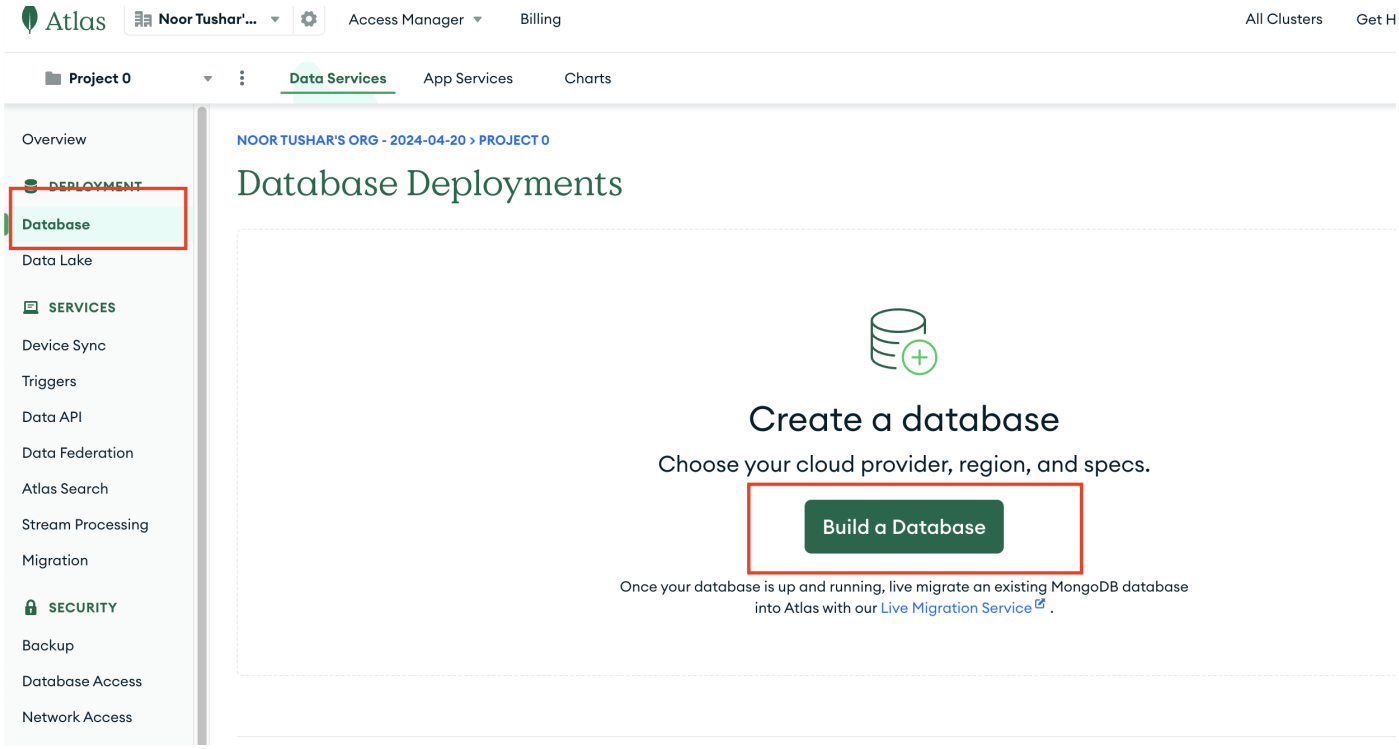
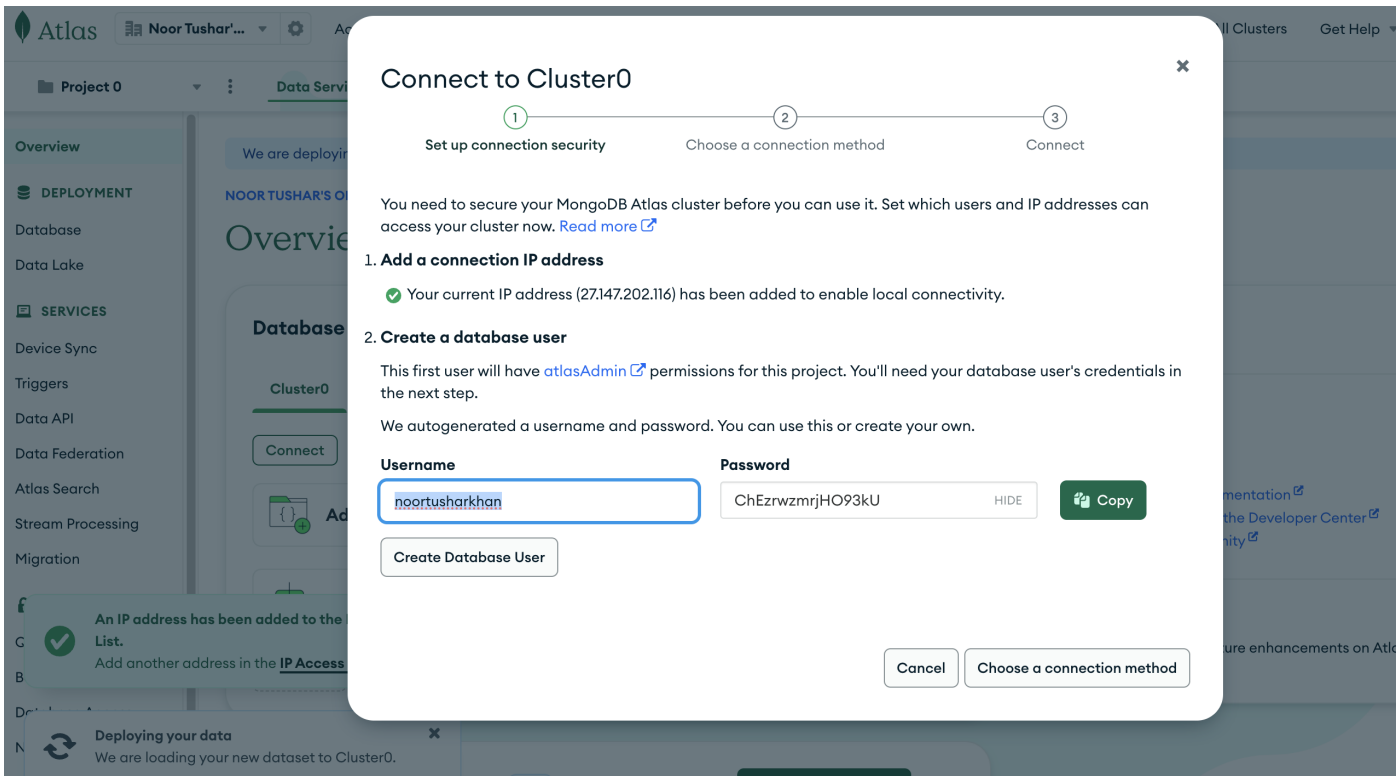
16-Simple-CRUD-Full-Version

1. Login - using google - noor.tushar.khan@gmail.com
2. 

3. 

