HW1: Mid-term assignment report

João Paulo Mendes Gaspar [114514], v2025-04-0925-04-09

1 Introduction 1
1.1 Overview of the work 1
1.2 Current limitations 1

2 Product specification 2
2.1 Functional scope and supported interactions 2
2.2 System architecture 5
2.3 API for developers 5

3 Quality assurance 6
3.1 Overall strategy for testing 6
3.2 Unit and integration testing 7
3.3 Functional testing 8
3.4 Code quality analysis 9
3.5 Continuous integration pipeline [optional] 12

4 References & resources 12

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.
It was requested a development of an application that allowed the management of canteens of a university that allowed the users to reserve meals for a selected food facility and providing a code that referenced internally to that reservation, which in turn allowed a management of each reservations by giving an option for the users to see its details and cancel their reservations and an option to the workers to check in a meal of a user, marking that reservation as used.

1.2 Current limitations

The system, as it's implemented, has some known limitations. Currently, the logic is quite simple as there are no checks of when meals are reserved, cancelled or checked-in, all of this can be done anytime, if its status allows it (for example, used codes cannot be checked-in). Also there is no complex logic in terms of meals, menus are a set of meals, however when you reserve a menu, you cannot reserve a meal in specific, what you reserve is just a place in the food facility, without also tracking if a spot gets available inside the canteen, this

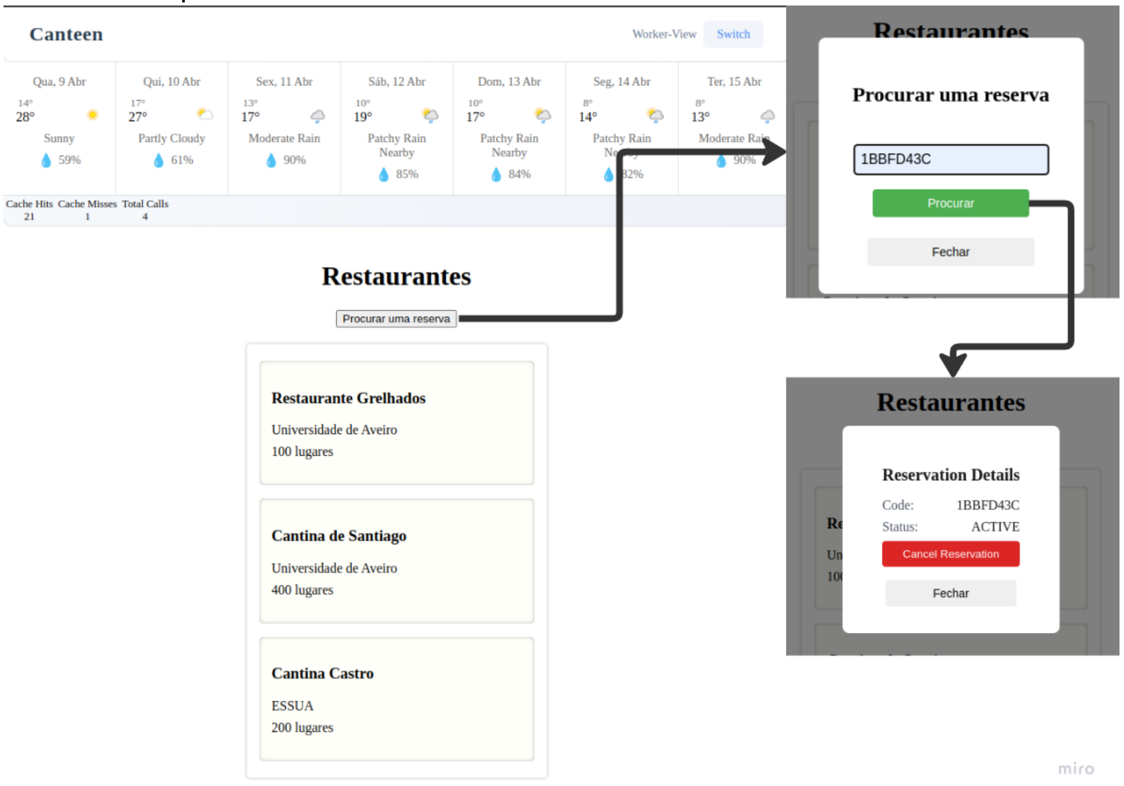
is, if the capacity of the facility is 200 people, only 200 people can reserve a meal, even if someone finished eating his meal effectively freeing space for someone else, however if anyone cancels a meal, a space gets open in that canteen for that day at that time (lunch/dinner).
Furthermore, although endpoints exist to create/delete meals/menus, there is no graphical interface for it in the current version, only to create new menus with defined meals.

2 Product specification

2.1 Functional scope and supported interactions

There are two main actors, Users and Workers.
Users and Workers can:
- See the available facilities;
- See the menus of the facilities for a given date and time (lunch or dinner)
- Reserve a menu (and get the internal ID of that reservation);
- Check information of their reservations;
- Cancel their reservations;
- See the weather information for the upcoming days.
Workers can also:
- Create new menus with meals for a specific restaurant, date and time
- See all reservations for a facility;
- Cancel/Check-in reservations.

In this first page, you can see all the restaurants, the weather (which is always present) and search for a specific reservation.



Canteen

Qua, 9 Abr	Qui, 10 Abr	Sex, 11 Abr	Sáb, 12 Abr	Dom, 13 Abr	Seg, 14 Abr	Ter, 15 Abr
14°	17°	13°	10°	10°	8°	8°
28°	27°	17°	19°	17°	14°	13°
Sunny	Partly Cloudy	Moderate Rain	Patchy Rain Nearby	Patchy Rain Nearby	Patchy Rain Ne	Moderate Rai
59%	61%	90%	85%	84%	32%	90%

