2024

Implementation Sprint 1 – TruckMotion

VERSION 1.0

NUNO MARMELEIRO 1190922
ROGÉRIO SOUSA 1191017
RAFAEL FAÍSCA 1180658
ÓSCAR FOLHA 1181600
RAFAEL OLIVEIRA 1191611

Index

Index of Figures

Index of Code Snippets

Document Approval

Name	Date	Signature
Rogério Sousa	19/05/2024	Rogério Sousa
Óscar Folha	19/05/2024	Óscar Folha
Rafael Faísca	19/05/2024	Rafael Faísca
Nuno Marmeleiro	19/05/2024	Nuno Marmeleiro
Rafael Oliveira	19/05/2024	Rafael Oliveira

Revision history

Version	Author	Description	Date
1.0	Everyone	Final Version	19-05-2024

1 Introduction

1.1 Purpose

This document serves the purpose of documenting certain aspects of the application developed in this sprint 1, the pipeline and the method of work.

In this sprint 1, the start of the implementation phase of secure software development life cycle was the main objective.

In more detail, sprint 1 aims to develop some use cases of the application documented before, development of the application pipeline having in mind security with tools described before to achieve SAST, DAST and SCA.

It was also configured some properties in terms of security, such as using https.

1.2 Document conventions

For the development of this document some conventions were followed, such as:

- IEEE bibliographic citation style.
- For the development of domain models, logical data model and use case diagrams it was used the Unified Modeling Language (UML) notation. [1]

1.3 Business Logic – What is TruckMotion?

TruckMotion is an application designed for package delivery management. Customers request deliveries through the app and managers assign these requests to the most suitable drivers to ensure packages are delivered to the correct destinations.

Customers enter personal information such as their name, email, and birthdate, and can connect multiple locations to their account. They also provide information about the companies they work for. Managers assign service requests to drivers and manage accounts. Drivers receive their assignments from managers, carry out the delivery and provide proof of delivery with a photo of the destination.

When customers request a delivery, they provide details like the service name, total weight of the items and expected delivery date. The status of a service can be active, finished, canceled, or pending, which customers can monitor through the app. Drivers update the status as they transport packages, marking a service as finished upon delivery.

TruckMotion ensures smooth package delivery operations, offering a system for customers to request services, managers to assign tasks and drivers to complete deliveries with proof of arrival.

1.4 Run the Application

Node is needed and used to create scripts for the management of the application.

The backend application is run locally by writing the command ***./mvnw***, however, it is intended to start the application with ***./mvnw "-Pdev,tls"*** so we have our TLS profile running.

For the front end, the application is run by the command ***npm run webapp:dev***.

1.5 Accomplished

Some basic Use Cases implementation was accomplished, such as creation/edition of users, service requests, transports and the development of the pipeline using SonarQube, OWASP ZAP and running the tests developed for the application. Achieving the use of some tools of SAST and IAST. Finally, the pipeline also does the deployment in a free service available online.

Adding to this, the security configuration was also achieved, including the authorization, authentication and encrypt algorithms for sensible data, such as passwords.

2 Domain-Driven Design

The Domain-Driven Design Diagram is represented in Figure 1.

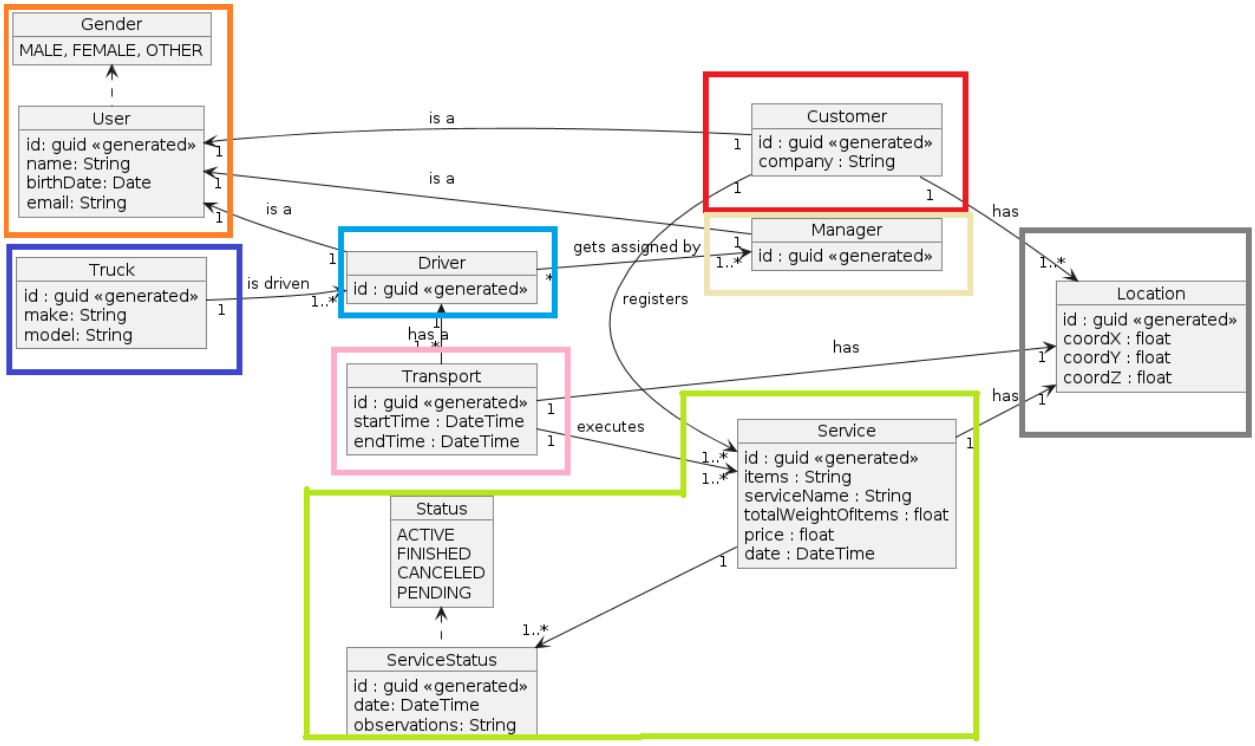

Figure 1 - Domain-Driven Design Diagram

The Manager is responsible for creating Drivers and Customers accounts and generally manages the application. The Customers asks for Services to be delivered at the Location desired and the Manager assigns a Driver with a Truck already assigned to execute the

Transport of the Customer's Service. Finalizing the Transport, the Driver updates the Service Status to Finished.

It is visible eight Aggregates:

- User – composed with User entity and Gender Enum.
- Truck – composed with Truck entity.
- Driver - composed with Driver entity.
- Transport - composed of the Transport entity.
- Service - composed of Service and ServiceStatus entities and the Status Enum.
- Location - composed with Location entity.
- Manager – composed with Manager entity.
- Customer - composed with Customer entity.

2.1 Application of DDD

As it was used JHipster to generate the application, the Domain-driven Design (DDD) wasn't implemented initially because one con about JHipster is that it is not compatible with this methodology.

So, there were made some changes on the application in order to have DDD applied on it:

- Resource Classes to Controller
  - JHipster generates Resource classes instead of Controllers, so those classes was changed to Controller.
- Repositories interfaces
  - It was created interfaces to have a standardized interface for each Entity Root.
- Value Objects
  - JHipster is not compatible with the usage of Value Objects. So, it was necessary to change the entire domain to be able to use them.
- Entity Roots
  - As DDD wasn't implemented initially, the concept of Entity Root and Aggregates didn't exist as well. So, this required some changes, especially on ServiceRequest and ServiceStatus entity. Because the ServiceRequest entity, being the root, is responsible for every change on its aggregate.
  - This included the removal of Repositories, Services and Controllers of ServiceStatus and the correction on ServiceRequest classes, in order to manage the ServiceStatus entity.
- Correct separation of Folders
  - The "application/controller" includes all the Controllers of each entity root.
  - The "infrastructure/repository" was created to have the Interface repositories for each Entity Root.
  - The "domain" folder includes the Service, Value objects and the Domain classes.

3 Deployment

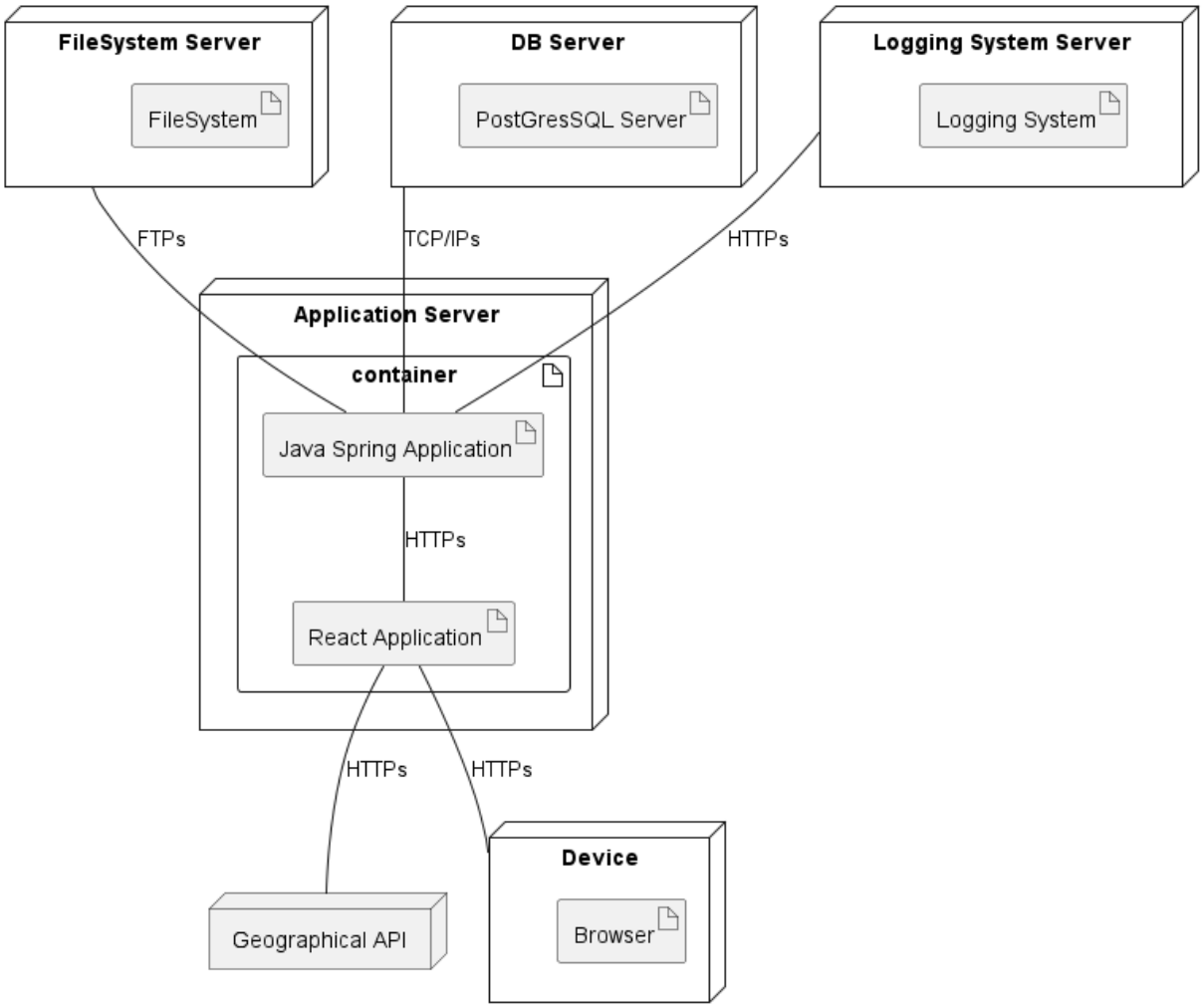

Figure 2 - Deployment Diagram

Our application now is composed based on Figure 2. We have our Postgres DB in a separate server, our backend and frontend components are combined in a docker image, and its container is deployed in a render server.

