Janeeb
Solutions

SYSTEM REQUIREMENTS SPECIFICATION

Extended Planning Instrument for Unpredictable Spaces and Environments

University Of Pretoria
janeeb.solutions@gmail.com

Introduction

Vision and Objectives

Given the potential associated with South Africa's logistics juggernaut, our project aims to revolutionize the efficiency of goods transportation. With an annual movement of 1.5 billion tonnes of goods, contributing 10% to the nation's GDP, and employing a million-strong workforce, the logistics sector is a cornerstone of the South African economy. Our mission is to design a system that transforms inefficiency into a symphony of precision, ensuring that every truck is a marvel of intelligent packing and optimal space management. This project isn't just an upgrade, we see it as a revolution in efficiency.

Business Need

The logistics sector in South Africa, despite its massive scale, suffers from significant inefficiencies in space utilization within transportation vehicles. These inefficiencies lead to increased operational costs, reduced profit margins, and environmental impacts due to suboptimal usage of fuel and increased trips. There is a clear business need for an advanced system that can dynamically adjust the placement of goods in real-time, optimizing the available space within logistics trucks. By addressing this need, our project will contribute to cost savings, improved profitability, and a reduction in environmental impact.

Project Scope

The scope of this project encompasses the development of a comprehensive system designed to enhance the efficiency of goods transportation within logistics trucks. Key components of the project include:

- **Dynamic Packing Algorithm:** An intelligent algorithm capable of real-time adjustment of goods placement, prioritizing efficient space utilization to maximize the number of goods transported in a single trip.
- **Learning Algorithm:** A robust learning algorithm that learns from historic gene pools and fitness evaluation to continuously enhance the packing algorithm's performance.
- **Manager Interface:** A user-friendly interface for logistics managers to input specific constraints, preferences, and priorities, facilitating customized packing solutions.
- **Real-time Dashboard:** A dashboard that displays real-time information about the packing progress and the algorithm's performance, providing valuable insights and facilitating decision-making.

- **Simulation Renderer:** A visual simulation tool to showcase the packing process, enhancing understanding and showcasing the algorithm's effectiveness.

By addressing these components, our project aims to deliver a scalable and high-performance system capable of transforming the logistics sector in South Africa.

User Stories

1. Logistics Managers

1.1 As a logistics manager, I need to input specific constraints and preferences for the packing algorithm to ensure that the loading process aligns with operational requirements.
1.2 As a logistics manager, I need to monitor real-time information about the packing progress and algorithm performance through the dashboard to make informed decisions and adjustments.

2. Truck Drivers

2.1 As a truck driver, I need to understand the sequence for unloading goods at various destinations to ensure a smooth and efficient delivery process.

3. Warehouse Staff

3.1 As a warehouse staff member, I need to prepare goods for loading according to the appropriate delivery to ensure the algorithm can optimize the correct packages to be packed.
3.2 As a warehouse staff member, I need to pack the shipment according to the system's real-time information to ensure accurate and efficient loading.

4. IT and System Administrators

4.1 As an IT and system administrator, I need to perform regular maintenance, updates, and troubleshooting to ensure the system operates smoothly and efficiently.
4.2 As an IT and system administrator, I need to manage user access and permissions to ensure that only authorized personnel can input data and access sensitive information.

Functional Requirements

Requirements

F1. Dynamic Packing Algorithm:

1.1 Real-Time Adjustment:

- 1.1.1 The algorithm dynamically adjusts the placement of goods in real-time.
- 1.1.2 The algorithm prioritizes efficient space utilization to maximize the number of goods transported in a single trip.

1.2 Optimization:

- 1.2.1 The algorithm optimizes the available space within the logistics truck to ensure the smallest amount of wasted space.
- 1.2.2 The algorithms optimizes the packing solution and considers the unpacking Solution based on where the current delivery is to.

1.3 Constraints and Preferences:

- 1.3.1 The algorithm takes into account specific constraints and preferences input by the logistics manager.

F2. Learning Algorithm:

2.1 Evolutionary Process:

- 2.1.1 The algorithm generates an initial population of packing solutions based on predefined criteria.
- 2.1.2 Each solution represents a possible packing configuration, encoded as a gene within the population.
- 2.1.3 Solutions undergo selection, crossover, and mutation to explore optimal packing configurations over successive generations.

2.2 Adaptation and Optimization**:**

- 2.2.1 The algorithm evaluates the fitness of each packing solution based on space utilization, load stability, and weight constraints.
- 2.2.2 The genetic algorithm adapts over time, evolving the packing solutions to improve efficiency and reduce wasted space.

