

HW1: Mid-term assignment report

Manuel Maria Guerra da Cruz Diaz [103645], v2023-04-11

1	Introduction	2
1.1	Overview of the work	2
1.2	Current limitations	2
2	Product specification	2
2.1	Functional scope and supported interactions	2
2.2	System architecture	4
2.3	API for developers	4
3	Quality assurance	7
3.1	Overall strategy for testing	7
3.2	Unit and integration testing	7
3.3	Code quality analysis	11
4	References & resources	12

1 Introduction

1.1 Overview of the work

Este relatório apresenta um projeto que visa fornecer informação sobre a qualidade do ar para uma região ou cidade específica. O projecto inclui uma aplicação/página web simples, uma integração com uma fonte externa de dados (API) de qualidade do ar, uma cache interna para recuperar dados de uma forma eficiente, e uma **REST API** para um acesso programático aos dados.

A interface desenvolvida permite, desta forma, aos utilizadores, seleccionar um país, um estado e uma cidade e visualizar a qualidade do ar para esse local, bem como os dados relacionados com o tempo. É possível ainda verificar os detalhes da cache.

1.2 Current limitations

Relativamente à API externa que decidi escolher, esta apresentou alguns problemas em algumas cidades. Às vezes quando pesquiso sobre uma determinada cidade, esta aparece que está disponível, porém quando vou pesquisar a informação dá erro.

Outra limitação que esta apresenta e, que só mais tarde me apercebi, é que o acesso aos valores de concentração de cada métrica em específico, só é possível com a versão paga da API.


Para além disso, algumas funcionalidades ficaram por implementar devido à falta de tempo, como por exemplo, as nomeadas para pontos extra (Second API e Continuous Integration Framework) e os testes funcionais, por exemplo, através do Selenium WebDriver.

2 Product specification

2.1 Functional scope and supported interactions

A aplicação que desenvolvi é muito simples. O utilizador tem à sua disposição apenas uma página, onde pode seleccionar o país, o estado e a cidade que pretende obter o tempo e a qualidade do ar. A informação aparece logo em baixo, assim que clica no botão “Get Data”, podendo apagar, de seguida, com o botão “Delete Data”. Para além disto, é possível ainda verificar os detalhes da cache clicando no botão de “Update Data”.

São apresentados 3 “dropdowns” para escolher o país, estado e cidade.



Select the country, State and City

select the country ▼

select the state ▼

select the city ▼

Get Data Delete Data

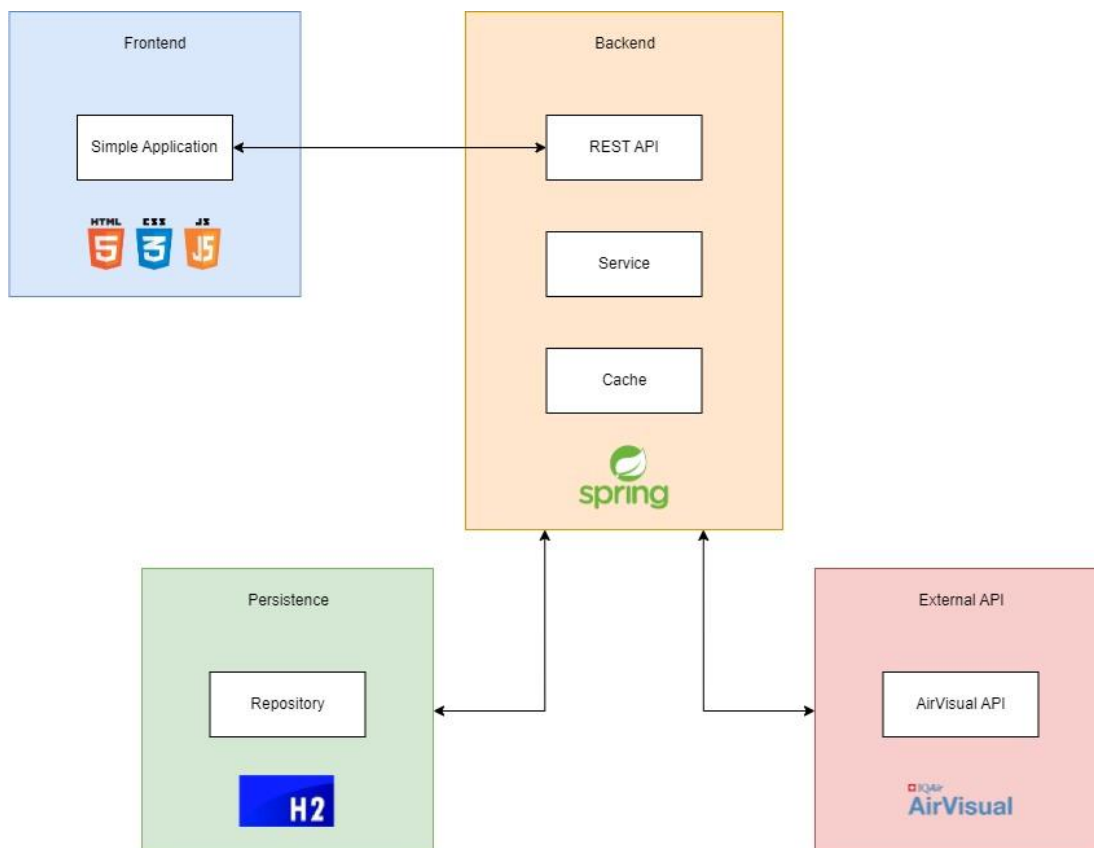
Logo abaixo, são nos mostrados os detalhes acerca do tempo e da qualidade do ar, nas secções “Weather” e “Air Quality”, respetivamente.

