Batch: D-2	**Roll No.: 16010123325**
Experiment / assignment / tutorial No._5___	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of CRUD Operations using MongoDB and Mongoose.

AIM: Demonstrate the use of Mongoose with CRUD operation.

Problem Definition:

1) Generate Database model

2) Create RESTful API

3) Demonstrate the Endpoints.

Resources used:
https://www.codecademy.com/article/what-is-crud-explained
https://restfulapi.net/

Expected OUTCOME of Experiment:

CO 2:

Books/ Journals/ Websites referred:

1. Shelly Powers Learning Node O' Reilly 2 nd Edition, 2016.

Pre Lab/ Prior Concepts:

Write details about the following content

- Mongoose CRUD operation
  - **Mongoose** is an Object Data Modeling (ODM) library for **MongoDB** and **Node.js**. It lets you define schemas and interact with MongoDB using JavaScript instead of raw queries.
  - **CRUD** stands for **Create, Read, Update, Delete** - the four basic database operations.

Operation	Mongoose Method	Description
Create	Model.create() or new Model().save()	Adds a new document to a collection
Read	Model.find() or Model.findById()	Retrieves one or multiple documents
Update	Model.updateOne() or Model.findByIdAndUpdate()	Modifies existing document(s)
Delete	Model.deleteOne() or Model.findByIdAndDelete()	Removes document(s) from the collection

  - **Example:**

```
// Schema & Model
const userSchema = new mongoose.Schema({ name: String, age: Number });
const User = mongoose.model("User", userSchema);

// Create
await User.create({ name: "Shreyans", age: 19 });

// Read
const users = await User.find();

// Update
await User.updateOne({ name: "Shreyans" }, { age: 20});

// Delete
await User.deleteOne({ name: "Shreyans" });
```

- RESTFul API
  - A **RESTful API** (Representational State Transfer) is a way to structure and access data over the web using **HTTP methods**.
  - Each **endpoint (URL)** represents a resource, and each HTTP method defines what to do with that resource.

HTTP Method	Operation	Example Endpoint	Description
POST	Create	/api/users	Add a new user
GET	Read	/api/users or /api/users/:id	Get all users or a specific one
PUT / PATCH	Update	/api/users/:id	Modify a specific user
DELETE	Delete	/api/users/:id	Remove a specific user

- **Example (Express.js):**

```
// RESTful Routes
app.post('/users', createUser);    // Create
app.get('/users', getUsers);       // Read all
app.get('/users/:id', getUser);    // Read one
app.put('/users/:id', updateUser); // Update
app.delete('/users/:id', deleteUser); // Delete
```

Methodology:

- Adopt a simple full-stack architecture: Express + MongoDB for the API and React + Vite for the client.
- Model data with Mongoose to enforce schema validations and provide convenient CRUD helpers.
- Expose RESTful endpoints under http://localhost:3000/api/drivers with clear resource semantics.
- Use environment variables (backend/.env) for configuration, including MONGODB_URI and PORT .
- Enable CORS for local development so the React app (http://localhost:5173) can call the API.
- Seed baseline data to make the app immediately testable and to validate the end-to-end flow.
- Keep the UI straightforward, focusing on filterable listings and a single form for create/edit.
- Centralize client–server communication in a small API service layer for maintainability.

Implementation Details:

Backend

backend/server.js
- Initializes Express with cors , morgan , and express.json .
- Reads PORT and MONGODB_URI from .env .
- Connects to MongoDB Atlas via Mongoose and starts the server.
- Mounts driver routes at /api/drivers .
- CORS allows http://localhost:5173 with methods GET, POST, PATCH, DELETE .

backend/models/Driver.js
- Driver schema fields: name , number , team , nationality , wins , podiums , points , debutYear , active .
- Validations and defaults; timestamps enabled.

backend/controllers/driverController.js
- createDriver(req, res) creates a driver.
- getDrivers(req, res) supports filters by team , active , name .
- updateDriver(req, res) applies partial updates via findByIdAndUpdate .
- deleteDriver(req, res) removes a driver by id.

backend/routes/driverRoutes.js
- Maps POST / , GET / , PATCH /:id , DELETE /:id to controller functions.

backend/scripts/seed.js
- Connects to DB using MONGODB_URI .
- Inserts initial set of drivers (e.g., Verstappen, Hamilton, Norris) only if collection is empty.

- Clean shutdown of the connection.

