HW1: Mid-term assignment report

Ana Alexandra Antunes [876543], v2020-03-27

1 Introduction 1
1.1 Overview of the work 1
1.2 Limitations 1

2 Product specification 1
2.1 Functional scope and supported interactions 1
2.2 System architecture 2
2.3 API for developers 2

3 Quality assurance 2
3.1 Overall strategy for testing 2
3.2 Unit and integration testing 2
3.3 Functional testing 2
3.4 Static code analysis 2
3.5 Continuous integration pipeline [optional] 2

4 References & resources 3

1 Introduction

1.1 Overview of the work

This project objective is to implement a very simple Spring Boot API to fetch air quality data from an external source, but, more important, implement tests and quality methods to test the product.

1.2 Limitations

The current project does not have a database to support, just an in memory cache to store data. The problem is, when the server is disconnected, the stored data is lost.

The current state of mock tests is very limited. In the future, more complex tests should be implemented.

Another planned feature was a message broker, to deal with multiple API calls from time to time and store the data properly.

2 Product specification

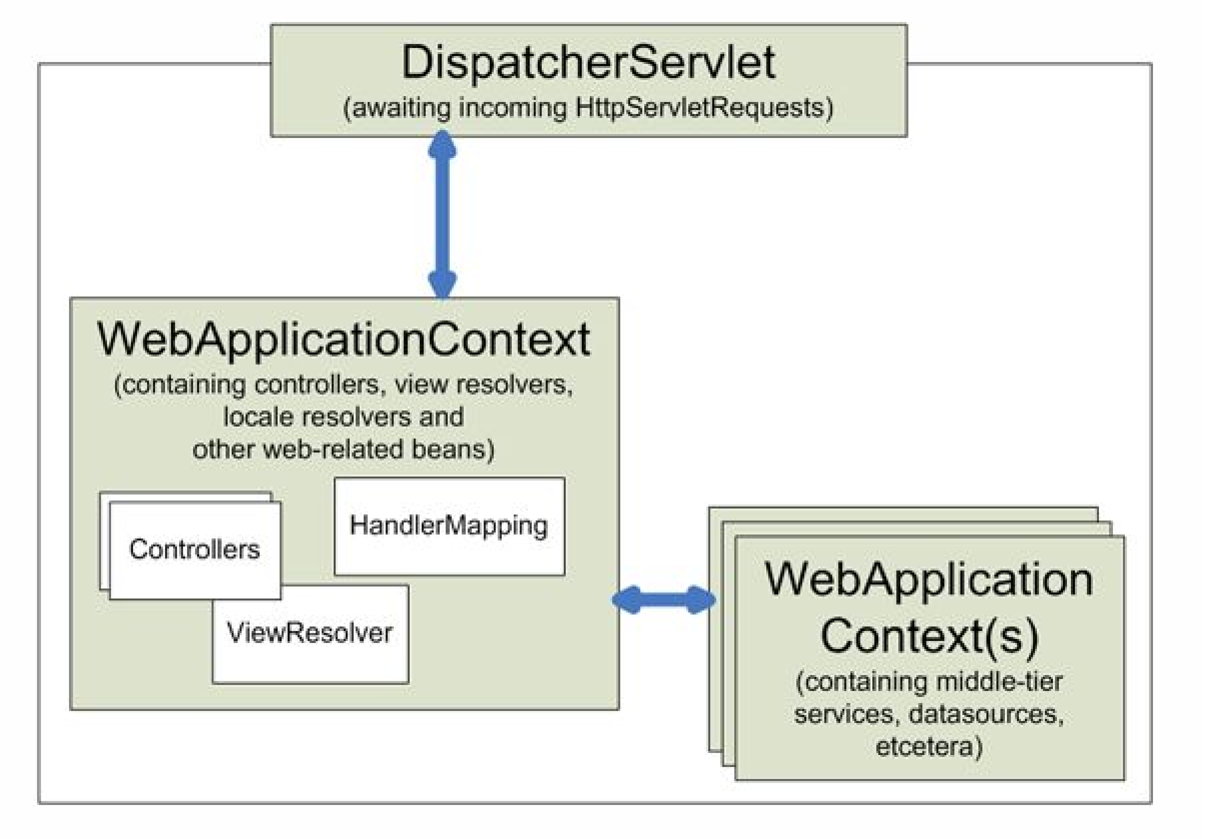
2.1 Functional scope and supported interactions

The application is a simple web page that fetch air quality and weather data from an external source[1]. It is designed for everyone that wants/needs to know air quality and weather for any city.

