HW1: Mid-term assignment report

Ana Alexandra Antunes [876543], v2025-10-22

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 application in question, ZeroMonos, aims to help people book the collection of large waste items through a website and a possible future mobile app.

1.2 Current implementation (faults & extras)

The system currently allows users to make requests through a website and check out the progress of their requests (or cancel them) through a unique token. For the staff, they can see all the requests and change their state.
For the municipalities, the current implementation makes use of Thymeleaf templates to populate the list of possible municipalities instead of using a dedicated endpoint for it, which hurts the direct use of the backend for a mobile application.

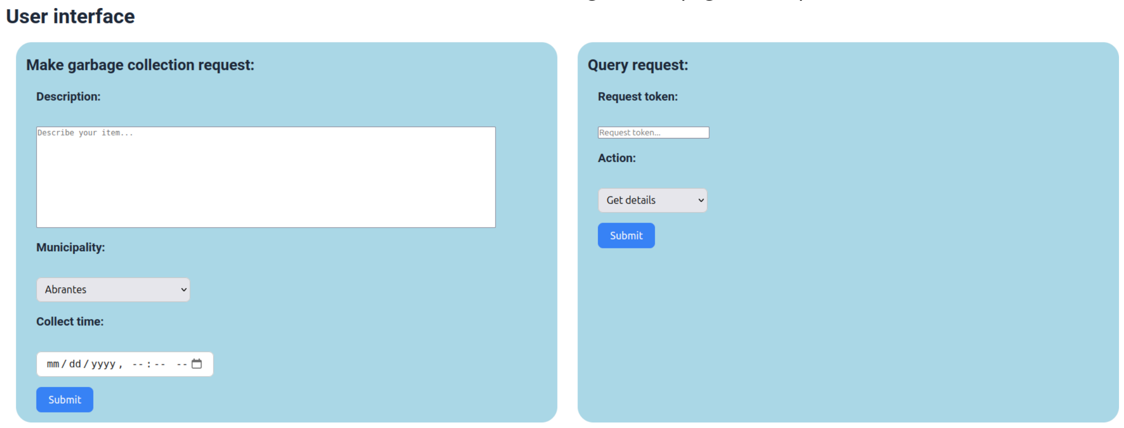
1.3 Use of generative AI

ChatGPT was used for generation of CSS for the web interface, as well as for most Javascript in the staff interface.
In terms of test code, ChatGPT generated code for accessing and changing private variables (specifically in MunicipalityClientTest), as well as a skeleton for repetitive test classes (RequestControllerTest, StateTransitionTest and RequestControllerIT).
The rest of the code and interface were without use of generative AI.

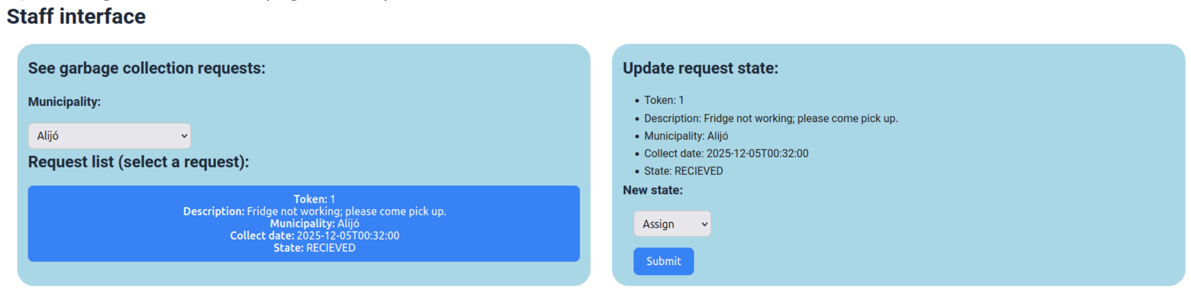
2 Product specification

2.1 Functional scope and supported interactions

The application will be used by 2 main types of people: the regular users and the staff.
The users will be the ones that make collection requests to the system (left box), specifying where and when the items can be collected. They can also use the identifier for a request to check out its status or decide to cancel it altogether (right box).



The staff are able to check the requests that the users make (left box) and act upon them, updating their state (right box).

