

ZeroMonos

Bulk Waste Collection Scheduling System

TQS - Software Testing and Quality

Midterm Assignment - Project Report

November 3, 2025

Contents

1	Introduction	2
1.1	Application Use Case	2
1.1.1	Product Name	2
1.1.2	General Purpose	2
1.2	Project Context	2
1.3	Use of Generative AI	2
2	Overview of the work	2
2.1	Current implementation (faults & extras)	2
2.1.1	Known Limitations	2
2.1.2	Extra Features Implemented	3
2.2	Product specification	4
2.2.1	Functional scope and supported interactions	4
2.2.2	System implementation architecture	4
2.2.3	API for developers	5
3	Quality assurance	6
3.1	Overall strategy for testing	6
3.2	Unit and integration testing	7
3.2.1	Unit Tests	7
3.2.2	Integration Tests	7
3.3	Acceptance testing	8
3.4	Non-functional testing	9
3.5	Code quality analysis	10
4	References & resources	11
4.1	Technologies and Frameworks	11
4.2	Documentation	11
4.3	Project Resources	12
4.4	External APIs	12

1 Introduction

1.1 Application Use Case

1.1.1 Product Name

ZeroMonos - Bulk Waste Collection Scheduling System

1.1.2 General Purpose

ZeroMonos is a web application designed for scheduling and managing bulk waste collections in Portuguese municipalities. The system serves as a bridge between citizens who need to dispose of large items (furniture, appliances, mattresses) and municipal waste management services.

The application provides two main user interfaces:

- **Citizen Interface:** Allows residents to schedule, query, and cancel bulk waste collection appointments
- **Staff Interface:** Enables municipal staff to view, filter, and manage booking statuses

The system supports all 308 Portuguese municipalities and enforces business rules such as date validation (no past dates, today, or Sundays), daily booking limits per municipality (32 bookings maximum), and maintains a complete audit trail of status changes.

This project was developed as part of the Software Testing and Quality course to demonstrate testing practices, code quality assurance, and software engineering best practices.

1.2 Project Context

The ZeroMonos system was developed as a midterm assignment for the Software Testing and Quality (TQS) course, focusing on demonstrating comprehensive software engineering practices including testing methodologies, code quality tools, and CI/CD integration.

1.3 Use of Generative AI

Generative AI tools were used throughout the development process to assist with code development, testing, refactoring, and quality improvements. The front-end application was heavily produced with Generative AI to speed up the styling of the HTML pages. All generated code was reviewed before being integrated into the project.

2 Overview of the work

2.1 Current implementation (faults & extras)

2.1.1 Known Limitations

The following features are missing or have known limitations that would be expected in a production-ready system:

Database Persistence

- The application uses H2 in-memory database, which means all data is lost when the application restarts
- No persistent storage layer configured for production use
- **Expected:** Integration with PostgreSQL or MySQL for production deployment

Authentication and Authorization

- No authentication system implemented
- No user accounts or role-based access control
- Staff panel is accessible without any authentication mechanism
- **Expected:** Secure authentication system (JWT, OAuth2, or Spring Security) for staff endpoints

API Pagination

- Staff endpoint `/api/staff/bookings` returns all bookings without pagination
- This can cause performance issues with large datasets
- **Expected:** Pagination parameters (page, size) for efficient data retrieval

Frontend Limitations

- Frontend is built with vanilla HTML/CSS/JavaScript (no modern framework)
- Basic error handling in the user interface
- **Expected:** Modern frontend framework (React, Vue.js, or Angular) and improved user experience

Missing Features (could be implemented if this project kept going)

- No email notifications for booking status changes
- No SMS notifications
- No ability to edit bookings after creation (change time slot or description for example)
- No export functionality (CSV, PDF reports)

2.1.2 Extra Features Implemented

The following additional features were implemented beyond the basic requirements:

CORS Configuration

- Centralized CORS configuration via `CorsConfig` class
- Whitelist-based approach (not using wildcard `*`)
- Configurable allowed origins through `application.properties`
- More secure than permissive `@CrossOrigin(origins = "*")` annotation

