1- Problem Definition

Create an API that will retrieve weather metrics of a specific city.

2- Execution

This program is written using spring boot and can be run on any JVM (1.8+). It exposes some REST API to get weather condition forecast information digest for a given city.

It employs a token based authentication. Therefore, in order to call any other service, you must login before and get a token. Then the token must be sent as a header parameter in every request.

2-1- Login API

The API documentation is available online here on postman.

Endpoint: host/api/v1/login

Credentials must be sent as a JSON body. Please don't forget to set the media type to application/json. you can login using the default account:

```
{
  "username": "admin",
  "password": "admin"
}
```

After successful authentication it returns a token like this:

```
{
  "token": "e26450c1-5673-4c81-8e61-50c8d7d772f2"
}
```

2-2- Data API

Endpoint: host/api/v1/data

This service provides the weather forecast information digest given a city as query parameter. Keep in mind that the token must be set in the header. Example:

Input:

token:56caaf8d-ff87-4fc8-ad78-e130bd3726c5

host/api/v1/data?city=Berlin

Output:

```
{
  "city": "Berlin",
    "country": "DE",
    "averageTemperatureDayTime3Days": 7.61,
    "averageTemperatureNightTime3Days": 6.09,
    "averagePressure3Days": 1004.46
}
```

3- Architecture

The below figure demonstrates the overall architecture of this application. Users send their request via REST API to the system. Then OpenWeatherService component talks to the original weather server and gets the raw results. The raw results are then fed into a DataParser component which processes the given information and returns a sanitized version of weather information. Finally, the result is returned to the user.

