

### Exercise 3

```
}  
PS C:\Users\Charan\Desktop\cloudss\Cloudsystemclass> node index.js  
Server running on port 3000  
Connected to MongoDB!  
PS C:\Users\Charan\Desktop\cloudss\Cloudsystemclass> 
```

HTTP PATCH http://localhost:3000/rides/6902e2944f63b308c0a67b36 Save No Environment

Send

Params Authorization Headers (9) Body Scripts Settings Code Cookie

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL J v E

```
1 {  
2   ... "status": "Unavailable"  
3 }
```

Body v Status: 200 OK Time: 7 ms Size: 247 B ...

Pretty Raw Preview JSON v

```
1 { "updated": 1 }
```

type a query (filter, values) or [construct query](#)

+ ADD DATA EXPORT DATA UPDATE DELETE 25 1-1

```
{  
  "_id": ObjectId("6902e2944f63b308c0a67b36"),  
  "pickupLocation": "Central Park",  
  "destination": "Times Square",  
  "driverID": "Driver23",  
  "status": "requested"  
}
```

HTTP

http://localhost:3000/rides/6902e2944f63b308c0a67b

Save

No Environment

DELETE

http://localhost:3000/rides/6902e2944f63b308c0a67b36

Send

Params

Authorization

Headers (9)

Body

Scripts

Settings

Code

Cookie

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

1 {

2 ...."status": "Unavailable"

3 }

Body

Status: 200 OK Time: 6 ms Size: 247 B

PrettyRawPreviewJSON

1 {"updated":1}



### This collection has no data

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

Import data

### Lab Questions:

1. What HTTP status code is returned when a ride is created successfully?  
**= 201 Inserted**
2. What is the structure of the response body?  
**= json({id: result.insertedId})**
3. What happens if the rides collection is empty?  
**= empty array ([])**
4. What data type is returned in the response (array/object)?  
**= array**
5. Catch the error when requesting PATCH or DELETE API, then try to fix the issue reported.  
**= error (400 bad request),**
6. If you try to update a non-existent ride ID, what status code is returned?  
**= Invalid ride ID or data**
7. What is the value of updated in the response if the update succeeds?  
**= updated "1"**
8. How does the API differentiate between a successful deletion and a failed one?  
**= If successful deletion {"deleted":1} and "200 Ok", If failed one {"ride not found":1} and "404 not found" will be displayed.**

9. Based on the exercise above, create the endpoints to handle the CRUD operations for users account

=

```
87 // user
88 app.get( '/user', async (req, res) => {
89   try {
90     const rides = await db.collection('user').find().toArray();
91     res.status(200).json(rides);
92   } catch (err) {
93     res.status(500).json({ error: "Failed to fetch user" });
94   }
95 });
96
97 // POST /rides - Create a new ride
98 app.post( '/user', async (req, res) => {
99   try {
100     const result = await db.collection('user').insertOne(req.body);
101     res.status(201).json({id: result.insertedId});
102   } catch (err) {
103     res.status(400).json({ error: "Invalid user data" });
104   }
105 });
106
107 // PATCH /rides/:id - Update ride status
108 app.patch( '/user/:id', async (req, res) => {
109   try {
110     const result = await db.collection('user').updateOne(
111       { _id: new ObjectId(req.params.id) },
112       { $set: { status: req.body.status } }
113     );
114
115     if (result.modifiedCount === 0) {
116       return res.status(404).json({ error: "user not found" });
117     }
118     res.status(200).json({ updated: result.modifiedCount });
119   } catch (err) {
120     // Handle invalid ID format or DB errors
121     res.status(400).json({ error: "Invalid user ID or data" });
122   }
123 });
124
125 // DELETE /rides/:id - Cancel a ride
126 app.delete( '/user/:id', async (req, res) => {
127   try {
128     const result = await db.collection('user').deleteOne(
129       { _id: new ObjectId(req.params.id) }
130     );
131
132     if (result.deletedCount === 0) {
133       return res.status(404).json({ error: "user not found" });
134     }
135     res.status(200).json({ deleted: result.deletedCount });
136   } catch (err) {
137     res.status(400).json({ error: "Invalid user ID or data" });
138   }
139 });
```

POST http://localhost:3000/user

Params Authorization Headers (9) Body Scripts Settings

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

```
1 {
2   ... "pickupLocation": "Central Park",
3   ... "destination": "Times Square",
4   ... "userId": "charan",
5   ... "password": "Charan123",
6   ... "status": "requested"
7 }
```

