HW1: Mid-term assignment report

Pedro Daniel Fidalgo de Pinho [109986], v2024-04-06

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 work consists in a webpage dedicated to selling bus tickets for trips.

1.2 Current limitations

There is no error handling in the frontend and we can request the same trip multiple times as long as we don't pay for it. (If we end up paying for all of them then we might go over the occupancy limit).
User name and phone (on the frontend form) doesn't do anything - it is purely for aesthetics.

2 Product specification

2.1 Functional scope and supported interactions

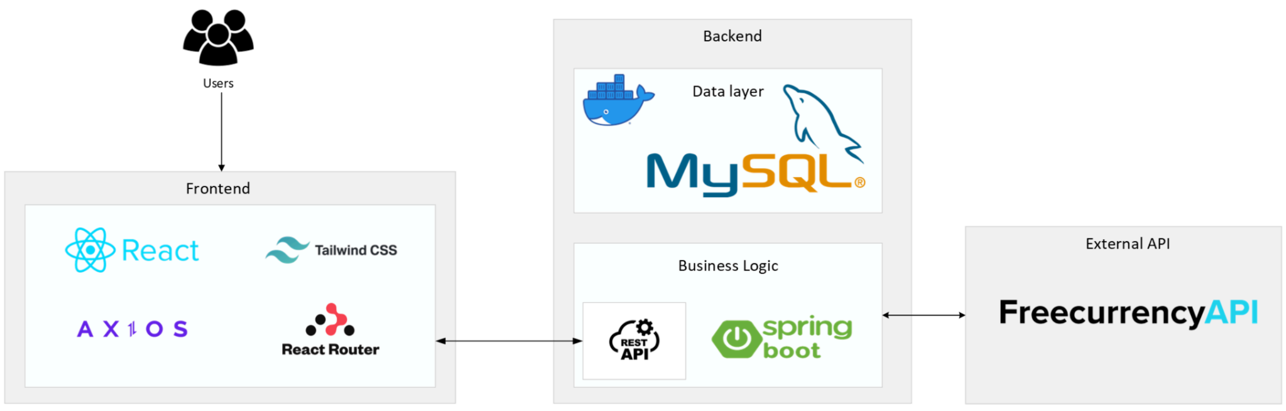
User accesses the web page, selects origin, destination and a date and looks for trips. Once it has found a trip to its liking then it selects it and is sent to another page where it is asked to provide information and select a currency. Afterwards it selects to reserve it and later can pay for it.
We can verify our reservations by clicking the button at the top left.

2.2 System architecture

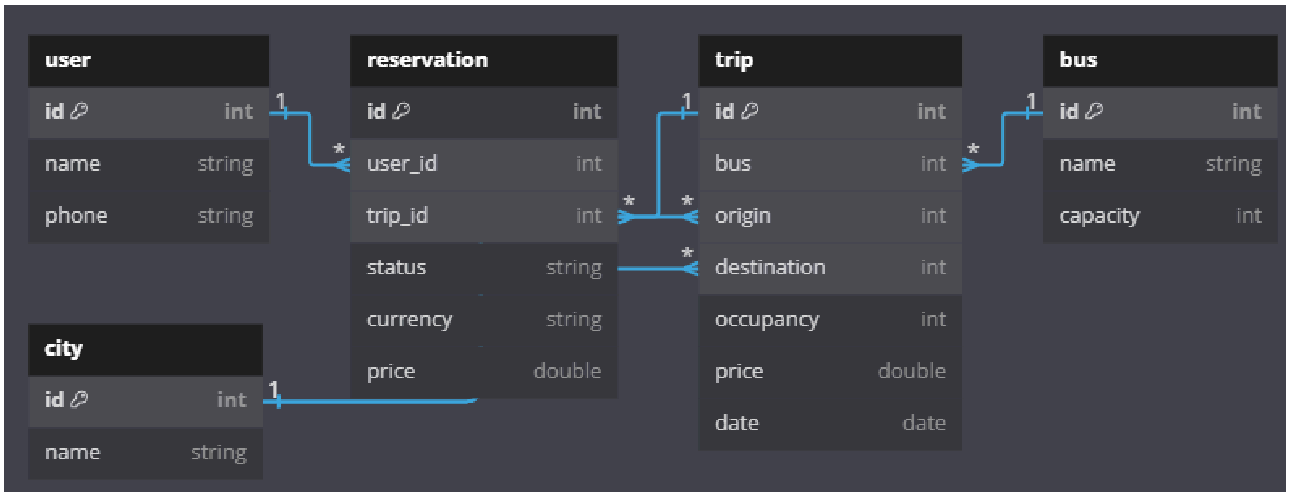
Frontend
React - JS library, however I used “Create React App” which is a framework to build the web app.
Tailwind CSS - basically CSS but made easy.
Axios - used to handle requests (promise-based HTTP Client)
React router - JavaScript framework that lets us handle client and server-side routing (it also includes basic features such as browser history and alike)