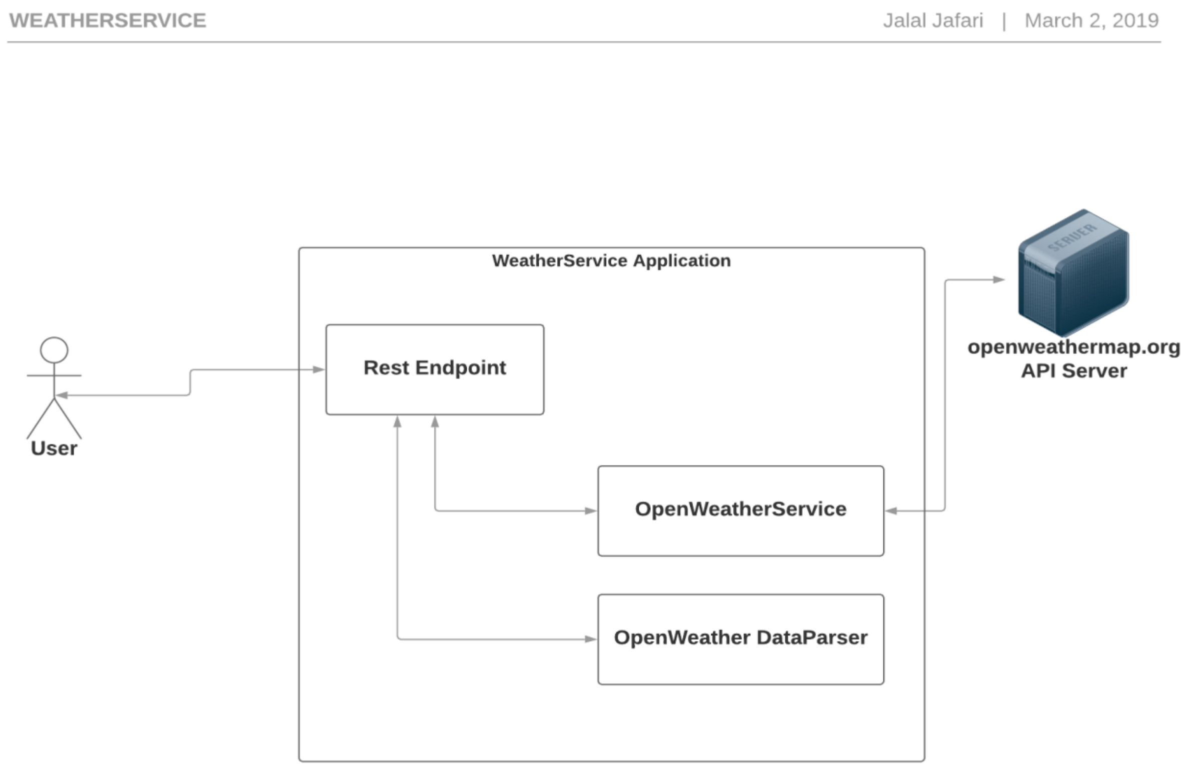


Figure 1 – Architecture

4- Design

In this section design decisions and the main purpose of core objects is explained.

4-1- OpenWeather API Layer

This layer is responsible to communicate with external weather service and parse the related data.

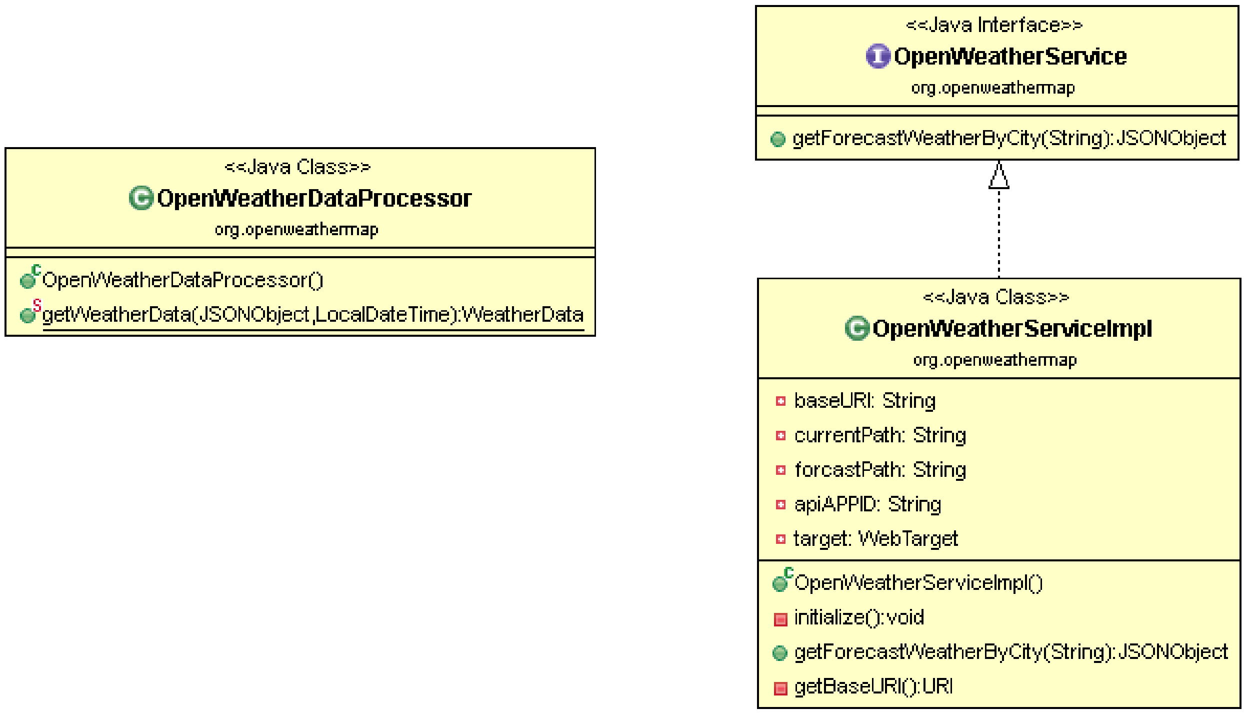


Figure 2- OpenWeatherService Layer Objects

1-1-1- OpenWeatherService

This is an interface which provides common methods to connect to openwaethermap.org API. Its implementation class (OpenWeatherServiceImpl) provides the implementation details. This class simply plays the role of a driver for the external service and separates the communication logic from the core.

