HW1: Mid-term assignment report

Sandra Andrade [84904], v2020-04-15

1 Introdução

1.1 Vista geral do trabalho

Este projeto foi desenvolvido no âmbito da disciplina de TQS, com o objetivo de desenvolver testes de software funcionais, unitários e de integração.

A solução desenvolvida é uma aplicação que ao selecionar o país, estado e cidade pretendida é apresentado o estado do tempo e os níveis de poluição do ar na cidade escolhida, naquele momento.

1.2 Limitações

Ao desenvolver esta aplicação como o design da página não é relevante para esta disciplina foi bastante discriminado, por isso é algo a melhorar.

Há um problema com os repositórios, pois há certos testes unitários que falham. Ou seja é algo que vai continuar a ser trabalhado.

Também é possível fazer muitos mais tipos de pesquisas que podem ser feitas com os dados que são acumulados, logo a aplicação pode ficar muito mais complexa;

A maior limitação passa pelo número limitado de chamadas que se pode fazer à API externa que foi escolhida. Para combater este problema é necessário gerar uma nova chave sempre que a anterior expira.

2 Especificações do produto

2.1 Âmbito funcional e interações suportadas

Esta aplicação pode ser utilizada por qualquer pessoa, pois a informação que disponibiliza é importante para todos.

O cenário mais provável é um utilizador antes de sair de casa quer saber os níveis de poluição do ar e estado do tempo, para poder tomar alguma precaução necessária.

2.2 Arquitetura do sistema

A aplicação foi desenvolvida através do Spring Boot, que utiliza o servidor Tomcat. Para desenvolver a camada web foi utilizada Thymeleaf e HTML.

AirVisual(https://api-docs.airvisual.com/?version=latest) é a Api exterior que é consumida.

Diagrama UML

https://app.creately.com/diagram/kh71TTkpiKt/view

2.3 API para desenvolvedores

GET	/api/allCounytries	retorna todos os países
GET	/api/allStates	retorna todos os estados
GET	/api/allCities	retorna todas as cidades
GET	/api/allPollution	retorna todos os registos de poluição
GET	/api/allWeather	retorna todos os registos de tempo
GET	/api/states/{country}	retorna todos os estados de um país
GET	/api/cities/{country}/{state}	retorna todas as cidades de um certo estado
GET	/api/pollution/{country}/{state}/{city}	retorna todos os registos poluição de uma certa cidade
GET	/api/weather/{country}/{state}/{city}	retorna todos os registos de tempo de uma certa cidade
GET	/api/currentPollution/{country}/{state}/{city}	retorna o registo de poluição mais recente
GET	/api/currentWeather/{country}/{state}/{city}	retorna o registo de tempo mais recente

3 Garantia de qualidade

3.1 Estratégia geral para testar

Para testar a aplicação foram realizados teste unitários através do JUnit e Mockito, os teste de integração foram feitos com Spring Boot MockMvc e REST-Assured e os testes funcionais com Selenium WebDriver.

3.2 Testes unitários e de integração

Models

Todas as Entidades são testadas com JUnit, onde são verificados todos os getters e setters.

```java
public class CountryTest {

    private Country country;

    public CountryTest(){
        country = new Country("Portugal");
    }

    @Test
    public void testGetCountry(){
        assertEquals("Portugal", country.getCountry());
    }



    @Test
    public void testSetCountry(){
        country.setCountry("Korea");
        assertEquals("Korea",country.getCountry());
    }


}
```

Conversores

Estes passam por testes, também JUnit, para verificar que a conversão Entidade-String e String-Entidade é feita da forma correta.

```java
public class CountryConverterTest {

    private CountryConverter converter;
    private Country country;

    public CountryConverterTest(){
        converter = new CountryConverter();
        country = new Country("Portugal");
        country.setId(0);
    }

    @Test
    public void convertFromDToString() {
        assertNull(converter.convertToDatabaseColumn(null));

        String expected_out = "{\"id\":0,\"country\":\"Portugal\"}";
        assertEquals(converter.convertToDatabaseColumn(country), expected_out);
    }

    @Test
    public void convertFromStringToD() {
        assertNull(converter.convertToDatabaseColumn(null));

        String to_convert = "{\"id\":0,\"country\":\"Portugal\"}";
        assertEquals(converter.convertToEntityAttribute(to_convert).getId(),country.getId());
        assertEquals(converter.convertToEntityAttribute(to_convert).getCountry(),country.getCountry());

    }

}
```

