HW1: Mid-term assignment report

José Luca Baptista Pereira [97689], v2023-03-24

1 Introduction

1.1 Overview of the work

This report presents the midterm individual project required for TQS, covering both the software product features and the adopted quality assurance strategy.
The project uses IPMA's API for weather warnings and IPMA's to give information on weather hazards in the several Portuguese districts. A cache, unit and integration tests where implemented. The backend uses the Spring framework as required and the frontend is very simple using only HTML and JS.

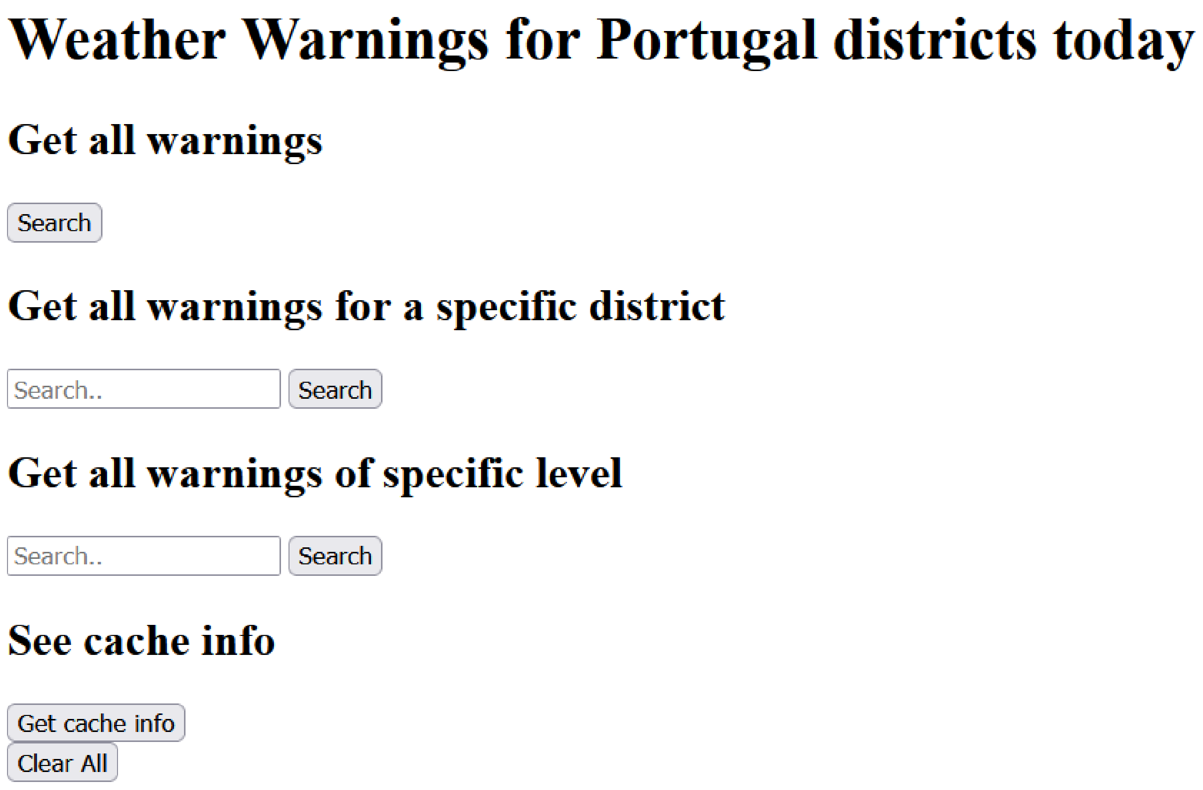
1.2 Current limitations

One of the limitations is the cache. It was implemented but is rather simple, this comes from the nature of the source of information, given that the warnings are updated every day and there are always the same number of them, its very easy to know when the information in the cache is up to date, and so the TTL for the cache information wasn´t applied to individual warnings but to the batch of all warnings once they are stored in the cache. This is a limitation because the cache can only be used when a call is made to get all the warnings.

Another limitation was the scope of the tests made. Unit and integration tests were made with very little problems, but the functional tests weren't made due to a lack of time.
The static code analysis with SonarQube was very problematic, trying to setup SonarQube took longer than developing the API itself and it wasn't possible to be made either.

1.3 Functional scope and supported interactions

The WebApp is very simple. The user will have two input boxes where it can query for a district name and get all the warnings for that district, or query for a severity level (green, yellow, orange or red) and get all the warnings of that severity.
It also has a button to get all the warnings and another to get the cache information (hits, misses and requests).