1-1-2- OpenWeatherDataProcessor

Since we are interested in a processed version of the data retrieved from the external API, we need a class to perform any required transformation on the data. This class is designed for this purpose. It currently has a method which takes the raw forecast JSON as input and returns a clean processed WeatherData object.

4-2- Domain Objects

Domain objects are described in this section.

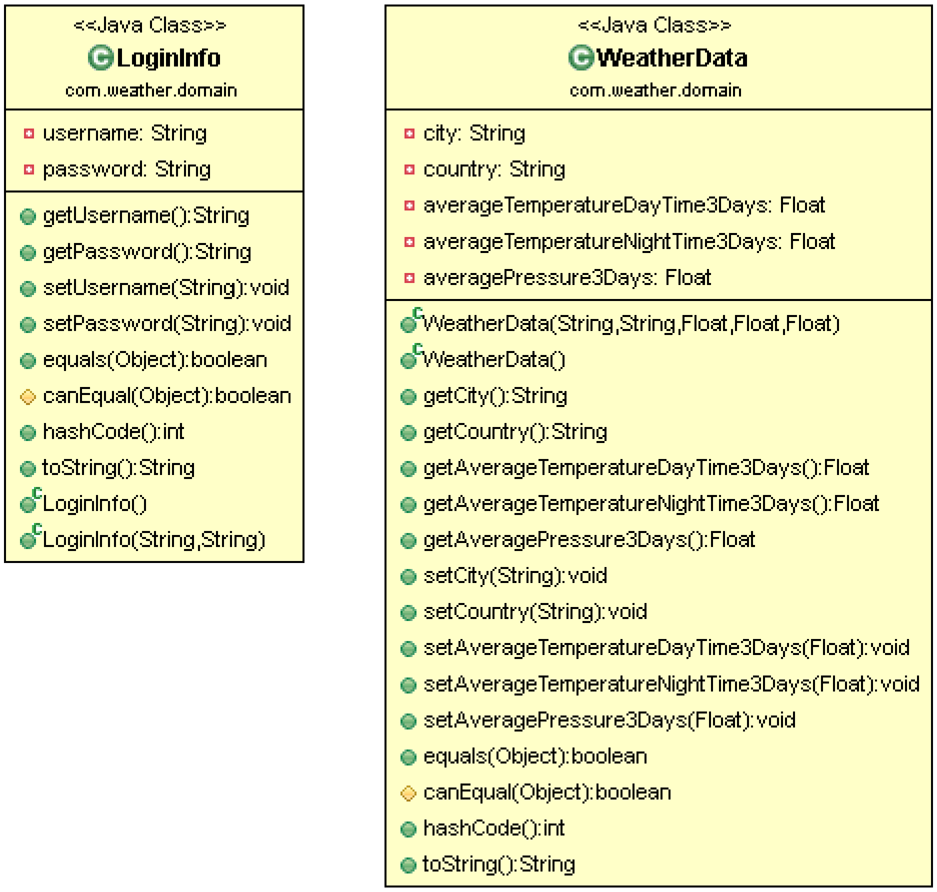


Figure 3- Domain objects

1-1-3- WeatherData

This is a knowing class to keep the desired and processed information about weather conditions.

1-1-4- LoginInfo

This is a knowing class to user credentials. It is mainly used to receive username and password in login process.

4-3- Controller Layer

Controller Layer which is responsible for handling REST requests is explained in this section.

