EAE Full-Stack challenge

# Objective

Recreate the provided sample vehicle dashboard interface and set up a backend to support it. The interface should display real-time data fetched from the backend and allow user interactions to modify backend values. Host the complete application on a static URL and provide the source code on GitHub.

# User Interface

The provided image below is a sample vehicle dashboard interface displaying information about a motor and a battery, along with some additional vehicle data. It includes various icons, gauges, and controls that provide both visual feedback and interactive elements. Your task is to recreate this interface.

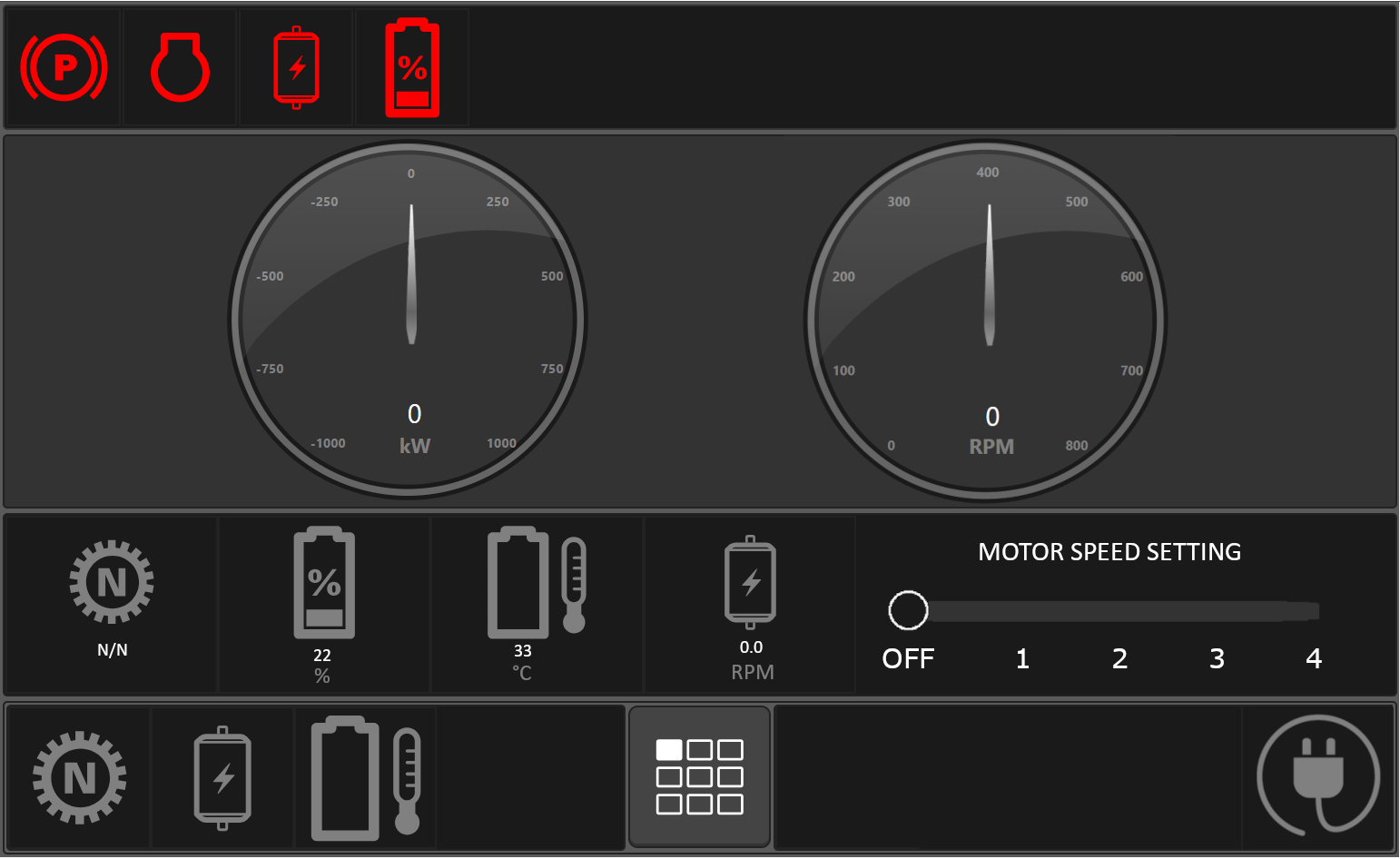


Figure 1 – Vehicle Dashboard Interface

**Interface Breakdown**

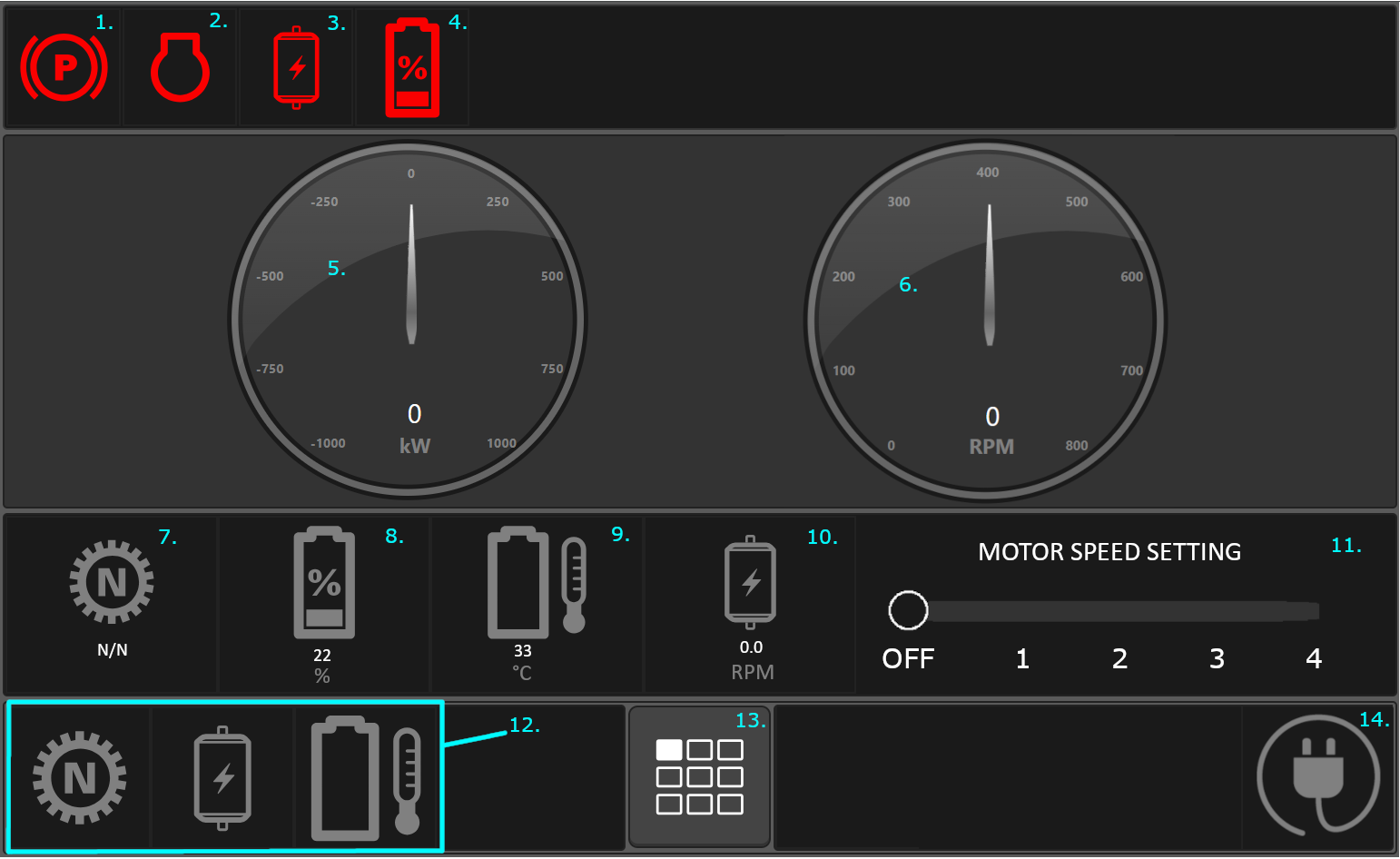
****

Figure 2 – Vehicle Dashboard Interface Annotated

Top Row Icons

1. **Parking Brake Indicator**
   * **Indicates whether the parking brake is engaged.**
2. **Check Engine Indicator**
   * Indicates the engine status.
3. **Motor Status Indicator**
   * Indicates whether the motor is operating at very high speeds.
4. **Battery Low Indicator**
   * **Indicates whether the battery has dropped below a defined threshold.**

**Gauges**

1. **Power Gauge**
   * **Displays the power drawn from the battery and consumed by the motor, or the power being put into the battery through charging.**
2. **Motor RPM Gauge**
   * **Displays the current RPM of the motor.**

**Middle Row Components**

1. **Gear Ratio** 
   * Shows the gear ratio of the motor.
2. **Battery Percentage**
   * Shows the remaining battery percentage.
3. **Battery Percentage**
   * Shows the current battery temperature.
4. **Motor RPM**
   * Shows the current RPM of the motor.
5. **Motor Speed Setting**
   * Slider that allows users to set the speed of the motor, with 1 being the slowest speed, and 4 being the fastest speed.