Cache Hits	Cache Misses	Total Calls
21	1	4

Restaurantes

Restaurante Grelhados
Universidade de Aveiro
100 lugares

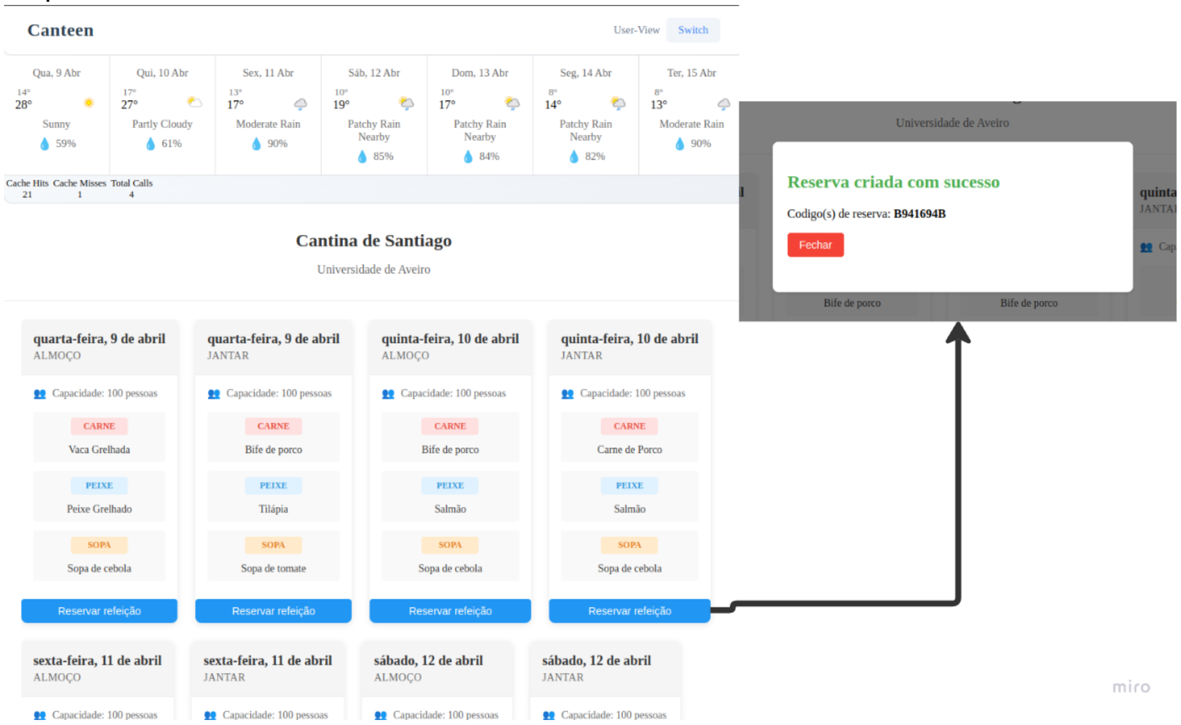
Cantina de Santiago
Universidade de Aveiro
400 lugares

Cantina Castro
ESSUA
200 lugares

Procurar uma reserva
1BBFD43C

Reservation Details
Code: 1BBFD43C
Status: ACTIVE

If a user clicks on a canteen, then it will be presented all meal options for each day and time, and if the user reserves a meal, then, if the operation is sucessful, the reservation code will be provided to him.


Canteen

User-View Switch

Qua, 9 Abr	Qui, 10 Abr	Sex, 11 Abr	Sáb, 12 Abr	Dom, 13 Abr	Seg, 14 Abr	Ter, 15 Abr
14° 28°	17° 27°	13° 17°	10° 19°	10° 17°	8° 14°	8° 13°
Sunny	Partly Cloudy	Moderate Rain	Patchy Rain Nearby	Patchy Rain Nearby	Patchy Rain Nearby	Moderate Rain
59%	61%	90%	85%	84%	82%	90%

Cache Hits	Cache Misses	Total Calls
21	1	4

Cantina de Santiago

Universidade de Aveiro

quarta-feira, 9 de abril ALMOÇO	quarta-feira, 9 de abril JANTAR	quinta-feira, 10 de abril ALMOÇO	quinta-feira, 10 de abril JANTAR
Capacidade: 100 pessoas	Capacidade: 100 pessoas	Capacidade: 100 pessoas	Capacidade: 100 pessoas
CARNE Vaca Grelhada	CARNE Bife de porco	CARNE Bife de porco	CARNE Carne de Porco
PEIXE Peixe Grelhado	PEIXE Tilápia	PEIXE Salmão	PEIXE Salmão
SOPA Sopa de cebola	SOPA Sopa de tomate	SOPA Sopa de cebola	SOPA Sopa de cebola
Reservar refeição	Reservar refeição	Reservar refeição	Reservar refeição

sexta-feira, 11 de abril ALMOÇO	sexta-feira, 11 de abril JANTAR	sábado, 12 de abril ALMOÇO	sábado, 12 de abril JANTAR
Capacidade: 100 pessoas	Capacidade: 100 pessoas	Capacidade: 100 pessoas	Capacidade: 100 pessoas