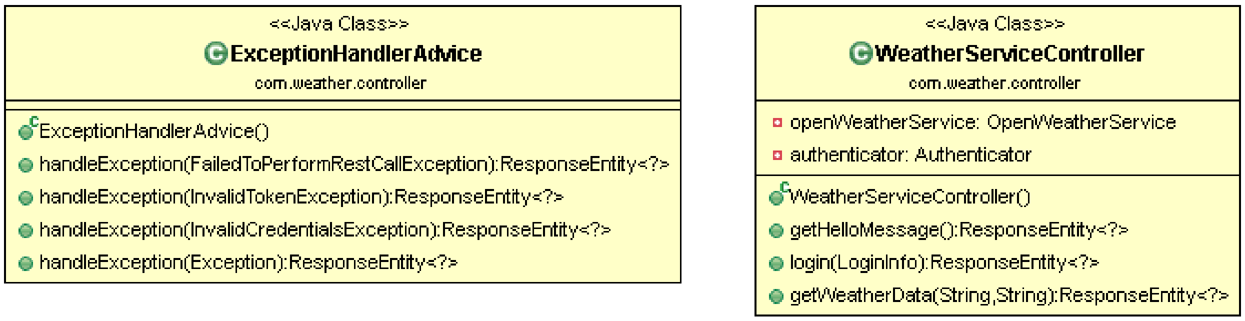

<<Java Class>>
ExceptionHandlerAdvice
com.weather.controller
ExceptionHandlerAdvice()
handleException(FailedToPerformRestCallException):ResponseEntity<?>
handleException(InvalidTokenException):ResponseEntity<?>
handleException(InvalidCredentialsException):ResponseEntity<?>
handleException(Exception):ResponseEntity<?>
<<Java Class>>
WeatherServiceController
com.weather.controller
openWeatherService: OpenWeatherService
authenticator: Authenticator
WeatherServiceController()
getHelloMessage():ResponseEntity<?>
login(LoginInfo):ResponseEntity<?>
getWeatherData(String,String):ResponseEntity<?>

Figure 4- Controller Layer

As part of the MVC pattern, the Rest Controller is designed to receive user requests, talk to the core and return a response. It currently offers two main methods: “login” and “data”. While the former’s functionality is obvious, the latter is used to retrieve the weather information for a given city as query parameter. In all services if the operation is successful a HttpStatus of OK is returned along with a JSON response.

In case of exception, a separate ExceptionHandlerAdvice is designed to keep the REST Controller codes clean (thanks to SpringBoot). ExceptionHandlerAdvice returns a proper JSON response and HttpStatus according to the occurred incident.

4-4- Authentication

In this application a simple, from scratch token based authentication approach is taken. AuthenticatorImpl class maintains a hash map of logged in users and their corresponding tokens. When a new user logs in with correct credentials the maps get updated. In all the REST calls isTokenValid is called to check user validity.

Please note that this approach is chosen here only for simplicity. More sophisticated schemes such as JWT, Basic Auth (not really a suitable approach for production) can be employed for this purpose. Spring security has a lot to offer.

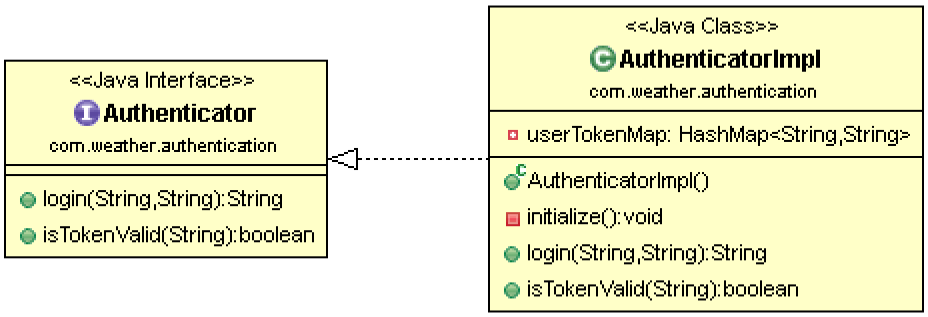

<<Java Interface>>
Authenticator
com.weather.authentication
login(String,String):String
isTokenValid(String):boolean
<<Java Class>>
AuthenticatorImpl
com.weather.authentication
userTokenMap: HashMap<String,String>
AuthenticatorImpl()
initialize():void
login(String,String):String
isTokenValid(String):boolean

Figure 5- Authentication

4-5- Exceptions

While most of the custom exception classes keep only a message regarding the incident, as a special case, FailedToPerformRestCallException class also maintains the http error code returned by the external service. This can be used to be more informative on REST responses.

