

INSA LYON
COMPUTER SCIENCES DEPARTMENT

PLD AGILE : Optimod'Lyon

Technical Documentation

Authors: Hazim ASRI
Nihal BOUTADGHART
Jassir HABBA
Ana MARTIN
Junior NOUKAM
Simon PERRET

Teacher:
Mme. LAFOREST

Contents

1	Class and Package Diagrams	1
1.1	Class Diagram	1
1.2	Package Diagram	2
2	System Architecture	3
3	CI/CD Documentation	3
3.1	Workflow Triggers	3
3.2	Jobs in the Workflow	4
3.3	Dependency Graph Update	4
3.4	Benefits of the Pipeline	4
4	Frontend Tests	4
4.1	Example Test Case	4

Introduction

This technical documentation aims to provide a comprehensive and detailed overview of the **[Optimod’Lyon]** application. It is intended for developers and technical personnel who seek to understand the technical decisions, architecture, and core functionalities of the system, as well as the steps required to configure, use, and maintain the code.

The project follows an **object-oriented development** approach, adhering to the principles of modularity, reusability, and maintainability. We adopted the **Agile Scrum methodology** to ensure iterative and incremental deliveries, allowing for quick adjustments to the needs and challenges encountered. The application is built on a **Model-View-Controller (MVC)** architecture and leverages modern technologies such as **Java**, **React**, and the **Google Maps API**.

This document is structured as follows:

- A description of the architecture and the technical choices made;
- The main components of the system and their responsibilities;
- The processes for installation, configuration, and deployment;
- The tests conducted to validate the application’s functionality;
- A section dedicated to best practices for contributors.

With this documentation, we aim to ensure a seamless transition for future teams tasked with maintaining or extending the application, while facilitating onboarding for new contributors.