Universidade de Aveiro

Reserva criada com sucesso

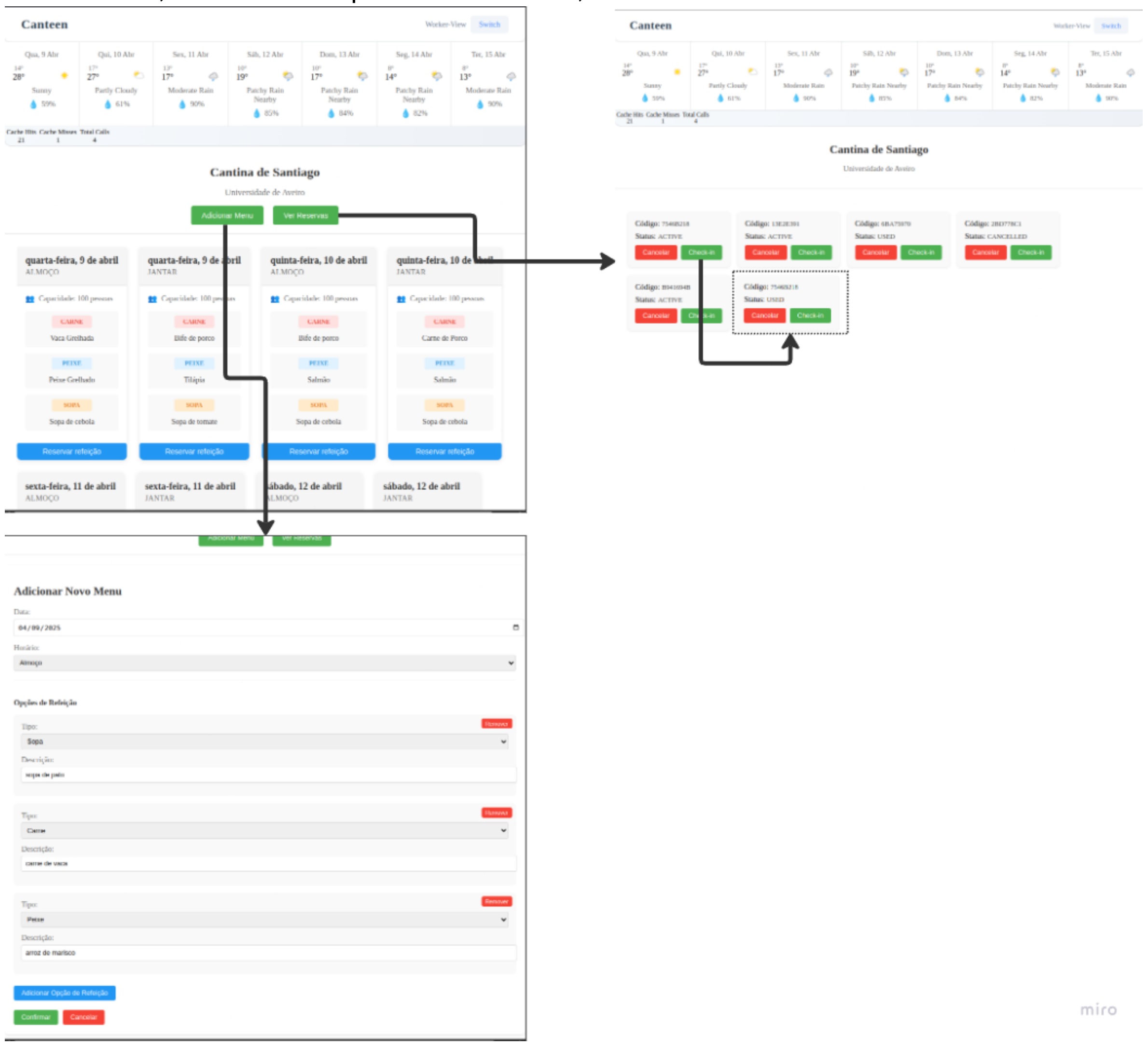
Codigo(s) de reserva: **B941694B**

Fechar

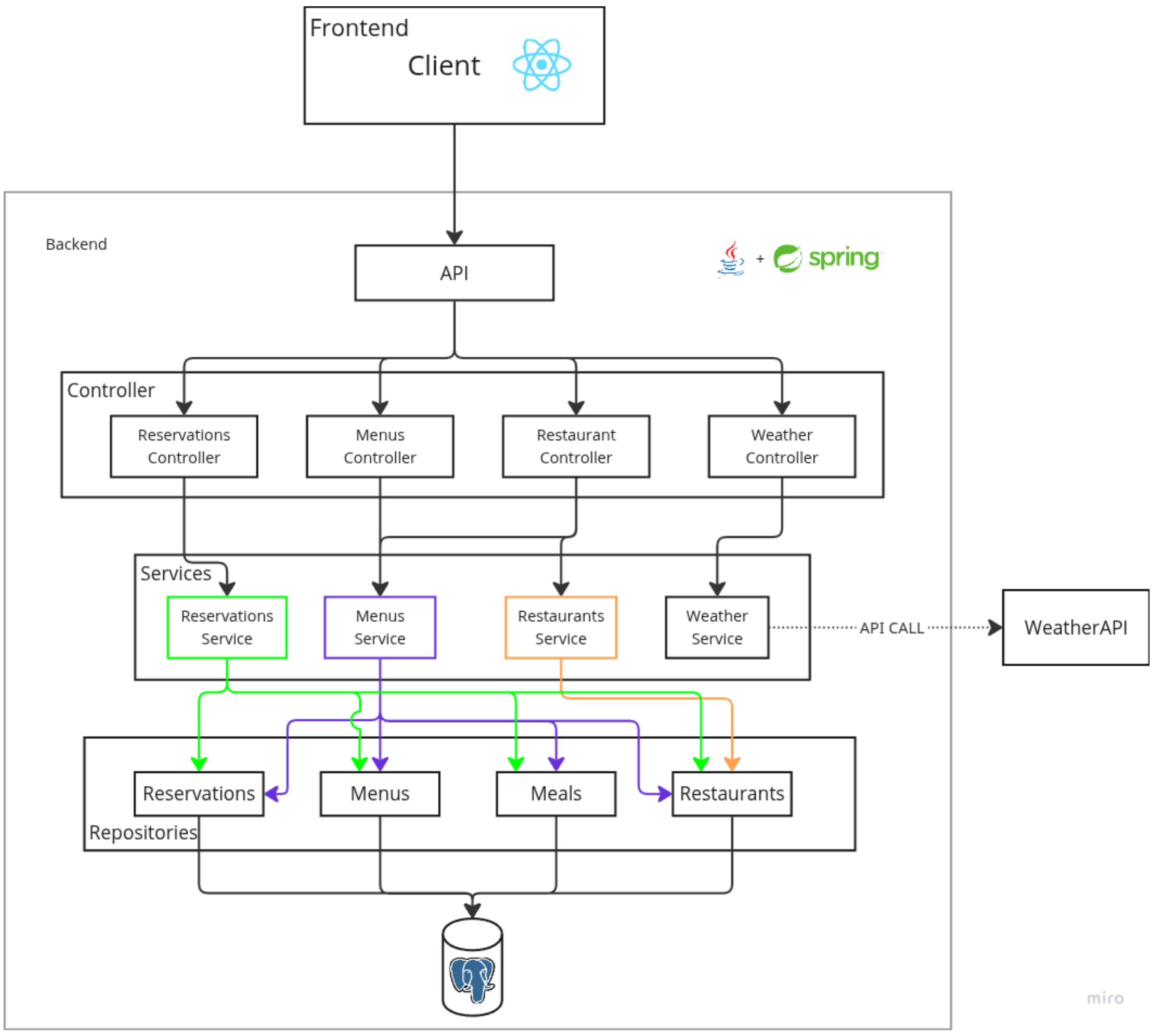
Bife de porco

Bife de porco

Furthermore, if the user is a worker (he can change by clicking on the switch button on the top-right) he will also be able to see two buttons, one to add a new menu and one to see all reservations, where he can proceed to cancel, or check-in made reservations