Weather Warnings for Portugal districts today

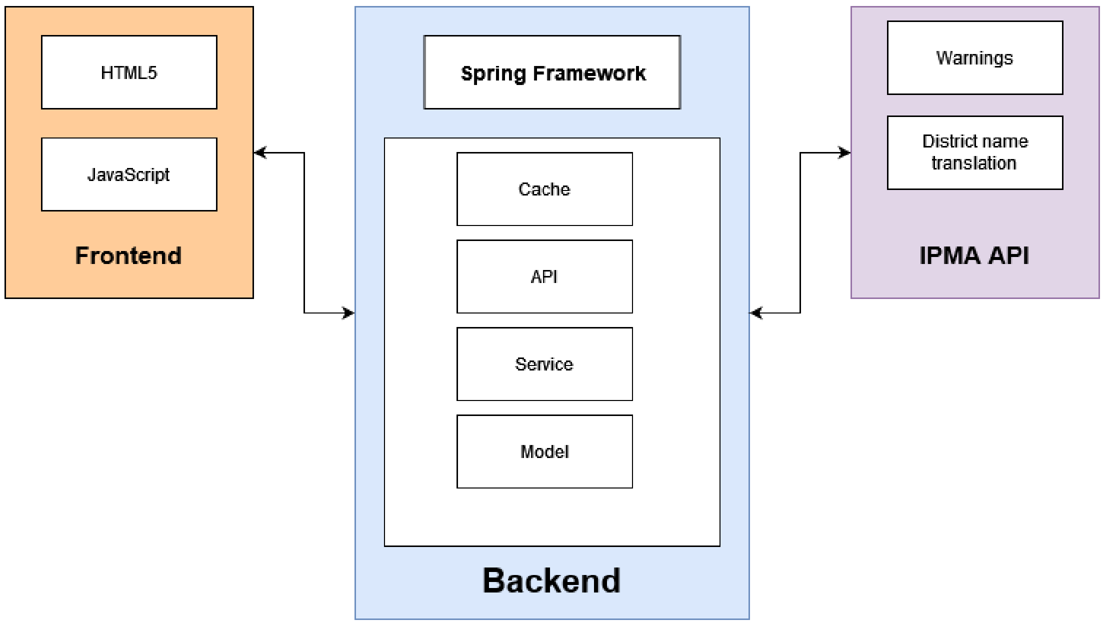
Get all warnings

Get all warnings for a specific district

Get all warnings of specific level

See cache info

1.4 System architecture



As said before the frontend is very simple and uses HTML and JavaScript to make the calls to our API. In API Spring was used as per required. We have a service that contacts both external APIs used. The main API to get the warnings and a secondary to translate the codes of the districts used in the first API into the actuals district names. There is also the cache, of which the limitations have been discussed and the RESTful API that allows calls to be made to fetch our data.

1.5 API for developers

The API allows for 5 different types of calls to be made:
Get all the warnings:
GET localhost:8080/all
Get all the warnings of a specific district:
GET localhost:8080/district/{district}
Get all the warnings of a certain severity:
GET localhost:8080/level/{level}
Get all the warnings of a certain type
GET localhost:8080/type/{type}
Get the cache information
GET

