

Testovanie REST servera

HTTP klient

RestUserDao

UINF/PAZ1c
epizóda 8



Testy REST controllera

- Nestačí iba volať testované metódy

```
@Test  
void testCreateUser() {  
    assertEquals(expected, userRestController.create(testUser));  
}
```

REST kontrolér závisí od anotácií

- Takýto test **netestuje** polovicu metódy

```
@PostMapping("/users")
@ResponseStatus(HttpStatus.CREATED)
public User create(@RequestBody User user) throws BadRequestException {
    return this.userDao.create(user);
}
```

- Metódy REST controllera nie sú určené na priame volanie z Javy.
- Sú určené na vzdialené volanie cez sieťový HTTP protokol
 - HTTP rieši Spring frejmworck cez **anotácie**
- Test by mal teda robiť to isté

Testy REST kontroléra

- Naštartujeme testovací REST server na náhodnom voľnom porte

@BeforeAll

```
static void startApplication() {
```

```
    System.setProperty("IN_MEMORY", "true");
```

```
    applicationContext = SpringApplication.run(EntranceRestApplication.class, "--server.port=0");
```

```
    var port = applicationContext.getEnvironment().getProperty("local.server.port",  
Integer.class);
```

```
    baseUrl = URI.create("http://localhost:" + port);
```

```
}
```

Volanie kontroléra z Testu

- Pripravenie HTTP požiadavky

```
@Test
void createReturnsCreatedUser() throws Exception {
    var newUser = new User(...);

    var request = HttpRequest.newBuilder(baseUri.resolve("/users"))
        .header("Content-Type", "application/json")
        .method(
            "POST", HttpRequest
                .BodyPublishers
                .ofString(OBJECT_MAPPER.writeValueAsString(newUser))
        ).build();
}
```

Z objektu urobíme JSON

Testy REST kontroléra

- Vykonanie požiadavky a parsovanie HTTP odpovede

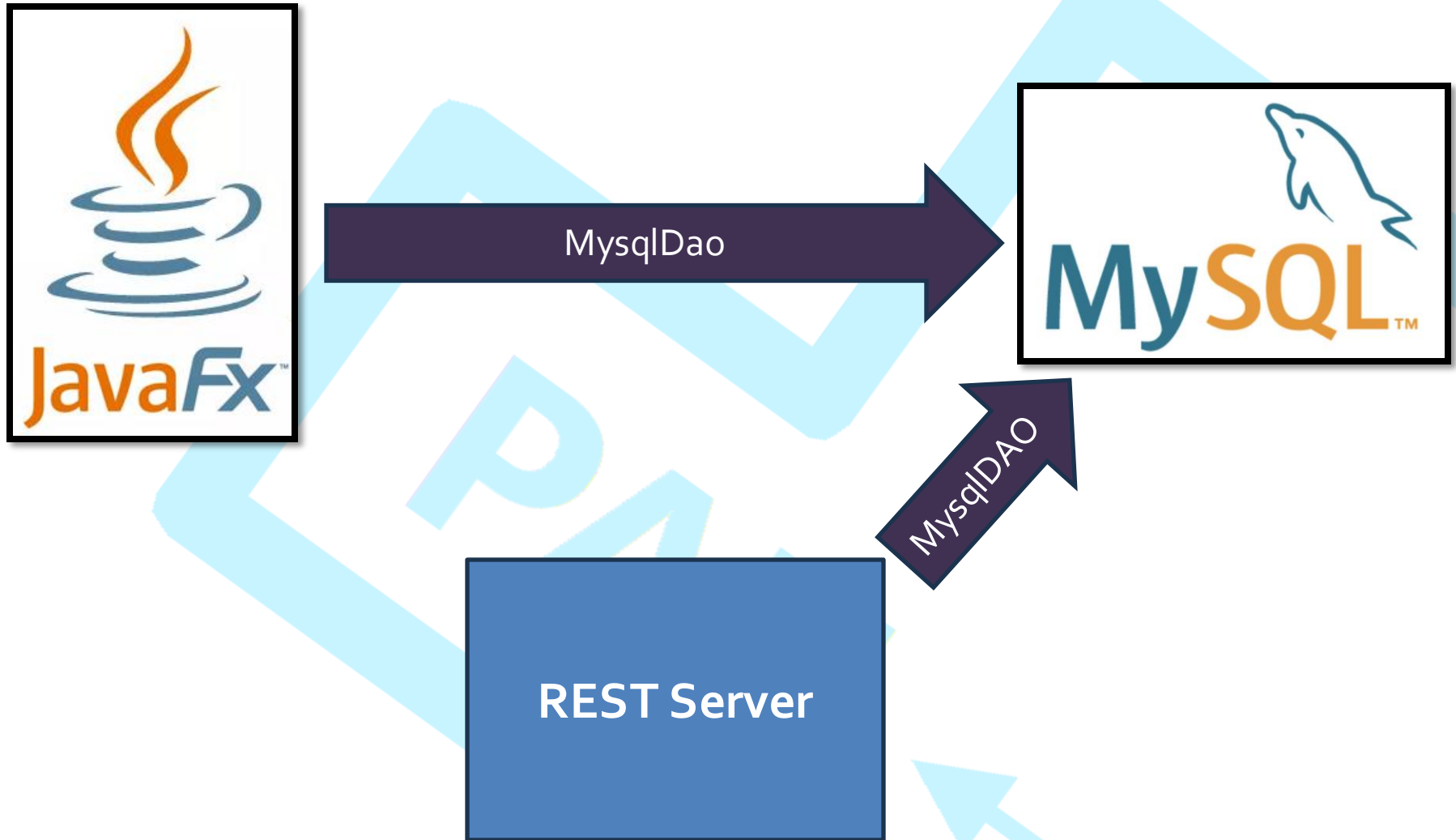
```
@Test
void createReturnsCreatedUser() throws Exception {
    var response = HTTP_CLIENT.send(
        request,
        HttpResponse.BodyHandlers.ofString()
    );

    assertEquals(201, response.statusCode());
    var createdUser = OBJECT_MAPPER.readValue(
        response.body(),
        User.class
    );
    assertTrue(createdUser.id() > 0);
}
```

