlangsamung” von DB-Anfragen möglich

Sonstiges

- Tests extrem schnell
- Tests sehr flexibel, aber im Normfall sehr kompakt und verständlich
- Vererbung: Testklassen können ggfs. abgeleitet und dann direkt als @SpringBootTest verwendet werden
- Mocking immer noch möglich
- @SpringBootTest usw. immer noch möglich
- Eingriff auf tiefer Ebene möglich (z.B. Anlage über HTTP-Endpunkt, Prüfung gegen Fake JPA Repo)
- Begrenzung Anzahl Datenbankzugriffe, ”Verlangsamung” von DB-Anfragen möglich
- anfänglich langsamere, später dafür schnellere Implementierung

04

Beispiel

Beispiel partieller Integrationstest - Setup

```
1   public class CustomerControllerTest {
2     private TestInitializer<CustomersController> initializer;
3     private Sequence s;
4     private CustomersController ctrl;
5
6     @BeforeEach
7     void setup {
8       s = Sequence.primes(); // Use prime numbers sequence
9       initializer = new CustomersControllerInitializer();
10      // Replace AddressesRepository with specific implementation
11      // for each test in this class
12      initializer.registry().replace(AddressesRepository.class,
13        new MySpecificTestAddressesRepo());
14      ctrl = initializer.init();
15    }
16  }
```

Beispiel partieller Integrationstest – Test

```
1   public class CustomerControllerTest {
2     ...
3     @Test
4     void shouldGetAllCustomers() {
5       Assertions.assertTrue(ctrl.getAllCustomers().isEmpty());
6       // Creates referenced address transitively and persists it
7       var c1 = new CustomerTest.Builder(s).build();
8       var actual = ctrl.getAllCustomers();
9
10      Assertions.assertEquals(1, actual.size());
11      Assertions.assertEquals(c1, actual.get(0));
12    }
13  }
```

05

Danke

secunet sucht Mitarbeiter

secunet sucht Mitarbeiter

- Du möchtest Teil einer seit Jahren erfolgreich wachsenden Firma sein?

secunet sucht Mitarbeiter

- Du möchtest Teil einer seit Jahren erfolgreich wachsenden Firma sein?
- Du möchtest im Bereich IT-Security tätig sein?

secunet sucht Mitarbeiter

- Du möchtest Teil einer seit Jahren erfolgreich wachsenden Firma sein?
- Du möchtest im Bereich IT-Security tätig sein?
- Du möchtest eine sinnstiftende Tätigkeit, um z.B. kritische Infrastrukturen zu schützen?

secunet sucht Mitarbeiter

- Du möchtest Teil einer seit Jahren erfolgreich wachsenden Firma sein?
- Du möchtest im Bereich IT-Security tätig sein?
- Du möchtest eine sinnstiftende Tätigkeit, um z.B. kritische Infrastrukturen zu schützen?
- Du lieferst gerne qualitativ hochwertige Ergebnisse?

secunet sucht Mitarbeiter

- Du möchtest Teil einer seit Jahren erfolgreich wachsenden Firma sein?
- Du möchtest im Bereich IT-Security tätig sein?
- Du möchtest eine sinnstiftende Tätigkeit, um z.B. kritische Infrastrukturen zu schützen?
- Du lieferst gerne qualitativ hochwertige Ergebnisse?
- Behördliche Prozesse (Evaluierung von Code, ...) sind kein Problem für dich?

secunet sucht Mitarbeiter

- Du möchtest Teil einer seit Jahren erfolgreich wachsenden Firma sein?
- Du möchtest im Bereich IT-Security tätig sein?
- Du möchtest eine sinnstiftende Tätigkeit, um z.B. kritische Infrastrukturen zu schützen?
- Du lieferst gerne qualitativ hochwertige Ergebnisse?
- Behördliche Prozesse (Evaluierung von Code, ...) sind kein Problem für dich?

→ Bewirb dich bei secunet Security Networks AG!


secunet