1 Class and Package Diagrams

1.1 Class Diagram

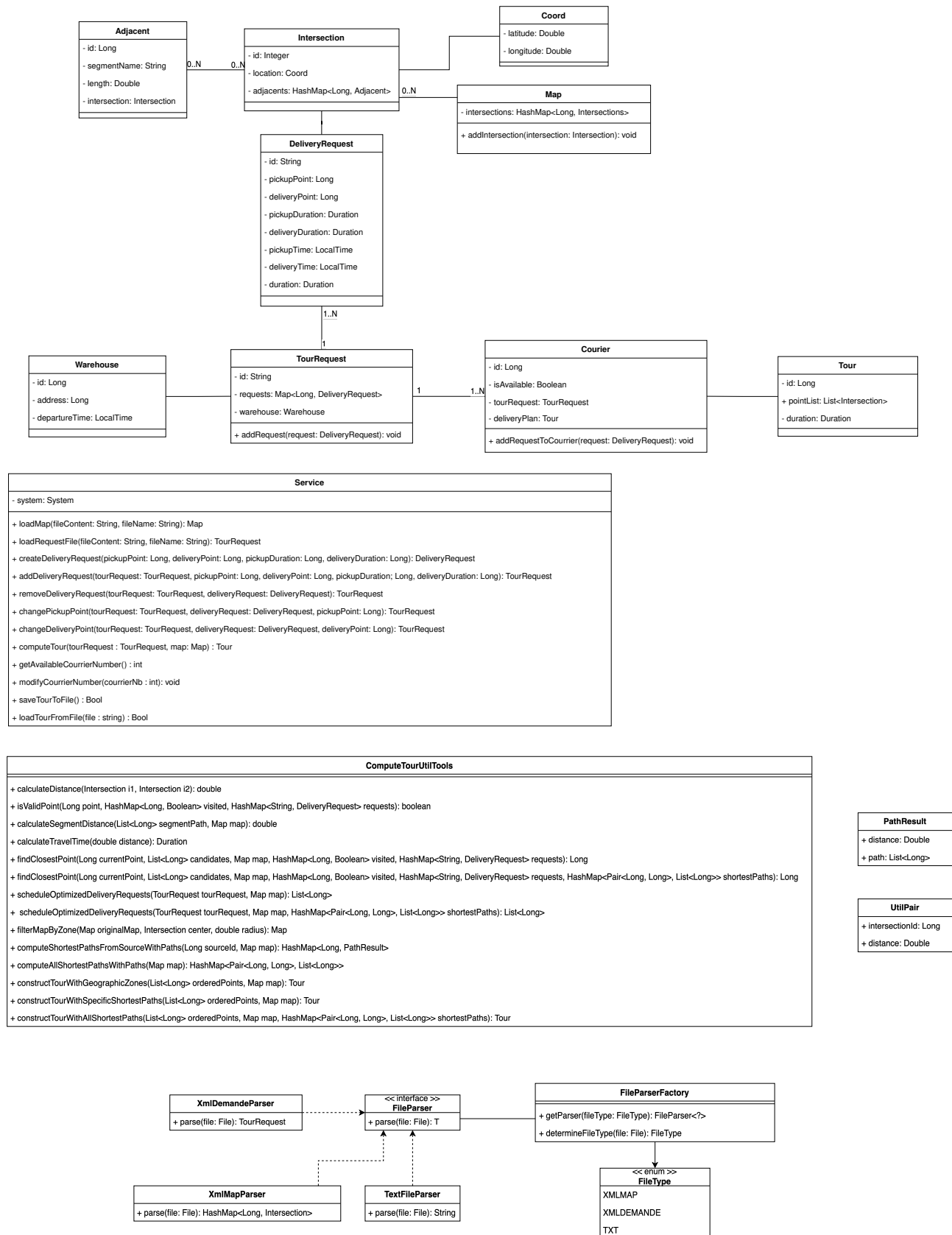


Figure 1: Class Diagram

The class diagram provides an overview of the structure of the system, including key classes, their