**ADD DATA** **EXPORT DATA** **UPDATE** **DELETE**

```
{
  "_id": ObjectId('690c2773f928630d8fdddba0'),
  "pickupLocation": "Central Park",
  "destination": "Times Square",
  "userId": "charan",
  "password": "Charan123",
  "status": "requested"
}
```

GET http://localhost:3000/user

Params Authorization Headers (7) Body Scripts Settings

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

Send Beautify

1

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 8 ms Size: 395 B

Pretty Raw Preview JSON

```
1 [{"_id":"690c2773f928630d8fdddba0","pickupLocation":"Central Park","destination":"Times Square","userId":"charan","password":"Charan123","status":"requested"}]
```

PATCH http://localhost:3000/user/690c2773f928630d8fdddba0

Params Authorization Headers (9) Body Scripts Settings

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

Send Beautify

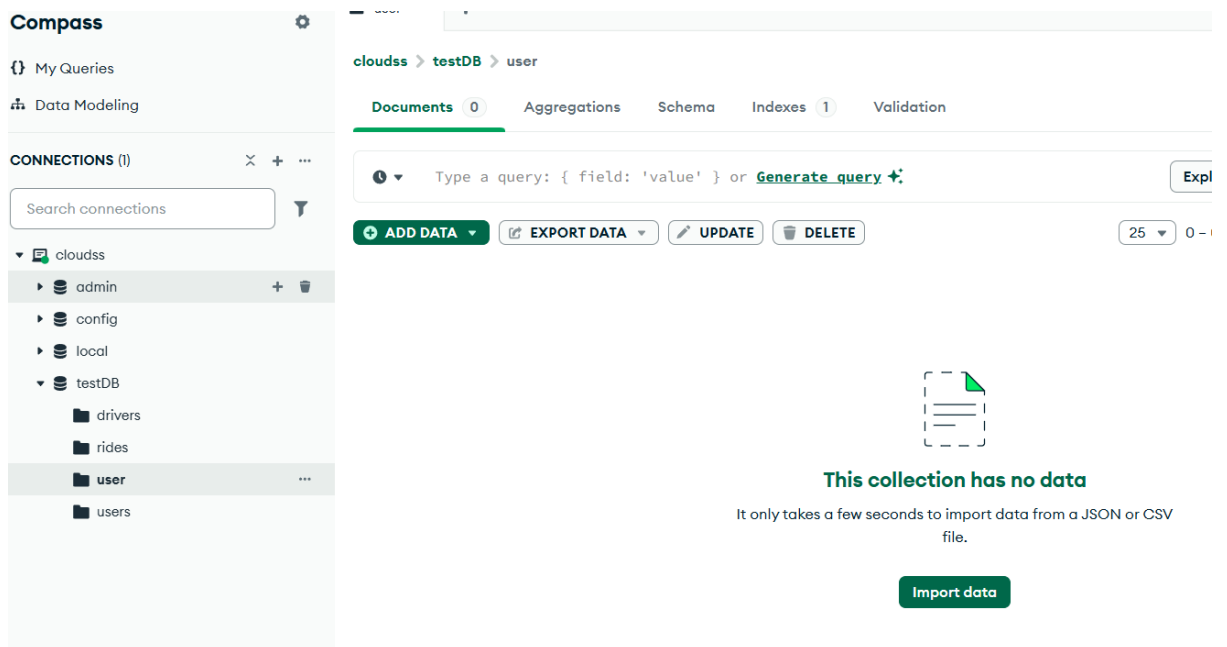
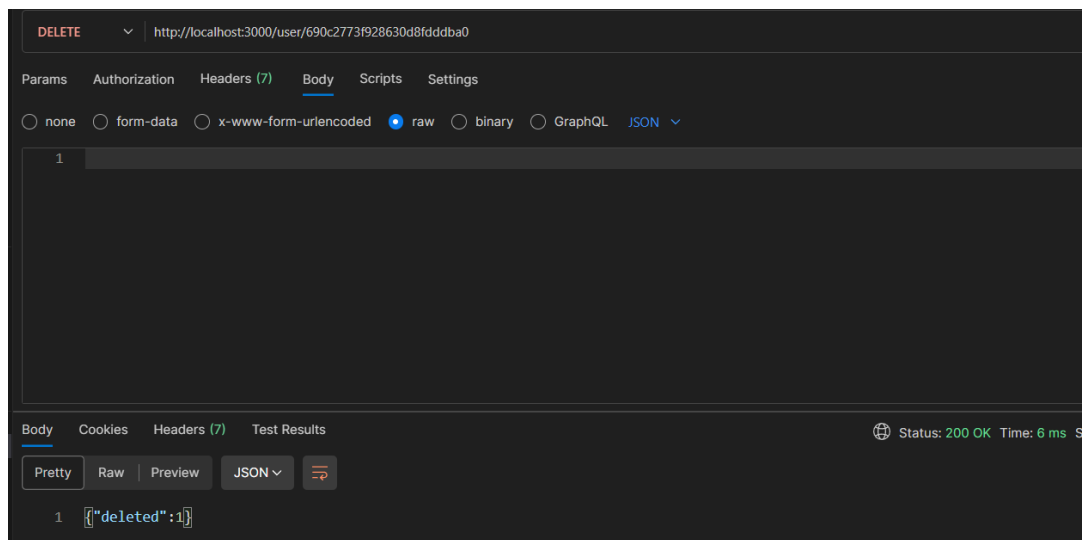
```
1 {
2   .. "status":"using"
3 }
```