Backend
MySql - database (this is run in a docker-compose)
Springboot - java framework built on top of spring

Architecture

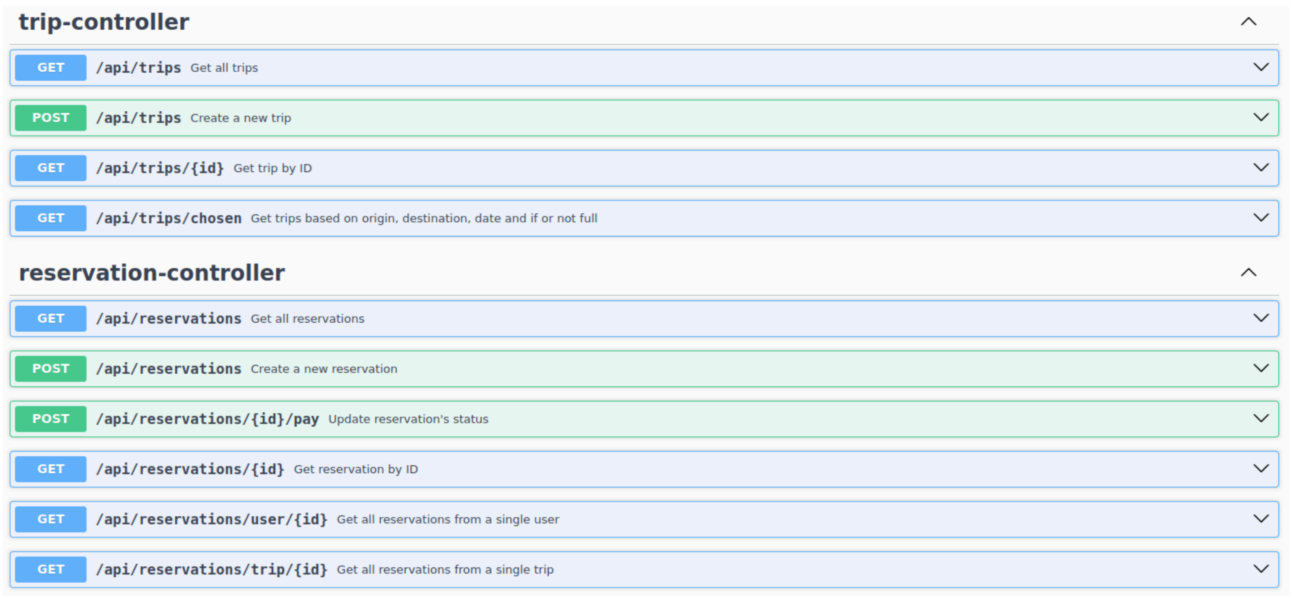


Database model



2.3 API for developers

(the following is an example as this does not contain all the endpoints)


trip-controller

GET /api/trips Get all trips

POST /api/trips Create a new trip

GET /api/trips/{id} Get trip by ID

GET /api/trips/chosen Get trips based on origin, destination, date and if or not full

