TODO App Backend

Adding Spring Security

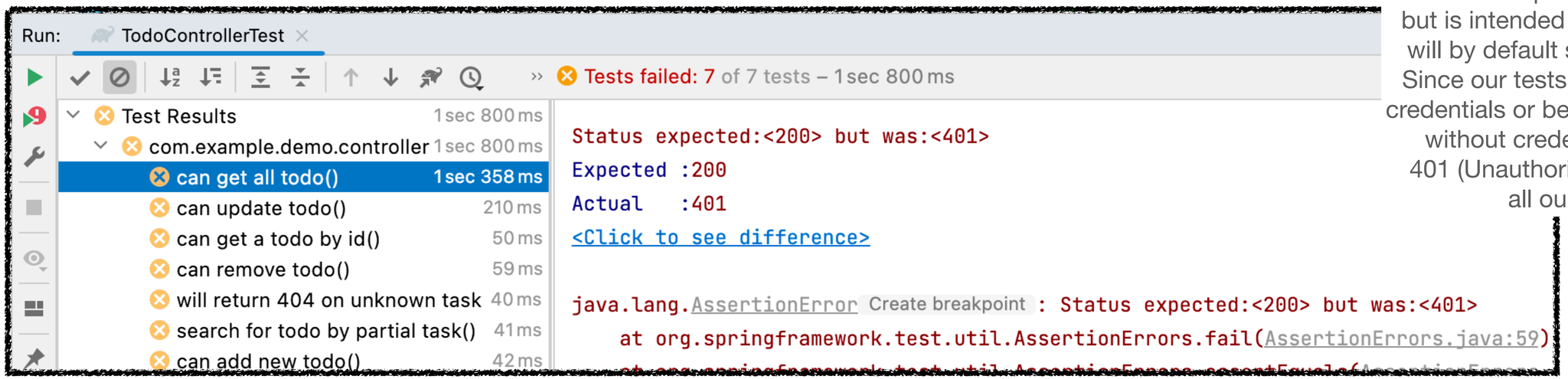
Exercise 10

build.gradle.kts (demo)

```
16      mavenCentral()
17  }
18
19  dependencies { this: DependencyHandlerScope
20      implementation("org.springframework.boot:spring-boot-starter-actuator")
21      implementation("org.springframework.boot:spring-boot-starter-data-jpa")
22      implementation("org.springframework.boot:spring-boot-starter-web")
23      implementation("org.springframework.boot:spring-boot-starter-security")
24      implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
25      implementation("org.jetbrains.kotlin:kotlin-reflect")
26      implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
27      runtimeOnly("com.h2database:h2")
28      testImplementation("org.springframework.boot:spring-boot-starter-test")
29      testImplementation("com.ninja-squad:springmockk:3.1.1")
30  }
```

Execute our Controller Tests

Exercise 10



All the tests are failing now ಠ_ಠ

-

This comes probably a bit surprising but is intended since spring security will by default secure all endpoints. Since our tests are not handling any credentials or better say send requests without credentials we getting a 401 (Unauthorized) status code for all our endpoints

Configure Spring Security

Exercise 10

Create a config package and add a ‚SecurityConfig‘ class. (The class can have every name you want.)

Annotate the class with @EnableWebSecurity to make this class a configuration bean

Extend ‚WebSecurityConfigurerAdapter‘.

-

It provides a convenient base class for creating a WebSecurityConfigurer instance. The implementation allows customization by overriding methods.

Configure Spring Security / add users

Exercise 10

```
@EnableWebSecurity
class SecurityConfig : WebSecurityConfigurerAdapter() {

    private val String.encoded : String!
        get() = PasswordEncoderFactories
            .createDelegatingPasswordEncoder()
            .encode( rawPassword: this)

    override fun configure(auth: AuthenticationManagerBuilder) {
        auth
            .inMemoryAuthentication() InMemoryUserDetailsManagerConfigu
            .withUser( username: "kigali-coder") UserDetailsManagerConfi
            .password("pa55w0rd".encoded)
            .roles("USER")
            .and() InMemoryUserDetailsManagerConfigurer<AuthenticationManage
            .withUser( username: "admin") UserDetailsManagerConfigurer<Au
            .password("password".encoded)
            .roles("ADMIN", "USER")
    }

}
```

We are creating a so called ‚extension property‘ to conveniently turn strings into password encoded strings.