HttpClient

- HttpClient-ov máme veľa
 - HTTPURLConnection zo starých verzií Javy
 - Apache HttpClient
 - Retrofit
 - **HttpClient z Javy 11**
 - Použili sme v teste
- Implementácia HTTP klienta je obľúbené programátorské cvičenie

Čo máme



Čo chceme



Prečo?

GUI appka beží na počítači zákazníka

- Dekompilácia
- Prenastavovanie premenných z externého debuggera
- Obchádzanie licencií
- Kompletný prístup do DB
 - No-no pre viac užívateľské appky s centrálnou DB

Row-Level Security

- Zabezpečenie aby užívateľ **A** NE mohol pristupovať k riadkom užívateľa **B**
 - Užívateľ môže vidieť iba svoje objednávky, iba svoje účty, atď.
- Okrem N-výnimiek
 - Teta za prepážkou môže vidieť aj cudzie objednávky, cudzie účty (ale iba vidieť), atď.
- To sa robí v aplikačnom kóde

Zabezpečenie aplikácie

Piráti, hackeri/crackeri vedia veľmi ľahko modifikovať hotový kód nejakej random Java appky.

- *if*-y budú vracať true, podhodia za interface inú implementáciu, ...

Ukradnúť prístupy do DB z našej bežiacej JavaFX appky je takmer triviálne.

Tenký klient

- GUI iba zobrazuje dáta - tenký klient
- Ťahá si ich z REST server appky, ktorá
 - kontroluje prihlásenia
 - kontroluje prístupy k riadkom
 - najmä cez cudzie kľúče a **role**
 - kontroluje oprávnenia užívateľa
 - pracuje s DB
 - ...

HttpClient wrapper

```
public class EntranceHttpClient {

    public EntranceHttpClient(ObjectMapper json, URI baseUri, HttpClient httpClient) {
        this.json = json;
        this.baseUri = baseUri;
        this.httpClient = httpClient;
    }

    public <T> EntranceHttpResponse<T> GETRequest(String path, Class<T> type) {
        var request = HttpRequest.newBuilder(this.baseUri.resolve(path)).GET().build();
        return executeRequest(request, type);
    }

    public <T> EntranceHttpListResponse<T> GETListRequest(String path, Class<T> type) {
        var request = HttpRequest.newBuilder(this.baseUri.resolve(path)).GET().build();
        return executeListRequest(request, type);
    }

    ...
}
```

HttpClient wrapper

```
public class EntranceHttpClient {  
    ...  
    public <T> EntranceHttpResponse<T> POSTRequest(String path, T body, Class<T> type) {  
        var request = HttpRequest.newBuilder(this.baseUri.resolve(path))  
            .header("Content-Type", "application/json")  
            .method("POST", HttpRequest.BodyPublishers.ofString(json.writeValueAsString(body)))  
            .build();  
        return executeRequest(request, type);  
    }  
  
    public <T> EntranceHttpResponse<T> PUTRequest(String path, T body, Class<T> type) {  
        ...  
    }  
}
```