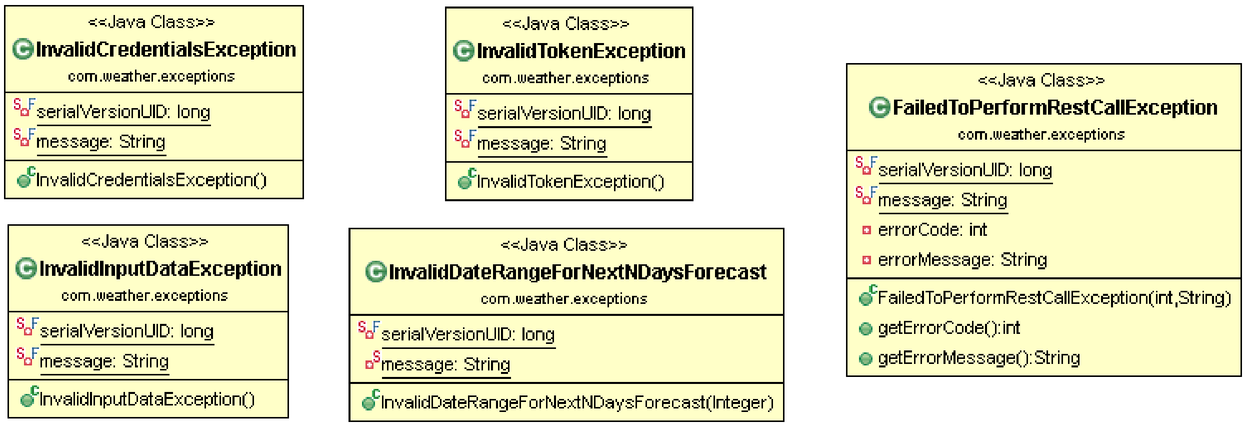


Figure 6-Custom Exception Classes

5- Tests

Unit test and integration tests are written in multiple classes. In this section we describe them.