Body Cookies Headers (7) Test Results

Status: 200 OK Time: 11 ms Size: 247 B

Pretty Raw Preview JSON

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



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

## App.js

```
Cloudsystemclass > dashboard > JS app.js > ...
1  // Dashboard app - generated from the project's routes in index.js
2  // Endpoints extracted from server code (routes in Cloudsystemclass/index.js)
3  const API_BASE = 'http://localhost:3000';
4
5  const endpoints = [
6    { method: 'GET', path: '/rides', description: 'Fetch all rides' },
7    { method: 'POST', path: '/rides', description: 'Create a new ride' },
8    { method: 'PATCH', path: '/rides/:id', description: 'Update ride status' },
9    { method: 'DELETE', path: '/rides/:id', description: 'Cancel a ride' },
10   { method: 'GET', path: '/user', description: 'Fetch all users' },
11   { method: 'POST', path: '/user', description: 'Create a new user' },
12   { method: 'PATCH', path: '/user/:id', description: 'Update user' },
13   { method: 'DELETE', path: '/user/:id', description: 'Delete user' }
14 ];
15
16 function $(sel) { return document.querySelector(sel); }
17
18 function renderEndpoints() {
19   const container = $('#endpoints');
20   container.innerHTML = '';
21
22   endpoints.forEach((ep, i) => {
23     const card = document.createElement('div');
24     card.className = 'card';
25
26     const title = document.createElement('h3');
27     title.textContent = `${ep.method} ${ep.path}`;
28     card.appendChild(title);
29
30     const desc = document.createElement('p');
31
32     desc.className = 'muted';
33     desc.textContent = ep.description || '';
34     card.appendChild(desc);
35
36     const controls = document.createElement('div');
37     controls.className = 'controls';
38
39     // ID input for routes with :id
40     let idInput = null;
41     if (ep.path.includes(':id')) {
42       idInput = document.createElement('input');
43       idInput.placeholder = 'id (for :id)';
44       idInput.className = 'small';
45       controls.appendChild(idInput);
46     }
47
48     // Body textarea for POST/PATCH
49     let bodyInput = null;
50     if (['POST', 'PATCH'].includes(ep.method)) {
51       bodyInput = document.createElement('textarea');
52       bodyInput.placeholder = 'JSON body (e.g. {"name": "Charan"})';
53       bodyInput.className = 'body';
54       controls.appendChild(bodyInput);
55     }
56
57     const sendBtn = document.createElement('button');
58     sendBtn.textContent = 'Send';
59     sendBtn.addEventListener('click', () => {
60       sendRequest(ep, idInput && idInput.value, bodyInput && bodyInput.value);
61     });
62     controls.appendChild(sendBtn);
63   });
64 }
```

```

59     sendRequest(ep, idInput && idInput.value, bodyInput && bodyInput.value);
60   });
61   controls.appendChild(sendBtn);
62
63   card.appendChild(controls);
64   container.appendChild(card);
65 });
66 }
67
68 async function sendRequest(ep, idValue, bodyText) {
69   let path = ep.path;
70   if (ep.path.includes(':id')) {
71     if (!idValue) return showResponse({ error: 'Please provide id for this endpoint' });
72     path = ep.path.replace(':id', encodeURIComponent(idValue));
73   }
74
75   const url = API_BASE + path;
76   const opts = { method: ep.method, headers: {} };
77
78   if (['POST', 'PATCH'].includes(ep.method) && bodyText) {
79     try {
80       opts.headers['Content-Type'] = 'application/json';
81       opts.body = JSON.stringify(JSON.parse(bodyText));
82     } catch (err) {
83       return showResponse({ error: 'Invalid JSON body' });
84     }
85   }
86
87   showResponse({ pending: true, url, opts: { ...opts, body: opts.body ? '<body>' : undefined } });
88
89   try {
90     const res = await fetch(url, opts);
91     const contentType = res.headers.get('content-type') || '';
92     let data;
93     if (contentType.includes('application/json')) {
94       data = await res.json();
95     } else {
96       data = await res.text();
97     }
98     showResponse({ status: res.status, headers: Object.fromEntries(res.headers.entries()), data });
99   } catch (err) {
100     showResponse({ error: String(err) });
101   }
102 }
103
104 function showResponse(obj) {
105   const pre = $('#responseContent');
106   pre.textContent = JSON.stringify(obj, null, 2);
107 }
108
109 renderEndpoints();
110
111 // Helpful hint if fetch is blocked by CORS
112 console.log('Dashboard loaded. If requests fail due to CORS, run the dashboard from a local HTTP server