Weather		
Temperature:	21 Â°C	
Pressure:	1015 hPa	
Humidity:	83 %	
Wind speed:	6.17 m/s	
Air Quality		
Air Quality Index (AQI) on US standard:		24
Air Quality Index (AQI) on China standard:		8
Main pollutant for the US AQI:		p2
Main pollutant for the Chinese AQI:		p2

Como dito anteriormente, e por último, aparecem os detalhes da cache, caracterizados por o número de “hits” (cache utilizada), “misses” (cache não utilizada, ou seja, utilização da API externa) e “requests” (pedidos).

Cache Details		
Hits:	0	
Misses:	0	
Requests:	0	
Update Cache		

2.2 System architecture



Como Podemos ver acima, o “frontend” foi feito a partir das tecnologias web **HTML**, **CSS** e **JavaScript**, com a integração do **JQuery** e **AJAX**.

O “Backend” foi feito através do **Spring Boot** e contém uma REST API, um “Service” que chama a API externa e uma implementação da Cache. Por último, o repositório utilizado foi o **H2**, com a conexão feita através do **JPA**.

2.3 API for developers

Existem vários endpoints que os developers podem utilizar na nossa API. Como vamos poder ver, temos vários /list endpoints que retornam todas as cidades, estados e países disponíveis. Outro endpoint para obter a informação que pretendemos e por último, um relativo aos detalhes da cache.

- **/api/list/** (obter todos os países disponíveis) – **GET**

GET

http://localhost:8080/api/list

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 295 ms

Size: 1.85 KB

Save as Example

...

Pretty

Raw

Preview

Visualize

JSON

...

```
1  [
2    "Albania",
3    "Algeria",
4    "Andorra",
5    "Angola",
6    "Argentina",
7    "Armenia",
8    "Australia",
9    "Austria",
10   "Azerbaijan",
11   "Bahamas",
12   "Bahrain",
13   "Bangladesh",
14   "Belgium",
15   "Belize",
16   "Bermuda",
17   "Bolivia",
```

- **/api/list/{country}** (obter todos os estados do país selecionado) – **GET**

GET

http://localhost:8080/api/list/USA

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 1195 ms

Size: 763 B

Save as Example

...