Frontend
frontend/src/services/api.js
- API_URL = 'http://localhost:3000/api' ; all calls under /drivers .
- Functions: fetchDrivers(filters) , createDriver(data) , updateDriver(id, data) , deleteDriver(id) .

frontend/src/components/DriverList.jsx
- Fetches drivers with filters: name , team , active .
- Displays a table with key fields and status indicator.
- Provides actions: Edit and Delete with styled buttons and icons.
- Loading and error feedback.

frontend/src/components/DriverForm.jsx
- Handles create/edit in one form; validation for required fields and debut year.
- onSave triggers refresh in parent; onCancel hides the form.
- Submits via createDriver or updateDriver .

frontend/src/App.jsx
- Top-level state for showing the form and tracking which driver to edit.
- "Add New Driver" button toggles the form; integrates list and form.
- Basic header/footer, container layout.

frontend/src/App.css
- F1-themed palette, improved layout, responsive styles.
- Styled filters, table, action buttons, form, loading spinner, and error messages.

Steps for execution:

Task 1: Backend API (Express + MongoDB)

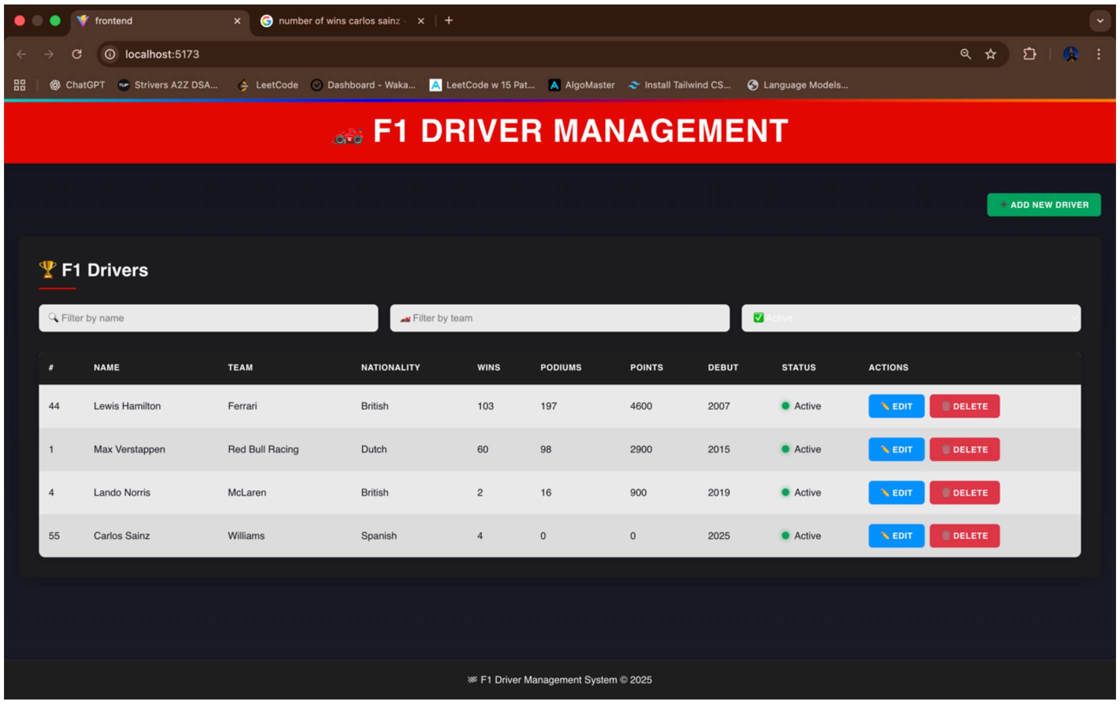
Task 2: Database Setup and Seeding

Task 3: Frontend UI (React + Vite)

- Start the client
  - This launches frontend at http://localhost:5173 and backend at http://localhost:3000 .

- Use the app
  - Open http://localhost:5173 .
  - Filter drivers by name, team, and status.
  - Add a new driver with the "Add New Driver" button.

- Edit an existing driver by clicking “Edit”.
- Delete a driver via “Delete” and confirm