4. username: noortusharkhan
   password: ChEzrwzmrjHO93kU
5. go back to terminal

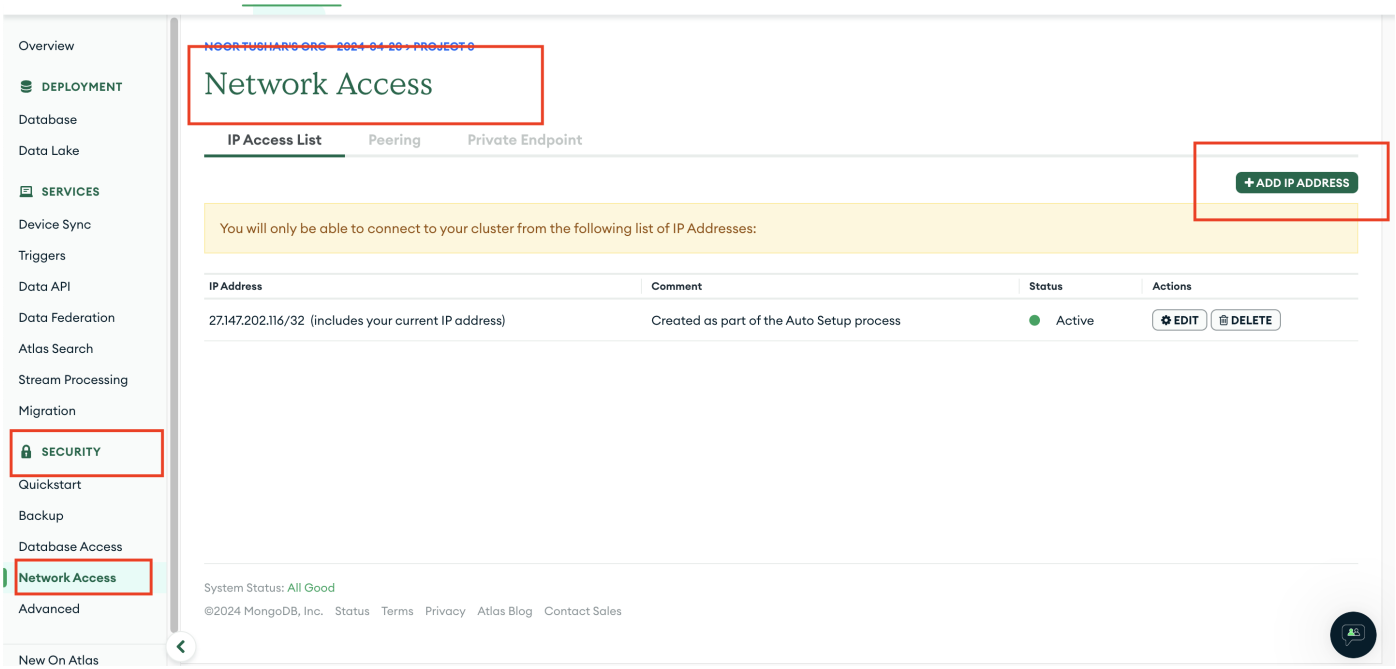
   create a server

   named example - `simple-crud-server`

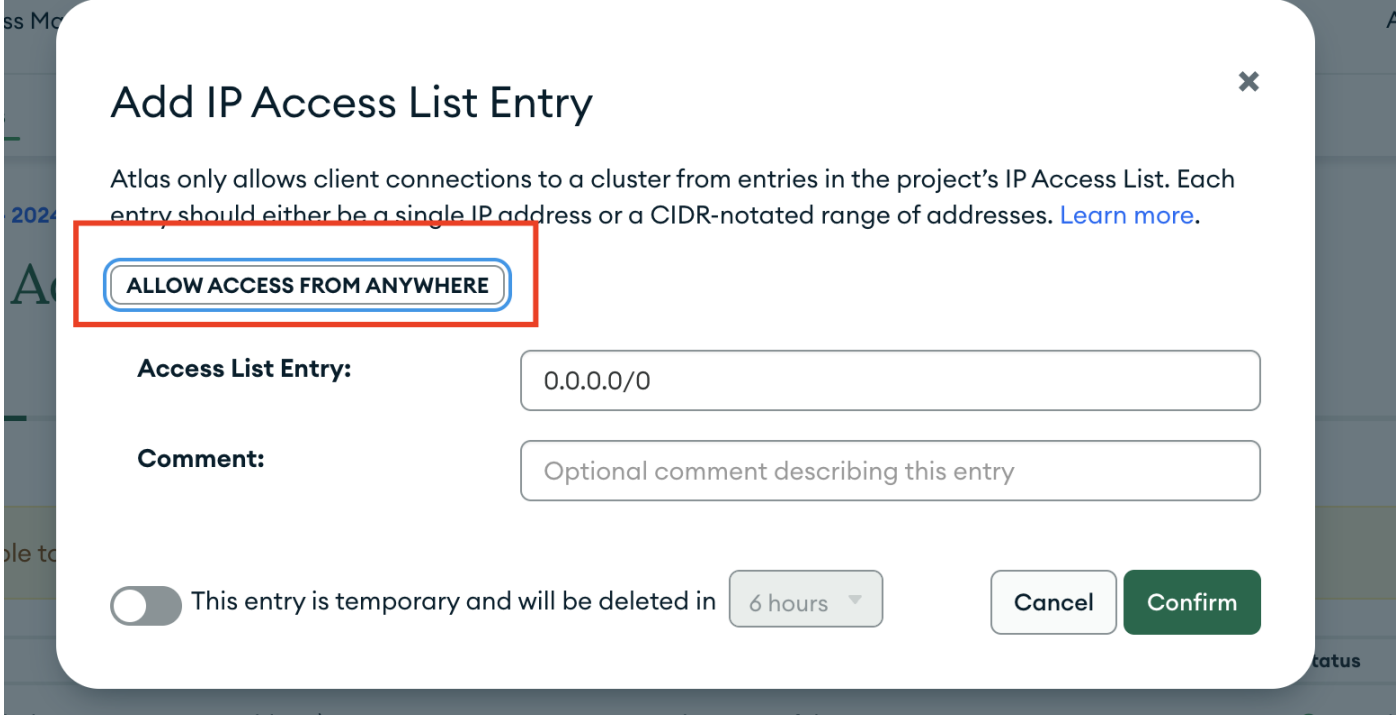
   ```
   npm init
   npm i express cors mongo
   ```