Even though the most secure way to deploy the application is having every artifact in different physical machines is acknowledged, this was not performed because the resources available for this project are academic and not professional. It is known that the application services are a weak point of the security and with the needed resources, the application should be based on the deployment diagram in the design documentation.

The link to our app is here.

FIY: Render is a free service; the application might be down when you try to access it. Contact one of our team members if you need to use it.

4 JHipster Overview

4.1 What is JHipster?

JHipster is a development platform used to generate, develop, and deploy Spring Boot applications combined with modern JavaScript frameworks such as Angular, React, or Vue. It integrates a wide array of best-in-class technologies and tools aimed at enhancing code generation, database management, and overall development efficiency.

4.2 How JHipster was Used for TruckMotion

In the development of the TruckMotion application, we utilized JHipster primarily through its generator. This tool scaffolded the complete project setup, including a Spring Boot back end and a modern front end using React. JHipster's generator facilitated the creation of a monolithic application structure that efficiently supported our initial requirements for the TruckMotion project.

We further leveraged JHipster Domain Language (JDL) to generate entities and their relationships, simplifying the design of the database schema and ensuring seamless integration with the application code. This was particularly useful for modeling complex entities like customers, drivers, and delivery requests within TruckMotion.

Once the application was generated, JHipster provided a streamlined development workflow. This included features such as hot reloading, which updated the application in real-time as changes were made, enhancing our development efficiency. Additionally, JHipster's support for continuous integration and continuous deployment (CI/CD) pipelines, with tools like Jenkins and GitLab CI, facilitated automated build and deployment processes, ensuring that our development and deployment processes were efficient and reliable. In this case, it was used github actions to comply with our method of work.

For security, we integrated Spring Security through JHipster, which provided out-of-the-box authentication and authorization solutions. JHipster's support for several security protocols, including OAuth2, JWT, and traditional session-based authentication, was crucial in ensuring that only authorized users could access specific parts of the TruckMotion application. These features were essential for protecting sensitive user data and maintaining secure interactions within the application.

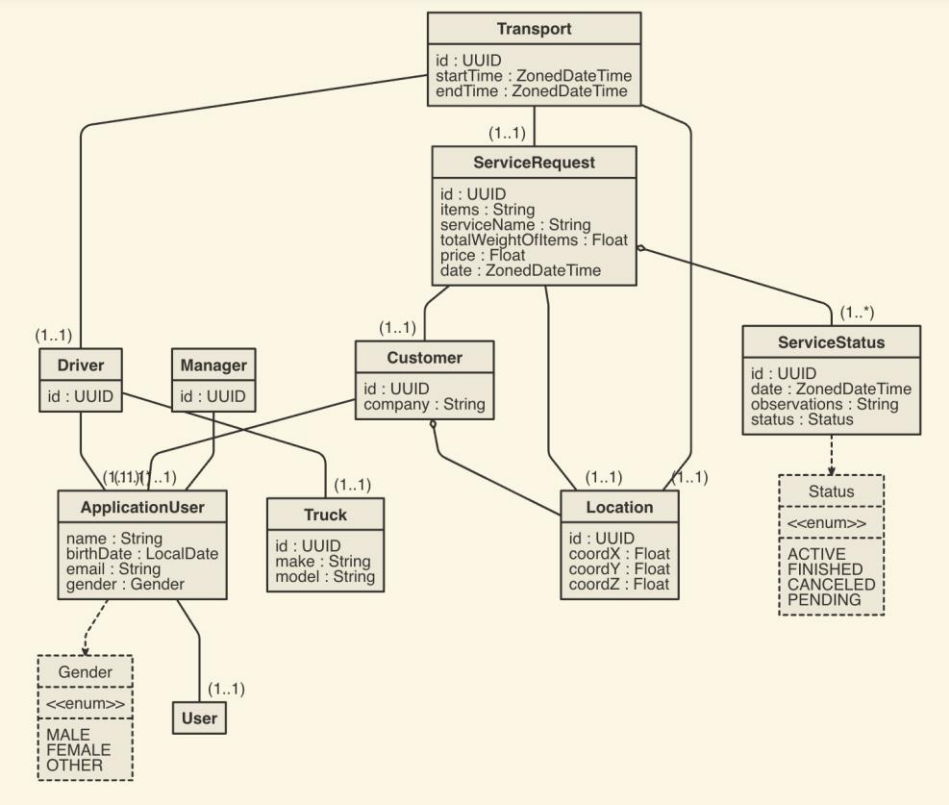

Figure 3 - Jhipster jdl

4.3 Security Vulnerabilities and Mitigation

Using JHipster involves dealing with a variety of libraries and dependencies, each of which can introduce potential vulnerabilities. One of the main challenges is dependency management. Given the rapid evolution of libraries and frameworks, it's crucial to regularly update these dependencies to patch any known vulnerabilities. Tools like Dependabot and Snyk can be integrated into the development workflow to automatically monitor and manage these updates.

Web applications generated by JHipster are susceptible to common security risks such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). XSS vulnerabilities can be mitigated by implementing proper input validation and output encoding. For CSRF, JHipster's integration with Spring Security provides built-in mechanisms to protect against these attacks, including the use of CSRF tokens.

SQL injection is another significant risk. JHipster mitigates this by leveraging Hibernate, an Object-Relational Mapping (ORM) tool that helps in constructing safe database queries. By using Hibernate's parameterized queries and avoiding concatenation of SQL queries, the risk of SQL injection is significantly reduced.

Additionally, implementing security headers such as Content Security Policy (CSP), X-Content-Type-Options, and X-Frame-Options adds another layer of protection by controlling how resources are loaded and restricting the framing of content, which helps prevent various forms of attacks.

4.4 Libraries and Tools used

For the development of TruckMotion, we chose to use JHipster due to its robust integration of best-in-class technologies. The backend of our application is built using Spring Boot, which provides a reliable framework for creating production-ready applications. We seamlessly integrated Spring Boot with Hibernate for ORM to ensure efficient database interactions. Additionally, we used Liquibase for managing database migrations, which helped us handle schema changes across different environments effectively.

For the front end, we opted for JHipster's support of modern JavaScript frameworks, specifically React. This enabled us to create a dynamic and responsive user interface. We utilized Webpack as our module bundler to compile JavaScript modules into single files or smaller chunks, optimizing the application's performance.

Security was a top priority in our development process. We leveraged Spring Security within JHipster to handle authentication and authorization. By utilizing JWT and OAuth2, we ensured flexible and secure access to the application, protecting sensitive user information and managing user sessions effectively.

In our DevOps practices, we decided to use Docker for containerization. Docker allowed us to encapsulate the application and its dependencies in containers, ensuring consistent performance across different environments. This approach addressed many environment-specific issues, providing predictable behavior and simplifying the deployment process. Furthermore, we integrated CI/CD tools such as Jenkins and GitLab CI to automate the build, test, and deployment processes. This automation was crucial in maintaining continuous integration and deployment with minimal manual intervention.

5 Use Cases

Some use cases were already implemented in this Sprint. However, since it wasn't the focus of it, some UCs are simple and not entirely completed in terms of functionalities and security. It will be improved on the next Sprint.

The following use cases were implemented:

- UC1 - As a Customer, I want to be able to register delivery services so that I can have the parcel delivered to my Location. I should only register delivery services for myself.
- UC3 - As a customer, I want to be able to add Locations associated to me so that I can choose one of them in the Service. I should only be able to add locations to myself.
- UC8 - As a manager, I want to be able to approve or reject a job so that I can control the Services. I should only approve and reject jobs if they are registered in the system by clients.
- UC9 - As a manager, I want to be able to dispatch services to drivers so that the Services are executed. I should only dispatch services if the requests are approved.
- UC10 - As a manager, I want to be able to register Drivers in the application so that the Transports can be executed by them. I should only be able to register them and not update their data.

- UC11 - As a manager, I want to be able to register Customers in the application so that I have the possibility of new Services be asked. I should only be able to register them and not update their data.
- UC12 - As a user of the system, I want to be able to login into the application so that I can access the application features. I should not be able to access the system without valid login credentials.
- UC13 - As a user of the system, I want to be able to change my password following the correct rules. I should only be able to change my password.

5.1 UC1 - As a Customer, I want to be able to register delivery services so that I can have the parcel delivered to my Location.

This use case is available on the following procedures:

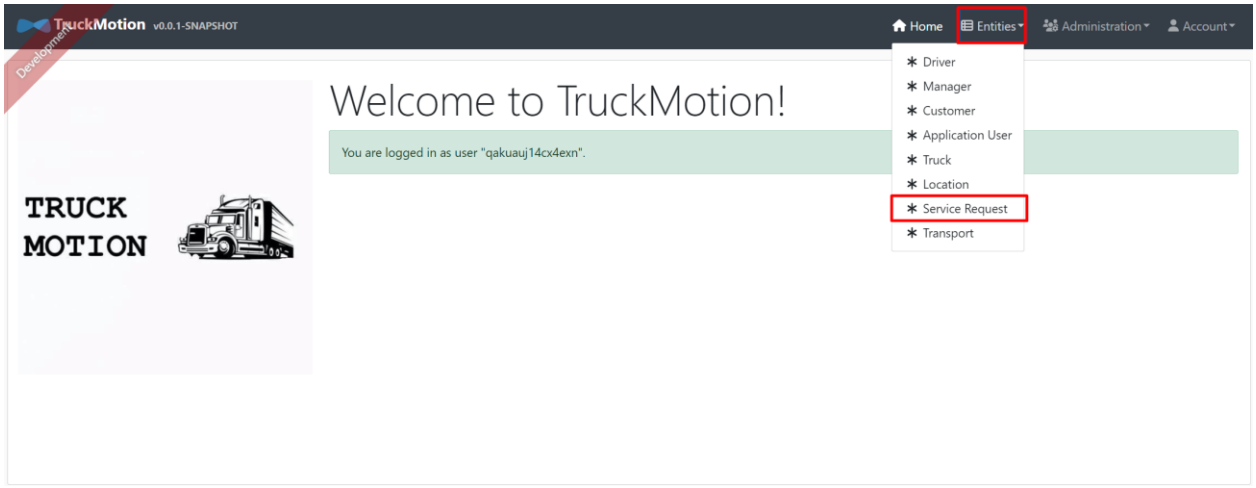

Figure 4 - Creation of Service Request (1)