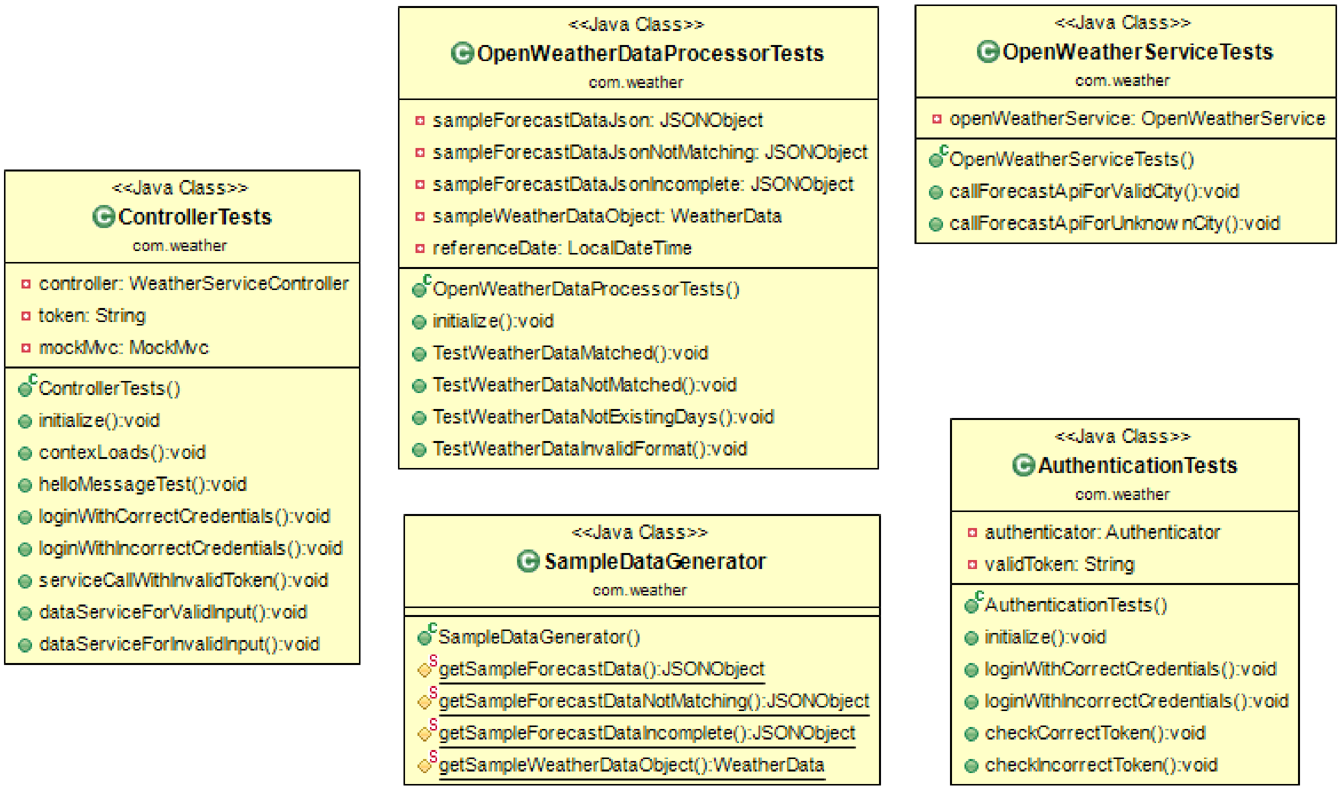

<<Java Class>>
OpenWeatherDataProcessorTests
com.weather
sampleForecastDataJson: JSONObject
sampleForecastDataJsonNotMatching: JSONObject
sampleForecastDataJsonIncomplete: JSONObject
sampleWeatherDataObject: WeatherData
referenceDate: LocalDateTime
OpenWeatherDataProcessorTests()
initialize():void
TestWeatherDataMatched():void
TestWeatherDataNotMatched():void
TestWeatherDataNotExistingDays():void
TestWeatherDataInvalidFormat():void
<<Java Class>>
OpenWeatherServiceTests
com.weather
openWeatherService: OpenWeatherService
OpenWeatherServiceTests()
callForecastApiForValidCity():void
callForecastApiForUnknownCity():void
<<Java Class>>
ControllerTests
com.weather
controller: WeatherServiceController
token: String
mockMvc: MockMvc
ControllerTests()
initialize():void
contexLoads():void
helloMessageTest():void
loginWithCorrectCredentials():void
loginWithIncorrectCredentials():void
serviceCallWithInvalidToken():void
dataServiceForValidInput():void
dataServiceForInvalidInput():void
<<Java Class>>
SampleDataGenerator
com.weather
SampleDataGenerator()
getSampleForecastData():JSONObject
getSampleForecastDataNotMatching():JSONObject
getSampleForecastDataIncomplete():JSONObject
getSampleWeatherDataObject():WeatherData
<<Java Class>>
AuthenticationTests
com.weather
authenticator: Authenticator
validToken: String
AuthenticationTests()
initialize():void
loginWithCorrectCredentials():void
loginWithIncorrectCredentials():void
checkCorrectToken():void
checkIncorrectToken():void

Figure 7- Test Classes

5-1- Unit Tests

Unit tests are as following.

1-1-5- OpenWeatherDataProcessorTests

This class tests for the correctness of processing performed on raw data read from external services.

1-1-6- OpenWeatherServiceTests

This class tests the driver layer of external weather API.

1-1-7- AuthenticationTests

Tests related to authentication are defined here.