6. `package.json` -> `"scripts" : { "start": "node index.js" }`
7. `index.js` ->

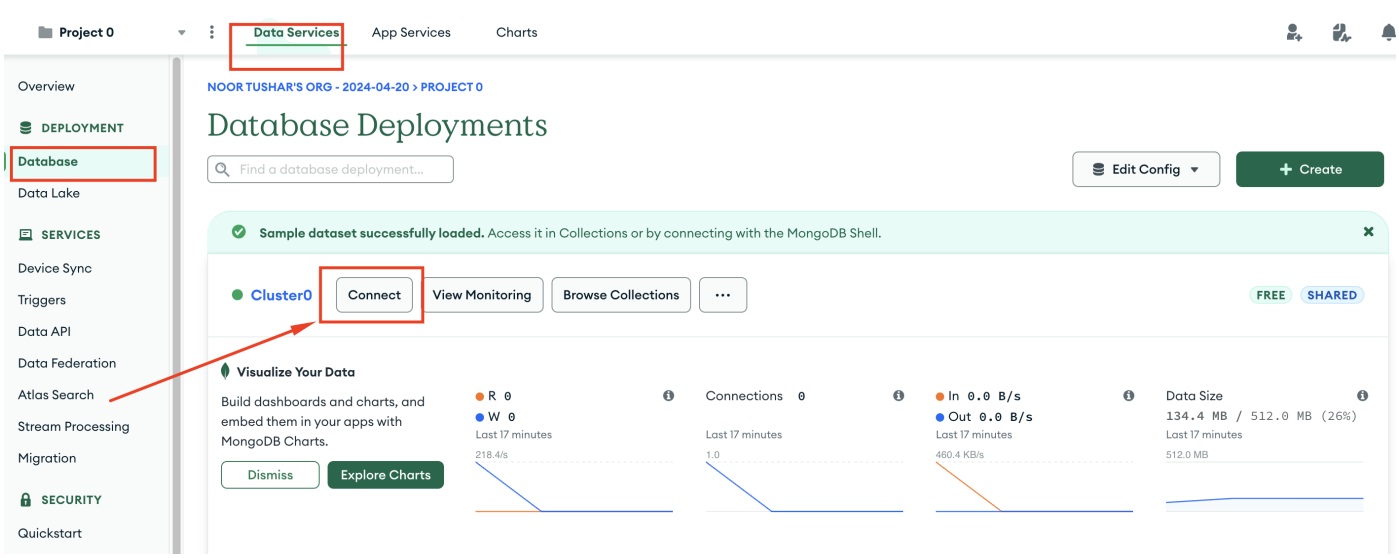
```
const express = require("express");
const cors = require("cors");

const app = express();

const port = process.env.PORT || 3000;

//middlewares
app.use(cors());
app.use(express.json());

//mongodb

//routes

app.get("/", (req, res) => {
   res.send("Simple Crud Server is working fine");
});

app.listen(port, () => {
   console.log(`app running at port: ${port}`);
});
```

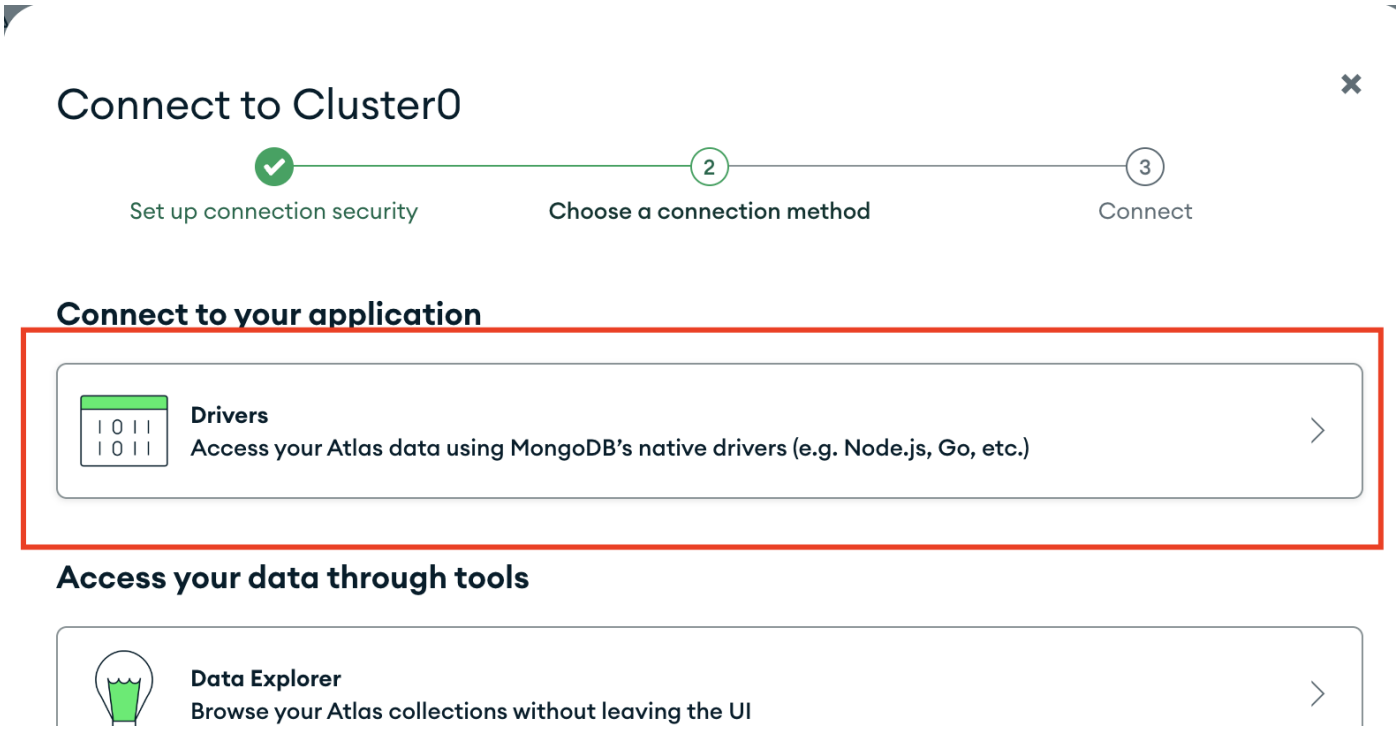


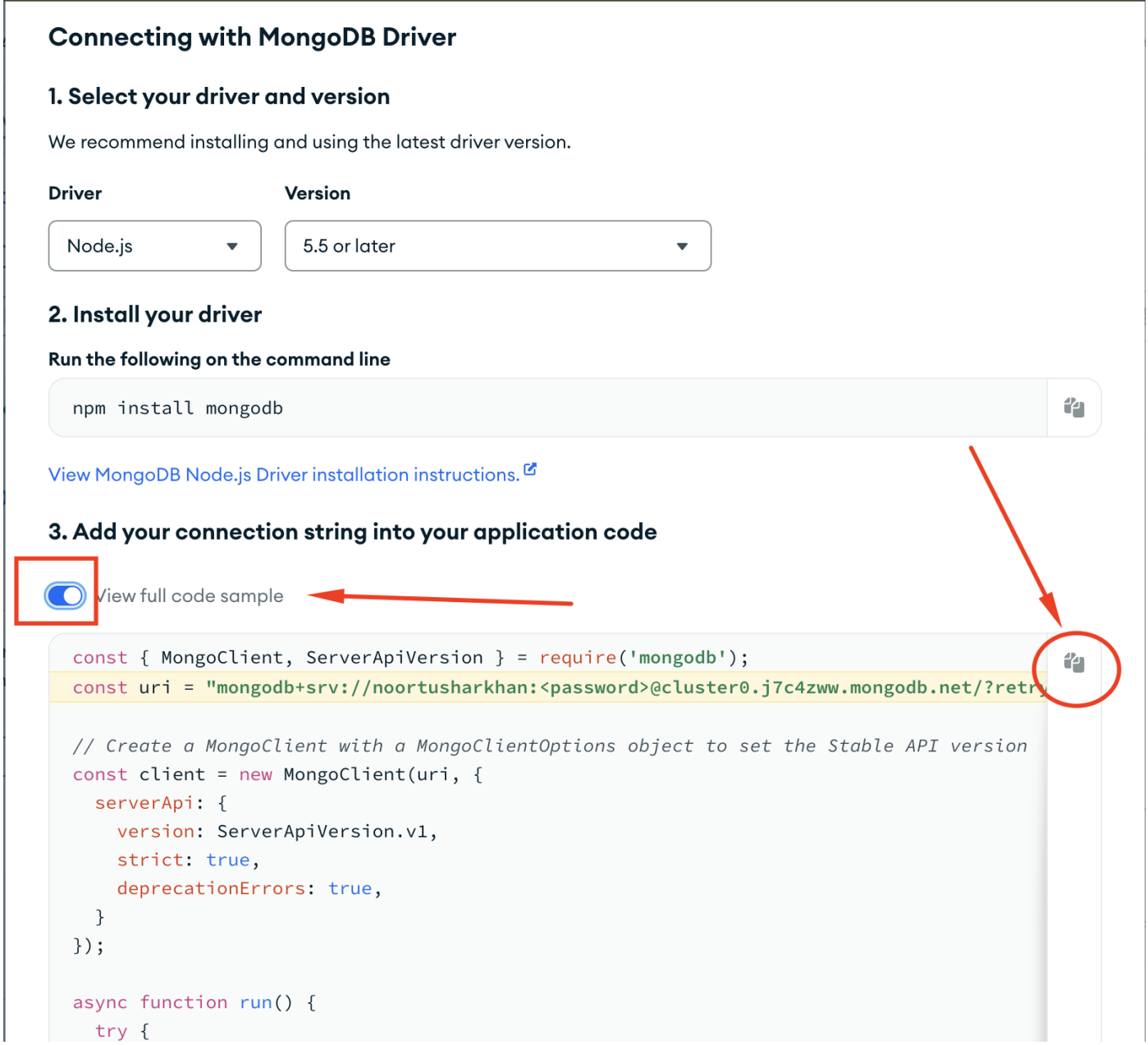
8.