F1 DRIVER MANAGEMENT

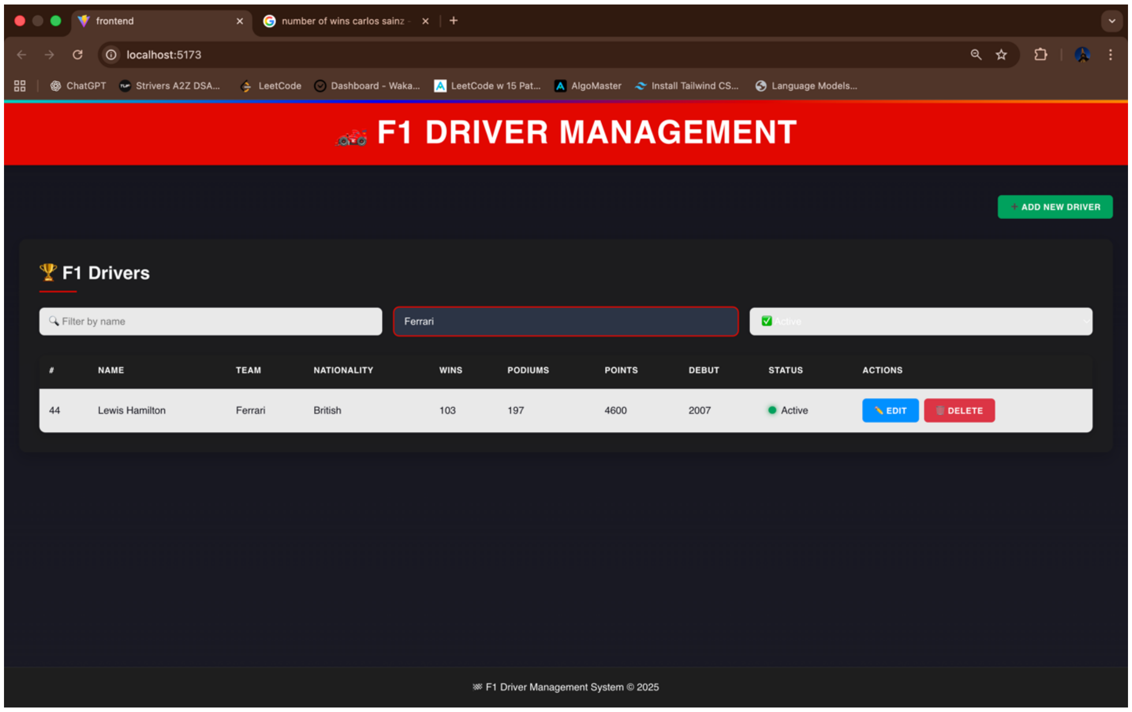
ADD NEW DRIVER

F1 Drivers

Filter by name | Filter by team | Active

#	NAME	TEAM	NATIONALITY	WINS	PODIUMS	POINTS	DEBUT	STATUS	ACTIONS
44	Lewis Hamilton	Ferrari	British	103	197	4600	2007	● Active	EDIT DELETE
1	Max Verstappen	Red Bull Racing	Dutch	60	98	2900	2015	● Active	EDIT DELETE
4	Lando Norris	McLaren	British	2	16	900	2019	● Active	EDIT DELETE
55	Carlos Sainz	Williams	Spanish	4	0	0	2025	● Active	EDIT DELETE