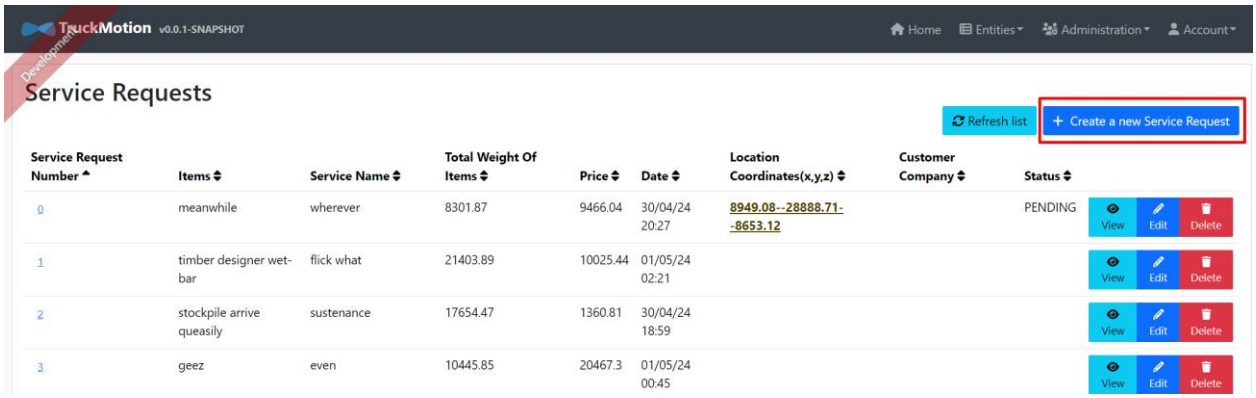

Figure 5 - Creation of Service Request (2)

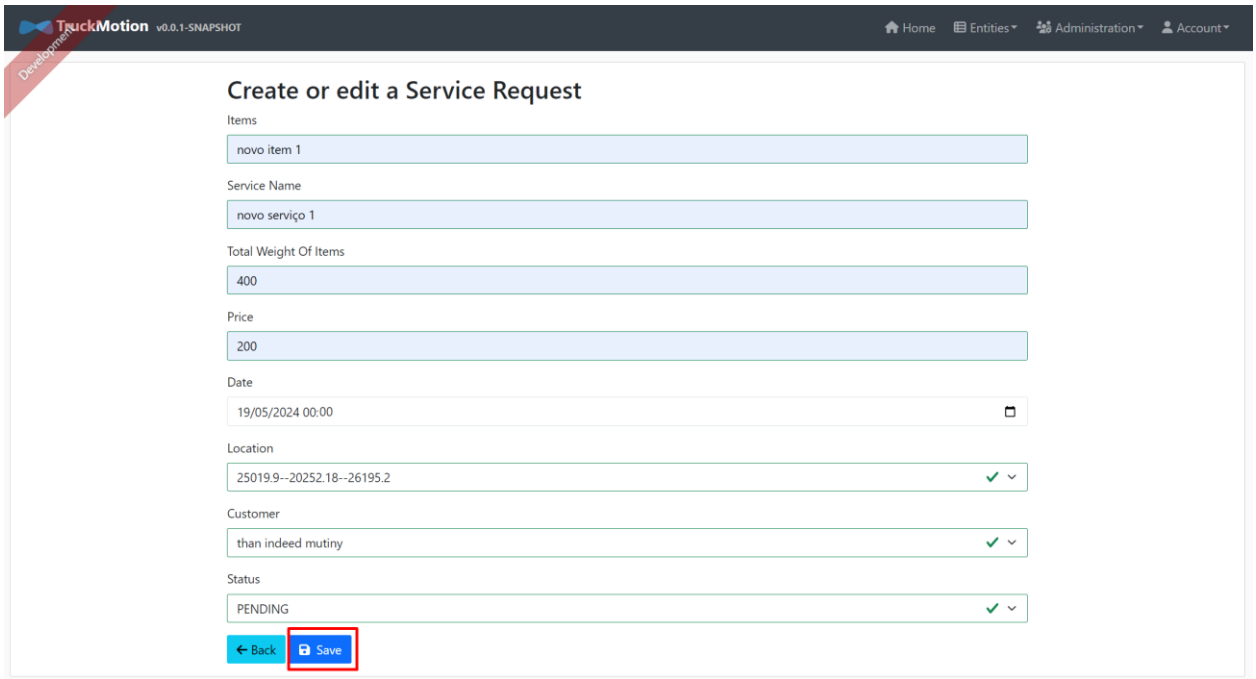

Figure 6 - Creation of Service Request (3)

The Status field returns the actual status of the Service, and it only changes the status when it is a different one. So, the workflow of status is not implemented yet, the user can change the status to anything.

Also, the Locations list shows all the Locations in the system, it must be improved to only show the Locations of the Customer selected and the Customer should be selected automatically for the user logged in.

5.2 UC3 - As a customer, I want to be able to add Locations associated to me so that I can choose one of them in the Service.

This use case is available on the following procedures:

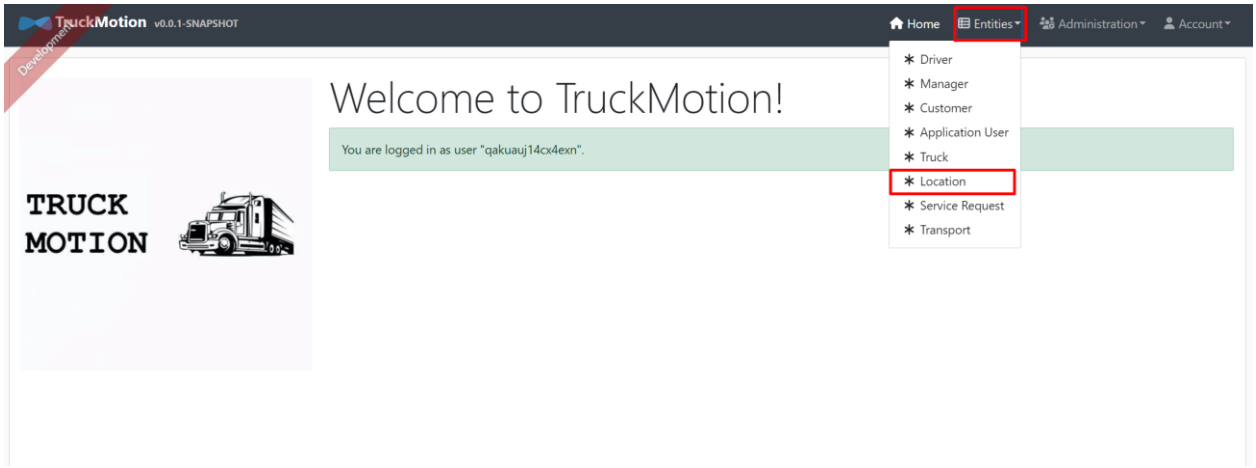

Figure 7 - Associate Locations to Customer (1)

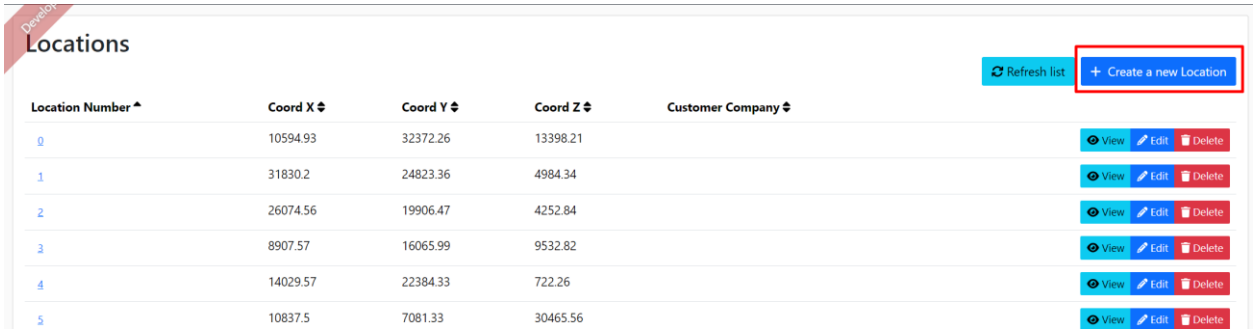

Figure 8 - Associate Locations to Customer (2)

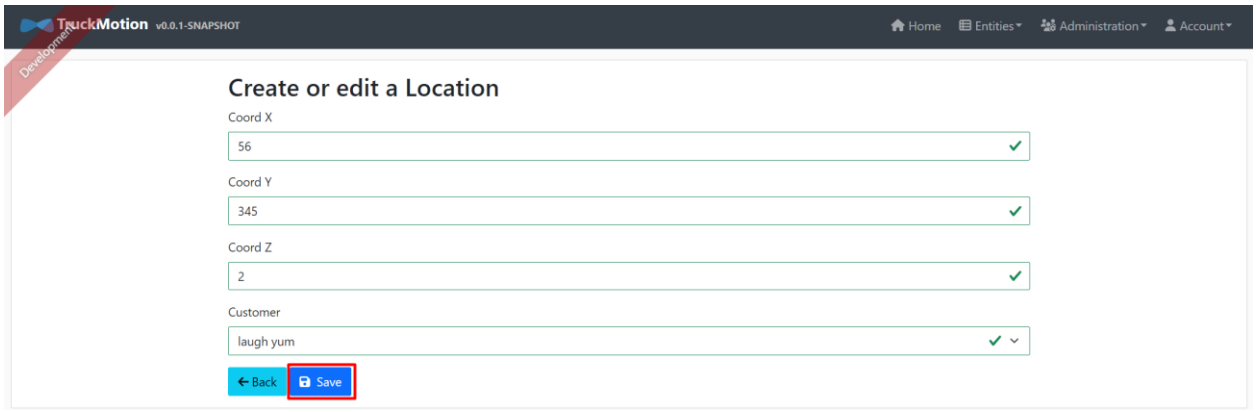

Figure 9 - Associate Locations to Customer (3)

This use case is in a simple phase of implementation. In the future, this will be altered to be more secure and much more appropriate for the application. At this point, this only allows to create the Location and associate a Customer to it.

5.3 UC8 - As a manager, I want to be able to approve or reject a job so that I can control the Services. I should only approve and reject jobs if they are registered in the system by clients.