Add IP Access List Entry
Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. Learn more.
ALLOW ACCESS FROM ANYWHERE
Access List Entry:
0.0.0.0/0
Comment:
Optional comment describing this entry
This entry is temporary and will be deleted in
6 hours
Cancel
Confirm

9.


Project 0
Data Services
App Services
Charts
Overview
DEPLOYMENT
Database
Data Lake
SERVICES
Device Sync
Triggers
Data API
Data Federation
Atlas Search
Stream Processing
Migration
SECURITY
Quickstart
NOOR TUSHAR'S ORG - 2024-04-20 > PROJECT 0
Database Deployments
Find a database deployment...
Edit Config
Create
Sample dataset successfully loaded. Access it in Collections or by connecting with the MongoDB Shell.
Cluster0
Connect
View Monitoring
Browse Collections
FREE
SHARED
Visualize Your Data
Build dashboards and charts, and embed them in your apps with MongoDB Charts.
Dismiss
Explore Charts
R 0
W 0
Last 17 minutes
Connections 0
In 0.0 B/s
Out 0.0 B/s
Data Size
134.4 MB / 512.0 MB (26%)


Connect to Cluster0
Set up connection security
2
Choose a connection method
3
Connect
Connect to your application
Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)
Access your data through tools
Data Explorer
Browse your Atlas collections without leaving the UI



10. Back to server index.js and paste it

also make sure to replace the password with the actual password

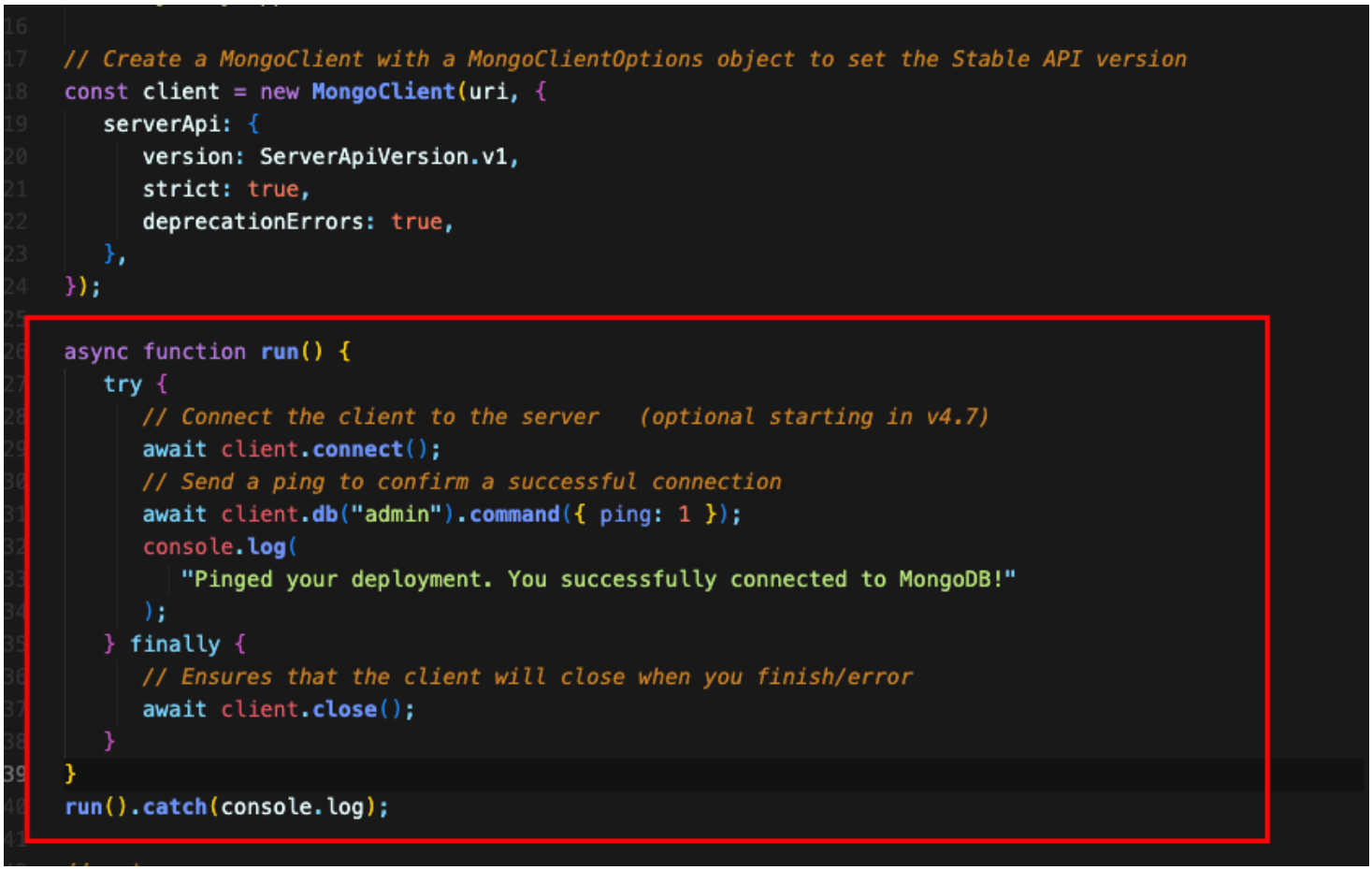
```
//mongodb

const { MongoClient, ServerApiVersion } = require("mongodb");
const uri =
   "mongodb+srv://noortusharkhan:ChEzrwzmrjHO93kU@cluster0.j7c4zww.mongodb.net/?retryWrites=true&
   w=majority&appName=Cluster0";

// Create a MongoClient with a MongoClientOptions object to set the Stable API version
const client = new MongoClient(uri, {
   serverApi: {
      version: ServerApiVersion.v1,
      strict: true,
      deprecationErrors: true,
   },
});

async function run() {
   try {
      // Connect the client to the server   (optional starting in v4.7)
      await client.connect();
      // Send a ping to confirm a successful connection
      await client.db("admin").command({ ping: 1 });
      console.log(
         "Pinged your deployment. You successfully connected to MongoDB!"
      );
   } finally {
      // Ensures that the client will close when you finish/error
      await client.close();
   }
}
run().catch(console.dir);

//routes
```