Repositórios

São verificadas todas as chamadas à Base de dados, através de testes JUnit.

```
@DataJpaTest
public class StateRepositoryTest {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private StateRepository stateRepository;

    @Test
    public void whenFindAllByCountry_thenReturnListStates(){
        State aveiro = new State("Aveiro","Portugal");
        State tokyo = new State("Tokyo","Japan");
        State porto = new State("Porto","Portugal");

        entityManager.persist(aveiro);
        entityManager.persist(tokyo);
        entityManager.persist(porto);

        List<State> states = stateRepository.findAllByCountry("Portugal");
        assertThat(states).hasSize(2).extracting(State::getState).containsOnly(aveiro.getState(),porto.getState());
    }

    @Test
    public void whenInvalidCountry_thenReturnEmptyList(){
        List<State> states = stateRepository.findAllByCountry("doesNotExist");
        assertThat(states).isEmpty();
    }

    @Test
    public void whenFindAll_thenReturnListAllStates(){
        State aveiro = new State("Aveiro","Portugal");
        State tokyo = new State("Tokyo","Japan");
        State porto = new State("Porto","Portugal");

        entityManager.persist(aveiro);
        entityManager.persist(tokyo);
        entityManager.persist(porto);

        List<State> states = stateRepository.findAll();
        assertThat(states).hasSize(3).extracting(State::getState).containsOnly(aveiro.getState(),porto.getState(),tokyo.getState());
    }

}
```

Serviços

Aqui deviam de ser testadas todas as funções, mas existe um erro que torna impossível de verificar todas as funções, sendo só possível testar as que retornam findAll de um repositório. Todos os testes possíveis foram desenvolvidos com Mockito e JUnit.

```java
@ExtendWith(MockitoExtension.class)
public class CountryServiceTest {

    @Mock( lenient = true)
    private CountryRepository repository;

    @InjectMocks
    private CountryService service;

    @BeforeEach
    public void setUp() {
        Country portugal = new Country("Portugal");
        Country japan = new Country("Japan");
        Country korea = new Country("Korea");
        List<Country> allCountries = Arrays.asList(portugal, japan, korea);
        Mockito.when(repository.findAll()).thenReturn(allCountries);
    }

    @Test
     public void getAllContries_returnAllCountries() {
        Country portugal = new Country("Portugal");
        Country japan = new Country("Japan");
        Country korea = new Country("Korea");

        List<Country> allCountries = service.getAllCountries();

        verifyFindAllCountryIsCalledOnce();
        assertThat(allCountries).hasSize(3).extracting(Country::getCountry).contains(portugal.getCount
    }
```

Para testar os serviço que consome a Api exterior é utilizado o PowerMock em conjunto com JUnit para verificar que todos os consumos são feitos de forma correta.

```java
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT, classes = AirQualityApplication.class)
@AutoConfigureMockMvc
public class ApiControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private CountryService countryService;

    @MockBean
    private StateService stateService;

    @MockBean
    private CityService cityService;

    @MockBean
    private PollutionService pollutionService;

    @MockBean
    private WeatherService weatherService;

    @Test
    public void allCountries_Status200() throws Exception{
        Country portugal = new Country("Portugal");
        Country japan = new Country("Japan");
        Country korea = new Country("Korea");
        List<Country> allCountries = Arrays.asList(portugal,japan,korea);
        given(countryService.getAllCountries()).willReturn(allCountries);
        mvc.perform(get("/api/allCountries")
                .contentType(MediaType.APPLICATION_JSON))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
                .andExpect(jsonPath("$", hasSize(greaterThanOrEqualTo(3))))
                .andExpect(jsonPath("$[0].country", is(portugal.getCountry())))
                .andExpect(jsonPath("$[1].country", is(japan.getCountry())))
                .andExpect(jsonPath("$[2].country", is(korea.getCountry())));
        verify(countryService, VerificationModeFactory.times(1)).getAllCountries();
        reset(countryService);
    }
```

Controllers

Testam que todos os endPoints retornam a informação correta e válida. Mas como já referido como há um problema com os Serviços os testes não estão completos.

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT, classes = AirQualityApplication.class)
@AutoConfigureMockMvc
public class ApiControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private CountryService countryService;

    @MockBean
    private StateService stateService;

    @MockBean
    private CityService cityService;

    @MockBean
    private PollutionService pollutionService;

    @MockBean
    private WeatherService weatherService;

    @Test
    public void allCountries_Status200() throws Exception{
        Country portugal = new Country("Portugal");
        Country japan = new Country("Japan");
        Country korea = new Country("Korea");
        List<Country> allCountries = Arrays.asList(portugal,japan,korea);
        given(countryService.getAllCountries()).willReturn(allCountries);
        mvc.perform(get("/api/allCountries")
                .contentType(MediaType.APPLICATION_JSON))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
                .andExpect(jsonPath("$", hasSize(greaterThanOrEqualTo(3))))
                .andExpect(jsonPath("$[0].country", is(portugal.getCountry())))
                .andExpect(jsonPath("$[1].country", is(japan.getCountry())))
                .andExpect(jsonPath("$[2].country", is(korea.getCountry())));
        verify(countryService, VerificationModeFactory.times(1)).getAllCountries();
        reset(countryService);
    }
```

3.3 Testes funcionais

Para fazer estes testes foi criado um projeto à parte onde podemos encontrar as próximas duas classes de testes.

Selenium WebDriver

Com o auxílio de um plugin do Chrome e o Katalon Automatic Recorder, é possível recriar uma interação com o sistema. Onde é selecionado um país, um estado e uma cidade, de seguida aparece a informação sobre o estado do tempo e poluição do ar atual daquela cidade. Sendo que cada passo é verificado e testado.