To approve or reject the Services, it is done on the Services Edit page:

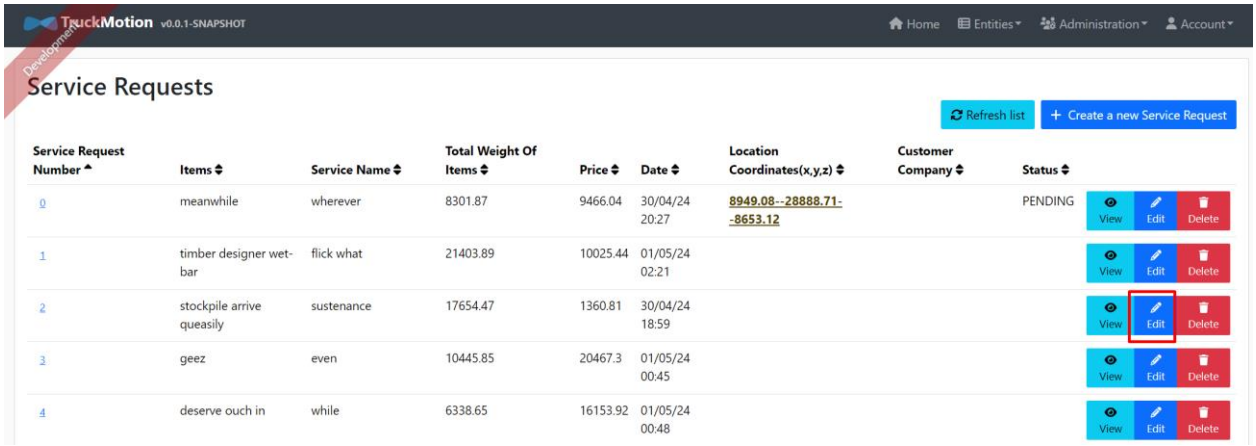

Figure 10 - Approve/Reject Services (1)

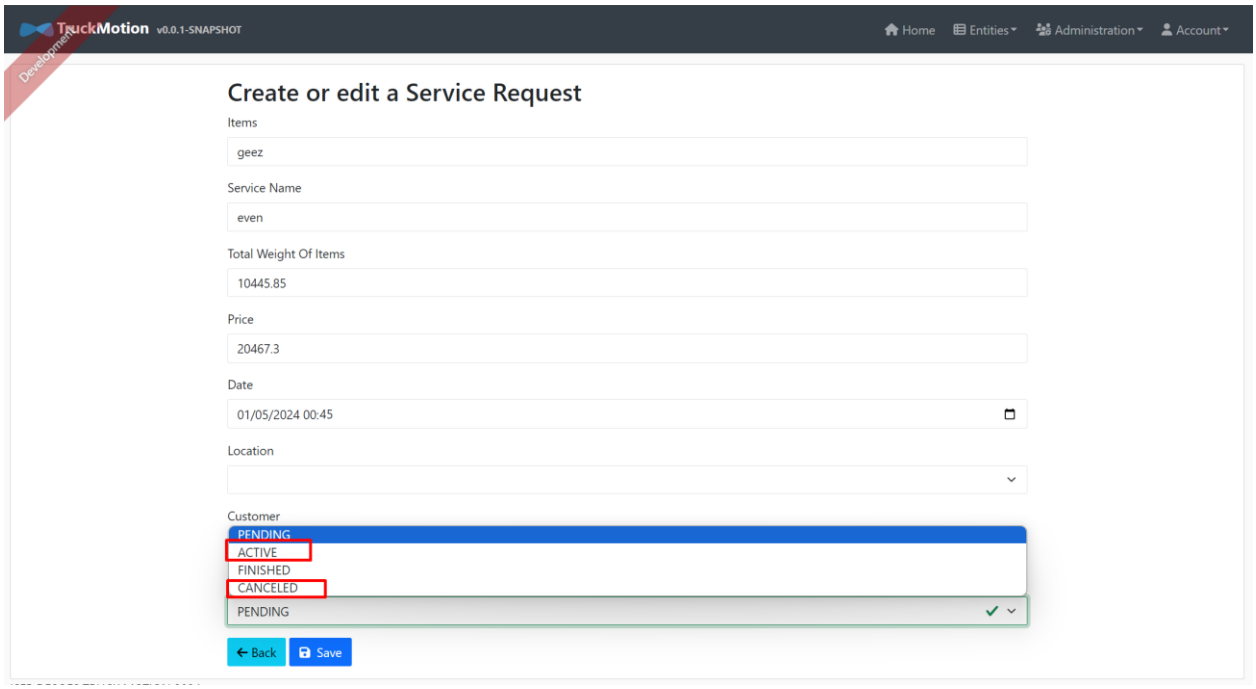

Figure 11 - Approve/Reject Services (2)

In the future, this will be improved to be more secured. For example, it will be available only for the Manager.

5.4 UC9 - As a manager, I want to be able to dispatch services to drivers so that the Services are executed. I should only dispatch services if the requests are approved.

This use case is also in a simple phase of implementation. It allows to create Transport entities and associate a ServiceRequest and a Driver manually. It is done by following the next steps:

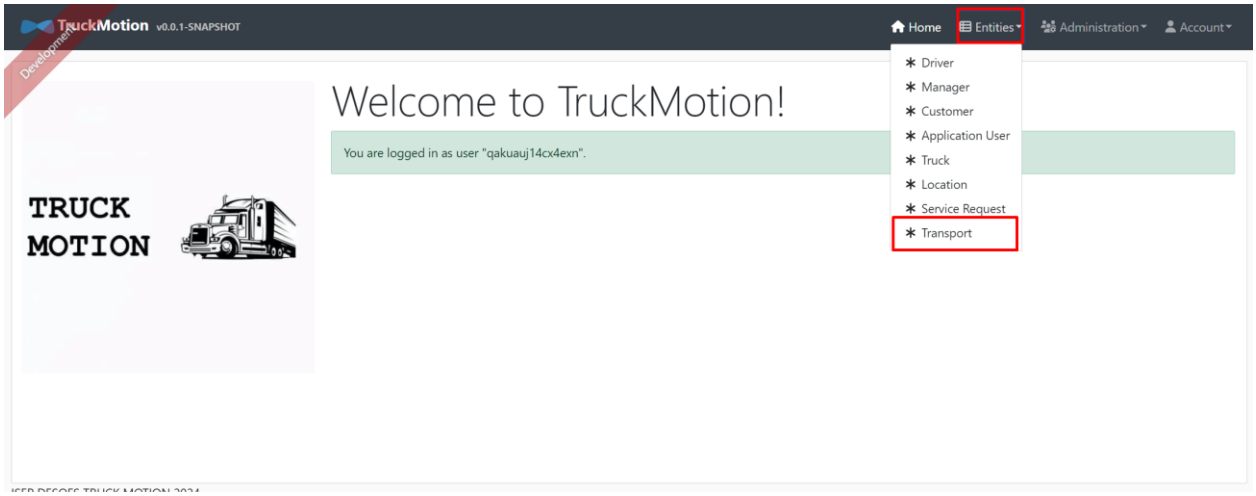

Figure 12 - Creation of Transports (1)

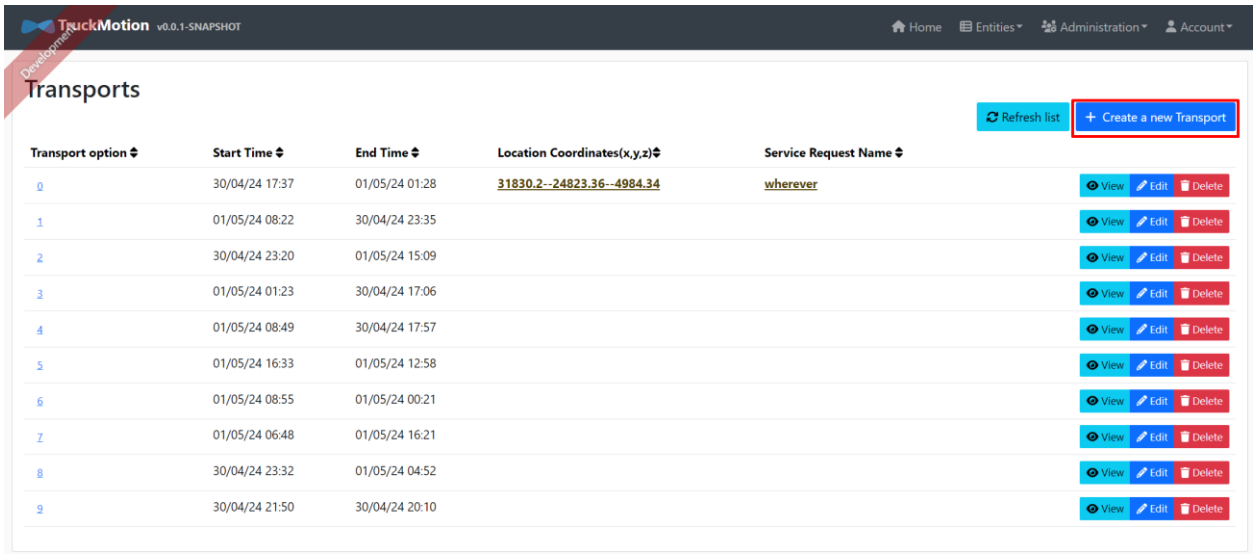

Figure 13 - Creation of Transports (2)

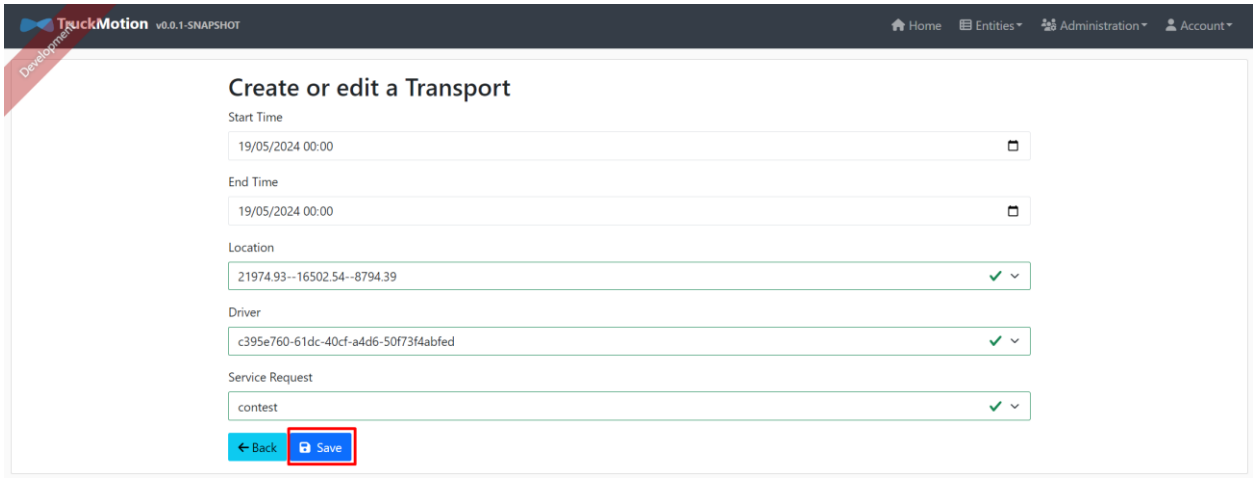

Figure 14 - Creation of Transports (3)

As can be seen many things shall be changed. For example, the Driver field is showed by the id of the entity instead of its name, the Location can be chosen, and it shouldn't, it should be associated automatically to the ServiceRequest's Location.

