Janeeb Solutions

ARCHITECTURAL SPECIFICATION

Extended Planning Instrument for Unpredictable Spaces and Environments

Architectural design strategy

Background

For our logistics optimization system, we aimed to develop a comprehensive solution that addresses the unique needs of managing and optimizing the placement of goods in logistics trucks. Our system leverages dynamic algorithms and machine learning models to enhance efficiency and space management. This application will assist logistics managers in predicting office capacities and managing desk allocations to prevent overbooking. Given the diverse and demanding requirements of such a system, we have focused on ensuring high performance, scalability, security, reliability, and usability.

Architectural Decisions

The architectural decisions have been based on the quality requirements of the system. We have used the quality requirements (listed in our System Requirements Specification: SRS) to identify our quality requirements which were used to identify our architectural patterns.

Architectural strategies

For the logistics optimization system described in the project, a Microservices Architecture is a suitable architectural pattern. This pattern breaks down the application into smaller, independently deployable services that communicate over a network. Each service is responsible for a specific piece of functionality and can be developed, deployed, and scaled independently.

1. Authentication/Authorization Service
    - Manages user registration, login, and password reset functionalities to ensure secure access.
2. Dynamic Packing Algorithm Service
    - Adjusts and optimizes the placement of goods in real-time, considering constraints and preferences.
3. Machine Learning Model Service
    - Learns from data and continuously improves the packing algorithm.
4. Manager Dashboard Interface Service
    - Provides an interface for inputting constraints, monitoring algorithm performance, and displaying real-time data.
5. Packer Instructions Service
    - Displays real-time packing solutions and integrates package QR scanning for verification.
6. Delivery Instructions Service
    - Organizes delivery destinations and provides offloading instructions to ensure efficient delivery processes.

In the microservices architecture, services communicate with each other through well-defined APIs. This approach ensures loose coupling, allowing each service to be developed, deployed, and maintained independently. For real-time data updates and event-driven communication, supabase’s built-in real-time functionality is used to facilitate efficient and reliable data exchange between services.

The microservices architecture offers several significant benefits. Enhanced flexibility allows teams to develop, deploy, and scale services independently, making it easier to manage and evolve the system. Improved maintainability is achieved through smaller, modular services that are easier to understand, test, and maintain. Greater resilience is another advantage, as the failure of one service does not necessarily impact others, thus improving the system's overall robustness. Additionally, technology agility allows different services to leverage the most suitable technologies for their specific requirements, optimizing performance and development efficiency.

Architectural quality 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 2-5 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.

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:

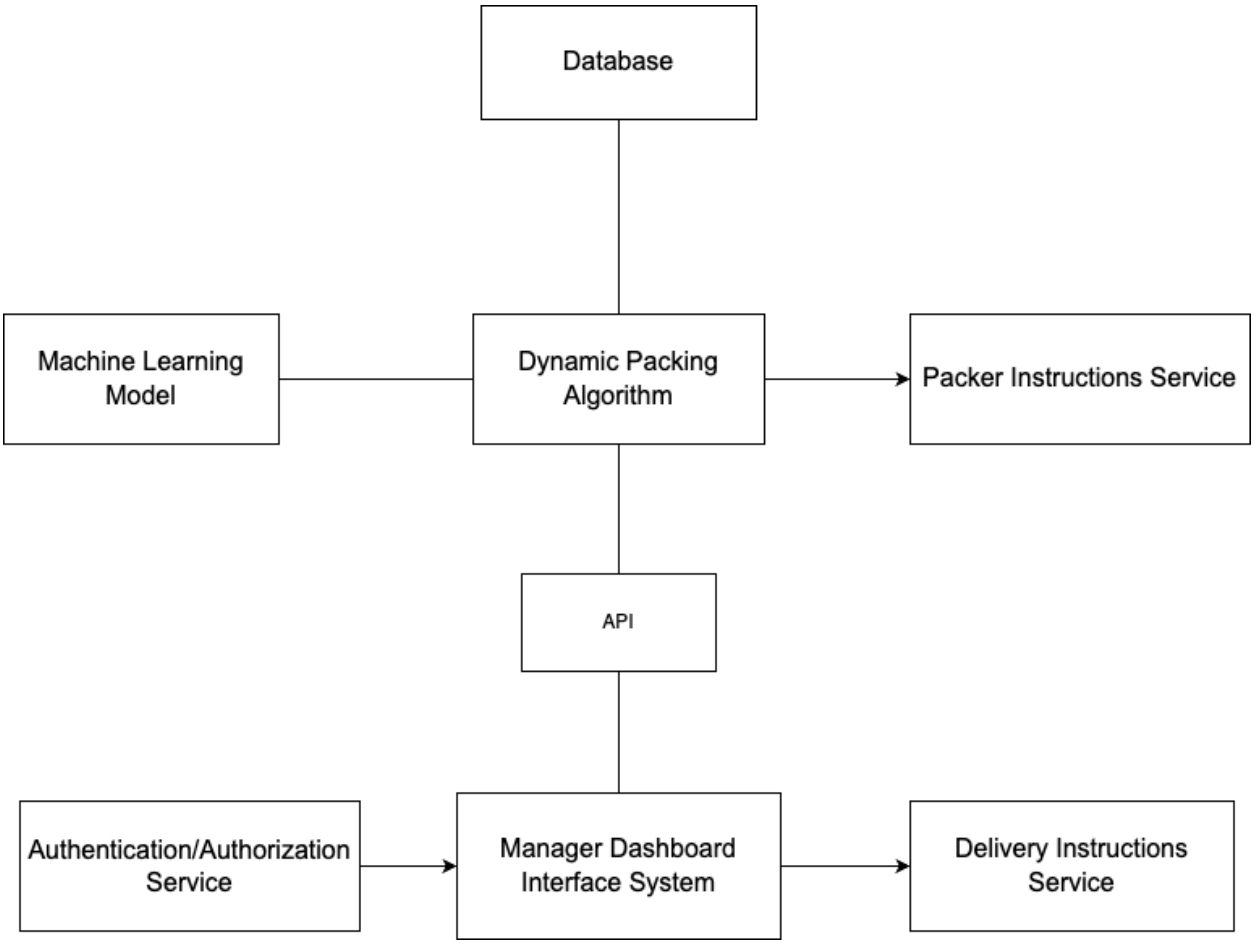
5.1. The application shall have an intuitive and user-friendly interface to ensure ease of use for all user types, including logistics managers, truck drivers, and warehouse staff.

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 design & pattern



Design Patterns Used

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, Machine Learning Model Service, 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. It will collect this feedback and pass it to the Machine Learning Model Service to improve the algorithm over time. Additionally, it will retrieve and display insights or recommendations from the Machine Learning Model Service.

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.

Architectural constraints

Data Constraints:

The system must utilize specific datasets provided in the linked documents for developing and training the machine learning models. These datasets form the foundation for the algorithm's learning and optimization processes. All data must be handled in compliance with relevant data protection regulations. Sensitive information must be encrypted, and access must be restricted to authorized personnel only.

Technology choices

Backend:

- Languages & Frameworks: Python
- Database: PostgreSQL on Supabase
- Machine Learning: TensorFlow
- 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: Vercel

Security:

- Auth/Access Control: OAuth 2.0, JWT
- Encryption: SSL/TLS

Collaboration:

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