Advance in Databases
Smart Vehicle Fine Management System

Mytreyan Jp 2022115102

Introduction

The primary goal of the Auto Fine system is to automate the fine notification process by ***simulating*** vehicle number detection and tracking traffic violations. The system detects the vehicle number and the registers the offence committed, to the database. Then the registered vehicle's owner details is queried for their contact details. It is used to notify the respective person about the offence and the fine amount for their vehicle.

Note: The Detection of vehicle number and offence are **simulated**. The project concentrates on the data processed in backend database.

System Architecture

- **Vehicle Number and Offense Detection (Simulation):** During this phase, the system simulates the identification of vehicle numbers and the offenses committed by drivers. The **Tech stack** includes:
  - **HTML, CSS, JS:** These are used to create a simulation framework that generates vehicles with registered numbers and connects to the server.
  - **PHP:** The PHP server facilitates communication between the application and the MongoDB database.
- **Database Management:** A MongoDB database is employed to store all data related to vehicles and offenses. It consists of two collections:
  - **Vehicledetails:** This collection records vehicle information, including the owner's name, phone number, email, and details about fines for various traffic violations.
  - **Vehiclespassed:** This collection stores records of vehicles that have passed through the system, current fine amount (if any) including the time and date of passage and the status of notifying the respective owner.
- **Query and Notification:** The system queries the database for the vehicle owner's details based on the detected vehicle number. Once the owner is identified, the system retrieves the relevant current incremented fine for the offense and notifies the respective owner.

Database Structure

The **'Registrations'** Database is made up of two collections. Below is the JSON format for the documents from each collection.

Vehiclesdetails


```
{
 "_id": {"$oid": "6700c58770778c41" },
 "registration_number": "TN09AB1234",
 "vehicle_model": "Honda City",
 "vehicle_type": "Car",
 "ownerdetails": {
  "name": "Surya",
  "state": "Tamil Nadu",
  "address": "123, Green Street, Chennai",
  "email": "surya123@gmail.com",
  "phone_number": "9876543210",
  "aadhaar_number": "1234-5678-9012",
  "license_details": "TN09-2023-LC4567"
 },
 "offence": {
  "number_of_offences": 1,
  "total_fine": 1500,
  "due_date_to_pay_fine": "2024-10-10",
  "offences": [
    {
     "type_of_offence": "Speeding",
     "fine_for_that_offence": 1500,
     "offence_issued_date": "2024-09-30",
     "offence_issued_time": "14:30"
    }
   ]
 },
"image_url": “path to image”
}
```


Vehiclespassed


```
 {
"_id": {
 "$oid": "6705368c2feae8e04f026779"},
 "registration_number": "KL09AP1234",
 "owner_name": "Manoj Pillai",
 "vehicle_type": "Auto",
 "totalfine": 3425,
 "phone_number": "+91-9891234567",
 "time_passed": "19:11:32",
 "status": "Message sent"
 }
```


As the number of vehicles and offences grows, this structure allows easy horizontal scaling through sharding, handling higher loads and more traffic efficiently. Each collection can be independently managed and scaled, allowing flexibility in expanding the database without disrupting the entire system.

Mytreyan Jp
2022115102

Sharding

In this project, sharding will improve performance by splitting the **vehicledetails collection** across multiple shards, each serving as a portion of the dataset. For this project, several fields in the vehicledetails collection can serve as potential shard keys:

- **Registration Number (registration_number)**: With high cardinality, this key uniquely identifies each vehicle, ensuring even data distribution.
- **Vehicle Type (vehicle_type):** This key might work if the data set includes a wide variety of vehicle types, though it risks uneven distribution if certain types are more prevalent.
- **Fine Status (fine_status):** This key could help organize data by fine status (e.g., "paid", "unpaid"), balancing shards based on vehicles with overdue fines.

In this project, **registration_number** or **fine_status** are recommended as shard keys due to their *uniqueness* and *high cardinality*, which helps distribute data effectively across shards.

Usages of Sharding:

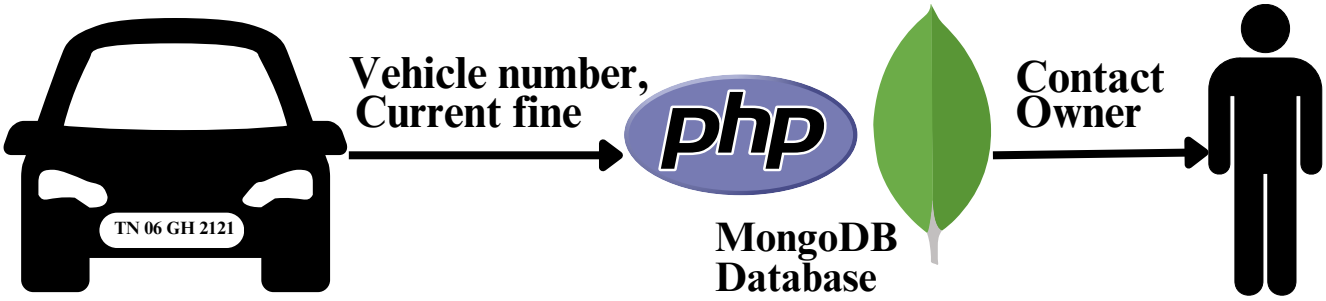
To utilize sharding's performance benefits, read and write operations are optimized as follows:

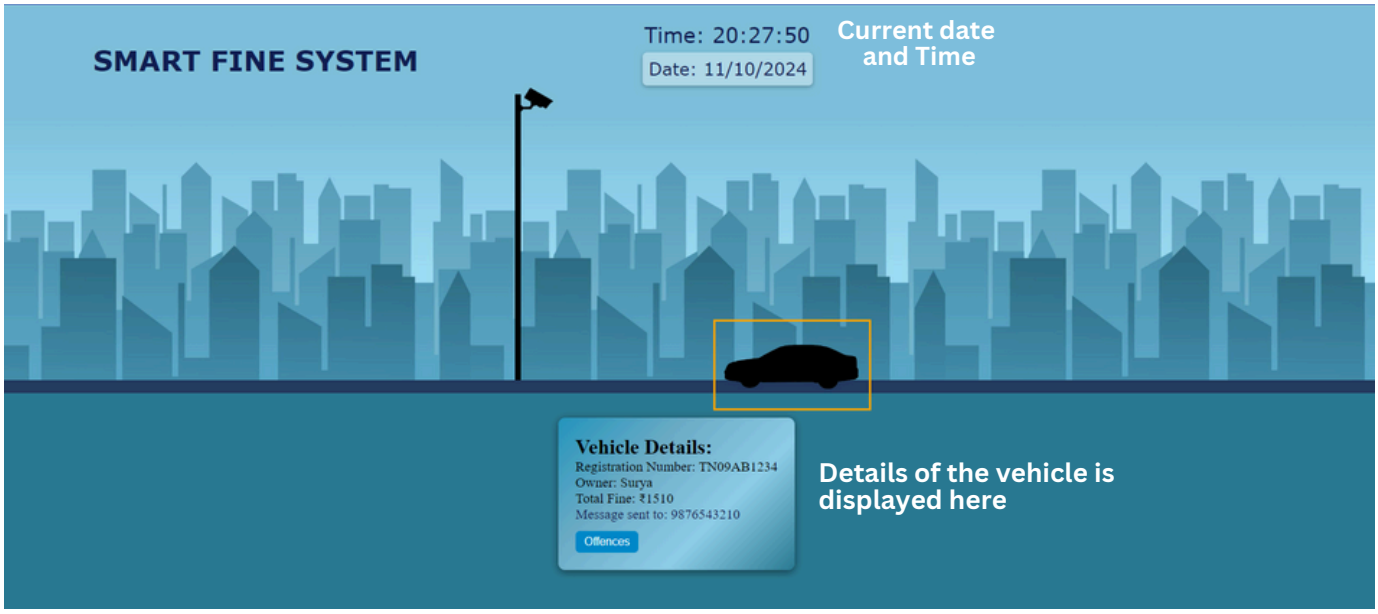
- **Targeted Queries:** By including the shard key in queries, the application directs operations to the correct shard, reducing cross-shard communication.
- **Secondary Reads:** For read-heavy applications that can tolerate slightly stale data, secondary reads are enabled to distribute the load across replicas within each shard.

Work flow

The system retrieves the registered vehicle's information from the **"vehiclesdetails"** collection and verifies the current fine accordingly. It checks the due date, and if the fine for the registered offense has not been settled by that date, a new fine is added to the total based on the *difference between the current date and the due date*. This is updated in the **"vehiclespassed"** collection.

Mytreyan Jp
2022115102





The information from the vehicles stored in the **"vehiclespassed"** collection can serve multiple purposes, including creating reports, analyzing traffic trends, and enhancing road safety initiatives. This data can be applied in *highway toll systems* and can also be employed in *metropolitan areas* for efficient monitoring of vehicles involved in offenses.

Mytreyan Jp
2022115102

System Operations

The Application provides multiple operations such as:

- **Dynamic Vehicle Details Display:** As each vehicle moves, its details (such as the vehicle number and owner information) are displayed in a panel on the screen.
- **Offence Details Toggle:** Clicking the "Offences" button toggles the visibility of the offence details panel, displaying the type of offence, fine amount, and date/time when it was issued.
- **CRUD Operations :**
  - **Create:** New vehicle entries are added to the database as they enter the system. Each vehicle's details, including owner and offences, are saved as a document in the "***vehiclespassed"*** collection.
  - **Read:** The project reads the details of each vehicle and their offences from the database when the animation starts or when interacting with the offence button.
  - **Update:** Offence details can be updated for an already registered vehicle. The project can update the offence field dynamically without altering other vehicle details.
  - **Delete:** Vehicle records can be removed from the collection if necessary (e.g., if the vehicle no longer needs to be displayed, or if the offences are cleared).
- **Database Optimizations:** Indexing is applied on fields like *vehicle_number* to speed up search and retrieval operations, especially when querying offences for specific vehicles.

Conclusion

This project successfully integrates frontend animation and dynamic database management using MongoDB to create an interactive system for visualizing vehicle movement and offences in real time. The application efficiently manages vehicle data, storing and retrieving details from the database to update the display as vehicles move across the screen.

Overall, the project demonstrates a comprehensive approach to database manipulation, animation, and user engagement, making it a powerful tool for monitoring vehicle movement and offences in an engaging and efficient manner.

Mytreyan Jp
2022115102