History Tracking

- Complete history of booking status changes tracked in `StateChange` entity
- All state transitions are recorded with timestamps
- History is visible in API responses and staff panel interface
- Provides full audit trail for booking lifecycle

API Documentation

- SpringDoc OpenAPI integration for automatic API documentation
- Interactive Swagger UI available at `/swagger-ui.html`
- Automatic API documentation generation from code annotations
- Complete documentation of all REST endpoints

2.2 Product specification

2.2.1 Functional scope and supported interactions

Citizen Functionality

- **Create Booking:** Schedule bulk waste collection by selecting municipality, date, time slot, and providing description
- **Query Booking:** Look up existing booking by token to view status and details
- **Cancel Booking:** Cancel a booking if it's in a cancellable state (RECEIVED or ASSIGNED)
- **List Municipalities:** Get list of all available municipalities (308 Portuguese municipalities)

Staff Functionality

- **List All Bookings:** View all bookings in the system with filtering capabilities
- **Filter by Municipality:** Filter bookings by specific municipality
- **Update Booking Status:** Change booking status through state transitions (RECEIVED → ASSIGNED → IN PROGRESS → COMPLETED → CANCELLED) although staff can jump through status if needed
- **View Booking History:** Access complete history of status changes for any booking

Business Rules

- Date validation: No past dates, today, or Sundays allowed
- Daily limit: Maximum 32 bookings per municipality per day
- Status transitions: Staff can update statuses following business logic
- Cancellation: Only bookings in RECEIVED or ASSIGNED states can be cancelled
- History tracking: All status changes are recorded with timestamps

2.2.2 System implementation architecture

The application follows a layered architecture pattern with clear separation of concerns:

1. Boundary Layer (Controllers)

- BookingController - Public API (`/api/bookings`)
 - POST `/api/bookings` - Create new booking
 - GET `/api/bookings/{token}` - Get booking by token
 - PUT `/api/bookings/{token}/cancel` - Cancel booking

- GET `/api/bookings/municipalities` - List all municipalities
- `StaffBookingController` - Administrative API (`/api/staff/bookings`)
 - GET `/api/staff/bookings` - List all bookings (with optional filters)
 - PATCH `/api/staff/bookings/{token}/status` - Update booking status
- `RestExceptionHandler` - Global exception handling with structured error responses

2. Service Layer

- `BookingServiceImplementation` - Core business logic
 - Date validation using `DateValidator`
 - Booking limit enforcement (32 per municipality)
 - Status change tracking via `StateChange`
 - DTO conversion with error handling
- `MunicipalityImportService` - Municipality data import from external API

3. Data Layer

- Entities: `Booking`, `Municipality`, `StateChange`
- Repositories: `BookingRepository`, `MunicipalityRepository`
- Relationships:
 - `Municipality` → `Booking` (1:N)
 - `Booking` → `StateChange` (1:N) with cascade and orphan removal

4. DTO Layer

- `BookingRequestDTO` - Input validation for booking creation
- `BookingResponseDTO` - Output formatting with history

Technology Stack

- **Backend:** Java 21, Spring Boot 3.5.7, Spring Data JPA
- **Database:** H2 (in-memory), JPA/Hibernate
- **Frontend:** HTML5, CSS3, JavaScript (ES6+)
- **Build:** Maven
- **Documentation:** SpringDoc OpenAPI (Swagger UI)

2.2.3 API for developers

The application provides a RESTful API with comprehensive documentation:

API Documentation

- Interactive Swagger UI available at `/swagger-ui.html`
- OpenAPI JSON specification at `/v3/api-docs`
- Automatic documentation generation from code annotations

- Complete endpoint descriptions, request/response schemas, and examples

API Endpoints

Public Endpoints (/api/bookings):

- POST /api/bookings - Create new booking
- GET /api/bookings/{token} - Get booking details by token
- PUT /api/bookings/{token}/cancel - Cancel booking
- GET /api/bookings/municipalities - List all municipalities

Staff Endpoints (/api/staff/bookings):