2.3 Continuous Improvement**:**

- 2.3.1 The algorithm refines its solutions with each iteration, updating the gene pool based on high-performing packing configurations from previous loads.
- 2.3.2 The algorithm retains knowledge of successful packing configurations to inform future decisions.

F3. Manager Interface:

3.1 Input Constraints and Preferences:

- 3.1.1 Logistics managers can input specific constraints for the packing algorithm.
- 3.1.2 Logistics managers can set preferences and priorities for packing.

3.2 User-Friendly Interface:

- 3.2.1 The interface is intuitive and easy to use for logistics managers.

3.3 Real-Time Monitoring:

- 3.3.1 Managers can monitor packing progress in real-time.

F4. Real-Time Dashboard:

4.1 Data Display:

- 4.1.1 The dashboard displays real-time information about the packing process.

4.2 Algorithm Performance:

- 4.2.1 The dashboard provides insights into the algorithm’s performance.

4.3 User Interaction:

- 4.3.1 Users can interact with the dashboard to view detailed information.

Subsystems

1. Authentication/Authorization Subsystem

- 1.1 User Registration
- 1.2 User Log-In
- 1.3 Password Reset:

2. Dynamic Packing Algorithm Subsystem

- 2.1 Real-Time Adjustment
- 2.2 Optimization
- 2.3 Constraints and Preferences

3. Learning Algorithm Subsystem:

- 3.1 Evolutionary Process
- 3.2 Adaptation and Optimization
- 3.3 Continuous Improvement

4. Manager Dashboard Interface Subsystem

- 4.1 Input Constraints and Preferences
- 4.2 Algorithm performance and monitoring.
- 4.3 Real-Time data display.

