REST API: Car Management System – Lorena Spallino

A description of the site

The site that I have decided to make to showcase my knowledge of REST API functionalities is a system that manages cars, for a car collection, or maybe even a car dealership. This website can be used by a car enthusiast or car dealership to keep tract of their car collection, add new cars, edit current cars, or delete cars from their collection. This site uses REST API to asynchronously read, create, update and delete cars from the system.

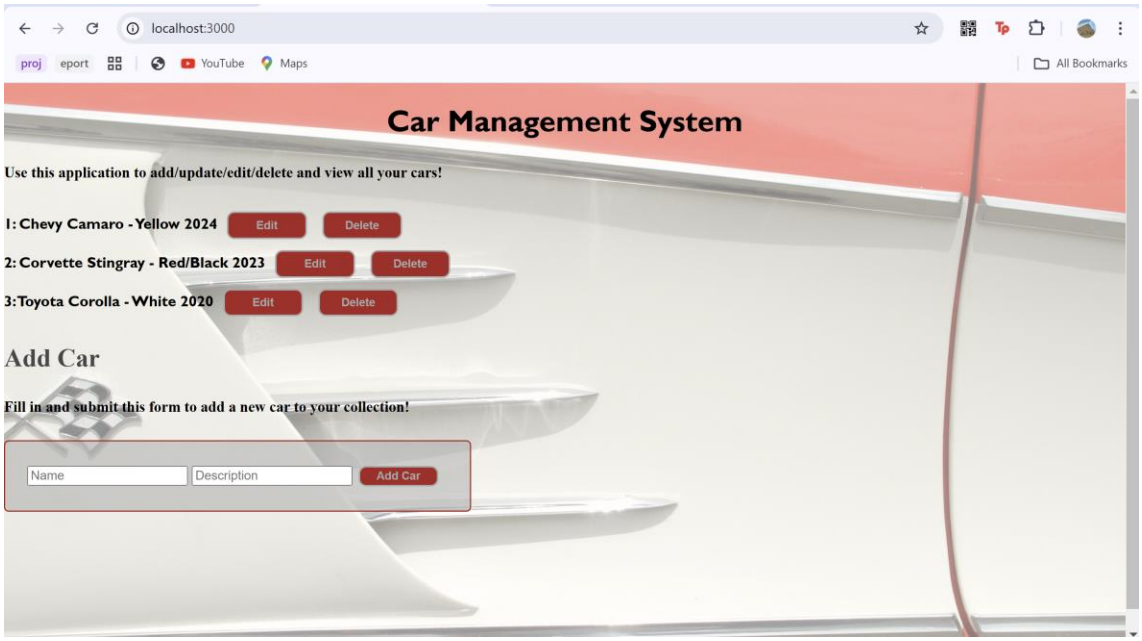
The web technologies from above that you used and why.

I mainly used JavaScript to configure the server and the REST API for this webpage, as well as HTML to display the page, manipulated using jQuery. I chose to manipulate the site with jQuery because of its simplicity, as well as the fact that it facilitates the development, and reduces development time. I used JSON to store and transmit all the car data for its simplicity to read and write, and faster transmission. Also, I used XML for the last assignment, so I wanted to use JSON this time, to practice it, and to learn more about using it in a practical application. To configure my REST API, I decided to use ‘express’ a framework that facilitates the definition of routes and endpoints in my API, as well as helps to handle the http methods. Also, I used CSS externally to create the layout of the website and establish a good user interface design to augment to user’s experience. Finally, I installed and configured ‘prettier’ on my visual studio code project to ensure that my source code was all properly formatted.