2.2 System implementation architecture



The architecture of the system has three main components:

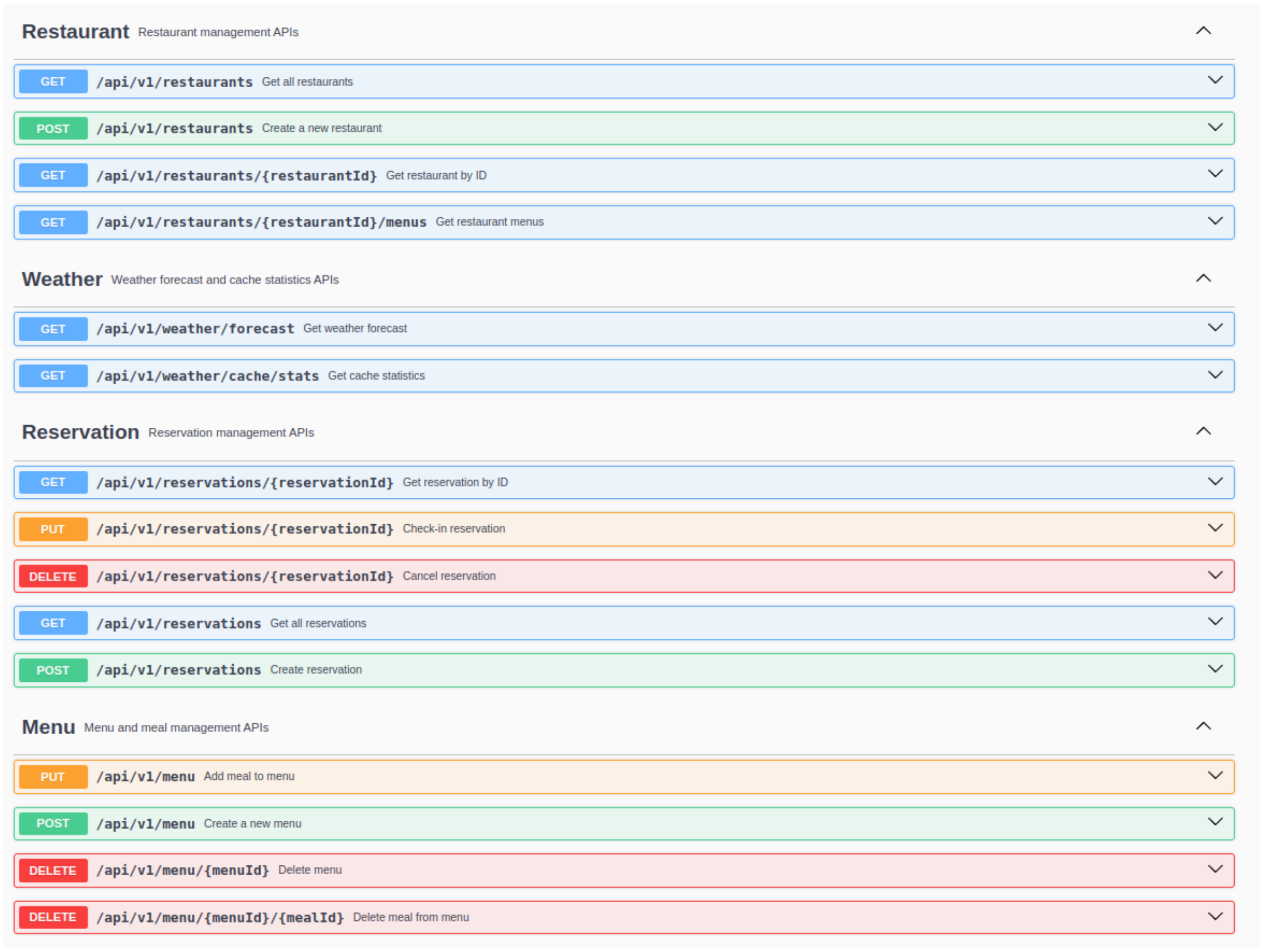
a) Frontend: Developed in React + Typescript

b) Backend: Developed in Java Springboot

c) Database: PostgreSQL database

2.3 API for developers

A developer has several endpoints, some with parameters, that he can use to further develop current frontend or even create a new application such as one for workers only where they can manage restaurants, menus and reservations.


Restaurant Restaurant management APIs

GET /api/v1/restaurants Get all restaurants
POST /api/v1/restaurants Create a new restaurant
GET /api/v1/restaurants/{restaurantId} Get restaurant by ID
GET /api/v1/restaurants/{restaurantId}/menus Get restaurant menus

Weather Weather forecast and cache statistics APIs

GET /api/v1/weather/forecast Get weather forecast
GET /api/v1/weather/cache/stats Get cache statistics

Reservation Reservation management APIs

GET /api/v1/reservations/{reservationId} Get reservation by ID
PUT /api/v1/reservations/{reservationId} Check-in reservation
DELETE /api/v1/reservations/{reservationId} Cancel reservation
GET /api/v1/reservations Get all reservations
POST /api/v1/reservations Create reservation

Menu Menu and meal management APIs

PUT /api/v1/menu Add meal to menu
POST /api/v1/menu Create a new menu
DELETE /api/v1/menu/{menuId} Delete menu
DELETE /api/v1/menu/{menuId}/{mealId} Delete meal from menu