5. Packer Instructions Subsystem

5.1 Real-Time packing solution display.

5.2 Package QR scanner.

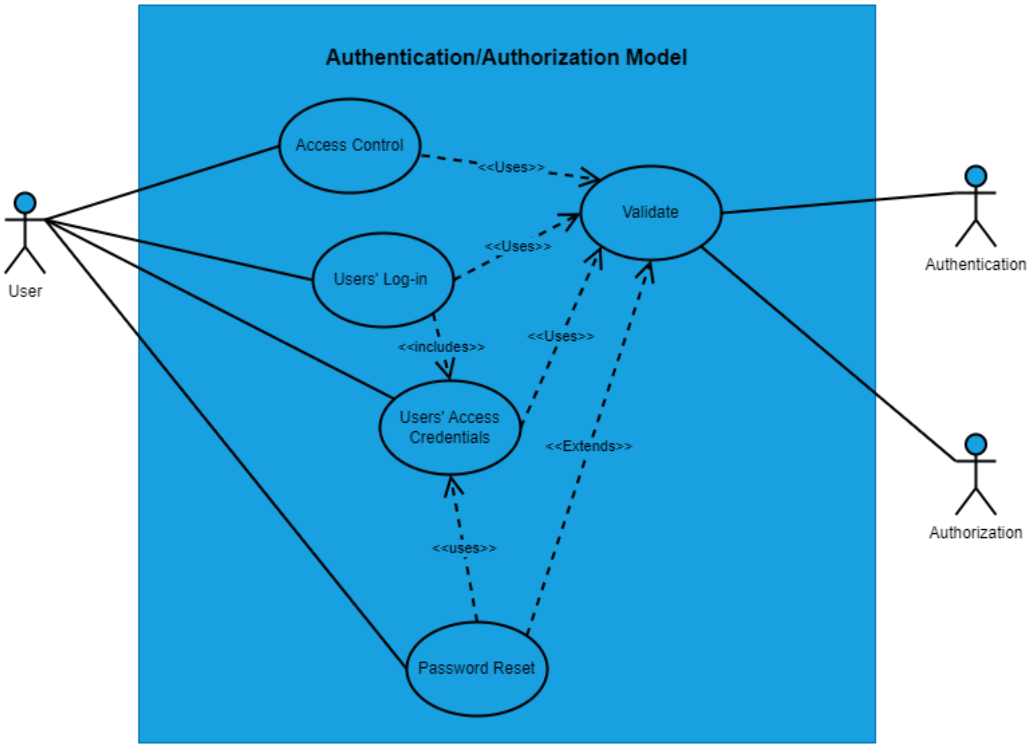
6. Delivery Instructions Subsystem

6.1. Delivery destination view.

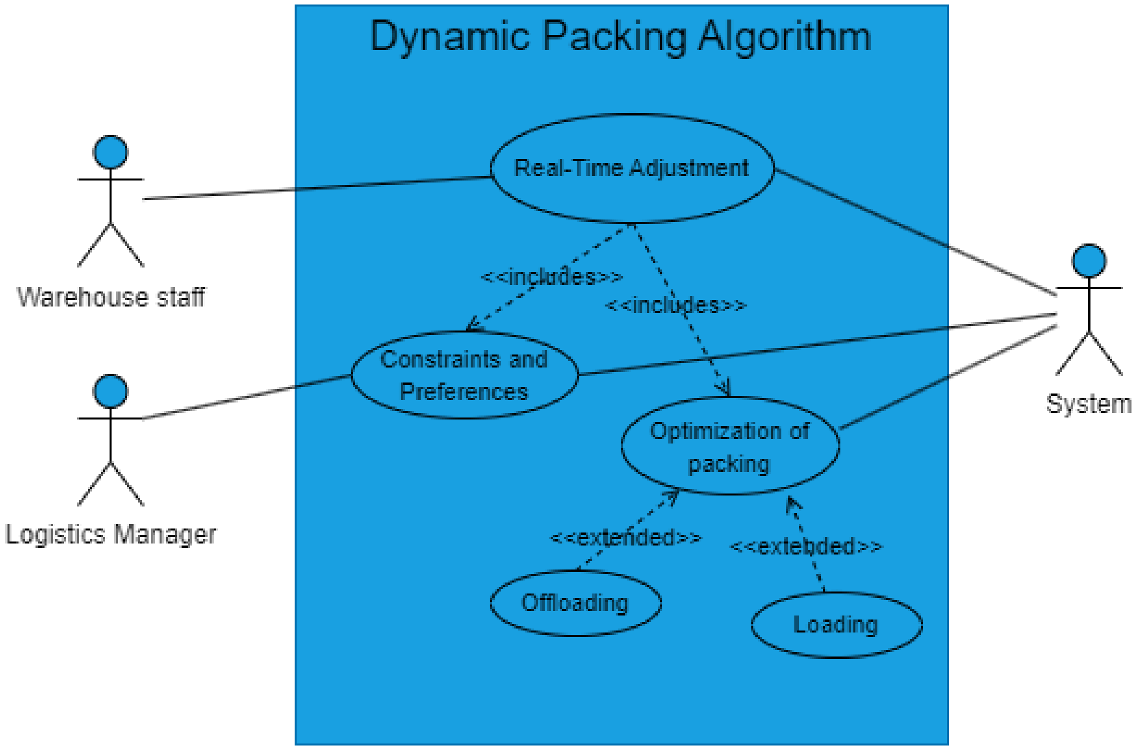
6.2. Offloading instructions.