Normally normal function and async await try catch finally er syntax ta ki?

```
// normal syntax of a function

function name() {
    //
}

name()

// asyn await function

function async func () {
    await //
}

func().catch(err=>console.log(err));

// try catch finally

try {
  // try use korle 'catch' ba 'finally' use korte hobe
  // jeita korar dorkar chilo oita korbe
} catch (error) {
  // korte jeye error face korle ai block e dhora khabe
} finally {
 // jeita korar chilo oita hok ba na hok error khak ba na khak
```

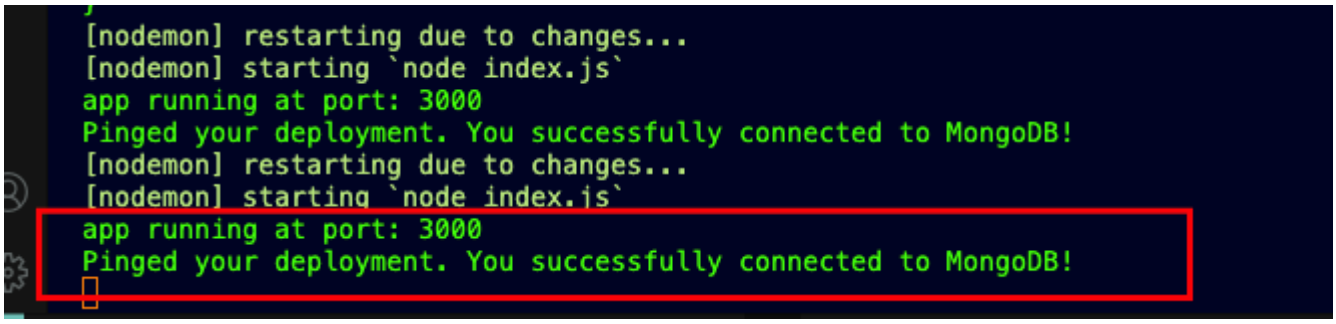
```
  // finally ai block ta execute hobe
}
```

so similarly our mongo db code looks like this: an async function



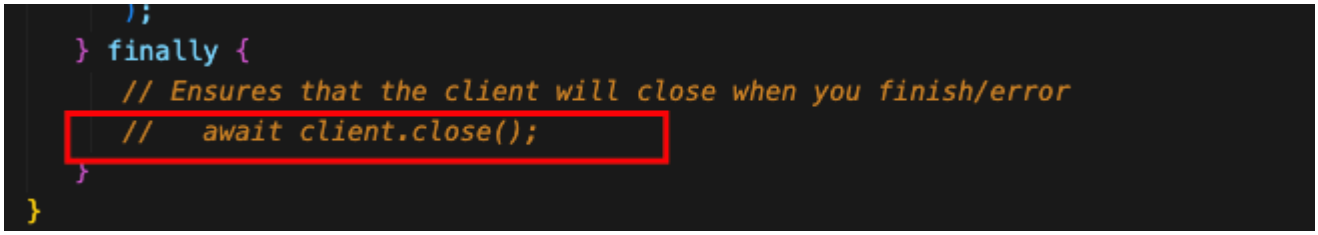
```
// Create a MongoClient with a MongoClientOptions object to set the Stable API version
const client = new MongoClient(uri, {
   serverApi: {
      version: ServerApiVersion.v1,
      strict: true,
      deprecationErrors: true,
   },
});

async function run() {
   try {
      // Connect the client to the server   (optional starting in v4.7)
      await client.connect();
      // Send a ping to confirm a successful connection
      await client.db("admin").command({ ping: 1 });
      console.log(
         "Pinged your deployment. You successfully connected to MongoDB!"
      );
   } finally {
      // Ensures that the client will close when you finish/error
      await client.close();
   }
}
run().catch(console.log);
```

we can see succsefully connected



```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
app running at port: 3000
Pinged your deployment. You successfully connected to MongoDB!
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
app running at port: 3000
Pinged your deployment. You successfully connected to MongoDB!
```

but we want to have the connection between our server and database not to end and thus will comment out `await client.close( )`



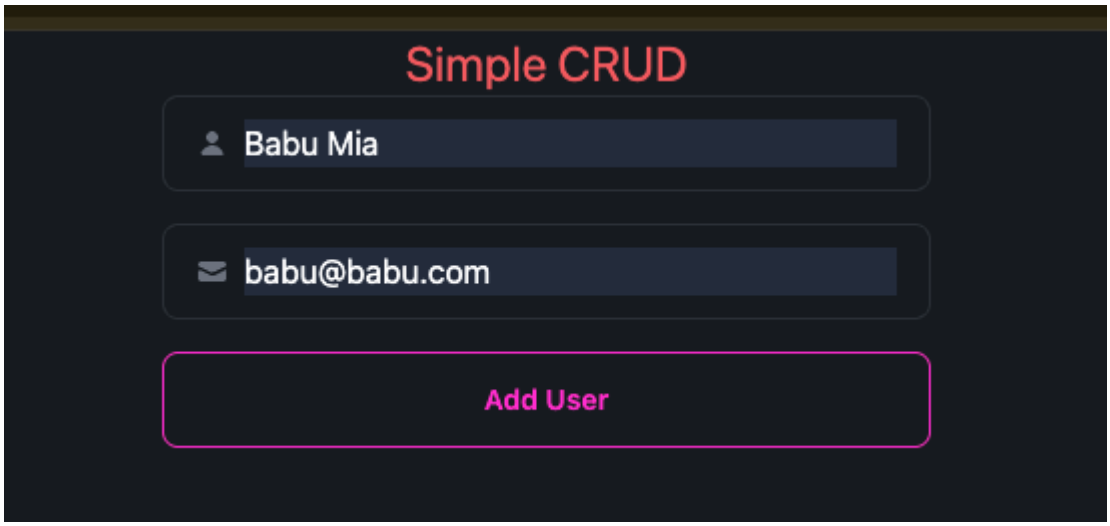
```
   } finally {
      // Ensures that the client will close when you finish/error
      //    await client.close();
   }
}
```

CLIENT SIDE - lets create one

```
simple-crud-client
```

CLIENT SIDE - lets create one

```
simple-crud-client
```



Now we have to send our field data to the backend server then database.