5.5 UC10 - As a manager, I want to be able to register Drivers in the application so that the Transports can be executed by them.

The creation of Drivers is almost completed, however there are some changes that must be made like being only available for the Managers. It is done with the following process:

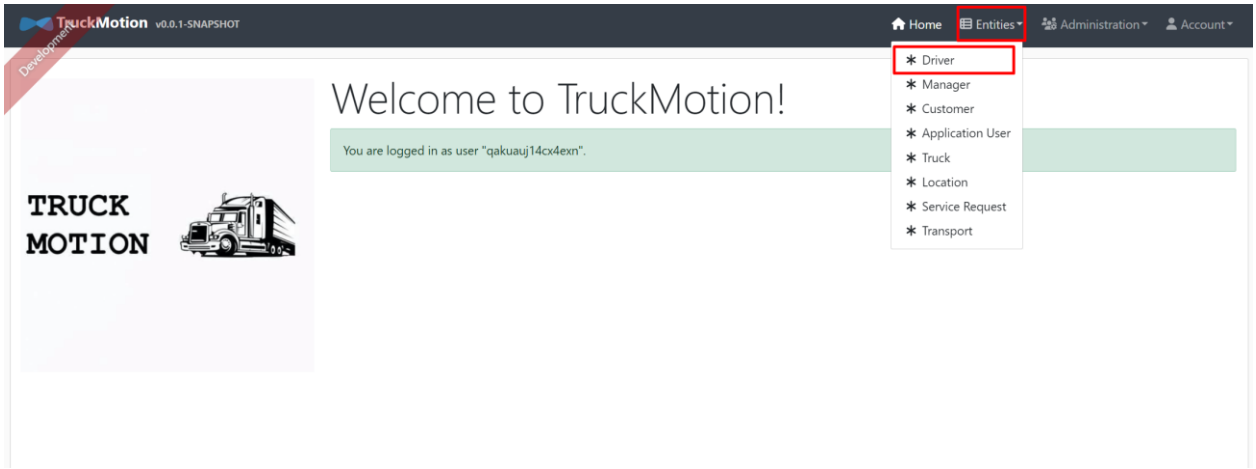

Figure 15 - Creation of Drivers (1)

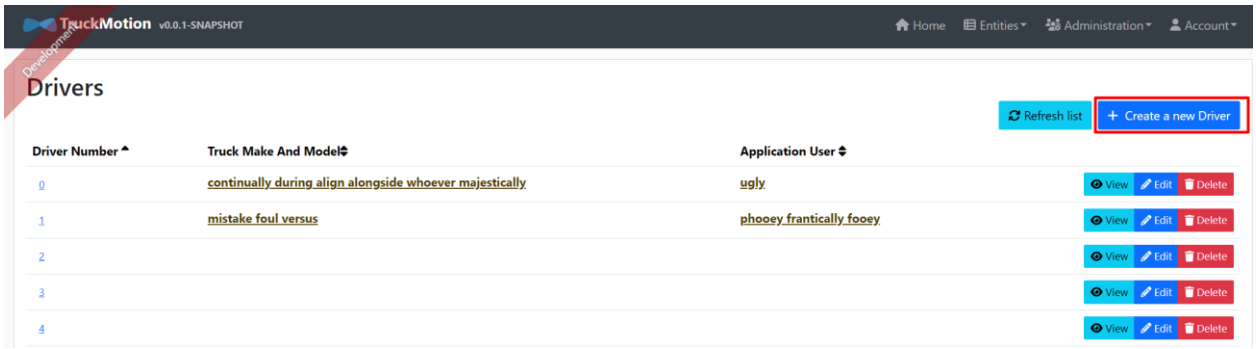

Figure 16 - Creation of Drivers (2)

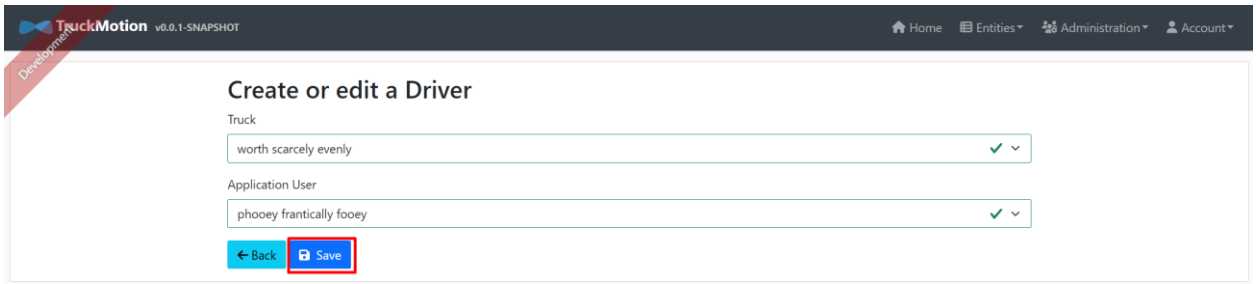

Figure 17 - Creation of Drivers (3)

5.6 UC11 - As a manager, I want to be able to register Customers in the application so that I have the possibility of new Services be asked. I should only be able to register them and not update their data.

The creation of Customers is almost completed, however there are some changes that must be made like being only available for the Managers. It is done with the following process:

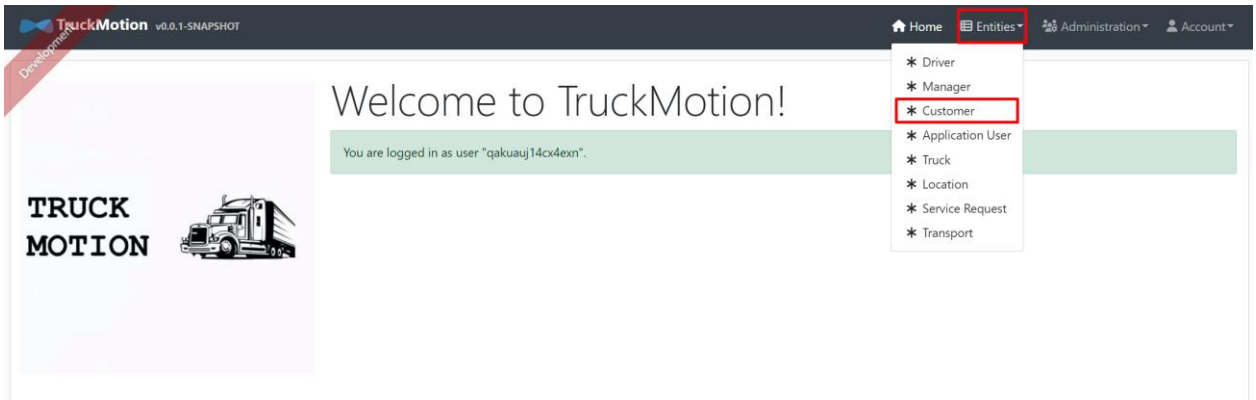

Figure 18 - Creation of Customers (1)

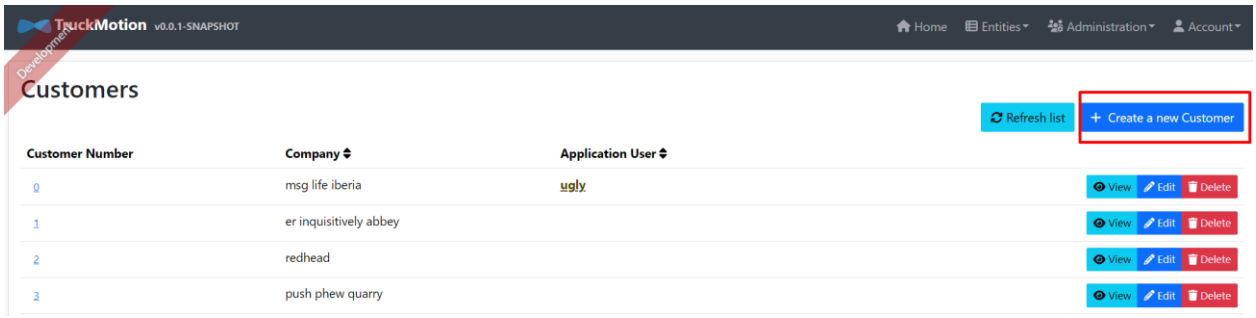

Figure 19 - Creation of Customers (2)

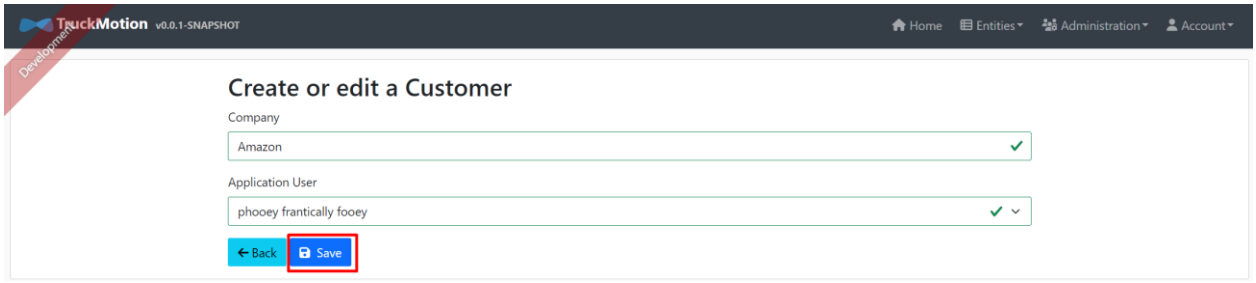

Figure 20 - Creation of Customers (3)

5.7 UC12 - As a user of the system, I want to be able to login into the application so that I can access the application features. I should not be able to access the system without valid login credentials.