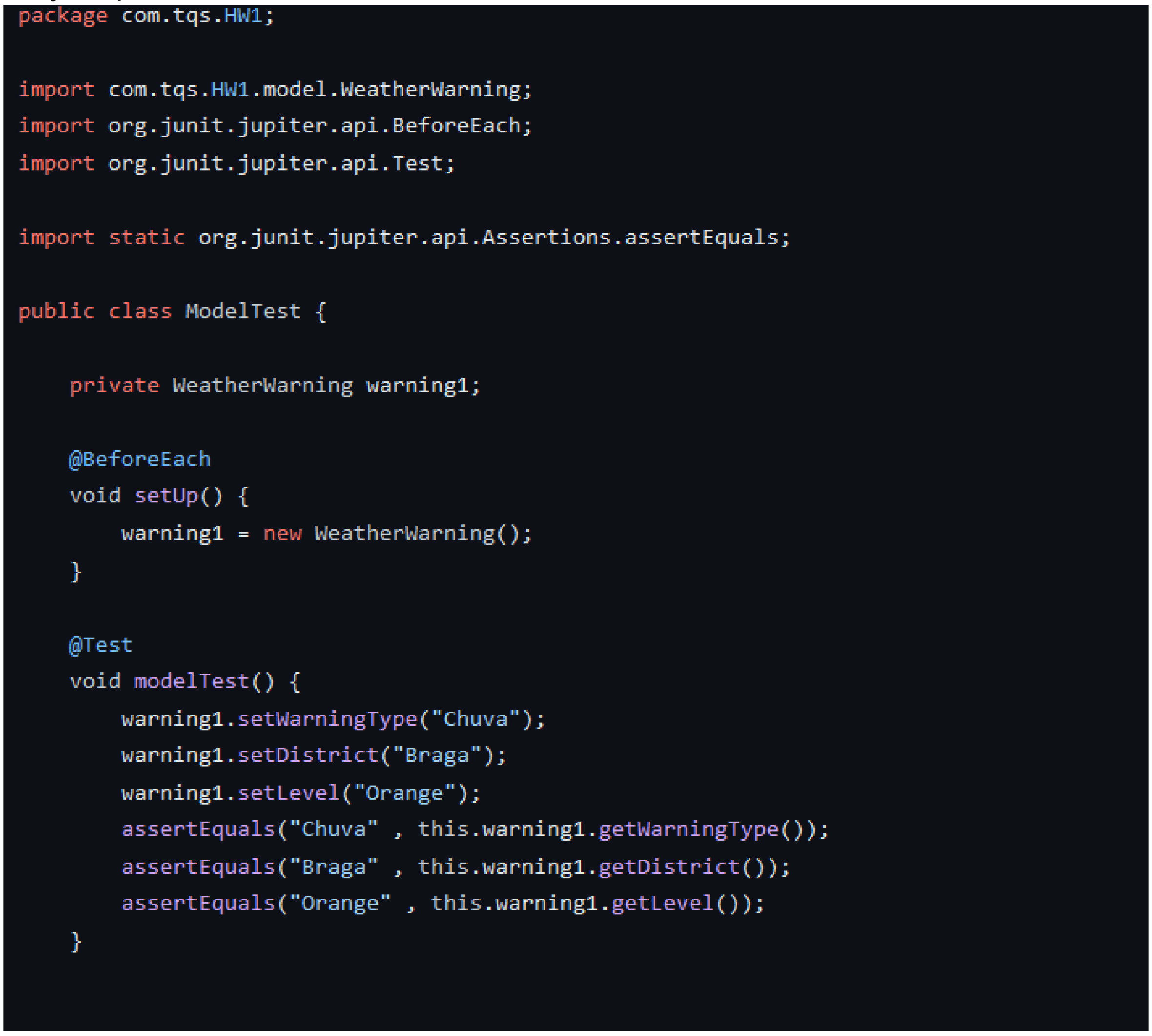
2 Quality assurance

2.1 Overall strategy for testing

The API was made before the test implementation so TDD was not used since the development goals where already fairly clear and there wasn’t an expectation for surprises to happen.
For unit testing Junit was used and integration tests were made using Spring Boot MockMVC. For functional tests there was an attempt to use cucumber and Selenium but there was no time as discussed in the limitations section. For the static code analysis SonarQube was attempted to be used but as also discussed there was extreme difficulty in the setup process.

