Technical Stack

Source File: generated-documents\technical-design\technical-stack.md
Generated: 08/07/2025 at 09:44:15
Generated by: Requirements Gathering Agent - PDF Converter

TechnicalStack

Generated by adpa-enterprise-framework-automation v3.1.6
Category: technical-design
Generated: 2025-07-05T17:04:50.120Z
Description:

Self-Charging Electric Vehicle (SCEV) Project: Technical Stack Overview

This document outlines the technical stack for the Self-Charging Electric Vehicle (SCEV) project, focusing on scalability, maintainability, and future-proofing. The project's complexity necessitates a modular and robust architecture.

1. Technology Stack Overview:

The SCEV project is a complex system integrating hardware and software components. The architecture will be divided into several layers:

- **Hardware Layer:** This layer encompasses the physical components of the energy harvesting system (photovoltaic panels, regenerative suspension, thermoelectric generators) and the vehicle's existing electrical system. Specific component choices will be determined during Milestone 1 (Component Feasibility & Simulation).

- **Embedded Systems Layer:** This layer involves microcontrollers and embedded systems responsible for managing the individual energy harvesting units. Data acquisition, pre-processing, and local control loops will reside here.
- **Data Acquisition and Processing Layer:** This layer gathers data from the embedded systems and transmits it to the cloud for further processing and analysis.
- **Cloud Processing and AI Layer:** This layer includes the AI-powered Energy Management Unit (EMU). This will be a cloud-based service leveraging machine learning for prediction, optimization, and user feedback.
- **User Interface Layer:** A user-friendly dashboard within the vehicle displays real-time energy generation data and system status.

2. Frontend Technologies:

- **Technology:** React.js with TypeScript
- **Version:** React 18, TypeScript 5
- **Justification:** React provides a component-based architecture, facilitating maintainability and scalability. TypeScript adds static typing for improved code quality and reduced runtime errors. The dashboard needs to be responsive and efficient, which React excels at.
- **Dependencies:** React Router, Material UI (or similar component library), charting library (e.g., Recharts).

3. Backend Technologies:

- **Technology:** Node.js with TypeScript and Express.js
- **Version:** Node.js 18, Express.js 4, TypeScript 5
- **Justification:** Node.js offers a non-blocking, event-driven architecture, well-suited for handling real-time data streams from the vehicle. TypeScript improves code maintainability, and Express.js provides a robust framework for building RESTful APIs.

- **Dependencies:** Mongoose (or similar ODM if a NoSQL database is chosen), JWT for authentication, appropriate cloud messaging service (e.g., AWS IoT Core, Google Cloud IoT Core).

4. Database Technologies:

- **Technology:** PostgreSQL (or a NoSQL database like MongoDB for scalability)
- **Version:** PostgreSQL 15 (or MongoDB 6)
- **Justification:** PostgreSQL offers robust relational database capabilities, ideal for structured data like vehicle telemetry and energy generation statistics. A NoSQL database like MongoDB could be considered for scalability if data volume becomes extremely large. The choice will depend on the data model and anticipated scale.

5. Infrastructure Components:

- **Cloud Provider:** AWS, Google Cloud Platform (GCP), or Azure (choice depends on existing infrastructure and expertise)
- **Services:** Cloud storage (for data logging and model training), serverless functions (for processing data streams), container orchestration (Kubernetes or similar for scalability), machine learning platform (for model training and deployment).
- **Justification:** Cloud infrastructure provides scalability, reliability, and cost-effectiveness. Serverless functions are ideal for handling asynchronous tasks like data processing, and a containerized architecture ensures easy deployment and scaling.

6. Development Tools:

- **IDE:** VS Code or similar
- **Linters:** ESLint, TSLint
- **Formatters:** Prettier
- **Version Control:** Git (see Section 10)

7. Testing Tools:

- **Unit Testing:** Jest, Mocha

- **Integration Testing:** Supertest, Cypress (for UI testing)
- **End-to-End Testing:** Selenium or Playwright (for broader system testing)

8. Monitoring Tools:

- **Application Monitoring:** Datadog, New Relic, Prometheus
- **Logging:** Winston (for Node.js backend), centralized logging service in the cloud (e.g., CloudWatch, Stackdriver).
- **Justification:** Comprehensive monitoring is crucial for a system of this complexity. Real-time monitoring of application performance and system health is necessary to identify and address issues promptly.

9. Deployment Tools:

- **CI/CD:** GitHub Actions, GitLab CI, Azure DevOps, AWS CodePipeline (choice depends on the chosen version control and cloud provider).
- **Containerization:** Docker
- **Orchestration:** Kubernetes (for scalability and management of microservices)

10. Version Control and CI/CD:

- **Version Control:** Git (GitHub, GitLab, or Bitbucket)
- **CI/CD Pipeline:** A robust CI/CD pipeline will be essential for automating the build, testing, and deployment processes. This will include automated testing at multiple levels (unit, integration, end-to-end) and deployment to staging and production environments. The specific tools will depend on the chosen cloud provider and team preferences.

Dependencies: All dependencies will be managed using package managers (npm or yarn) and clearly documented in package.json and package-lock.json (or yarn.lock).

This technical stack provides a solid foundation for the SCEV project, addressing scalability, maintainability, and future-proofing. Specific technologies and versions may be refined during the project's lifecycle based on further research and testing. The modular design allows for incremental development and easier integration of new technologies as they emerge.