-

If you want to read more about extension properties you can have a look here:
https://kotlinlang.org/docs/extensions.html#extension-properties

Create some users. For simplicity we use in memory / hardcoded users with credentials here, but they can also live in a database etc.

-

We have to different users with different roles.

We are using our ‚encoded‘- extension property to conveniently turn strings into password encoded strings

Configure HTTP Security

Exercise 10

```
@EnableWebSecurity
class SecurityConfig : WebSecurityConfigurerAdapter() {

    override fun configure(http: HttpSecurity?) {
        http { this: HttpSecurityDsl
            csrf { disable() }
            httpBasic {}
            authorizeRequests { this: AuthorizeRequestsDsl
                authorize( pattern: "/**", permitAll)
            }
        }
    }

    private val String encoded : String
```

Add custom http security by overriding corresponding configure method of WebSecurityConfigurerAdapter

-

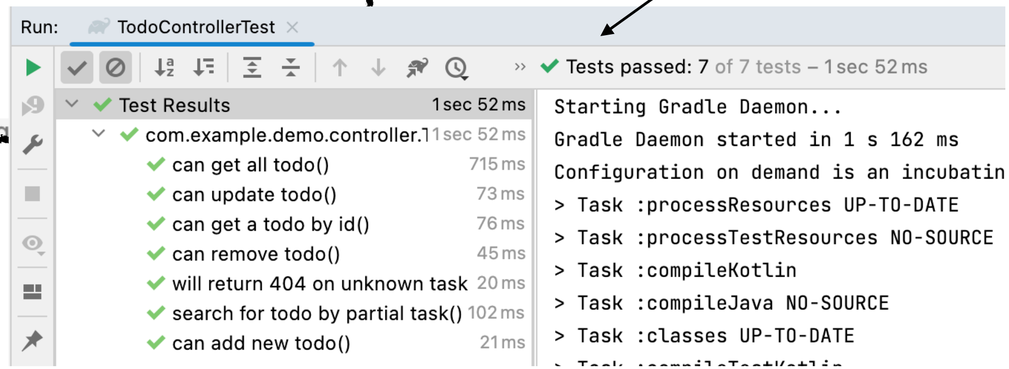
We disable csrf (cross site request forgery) for now since its enabled by default, but we don't want to focus on that for now

-

We decide we want to use a basic authentication that will work with username password credentials

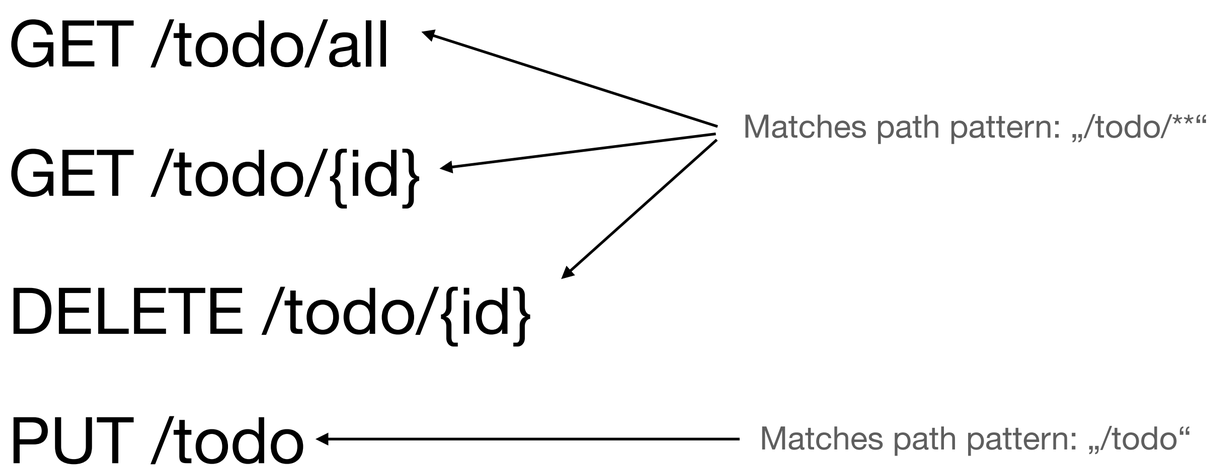
-

We just permit all the paths to be not secured for now (should lead to passing test again since this disables security for all endpoints)