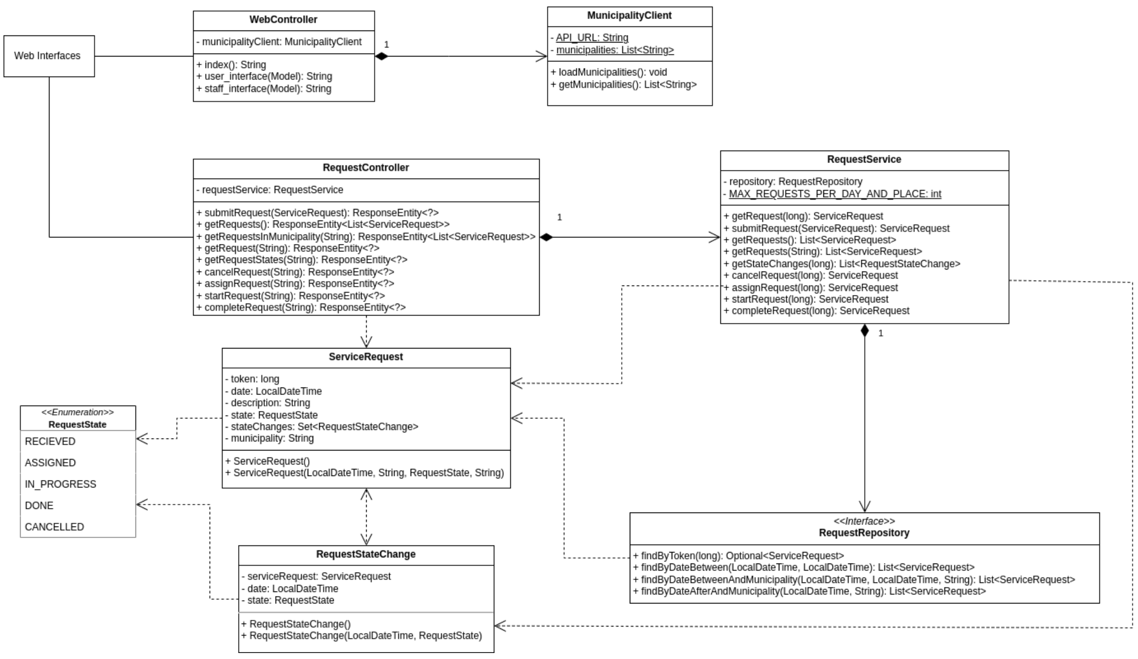


Token: 1
Description: Fridge not working; please come pick up.
Municipality: Alijó
Collect date: 2025-12-05T00:32:00
State: RECIEVED

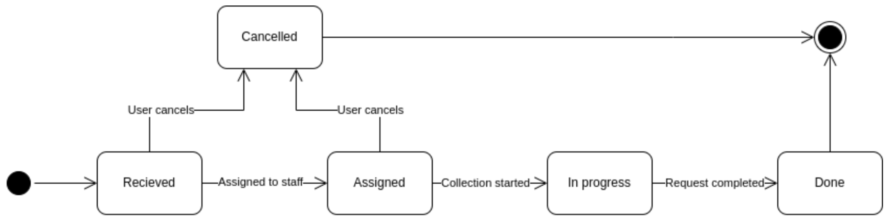
- Token: 1
- Description: Fridge not working; please come pick up.
- Municipality: Alijó
- Collect date: 2025-12-05T00:32:00
- State: RECIEVED

2.2 System implementation architecture

The system uses HTML (Thymeleaf templates), CSS and Javascript on the web frontend. On the backend, Java with Spring Boot is used. The first image shows the system architecture, which is a layered architecture with the presentation layer in the controllers and web interface, business layer with the services and MunicipalityClient, and persistence layer with the repositories.