1-1-8- SampleDataGenerator

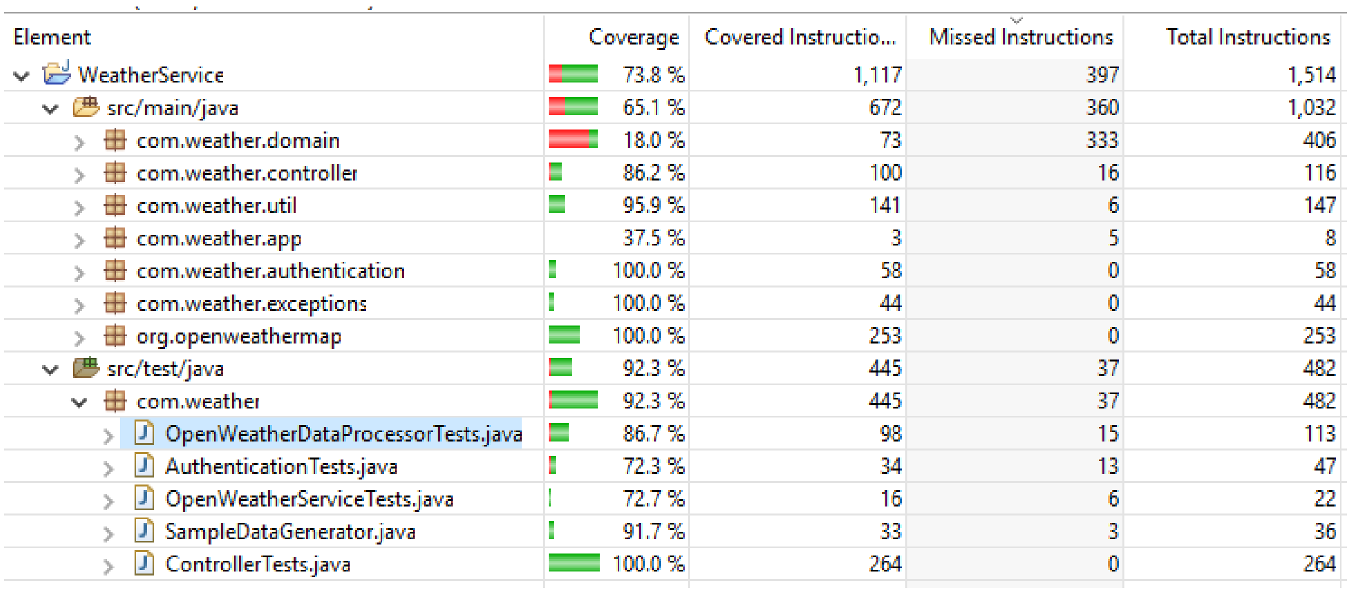
This class simple generates some sample data for test purposes, only to keep test modules clean.

5-2- Integration Tests

Integration test for this application is performed via MockMVC tool provided by SpringBoot. The tests can be seen in the “ControllerTests” class.

5-3- Coverage

Below figure demonstrates the code coverage for the tests. Note that domain classes indicate low percent because of auto generated getters and setters.



Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
WeatherService	73.8 %	1,117	397	1,514
src/main/java	65.1 %	672	360	1,032
com.weather.domain	18.0 %	73	333	406
com.weather.controller	86.2 %	100	16	116
com.weather.util	95.9 %	141	6	147
com.weather.app	37.5 %	3	5	8
com.weather.authentication	100.0 %	58	0	58
com.weather.exceptions	100.0 %	44	0	44
org.openweathermap	100.0 %	253	0	253
src/test/java	92.3 %	445	37	482
com.weather	92.3 %	445	37	482
OpenWeatherDataProcessorTests.java	86.7 %	98	15	113
AuthenticationTests.java	72.3 %	34	13	47
OpenWeatherServiceTests.java	72.7 %	16	6	22
SampleDataGenerator.java	91.7 %	33	3	36
ControllerTests.java	100.0 %	264	0	264

Figure 8 - Coverage