2.2 Unit and integration testing

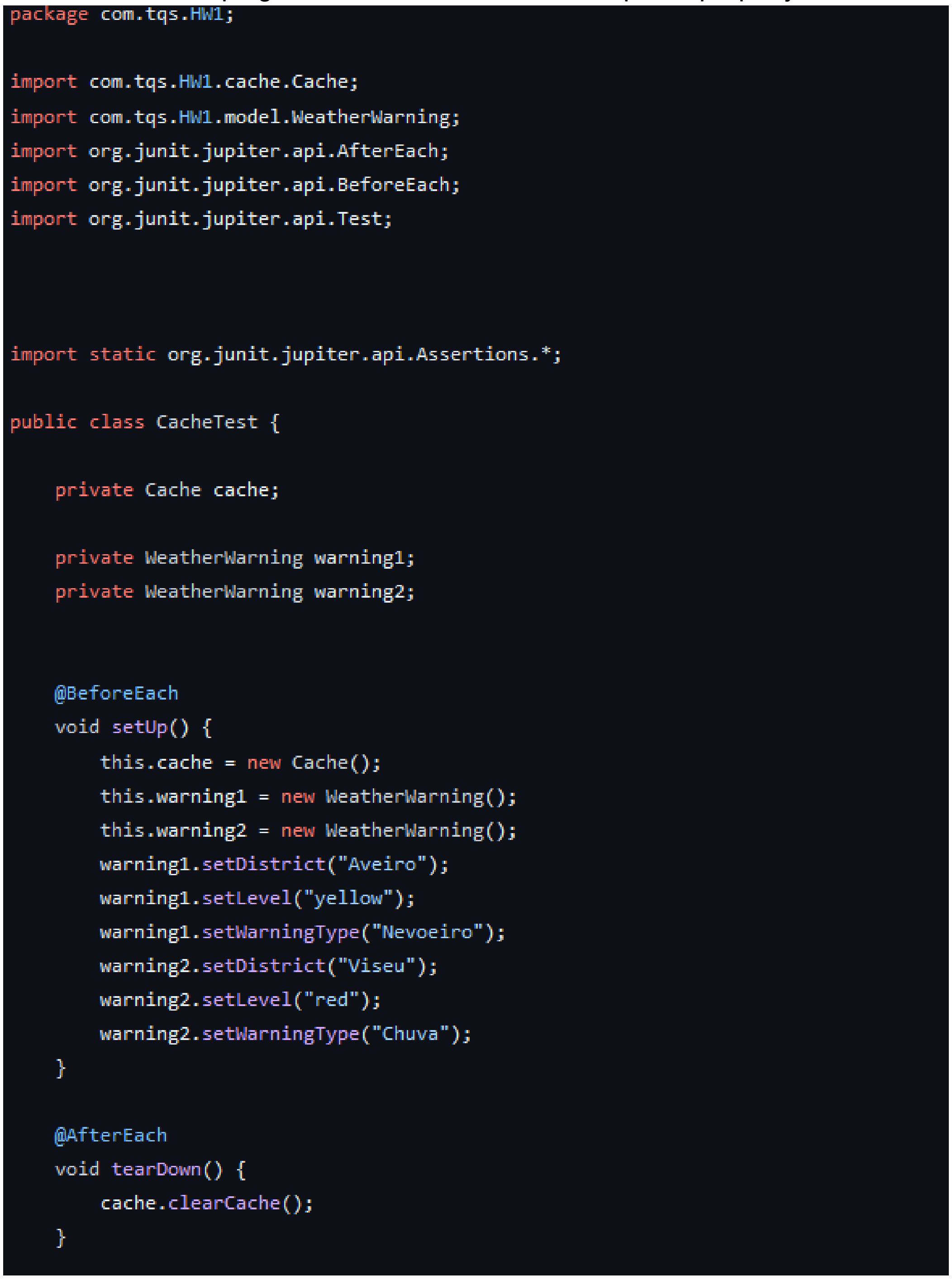
Unit test was used to test both the model and the cache.
To test the WeatherWarning.java (the model class) tests were performed to assure that its methods were working properly. Given that it was a very simple class the test where also fairly simple.