```java
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT, classes = AirQualityApplication.class)
@AutoConfigureMockMvc
public class ApiControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private CountryService countryService;

    @MockBean
    private StateService stateService;

    @MockBean
    private CityService cityService;

    @MockBean
    private PollutionService pollutionService;

    @MockBean
    private WeatherService weatherService;

    @Test
    public void allCountries_Status200() throws Exception{
        Country portugal = new Country("Portugal");
        Country japan = new Country("Japan");
        Country korea = new Country("Korea");
        List<Country> allCountries = Arrays.asList(portugal,japan,korea);
        given(countryService.getAllCountries()).willReturn(allCountries);
        mvc.perform(get("/api/allCountries")
                .contentType(MediaType.APPLICATION_JSON))
                .andDo(print())
                .andExpect(status().isOk())
                .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
                .andExpect(jsonPath("$", hasSize(greaterThanOrEqualTo(3))))
                .andExpect(jsonPath("$[0].country", is(portugal.getCountry())))
                .andExpect(jsonPath("$[1].country", is(japan.getCountry())))
                .andExpect(jsonPath("$[2].country", is(korea.getCountry())));
        verify(countryService, VerificationModeFactory.times(1)).getAllCountries();
        reset(countryService);
    }
```

REST-Assured

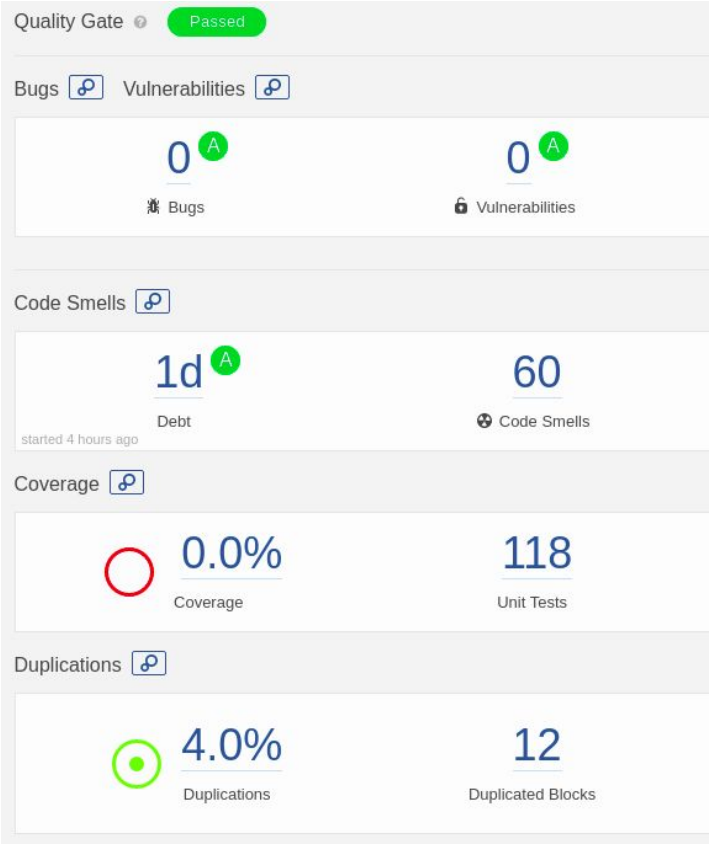
Através desta tecnologia são testados todos os endPoints da Api.

```java
@Test
public void getAllCountriesTest() {
    String uriBase = "http://localhost:8080/api/allCountries";

    given()
            .relaxedHTTPSValidation()
            .when()
            .get(uriBase)
            .then()
            .statusCode(200)
            .contentType(ContentType.JSON);
}
```

3.4 Análise de código estático

Para a análise do código foi utilizado o SonarQube.



Há muitos code smell pois as palavras country, countries, state, states, city e cities são repetidas muitas vezes para estabelecer várias variáveis. Isto foi levado em causa mas a troca do nome dessas variáveis ia tornar o código difícil de ler.

4 Referências e recursos

Project resources

Repositório Git:

https://github.com/S-Andrade/TQS_HW1

Reference materials

https://www.baeldung.com/start-here
https://spring.io/guides/