Lets see which mappings we have

Exercise 10



Lets decide that all the endpoints that are matching „/todo/**“ are accessible by users and all the endpoints that match „/todo“ are only accessible by admins.

-

This would lead to the behavior that only users with the role admin are able to add new todos

Adjust config

Exercise 10

```
override fun configure(http: HttpSecurity?) {
    http { this: HttpSecurityDsl
        csrf { disable() }
        httpBasic {}
        authorizeRequests { this: AuthorizeRequestsDsl
            authorize( pattern: "/todo", hasAuthority( authority: "ROLE_ADMIN"))
            authorize( pattern: "/todo/**", hasAuthority( authority: "ROLE_USER"))
            authorize( pattern: "/**", permitAll)
        }
    }
}
```

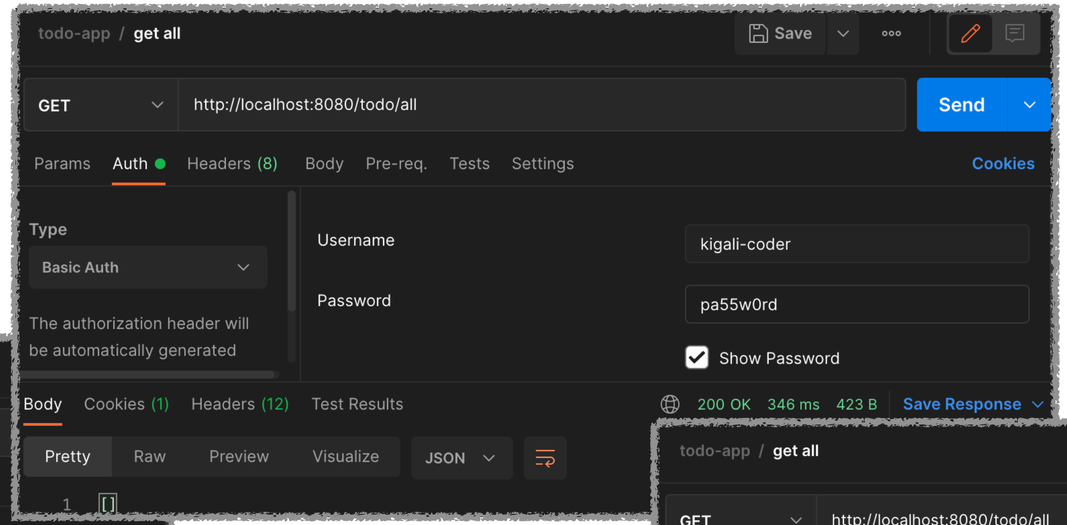
Lets decide that all the endpoints that are matching „/todo/**“ are accessible by users and all the endpoints that match „/todo“ are only accessible by admins.
-
This would lead to the behavior that only users with the role admin are able to add new todos

All none matching endpoints would fallback to permitAll and thereby are not secured