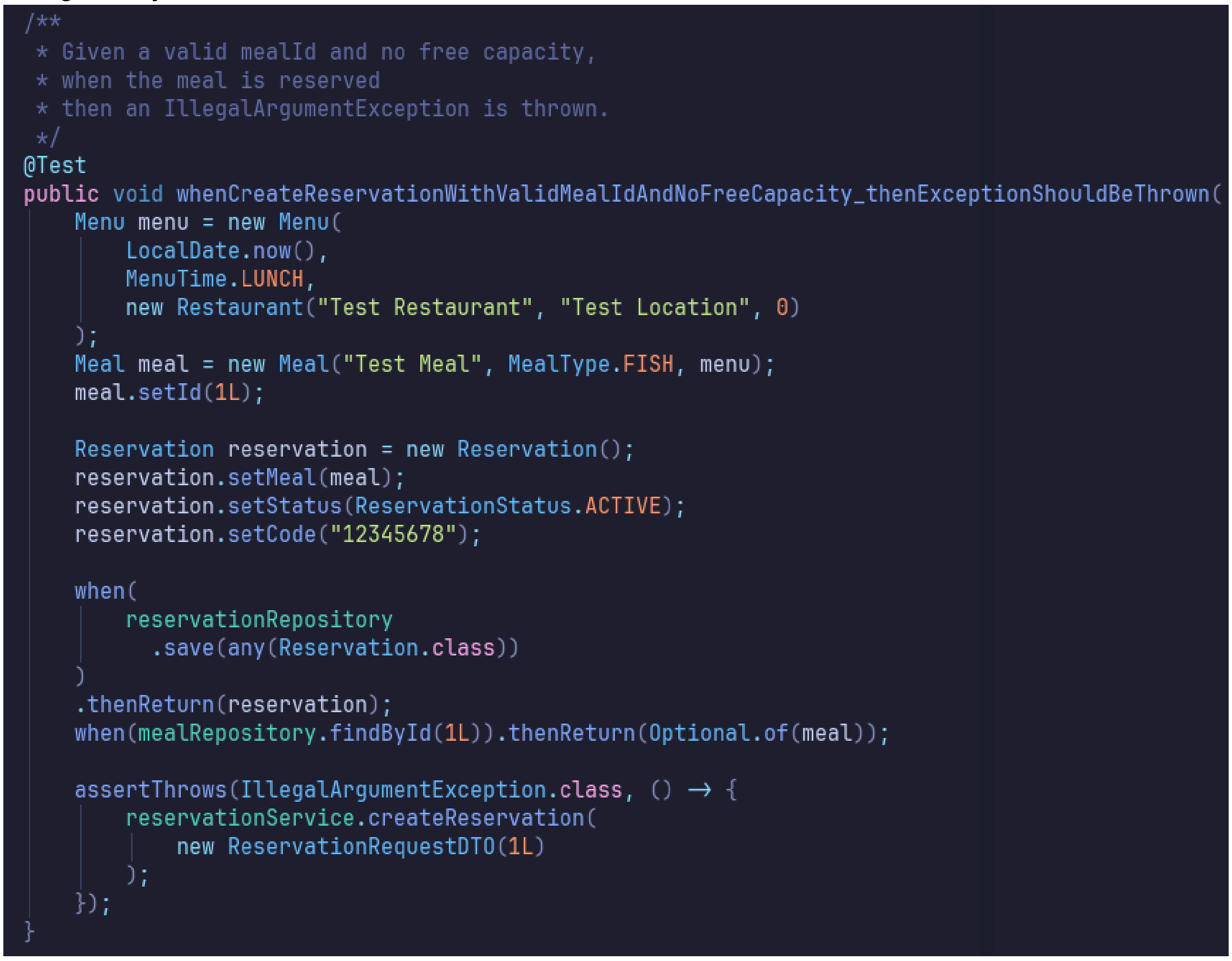
There is also an endpoint to gather weather forecasts and statistical data about the implemented cache.
These endpoints can be accessed by going to the link http://localhost:8080/swagger-ui/index.html

3 Quality assurance

3.1 Overall strategy for testing

For this project, focusing on backend, I decided to follow a mix of test development strategies.
Firstly I wrote tests based on BDD foundation by looking at each method and writing given/when/then tests for usual cases where normal values were passed to each method, and followed it by also inserting some important and common edge cases such as null values, and out of bounds use cases such as the usage of invalid arguments, for example, the input of id's not associated to any entity such as restaurants/meals/menus, and for common errors such as over reserving a full restaurant, etc.
As such, each one of the tests has a reason to exist, which are shown in a comment before each test method in a Given/When/Then way, as I focus on maximizing the scope of the tests to ensure bugs and errors are minimal while also trying to minimize duplicate tests, nonetheless some parts of some tests will happen twice, as some actions are prerequisites to

test other parts of a method, however, these duplication is necessary in order for all parts to be rigorously tested.



```java
/**
 * Given a valid mealId and no free capacity,
 * when the meal is reserved
 * then an IllegalArgumentException is thrown.
 */
@Test
public void whenCreateReservationWithValidMealIdAndNoFreeCapacity_thenExceptionShouldBeThrown(
    Menu menu = new Menu(
        LocalDate.now(),
        MenuTime.LUNCH,
        new Restaurant("Test Restaurant", "Test Location", 0)
    );
    Meal meal = new Meal("Test Meal", MealType.FISH, menu);
    meal.setId(1L);

    Reservation reservation = new Reservation();
    reservation.setMeal(meal);
    reservation.setStatus(ReservationStatus.ACTIVE);
    reservation.setCode("12345678");

    when(
        reservationRepository
          .save(any(Reservation.class))
    )
    .thenReturn(reservation);
    when(mealRepository.findById(1L)).thenReturn(Optional.of(meal));

    assertThrows(IllegalArgumentException.class, () → {
        reservationService.createReservation(
            new ReservationRequestDTO(1L)
        );
    });
}
```

Then, as I already had tests written, I started writing the code for the class I was implementing, effectively following a Test-Driven Development, and if necessary fixing errors that were appearing in the tests themselves.
So, I followed a TDD approach through behaviour driven tests. However, I did not use BDD specific tools as I felt it gave me more flexibility in writing tests.
Following the unit tests, in order to be confident in both my existing tests, my implementations and the behaviour I defined for my mocks, I proceeded to implement integration tests, these integration tests tested the model as a whole, entering it only through the controllers themselves by sending API requests.

3.2 Unit and integration testing

3.2.1 Unit Tests

In order to test repositories, I've only made tests for custom queries, this is, self-made queries with *@Query* annotation.
For all the controllers and services, I went deeper and unit-tested all their methods, isolating them through the usage of mocks with *Mockito* for their dependencies. However, for controller tests, as expected, there was also a need to use a mock REST service that allowed

for API calls to each controller, as such *Springboot's WebMvcTest* was used to send these requests.
For the Weather service unit tests, there was also a need to mock the API calls to *weatherapi*, as such, I as I was using RestTemplate to send requests to this API, I had to, by also using Mockito, mock the result of the api call.