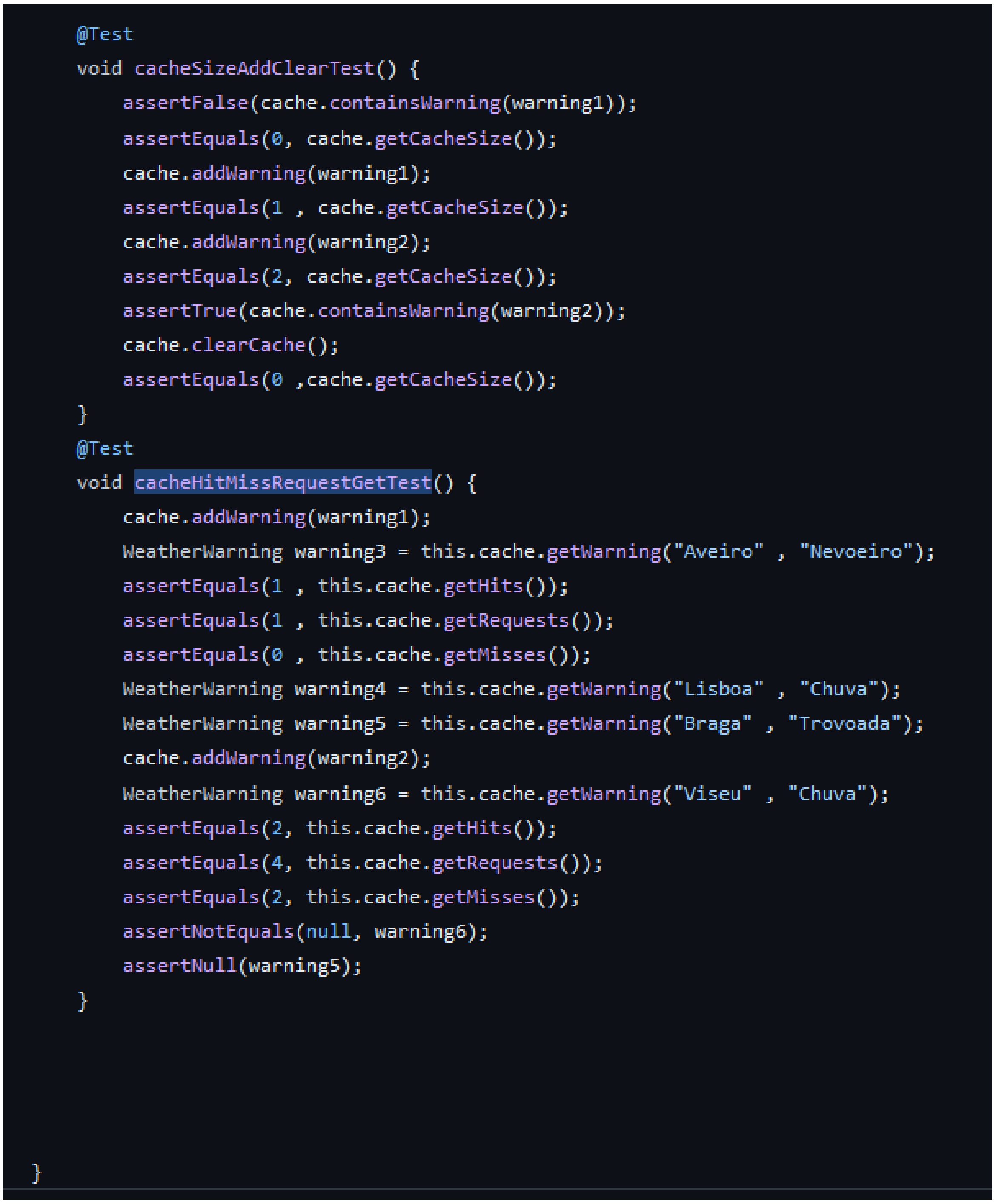
```java
package com.tqs.HW1;

import com.tqs.HW1.model.WeatherWarning;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class ModelTest {

    private WeatherWarning warning1;

    @BeforeEach
    void setUp() {
        warning1 = new WeatherWarning();
    }

    @Test
    void modelTest() {
        warning1.setWarningType("Chuva");
        warning1.setDistrict("Braga");
        warning1.setLevel("Orange");
        assertEquals("Chuva" , this.warning1.getWarningType());
        assertEquals("Braga" , this.warning1.getDistrict());
        assertEquals("Orange" , this.warning1.getLevel());
    }
```

ModelTest.java

The cache methods where also tested to ensure they behaved accordingly to the expectations given that it was a crucial part of the project since if not working properly the results of the searches and calls to the API could be corrupted.
Two tests where made the first **cacheSizeAddClearTest()** was made to assure that the cache returned its size properly, very important since its size determines whether it can be used to fetch data. The other test **cacheHitMissRequestGetTest()** was used to ensure that the cache was keeping track of its hit, misses and requests properly.