Pretty

Raw

Preview

Visualize

JSON

...

```
1  [
2    "Alabama",
3    "Alaska",
4    "Arizona",
5    "Arkansas",
6    "California",
7    "Colorado",
8    "Connecticut",
9    "Delaware",
10   "Florida",
11   "Georgia",
12   "Guam",
13   "Hawaii",
14   "Idaho",
15   "Illinois",
16   "Indiana",
17   "Iowa",
```

- **/api/list/{country}/{state}** (obter todas as cidades do estado selecionado) – GET

GET ▼ http://localhost:8080/api/list/USA/California Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 449 ms Size: 10.23 KB Save as Example

Pretty Raw Preview Visualize JSON ▼ ≡

```

1  [
2    "Acalanes Ridge",
3    "Ahwahnee",
4    "Alameda",
5    "Alamo",
6    "Albany",
7    "Alhambra",
8    "Aliso Viejo",
9    "Alpine",
10   "Alta Sierra",
11   "Altadena",
12   "Alturas",
13   "Alum Rock",
14   "Amador",
15   "American Canyon",
16   "Anaheim",
17   "Anderson",

```

- **/api/list/{country}/{state}/{city}** (obter o tempo e a qualidade do ar) – GET

GET ▼ http://localhost:8080/api/USA/California/Alamo Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 335 ms Size: 428 B Save as Example

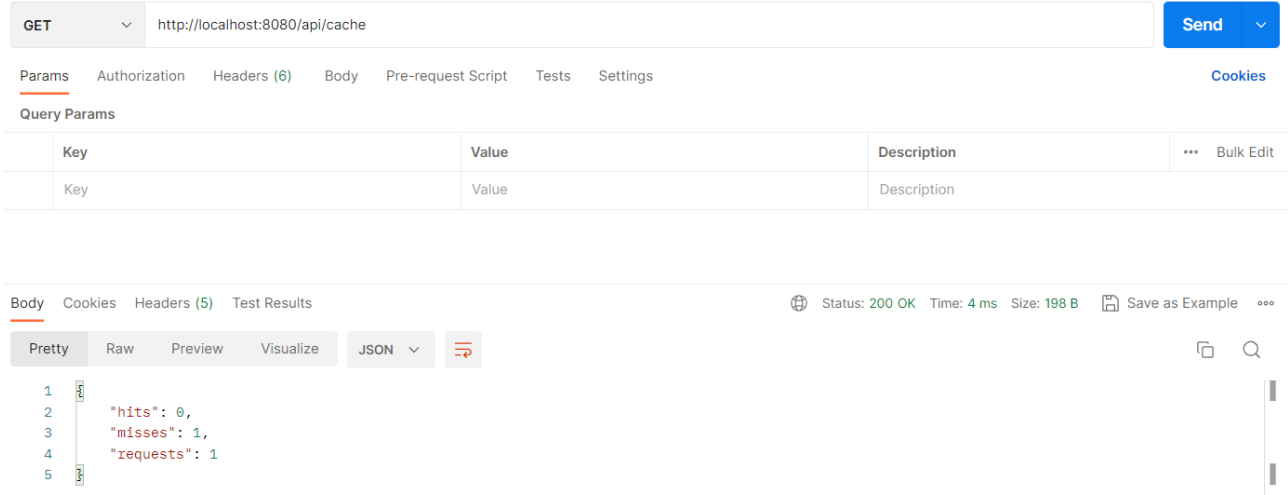
Pretty Raw Preview Visualize JSON ▼ ≡

```

1  {
2    "id": 0,
3    "name": "Alamo",
4    "state": "California",
5    "country": "USA",
6    "timestamp": "2023-04-11T01:00:00.000Z",
7    "weather": {
8      "temperature": 18.0,
9      "pressure": 1015.0,
10     "humidity": 62.0,
11     "windSpeed": 5.66
12   },
13   "airQuality": {
14     "aqiUs": 0.0,
15     "aqiCn": 0.0,
16     "mainPollutantUs": "p2",
17     "mainPollutantCn": "p2"
18   }
19 }

```

- **/api/cache** (obter os detalhes da cache) – **GET**



GET Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results Status: 200 OK Time: 4 ms Size: 198 B Save as Example

Pretty Raw Preview Visualize JSON ⌵ ⌵

```
1 {
2   "hits": 0,
3   "misses": 1,
4   "requests": 1
5 }
```

3 Quality assurance

3.1 Overall strategy for testing

Relativamente aos testes, para os unitários utilizei o **Junit5** nos testes de integração e nos de serviço, **Mockito** e **SpringBoot MockMvc**.

Adotei uma estratégia “Test Driven Development” (TDD) e à medida que ia desenvolvendo a estrutura do meu projeto ia testando de seguida, de modo a evitar alterar grandes quantidades de Código e certificar-me de que aquilo que ia fazendo estava a ir de acordo com as minhas expectativas.