The user enter the country, state and city desired and the application returns the data. If the user request data that already was requested before, the application will fetch data from de memory cache, and will not make another API call.

2.2 System architecture

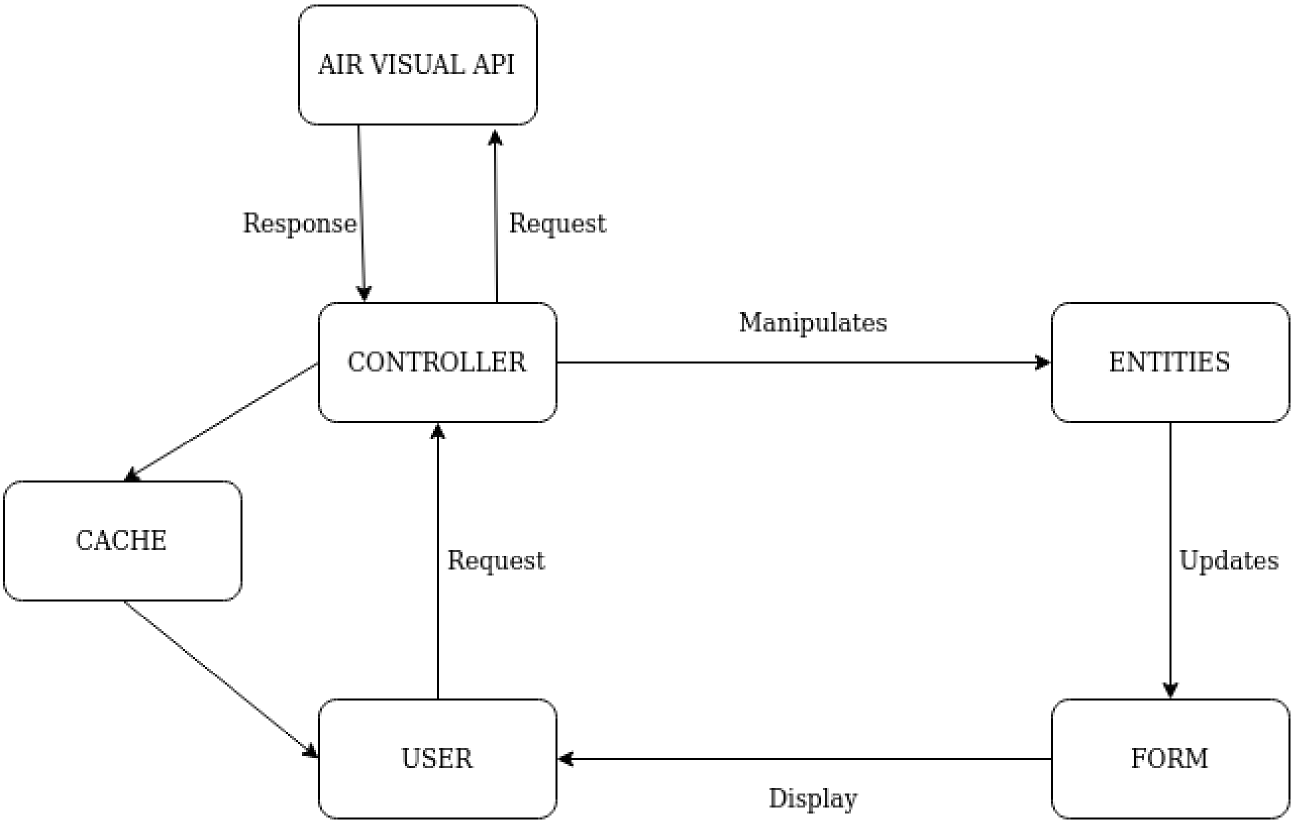
This project is based on the Web MVC architecture.


