Spring 2022 / CMPS 350 / B01 / Lab Final

Name and ID #:

Copy the `final` directory from the Teams lab channel into your personal GitHub repository under `exams/final`. Implement your application under the `application` directory with the specifications listed hereinafter. Push your work regularly to your repository throughout the exam to avoid any surprises.

Summary

We are going to develop an application that allows us to manage a collection of traffic violations along with an associated collection of vehicles. We will start by creating a server with an API that allows us to interact with our collections of vehicles and violations, following the same patterns and techniques used throughout our lab sessions using `Node` along with `Express`, `Mongoose`, and `MongoDB`. We will then test our API using a basic client-side interface rendered using `Handlebars` that lists all of the vehicles along with their corresponding violations and provides a way for paying a given violation.

Prerequisites

Before we get started, make sure that your `MongoDB` instance is up and running. The application structure and files are already provided under the `final/application` directory. Open this directory using your favorite IDE and install all of the dependencies listed in `package.json`; you won't need to install any extra modules further down the line.

Requirements

We want to be able to read all vehicles, create a new vehicle, and update a vehicle. We also want to be able to create a new violation for a given vehicle, read violations for a given vehicle, and update a violation.

You should implement these functionalities using the following nouns in your RESTful API:

- `/api/vehicles`
- `/api/vehicles/:vehicleid`
- `/api/vehicles/:vehicleid/violations`
- `/api/violations/:violationid`

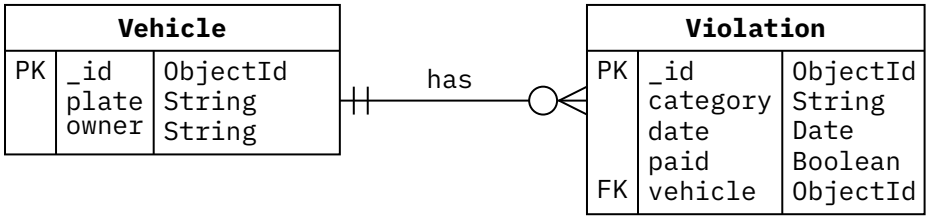
You will have to pick the correct verb and noun for every request and include that in your router. You will be using JSON to serve/parse all of your response/request bodies. These requests should be paired with a corresponding method in your service to handle the request and return a response. Your service should use a repository to interact with your database and collections. You will establish a connection to your

database in your repository and populate the collections if they're empty using the contents of the provided in `data/traffic.json` file.

Your client-side implementation will then use your API to fetch all the vehicles and their violations, render them using `Handlebars`, and provide a button to pay a violation using your API.

Instructions

1. [5%] Use `Mongoose` to create a `Vehicle` model and a `Violation` model using schemas that model the fields and relationship depicted below. All of the fields are required.



2. [15%] Create a repository that provides the methods needed for your API to implement the aforementioned requirements: reading all vehicles, creating and updating a vehicle; and reading all violations for a given vehicle, creating a violation for a given vehicle, and updating a violation.

3. [15%] Provide an initialize method that establishes a connection to the `MongoDB` `traffic` database and populate its `vehicles` and `violations` collections (if they are empty) with documents created from the contents of `data/traffic.json`. Make sure you assign document references correctly.

4. [15%] Create your service class with the required methods to handle all of the aforementioned requests while making use of your repository.

5. [10%] Create your router with the required entries to handle all of the aforementioned requests. Make sure you bind each request to the corresponding method in your service.

6. [5%] Create your server with the required middleware and attach your router instance.

7. [15%] Build a client-side web application that uses your API requests and `Handlebars` to render all the vehicles along with their violations. Display all the fields except `_id` and `__v`. You can use a single table or nested lists to display the vehicles and violations, for example.

8. [5%] Add a button to every violation that allows the end-user to "pay" that violation, that is, set the paid field of a given violation in your database to true using our API requests.

9. [10%] Update your API to provide server-side rendering of the index page using `Handlebars` and make that accessible through the `/ssr` path in your router.

10. [5%] Update your API to return all vehicles with their violations populated so that they can be fetched in one go instead of having to use multiple requests for each vehicle.

11. [Bonus: 10%] Use the `Mocha` and `Chai` libraries to test your API for reading all vehicles and creating a new vehicle.

Add screenshots to document your application in the `screenshots` directory under the `final` directory. The thoroughness of the implementation and its use of best practices will be taken into account when assigning the final grade.