The login is completed, and it can be done doing the following steps:

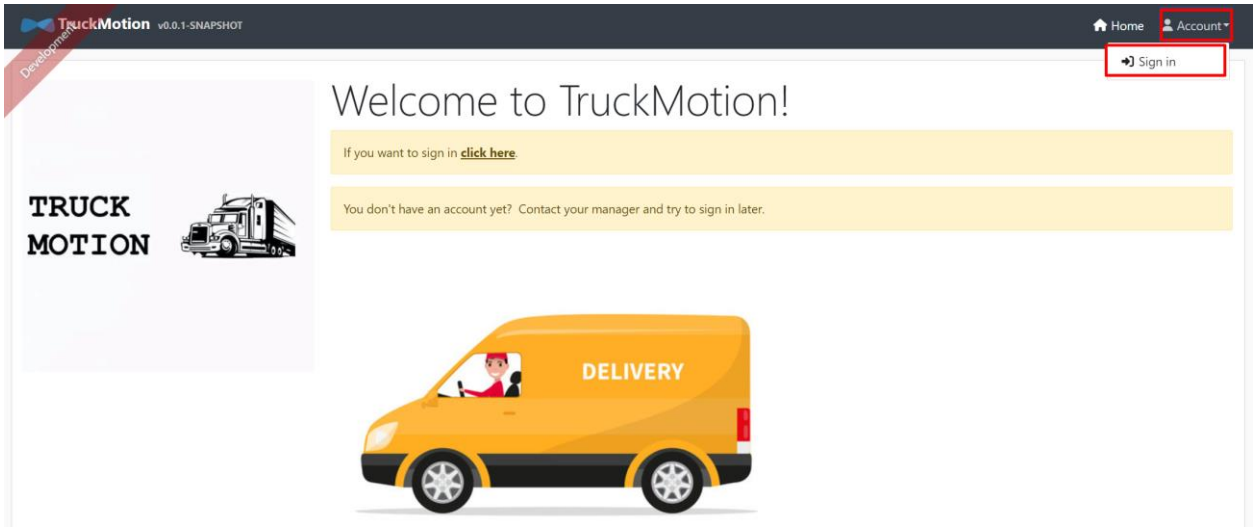

Figure 21 - Login (1)

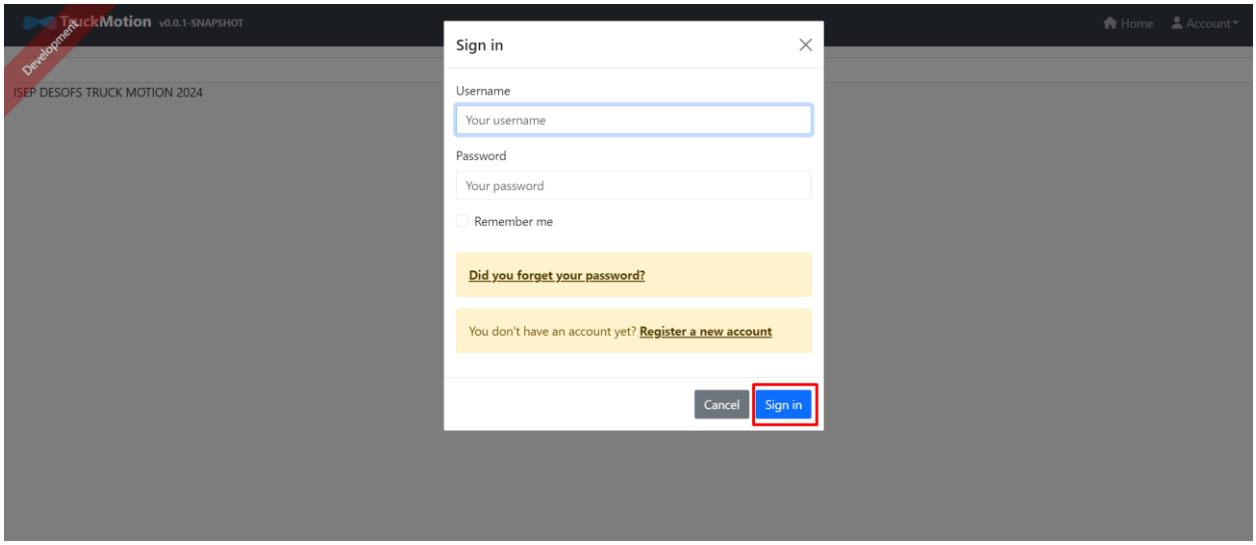

Figure 22 - Login (2)

5.8 UC13 - As a user of the system, I want to be able to change my password following the correct rules. I should only be able to change my password.

The user can already change his password. This follows the correct rules specified and it can be done with the following procedure:

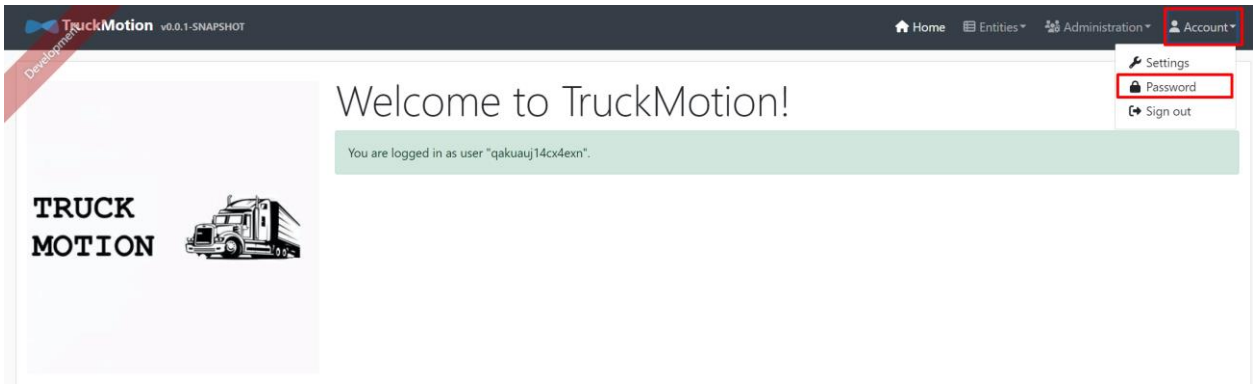

Figure 23 - Change password (1)

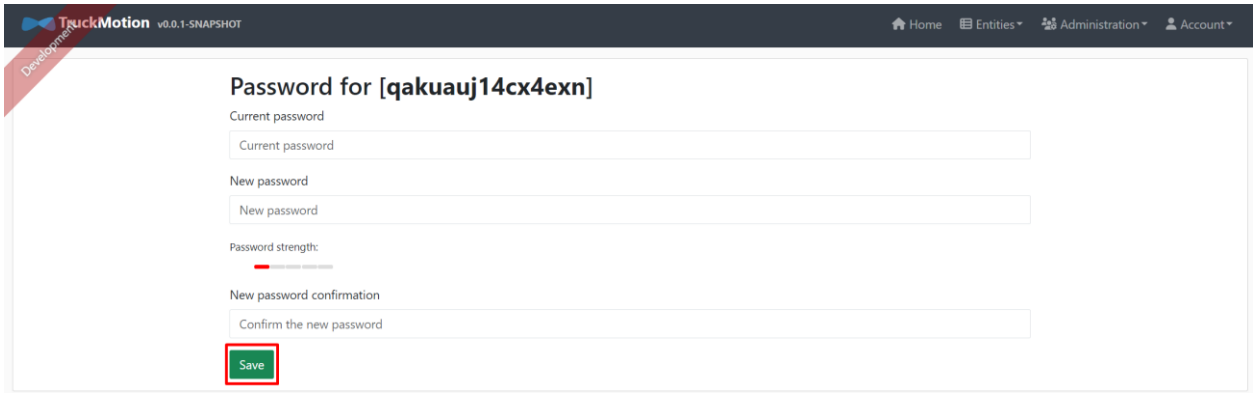

Figure 24 - Change password (2)

6 Security Configuration

By integrating Spring Security, our application benefits from a comprehensive and customizable security framework. This includes CSP, CORS, JWT token authentication, and robust password encryption, ensuring the application is secure against various threats and vulnerabilities.

The security configuration is managed in the ***SecurityConfiguration.java*** class. This setup includes Content Security Policy (CSP), Cross-Origin Resource Sharing (CORS), permission policies, and endpoint access controls.

6.1 Key Security Features

- ***CSP and CORS Configuration***
  - **CSP (Content Security Policy)**: Protects against XSS (Cross-Site Scripting) attacks by specifying trusted sources for content. Defined in the YAML configuration as:

```
jhipster:
  security:
    content-security-policy: "default-src 'self'; frame-src 'self' data:; script-src 'self' 'unsafe-inline' 'unsafe-eval'
https://storage.googleapis.com; style-src 'self' 'unsafe-inline'; img-src 'self' data:; font-src 'self' data:"
```

Code Snippet 1 - CSP Configuration

- ***default-src 'self'***
  - Only content (e.g., scripts, styles, images) from the same origin (the application's own domain) is allowed by default. This is a general fallback directive for other content types not explicitly specified.
- ***frame-src 'self' data***
  - Allows the application to display content in iframes from the same origin and from ***data:*** URIs. This can be useful for embedding small amounts of content directly within the HTML document.
- ***script-src 'self' 'unsafe-inline' 'unsafe-eval'*** *https://storage.googleapis.com*
  - Scripts can be loaded and executed from the same origin (self).

    - ***unsafe-inline*** allows inline JavaScript to be executed, which can pose security risks but may be necessary for certain functionalities.
    - ***unsafe-eval*** allows the use of eval() and similar methods for executing code, which can also pose security risks.
    - Scripts from https://storage.googleapis.com are explicitly allowed, enabling the application to load scripts stored there.
- ***style-src 'self' 'unsafe-inline'***
    - Styles can be loaded from the same origin (***self***).
    - ***unsafe-inline*** allows inline CSS styles to be applied, which can be a security risk but may be necessary for certain functionalities.
- ***img-src 'self' data***
    - Images can be loaded from the same origin (***self***).
    - ***data:*** URIs are allowed for images, enabling the embedding of small images directly within the HTML document.
- ***font-src 'self' data***
    - Fonts can be loaded from the same origin (***self***).
    - ***data:*** URIs are allowed for fonts, enabling the embedding of fonts directly within the HTML document.
- **Security Implications**
    - ***'self'***: Restricts most resources to be loaded only from the same origin, reducing the risk of cross-site scripting (XSS) and other injection attacks.
    - ***'unsafe-inline'*** and ***'unsafe-eval'***: While necessary for some applications, these directives significantly weaken the policy by allowing inline scripts and the use of *eval()*, which can make the application vulnerable to XSS attacks.
    - Specific External Sources: Allowing scripts from specific external sources (like https://storage.googleapis.com/) ensures that only trusted sources are used.
        - **CORS (Cross-Origin Resource Sharing):** Controls how resources are shared between different origins to enhance security. Defined as:

```
jhipster:
  cors:
    allowed-origins: "http://localhost:8100,http://localhost:9000"
    allowed-methods: "*"
    allowed-headers: "*"
    exposed-headers: "Authorization,Link,X-Total-Count,X-${jhipster.clientApp.name}-alert,X-${jhipster.clientApp.name}-
error,X-${jhipster.clientApp.name}-params"
    allow-credentials: true
    max-age: 1800
```

Code Snippet 2 - CORS (Cross-Origin Resource Sharing)