reservation-controller

GET /api/reservations Get all reservations

POST /api/reservations Create a new reservation

POST /api/reservations/{id}/pay Update reservation's status

GET /api/reservations/{id} Get reservation by ID

GET /api/reservations/user/{id} Get all reservations from a single user

GET /api/reservations/trip/{id} Get all reservations from a single trip

3 Quality assurance

3.1 Overall strategy for testing

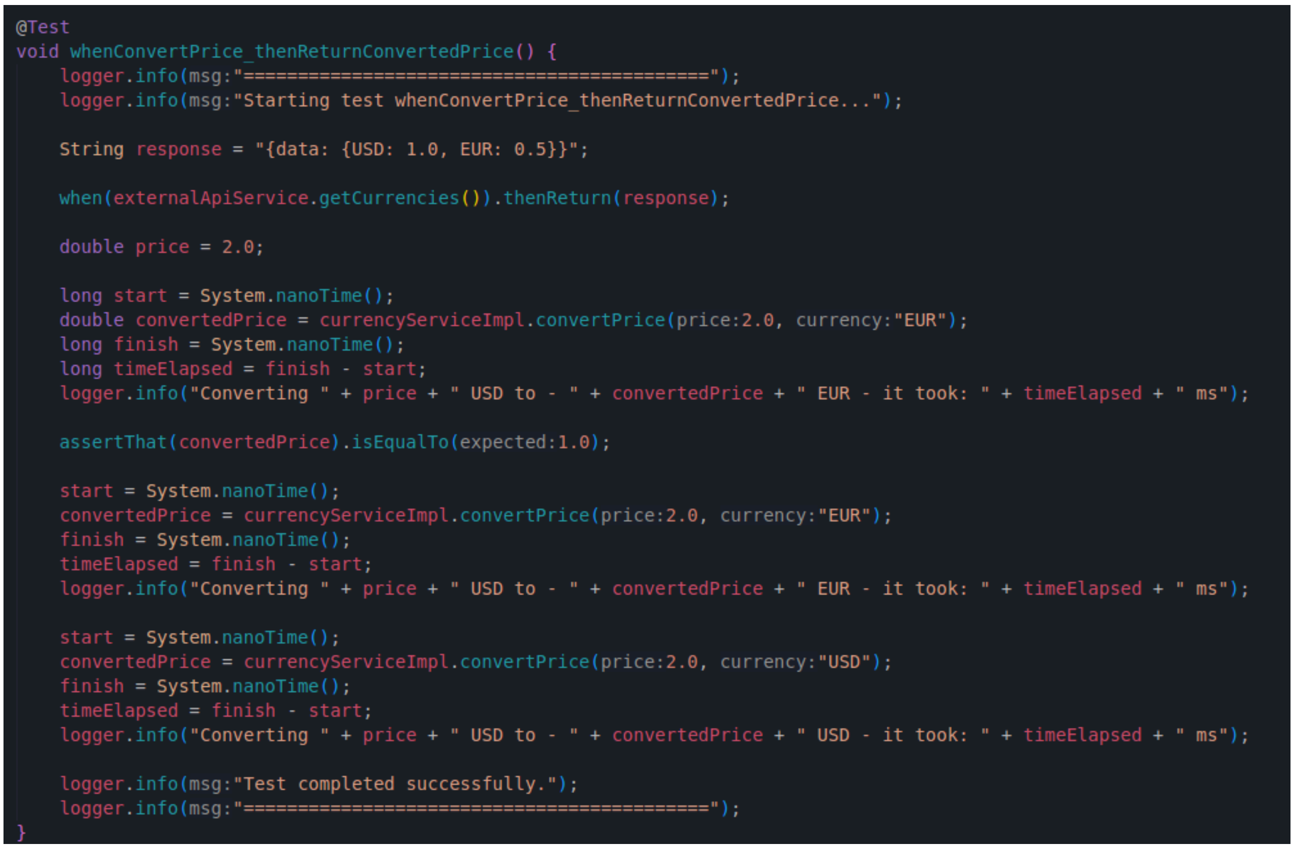
I tried to follow a TDD, however I struggled a lot to begin with, so as I got deeper into developing it I started putting tests to the side and eventually came back to them. This is because I wasn't exactly certain how to begin - which led me to waste a lot of time with pointless functions such as creating buses, creating cities, etc.

As I neared the end I started understanding tests a lot better, however I still feel like I need more practice to become good at it.

3.2 Unit and integration testing

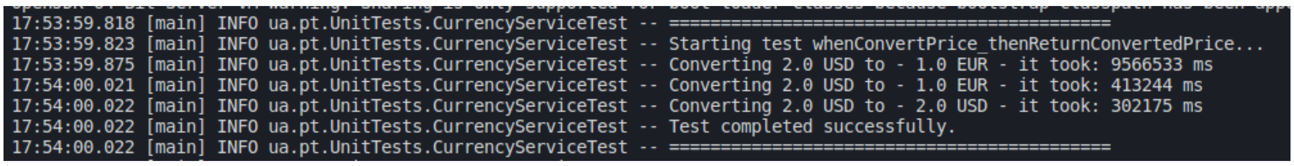
As mentioned previously I wasted quite a lot of time with pointless tests, however that does mean that I unit tested everything - including cache.
For ITs I decided to follow what we learned during classes and test the controller in an integration situation.

Cache test example



```java
@Test
void whenConvertPrice_thenReturnConvertedPrice() {
    logger.info(msg:"===========================================");
    logger.info(msg:"Starting test whenConvertPrice_thenReturnConvertedPrice...");

    String response = "{data: {USD: 1.0, EUR: 0.5}}";

    when(externalApiService.getCurrencies()).thenReturn(response);

    double price = 2.0;

    long start = System.nanoTime();
    double convertedPrice = currencyServiceImpl.convertPrice(price:2.0, currency:"EUR");
    long finish = System.nanoTime();
    long timeElapsed = finish - start;
    logger.info("Converting " + price + " USD to - " + convertedPrice + " EUR - it took: " + timeElapsed + " ms");

    assertThat(convertedPrice).isEqualTo(expected:1.0);

    start = System.nanoTime();
    convertedPrice = currencyServiceImpl.convertPrice(price:2.0, currency:"EUR");
    finish = System.nanoTime();
    timeElapsed = finish - start;
    logger.info("Converting " + price + " USD to - " + convertedPrice + " EUR - it took: " + timeElapsed + " ms");

    start = System.nanoTime();
    convertedPrice = currencyServiceImpl.convertPrice(price:2.0, currency:"USD");
    finish = System.nanoTime();
    timeElapsed = finish - start;
    logger.info("Converting " + price + " USD to - " + convertedPrice + " USD - it took: " + timeElapsed + " ms");

    logger.info(msg:"Test completed successfully.");
    logger.info(msg:"===========================================");
}
```