Bottom Row Buttons

1. **Gear, Motor, Battery Temperature Buttons**
   * Intended to bring users to views with more information on the gear, motor, or battery temperature.
   * NOTE: These information pages do **NOT** need to be created for the challenge application. These buttons should not be functional.
2. **View Menu Button**
   * Intended to bring users to a selection menu with other information views.
   * NOTE: This menu page does **NOT** need to be created for the challenge application. This button should not be functional.
3. **Charging Button**
   * Simulates connecting the battery to a charger and indicates whether the battery is charging.

# Backend and Database

The backend should be set up using any cloud service of your choice. The database should be hosted separately and should be used to store and manage all values displayed on the frontend. The backend should handle the emulation of motor RPM and battery levels, and nothing should be emulated on the frontend. All values displayed on the screen should be fetched from the database, and the database should be updated based on user interactions and changes through REST APIs that you create.

Requirements

**Backend Setup**

* Use a cloud service like Google Cloud Platform (GCP), Microsoft Azure, or Amazon Web Services (AWS) to set up your backend.
* Create a database using any database technology of your choosing. Host the database separately to store all necessary values such as motor RPM, battery percentage, charging state, and power consumption. The database should be set up with **read-only public access**, for easy viewing of the data for testing and evaluation purposes.

**Database Management**

* Ensure all values displayed on the frontend are fetched from the database. This includes:
  + Indicator light states
  + Power consumption or input
  + Motor RPM
  + Gear ratio
  + Remaining battery percentage
  + Battery temperature
  + Charging state
* The backend should emulate real-time changes in motor RPM, power consumption, and battery levels.
* Update the battery percentage and temperature in the database whenever the motor is in use, or whenever the battery is charging.
* Record charging state, battery levels, power consumption, and RPM in the database, and ensure these values are updated based on user interactions.

**REST APIs**

* Create the necessary REST APIs to:
  + Retrieve data to populate frontend values.
  + Update values in the database based on user interactions (e.g., motor speed changes, charging button press).

**Real-Time Database Interaction**

* Ensure that the database values are kept up-to-date and reflect real-time changes.
* Your application should emulate a system in which other components and dashboards in the vehicle could access and modify values in the database in real-time. (You do not need to implement any logic to emulate any other components or dashboards for this challenge).

# Functional Requirements

Your completed application should exhibit the following behavior:

**Status Indicator Lights**

* Parking Brake
  + Should be active when the parking brake is engaged.
  + Parking brake state should be read from the database but should not be modifiable from this dashboard.
* Check Engine
  + Should be active when the engine needs service.
  + Check engine state should be read from the database but should not be modifiable from this dashboard.
* Motor
  + Should be active when the motor is operating above a certain RPM.
* Battery Low
  + Should be active when the battery percentage dips below a certain level.

**Power Gauge**

* Should display the current power consumption or input.
* The needle should point to the position corresponding to the power level.
* The needle should spin with fluid animation when power levels change (not snap instantaneously to the correct value).
* Power levels should be positive when power is being consumed from the battery through the motor’s operation. The faster the motor runs, the higher the power consumption should be.
* Power levels should be negative when the battery is charging.

**Motor RPM Gauge**

* Should display the current RPM of the motor.
* The needle should point to the position corresponding to the RPM.
* The needle should spin with fluid animation when RPM changes.
* When the motor is off, RPM should be 0, and RPM should increase as users increase the speed using the Motor Speed Setting slider.
* When the battery is charging, the motor RPM should be 0.

**Gear Ratio**

* Should be read from the database and should not be modifiable from this dashboard.

**Battery Percentage**

* Should be read from the database and updated in the database through backend emulation.
* Should decrease over time when the motor is in use.
* Should increase over time when the battery is charging.

**Battery Temperature**

* Should be read from the database and updated in the database through backend emulation.
* Battery temperature should increase as the motor speed increases.

**Motor Speed Setting Slider**

* Should change the RPM of the motor, with RPM being set to 0 when slider is in OFF position, and with RPM increasing as the speed setting increases.

**Charging Button**

* Should update the charging state and indicate to the user whether we are charging.
* When charging, the motor should be disabled, and the battery percentage should increase over time.

# Submission

To submit your application and source code:

* Host your complete application on a static URL and send us this URL.
* Provide a public GitHub repository containing all the code for the frontend backend.
* Create a README file with setup instructions so that we can build and run your code.
* Provide us a link to your read-only publicly accessible database.

Additional Notes:

* Feel free to use any technology stack of your choosing for the frontend, backend, and database, and any service you like to host the application.
* Document your code with meaningful comments.