```

## Index.html

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width,initial-scale=1" />
6   <title>API Dashboard</title>
7   <link rel="stylesheet" href="styles.css" />
8 </head>
9 <body>
10   <header>
11     <h1>Simple API Dashboard</h1>
12     <p class="muted">Calls the local API server (http://localhost:3000). Ensure the server is running and C
13   </header>
14
15   <main>
16     <section id="endpoints"></section>
17
18     <section id="response">
19       <h2>Response</h2>
20       <pre id="responseContent">No response yet.</pre>
21     </section>
22   </main>
23
24   <script src="app.js"></script>
25 </body>
26 </html>
27
```

## Styles.css

```
1 :root{--bg: #f6f8fa;--card: #fff;--accent: #0366d6;--muted:#666}
2 *{box-sizing:border-box;font-family:Segoe UI,Roboto,Helvetica,Arial,sans-serif}
3 body{margin:0;background:var(--bg);color:#111}
4 header{padding:18px 20px;background:var(--card);border-bottom:1px solid #e6e6e6}
5 header h1{margin:0;font-size:20px}
6 .muted{color:var(--muted);margin:6px 0}
7 main{padding:20px;display:flex;gap:20px}
8 #endpoints{flex:1;display:grid;grid-template-columns:repeat(auto-fit,minmax(260px,1fr));gap:12px}
9 .card{background:var(--card);padding:12px;border-radius:8px;box-shadow:0 1px 2px rgba(0,0,0,0.03)}
10 .card h3{margin:0 0 6px 0;font-size:15px}
11 .controls{display:flex;flex-direction:column;gap:8px;margin-top:8px}
12 .controls .small{width:100%;padding:6px}
13 .controls .body{min-height:80px;padding:8px;font-family:monospace}
14 button{background:var(--accent);color:#fff;border:none;padding:8px 10px;border-radius:6px;cursor:pointer}
15 button:hover{opacity:.95}
16 #response{width:420px;max-width:40%;min-width:320px}
17 #response pre{background:#0b1220;color:#fff;padding:12px;border-radius:8px;overflow:auto;max-height:70v
18
19 @media (max-width:900px){main{flex-direction:column} #response{width:100%;max-width:100%}}
20
```

Result:

GET /rides

Fetch all rides

Send

DELETE /rides/id

Cancel a ride

id (for id)

Send

PATCH /user/id

Update user

id (for id)

JSON body (e.g. {"name":"Charan"})

POST /rides

Create a new ride

{  
 "name":"Charan"  
}

Send

GET /user

Fetch all users

Send

DELETE /user/id

Delete user

id (for id)

Send

PATCH /rides/id

Update ride status

id (for id)

JSON body (e.g. {"name":"Charan"})

Send

POST /user

Create a new user

JSON body (e.g. {"name":"Charan"})

Send

Response

```
{  
  "status": 200,  
  "headers": {  
    "connection": "keep-alive",  
    "content-length": "52",  
    "content-type": "application/json; charset=utf-8",  
    "date": "Wed, 12 Nov 2025 16:13:55 GMT",  
    "etag": "W/\"34-cWEGMdX1myRtvHNuHjkmolUVQug\"",  
    "keep-alive": "timeout=5",  
    "x-powered-by": "Express"  
  },  
  "data": {  
    {  
      "_id": "6914b21cc551181d6e4ee6d1",  
      "Name": "Charan"  
    }  
  }  
}
```

rides

+

cloudss > testDB > rides

Documents 1

Aggregations

Schema

Indexes 1

Validation

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

Explain

Reset

ADD DATA

EXPORT DATA

UPDATE

DELETE

25

1 - 1 of 1

↺

\_id: ObjectId('6914b21cc551181d6e4ee6d1')

Name : "Charan"