Web App Project

2025

ECU Error Manager

A full-stack application available in multiple languages for managing a comprehensive database of all vehicle ECU error codes and their descriptions.

Francesco Simoni

ECU-error-manager

A full-stack application available in multiple languages for managing a comprehensive database of all vehicle ECU error codes and their descriptions.

1 Introduction

The **ECU-error-manager** app is designed to streamline the diagnostic process by eliminating the need for mechanics and service assistants to navigate through often unintuitive, vendor-specific applications. These traditional tools can slow down troubleshooting by forcing users to manually search for error codes.

With ECU-error-manager, users can quickly access detailed descriptions of errors generated by any vehicle's ECU. The app features multiple settings that allow flexible searches based on the code type provided by the display—ensuring compatibility with both older and newer vehicle models. This adaptability makes it a valuable tool for efficient diagnostics and faster repair workflows.

For this project, given the very small database (only 20 MB), it wasn't strictly necessary to build a full-stack application with separate client, server, and database layers. In such cases, a pure React application would suffice to simplify data fetching, adding for example the DB file in the public folder if used locally by assistants. However, the full-stack architecture was intentionally implemented as both an exercise and a demonstration of modern web development practices.

The application is available in two versions.

Note: The deployment keys must be loaded appropriately. For example, the Render key should be set in the Netlify environment variables, and the MongoDB key should be configured in the Render environment variables.

---

2 Version 1: Node, React & MongoDB

Built using Node, React, and MongoDB, this version features:

- **Client:** A React-based front-end that sends requests for error code descriptions.
- **Server:** A Node server that processes client requests, queries the MongoDB database, and returns the results.
- **Database:** MongoDB Atlas stores the comprehensive database of ECU error codes.

Setup & How to Use:

1. Client Setup:

Create a React app using:

```
npx create-react-app@latest my-app
```

Overwrite the `App.js` and `index.js` in the `src` folder with those from this repository.

Run the following commands in the client directory:

```
npm install
npm install @mui/material @emotion/react @emotion/styled dompurify
npm start
```

Use `npm run build` to generate the production build folder to be published on Netlify. Adjust the API links as needed. Remember to add the Render environment variable in your Netlify settings.

2. Server Setup:
Run the server locally with:

```
node app.js
```

Or deploy the server on Render. Ensure that the MongoDB access key is added to the environment variables on Render.

3. Database Setup:
Load your database on MongoDB Atlas manually via the UI or by using the following command:

```
mongoimport --db YOUR_COLLECTION --collection your-db-name --file your-db-file.json --jsonArray
```

Test if it loaded correctly by connecting to your collection:

```
use YOUR_COLLECTION
db.your-db-name.find().pretty()
```

3 Version 2: Flask & React

This version is built using Flask (Python) for the server-side and React for the client-side, with the following characteristics:

- **Client:** The same as the Node Version.
- **Server-Database:** The Flask server integrates the database directly within the server, eliminating the need for an external database.

Setup & How to Use:

1. Client Setup:
The React client setup is similar to Version 1. Make sure to adjust the API endpoints to point to your Flask server. Run the following commands in the client directory:

```
npx create-react-app@latest my-app
npm install
npm install @mui/material @emotion/react @emotion/styled dompurify
npm start
```

2. Server Setup:
It is recommended to use a virtual environment for running the Flask server locally:

```
python -m venv venv
source venv/bin/activate   # On Windows use: .\venv\Scripts\activate
pip install flask flask-cors
```

Run the Flask server using the Python command.

Deployment Reminder:
As with Version 1, ensure that all necessary access keys are loaded into the deployment environment. For example, configure the relevant keys (such as for Render or any other service) in the corresponding environment variables on your chosen deployment platform.

4 Conclusions

This project demonstrates two different full-stack approaches to managing ECU error code databases. Whether using Node, React, and MongoDB or Flask and React with an integrated database, the goal is to offer a flexible, user-friendly tool for quick diagnostics and efficient repair workflows. The dual-version approach also serves as a practical exercise in modern web development, providing a platform to learn and compare different tech stacks and deployment strategies.