So first we have to create an API:

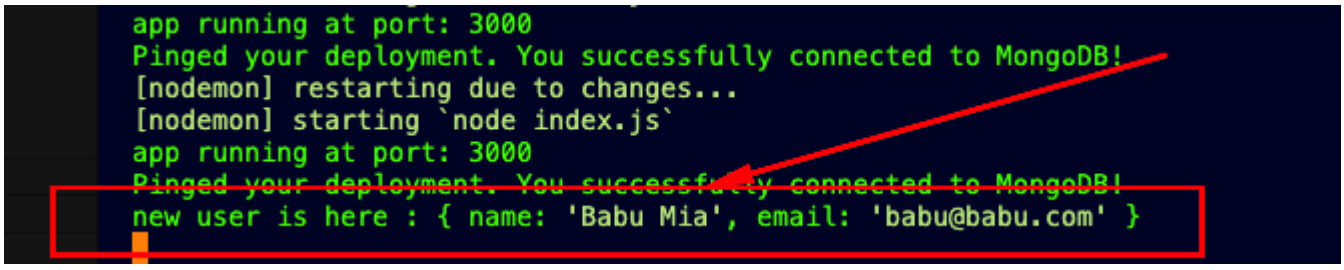
so akhon kothae korbo api create ta? mogodb r function tar bhitore:

```
async function run() {
  try {
    // Connect the client to the server   (optional starting in v4.7)
    await client.connect();

    // POST API
    app.post("/users", async (req, res) => {
      const user = req.body;
      console.log("new user is here :", user);
    });

    // Send a ping to confirm a successful connection
    await client.db("admin").command({ ping: 1 });
    console.log(
      "Pinged your deployment. You successfully connected to MongoDB!"
```

CLIENT side e jai akhon:

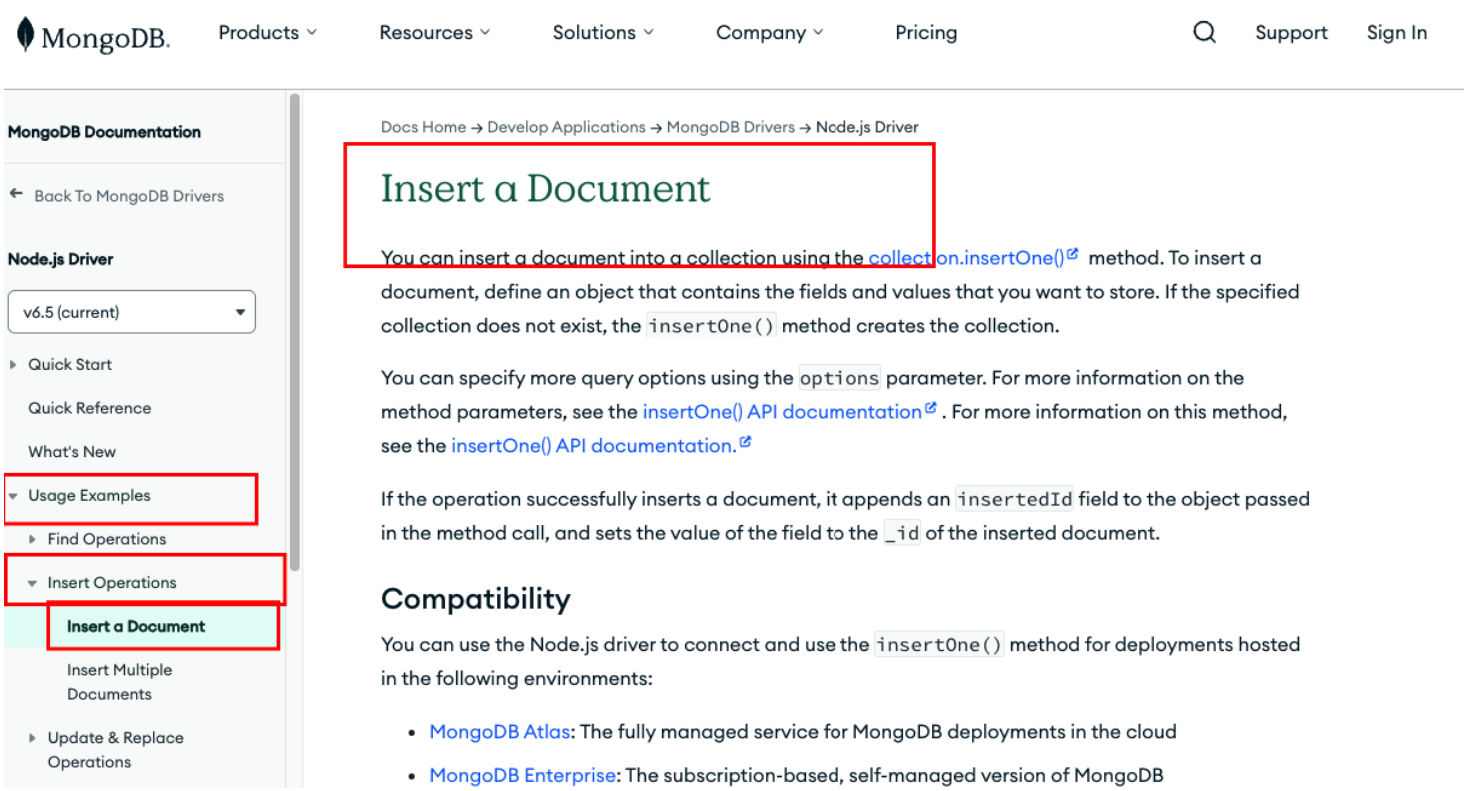
```
const Home = () => {
    const handleAddUser = (event) => {
        event.preventDefault();

        const form = event.target;
        const name = form.name.value;
        const email = form.email.value;

        const user = { name, email };
        console.log(user);

        fetch("http://localhost:3000/users", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify(user),
        })
            .then((res) => res.json())
            .then((data) => console.log(data))
            .catch((err) => console.log(err));
    };
```

Add user akhon Client side e click hoile => SERVER side er console e:



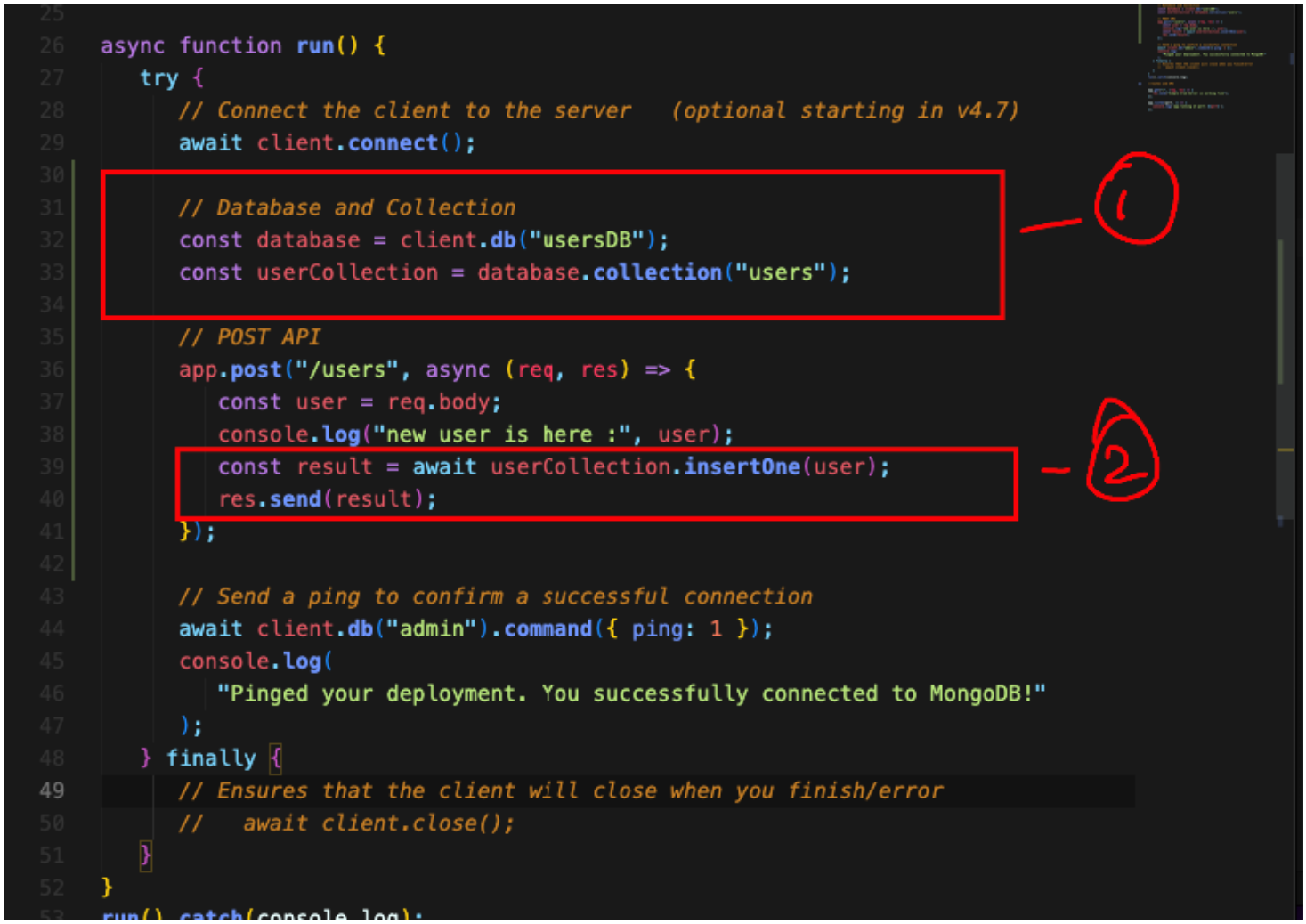
Akhon kaaj hoilo server thike database e pathabo.. EXCITED !!!

