Janeeb Solutions

CODING STANDARDS

Extended Planning Instrument for Unpredictable Spaces and Environments

Background

South Africa's logistics sector, crucial to the economy, moves 1.5 billion tonnes of goods annually, contributing 10% to the GDP and employing over a million people. Despite its importance, the sector faces inefficiencies in space utilization, leading to increased costs and environmental impacts. Our project aims to revolutionize this by developing an advanced system that optimizes truck space through a dynamic packing algorithm, a machine learning model, a user-friendly manager interface, a real-time dashboard, and a visual simulation tool. This system will enhance efficiency, profitability, and sustainability in the logistics sector.

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 machine learning models, 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

Technologies

Frontend

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

Backend:

Languages & Frameworks: Python
Database: PostgreSQL on Supabase
Machine Learning: TensorFlow
APIs: RESTful APIs

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.

GitHub Information

Git Structure 🌳

Our repository follows a **mono repo** structure to keep all project components organized in a single repository, facilitating easy management and collaboration.

Git Organization and Management 🗂️

Our main branching strategy and organization method is to use **GitFlow**. We maintain a clear and organized Git structure with branches for development, testing, and production to ensure smooth workflow and code quality.

GitFlow Branching Strategy 🔀

Our branching strategy includes:
Main Branch: This is our primary branch used for deployment. It contains the stable version of our application and is updated periodically from the Development branch.
Development Branch: This branch serves as the central hub for all development activities. It is branched off from the Main branch and is used to integrate features and components before they are deemed stable enough for release.
Feature Branches: For individual features or fixes.
Release Branches: For preparing releases.

Feature Branching 🌿

When developing new features or components, team members should adhere to the following process:

1. **Create a Feature Branch:** Branch off from the Development branch using the naming convention dev/<feature-name> for general features or dev/<parent-feature>/<sub-feature> for more specific branches.
2. **Develop the Feature:** Work on the feature within this branch, committing changes regularly.
3. **Open a Pull Request:** Once the feature is complete, open a pull request to merge the changes back into the Development branch. This allows for code review and automated checks using GitHub Actions.
4. **Merge into Development:** After review and successful checks, the feature branch is merged into the Development branch.

Merging to Main 🔗

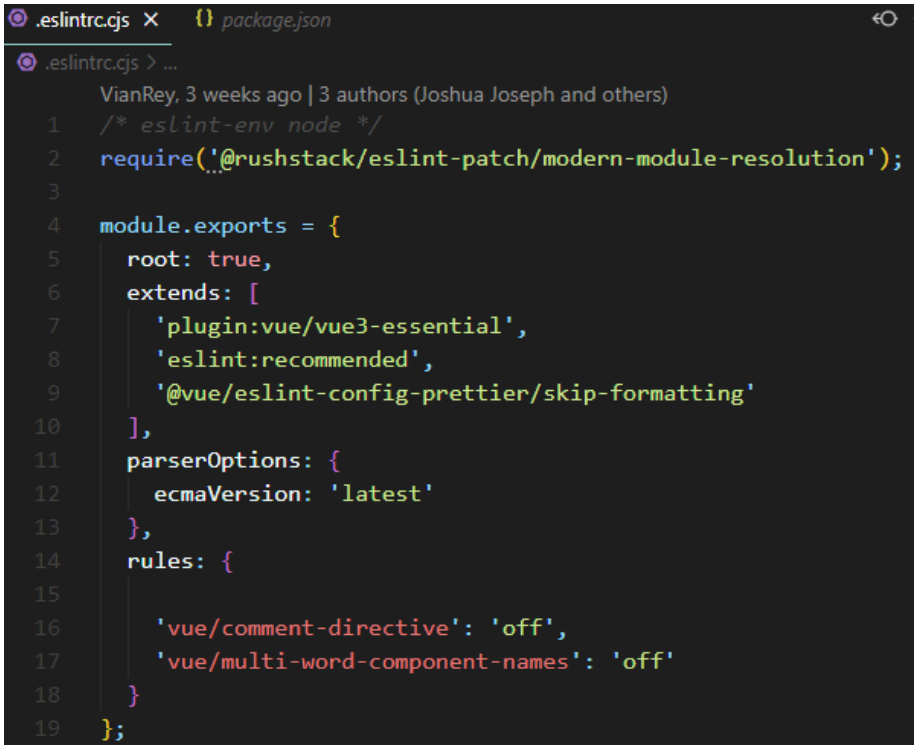
Once the Development branch is stable and has undergone thorough testing, it is merged back into the Main branch for release. This ensures that the Main branch always contains a stable version of the application ready for deployment. A merge to main cannot occur without passing a review from the Team leader and one other team member. This ensures that no merges to main are accidental, keeping the integrity of the main branch.

