

Step 1

```
JS Index1.js U ●
Week 3 > JS Index1.js > ...
10  async function connectToMongoDB() {
11    } catch (err) {
12      console.error("Error connecting to MongoDB:", err);
13    }
14  }
15
16  connectToMongoDB();
17
18  app.listen(port, () => {
19    console.log(`Server running on port ${port}`);
20  });
21
22  app.get('/rides', async (req, res) => {
23    try {
24      const rides = await db.collection('rides').find().toArray();
25
26      res.json(rides);
27    } catch (err) {
28      console.error("Error fetching rides:", err);
29    }
30  });
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Nickson Goh\Desktop\Cloud System\Cloud\berr2243-2025gmc\Week 3> node index1.js
Server running on port 3000
Connected to MongoDB!
PS C:\Users\Nickson Goh\Desktop\Cloud System\Cloud\berr2243-2025gmc\Week 3>
```

POST

POST http://localhost:3000/rides

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 [
2   ...
3     "pickupLocation": "Central Park",
4     "destination": "Times Square",
5     "driverId": "DRIVER123",
6     "status": "requested"
7 ]
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview

```
1 [{"id": "690c19c911ca62384a3abc5b"}]
```

POST (Output)

berr2243-2025gmc > testDB > rides

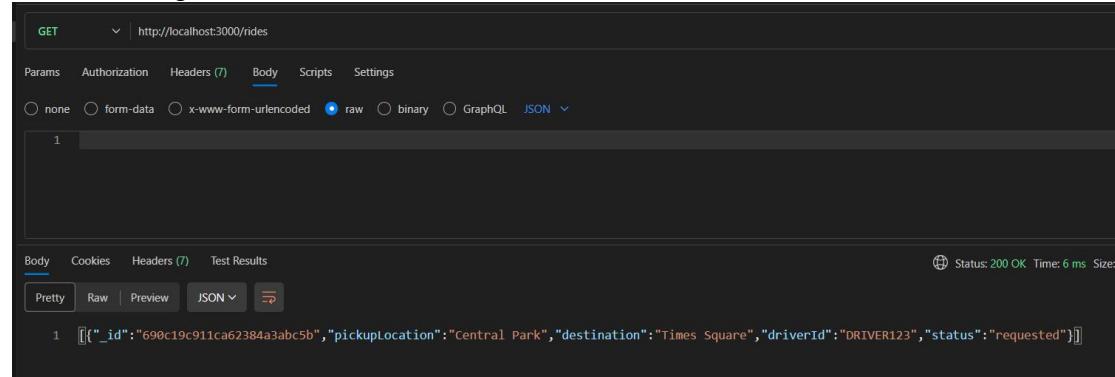
Documents 1 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#) ↗

[+ ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#)

```
_id: ObjectId('690c19c911ca62384a3abc5b')
pickupLocation : "Central Park"
destination : "Times Square"
driverId : "DRIVER123"
status : "requested"
```

GET From MongoDB



GET | http://localhost:3000/rides

Params Authorization Headers (7) Body Scripts Settings

(none form-data x-www-form-urlencoded raw binary GraphQL JSON)

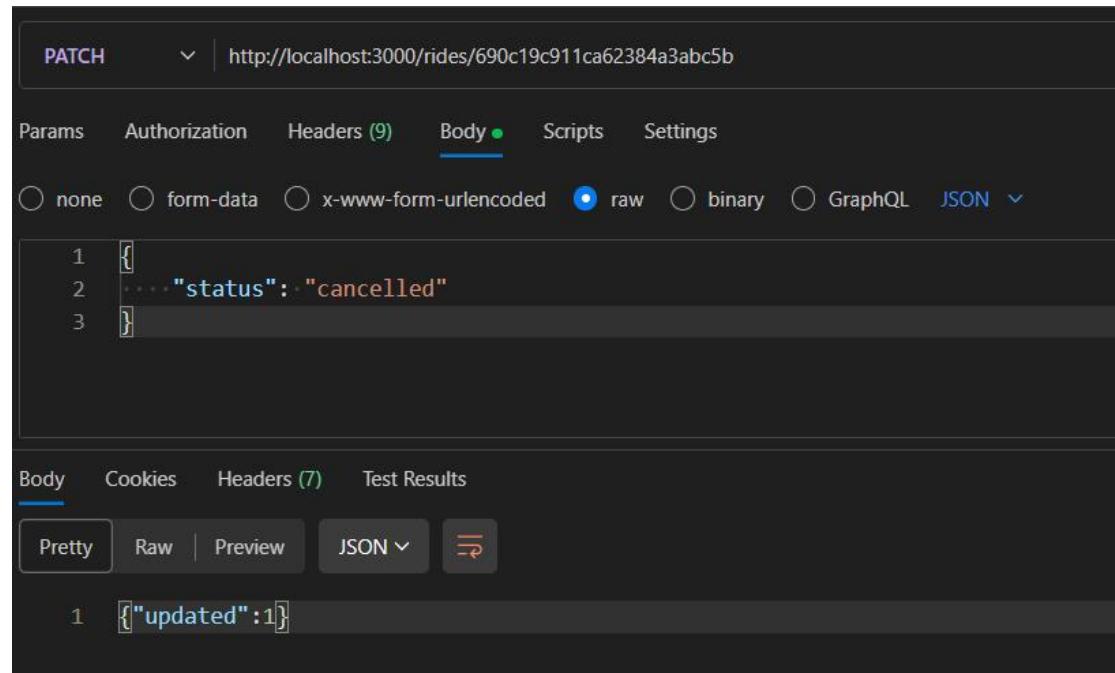
1

Body Cookies Headers (7) Test Results

Pretty Raw Preview JSON

1 [{"_id": "690c19c911ca62384a3abc5b", "pickupLocation": "Central Park", "destination": "Times Square", "driverId": "DRIVER123", "status": "requested"}]

