Apollo Engineering Coding Exercise

Problem Statement

For this exercise you will build a simple web service that provides CRUD-style API access to data stored in a database system. You may use any programming language or database system you'd like.

1. Create a database table that holds records representing Vehicles.

 * A vehicle is uniquely identified by the VIN (Vehicle identification number). In addition, the following additional attributes need to be considered:

  * manufacturer name (string)

  * description (string)

  * horse power (integer)

  * model name (string)

  * model year (integer)

  * purchase price (decimal)

  * fuel type (string)

 * VIN has a unique constraint (case-insensitive).

2. Implement the following API endpoints:

Method	**URI**	**Response Status**	**Request Entity**	**Response Entity**	**Side Effects**
`GET`	`/vehicle`	`200 OK`	JSON-formatted representation of vehicles	JSON-formatted representation of all the records in the `Vehicle` table	
`POST`	`/vehicle`	`201 Created`	JSON-formatted representation of a unique vehicle	JSON-formatted representation of the new vehicle, including its unique identifier	System assigns a unique identifier and inserts a record into the `Vehicle` table
`GET`	`/vehicle/{:vin}`	`200 OK`	JSON-	JSON-	

			formatted representation of a unique vehicle	formatted representation of the identified vehicle	
`PUT`	`/vehicle/{:vin}`	`200 OK`	JSON-formatted representation of an existing vehicle	JSON-formatted representation of the updated vehicle	System updates the corresponding record in the `Vehicle` table
`DELETE`	`/vehicle/{:vin}`	`204 No Content`	No response body	System deletes the corresponding record from the `Vehicle` table	

3. Whenever the server cannot parse a request entity as a JSON representation of a Vehicle, it should respond with `400 Bad Request` response and a JSON-formatted error representation.

4. Whenever the server is able to parse a request entity as a JSON representation of a Vehicle, but the entity is invalid due to null or malformed attributes, or other validation failures, it should respond with `422 Unprocessable Entity` and a JSON-formatted representation of the encountered errors.

5. Ensure that the project is buildable from the Unix command line. Please do not submit a project that can only be built with Eclipse or other IDE.

6. Implement whatever unit and component tests you feel are useful to ensure the quality of your work. Tests should be able to run via Maven, SBT, Gradle, PyTest or another command line tool that is easily installable with Homebrew or a similar package manager. **Do not** use GUI based tools for your testing.

7. Provide a README file and/or create whatever other documentation you would want to provide to other members of your team who may be tasked to maintain or extend the service.

The exercise should generally not take more than 3 or 4 hours, although you're free to take as much time as you'd like to work on it. If you don't finish within a few hours, that's okay; submit what you've got anyway.

To submit the exercise

1. Push the code to a public [GitHub](https://github.com/) repository. **We cannot accept email attachments or file sharing links.**

2. The application should be runnable through your localhost. Deploying to a public cloud such as AWS, Heroku is optional.

Please follow all of the above instructions and send code that you are proud to show us!