attributes, and relationships. It reflects the modularity and reusability principles applied during the design phase.

1.2 Package Diagram

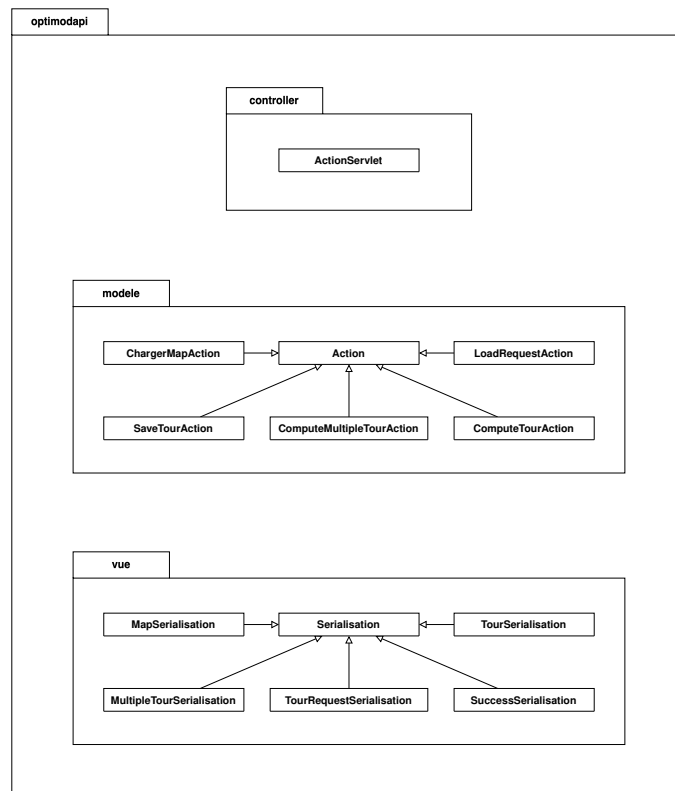
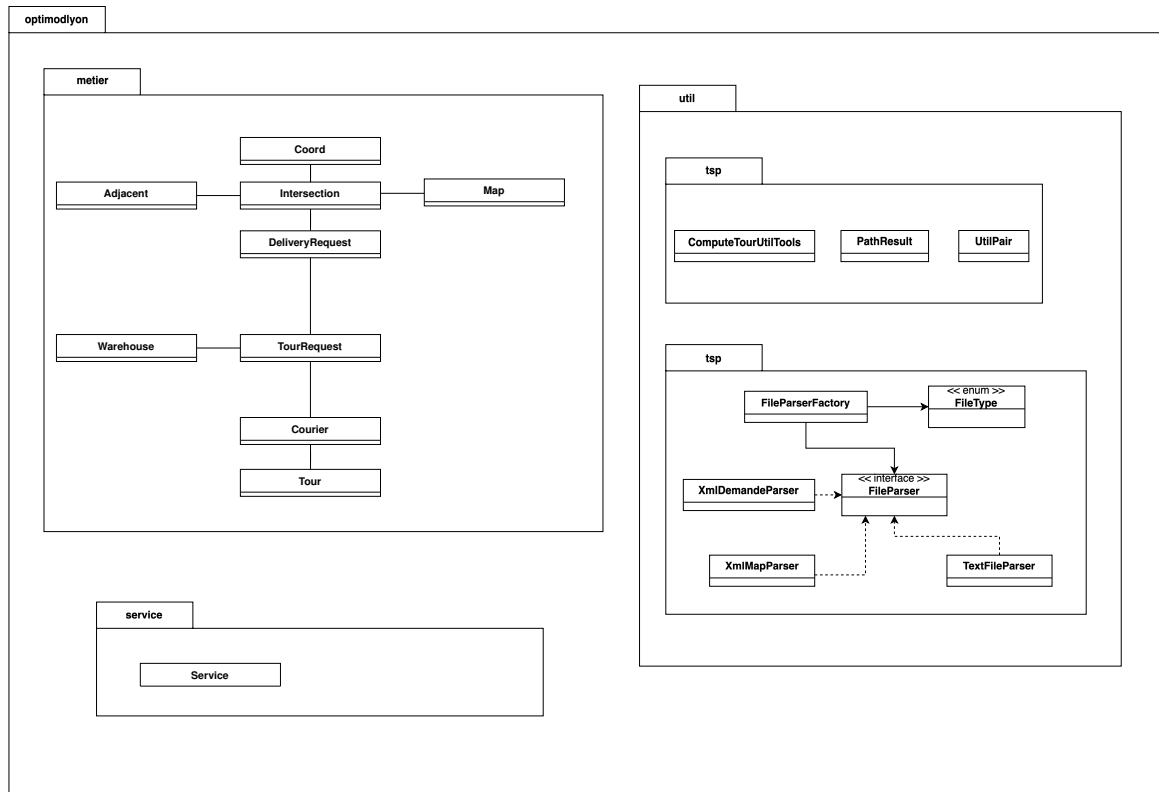