Output



```
17:53:59.818 [main] INFO ua.pt.UnitTests.CurrencyServiceTest -- ===========================================
17:53:59.823 [main] INFO ua.pt.UnitTests.CurrencyServiceTest -- Starting test whenConvertPrice_thenReturnConvertedPrice...
17:53:59.875 [main] INFO ua.pt.UnitTests.CurrencyServiceTest -- Converting 2.0 USD to - 1.0 EUR - it took: 9566533 ms
17:54:00.021 [main] INFO ua.pt.UnitTests.CurrencyServiceTest -- Converting 2.0 USD to - 1.0 EUR - it took: 413244 ms
17:54:00.022 [main] INFO ua.pt.UnitTests.CurrencyServiceTest -- Converting 2.0 USD to - 2.0 USD - it took: 302175 ms
17:54:00.022 [main] INFO ua.pt.UnitTests.CurrencyServiceTest -- Test completed successfully.
17:54:00.022 [main] INFO ua.pt.UnitTests.CurrencyServiceTest -- ===========================================
```

3.3 Functional testing

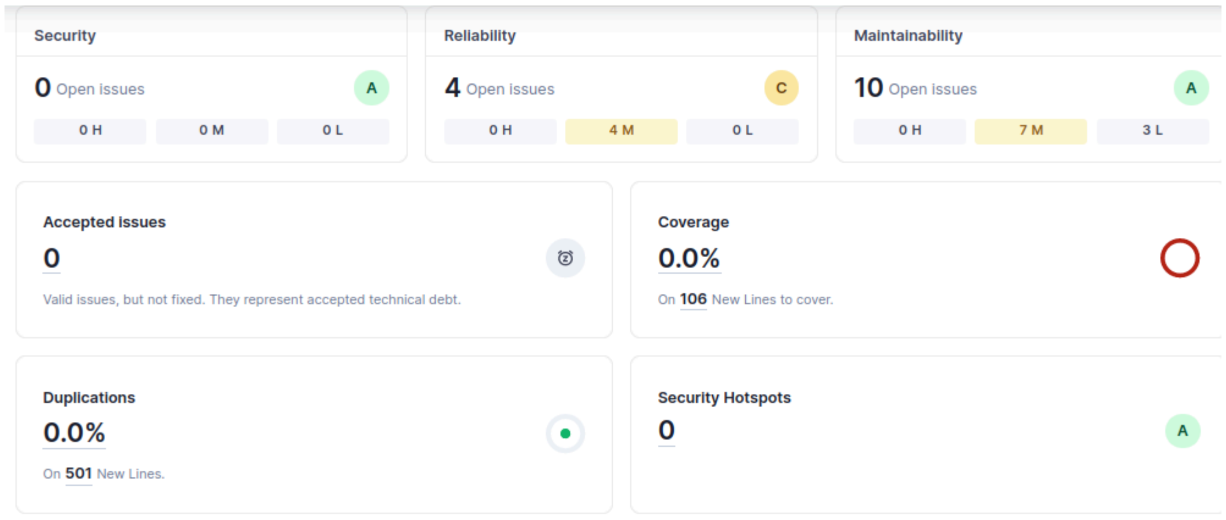
To test our application we used selenium where our user is assumed to be the one with ID=1.
In terms of code wise, I simply used the selenium IDE to record and exported it to java where I made very few changes.
The ideal scenario here would've been to use both BDD and page classes.

3.4 Code quality analysis

To guarantee good code quality I used Sonarqube locally, where I created two projects - one for the frontend and another for the backend.
This could've been done with Sonarcloud but the moment I learned that it was a bit too late.

Frontend

When inspecting the frontend code I expected it to be quite clean but to my surprise there were some code smells and reliability problems which were fixed later.
The coverage stat, as we can see, looks really bad. It has zero coverage across the software, however this is not alarming since it refers to tests related to javascript code and alike. This could be tested with Jest or Mocha, however this is not within this project's scope.