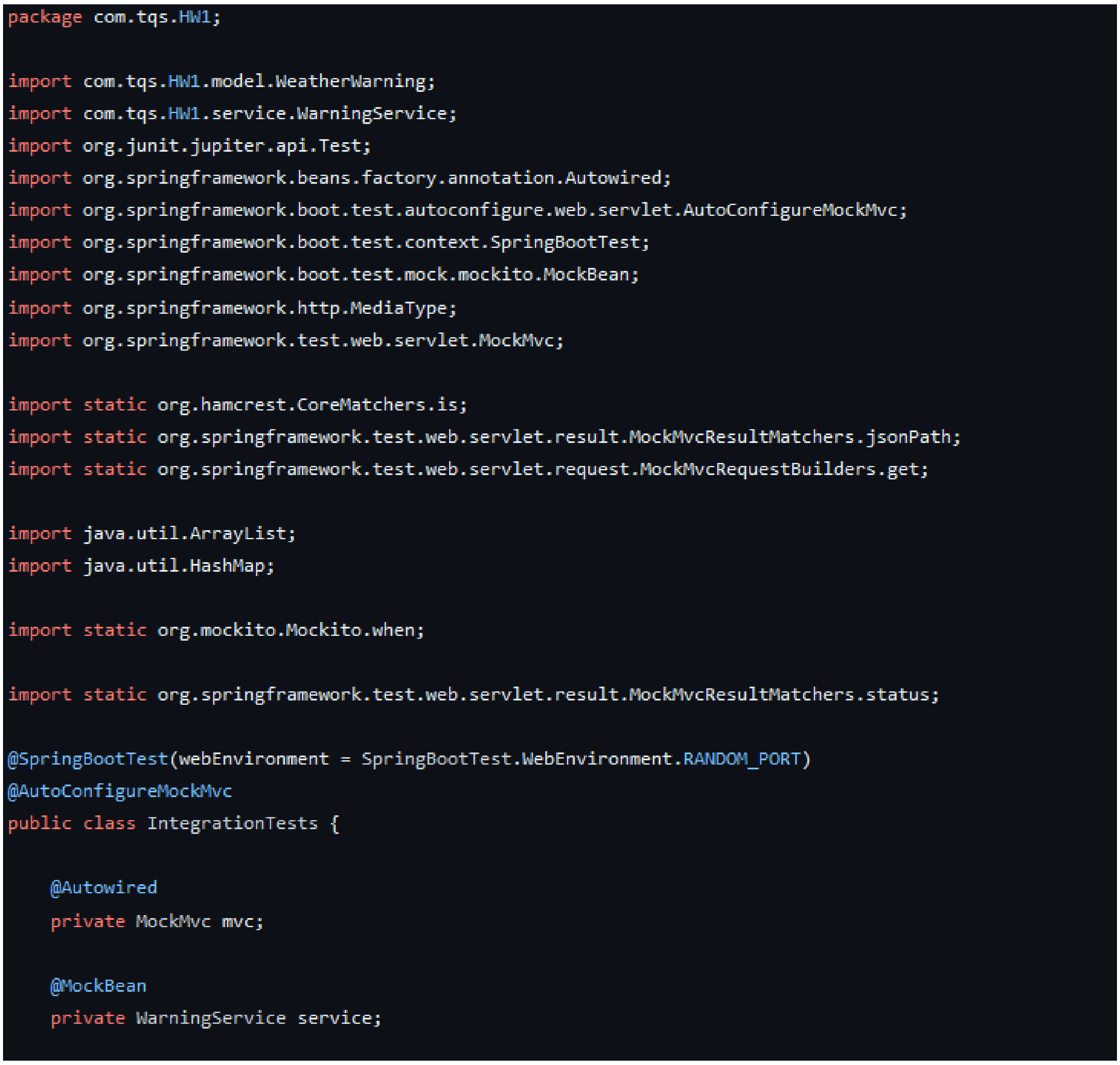
```java
package com.tqs.HW1;

import com.tqs.HW1.cache.Cache;
import com.tqs.HW1.model.WeatherWarning;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;



import static org.junit.jupiter.api.Assertions.*;

public class CacheTest {

    private Cache cache;

    private WeatherWarning warning1;
    private WeatherWarning warning2;


    @BeforeEach
    void setUp() {
        this.cache = new Cache();
        this.warning1 = new WeatherWarning();
        this.warning2 = new WeatherWarning();
        warning1.setDistrict("Aveiro");
        warning1.setLevel("yellow");
        warning1.setWarningType("Nevoeiro");
        warning2.setDistrict("Viseu");
        warning2.setLevel("red");
        warning2.setWarningType("Chuva");
    }

    @AfterEach
    void tearDown() {
        cache.clearCache();
    }
```



```
    @Test
    void cacheSizeAddClearTest() {
        assertFalse(cache.containsWarning(warning1));
        assertEquals(0, cache.getCacheSize());
        cache.addWarning(warning1);
        assertEquals(1 , cache.getCacheSize());
        cache.addWarning(warning2);
        assertEquals(2, cache.getCacheSize());
        assertTrue(cache.containsWarning(warning2));
        cache.clearCache();
        assertEquals(0 ,cache.getCacheSize());
    }
    @Test
    void cacheHitMissRequestGetTest() {
        cache.addWarning(warning1);
        WeatherWarning warning3 = this.cache.getWarning("Aveiro" , "Nevoeiro");
        assertEquals(1 , this.cache.getHits());
        assertEquals(1 , this.cache.getRequests());
        assertEquals(0 , this.cache.getMisses());
        WeatherWarning warning4 = this.cache.getWarning("Lisboa" , "Chuva");
        WeatherWarning warning5 = this.cache.getWarning("Braga" , "Trovoada");
        cache.addWarning(warning2);
        WeatherWarning warning6 = this.cache.getWarning("Viseu" , "Chuva");
        assertEquals(2, this.cache.getHits());
        assertEquals(4, this.cache.getRequests());
        assertEquals(2, this.cache.getMisses());
        assertNotEquals(null, warning6);
        assertNull(warning5);
    }




}
```