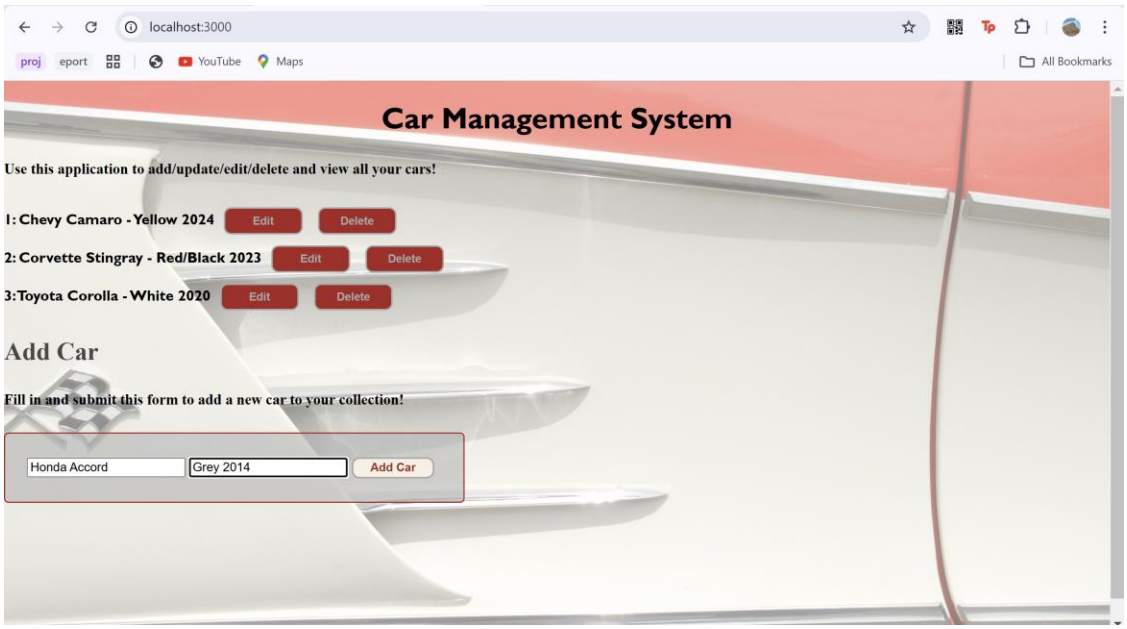
Screenshots of your website.

The following screenshots show the design, and functionality of my site:

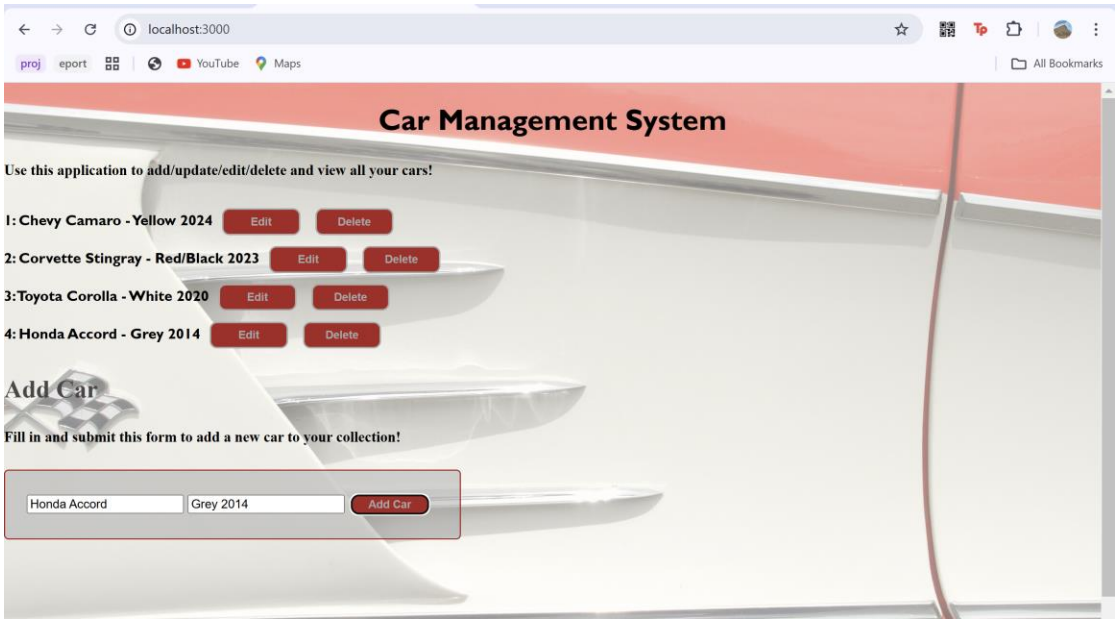
Landing Page: After starting the server and navigating to the correct port (3000) the user will land on the html page. The user will see the car management system site, where they can add edit delete and view their cars. This is shown below:



If they want to add a new car to their collection, the user can fill out the ‘Add Car’ form. As shown in the image below, including the name and description of their new car.