https://www.mongodb.com/docs/drivers/node/current/fundamentals/crud/

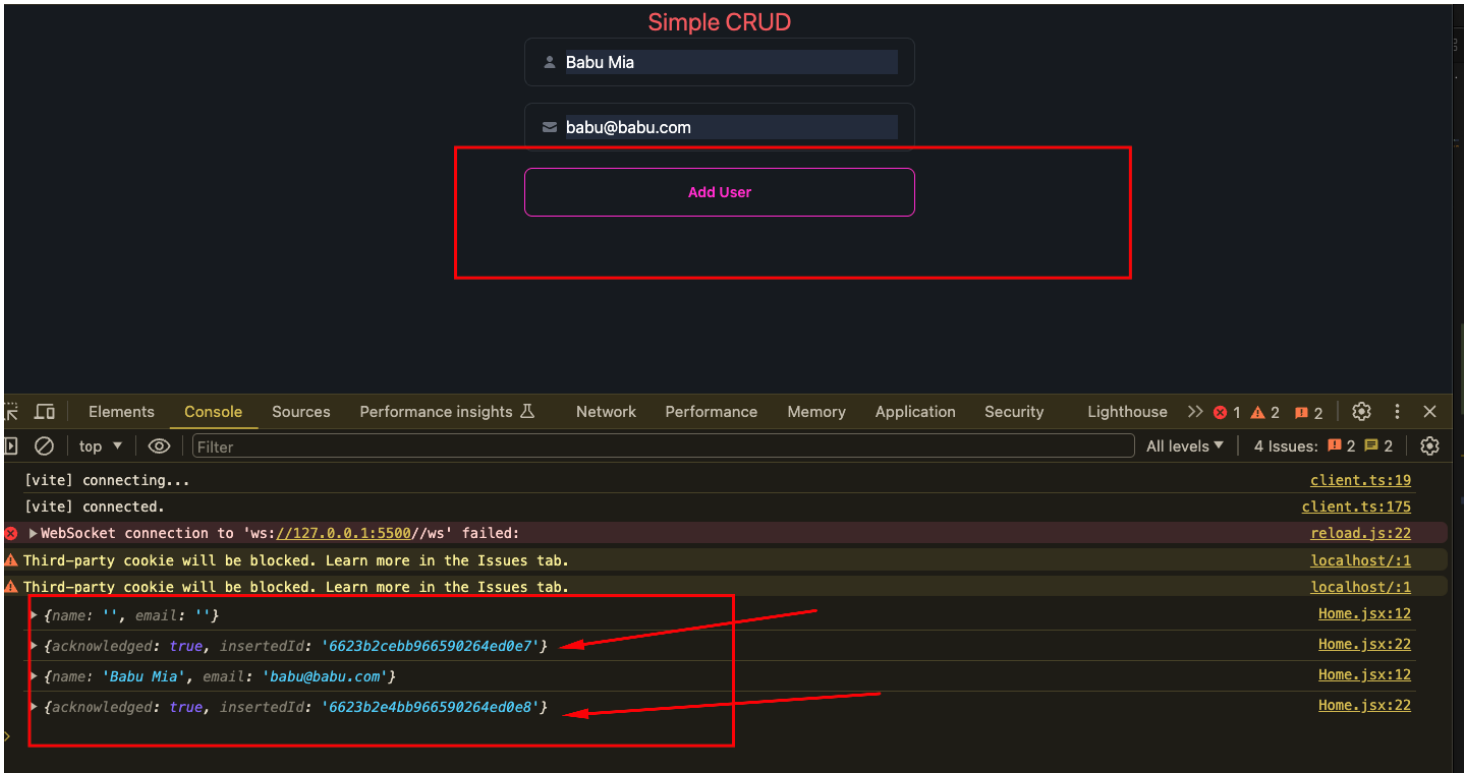


```
import { MongoClient } from "mongodb";

// Replace the uri string with your MongoDB deployment's connection string.
const uri = "<connection string uri>";

// Create a new client and connect to MongoDB
const client = new MongoClient(uri);

async function run() {
  try {
    // Connect to the "insertDB" database and access its "haiku" collection
    const database = client.db("insertDB");
    const haiku = database.collection("haiku");

    // Create a document to insert
    const doc = {
      title: "Record of a Shriveled Datum",
      content: "No bytes, no problem. Just insert a document, in MongoDB",
    }
    // Insert the defined document into the "haiku" collection
    const result = await haiku.insertOne(doc);

    // Print the ID of the inserted document
    console.log(`A document was inserted with the _id: ${result.insertedId}`)
  } finally {
     // Close the MongoDB client connection
    await client.close();
  }
}
// Run the function and handle any errors
run().catch(console.dir);
```