DispatcherServlet
(awaiting incoming HttpServletRequests)
WebApplicationContext
(containing controllers, view resolvers, locale resolvers and other web-related beans)
Controllers
HandlerMapping
ViewResolver
WebApplication Context(s)
(containing middle-tier services, datasources, etcetera)

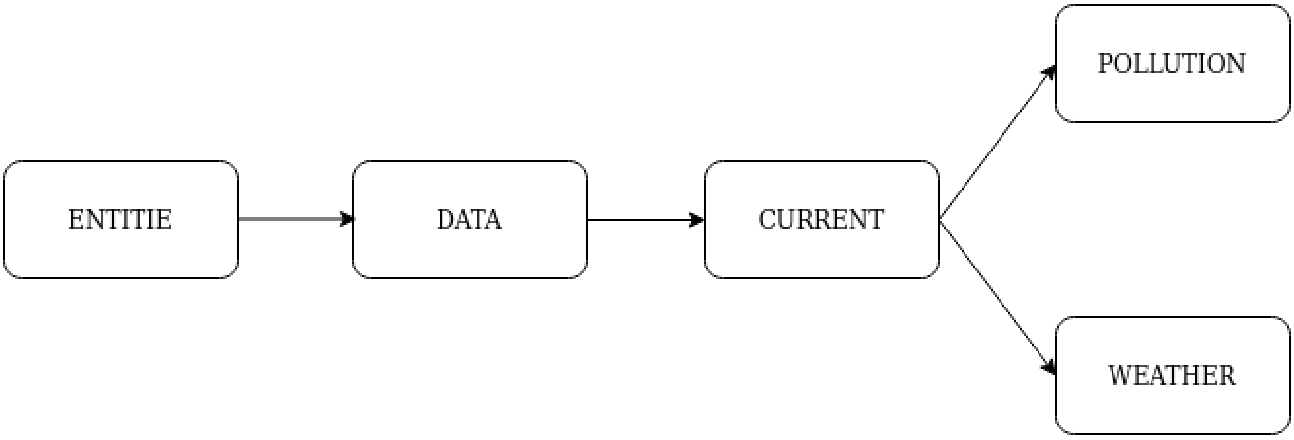
https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html

We have 4 modules in this project:

- The *app* module: this is the controller of our application. It handles all the calls to the API, also, stores data into the memory cache.
- The *entities* module: this is where the data is modeled. Contains classes that provides getters/setters and the data structure for the project.
- The *form* module: here we find the class to support the access into the data by frontend.
- The *cache* module: this is the class to support the in memory cache of the implementation. It has methods to get, put and delete data from the cache. The cache perse is a synchronizedMap.



Architecture design



Model design

Technologies used:

2.3 API for developers

https://app.swaggerhub.com/apis-docs/viniciusbenite/your-api/1.0.0#/

3 Quality assurance

3.1 Overall strategy for testing

The strategy for testing this project was to: mainly use of unit testing using JUnit, Mockito and REST-Assured tests. Also, the use of Selenium Webdriver to test the user interface.

3.2 Unit and integration testing

The tests cases include web layer tests, namely check if the controller is being loaded and check the greeting message of the web page, using Junit, assertions and SpringBoot Test.

```
@Test
public void contextLoads() throws Exception{
  assertThat(controller).isNotNull();
}
```

Also, this project has unit tests for check the API call, the initiation of the memory cache, the insertion of data into the memory cache, the retrieve of data from the memory cache and the expire time of the objects in the cache.

```
@Test
public void memCachePutTest() throws Exception {
    // given
    MemoryCache<Entitie> memCache = new MemoryCache<>();
    Form form = new Form();
    form.setCity("Aveiro");
    form.setCountry("Portugal");
    form.setState("Aveiro");
    Entitie et = restTemplate.getForObject("http://api.airvisual.com/v2/city?city="+ form.getCity() +
"&state=" + form.getState() + "&country="+ form.getCountry()
+"&key=2fa6f39e-db55-44e5-8efa-cf202e452a4b", Entitie.class);
    memCache.put(form.getCity(), et);
    // when
    Entitie getMemCacheObj = memCache.get("Aveiro");
    // then
    assertEquals("Entitie object expected: ", et, getMemCacheObj);
}
@Test
public void memCacheGetTest() throws Exception {
    // given
    MemoryCache<Entitie> memCache = new MemoryCache<>();
    Form form = new Form();
    form.setCity("Aveiro");
    form.setCountry("Portugal");
    form.setState("Aveiro");
    Entitie et = restTemplate.getForObject("http://api.airvisual.com/v2/city?city="+ form.getCity() +
"&state=" + form.getState() + "&country="+ form.getCountry()
+"&key=2fa6f39e-db55-44e5-8efa-cf202e452a4b", Entitie.class);
    memCache.put(form.getCity(), et);
    // when
    Entitie response = memCache.get("Aveiro");
    // then
    assertEquals("Results from API call and cache must be the same", response, et);
}
```

Here, the objects putted into the memory cache has to be identical to the API call.
The same tests were done using Mockito, as well.