Use Case Diagrams

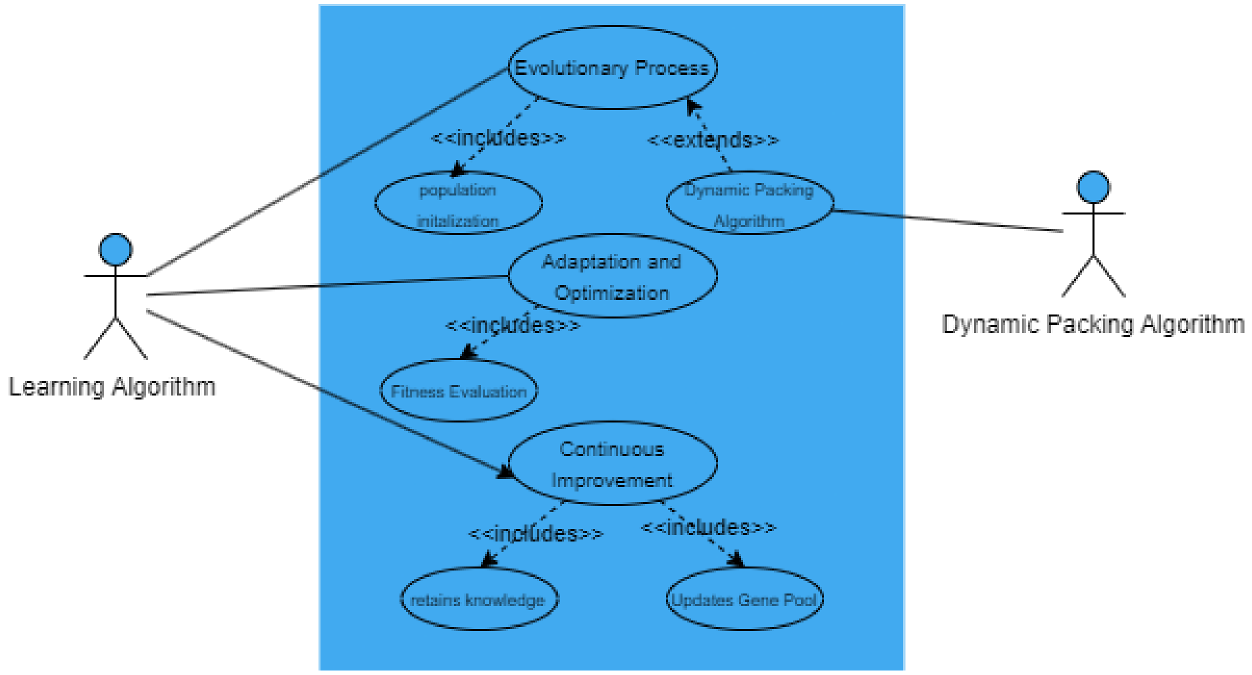
1. Authentication/Authorization



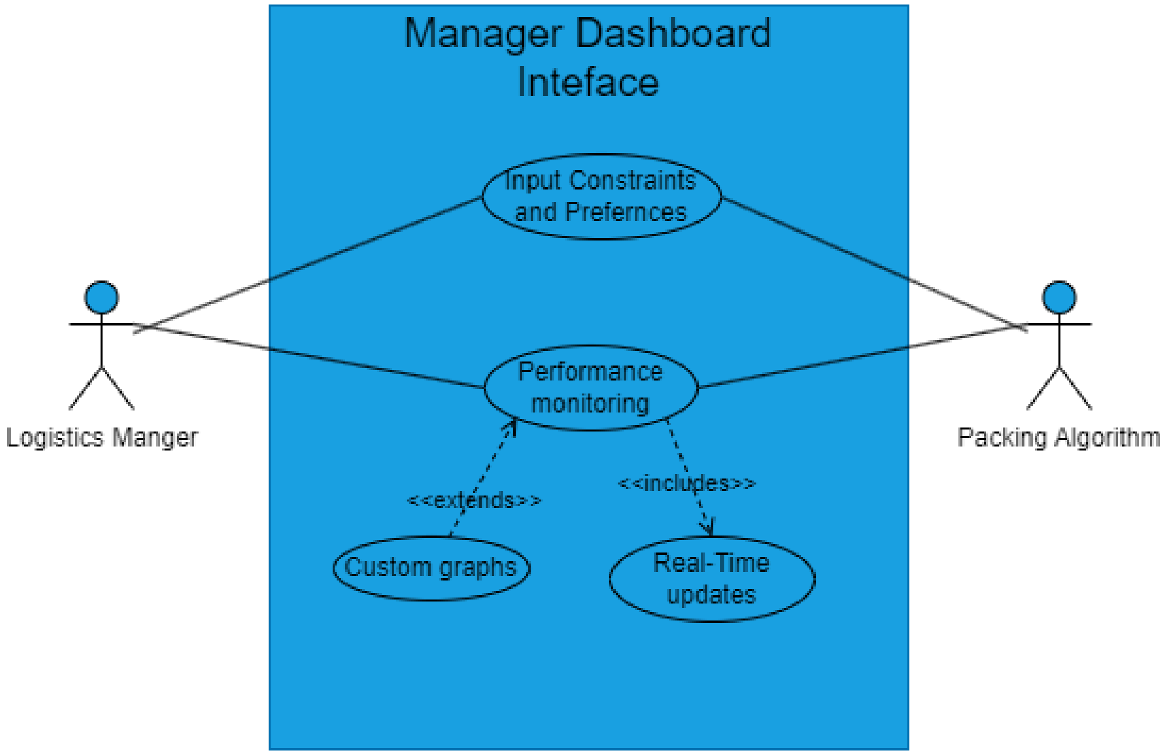
2. Dynamic Packing Algorithm



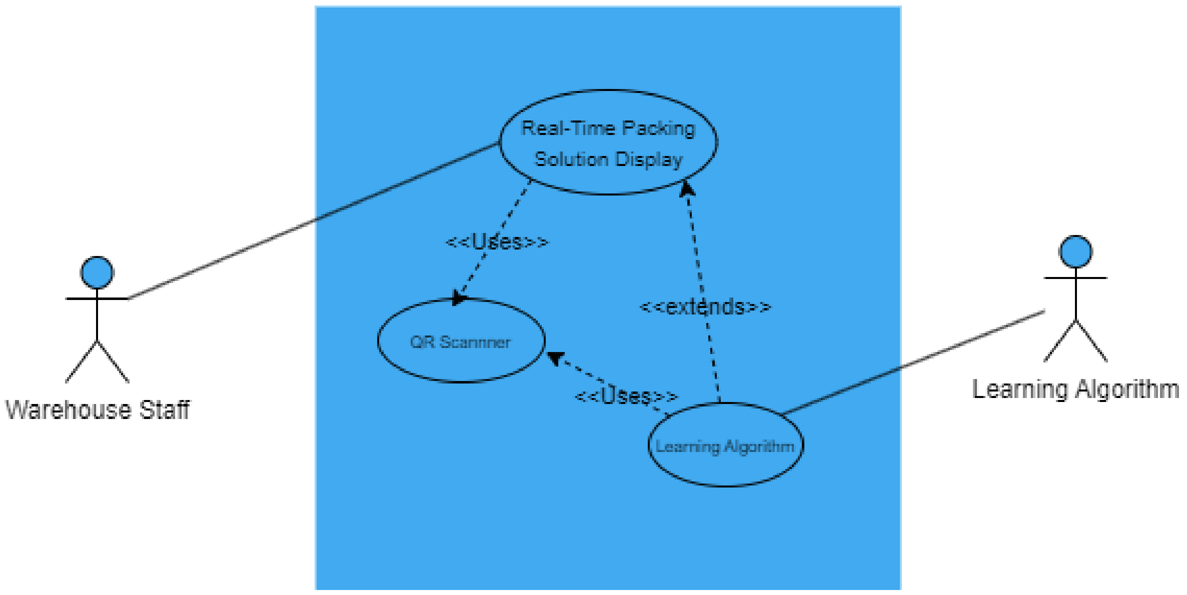
3. Learning Algorithm



4. Manager Dashboard Interface

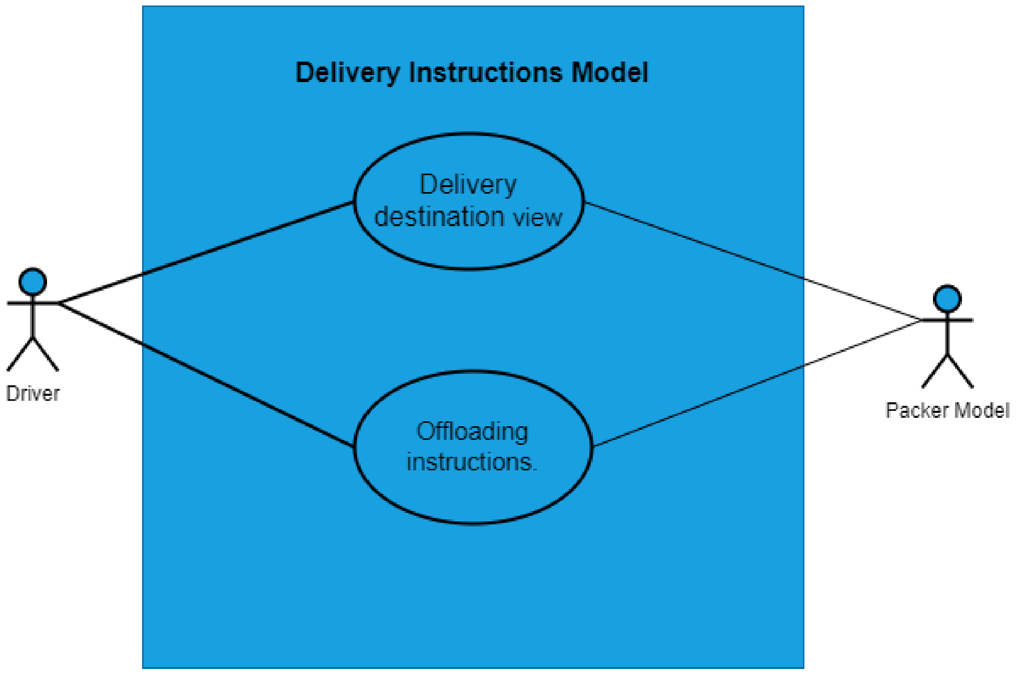