3.2.2 Integration Tests

Finally, after implementing all the methods and testing them through the unit tests, I needed to test the system as a whole, which means that no mocks could be used used. To do so I had to have a REST service that allowed me to send requests to my controller, which I diverged for the controller tests and decided to use RestAssured as it offered a clean and easy to use REST service that also allowed to make assertions.
A database was also necessary for this phase, as I couldn't mock the repository functions, to fully test the system, as such I resorted to Test Containers, that allowed to have a PostgreSQL test database that was initialized with flyway, that allows database definition versioning control.
The integration tests were based on the previously defined controller unit tests.
There was room for improvement by creating additional integration tests by testing the lower layers (service-to-database), instead of relying only on controller-level tests as it will have fewer weak links than testing from the services down.

3.3 Functional testing

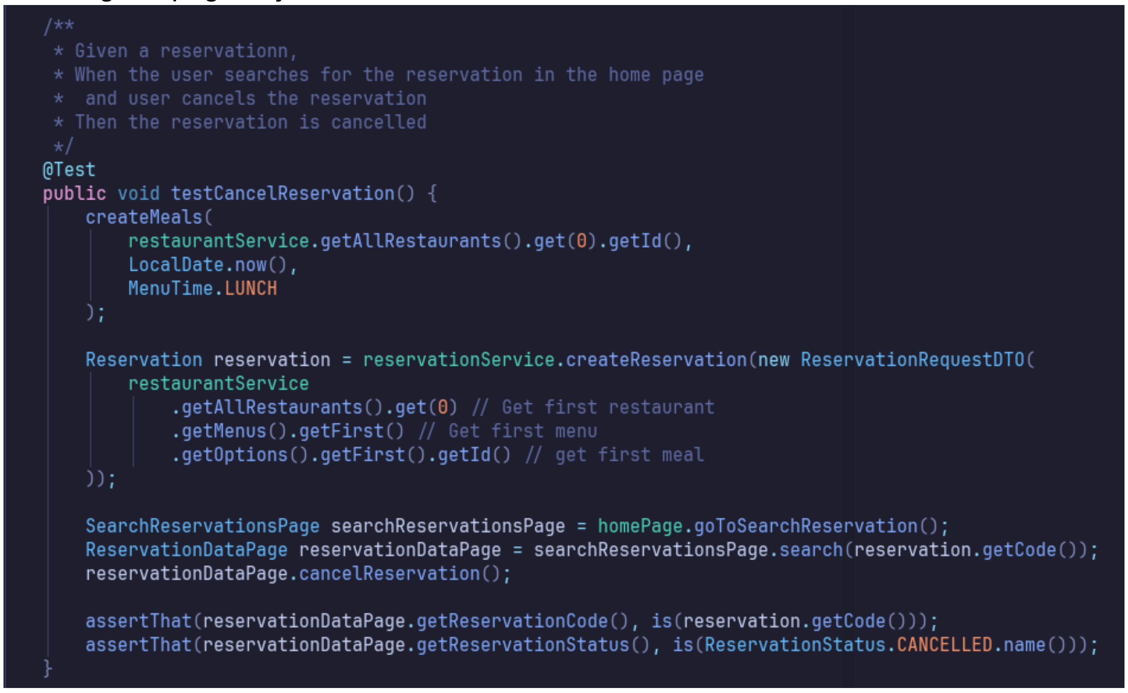
To test my frontend, I created page objects of my application and then looked at the use cases of each user profile (User and Worker) and created tests for each of them.
I've tested the following use cases:

d) Given there are 3 restaurants in the system, when the user go to the webpage, then he has to see those 3 restaurants
e) When the user goes to the home page, then the user must be able to see the weather information
f) Given a restaurant with 4 meals, when the user goes to the restaurant's meals page, then he must see those 4 meals;
g) Given a restaurant with 3 meals, when the user reserves a meal, then a success message with the reservation details is shown.
h) Given a valid reservation, when the user searches for the reservation in the home page, the the reservation details must show;
i) (*) Given a reservation, when the user searches for a reservation and there he cancels the said reservation, then the reservation status must be updated to cancelled.
j) Given the user is a worker and there exists a restaurant, when the user goes to the meals page then he has the options to add a new meal and one option to see all reservations; and when the user adds a new valid menu, then, after refreshing the page, the user can see the menu added to the list of upcoming meals
k) Given a restaurant with meals, when the user reserves a menu, and a worker goes to the all reservations page, then the worker can see the user's reservation in the list;

l) Given the user is a worker and there exists an active reservation, when the user goes to all reservations page and clicks on check-in reservation in an active reservation, then the status of the reservation becomes used;

m) Given the user is a worker and there exists an active reservation, when the user goes to all reservations page and clicks on cancel reservation in an active reservation, then the status of the reservation becomes cancelled;

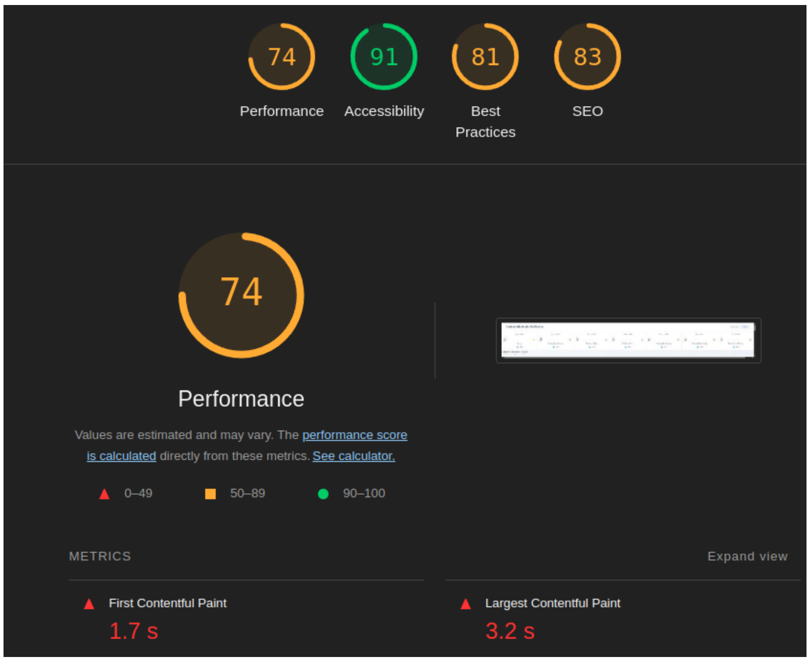
In the following figure, it«s presented one of the frontend test (*), where both the test and the usage of page objects can be seen



```java
/**
 * Given a reservationn,
 * When the user searches for the reservation in the home page
 *  and user cancels the reservation
 * Then the reservation is cancelled
 */
@Test
public void testCancelReservation() {
    createMeals(
        restaurantService.getAllRestaurants().get(0).getId(),
        LocalDate.now(),
        MenuTime.LUNCH
    );

    Reservation reservation = reservationService.createReservation(new ReservationRequestDTO(
        restaurantService
            .getAllRestaurants().get(0) // Get first restaurant
            .getMenus().getFirst() // Get first menu
            .getOptions().getFirst().getId() // get first meal
    ));

    SearchReservationsPage searchReservationsPage = homePage.goToSearchReservation();
    ReservationDataPage reservationDataPage = searchReservationsPage.search(reservation.getCode());
    reservationDataPage.cancelReservation();

    assertThat(reservationDataPage.getReservationCode(), is(reservation.getCode()));
    assertThat(reservationDataPage.getReservationStatus(), is(ReservationStatus.CANCELLED.name()));
}
```