Security
0 Open issues A
0 H 0 M 0 L

Reliability
4 Open issues C
0 H 4 M 0 L

Maintainability
10 Open issues A
0 H 7 M 3 L

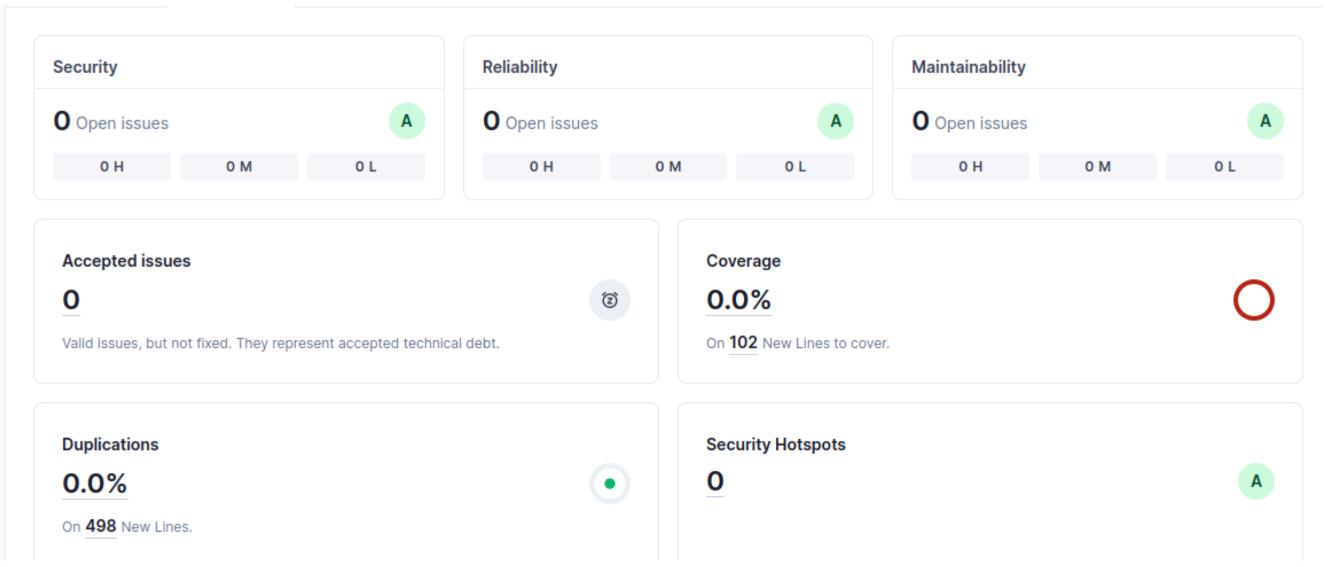
Accepted issues
0
Valid issues, but not fixed. They represent accepted technical debt.

Coverage
0.0%
On 106 New Lines to cover.

Duplications
0.0%
On 501 New Lines.

Security Hotspots
0 A

After correcting the code code smells and alike we ended up with the following


Security
0 Open issues A
0 H 0 M 0 L

Reliability
0 Open issues A
0 H 0 M 0 L

Maintainability
0 Open issues A
0 H 0 M 0 L

Accepted issues
0
Valid issues, but not fixed. They represent accepted technical debt.

Coverage
0.0%
On 102 New Lines to cover.

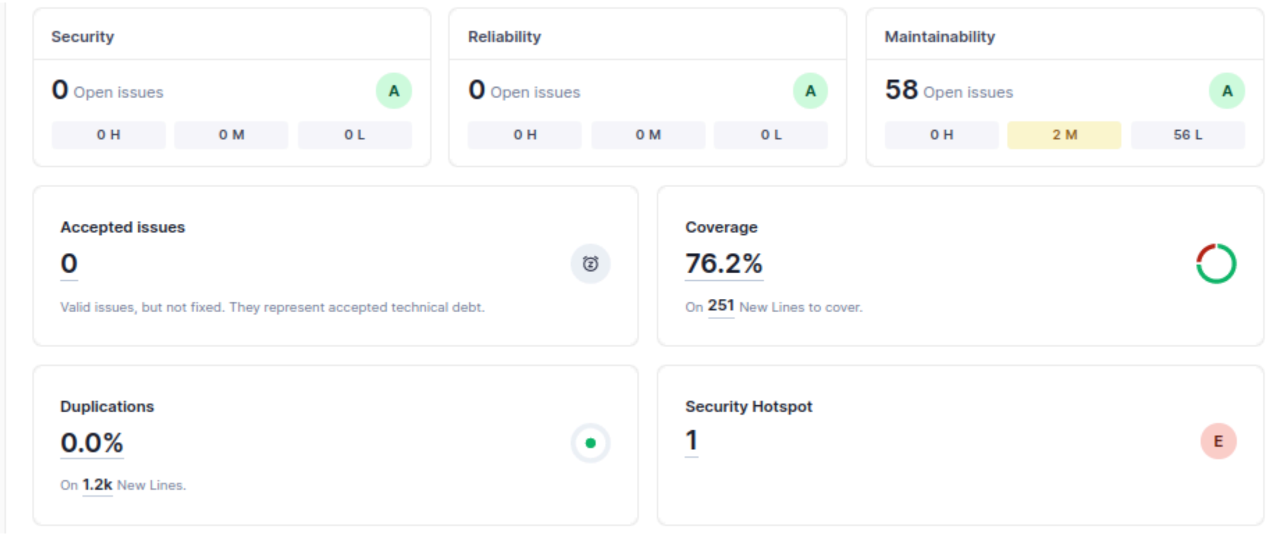
Duplications
0.0%
On 498 New Lines.

