Teil 3 Architektur

Entwicklung eines Energiemanagement-Dashboards in TypeScript unter Verwendung von Entwurfsmustern

Software Engineering AI

SoSe 2025

Matias Acuna

947551

a) **Planung**

1. **Überblick über die gewählte Architektur**: Die allgemeine Architektur, die ich für dieses Modell gewählt habe, ist MVC (Model - View - Controller), da sie klar aufteilt, was jeder Teil der Anwendung tun soll: was er tut (Model), wie er aussieht (View) und wie alles zusammenhängt (Controller). Dieses Modell macht es auch einfacher, mit mehreren Personen effizienter zu arbeiten, während der Code sauberer bleibt, und in späteren Phasen eines Projekts erleichtert es auch die Skalierung der Anwendung.

    - Modell: Systemlogik und Gerätesimulation (Klassen Device, EnergyDataPublisher)
    - View: HTML + DOM zur Anzeige von Daten und Ergebnissen
    - **Controller**: EnergyDashboard-Klasse, die Modelle mit der Ansicht verbindet, Strategien anwendet usw.

2. Prinzipien und Patterns : Die folgenden Muster und Grundsätze werden für die folgenden Aktionen verwendet

    - **Observer**: In diesem Fall benachrichtigt **Device** das **EnergyDashboard**, das sowohl als Observer/Controller fungiert – es hört auf Updates und reagiert entsprechend innerhalb der MVC-Struktur.
    - **Strategy**: In diesem Fall wählt **DataAnalyzer** je nach Situation unterschiedliche Herangehensweisen zur Datenanalyse und nutzt die Methode, die am besten passt.

- **Factory (optional)**: Wird verwendet, um Geräte wie Kühlschränke oder Heizungen einfach zu erstellen, ohne sich um die Details jedes Typs kümmern zu müssen.

3. Integration der Web-Oberfläche: Ich werde drei Abschnitte verwenden, um die Web-Iterfas in meine zuvor erläuterte Architektur integrieren zu können

    - Mit HTML: Damit kann ich die grundlegenden Abschnitte anzeigen: aktueller Verbrauch, Verlauf, Analyseergebnisse.

    - Mit TypeScript: Damit kann ich die Daten verarbeiten und das DOM mit document.getElementById().innerHTML = ... aktualisieren.

    - Das EnergyDashboard fungiert als Brücke zwischen den simulierten Daten und der Ansicht.

4. Modularität und Erweiterbarkeit:

    Um Modularität und Erweiterbarkeit zu sichern, habe ich eine Lösung gefunden, bei der jedes Gerät von einer gemeinsamen Basis, nämlich Device, ausgeht, so dass neue Typen bei Bedarf auf relativ einfache Weise hinzugefügt werden können. Für die Analyse entschied ich mich, verschiedene Methoden zu verwenden, die denselben Regeln folgen, so dass ich verschiedene Analysemethoden hinzufügen kann, ohne alles zu sehr zu verkomplizieren. Und schließlich kann ich dank der Verwendung von Schnittstellen alles zusammenfügen und auf diese Weise auch neue Module hinzufügen, ohne den Hauptteil des Systems zu berühren, wodurch das System flexibel und einfach zu bedienen bleibt.

5. UML-Klassendiagramm