3.3 Functional testing

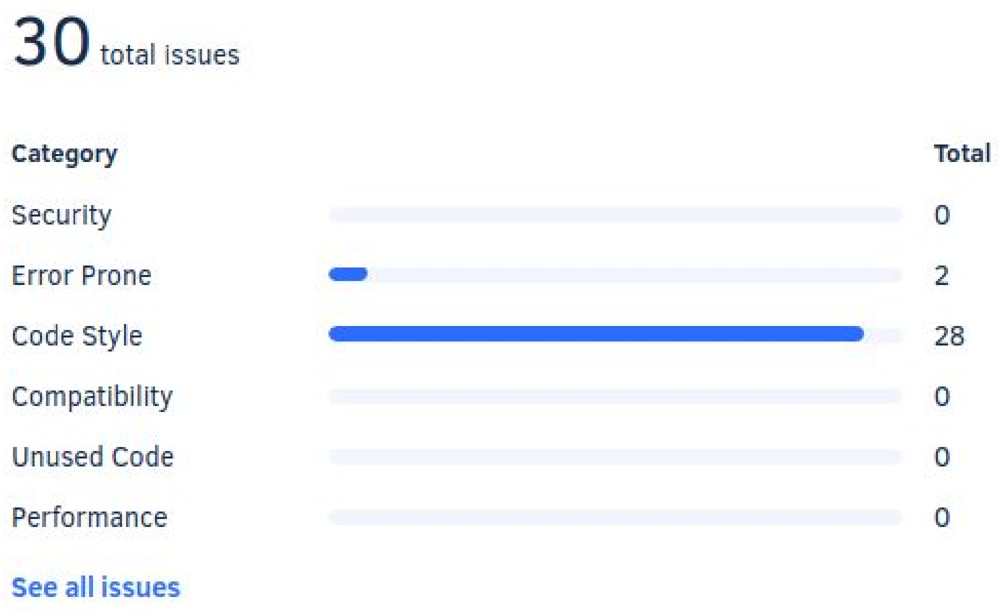
[which test cases did you considered? How were they implemented?]
[may add screenshots/code snippets]
For functional testing, this project used Selenium Webdriver. We tested if the user got the correct that that he desired, if he could go back and search for another desired city properly and if the user got an error message when invalid input was given to the form.

```
@Test
public void testFormInput() {
    driver.get("http://localhost:8080/");
    FormInputPage formPage = new FormInputPage(driver);
    ResultPage resultPage = formPage.validSearch("Brazil", "Sao Paulo", "Pederneiras");
    assertThat(resultPage.getMessageText(), is("Air Quality Data"));
    assertThat(resultPage.getCityText(), is("City: Pederneiras"));
}
```

Worth to mention that, for the Selenium test, the project used the Page Object Model (POM) to structure the test.

3.4 Static code analysis

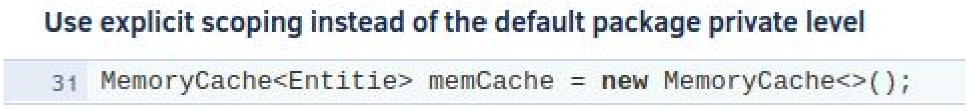
For static code analysis I used Codacy.



30 total issues

Category	Total
Security	0
Error Prone	2
Code Style	28
Compatibility	0
Unused Code	0
Performance	0

The main problems found was unused imports and the use of explicit scoping instead of the default package, such as:



Use explicit scoping instead of the default package private level

```
MemoryCache<Entitie> memCache = new MemoryCache<>();
```

4 References & resources

Project resources

- Git repository: https://github.com/viniciusbenite/tqsproject1
- Video demo in Git repository.

Reference materials

[1] https://www.iqair.com/
https://docs.spring.io/
https://www.selenium.dev/documentation/en/
https://junit.org/junit5/docs/current/user-guide/
https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito
TQS course materials.