5. Packer Instructions

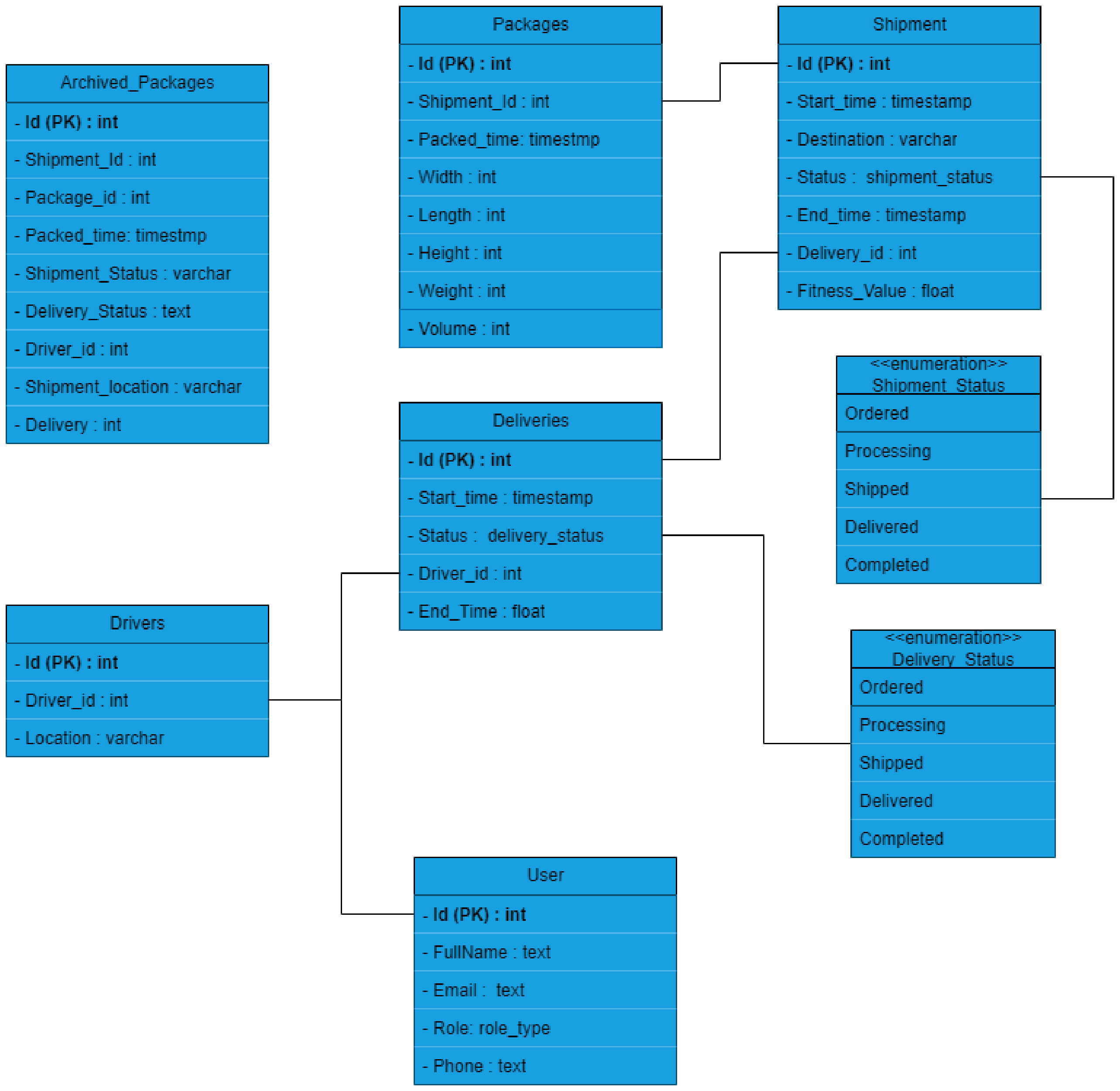


2

6. Delivery Instructions



Class Diagram



Non-Functional Requirements

NF1. Performance:

1.1. The application shall have fast response times for user interactions.
1.2. The application's database operations, implemented with Supabase, should have efficient query execution times to ensure quick retrieval and storage of data.
1.3. The system should return the optimal packing route within 1-3 minutes of the provided configuration/input parameters.
1.4. The real time 3D render should reflect optimizations in real time without any delay or buffering.

