

...
}
Der Zugriff erfolgt nun über die Repository Klasse und über Methodennamen oder eigene JPQL Abfragen definiert durch die @Query Annotation:

```
package ch.std.jumpstart.repository;&#xA;import java.util.List;&#xA;import org.springframework.data.jpa.repository.JpaRepository;&#xA;import org.springframework.data.jpa.repository.Query;&#xA;import ch.std.jumpstart.jpa.City;&#xA;public interface CityRepository extends JpaRepository {&#xA;    Optional findByName(String name);&#xA;    @Query(&#34;SELECT c FROM City c WHERE c.name LIKE %?1%&#34;)&#xA;    public List<City> findByNameLike(String name);&#xA;}&#xA;}
```

Der Zugriff auf das Repository erfolgt nun je nach Design direkt im REST Controller:

```
package ch.std.jumpstart.rest;&#xA;...&#xA;import ch.std.jumpstart.jpa.City;&#xA;import ch.std.jumpstart.repository.CityRepository;&#xA;@RestController&#xA;public class CityController {&#xA;    @Autowired&#xA;    CityRepository cityRepository;&#xA;    @GetMapping(&#34;/rest/cities&#34;)&#xA;    public City[] getCities(@RequestParam(value = &#34;value&#34;, required = false) String value) {&#xA;        List<City> cities = null;&#xA;        if (value == null) {&#xA;            cities = (List<City>) cityRepository.findAll();&#xA;        } else {&#xA;            cities = (List<City>) cityRepository.findByNameLike(value);&#xA;        }&#xA;        return cities.toArray(new City[0]);&#xA;    }&#xA;    @GetMapping(&#34;/rest/city/{id}&#34;)&#xA;    public City getCityById(@PathVariable Long id) {&#xA;        City city = cityRepository.findById(id).orElseThrow(() -> new CityNotFoundException(id));&#xA;        return city;&#xA;    }&#xA;    ...&#xA;}&#xA;}
```

Die Rückgabe von JPA Entity Instanzen direkt im REST Controller via JSON ist keine gute Praxis. Die JSON Response muss von der inneren JPA Struktur getrennt werden, z.B. via Map:

```
@GetMapping(&#34;/city/{id}&#34;)&#xA;public Map getCityById(@PathVariable Long id) {&#xA;    City city = cityRepository.findById(id).orElseThrow(() -> new CityNotFoundException(id));&#xA;    if (city == null) {&#xA;        return null;&#xA;    }&#xA;    return city.toMap();&#xA;}&#xA;}&#xA;public Map toMap() {&#xA;    Map map = new LinkedHashMap<>();&#xA;    map.put(&#34;id&#34;, this.id);&#xA;    map.put(&#34;name&#34;, this.name);&#xA;    return map;&#xA;}&#xA;}
```

Das direkte Rendering über die toMap-Methode via JPA Entity ist auch nicht optimal, besser wäre eine explizite Trennung via DTO (Data Transfer Object) oder Renderer Ebene.

Generic REST Controller

Ein noch besseres REST Controller Design sehen sie im Blog [Generic REST Endpoint mit Spring Boot](#)