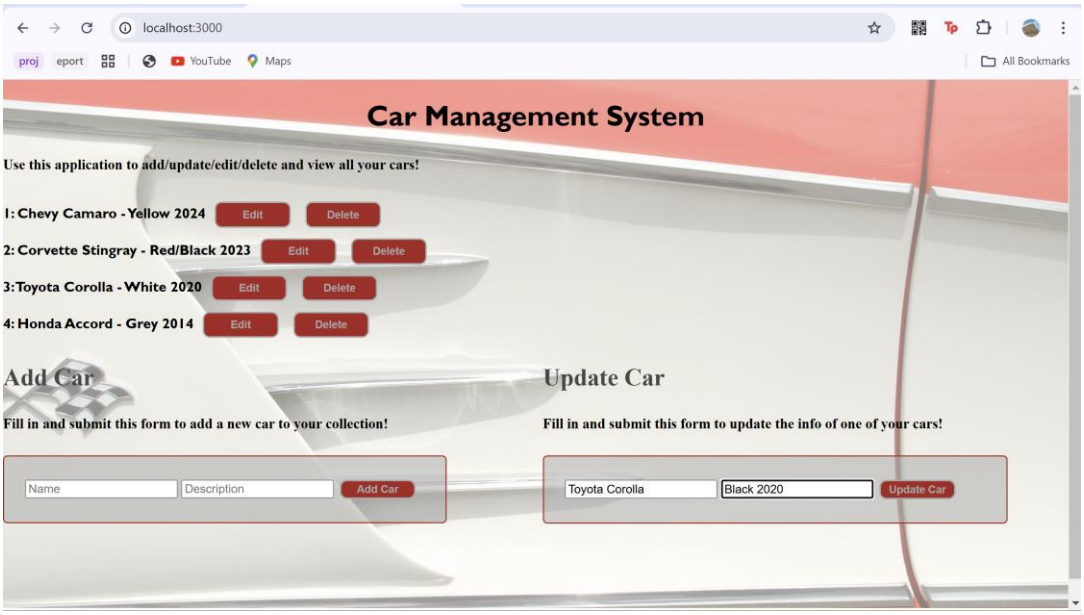
When the user clicks the ‘Add Car’ button, the form will clear, and an asynchronous request will be made to create the JSON for that car, add it to the cars.json document, then display all the cars, including the new one. This is shown below.



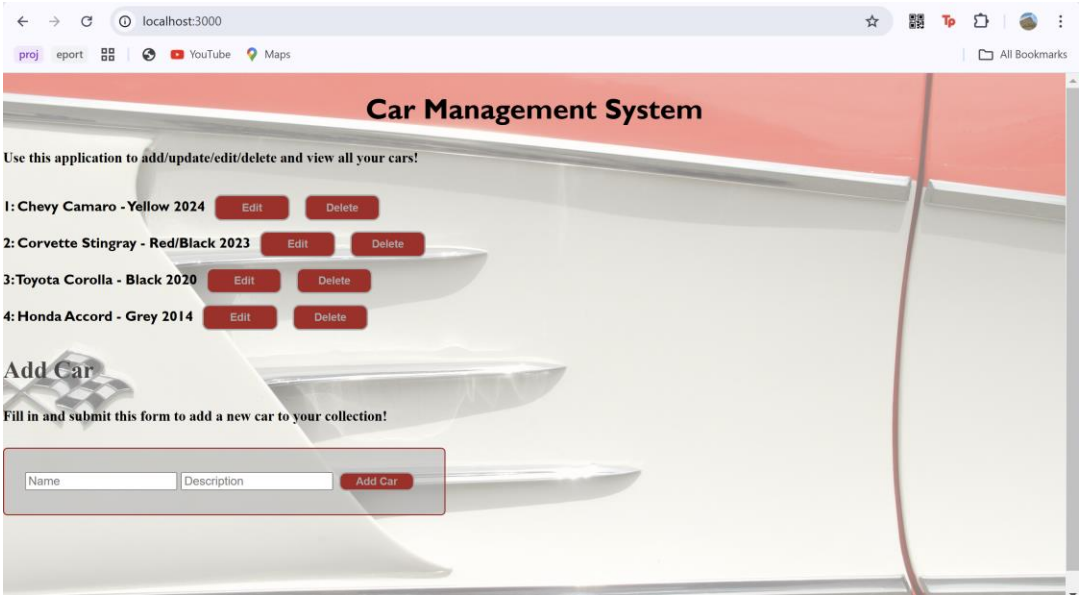
If the user notices an error in one of their cars, they can edit it using the ‘Edit’ button beside the respective car. They will then see an ‘Update Car’ form pop up, which they