- **Endpoint Authorization**
    - Configures which endpoints require authorization, which are public, and which require specific roles.
    - Example configuration in ***SecurityConfiguration.java***:

```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http, MvcRequestMatcher.Builder mvc) throws Exception {
    http
        .cors(withDefaults())
        .csrf(csrf -> csrf.disable())
        .addFilterAfter(new SpaWebFilter(), BasicAuthenticationFilter.class)
        .headers(headers -> headers
            .contentSecurityPolicy(csp -> csp.policyDirectives(jHipsterProperties.getSecurity().getContentSecurityPolicy()))
            .frameOptions(FrameOptionsConfig::sameOrigin)
            .referrerPolicy(referrer ->
referrer.policy(ReferrerPolicyHeaderWriter.ReferrerPolicy.STRICT_ORIGIN_WHEN_CROSS_ORIGIN))
            .permissionsPolicy(permissions -> permissions.policy("camera=(), fullscreen=(self), geolocation=(), gyroscope=(),
magnetometer=(), microphone=(), midi=(), payment=(), sync-xhr=()"))
        )
        .authorizeHttpRequests(authz -> authz
            .requestMatchers(mvc.pattern("/index.html"), mvc.pattern("/*.js"), mvc.pattern("/*.txt"), mvc.pattern("/*.json"),
mvc.pattern("/*.map"), mvc.pattern("/*.css")).permitAll()
            .requestMatchers(mvc.pattern("/*.ico"), mvc.pattern("/*.png"), mvc.pattern("/*.svg"),
mvc.pattern("/*.webapp")).permitAll()
            .requestMatchers(mvc.pattern("/app/**")).permitAll()
            .requestMatchers(mvc.pattern("/i18n/**")).permitAll()
            .requestMatchers(mvc.pattern("/content/**")).permitAll()
            .requestMatchers(mvc.pattern("/swagger-ui/**")).permitAll()
            .requestMatchers(mvc.pattern(HttpMethod.POST, "/api/authenticate")).permitAll()
            .requestMatchers(mvc.pattern(HttpMethod.GET, "/api/authenticate")).permitAll()
            .requestMatchers(mvc.pattern("/api/register")).permitAll()
            .requestMatchers(mvc.pattern("/api/activate")).permitAll()
            .requestMatchers(mvc.pattern("/api/account/reset-password/init")).permitAll()
            .requestMatchers(mvc.pattern("/api/account/reset-password/finish")).permitAll()
            .requestMatchers(mvc.pattern("/api/admin/**")).hasAuthority(AuthoritiesConstants.ADMIN)
            .requestMatchers(mvc.pattern("/api/**")).authenticated()
            .requestMatchers(mvc.pattern("/v3/api-docs/**")).hasAuthority(AuthoritiesConstants.ADMIN)
            .requestMatchers(mvc.pattern("/management/health")).permitAll()
            .requestMatchers(mvc.pattern("/management/health/**")).permitAll()
            .requestMatchers(mvc.pattern("/management/info")).permitAll()
            .requestMatchers(mvc.pattern("/management/prometheus")).permitAll()
            .requestMatchers(mvc.pattern("/management/**")).hasAuthority(AuthoritiesConstants.ADMIN)
        )
        .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .exceptionHandling(exceptions -> exceptions
            .authenticationEntryPoint(new BearerTokenAuthenticationEntryPoint())
            .accessDeniedHandler(new BearerTokenAccessDeniedHandler())
        )
        .oauth2ResourceServer(oauth2 -> oauth2.jwt(withDefaults()));
    return http.build();
}
```

Code Snippet 3 - Endpoint Authorization

6.2 TLS and HTTP/2

```yaml
server:
  ssl:
    key-store: classpath:config/tls/keystore.p12
    key-store-password: password
    key-store-type: PKCS12
    key-alias: selfsigned
    ciphers: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_AES_
    enabled-protocols: TLSv1.2
  http2:
    enabled: true
```

Code Snippet 4 - TLS and HTTP/2 Spring Profile

As said in the documentation design, it is the best practice to use TLS, the protocol to encrypt data between http requests, making it secure.

There is a Spring Profile that is used to start the application with TLS and HTTP/2 configurated, as seen in Code Snippet 4, with an SSL key certificated registered.

This SSL certificate is needed to be installed to the application when running locally.

This profile is used to deploy the application.

6.3 Authentication and Authorization

Our application uses JWT tokens with OAuth2 for authentication and authorization, configured in the ***SecurityJwtConfiguration*** class. This setup ensures secure token-based authentication.

6.3.1 Example JWT Token


```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImV4cCI6MTcxNjE1Mzc2MCwiYXV0aCI6IlJPTEVfQURNSU4iLCJpYXQiOjE3MTYwNjczNjB9.qdt758QLnnIPeMlmCZyxMWTlKe8WbUw6Mi3dY1NIRErWoSzCKHs9ZpQ-8o3_rBD43LV53imloWpiKQlsBJJdfw
```


6.4 Password Encryption

For password encryption, we use the ***org.springframework.security.crypto.password.PasswordEncoder*** from Spring Security. In the ***UserService.java*** class, passwords are encrypted using the AES-256 algorithm before being stored in the database. Decryption is also handled by the same library to ensure secure access.

```
String encryptedPassword = passwordEncoder.encode(password);
```

Code Snippet 5 - Password Encryption

6.5 UUID for Identifiers

All IDs in our application utilize the `UUID` class, which ensures unique and secure identification for entities.

This UUID class in java uses by default the version 4, which is more secure than the v1, v2 and v3.

6.6 Application Roles

As said before, there are 3 application roles that were configurated, such as Driver, Manager and Customer.

Their permissions were already specified in the Design Documentation, although, for development purposes, there is one more role, Admin, that basically allows to administer the application as a whole and see certain data.

For this sprint, only the roles were configurated and some permissions because the development focus was not as high as the configuration of the system and the pipeline implementation. In sprint 2, it is intended to focus on this topic.

There is by default the administrator of application inside the bootstrap data. This administrator is using not predictable credentials such as referenced before as a good security practice.

6.7 Database Roles

The database, as specified before, also contains the three roles mentioned, one for the administrator for development purposes and configuration, another for the read user and another for the write/read user.

Although this is configurated, it is not being used yet in the system, the application is configurated to use the read/write user. This will take some work to be done and configurate each endpoint to use the right user from the database, so it is intended to be done in sprint 2.

Same as the application users, the database users and roles are not using predictable credentials.

7 Logging

In this sprint, logging using another component and physical machine is not directly implemented.

The only type of logging developed is inside the application server that prints all the actions made to the server machine. This is intended for development purposes but it only registers runtime the requests made, and actions performed.

For implementing the logging designed before, it is needed to setup another machine in another location, for security reasons, and associate all the actions with the users which performed them. The message can be the same as that is already being printed out.

8 GitHub Actions

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository or deploy merged pull requests to production.

8.1.1 Configuration Created

In order to create workflows, a folder named ***.github*** was created with a folder inside called **workflows**. The workflows folder is where you need to add the workflow files (yml files) to be recognized by GitHub Actions.

Two workflows were created, one that does most of the work and an extra workflow file specifically for OWASP ZAP scan (the reason for it will be explained later).

All private information (tokens, passwords, webhooks) were hidden from the workflow files by using GitHub repository Secrets.

8.1.2 Application CI

This workflow is the workflow that does most of the heavy lifting.

```yaml
name: Application CI
on:
  [push, pull_request,workflow_dispatch]
jobs:
  pipeline:
    name: TruckMotion pipeline
    runs-on: ubuntu-latest
    if: "!contains(github.event.head_commit.message, '[ci skip]') && !contains(github.event.head_commit.message, '[skip ci]') &&
!contains(github.event.pull_request.title, '[skip ci]') && !contains(github.event.pull_request.title, '[ci skip]')"
    timeout-minutes: 40
    env:
      NODE_VERSION: 20.12.2
      SPRING_OUTPUT_ANSI_ENABLED: DETECT
      SPRING_JPA_SHOW_SQL: false
      JHI_DISABLE_WEBPACK_LOGS: true
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20.12.2
      - uses: actions/setup-java@v4
        with:
          distribution: 'temurin'
          java-version: 17
      - name: Install Node.js packages
        working-directory: ./Deliverables/Sprint1/Sprint1_TruckMotion
        run: npm install
      - name: Run backend test
        working-directory: ./Deliverables/Sprint1/Sprint1_TruckMotion
        run: |
          chmod +x mvnw
          npm run ci:backend:test
      - name: Run frontend test
        working-directory: ./Deliverables/Sprint1/Sprint1_TruckMotion
        run: npm run ci:frontend:test
      - name: Cache SonarCloud packages
        uses: actions/cache@v3
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
```

Code Snippet 6 - Application CI (1)

```
    - name: Cache Maven packages
      uses: actions/cache@v3
      with:
        path: ~/.m2
        key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
        restore-keys: ${{ runner.os }}-m2
    - name: Build and analyze
      working-directory: ./Deliverables/Sprint1/Sprint1_TruckMotion
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  # Needed to get PR information, if any
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
      run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -DskipITs -Dsonar.projectKey=RafaelFaisca32_desofs2024_M1C_3
    - name: Depcheck
      uses: dependency-check/Dependency-Check_Action@main
      id: Depcheck
      env:
        JAVA_HOME: /opt/jdk
      with:
        project: 'truckmotion'
        path: './Deliverables/Sprint1/Sprint1_TruckMotion'
        format: 'HTML'
        args: >
          --enableRetired
    - name: Upload Test results
      uses: actions/upload-artifact@master
      with:
        name: Depcheck report
        path: ${{github.workspace}}/reports
    - name: Login to Docker Hub
      uses: docker/login-action@v3
      with:
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKERHUB_TOKEN }}
    - name: Build Docker Image
      working-directory: ./Deliverables/Sprint1/Sprint1_TruckMotion
      run:
        npm run java:docker
    - name: Push Docker Image
      run: |
        docker push ${{ secrets.DOCKER_HUB_REPO }}
    - name: Webhook for render deployment
      uses: distributhor/workflow-webhook@v3
      with:
        webhook_url: ${{ secrets.RENDER_WEBHOOK }}
```

Code Snippet 7 - Application CI (2)

It will now be explained what every step does and some extra configuration.

Secrets

- secrets.GITHUB_TOKEN: The GITHUB_TOKEN secret is a GitHub App installation access token.
- secrets.SONAR_TOKEN: This token is used in order to connect SonarCloud with the Application.
- secrets.DOCKER_USERNAME: Username of the Docker Account that is used in order to login to the Docker Hub.
- secrets.DOCKERHUB_TOKEN: Token of the Docker Account that is used in order to login to the Docker Hub.
- secrets.DOCKER_HUB_REPO: Name of the Docker Hub Repository.
- secrets.RENDER_WEBHOOK: Link in order to trigger a new redeployment in Render.

On property with the pipeline "if"

The On property is the setting that allows to select when the workflow is supposed to be executed. In this case, the workflow will be executed when there is a push or a pull request. The workflow_dispatch setting is used to be able to manually run the job for testing purposes.

The if setting is used to skip the workflow execution if needed when doing a push. So, if you add **[ci skip]** or **[skip ci]** the pipeline won't be executed.

Runs-on and working-directory
The Runs-on property is used to select which Virtual Machine will be used by GitHub Actions to execute the workflow.

The working-directory tag is used to move to a specific folder inside the repository (similar behavior to cd on terminal).

actions/checkout@v4
This action checks-out your repository under $GITHUB_WORKSPACE, so your workflow can access it.

actions/setup-node@v4
This action is used to download a specific node version and add to the PATH of the VM.

actions/setup-java@v4
This action is used to download a specific Java version and add to the PATH of the VM.

Install Node.js packages
In this step, the npm packages will be installed.