Linter

Overview of ESLint

ESLint is a powerful tool for identifying and reporting on patterns in JavaScript. It helps ensure code quality and consistency by enforcing coding standards and best practices. When configured properly, ESLint can significantly improve the development process by catching errors early and promoting a uniform codebase

EsLint configuration



Benefits of Using This ESLint Configuration for Vue Projects

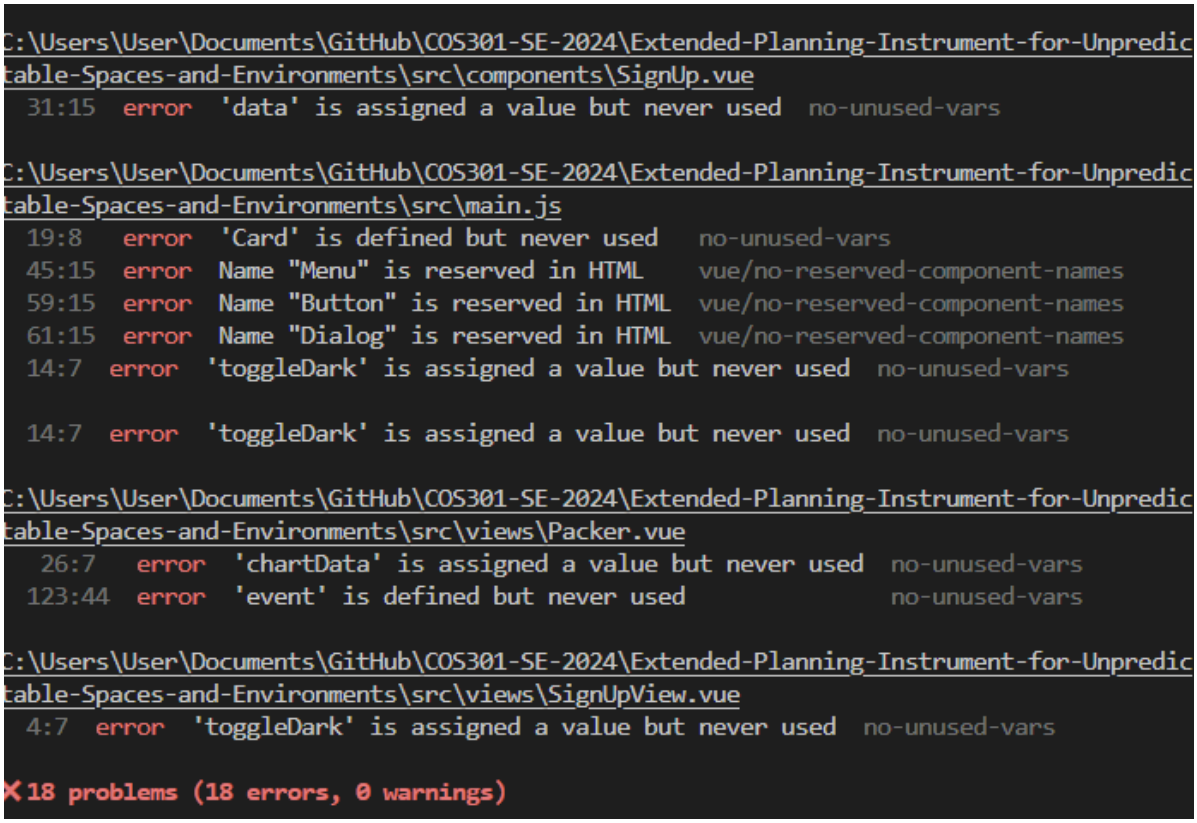
1. **Comprehensive Vue Support:**
    - The plugin:vue/vue3-essential plugin ensures that all Vue-specific syntax and best practices are enforced, catching errors that are unique to Vue development and promoting efficient, maintainable code.
2. **General JavaScript Best Practices:**
    - eslint:recommended provides a strong foundation of JavaScript best practices, catching common mistakes and enforcing good coding standards across the entire codebase.
3. **Seamless Integration with Prettier:**
    - By including @vue/eslint-config-prettier/skip-formatting, this configuration avoids conflicts between ESLint and Prettier, allowing Prettier to manage code formatting while ESLint focuses on logic and syntax errors. This results in a cleaner, more readable codebase with consistent styling.
4. **Flexibility and Customization:**
    - Custom rules like turning off vue/comment-directive and vue/multi-word-component-names provide flexibility to adapt the linting

rules to the project's specific needs and preferences, making it easier to integrate with existing coding standards and workflows.

5. **Modern JavaScript Features:**
    - Supporting the latest ECMAScript version allows developers to utilize modern JavaScript features, improving code efficiency and readability.

Running the Linter

The run the linter simply type **npm run lint in the terminal of the program.** This will run the linter and clearly identity each section of the code that doesn't meet the coding linting standards.i.e.



By using this ESLint configuration, the project can maintain high code quality, improve consistency, and streamline the development process, all of which are crucial for successful and scalable software application