can fill out to edit that car. Then, they can click the ‘Update Car’ button, which will asynchronously make the updates to that specific car, save it, and load the cars again from the json document. This is shown below:

Step 1: Filling out the form:

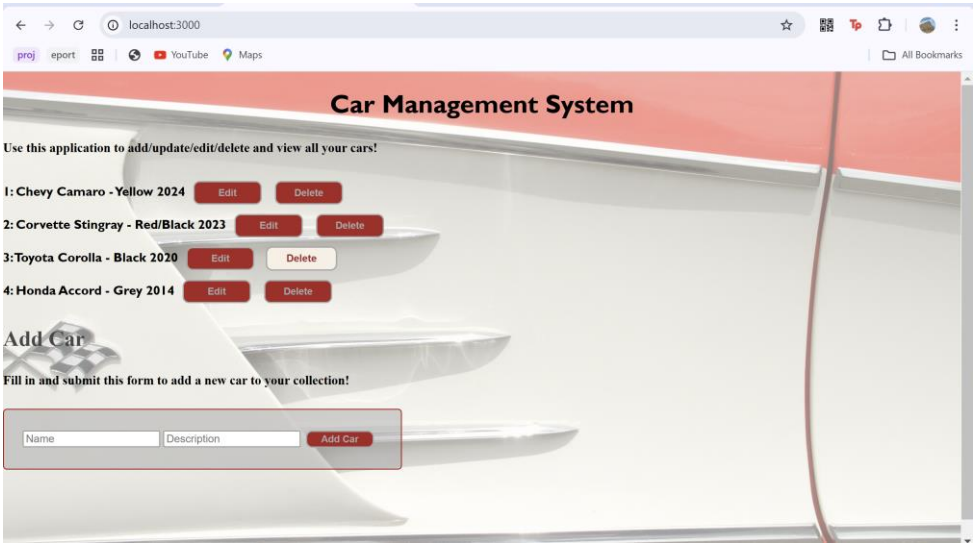


Step 2: Update Car – (Notice how the colour of the Toyota Crolla changed):