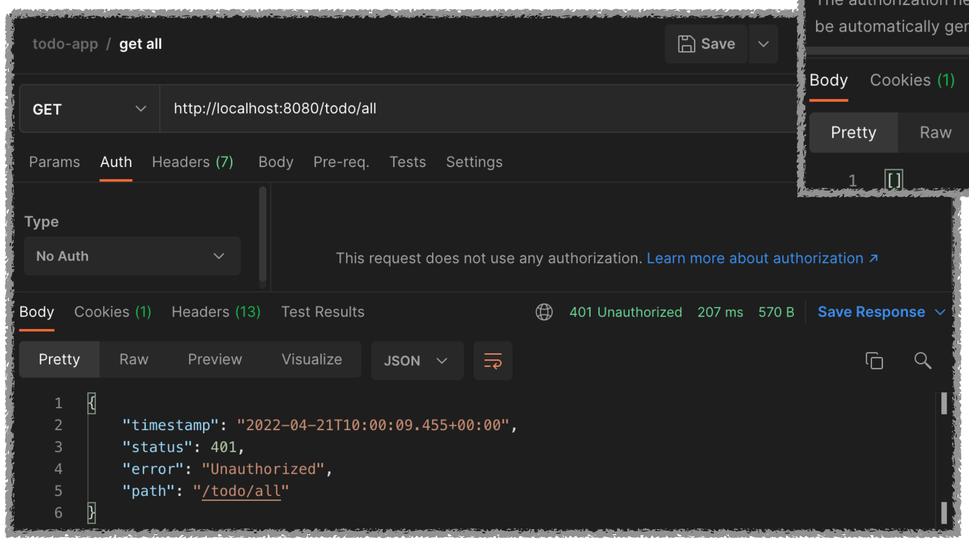
Lets try user endpoint using postman

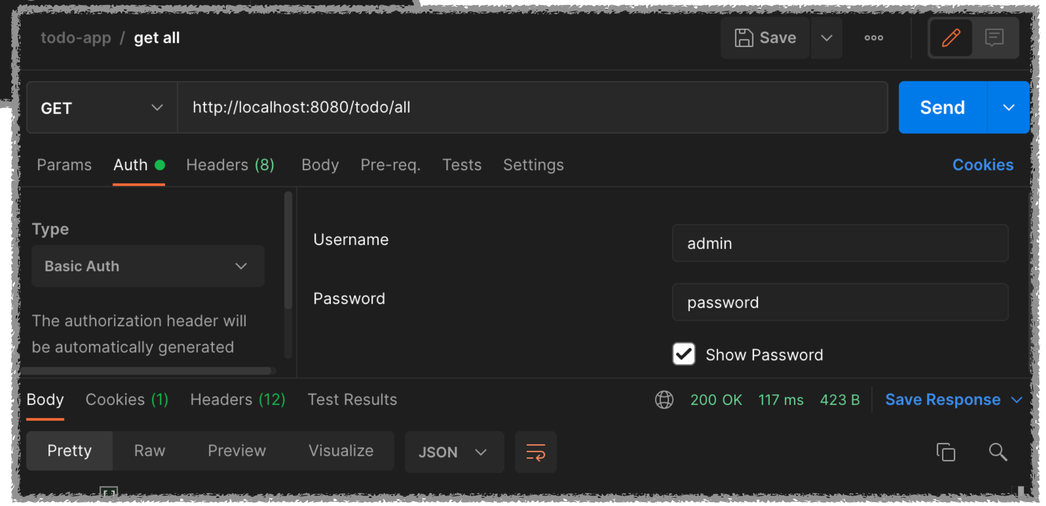
Exercise 10

Can call endpoint with user credentials



No credentials send leads to 401



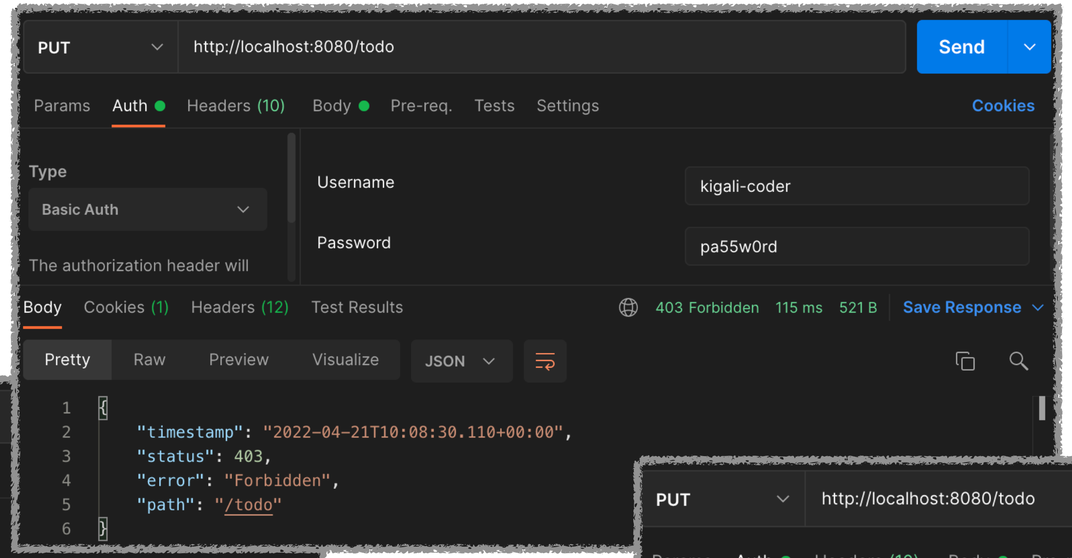


Can call endpoint with admin credentials

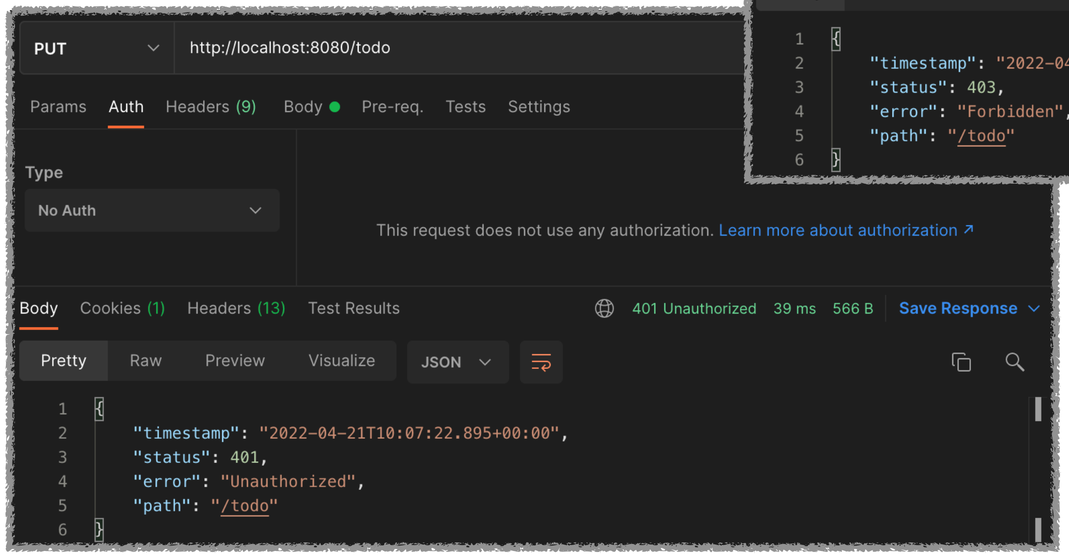
Lets try admin endpoint using postman

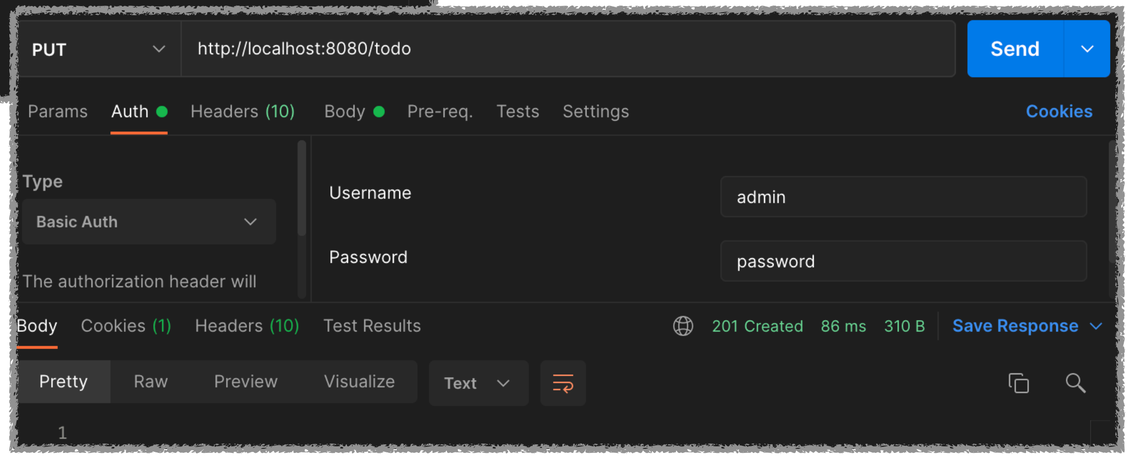
Exercise 10

Can not call admin endpoint with user credentials



No credentials send leads to 401





Can call endpoint with admin credentials

Lets adjust the tests to check this

Exercise 10

build.gradle.kts

```
dependencies { this: DependencyHandlerScope
    implementation("org.springframework.boot:spring-boot-starter-actuator")
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")
    implementation("org.springframework.boot:spring-boot-starter-web")
    implementation("org.springframework.boot:spring-boot-starter-security")
    implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
    implementation("org.jetbrains.kotlin:kotlin-reflect")
    implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8")
    runtimeOnly("com.h2database:h2")
    testImplementation("org.springframework.boot:spring-boot-starter-test")
    testImplementation("org.springframework.security:spring-security-test")
    testImplementation("com.ninja-squad:springmockk:3.1.1")
}
```