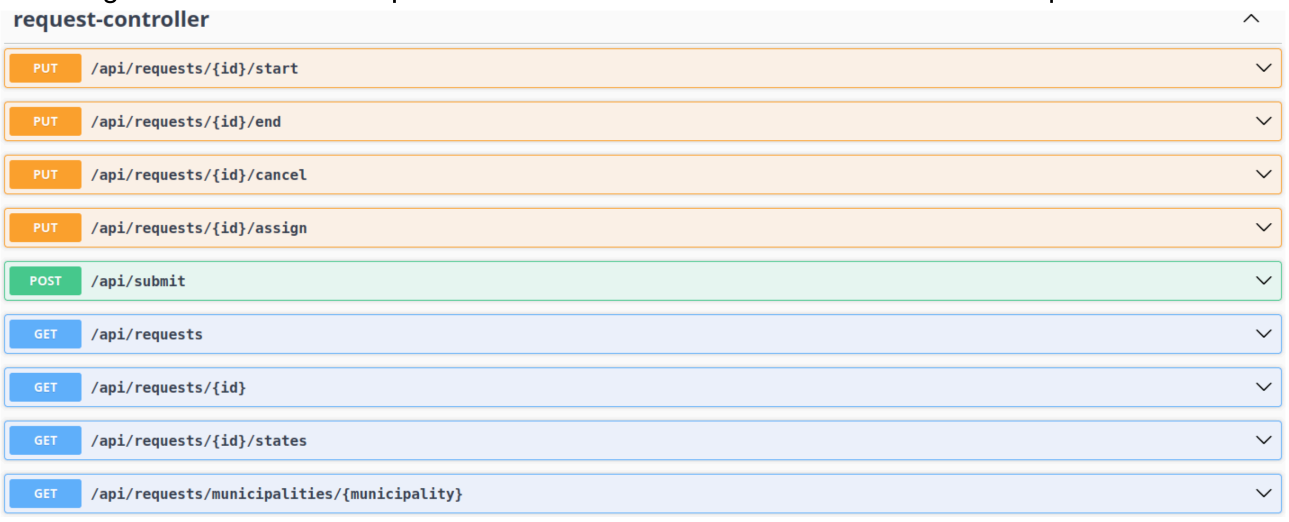

Web Interfaces
WebController
- municipalityClient: MunicipalityClient
+ index(): String
+ user_interface(Model): String
+ staff_interface(Model): String
1
MunicipalityClient
- API_URL: String
- municipalities: List<String>
+ loadMunicipalities(): void
+ getMunicipalities(): List<String>
RequestController
- requestService: RequestService
+ submitRequest(ServiceRequest): ResponseEntity<?>
+ getRequests(): ResponseEntity<List<ServiceRequest>>
+ getRequestsInMunicipality(String): ResponseEntity<List<ServiceRequest>>
+ getRequest(String): ResponseEntity<?>
+ getRequestStates(String): ResponseEntity<?>
+ cancelRequest(String): ResponseEntity<?>
+ assignRequest(String): ResponseEntity<?>
+ startRequest(String): ResponseEntity<?>
+ completeRequest(String): ResponseEntity<?>
1
RequestService
- repository: RequestRepository
- MAX_REQUESTS_PER_DAY_AND_PLACE: int
+ getRequest(long): ServiceRequest
+ submitRequest(ServiceRequest): ServiceRequest
+ getRequests(): List<ServiceRequest>
+ getRequests(String): List<ServiceRequest>
+ getStateChanges(long): List<RequestStateChange>
+ cancelRequest(long): ServiceRequest
+ assignRequest(long): ServiceRequest
+ startRequest(long): ServiceRequest
+ completeRequest(long): ServiceRequest
1
<<Enumeration>>
RequestState
RECIEVED
ASSIGNED
IN_PROGRESS
DONE
CANCELLED
ServiceRequest
- token: long
- date: LocalDateTime
- description: String
- state: RequestState
- stateChanges: Set<RequestStateChange>
- municipality: String
+ ServiceRequest()
+ ServiceRequest(LocalDateTime, String, RequestState, String)
<<Interface>>
RequestRepository
+ findByToken(long): Optional<ServiceRequest>
+ findByDateBetween(LocalDateTime, LocalDateTime): List<ServiceRequest>
+ findByDateBetweenAndMunicipality(LocalDateTime, LocalDateTime, String): List<ServiceRequest>
+ findByDateAfterAndMunicipality(LocalDateTime, String): List<ServiceRequest>
RequestStateChange
- serviceRequest: ServiceRequest
- date: LocalDateTime
- state: RequestState
+ RequestStateChange()
+ RequestStateChange(LocalDateTime, RequestState)

This is a diagram of the evolution of the state of a ServiceRequest.



2.3 API for developers

The image below shows the OpenAPI documentation for the backend API developed.



request-controller

PUT /api/requests/{id}/start

PUT /api/requests/{id}/end

PUT /api/requests/{id}/cancel

PUT /api/requests/{id}/assign

POST /api/submit

GET /api/requests

GET /api/requests/{id}

GET /api/requests/{id}/states

GET /api/requests/municipalities/{municipality}

The PUT endpoints have the same purpose: update the status of a request, if possible. When they can, they return the updated request. If not, they return either a 404 Not Found or a 400 Bad Request if the token is not valid.

