README - Emissions Dashboard Application

# Project Overview: Emissions Dashboard Application (Angular v19)

This application is built using Angular v19, leveraging the latest improvements in performance, signals, and standalone components. The primary focus of this app is to visualize and interact with emission-related data via a dynamic dashboard UI.

# State Management with NgRx Signal Store

To handle the application's state effectively, we use NgRx Signal Store, a lightweight and reactive solution for state management introduced in newer versions of NgRx. This store is responsible for managing:

* - Emission data fetched from remote APIs
* - Consolidated totals, including:
* - Total emissions for a specific year
* - Breakdown of emissions by category (e.g., Scope 1, Scope 2, Scope 3)

## Benefits of Signal Store:

* - Reactive computation: With Angular Signals, derived values are automatically recalculated when base state changes.
* - Improved maintainability: The state logic is encapsulated and declarative.
* - Isolation: The emission state is isolated from UI logic.

# Styling with Tailwind CSS

The UI is styled using Tailwind CSS, which offers utility-first classes that enable rapid development and consistent design.

## Why Tailwind?

* - Scalability: Maintain a consistent design language across components.
* - Extensibility: Tailwind’s configuration allows for quick overrides and design tokens.
* - Productivity: Styling changes can be made directly in the template using utility classes.

# Component Architecture

The dashboard is composed of self-contained, standalone components, each managing its own logic and presentation. Angular’s new @defer and signal-driven APIs are used to improve the initial load performance and UI responsiveness.

# Summary of Technology Stack

|  |  |
| --- | --- |
| Technology | Purpose |
| Angular v19 | Core framework |
| NgRx Signal Store | Reactive state management for dashboard data |
| Tailwind CSS | Utility-first styling framework |
| RxJS | Stream handling for API and async operations |
| @defer syntax | Optimized rendering of async components |

# Key Benefits

* - High performance and responsive UI through reactive signals and deferred rendering
* - Predictable state with clean separation of business logic from UI
* - Modern and maintainable styling with Tailwind
* - Easy unit testing of logic using signal-store and selector-driven testing