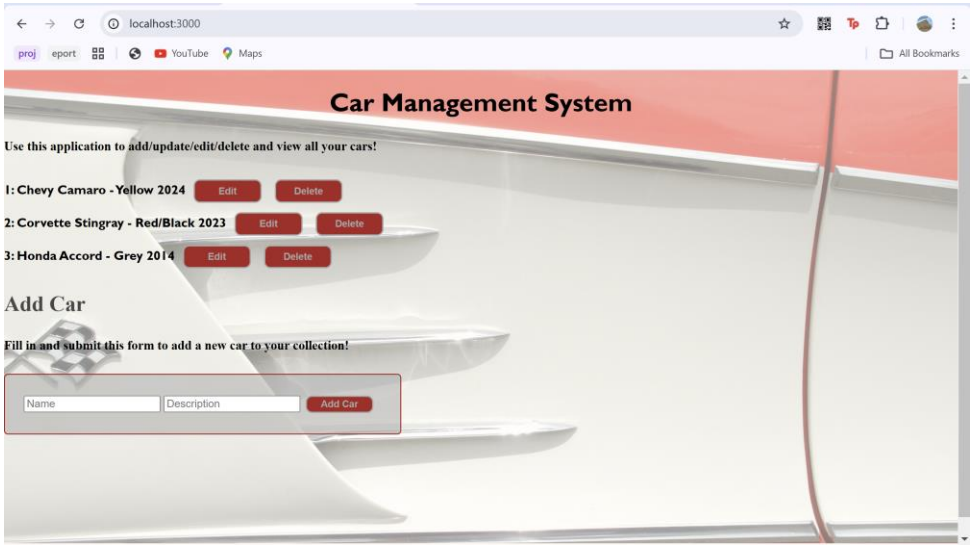


Finally, if the user wises to delete a car from their system, if, for example they sold it, they can use the ‘Delete’ button beside that car to asynchronously remove the car from their collection (deleting the car info from the cars.json). Then the ID of any car after the deleted one will be ‘reassigned’ so that they are still in sequential order. This is shown below:

Step 1: Determine which car to delete, and use ‘Delete’ button next to that car:

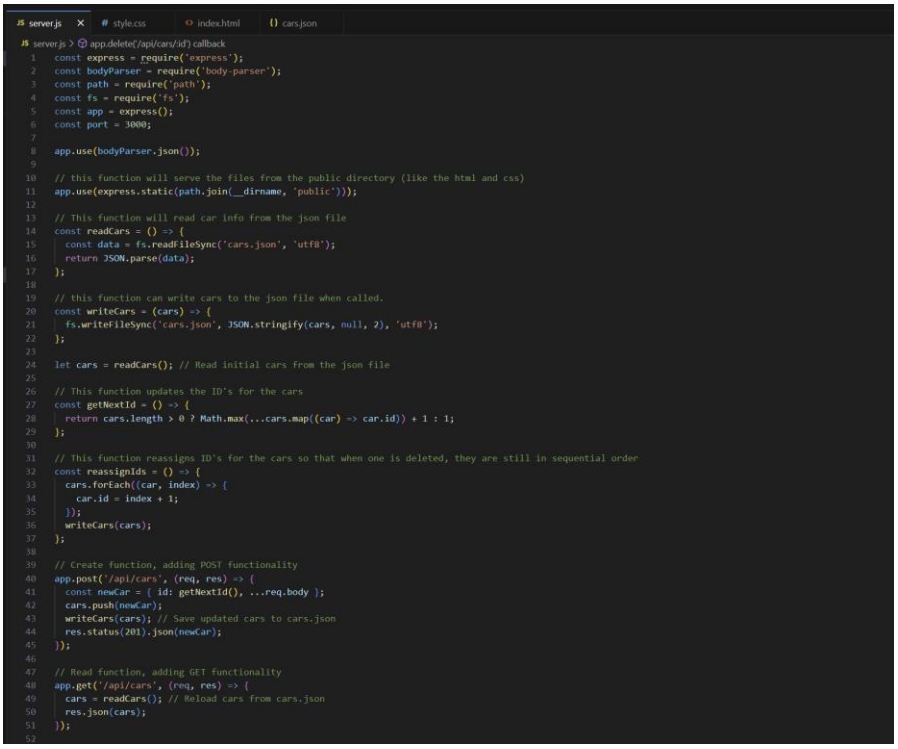


Step 2: Delete car, and view the rest of your car collection:

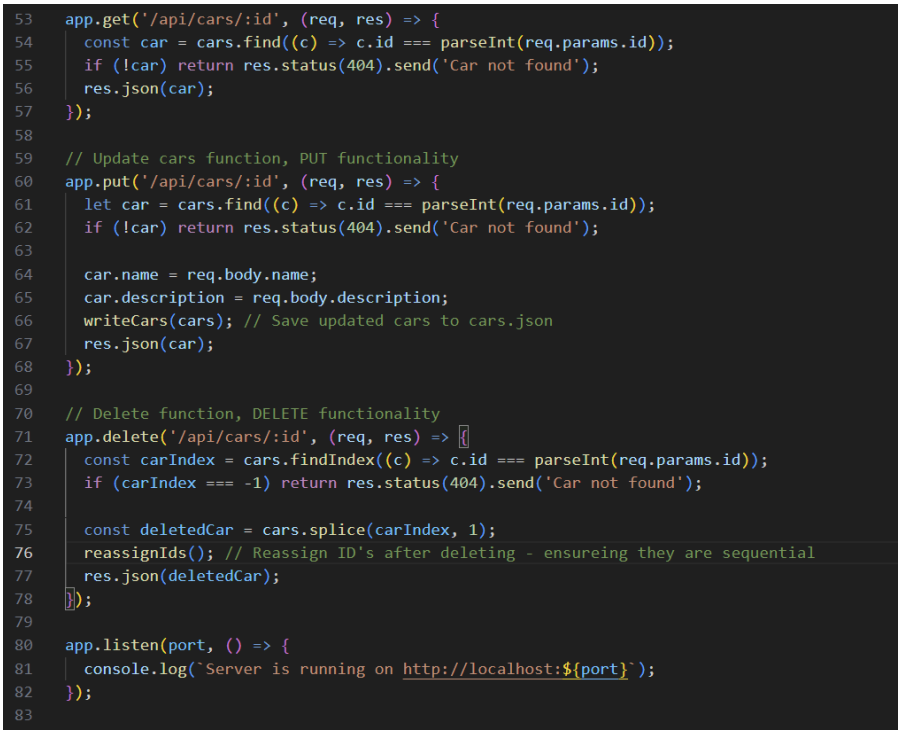


Source Code Screenshots:

Server.js:



```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const app = express();
const port = 3000;

app.use(bodyParser.json());

// this function will serve the files from the public directory (like the html and css)
app.use(express.static(path.join(__dirname, 'public')));

// This function will read car info from the json file
const readCars = () => {
  const data = fs.readFileSync('cars.json', 'utf8');
  return JSON.parse(data);
};

// this function can write cars to the json file when called.
const writeCars = (cars) => {
  fs.writeFileSync('cars.json', JSON.stringify(cars, null, 2), 'utf8');
};

let cars = readCars(); // Read initial cars from the json file

// This function updates the ID's for the cars
const getNextId = () => {
  return cars.length > 0 ? Math.max(...cars.map((car) => car.id)) + 1 : 1;
};

// This function reassigns ID's for the cars so that when one is deleted, they are still in sequential order
const reassignIds = () => {
  cars.forEach((car, index) => {
    car.id = index + 1;
  });
  writeCars(cars);
};

// Create function, adding POST functionality
app.post('/api/cars', (req, res) => {
  const newCar = { id: getNextId(), ...req.body };
  cars.push(newCar);
  writeCars(cars); // Save updated cars to cars.json
  res.status(201).json(newCar);
});

// Read function, adding GET functionality
app.get('/api/cars', (req, res) => {
  cars = readCars(); // Reload cars from cars.json
  res.json(cars);
});
```



```
app.get('/api/cars/:id', (req, res) => {
  const car = cars.find((c) => c.id === parseInt(req.params.id));
  if (!car) return res.status(404).send('Car not found');
  res.json(car);
});

// Update cars function, PUT functionality
app.put('/api/cars/:id', (req, res) => {
  let car = cars.find((c) => c.id === parseInt(req.params.id));
  if (!car) return res.status(404).send('Car not found');

  car.name = req.body.name;
  car.description = req.body.description;
  writeCars(cars); // Save updated cars to cars.json
  res.json(car);
});

// Delete function, DELETE functionality
app.delete('/api/cars/:id', (req, res) => {
  const carIndex = cars.findIndex((c) => c.id === parseInt(req.params.id));
  if (carIndex === -1) return res.status(404).send('Car not found');

  const deletedCar = cars.splice(carIndex, 1);
  reassignIds(); // Reassign ID's after deleting - ensureing they are sequential
  res.json(deletedCar);
});

app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

Index.html (manipulated with jQuery):

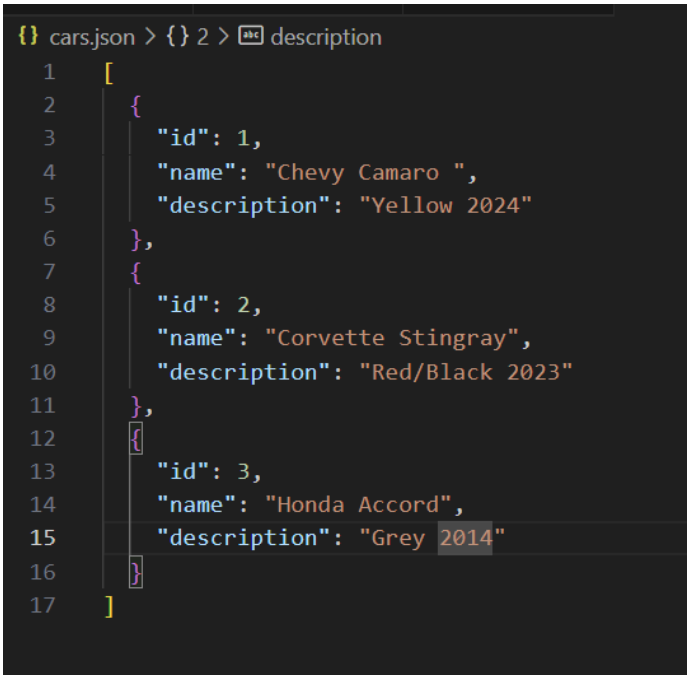
public > index.html > html > body > script