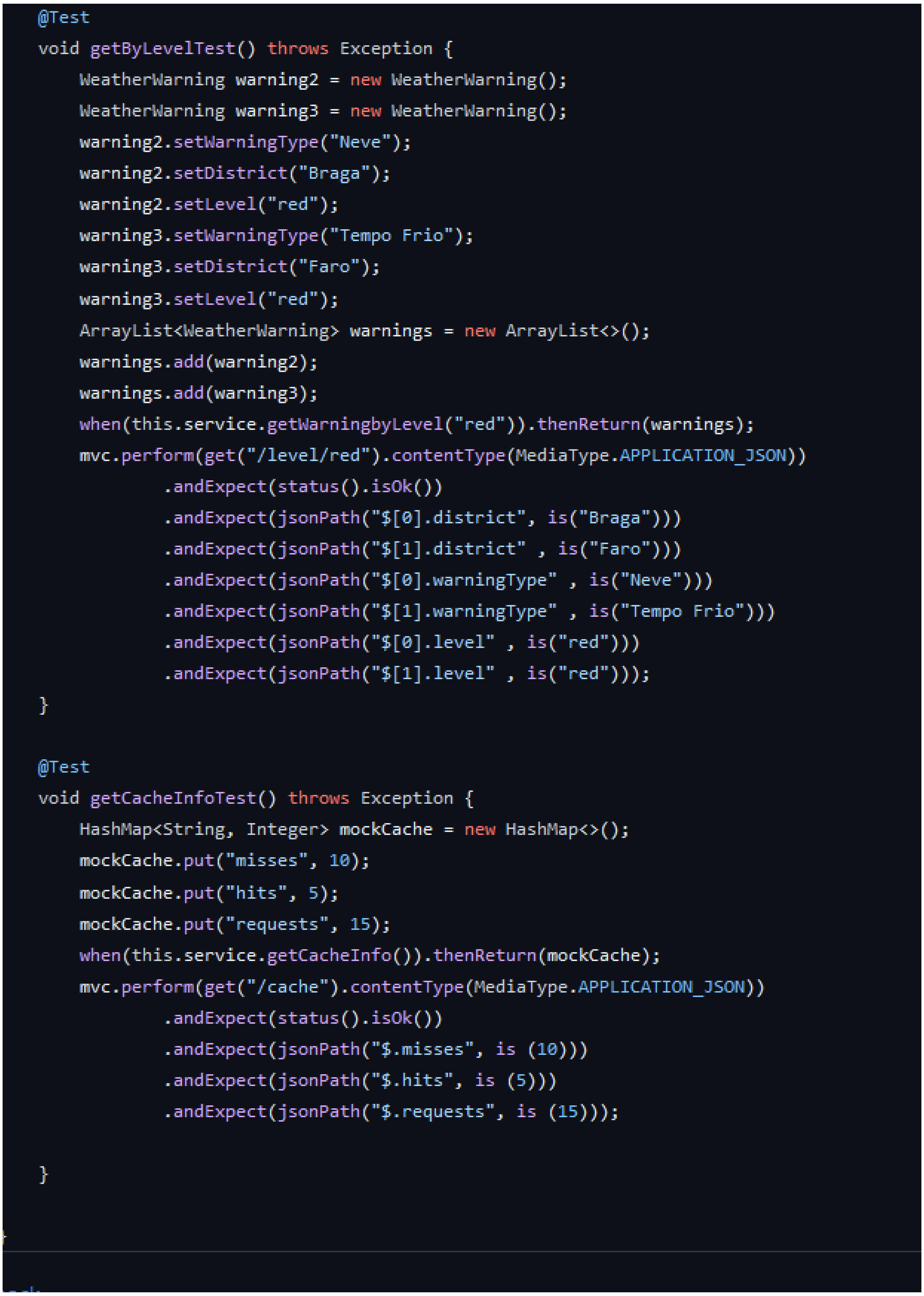
CacheTest.java

The integration tests were made using SpringBoot MockMvc. This was sued to test the endpoints of the API and the service methods.
There were made four tests. **getAllWarningsTests()** to test the endpoint to get all the warnings, **getByDistrictTest()** to test the endpoint to get warning by district, **getByLevelTest(),** to test the get by severity endpoint, and **getCacheInfoTest()** to test the endpoint where the information of the cache is fetched.



```java
package com.tqs.HW1;

import com.tqs.HW1.model.WeatherWarning;
import com.tqs.HW1.service.WarningService;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

import static org.hamcrest.CoreMatchers.is;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import java.util.ArrayList;
import java.util.HashMap;

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureMockMvc
public class IntegrationTests {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private WarningService service;
```

```java
@Test
void getAllWarningsTests() throws Exception {
    WeatherWarning warning = new WeatherWarning();
    warning.setWarningType("Chuva");
    warning.setDistrict("Aveiro");
    warning.setLevel("Orange");
    ArrayList<WeatherWarning> warnings = new ArrayList<>();
    warnings.add(warning);
    when(this.service.getAllWarnings()).thenReturn(warnings);
    mvc.perform(get("/all").contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].district", is("Aveiro")))
            .andExpect(jsonPath("$[0].warningType", is("Chuva")))
            .andExpect(jsonPath("$[0].level", is("Orange")));


}

@Test
void getByDistrictTest() throws Exception {
    WeatherWarning warning2 = new WeatherWarning();
    WeatherWarning warning3 = new WeatherWarning();
    warning2.setWarningType("Chuva");
    warning2.setDistrict("Guarda");
    warning2.setLevel("orange");
    warning3.setWarningType("Nevoeiro");
    warning3.setDistrict("Guarda");
    warning3.setLevel("red");
    ArrayList<WeatherWarning> warnings = new ArrayList<>();
    warnings.add(warning2);
    warnings.add(warning3);
    when(this.service.getDistrictWarnings("Guarda")).thenReturn(warnings);
    mvc.perform(get("/district/Guarda").contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].district", is("Guarda")))
            .andExpect(jsonPath("$[1].district" , is("Guarda")))
            .andExpect(jsonPath("$[0].warningType" , is("Chuva")))
            .andExpect(jsonPath("$[1].warningType" , is("Nevoeiro")))
            .andExpect(jsonPath("$[0].level" , is("orange")))