Let’s add a test scope dependency.
Those dependency will only be
available in our test sources
(e.g. src/test folder of our project).

Lets adjust the tests to check this

Exercise 10

This needs to match a role we defined in our SpringSecurity bean

Before

```
@Test
fun `can get a todo by id`() {
    val persistedTestData : (Mutable)List<TodoEntity!> = todoRepository.saveAll(testData)

    mockMvc.get( urlTemplate: "/todo/${persistedTestData[1].id}")
        .andExpect { this: MockMvcResultMatchersDsl
            status { isOk() }
            content { this: ContentResultMatchersDsl
                jsonPath( expression: "task", Matchers.containsString( substring: "cooking a meal"))
            }
        }
}
```

Old tests don't cares about credentials since they were not needed until we configured spring security

After

```
@Test
@WithMockUser(roles = ["USER"])
fun `can get a todo by id`() {
    val persistedTestData : (Mutable)List<TodoEntity!> = todoRepository.saveAll(testData)

    mockMvc.get( urlTemplate: "/todo/${persistedTestData[1].id}") { this: MockHttpServletRequestDsl

    }.andExpect { this: MockMvcResultMatchersDsl
        status { isOk() }
        content { this: ContentResultMatchersDsl
            jsonPath( expression: "task", Matchers.containsString( substring: "cooking a meal"))
        }
    }
}

@Test
fun `can NOT get a todo by id with unauthenticated user`() {
