TODO App Backend

Working with ‚ResponseEntity‘

Exercise 7

- Let us fine-tuning HTTP responses using Spring by setting the body, status, and headers of an HTTP response using ResponseEntity

ResponseEntity represents the whole HTTP response:

- status code
- headers
- body

As a result, we can use it to fully configure the HTTP response.

Use ResponseEntity

Exercise 7

Our former implementation:

```
@RestController
@RequestMapping("/todo")
class TodoController(
    private val todoService: TodoService
) {
    @GetMapping("/all")
    fun getAll() : List<TodoResponse>  =
        todoService.getAll().toResponse()
```

We didn't care about the status code etc.
we just return our response object directly
and thereby rely on spring-boot default (status 200)

New implementation:

```
@RestController
@RequestMapping("/todo")
class TodoController(
    private val todoService: TodoService
) {
    @GetMapping("/all")
    fun getAll() : ResponseEntity<List<TodoResponse>!>  = ResponseEntity
        .ok()
        .body(todoService.getAll().toResponse())
```

Now we are returning a ResponseEntity
(simular to what we did in the exception handler during exercise 6)
that is wrapping a defined status code and a body that we pass.
ResponseEntity is a generic type. Consequently, we can use any type as the response body.
Spring takes care of the rest like Serialization of body to json and setting response status.

Use meaningful Status codes for certain use-cases

Exercise 7

Before

```
@PutMapping
fun putTodo(
    @RequestBody todoRequest: TodoRequest
) {
    val todo : TodoEntity  = todoRequest.id?.let { it: Int
        todoService.getById(it).apply { this: TodoEntity
            task = todoRequest.task
            completed = todoRequest.completed
        }
    } ?: todoRequest.toEntity()

    todoService.createOrUpdate(todo)
}
```

After

```
@PutMapping
fun putTodo(
    @RequestBody todoRequest: TodoRequest
): ResponseEntity<Unit> {
    val todo : TodoEntity  = todoRequest.id?.let { it: Int
        todoService.getById(it).apply { this: TodoEntity
            task = todoRequest.task
            completed = todoRequest.completed
        }
    } ?: todoRequest.toEntity()

    todoService.createOrUpdate(todo)
    return ResponseEntity.status(201).build()
}
```

Since we want to return a ResponseEntity without a body here, we pass Kotlin build-in type „Unit“ (which is equivalent to Javas „void“) as generic.

We don’t return a repose body but a http status code that fits our use-case better. Instead of the default status code (200 OK) we are now returning status code 201 (Created).

Overview of all http status codes and there meaning:

- https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Check if this works / adjust tests

Exercise 7

```
@Test
fun `can add new todo`() {
    mockMvc.put( urlTemplate: "/todo") { this: MockHttpServletRequestDsl
        contentType = MediaType.APPLICATION_JSON
        content = TodoRequest(task = "test").asJsonString()
    }.andExpect { this: MockMvcResultMatchersDsl
        status { isCreated() }
    }
}

@Test
fun `can update todo`() {
    val anExistingTodoId : Int = todoRepository.saveAll(testData).random().id

    mockMvc.put( urlTemplate: "/todo") { this: MockHttpServletRequestDsl
        contentType = MediaType.APPLICATION_JSON
        content = TodoRequest(
            id = anExistingTodoId,
            task = "check if updating a task works"
        ).asJsonString()
    }.andExpect { this: MockMvcResultMatchersDsl
        status { isCreated() }
    }

    val updatedTodo : TodoEntity? = todoRepository.findByIdOrNull(anExistingTodoId)
    Assertions.assertThat(updatedTodo?.task).isEqualTo("check if updating a task works")
}
```

Since we changed the return http status code to 201 (created) we check if this is working