NF2. Scalability:

2.1. The backend infrastructure, particularly the database layer hosted on Supabase, should be capable of handling increasing loads as the user base expands.
2.2. The system should be scalable in the sense the algorithm could potentially be used for packing cargo trains, or shipping containers

NF3. Security:

3.1. All user authentication and authorization processes, including registration, login, and password reset, must follow the best practices for safe transfer and storage of sensitive user information.
3.2. Data stored in the Supabase database must be encrypted to protect user privacy and comply with relevant data protection regulations.
3.3. All sensitive information is POPI ACT compliant ensuring data integrity and user safety.

NF4. Reliability:

4.1. Error handling mechanisms shall be implemented to handle erroneous input and provide helpful feedback to users in case of unexpected errors.
4.2. Continuous integration and deployment pipelines (CI/CD) shall be set up using GitHub Actions to ensure reliable software releases.

NF 5. Usability:

5.1. The application shall have an intuitive and user-friendly interface to ensure ease of use for all user types, including logistics managers, truck drivers, and warehouse staff.
5.2. The interface should be accessible and provide a consistent user experience across various devices and screen sizes.
5.3. The system shall provide clear instructions and feedback to users, guiding them through the process and informing them of any errors or required actions.

5.4 The system shall provide real-time updates to users, ensuring they are always informed of the current status and any changes in the packing process.

Architectural Patterns

To optimize the system outlined in the project, we have integrated three architectural patterns: the three-tier layered architecture, service-oriented architecture (SOA), and an API gateway architecture. These patterns provide a foundational framework for the Extended Planning Instrument for Unpredictable Spaces and Environments (EPIUSE) service, ensuring optimal performance and scalability.

At the core of the system is the manager dashboard, which has full access to all system functionalities. To support this component, we have implemented a three-tier layered architecture, consisting of the security layer, application layer, and backend layer.

- **Security Layer**: Responsible for authentication, authorization, and enforcing access control to protect sensitive data and operations.
- **Application Layer**: This is the interface between the user and the system, it ensures efficient communication between the user interface and the backend of our system at its entirety.
- **Backend Layer**: Handles data management, storage, and interaction with the database, ensuring the system's infrastructure is scalable and reliable. Further it handles the data that is generated by the algorithm we use to generate the packing solution.

This layered approach promotes modularity, maintainability, and enhanced security across the system. Given that the system offers a variety of services, we have adopted a service-oriented architecture (SOA) to ensure modularity and scalability. In this architecture:

- **Service Consumers** include roles such as Drivers, Warehouse Staff, and IT/System Interfaces, each interacting with the system to fulfill specific operational tasks.
- **Service Providers** comprise components like the Learning Algorithm, Dynamic Packing Algorithm, and Packer Instructions, which deliver the essential functionality and processing required by the consumers.

The **Service Bus** within the SOA is represented by the API, acting as the gateway between service providers and consumers. The API ensures seamless communication, facilitating the flow of data and service requests between the components.

This design creates a cohesive system where the API enables efficient interaction between users and services. The modular approach of SOA allows for flexibility, enabling services to be added, updated, or removed independently, ensuring the system remains adaptable and responsive to evolving requirements.

In the **API Gateway Architecture**, the API gateway serves as the central access point for the system, connecting the **Client Application** to the **Microservices**, which represent the various algorithms used to solve specific problems within the Extended Planning Instrument for Unpredictable Spaces and Environment.

1. **Client Application**: This is the interface through which users (e.g., warehouse staff, drivers, or system administrators) interact with the system, making service requests or retrieving data.
2. **Microservices**: The system's microservices consist of the **Learning Algorithm**, **Dynamic Packing Algorithm**, and **Packer Instructions**. Each microservice is responsible for handling a specific task within the system, such as solving the shortest delivery path, optimizing packing processes, or generating operational data for the manager.
3. **API Gateway**:
   - The API gateway acts as a gatekeeper, ensuring that only authorized requests pass through and reach the microservices.
   - It provides a **unified entry point** for the client application, enabling seamless interaction with the system's different algorithms.
   - The gateway also performs several critical functions, such as:
     - **Security Enforcement**: Verifying user identities and permissions before granting access to sensitive services.
     - **Request Routing**: Directing requests to the appropriate microservices based on the client's needs.
     - **Protocol Translation**: Converting client requests into formats that the microservices can process.
     - **Load Balancing**: Distributing incoming traffic efficiently across multiple instances of the microservices to ensure optimal system performance.
     - **Monitoring**: Tracking system activity, ensuring that services are running smoothly, and identifying potential bottlenecks or failures.

By employing the API Gateway Architecture, the system simplifies the interaction between the client and microservices, improves security and performance, and provides a scalable solution for managing complex operations.