3.4 Non functional testing

In order to test the application non-functionally, I used two different methods.
Firstly, using lightouse, I audited my frontend to address performance, accessibility and user-experience
With this test, I could understand that there was noticeable performance problems rooted in my application and that the color scheme chosen didn’t present enough contrast between text and elements which could be an issue for some users.



74 Performance | 91 Accessibility | 81 Best Practices | 83 SEO

74

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

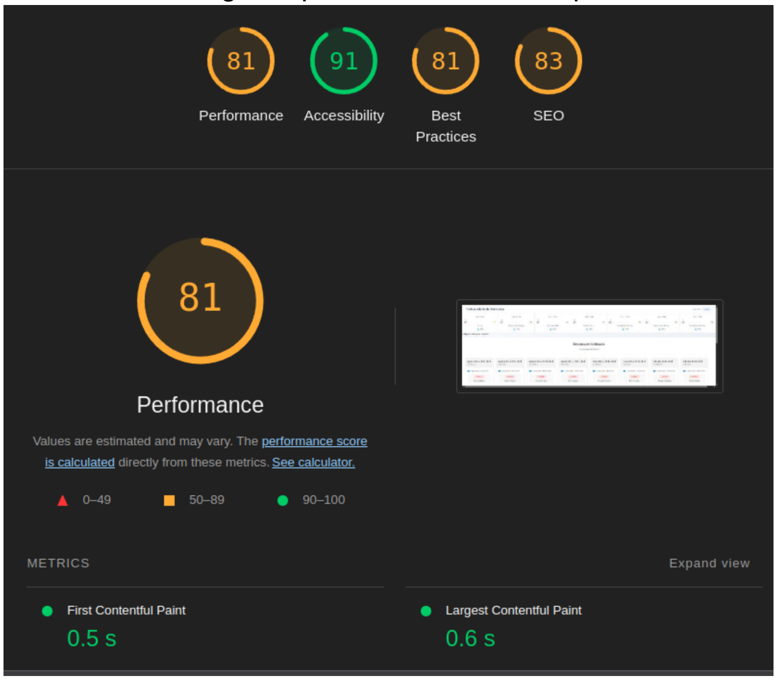
METRICS

Expand view

▲ First Contentful Paint
1.7 s

▲ Largest Contentful Paint
3.2 s

However, after further investigation I discovered that this happens I was testing on a development environment, after building the app and providing the production app, performance increased significantly, however it's still quite low due to cumulative layout shifts, that happen due to a loading component that I have implemented.