            .andExpect(jsonPath("$[1].level" , is("red")));

}
```



```java
    @Test
    void getByLevelTest() throws Exception {
        WeatherWarning warning2 = new WeatherWarning();
        WeatherWarning warning3 = new WeatherWarning();
        warning2.setWarningType("Neve");
        warning2.setDistrict("Braga");
        warning2.setLevel("red");
        warning3.setWarningType("Tempo Frio");
        warning3.setDistrict("Faro");
        warning3.setLevel("red");
        ArrayList<WeatherWarning> warnings = new ArrayList<>();
        warnings.add(warning2);
        warnings.add(warning3);
        when(this.service.getWarningbyLevel("red")).thenReturn(warnings);
        mvc.perform(get("/level/red").contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$[0].district", is("Braga")))
                .andExpect(jsonPath("$[1].district" , is("Faro")))
                .andExpect(jsonPath("$[0].warningType" , is("Neve")))
                .andExpect(jsonPath("$[1].warningType" , is("Tempo Frio")))
                .andExpect(jsonPath("$[0].level" , is("red")))
                .andExpect(jsonPath("$[1].level" , is("red")));
    }

    @Test
    void getCacheInfoTest() throws Exception {
        HashMap<String, Integer> mockCache = new HashMap<>();
        mockCache.put("misses", 10);
        mockCache.put("hits", 5);
        mockCache.put("requests", 15);
        when(this.service.getCacheInfo()).thenReturn(mockCache);
        mvc.perform(get("/cache").contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.misses", is (10)))
                .andExpect(jsonPath("$.hits", is (5)))
                .andExpect(jsonPath("$.requests", is (15)));

    }
```

IntegrationTests.java

2.3 Functional testing

Unfortunately, no functional tests were made as discussed in the limitations section.

2.4 Code quality analysis

Unfortunately, no static code analysis was made as discussed in the limitations section.

3 References & resources

Project resources

Resource:	**URL/location:**
Git repository	https://github.com/LucaKnowsStuff/TQS_97689/tree/main/HW1
Video demo	https://github.com/LucaKnowsStuff/TQS_97689/tree/main/HW1/Screencast from 11-04-2023 10:49:15.webm **and** https://github.com/LucaKnowsStuff/TQS_97689/tree/main/HW1/Screencast from 11-04-2023 10:48:17.webm

Reference materials

IPMA API page: https://api.ipma.pt/
IPMA weather warning API: https://api.ipma.pt/open-data/forecast/warnings/warnings_www.json
IPMA list of identifiers for districts and autonomous islands: https://api.ipma.pt/open-data/distrits-islands.json