- GET /api/staff/bookings - List all bookings (supports municipality filter)
- PATCH /api/staff/bookings/{token}/status - Update booking status

Error Handling

- Structured error responses with HTTP status codes
- Consistent error format: {timestamp, status, error, message, path}
- Custom exceptions for better error categorization
- Global exception handler for consistent error responses

3 Quality assurance

3.1 Overall strategy for testing

The project follows a **testing pyramid** approach with multiple levels of testing to ensure comprehensive coverage:

1. **Unit Tests** (Base of pyramid) - Largest number of tests, fast execution, high isolation
2. **Integration Tests** (Middle layer) - API-level testing, moderate execution time
3. **Acceptance Tests** (Top layer) - BDD scenarios (Cucumber) and E2E tests (Selenium), slower execution

Testing Principles

- **Isolation:** Unit tests use mocks and stubs to isolate components
- **Coverage:** Aim for high code coverage (80%+ for new code)
- **Maintainability:** Tests are well-structured and easy to maintain
- **Speed:** Fast-feedback loop with quick unit tests

Test Distribution

- **Unit Tests:** Tests covering services, controllers, repositories, and utilities
- **Integration Tests:** API tests with RestAssured
- **BDD Tests:** Cucumber scenarios
- **E2E Tests:** Selenium tests for both client and staff interfaces

3.2 Unit and integration testing

3.2.1 Unit Tests

Isolated tests using mocks and stubs:

Service Layer Testing

- `BookingServiceImplementationTest`
 - Tests for booking creation with various validation scenarios
 - Tests for booking retrieval, cancellation, and status updates
 - Tests for exception handling and error scenarios
 - Tests for DTO conversion and edge cases
 - Custom exception constructor testing
- `BookingControllerTest` - MockMvc testing for the user endpoints
- `StaffBookingControllerTest` - Staff endpoints testing
- `RestExceptionHandlerTest` - Exception handling testing

Repository Layer Testing

- `BookingRepositoryTest` - `@DataJpaTest` with `TestEntityManager`
- `MunicipalityRepositoryTest` - Municipality data access testing

Utility Testing

- `TestDate` - Date validation utilities
- `DateValidator` - Date validation logic
- `HistoryMapper` - History mapping utilities

Exception Testing

- Custom exception constructors tested:
 - `SpringDocException` - All constructors tested
 - `DtoConversionException` - All constructors tested
 - `BookingServiceException` - All constructors tested

3.2.2 Integration Tests

API-level testing with RestAssured:

API Tests

- `BookingApiTest`
 - Full API workflow testing
 - Endpoint functionality verification
 - Request/response validation
 - Error handling verification
- `BookingApiEdgeCasesTest`

- Edge cases and boundary conditions
- Concurrent request handling
- Invalid input scenarios
- Error code verification
- **MunicipalityApiLoadTest**
 - Verification of 308 municipalities loaded from external API
 - API endpoint verification
 - Repository consistency checks

Test Configuration

- Testcontainers for database isolation
- Spring Boot test context with random port
- RestAssured for HTTP testing
- Awaitility for asynchronous testing

3.3 Acceptance testing

Acceptance testing is performed using Behavior-Driven Development (BDD) with Cucumber:

BDD Scenarios

*Booking Feature (**booking.feature**):*

- Create booking with valid data
- Create booking with non-existent municipality
- Create booking with past date
- Create booking for Sunday
- Query existing booking
- Query non-existent booking
- Cancel valid booking
- Cancel already cancelled booking
- Complete booking workflow

*Staff Feature (**staff.feature**):*

- List all bookings
- Filter bookings by municipality
- Filter by non-existent municipality
- Update status to ASSIGNED
- Update status to IN_PROGRESS
- Update status to COMPLETED

- Update status of non-existent booking
- Complete staff workflow

End-to-End Testing

Selenium WebDriver tests for UI functionality:

Client View (ClientViewSeleniumTest):

- Form loading and municipality autocomplete
- Form submission with valid data
- Form validation for required fields
- Booking lookup by token
- Invalid token handling
- Booking cancellation from lookup page