Security Hotspots
0 A

Backend

When it comes to code analysis I decided to verify the code every so often, (and sometimes add/fix one or both, code smells and coverage).
We can observe the before and after in the following images

(around middle of the project)


Security
0 Open issues A
0 H | 0 M | 0 L

Reliability
0 Open issues A
0 H | 0 M | 0 L

Maintainability
58 Open issues A
0 H | 2 M | 56 L

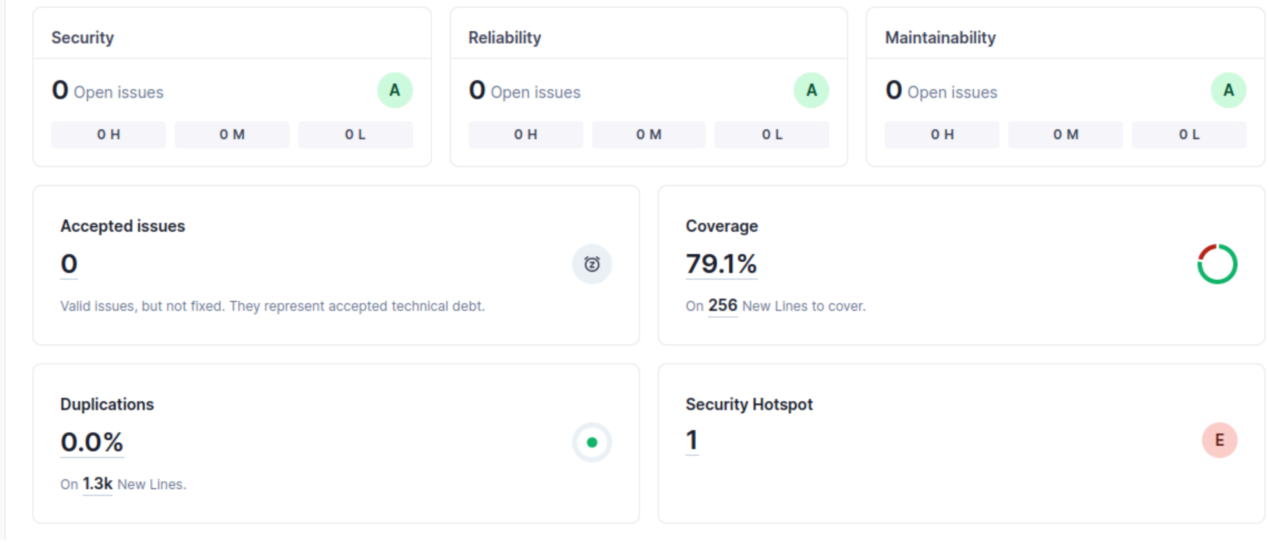
Accepted issues
0
Valid issues, but not fixed. They represent accepted technical debt.

Coverage
76.2%
On 251 New Lines to cover.

Duplications
0.0%
On 1.2k New Lines.

Security Hotspot
1 E

(end of the project)


Security
0 Open issues A
0 H | 0 M | 0 L

Reliability
0 Open issues A
0 H | 0 M | 0 L

Maintainability
0 Open issues A
0 H | 0 M | 0 L

Accepted issues
0
Valid issues, but not fixed. They represent accepted technical debt.

Coverage
79.1%
On 256 New Lines to cover.

Duplications
0.0%
On 1.3k New Lines.

Security Hotspot
1 E

The after image shows two concerning things. One the coverage isn't 100% and two there's a security hotspot.

Addressing the coverage first, this number would be around 96% if it wasn't including spring boot app runner and data initialization. This can be excluded if we explicitly tell it to exclude certain files and/or folders. However after reading online some opinions and reasoning about this metric I did decide to not bother changing it since this isn't a public (and complex) application. This metric helps us understand what is not covered within our code but our aim shouldn't be to make it be 100%, it should be to clean as much as possible. In this case if it were to go to 100% I wouldn't have gained anything but the right to proclaim that it was 100%.
In big projects however this could become annoying after some time and excluding said files