Figure 2: Package Diagram

The package diagram illustrates the organization of the codebase into logical groupings. It showcases the separation of concerns, where each package handles a specific set of responsibilities, ensuring scalability and maintainability.

2 System Architecture

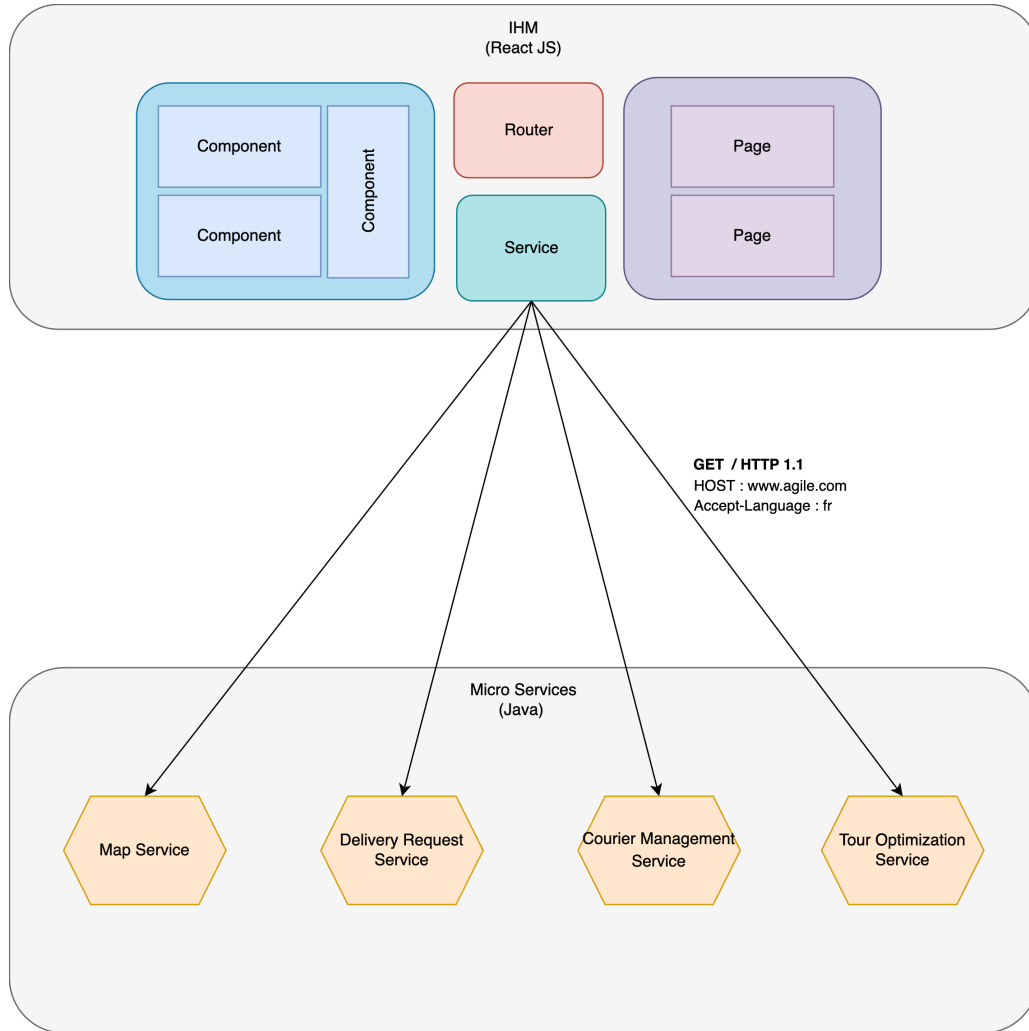


Figure 3: Package Diagram

The system is designed using a **Model-View-Controller (MVC)** architecture. The backend, developed in **Java**, serves as the core of the application, providing APIs to communicate with the frontend. The frontend, developed in **React**, interacts with the Google Maps API to display the network and manage user interactions.

3 CI/CD Documentation

This pipeline configuration file defines a Continuous Integration and Continuous Deployment (CI/CD) workflow for a Java project using Maven as the build system. It automates essential tasks such as building, testing, optimizing execution time, and ensuring code quality. By automating these processes, the pipeline contributes to maintaining a robust, efficient, and secure delivery workflow. The pipeline is triggered by changes made to the develop branch, ensuring that all updates are validated continuously before being merged into production.

3.1 Workflow Triggers

Triggers determine the conditions under which the workflow is executed. For this pipeline:

- The workflow is automatically triggered on every push to the develop branch.

- This setup ensures continuous validation of new changes, promoting consistent code quality and reducing integration issues.

3.2 Jobs in the Workflow

The workflow is composed of multiple jobs that automate different stages of the CI/CD process.

- **Repository Checkout:** The flow uses the `actions/checkout@v4` action to clone the current repository into the workspace. This step ensures that all project files are available for subsequent tasks, such as building and testing.
- **Java Configuration:** The environment is set up using `actions/setup-java@v4` to install JDK 11. Maven is configured with dependency caching enabled. This optimization allows the workflow to reuse previously downloaded dependencies, significantly reducing execution time and improving overall efficiency.
- **Build and Test:** Maven is used to build the project and execute all unit and integration tests. Any build failures or test errors immediately halt the pipeline, ensuring that only valid changes progress further.

3.3 Dependency Graph Update

The pipeline generates a dependency graph that maps all project dependencies and their relationships.

- This graph helps identify outdated or vulnerable dependencies more accurately.
- By maintaining an up-to-date dependency graph, the pipeline improves the project's security posture and ensures compatibility with external libraries.

3.4 Benefits of the Pipeline

The CI/CD pipeline provides several advantages:

- **Improved Code Quality:** Automated tests ensure that new code adheres to the project's standards and does not introduce regressions.
- **Efficiency:** Dependency caching and automated processes reduce manual intervention and save time during development cycles.
- **Enhanced Security:** The dependency graph provides better visibility into potential vulnerabilities.
- **Scalability:** The pipeline supports a consistent and repeatable process, making it suitable for teams of all sizes.

This CI/CD pipeline is a cornerstone of maintaining high-quality and efficient continuous delivery in modern software development practices. Next Steps

4 Frontend Tests

The frontend was tested using **Jest** and **React Testing Library**. The tests focused on the following aspects:

- Component rendering and behavior under various conditions.
- Integration with the backend API, ensuring proper data exchange.
- Edge cases such as invalid inputs and network failures.

4.1 Example Test Case

```
test('Displays map with markers for delivery points', async () => {
  render(<MapComponent />);
  expect(screen.getByTestId('map')).toBeInTheDocument();
  await waitFor(() => expect(screen.getAllByTestId('marker').length).toBeGreaterThan(0));
});
```