3.2 Unit and integration testing

Começando pelos **testes unitários**, eu implementei testes nas entidades **City**, **Weather** e **AirQuality**. Podemos ver em baixo vários “asserts” de modo a testar os “getters” implementados nas respetivas classes.

```

public class CityTests {

    @Test
    void gettersTest(){
        Weather weather = new Weather(temperature:21.0, pressure:95.6, humidity:15.0, windSpeed:5.4);

        AirQuality airQuality = new AirQuality(aqiUs:10.0, aqiCn:9.2, mainPolluentUs:"p1", mainPolluentCn:"p1");

        City city = new City(name:"London", state:"London", country:"United Kingdom", timestamp:"timestamp", weather, a

        assertEquals(expected:"London", city.getName());
        assertEquals(expected:"London", city.getState());
        assertEquals(expected:"United Kingdom", city.getCountry());
        assertEquals(expected:"timestamp", city.getTimestamp());
        assertEquals(weather, city.getWeather());
        assertEquals(airQuality, city.getAirQuality());

        assertEquals(expected:21.0, weather.getTemperature());
        assertEquals(expected:95.6, weather.getPressure());
        assertEquals(expected:15.0, weather.getHumidity());
        assertEquals(expected:5.4, weather.getWindSpeed());

        assertEquals(expected:10.0, airQuality.getAqiUs());
        assertEquals(expected:9.2, airQuality.getAqiCn());
    }
}

```

Também implementei este tipo de testes na classe **Cache**, como por exemplo, para testar o adicionar e o remover de um elemento:

```

public class CacheTests {
    private AirQualityCache airQualityCache;

    @BeforeEach
    void setUp() throws InterruptedException {
        this.airQualityCache = new AirQualityCache(cleanTime:1);
    }

    @AfterEach
    void tearDown(){
        this.airQualityCache.clearCache();
    }

    @Test
    void addValueTest(){
        assertEquals(expected:0, this.airQualityCache.getCacheSize());
        this.airQualityCache.addValue(key:"Los Angeles", new City(name:"Los Angeles", state:"California", country:"USA"));
        assertEquals(expected:1, this.airQualityCache.getCacheSize());
        assertEquals(expected:true, this.airQualityCache.containsCity(key:"Los Angeles"));
    }

    @Test
    void deleteValueTest(){
        assertEquals(expected:0, this.airQualityCache.getCacheSize());
        this.airQualityCache.addValue(key:"Los Angeles", new City(name:"Los Angeles", state:"California", country:"USA"));
        this.airQualityCache.addValue(key:"Aveiro", new City(name:"Aveiro", state:"Aveiro", country:"Portugal"));
        this.airQualityCache.deleteValue(key:"Los Angeles");
        assertEquals(expected:1, this.airQualityCache.getCacheSize());
        assertEquals(expected:false, this.airQualityCache.containsCity(key:"Los Angeles"));
    }
}

```

Para a classe **CityRepository** também foram implementados testes unitários com a ajuda da classe **TestEntityManager** e da anotação **@DataJpaTest**.

A maior parte dos testes gerados para esta classe são feitos pelo **JPA**, então implementei apenas estes testes:


```

@Test
void findCityByNameAndIdTest(){
    City city = new City(name:"Los Angeles", state:"California", country:"USA");
    testEntityManager.persistAndFlush(city);

    City cityByName = cityRepository.findByName(city.getName());
    assertThat(cityByName).isEqualTo(city);

    City cityById = cityRepository.findById(city.getId()).orElse(other:null);
    assertThat(cityById).isNotNull();
    assertThat(cityById.getName()).isEqualTo(city.getName());
}

@Test
void findAllCitiesTest(){
    City city1 = new City(name:"Aveiro", state:"Aveiro", country:"Portugal");
    City city2 = new City(name:"Agueda", state:"Aveiro", country:"Portugal");
    City city3 = new City(name:"Espinho", state:"Aveiro", country:"Portugal");

    testEntityManager.persist(city1);
    testEntityManager.persist(city2);
    testEntityManager.persist(city3);
    testEntityManager.flush();

    List<City> cities = cityRepository.findAll();

    assertThat(cities).hasSize(expected:3).extracting(City::getName).containsOnly(city1.getName(), city2.getName(), city3.getN
}