Spring @DataJpaTest

Mit der Spring @DataJpaTest Annotation können Repositories einfach getestet werden. In Kombination mit der SpringRunner.class sind damit JPA Komponenten effizient mit der InMemory Datenbank H2 testbar. Alternativ kann mit einer realen Datenbank wie mysql gearbeitet werden. Das folgende Listing zeigt den JPA Test für das CityRepository:

```
package ch.std.jumpstart.repository;&#xA;import static org.junit.Assert.assertEquals;&#xA;import static org.junit.Assert.assertNotNull;&#xA;import static org.junit.Assert.assertTrue;&#xA;...&#xA;import ch.std.jumpstart.jpa.City;&#xA;import ch.std.jumpstart.repository.BookRepository.IsbnTitleOnly;&#xA;@RunWith(SpringRunner.class)&#xA;@DataJpaTest&#xA;@ActiveProfiles(&#34;test&#34;)&#xA;public class CityRepositoryIntegrationTest {&#xA;    @Autowired&#xA;    private CityRepository cityRepository;&#xA;    private City city;&#xA;    private List<City> cityList;&#xA;    @Before&#xA;    public void setup() {&#xA;        city = new City(&#34;Bern&#34;);&#xA;        this.cityRepository.save(city);&#xA;        this.cityList = new Array<>();&#xA;        for (int i = 0; i < 10; i++) {&#xA;            City localCity = new City(&#34;city-&#34; + i);&#xA;            this.cityRepository.save(localCity);&#xA;            this.cityList.add(localCity);&#xA;        }&#xA;    }&#xA;    @After&#xA;    public void tearDown() {&#xA;        this.cityRepository.deleteAllInBatch();&#xA;    }&#xA;    @Test&#xA;    public void testFindById() {&#xA;        City foundCity = this.cityRepository.findById(city.getId()).orElse(null);&#xA;        assertNotNull(foundCity);&#xA;        assertEquals(this.city, foundCity);&#xA;    }&#xA;    @Test&#xA;    public void testFindByName() {&#xA;        City foundCity = this.cityRepository.findByName(&#34;Bern&#34;).orElse(null);&#xA;        assertNotNull(foundCity);&#xA;        assertEquals(this.city, foundCity);&#xA;    }&#xA;}
```

Spring @SpringBootTest

Spring Boot Integrationstests lassen sich mit der `@SpringBootTest` Annotation schnell und einfach erstellen. Via `RandomPort` wird der Endpoint und damit die `RestController` gestartet und solche sind nun via `TestRestTemplate`, der `REST Client` Klasse, testbar. Das folgende Listing zeigt den Integrationstest zum `City Controller`:

```
package ch.std.jumpstart;
import ch.std.jumpstart.jpa.City;
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@ActiveProfiles("test")
public class CityControllerTests {
    @LocalServerPort
    private int port;
    @Autowired
    private ServletContext servletContext;
    @Autowired
    private TestRestTemplate restTemplate;
    private List<City> citiesList = new ArrayList<>();
    @Before
    public void setup() {
        this.citiesList.clear();
        String postUrl = this.getUrl("/rest/city");
        HttpEntity<City> request = new HttpEntity(new City("Bern"));
        City bernCity = this.restTemplate.postForObject(postUrl, request, City.class);
        this.citiesList.add(bernCity);
    }
    @Test
    public void testGetCities() throws Exception {
        String url = this.getUrl("/rest/cities?value=Bern");
        City[] cities = this.restTemplate.getForObject(url, City[].class);
        assertNotNull(cities);
        assertEquals(1, cities.length);
        City city = cities[0];
        assertNotNull(city);
        assertEquals("Bern", city.getName());
    }
    @Test
    public void testGetAllCities() throws Exception {
        String url = this.getUrl("/rest/cities");
        City[] cities = this.restTemplate.getForObject(url, City[].class);
        assertNotNull(cities);
        assertEquals(this.citiesList.size(), cities.length);
        for (City city : this.citiesList) {
            assertTrue(citiesList.contains(city));
        }
    }
    @Test
    public void testGetCityById() throws Exception {
        String url = this.getUrl("/rest/cities?value=Bern");
        City[] cities = this.restTemplate.getForObject(url, City[].class);
        assertNotNull(cities);
        assertEquals(1, cities.length);
        City city = cities[0];
        String urlById = this.getUrl("/rest/city/" + city.getId());
        City cityById = this.restTemplate.getForObject(urlById, City.class);
        assertEquals(city, cityById);
    }
    public String getUrl(String path) {
        return "http://localhost:" + port + servletContext.getContextPath() + path;
    }
}
```

Feedback

War dieser Blog für Sie wertvoll. Wir danken für jede Anregung und Feedback

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/2013>