    mockMvc.get( urlTemplate: "/todo/1").andExpect { this: MockMvcResultMatchersDsl
        status { isUnauthorized() }
    }
}
```

Since we ‚spring-security-test' dependency we can use some really handy mocking annotation that is provided by the dependency to be able to run test with certain users, roles, etc.

We also want to automatically check if unauthorized users can not access the endpoint. therefore we add a new test to make sure.

Adjust more tests

Exercise 10

Before

```
@Test
fun `will return 404 on unknown task id requested`() {
    mockMvc.get( urlTemplate: "/todo/1337")
        .andExpect { this: MockMvcResultMatchersDsl
            status { isNotFound() } // check if status is 404 (not found)
        }
}
```

After

```
@Test
@WithMockUser(roles = ["USER"])
fun `will return 404 on unknown task id requested`() {
    mockMvc.get( urlTemplate: "/todo/1337")
        .andExpect { this: MockMvcResultMatchersDsl
            status { isNotFound() } // check if status is 404 (not found)
        }
}
```

Adjusted test stays the same, but we add an user mock

Practical Exercise

Exercise 10

Depending on what we learned about how to mock users,
Please adjust/add all necessary tests to make sure our endpoints:

- still work with appropriate user
- Will return 401 without given user
- Create or update a todo can only be done by role ‚ADMIN‘ and not by role ‚USER‘ (should get 403-Forbidden)