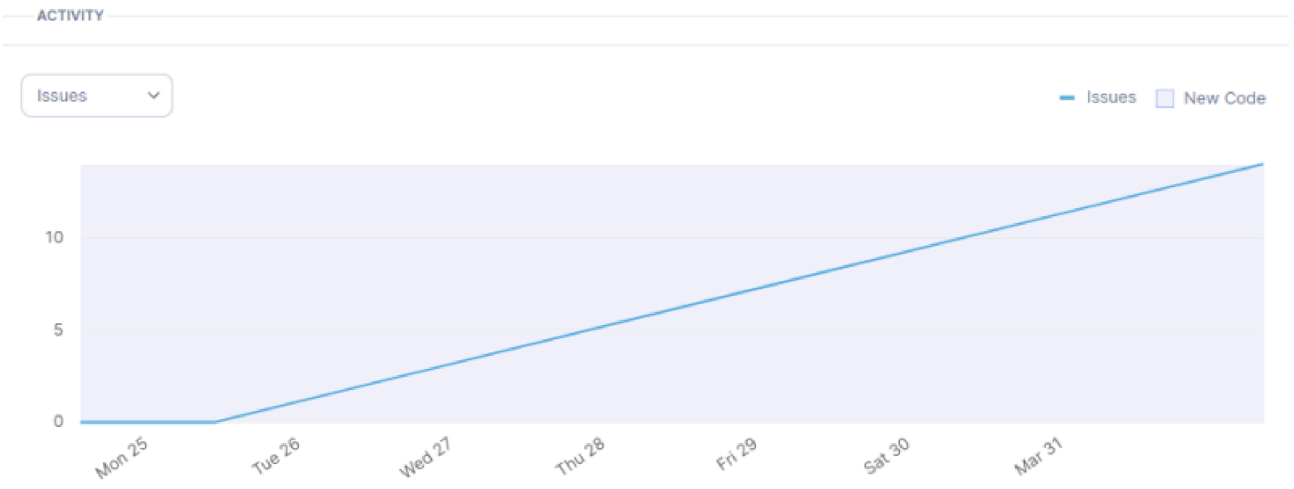
could be a good option.
In my case however, I would end up with around 96% coverage. This is because I do not have a test to cover a data formatter try catch and testing the external API (not mocking). So overall I feel that it is reasonable.

Now to address the hotspot security problem, it is simply the fact that my API key is visible to anyone who has access to my private repository. In this case it isn't a big concern, however in a public repo (or on a repository shared with multiple people (even if private)) we would want to keep the API key in a config (or env) file.

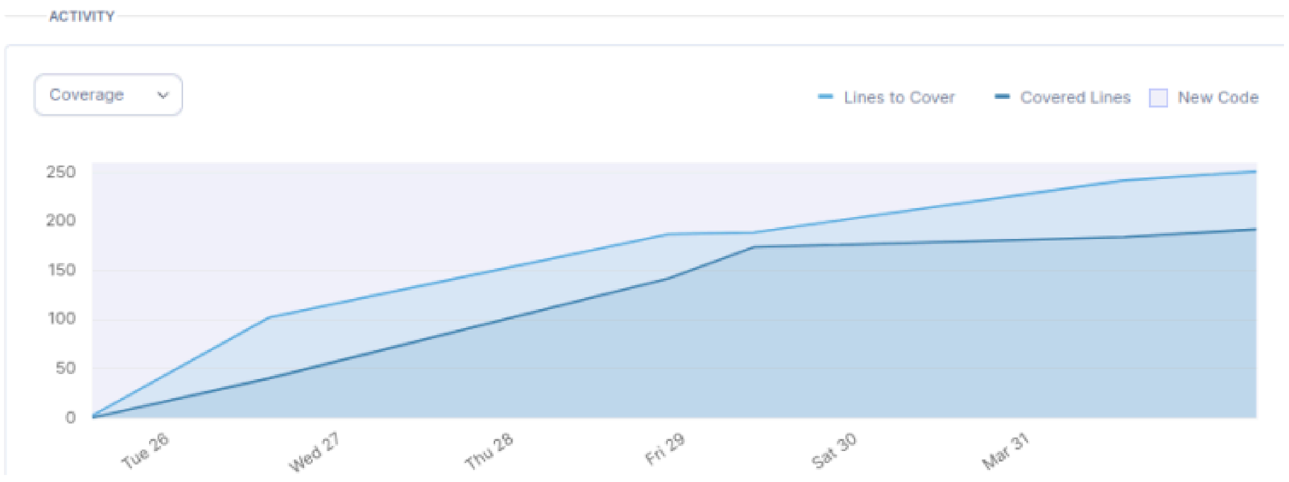
Comparison between frontend vs backend

Whilst I decided to verify the backend every so often, the frontend was approached differently. This led to this interesting graph comparison which demonstrates that verifying regularly helps understanding where the debt started building up.

(Frontend)