```

Para os testes ao nível da classe **ServiceImplementation** utilizei **Mocks** e testei cada uma das funções implementadas, que vão buscar informação à API.

```

@ExtendWith(MockitoExtension.class)
public class ServiceTests {

    @Mock(lenient = true)
    private CityRepository cityRepository;

    @InjectMocks
    private ServiceImplementation serviceImplementation;

    @BeforeEach
    void setUp() throws InterruptedException {
        this.serviceImplementation = new ServiceImplementation();
        City c1 = new City(name:"Aveiro", state:"Aveiro", country:"Portugal");
        City c2 = new City(name:"Agueda", state:"Aveiro", country:"Portugal");
        ArrayList<City> cidades = new ArrayList<>();
        cidades.add(c1);
        cidades.add(c2);

        when(cityRepository.findByName(c1.getName())).thenReturn(c1);
        when(cityRepository.findByName(c2.getName())).thenReturn(c2);
        when(cityRepository.findAll()).thenReturn(cidades);
    }
}

```

```

@Test
void getCitiesTest() throws IOException, URISyntaxException {
    ArrayList<City> cities = new ArrayList<>();
    cities = serviceImplementation.getCities(country:"USA", state:"California");

    City city1 = new City(name:"Acalanes Ridge", state:"California", country:"USA");
    City city2 = new City(name:"Blythe", state:"California", country:"USA");

    assertThat(cities.contains(city1));
    assertThat(cities.contains(city2));
}

@Test
void getStatesTest() throws IOException, URISyntaxException {
    ArrayList<String> states = new ArrayList<>();
    states = serviceImplementation.getStates(country:"Portugal");

    assertThat(states.contains(o:"Braga"));
    assertThat(states.contains(o:"Lisbon"));
}

@Test
void getCountriesTest() throws IOException, URISyntaxException {
    ArrayList<String> countries = new ArrayList<>();
    countries = serviceImplementation.getCountries();

    assertThat(countries.contains(o:"Gabon"));
    assertThat(countries.contains(o:"Georgia"));
    assertThat(countries.contains(o:"Poland"));
    assertThat(countries.contains(o:"Spain"));
}

@Test
void getCityDataTest() throws IOException, URISyntaxException {
    City foundCityAPI = serviceImplementation.getCityData(country:"USA", state:"California", city:"Los Angeles");

    assertThat(foundCityAPI.getName().isEqualTo(expected:"Los Angeles"));
    assertThat(foundCityAPI.getState().isEqualTo(expected:"California"));
    assertThat(foundCityAPI.getCountry().isEqualTo(expected:"USA"));
}

```

Avançando para os **testes de integração**, onde foi utilizado o SpringBoot MockMvc, eu testei cada “endpoint” que foi criado. Em cada um dos testes criei uma declaração “given”, associada a uma resposta do serviço e, de seguida, fiz um pedido “get” para o “endpoint” para verificar se a conexão sucedeu ou não.

```

@WebMvcTest(CityRestController.class)
class CityControllerTests {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private ServiceImplementation service;

    @BeforeEach
    void setUp(){
    }

    @Test
    void getAllCitiesTest() throws Exception {
        ArrayList<City> cities = new ArrayList<>();
        cities.add(new City(name:"Alamo", state:"California", country:"USA"));
        cities.add(new City(name:"Arden-Arcade", state:"California", country:"USA"));

        given(service.getCities(country:"USA", state:"California")).willReturn(cities);

        mvc.perform(get(urlTemplate:"/api/list/{country}/{state}", ..uriVariables:"USA", "California").contentType(MediaType.APPLICATION_JSON)
            .andExpect(jsonPath(expression:"$[0]", is(value:"Alamo"))).andExpect(jsonPath(expression:"$[1]", is(value:"Arden-Arcade"))));
    }
}