```html
<!doctype html>
<html>
  <head>
    <title>REST API with Node.js</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <link rel="stylesheet" href="style.css" />
  </head>

  <body>
    <h1>Car Management System</h1>
    <h5>
      Use this application to add/update/edit/delete and view all your cars!
    </h5>
    <div id="cars"></div>
    <div id="columns">
      <div id="col1">
        <h2>Add Car</h2>
        <h5>
          Fill in and submit this form to add a new car to your collection!
        </h5>
        <form id="addCarForm">
          <input type="text" id="name" placeholder="Name" required />
          <input
            type="text"
            id="description"
            placeholder="Description"
            required
          />
          <button type="submit">Add Car</button>
        </form>
      </div>

      <div id="col2">
        <form id="updateCarForm" style="display: none">
          <h2>Update Car</h2>
          <h5>
            Fill in and submit this form to update the info of one of your cars!
          </h5>
          <div id="updateForm">
            <input type="hidden" id="updateId" />
            <input
              type="text"
              id="updateName"
              placeholder="New Name"
              required
            />
            <input
              type="text"
              id="updateDescription"
              placeholder="New Description"
              required
            />
            <button type="submit">Update Car</button>
          </div>
        </form>
      </div>
    </div>

    <script>
      const apiUrl = 'http://localhost:3000/api/cars';
      // This function asynchronously 'gets' all the car data. this is the READ function in CRUD
      // This function also adds buttons next to each element for edit and delete, which ass the UPDATE and DELETE functionalities.
      function fetchCars() {
        $.get(apiUrl, function (cars) {
          $('#cars').empty();
          // This uses a foreach loop to iterate through the json data.
          cars.forEach((car) => {
            $('#cars').append(`
                        <div>
                            ${car.id}: ${car.name} - ${car.description}
                            <button onclick="editCar(${car.id}, '${car.name}', '${car.description}')">Edit</button>
                            <button onclick="deleteCar(${car.id})">Delete</button>
                        </div>
                    `);
          });
        });
      }
```

Index.html (continued)

```
    $('#addCarForm').on('submit', function (event) {
      event.preventDefault();
      //This code block takes the values from the 'addCarForm' into a variable, for the new car
      const newCar = {
        name: $('#name').val(),
        description: $('#description').val(),
      };
      // This function uses ajax to make an async request to add the new car information to the json data and displays it with the others.
      // This funtion adds the CREATE funtionality in CRUD.
      $.ajax({
        url: apiUrl,
        type: 'POST',
        contentType: 'application/json',
        data: JSON.stringify(newCar),
        success: function (response) {
          fetchCars();
        },
        error: function (error) {
          console.error('Error:', error);
        },
      });
    });