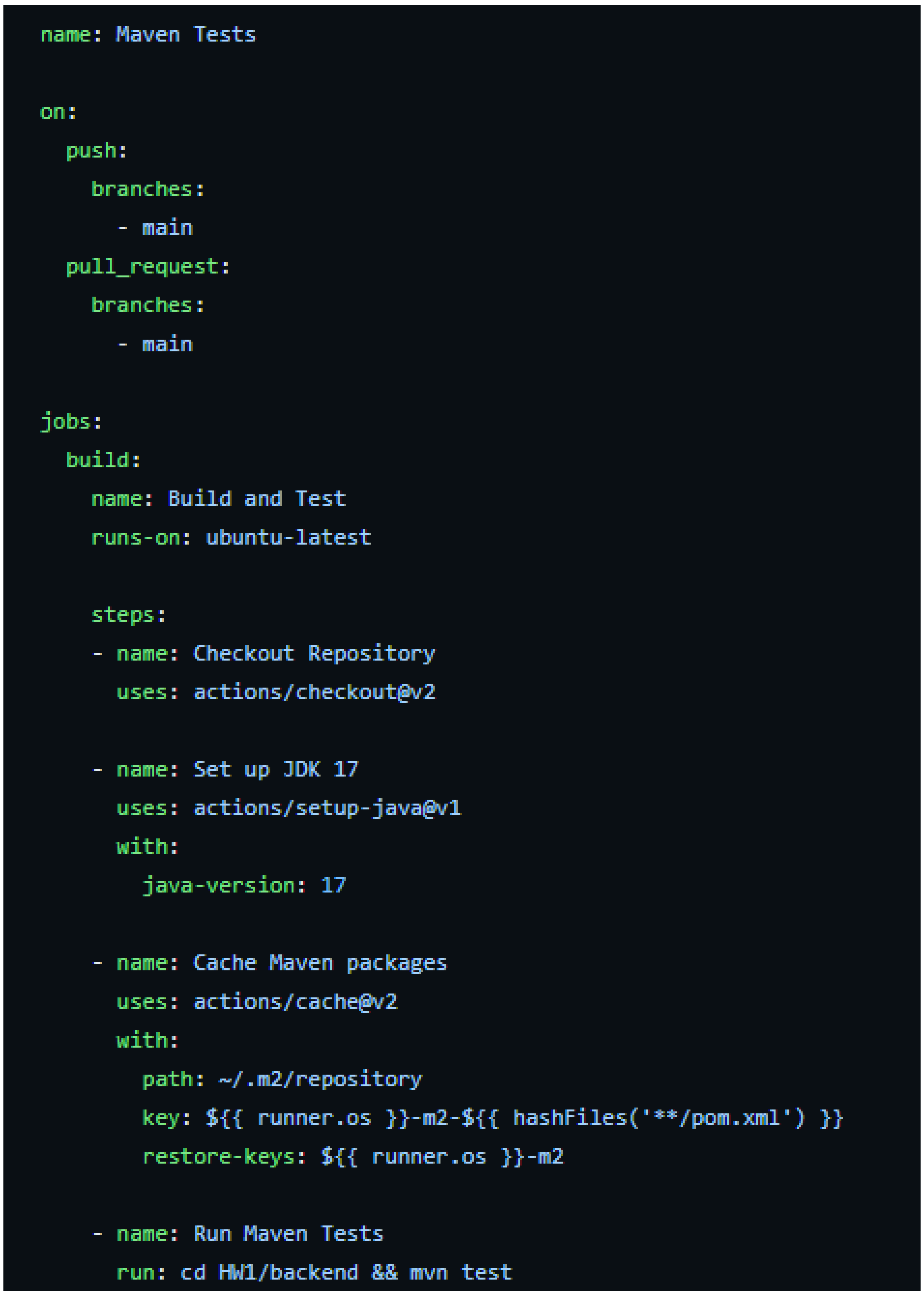


(Backend)



3.5 Continuous integration pipeline [optional]

This github is under a CI pipeline which is a workflow - this triggers github actions and it runs tests every time a commit is sent.



```yaml
name: Maven Tests

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    name: Build and Test
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v2

    - name: Set up JDK 17
      uses: actions/setup-java@v1
      with:
        java-version: 17

    - name: Cache Maven packages
      uses: actions/cache@v2
      with:
        path: ~/.m2/repository
        key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
        restore-keys: ${{ runner.os }}-m2

    - name: Run Maven Tests
      run: cd HW1/backend && mvn test
```

4 References & resources

Project resources

Resource:	**URL/location:**
Git repository	https://github.com/pdPinho/TQS_109986/tree/main/HW1
Video demo	https://github.com/pdPinho/TQS_109986/blob/main/HW1/demo-video.mp4
QA dashboard (online)	—
CI/CD pipeline	https://github.com/pdPinho/TQS_109986/blob/main/.github/workflows/build.yml
Deployment ready to use	—-

Reference materials

Frontend:

Tailwind CSS
https://tailwindcss.com/

React
https://react.dev/
https://create-react-app.dev/docs/getting-started

React Router
https://reactrouter.com/en/main

Sonarqube
https://medium.com/allient/static-analysis-using-sonarqube-in-a-react-webapp-dd4b335d6062

Axios
https://axios-http.com/

Backend:

Cache
https://www.baeldung.com/spring-cache-tutorial
https://www.baeldung.com/spring-boot-caffeine-cache

Sonarqube
https://www.baeldung.com/sonar-qube

Currency API
https://freecurrencyapi.com/

Swagger
https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api#dependency

(I feel that they changed how swagger works and this no longer works -

so instead use the following https://springdoc.org/
and if you encounter problems https://github.com/springdoc/springdoc-openapi/issues/2205)

Github:

Workflow

Create Your First Github Actions | Spring Boot Backend #11

Visual Studio Code

Sonarlint

https://marketplace.visualstudio.com/items?itemName=SonarSource.sonarlint-vscode

(sadly I couldn’t really make use of this since my computer lacks RAM, however it seems powerful)