PATCH



PATCH | http://localhost:3000/rides/690c19c911ca62384a3abc5b

Params Authorization Headers (9) Body Scripts Settings

(none form-data x-www-form-urlencoded raw binary GraphQL JSON)

1 {
2 "status": "cancelled"
3 }

Body Cookies Headers (7) Test Results

Pretty Raw Preview JSON

1 {"updated": 1}

berr2243-2025gmc > testDB > rides

Documents 1 Aggregations Schema Indexes 1 Validation

⌚ ▾ Type a query: { field: 'value' } or [Generate query](#) ↗

[+ ADD DATA](#)

[EXPORT DATA](#)

[UPDATE](#)

[DELETE](#)

```
_id: ObjectId('690c19c911ca62384a3abc5b')
pickupLocation : "Central Park"
destination : "Times Square"
driverId : "DRIVER123"
status : "cancelled"
```

DELETE

DELETE http://localhost:3000/rides/690c19c911ca62384a3abc5b

Params Authorization Headers (7) **Body** Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL [JSON](#)

```
1
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview [JSON](#)

```
1  [{"deleted":1}]
```

berr2243-2025gmc > testDB > rides

Documents 0 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

25 0 - 0 of 0



This collection has no data

It only takes a few seconds to import data from a JSON or CSV file.

Lab Questions

Answer by testing your API in Postman and observing responses.

1. POST Request:

- o What HTTP status code is returned when a ride is created successfully?

=**201 Created**

- o What is the structure of the response body?

=**json({ id: result.insertedId })**

2. GET Request:

- o What happens if the rides collection is empty?

=**empty array ([])**

- o What data type is returned in the response (array/object)?

=**array**

3. Fix PATCH and DELETE Error:

- o Catch the error when requesting PATCH or DELETE API, then try to fix the issue reported.

=**error (400 bad request)**

To fix, add the ObjectId.

```
const { MongoClient, ObjectId } = require('mongodb');
```

- o If you try to update a non-existent ride ID, what status code is returned?

=**Invalid ride Id or data**

- o What is the value of updated in the response if the update succeeds?

=**{"updated":1}**

- o How does the API differentiate between a successful deletion and a failed one?

=**if successful {"deleted":1} and status 200 OK if unsuccessful {"error":"Ride not found"} and status 404 not found.**

4. Users Endpoints:

- o Based on the exercise above, create the endpoints to handle the CRUD operations for users account

POST

berr2243-2025gmc > testDB > users

Documents 45 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#) ⚡

[+ ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#)

```
_id: ObjectId('690c26d3064373683d006f3e')
pickupLocation : "Central Park"
destination : "Times Square"
userId : "User123"
status : "requested"
```

GET

GET http://localhost:3000/users

Params Authorization Headers (7) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

Body Cookies Headers (7) Test Results

Pretty Raw Preview JSON

1 [{"_id": "690c26d3064373683d006f3e", "pickupLocation": "Central Park", "destination": "Times Square", "userId": "User123", "status": "requested"}]]

PATCH

HTTP <http://localhost:3000/users/690c26d3064373683d006f3e>

PATCH [▼](#) | http://localhost:3000/users/690c26d3064373683d006f3e

Params Authorization Headers (9) **Body** Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL [JSON](#) [▼](#)

```
1 [ {  
2   .... "status": "Cancelled"  
3 } ]
```

Body Cookies Headers (7) Test Results

[Pretty](#) [Raw](#) | [Preview](#) [JSON](#) [▼](#)

```
1 [ {"updated": 1} ]
```

berr2243-2025gmc > testDB > users

Documents 1 Aggregations Schema Indexes 1 Validation

[🕒](#) Type a query: { field: 'value' } or [Generate query](#)

[+ ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#)

```
_id: ObjectId('690c26d3064373683d006f3e')
pickupLocation: "Central Park"
destination: "Times Square"
userId: "User123"
status: "Cancelled"
```

5. FrontEnd:

- o Upload the Postman JSON to any AI tools, and generate a simple HTML and JS Dashboard for you

The screenshot shows a web-based dashboard with the following structure:

- Rides**: A table with columns: ID, Pickup, Destination, Driver ID, Status, Actions. One row is shown: 6912201957e3cb9cb5f1fc55, Central Park, Times Square, DRIVER123, Cancelled, with 'Update' and 'Delete' buttons.
- Add Ride**: A form with fields: pickupLocation, destination, driverId, status, and an 'Add Ride' button.
- Users**: A table with columns: ID, Pickup, Destination, User ID, Status, Actions. Two rows are shown: 690c28d3064373683d006f0e, Central Park, Times Square, User123, Cancelled, and 691204c63b7b24b5faca97f3, Central Park, Times Square, user123, requested, with 'Update' and 'Delete' buttons.
- Add User**: A form with fields: pickupLocation, destination, userId, status, and an 'Add User' button.