F1 Driver Management System © 2025



F1 DRIVER MANAGEMENT

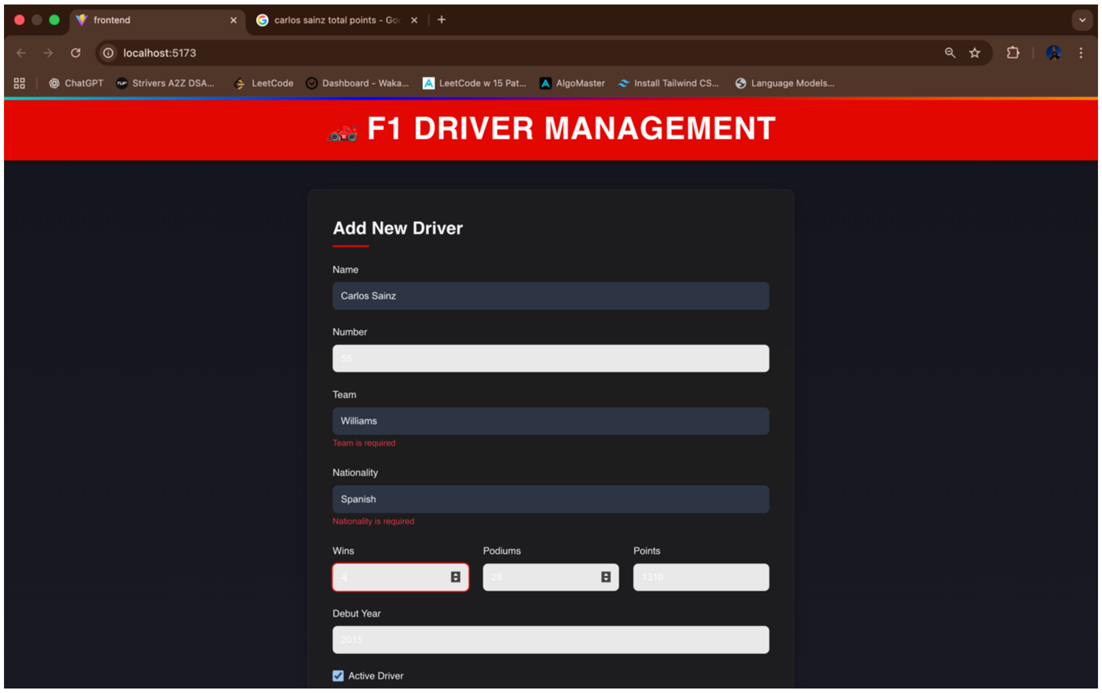
ADD NEW DRIVER

F1 Drivers

Filter by name | Ferrari | Active

#	NAME	TEAM	NATIONALITY	WINS	PODIUMS	POINTS	DEBUT	STATUS	ACTIONS
44	Lewis Hamilton	Ferrari	British	103	197	4600	2007	● Active	EDIT DELETE

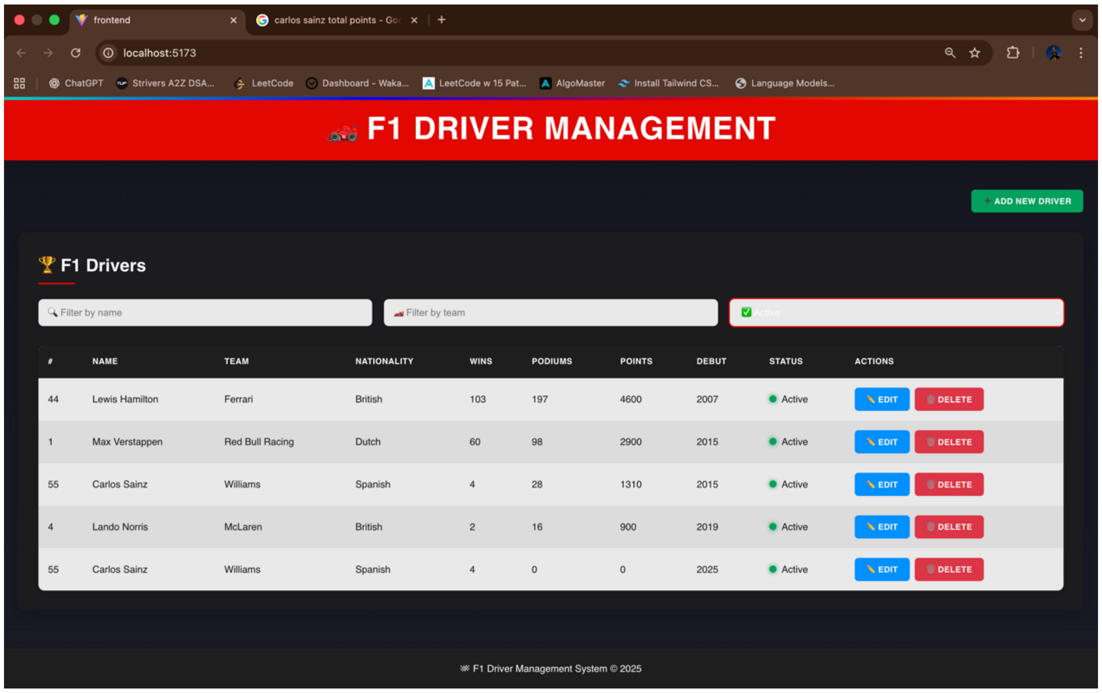
F1 Driver Management System © 2025



F1 DRIVER MANAGEMENT

Add New Driver

Name	Carlos Sainz
Number	55
Team	Williams
	Team is required
Nationality	Spanish
	Nationality is required
Wins	4
Podiums	28
Points	1310
Debut Year	2015
Active Driver	✓



F1 DRIVER MANAGEMENT

🏆 F1 Drivers

#	NAME	TEAM	NATIONALITY	WINS	PODIUMS	POINTS	DEBUT	STATUS	ACTIONS
44	Lewis Hamilton	Ferrari	British	103	197	4600	2007	● Active	EDIT DELETE
1	Max Verstappen	Red Bull Racing	Dutch	60	98	2900	2015	● Active	EDIT DELETE
55	Carlos Sainz	Williams	Spanish	4	28	1310	2015	● Active	EDIT DELETE
4	Lando Norris	McLaren	British	2	16	900	2019	● Active	EDIT DELETE
55	Carlos Sainz	Williams	Spanish	4	0	0	2025	● Active	EDIT DELETE

F1 Driver Management System © 2025

Conclusion:

The above experiment demonstrates data base modelling , and performance of CRUD operations , and setting up RESTful APIs and testing of the API endpoints.