81 Performance | 91 Accessibility | 81 Best Practices | 83 SEO

81

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

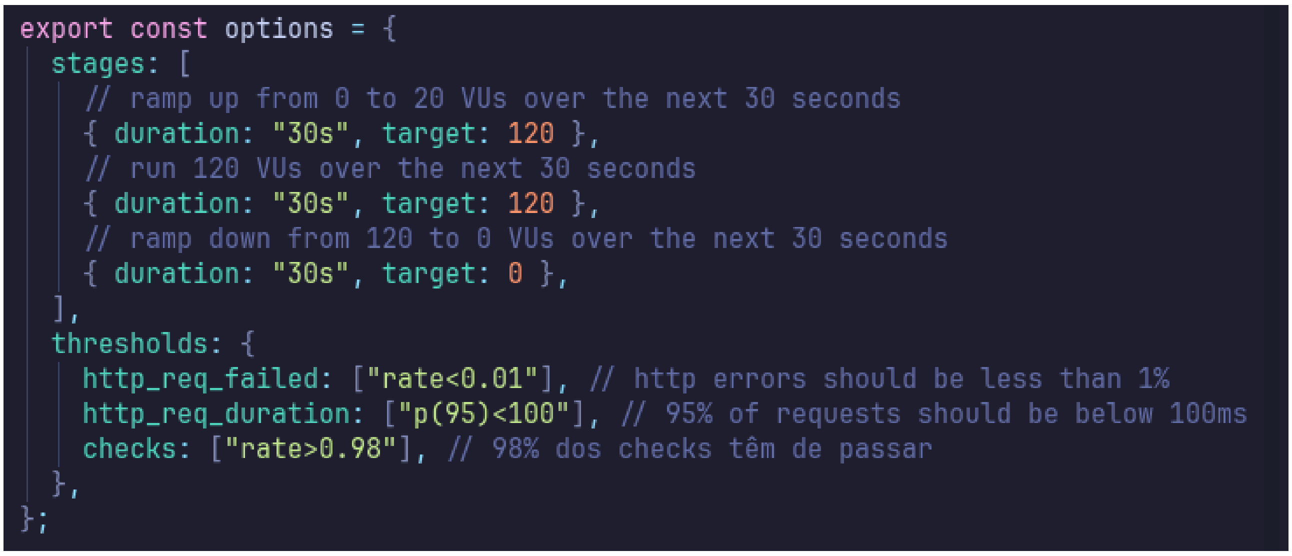
METRICS

Expand view

● First Contentful Paint
0.5 s

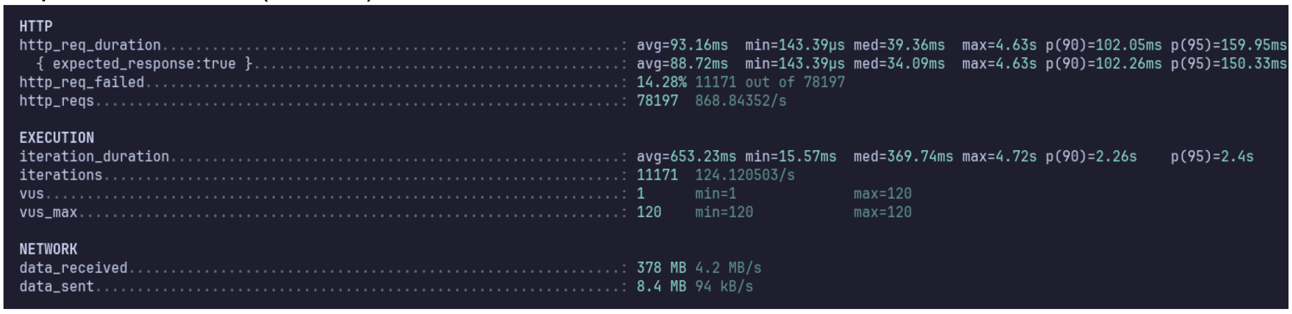
● Largest Contentful Paint
0.6 s

Furthermore, I also load tested my backend using K6, by making GET requests from 120 VU's for 1m30.



```
export const options = {
  stages: [
    // ramp up from 0 to 20 VUs over the next 30 seconds
    { duration: "30s", target: 120 },
    // run 120 VUs over the next 30 seconds
    { duration: "30s", target: 120 },
    // ramp down from 120 to 0 VUs over the next 30 seconds
    { duration: "30s", target: 0 },
  ],
  thresholds: {
    http_req_failed: ["rate<0.01"], // http errors should be less than 1%
    http_req_duration: ["p(95)<100"], // 95% of requests should be below 100ms
    checks: ["rate>0.98"], // 98% dos checks têm de passar
  },
};
```

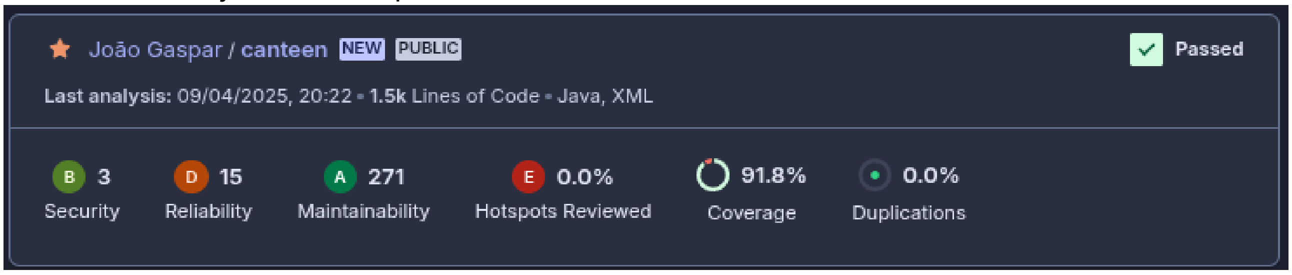
There was a drop in performance, as expected, nonetheless it responded decently as the average time to respond to requests was around 90ms. However, there was some http requests that failed (14.28%)



```
HTTP
http_req_duration.......................................................: avg=93.16ms  min=143.39µs med=39.36ms  max=4.63s p(90)=102.05ms p(95)=159.95ms
  { expected_response:true }............................................: avg=88.72ms  min=143.39µs med=34.09ms  max=4.63s p(90)=102.26ms p(95)=150.33ms
http_req_failed.........................................................: 14.28% 11171 out of 78197
http_reqs...............................................................: 78197  868.84352/s

EXECUTION
iteration_duration......................................................: avg=653.23ms min=15.57ms  med=369.74ms max=4.72s p(90)=2.26s    p(95)=2.4s
iterations..............................................................: 11171  124.120503/s
vus.....................................................................: 1      min=1              max=120
vus_max.................................................................: 120    min=120            max=120

NETWORK
data_received...........................................................: 378 MB 4.2 MB/s
data_sent...............................................................: 8.4 MB 94 kB/s
```

3.5 Code quality analysis

For static code analysis, I used Sonarcloud alongside with Github actions that automatically runs a code analysis in each update made to the backend code.



João Gaspar / canteen NEW PUBLIC Passed

Last analysis: 09/04/2025, 20:22 • 1.5k Lines of Code • Java, XML

B 3	D 15	A 271	E 0.0%	91.8%	0.0%
Security	Reliability	Maintainability	Hotspots Reviewed	Coverage	Duplications

From this analysis, some things have to be said. Firstly, security flags are to be ignored as they relate to CORS origins which weren't a necessity for this project to be secure. Moreover, of the 271 maintainability checks, 160 were about tests methods and attributes having a public modifier. The rest were problems that should be indeed fixed, but aren't critical, except 8 of them, where 5 were flagged as I had entities that had Many-to-one and One-to-many relationships, so it was difficult to fix these errors. There other 3 were literals that were repeatedly used in tests error messages, so it didn't make sense to fix them.
Of the test coverage, I achieved a 91.8% test coverage. Most of the code not analyzed were if statements branches that, although required extra effort to fix, would be better to do those. Nonetheless, I did not consider them to be critical to be tested, not at this point as the effort to do it wasn't justified. Also, there was code that wasn't covered in the Enums, however it

was getters method and code that converted string to enum based on its name, which also pushed the overall test coverage percentage down.
One code smells that the static code analysis addressed that I missed was a field that I gave the same name as the class I was using, a variable called cache of the type Cache, which correctly alerted me to rename to a more descriptive name, which I didn't do however I do understand the problem of not doing so. Another problem I was warned about was @Autowired components, which I wasn't aware that it was discouraged, I always favoured them to use wired constructors, but constructors allow the usage of final fields which all the @Autowired variables should be.

3.6 Continuous integration pipeline

For a continuous integration pipeline, I only added Sonarcloud, which runs all tests, run a static code analysis and outputs test coverage results visible in Sonarcloud project website. Besides that, no further CI pipeline was developed

References & resources

Project resources

Resource:	**URL/location:**
Git repository	https://github.com/Affapple/TQS_114514/tree/main/HW1-114514
Video demo	https://youtu.be/uktohuPVdGk
QA dashboard (online)	https://sonarcloud.io/project/overview?id=Affapple_TQS_114514