POST /api/submit:

- Submits a request
- Body: the request to be submitted

- Response:
    - 201 Created if the request was submitted without problem
    - 400 Bad Request if the request couldn't be made because of the date or request overflow (reason is given in the response body).

GET /api/requests returns the list of all requests.
GET /api/requests/{id}:

- Returns a request given its token
- Response:
    - 200 Ok and the request in the body if a valid token is given.
    - 404 Not found if the token does not exist
    - 400 Bad Request if the token is invalid

GET /api/requests/{id}/states

- Returns the ordered history of states of a request given its token
- Response:
    - 200 Ok and the list in the body if a valid token is given.
    - 404 Not found if the token does not exist
    - 400 Bad Request if the token is invalid

GET /api/requests/municipality/{municipality} returns the list of all requests for a municipality.

3 Quality assurance

3.1 Overall strategy for testing

During development I didn't follow a full TDD approach, since I didn't code the tests before any real feature implementation, but my approach was doing a rough implementation of one or more features and then revise and update them using tests.

3.2 Unit and integration testing

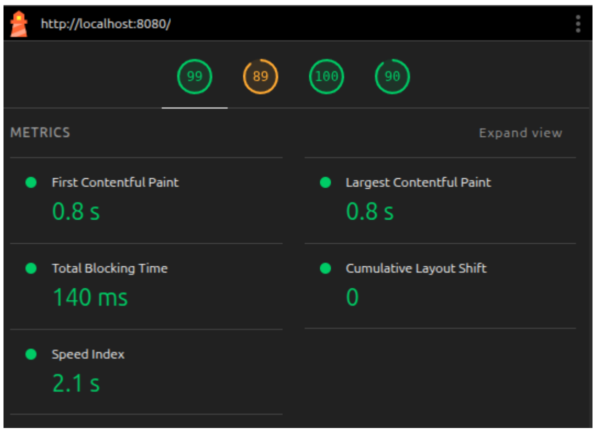
The unit tests consist of the majority of the total tests. I implemented tests for every service and controller, with tests for all possible paths in them. These tests ensure that the core logic of the services works as intended. The tests make use of Mocks and Spys of the elements not currently being tested, such as repositories.
Integration tests were used to test the functioning of the municipalities API and the overall functionality of the system. REST-Assured was used in this step to ensure that the system produces the desired responses when the API is being used.

3.3 Acceptance testing

The acceptance tests were developed using Selenium. They were first recorded in Selenium IDE and then exported into Java tests. Due to time constraints, the implementation of these tests is not desirable, since they don't use Cucumber or Page Object Model. The tests simulate the tasks that the 2 roles of the system use: the user creates requests and checks them out, opting or not to cancel them, and the staff sees all requests and changes their state.

3.4 Non-functional testing

I used Lighthouse in the Chromium browser to run performance tests on the three pages I developed, with the results below (from left to right, main page, user and staff interfaces). The settings used were the default values.