Staff View (StaffViewSeleniumTest):

- Booking table loading
- Municipality filtering
- Filter clearing
- Status updates
- History viewing
- Empty state handling

Testing Approach

- Explicit waits using `WebDriverWait` instead of `Thread.sleep()` to avoid issues on SonarQube
- Robust element waiting with `ExpectedConditions`
- Modern Selenium API usage (`getDomProperty`, `getDomAttribute`)
- Asynchronous testing with `Awaitility`

3.4 Non-functional testing

Non-functional testing covers aspects beyond functional correctness:

Performance Testing

- Concurrent request handling (`BookingApiEdgeCasesTest`)
- Multiple simultaneous bookings for same municipality
- Load testing with concurrent threads
- Response time verification

Reliability Testing

- Error handling and recovery

- Exception propagation testing
- Database connection handling
- External API integration resilience

Usability Testing

- Selenium tests verify UI functionality
- Form validation feedback
- Error message clarity
- User interaction flows

Security Considerations

- CORS configuration testing
- Input validation testing
- SQL injection prevention (via JPA)
- XSS prevention in frontend

3.5 Code quality analysis

Code quality is ensured through multiple tools and practices:

SonarQube/SonarCloud Analysis

- **Code Coverage:** 80%+ for new code
- **Code Smells:** All identified issues resolved
- **Bugs:** No bugs detected
- **Vulnerabilities:** No security vulnerabilities found
- **Technical Debt:** Minimal technical debt
- **Quality Gate:** Passing all quality gates

JaCoCo Coverage Reports

- Automatic coverage reports generated on test execution
- HTML reports available at `target/site/jacoco/index.html`
- Coverage metrics tracked per package and class
- Branch coverage analysis
- Line coverage analysis

Quality Improvements

- Custom exception classes for better error handling
- Utility classes with private constructors
- Improved logging practices (avoiding duplicate logging)

- Code refactoring for better maintainability
- Almost all SonarQube issues resolved

Quality Metrics

- **Coverage on New Code:** 80%+ (target: 80%)
- **Duplications:** 0.0% (target: 3.0%)
- **Security Hotspots:** 0 (target: 0)
- **New Issues:** 0 (target: 0)
- **Maintainability Rating:** A
- **Reliability Rating:** A
- **Security Rating:** A

Note: all metrics can be seen in the docs or in the README.md in the project repository.

4 References & resources

4.1 Technologies and Frameworks

- Spring Boot 3.5.7 - <https://spring.io/projects/spring-boot>
- Java 21 - <https://www.oracle.com/java/>
- JUnit 5 - <https://junit.org/junit5/>
- RestAssured 5.4.0 - <https://rest-assured.io/>
- Cucumber 7.18.0 - <https://cucumber.io/>
- Selenium 4.19.1 - <https://www.selenium.dev/>
- JaCoCo - <https://www.jacoco.org/jacoco/>
- SonarQube/SonarCloud - <https://www.sonarqube.org/>
- SpringDoc OpenAPI - <https://springdoc.org/>
- Maven - <https://maven.apache.org/>

4.2 Documentation

- Spring Boot Documentation - <https://docs.spring.io/spring-boot/docs/current/reference/html/>
- Spring Data JPA Documentation - <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>
- REST Assured Documentation - <https://rest-assured.io/docs/>
- Cucumber Documentation - <https://cucumber.io/docs/cucumber/>
- Selenium Documentation - <https://www.selenium.dev/documentation/>

4.3 Project Resources

- GitHub Repository: <https://github.com/goncaloosimoes/ZeroMonos>
- SonarCloud Project: https://sonarcloud.io/project/overview?id=goncaloosimoes_ZeroMonos
- Video Demonstration: <https://github.com/goncaloosimoes/ZeroMonos/blob/main/video.mp4>
- API Documentation: Available at `/swagger-ui.html` when running locally
- Code Coverage Report: Available at `target/site/jacoco/index.html` after running tests

4.4 External APIs

- Portuguese Municipalities API: <https://json.geoapi.pt/municipios> - Used for loading municipality data