```

```

@Test
void getAllStatesTest() throws Exception {
    ArrayList<String> states = new ArrayList<>();
    states.add(e:"Asturias");
    states.add(e:"Catalunya");
    states.add(e:"Ceuta");

    given(service.getStates(country:"Spain")).willReturn(states);

    mvc.perform(get(urlTemplate:"/api/list/{country}", ...uriVariables:"Spain").contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath(expression:"$[0]", is(value:"Asturias"))).andExpect(jsonPath(expression:"$[1]", is(value:"Catal"))
}

@Test
void getCityDataTest() throws Exception {
    City city = new City(name:"Alamo", state:"California", country:"USA");

    given(service.getCityData(country:"USA", state:"California", city:"Alamo")).willReturn(city);

    mvc.perform(get(urlTemplate:"/api/{country}/{state}/{city}", ...uriVariables:"USA", "California", "Alamo").contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath(expression:"$.name", is(value:"Alamo")));
}

@Test
void getCacheDetailsTest() throws Exception {
    HashMap<String, Integer> cache = new HashMap<>();
    cache.put(key:"hits", value:5);
    cache.put(key:"misses", value:7);
    cache.put(key:"requests", value:2);

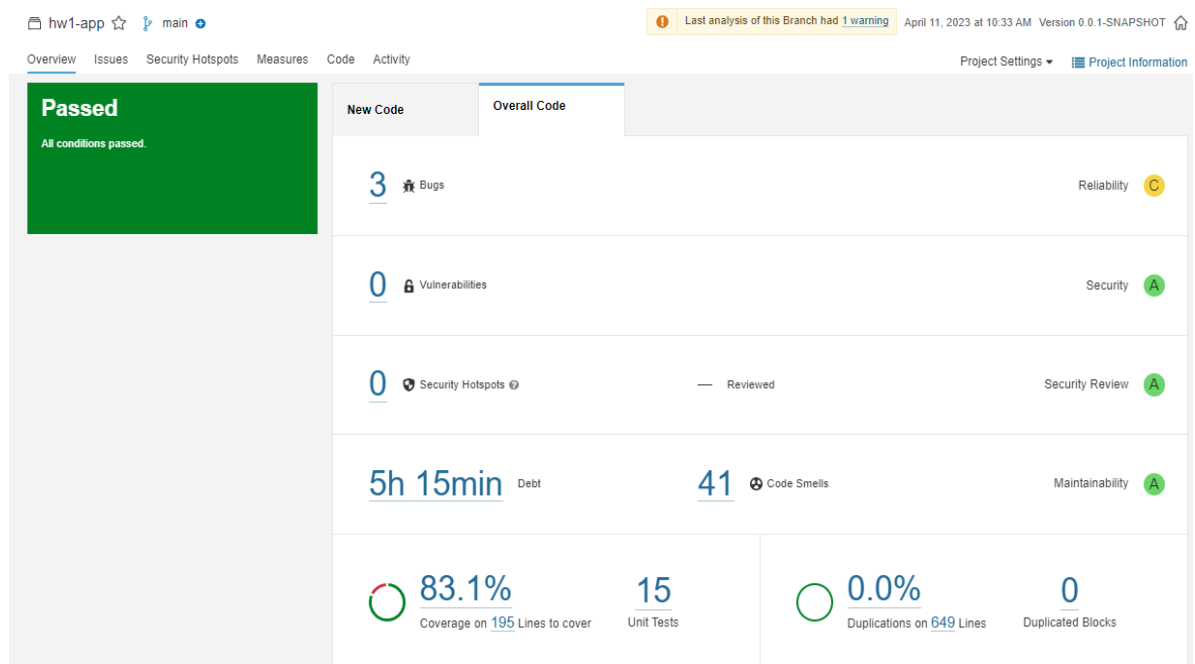
    given(service.getCacheDetails()).willReturn(cache);

    mvc.perform(get(urlTemplate:"/api/cache").contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath(expression:"$.hits", is(value:5)))
        .andExpect(jsonPath(expression:"$.misses", is(value:7)))
        .andExpect(jsonPath(expression:"$.requests", is(value:2)));
}

```

3.3 Code quality analysis

Utilizei a ferramenta **SonarQube** para realizar uma análise código e os resultados foram os seguintes:



4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/Manuel-Diaz17/tqs_103645
Video demo	Vídeo publicado no meu repositório

Reference materials

API Externa utilizada: <https://www.iqair.com/commercial-air-quality-monitors/api>