Run backend test
In this step, the created backend unit tests will be run.

Run frontend test
In this step, the created frontend tests will be run.

Cache SonarCloud packages and Cache Maven packages
Both steps are used to cache the packages after the first run to save some time on the pipeline execution and speed up the process.

Build and analyze
This step is used to build and analyze the application and it will execute SonarQube on top of the app and send the data to the SonarCloud server.

Depcheck
This step is used to run a DependencyCheck over the Application to identify possible vulnerabilities related to app dependencies.

Upload Test results
This step is used to upload the test results made by DependencyCheck as an artifact of the GitHub Actions.

Login to Docker Hub
This step is used to login to Docker Hub to push the image of the app for deployment.

Build Docker Image
This step builds the docker Image of the Application.

Jib was used to create the docker image without the need of a Docker daemon.

Push Docker Image
This step is used to push the image to the Docker Hub.

Webhook for render deployment
This step is used to force a redeployment of the app with a webhook before finishing the workflow.

8.1.3 ZAP Pipeline

```
name: ZAP Pipeline
on:
    workflow_dispatch:
jobs:
  pipeline:
    name: TruckMotion ZAP pipeline post-deployment
    runs-on: ubuntu-latest
    if: ${{ github.event_name == 'workflow_dispatch' }}
    timeout-minutes: 40
    env:
      JHI_DISABLE_WEBPACK_LOGS: true
    steps:
      - name: ZAP Scan
        uses: zaproxy/action-full-scan@v0.10.0
        with:
          token: ${{ secrets.GITHUB_TOKEN }}
          target: 'https://truckmotion.onrender.com/'
```

Code Snippet 8 - ZAP Pipeline

This pipeline will basically do an analysis of our newly deployed version of the application. It will also add the report of the results as an artifact to GitHub Actions.

The reason why this is separate and can only be run manually (workflow_dispatch) is because the Render servers are extremely slow to start, so if it was on the other workflow file it wouldn't do the analysis properly.

8.2 Deployment process

To deploy the application, a Docker image using Jib was created and was pushed to a repository on Docker Hub.

By using Render we can create Web Services that can deploy from existing images, in our case, the Docker Image that was pushed to a specific repository.

So, after that, all we need to do is to force a redeployment of the Application every time there are changes pushed to Master, which is done by the step **"Webhook for render deployment".**

9 Security Reports - Github Artifacts

9.1 SAST

This report analyzes the source code or binaries of an application without executing it to find security vulnerabilities.

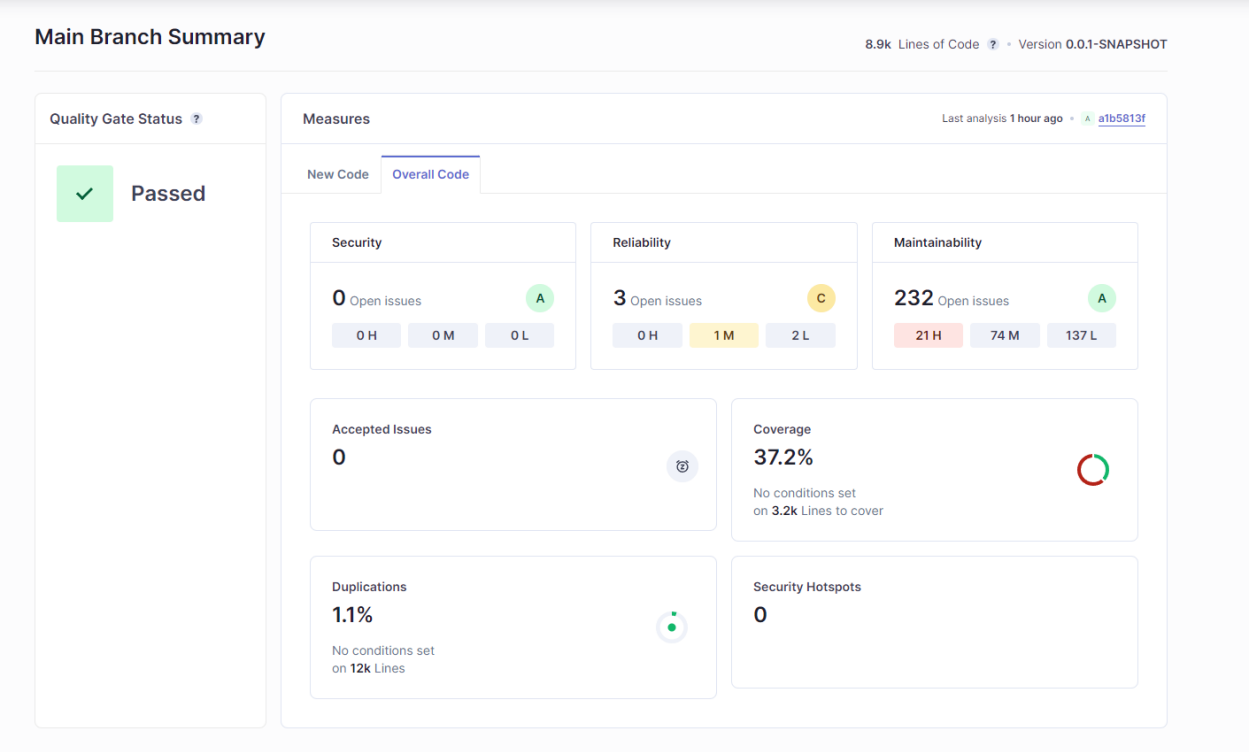

Figure 25 - SonarCloud Report

We used Sonarcloud to get this report as you can see in Figure 25. You can access it here.

Security and Maintainability have great results even though reliability can be improved. The coverage is low, so in the next sprint we will try to improve it as well.

9.2 SCA

Focus on identifying and managing risks associated with open source and third-party components used in software.

Provide details on known vulnerabilities in these components, license compliance issues, and potential operational risks.

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
app.js		pkg:javascript/angularjs@1.8.2	HIGH	7		3
async:1.5.2	cpe:2.3:a:async_project:async:1.5.2:*:*:*:*:*:*:*	pkg:npm/async@1.5.2	HIGH	1	Highest	7
axios.js		pkg:javascript/axios@0.21.4	MEDIUM	2		3
axios.min.js		pkg:javascript/axios@0.21.4	MEDIUM	2		3
axios:0.21.4	cpe:2.3:a:axios:axios:0.21.4:*:*:*:*:*:*:*	pkg:npm/axios@0.21.4	MEDIUM	2	Highest	8
braces:3.0.2	cpe:2.3:a:braces_project:braces:3.0.2:*:*:*:*:*:*:*	pkg:npm/braces@3.0.2	HIGH	1	Highest	8
ejs:3.1.10	cpe:2.3:a:ejs:ejs:3.1.10:*:*:*:*:*:*:*	pkg:npm/ejs@3.1.10	CRITICAL	1	Highest	8
json5:0.5.1	cpe:2.3:a:json5_json5:0.5.1:*:*:*:*:*:*:*	pkg:npm/json5@0.5.1	HIGH	1	Highest	7
loader-utils:0.2.17	cpe:2.3:a:webpack.js:loader-utils:0.2.17:*:*:*:*:*:*:*	pkg:npm/loader-utils@0.2.17	CRITICAL	3	Highest	6
micromatch:4.0.5		pkg:npm/micromatch@4.0.5	HIGH	1		8
truck-motion-0.0.1-SNAPSHOT.jar: commons-compress-1.21.jar	cpe:2.3:a:apache:commons_compress:1.21:*:*:*:*:*:*:*	pkg:maven/org.apache.commons/commons-compress@1.21	MEDIUM	2	Highest	104
truck-motion-0.0.1-SNAPSHOT.jar: jackson-databind-2.15.4.jar	cpe:2.3:a:fasterxml:jackson-databind:2.15.4:*:*:*:*:*:*:* cpe:2.3:a:fasterxml:jackson-modules-java8:2.15.4:*:*:*:*:*:*:*	pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.15.4	MEDIUM	1	Highest	40
truck-motion-0.0.1-SNAPSHOT.jar: nimbus-jose-jwt-9.24.4.jar	cpe:2.3:a:connect2id:nimbus_jose\+jwt:9.24.4:*:*:*:*:*:*:*	pkg:maven/com.nimbusds/nimbus-jose-jwt@9.24.4	HIGH	1	Highest	52
truck-motion-0.0.1-SNAPSHOT.jar: undertow-core-2.3.12.Final.jar	cpe:2.3:a:redhat:undertow:2.3.12:*:*:*:*:*:*:*	pkg:maven/io.undertow/undertow-core@2.3.12.Final	HIGH	2	Highest	41
truck-motion-0.0.1-SNAPSHOT.jar: undertow-servlet-2.3.12.Final.jar	cpe:2.3:a:redhat:undertow:2.3.12:*:*:*:*:*:*:*	pkg:maven/io.undertow/undertow-servlet@2.3.12.Final	HIGH	1	Highest	40
truck-motion-0.0.1-SNAPSHOT.jar: xnio-api-3.8.8.Final.jar	cpe:2.3:a:redhat:xnio:3.8.8:*:*:*:*:*:*:*	pkg:maven/org.jboss.xnio/xnio-api@3.8.8.Final	HIGH	1	Highest	42

Figure 26 - Dependecy Check Report

We use dependency check for this, and you can find the report (Figure 26) in the **Deliverables\Sprint1\reports\sca** folder.

We have these vulnerabilities to audit or fix in the next sprint.

A study will be made to see if the application is really affected by these vulnerabilities. If it is, we will upgrade them to a version that is clear of vulnerabilities or find a new library that does the same.

9.3 DAST

Assess an application in its running state to find vulnerabilities that could be exploited in real-world attacks. We used ZAP to get this information.

ZAP Version: 2.15.0

ZAP is supported by the Crash Override Open Source Fellowship

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	5
Low	0
Informational	4
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
CSP: Wildcard Directive	Medium	2
CSP: script-src unsafe-eval	Medium	2
CSP: script-src unsafe-inline	Medium	2
CSP: style-src unsafe-inline	Medium	2
Proxy Disclosure	Medium	25
Information Disclosure - Suspicious Comments	Informational	1
Modern Web Application	Informational	2
Non-Storable Content	Informational	10
Storable and Cacheable Content	Informational	2

Figure 27 - ZAP Report

You can find the report (Figure 27) in this folder **Deliverables\Sprint1\reports\dast\zap_scan**.

Even though we implemented and added content security policy there are some medium vulnerabilities on the application that we will tackle in the next sprint.

10 ASVS v4 Checklist

The checklist in this sprint indicates what was implemented and the applications serve, rather than the optimal and real objectives that were shown in the Design Documentation. In the Design Documentation, there was a real evaluation of the objective ASVS checklist.

In this sprint, the checklist shows what was implemented and achieved in this sprint with some comments towards the topic and the source code references to justify the topic result.

The excel and everything correlated to it can be found here.

11 References

[1] I. Object Management Group®, “UML,” Unified Modeling Language, 2023. [Online]. Available: https://www.uml.org/.