http://localhost:8080/

METRICS — Expand view

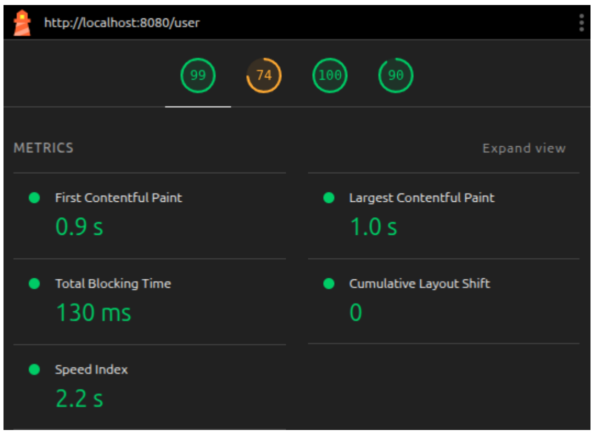
First Contentful Paint
0.8 s

Largest Contentful Paint
0.8 s

Total Blocking Time
140 ms

Cumulative Layout Shift
0

Speed Index
2.1 s



http://localhost:8080/user

METRICS — Expand view

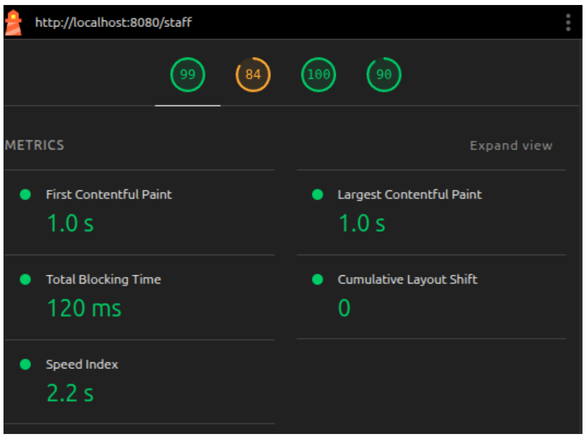
First Contentful Paint
0.9 s

Largest Contentful Paint
1.0 s

Total Blocking Time
130 ms

Cumulative Layout Shift
0

Speed Index
2.2 s



http://localhost:8080/staff

METRICS — Expand view

First Contentful Paint
1.0 s

Largest Contentful Paint
1.0 s

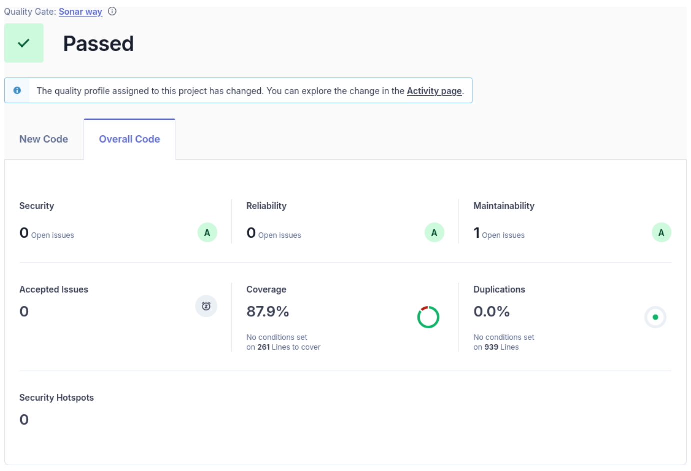
Total Blocking Time
120 ms

Cumulative Layout Shift
0

Speed Index
2.2 s

3.5 Code quality analysis

For static code analysis, I made use of the SonarQube cloud platform. The code used the default “sonar way” quality gate for verification. The image below shows the general results of the last analysis made on the overall code, which shows an 87.9% code coverage statistic. The remaining code not covered in testing amounts to small branches not being fully covered and getters/setters in data classes.
As for the issue in the “Maintainability” section, it’s a high severity bad smell, but due to the nature of the code in that specific portion of code, I couldn’t find a good way to make the intended change without sacrificing aspects such as testing.


Quality Gate: Sonar way

Passed

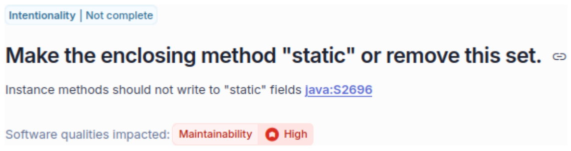
The quality profile assigned to this project has changed. You can explore the change in the Activity page.

New Code | Overall Code

Security	Reliability	Maintainability
0 Open issues (A)	0 Open issues (A)	1 Open issues (A)

Accepted Issues	Coverage	Duplications
0	87.9% No conditions set on 261 Lines to cover	0.0% No conditions set on 939 Lines

Security Hotspots
0


Intentionality | Not complete

Make the enclosing method "static" or remove this set.

Instance methods should not write to "static" fields java:S2696

Software qualities impacted: Maintainability High

3.6 Continuous integration pipeline [optional]

In GitHub I created a public repository for this project and added 2 workflows:

- build.yml: runs the sonar maven plugin which sends the code to Sonarcloud for code quality analysis.
- maven.yml: Runs all unit and integration tests.

Both of them run after every push and pull request, though there were no pull requests made.

4 References & resources

Project resources

Resource:	URL/location:
Video demo	In git repository: /docs/Video.mkv
QA dashboard (online)	https://sonarcloud.io/summary/overall?id=Apmds_ZeroMonos&branch=master
CI/CD pipeline	https://github.com/Apmds/ZeroMonos/actions

Reference materials

These resources helped with the making of this project, with their setup guides and tutorials:

- https://www.baeldung.com/
- https://www.digitalocean.com/community/tutorials/
- https://www.guru99.com/alert-popup-handling-selenium.html