HttpClient wrapper

```
public class EntranceHttpClient {  
    ...  
    private <T> EntranceHttpResponse<T> executeRequest(HttpRequest request, Class<T> type) {  
        HttpResponse<String> response = null;  
        try {  
            response = this.httpClient.send(request, HttpResponse.BodyHandlers.ofString());  
        } catch (InterruptedException e) {  
            Thread.currentThread().interrupt();  
            throw new RestClientException("REST request interrupted", e);  
        } catch (Exception e) {  
            throw new RestClientException("REST request failed", e);  
        }  
        ...  
    }  
}
```

HttpClient wrapper

```
...
var statusCode = response.statusCode();
if (statusCode >= 500) {
    throw new RestClientException("Unexpected REST response status: " + statusCode);
}

ApiError error = null;
if (statusCode >= 400 && statusCode < 500) {
    error = json.readValue(response.body(), ApiError.class);
}

T result = null;
if (statusCode >= 200 && statusCode < 300 && !response.body().isBlank()) {
    result = json.readValue(response.body(), type);
}

return new EntranceHttpResponse<>(statusCode, result, error == null ? null :
error.errorMessage());
}
```

HttpClient wrapper

- Kvôli typovému obmedzeniu Javy, musíme implementovať zvlášť variant pre mapovanie JSON polí do List<T>

```
private <T> EntranceHttpListResponse<T> executeListRequest(HttpRequest request, Class<T> type) {  
    ...  
    List<T> result = Collections.emptyList();  
    if (statusCode >= 200 && statusCode < 300 && !response.body().isBlank()) {  
        var listType = json.getTypeFactory().constructCollectionType(List.class, type);  
        result = json.readValue(response.body(), listType);  
    }  
  
    return new EntranceHttpListResponse<>(statusCode, result, error == null ? null :  
    error.errorMessage());  
}
```

HttpClient Wrapper

- Špeciálne znaky a symbol + sa nemajú dávať do URL HTTP požiadaviek.
 - Napr. `/users/name/Janči` treba enkódovať na `/users/name/Jan%C4%8Di`

```
public static String encodePathSegment(String value)
{
    return URLEncoder.encode(value,
StandardCharsets.UTF_8).replace("+", "%20");
}
```

UserRestDao

```
public class RestUserDao implements UserDao {  
    ...  
    @Override  
    public User findById(Long id) throws NotFoundException {  
        var response = this.httpClient.getRequest("users/id/" + id, User.class);  
        if (response.statusCode() == 404) {  
            throw new NotFoundException(response.errorMessage());  
        }  
        return response.body();  
    }  
    ...  
}
```

Factory naučíme rozhodovat o DAO

```
public enum Factory {  
    ...  
  
    public enum DaoType {  
        IN_MEMORY,  
        MYSQL,  
        REST  
    }  
  
    private DaoType daoType = DaoType.MYSQL;  
  
    private EntranceHttpClient entranceHttpClient;  
  
    public void setDaoType(DaoType daoType) {  
        this.daoType = Objects.requireNonNull(daoType);  
        this.userDao = null;  
        this.chipReaderDao = null;  
        this.userService = null;  
        this.entranceHttpClient = null;  
    }  
  
    private URI getRestBaseUri() {  
        return URI.create(System.getProperty("REST_BASE_URL", "http://localhost:8080"));  
    }  
  
    private EntranceHttpClient getEntranceHttpClient() {  
        if (this.entranceHttpClient == null) {  
            this.entranceHttpClient = new EntranceHttpClient(  
                new ObjectMapper(), getRestBaseUri(), HttpClient.newHttpClient());  
        }  
        return this.entranceHttpClient;  
    }  
    ...  
}
```

Factory naučíme rozhodovat o DAO

```
public UserDao getUserDao() {  
    if (this.userDao != null) {  
        return this.userDao;  
    }  
  
    switch (this.daoType) {  
        case IN_MEMORY -> this.userDao = new MemoryUserDao();  
        case MYSQL -> this.userDao = new MysqlUserDao(this.getJdbcOperations());  
        case REST -> this.userDao = new RestUserDao(getEntranceHttpClient());  
    }  
  
    return this.userDao;  
}
```

Otázky?