In conclusion, the integration of the three-tier layered architecture, service-oriented architecture (SOA), and API gateway architecture forms a robust and scalable foundation for the Extended Planning Instrument for Unpredictable Spaces and Environments (EPIUSE) service. The three-tier architecture ensures modularity, maintainability, and security through its distinct security, application, and backend layers, each managing critical aspects such as data protection and the systems infrastructure reliability. SOA allows the system to offer diverse services by connecting service consumers like drivers, warehouse staff and IT interface with essential components such as the learning algorithm and dynamic packing algorithm, while the service bus (API) facilitates seamless communication between these components.

The API gateway architecture further enhances the system by serving as the central access point, connecting the client application to various microservices and acting as a gatekeeper to secure, efficient request handling. This architecture supports critical functions like security enforcement, request routing, and load balancing, providing a unified interface that simplifies interactions between users and services while promoting performance and scalability.

Together, these architectural patterns create a cohesive, flexible, and high-performing system, designed to adapt to evolving requirements and optimize complex operations within the EPIUSE service.

Design Patterns

Facade

The Manager Dashboard Interface will act as the facade, providing a simplified and unified interface for the logistics manager to interact with the underlying subsystems like the Dynamic Packing Algorithm Service,Learning Algorithm, Packer Instructions Service, and Delivery Instructions Service.

The facade will expose methods for the manager to input constraints, preferences, and priorities for the packing algorithm. It will communicate with the Dynamic Packing Algorithm Service, translating the user input and retrieving the generated packing solution for presentation.

The facade will also provide an interface for the manager to review and provide feedback on the packing solutions provided from the learning algorithm. It will act as a way to monitor the algorithm's performance.

Furthermore, the facade will allow the manager to access packing instructions for warehouse staff and delivery instructions for truck drivers by communicating with the respective services and presenting the information in a user-friendly format.

By acting as the facade, the Manager Dashboard Interface will handle the complexities of communicating with the subsystems, promoting better code organization, readability, and separation of concerns. It also facilitates easier maintenance and extensibility, as changes to the subsystems won't affect the facade's interface as long as the contracts remain unchanged.

Singleton

The Singleton Pattern ensures that a class has only one instance and provides a global point of access to that instance. This is particularly useful for managing shared resources and maintaining consistent state across the system.

A singleton database connection can efficiently manage database connections, optimizing resource utilization and maintaining connection limits for improved performance and stability.

Security services managing authentication and authorization also benefit from the Singleton Pattern by providing a consistent and universally accessible point of control, enhancing overall security.

Constraints

The primary constraint for our logistics optimization system is the requirement to use specific datasets provided in the linked documents. These datasets will serve as the foundation for developing and training the learning Algorithm, as well as for testing and validating the packing algorithms. There are no additional technical or operational constraints at this stage.

Data Set 1

Data Set 2

Service Contracts

Register Request and Response

Register Request:
- email (string): The user's email address.
- password (string): The user's chosen password.
- firstName (string): The user's first name.
- lastName (string): The user's last name.
- country (countryEnum): The user's country, represented as an enumeration value.

This information is sent when a user wants to create a new account.

Register Response:
- status (string): The status of the registration request, indicating success or failure.
- message (string): A message providing additional information about the status.
- timestamp (number): The time when the response was generated.

This information is returned after a user attempts to register, indicating the outcome of their registration attempt.

Login Request and Response

Login Request:

- email (string): The user's email address.
- password (string): The user's password.

This information is sent when a user wants to log into their account.

Login Response:
- status (string): The status of the login request, indicating success or failure.
- timestamp (number): The time when the response was generated.
- userWithToken (IUserResponse): An object containing user information along with an authentication token.

This information is returned after a user attempts to log in, indicating the outcome of their login attempt and providing the necessary authentication details if successful.

Technology Requirements

Backend:
- Languages & Frameworks: Python
- Database: PostgreSQL on Supabase
- APIs: RESTful APIs

Frontend:
- Languages & Frameworks: JavaScript, Vue, PrimeVue, Tailwind CSS
- Visualization: Three.js.

DevOps:
- Version Control: Git, GitHub
- CI/CD: GitHub Actions
- Containerization: Docker
- Cloud Platforms: Netlify

Security:
- Auth/Access Control: OAuth 2.0
- Encryption: SSL/TLS

Collaboration:
- Communication: Discord, WhatsApp, Google Drive
- Project Management: GitHub Project Board
- Documentation: Google Docs.