    // When the updateCarForm is submitted this function is called to asyncronously update the respective car info.
    $('#updateCarForm').on('submit', function (event) {
      event.preventDefault();

      const id = $('#updateId').val();
      const updatedCar = {
        name: $('#updateName').val(),
        description: $('#updateDescription').val(),
      };

      $.ajax({
        url: `${apiUrl}/${id}`,
        type: 'PUT',
        contentType: 'application/json',
        data: JSON.stringify(updatedCar),
        success: function (response) {
          $('#updateCarForm').hide();
          fetchCars();
        },
        error: function (error) {
          console.error('Error:', error);
        },
      });
    });

    function editCar(id, name, description) {
      $('#updateId').val(id);
      $('#updateName').val(name);
      $('#updateDescription').val(description);
      $('#updateCarForm').show();
    }

    //This function uses an ajax async request to delete the info of one of the cars.
    //This adds the DELETE function in CRUD.
    function deleteCar(id) {
      $.ajax({
        url: `${apiUrl}/${id}`,
        type: 'DELETE',
        success: function (response) {
          fetchCars();
        },
        error: function (error) {
          console.error('Error:', error);
        },
      });
    }

    $(document).ready(function () {
      fetchCars();
    });
  </script>
 </body>
</html>
```

Cars.json (updates as user uses the site):



What you want to accomplish with the site development why you chose this, and how it is supposed to work

The task I wanted to accomplish with this website was for individuals to keep track of their car collections. I chose to make this site because, I really love cars, I enjoy working on cars, watching car shows, and when it was time for me to buy my car, I really enjoyed the process of searching for cars and going to the dealerships. I wanted to create not only something that I feel passionately about, but something practical for car dealerships or individuals to use to keep track of and manage their car collections. This site is supposed to work by allowing the user to add each of their cars in their collection to the system by using the ‘Add Car’ form, which will then add this information to the cars.json file, asynchronously. Then the user can asynchronously manage their collection by editing the info of each of their cars, which will of course update the values

in the cars.json file, as well as delete any car from their collection. They can use this website to completely modify, update their cars, add new ones delete ones, or simply view their car collection all in an easy and user-friendly way.

Short descriptions of the struggles you encountered.

- Configuring Prettier:
  - I had never used ‘prettier’ before, or even heard about it, so it was a challenge for me to configure it so that it would automatically format my source code. After some research, and learning about how to properly configure this, I learned how to install it on my VS code, enable it and configure it within my project to format JavaScript, HTML and CSS automatically. This was beneficial because it formatted by code when I saved it, so I didn’t need to take any more steps to format it in a third-party website.
- Setting up a Node.js project:
  - Being that I never used node.js in any other classes, this was a completely new concept for me. It was very difficult for me to wrap my head around at first. Setting up the server and the project as a whole, was different than how I have done it before, and so it was a little challenging for me. After researching and testing, I finally figured it out, and created a properly setup project, that runs without any problems!

References

Building a RESTful API with Node.js and Express: A Comprehensive Guide. (2024, December 4). Codez Up. Retrieved December 10, 2024, from https://codezup.com/node-js-express-restful-api-tutorial/

Codex, A. C. (2023, July 30). *What are the best libraries for using Node.js with a REST API?* Reintech. Retrieved December 10, 2024, from https://reintech.io/blog/using-node-js-with-a-rest-api-the-best-libraries-for-developers?

Creating Efficient RESTful APIs with Node.js and Express.js Development. (2024, December 2). Codez Up. Retrieved December 10, 2024, from https://codezup.com/creating-restful-apis-with-node-js-and-express-js/

Hall, E. (2024, November 2). *RESTful Web Services Tutorial: What is REST API with Example*. Guru99. Retrieved December 10, 2024, from https://www.guru99.com/restful-web-services.html

Jain, S. (2022, December 22). *RESTful Web Services*. GeeksforGeeks. Retrieved December 10, 2024, from https://www.geeksforgeeks.org/restful-web-services/

Usage and example. (n.d.). Node.js. Retrieved December 10, 2024, from https://nodejs.org/api/synopsis.html