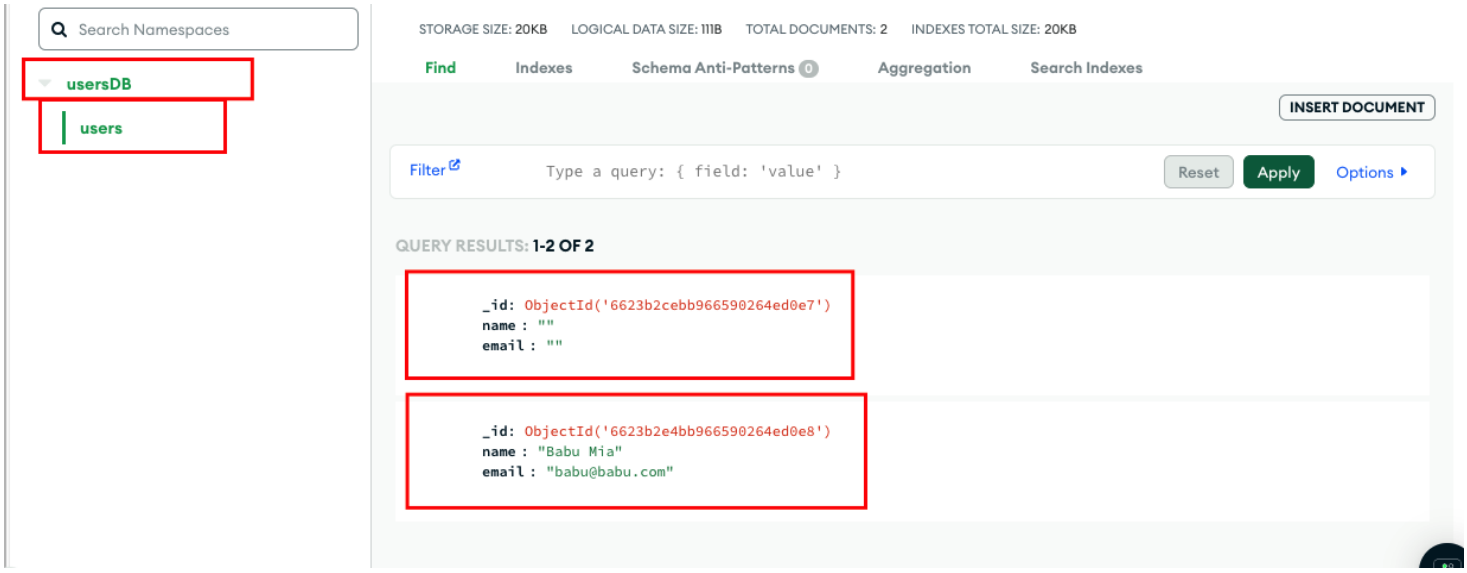
so we already have similar code setup in our express now going to edit here and there:



UI:

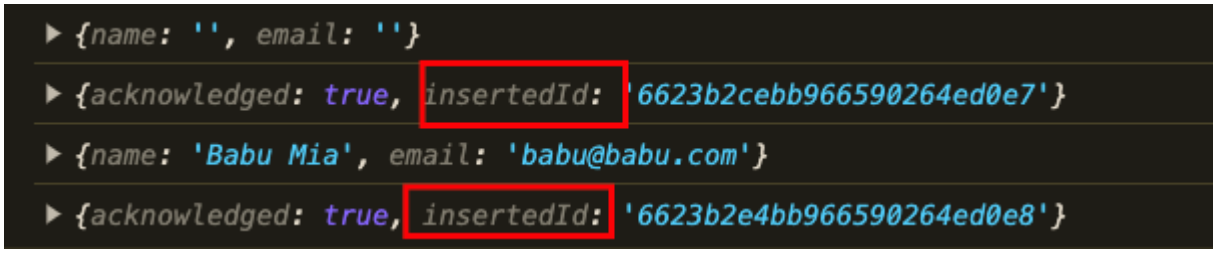


in atlas database:



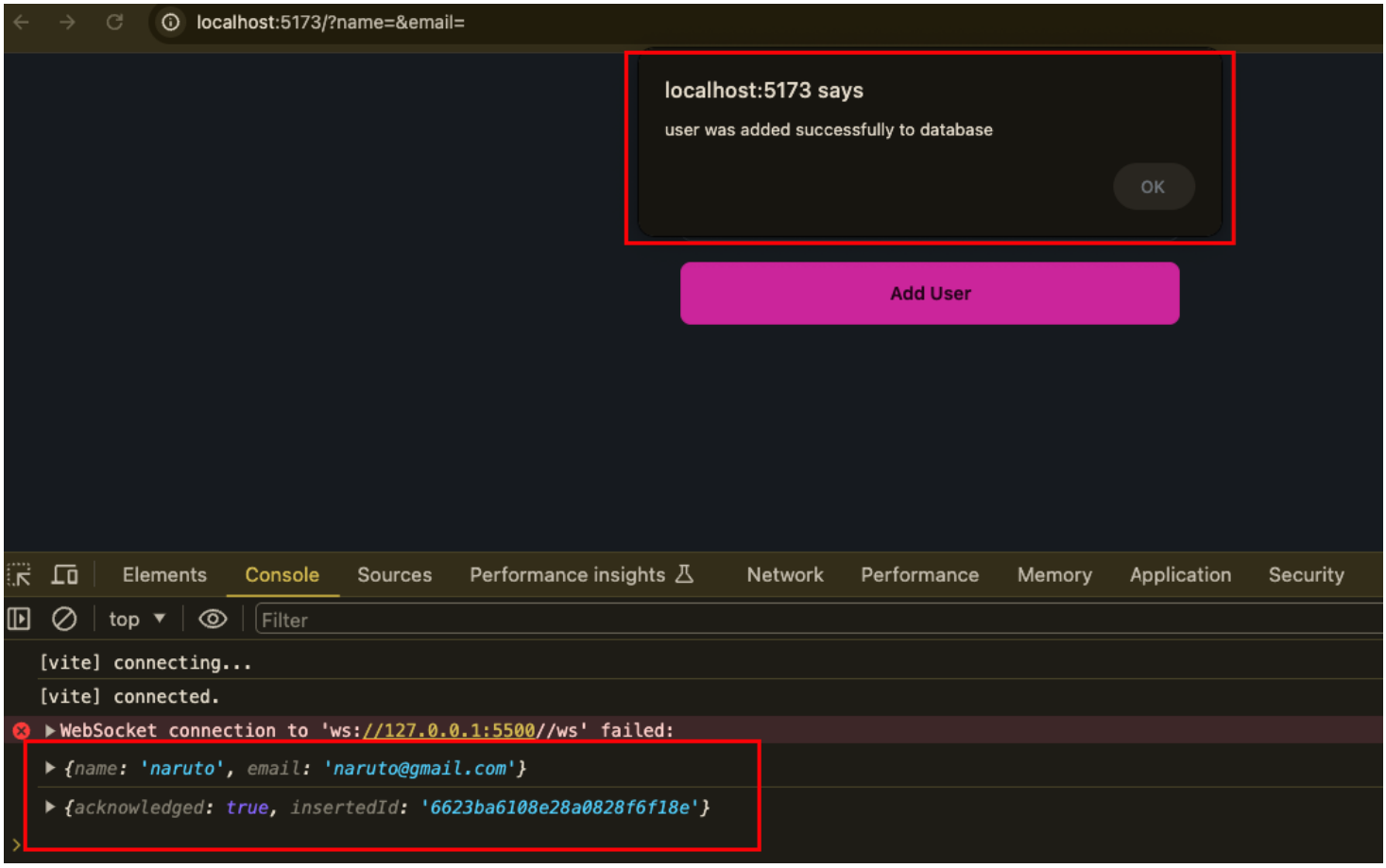
Yes we did it !

on the UI we are getting a response as data, and inside the data we have `insertedId`



so we can write an if statement and an alert which would let us know that an user was added.

```
    fetch("http://localhost:3000/users", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(user),
    })
        .then((res) => res.json())
        .then((data) => {
            console.log(data);
            if (data.insertedId) {
                alert("user was added successfully to database");
                form.reset();
            }
        })
        .catch((err) => console.log(err));  TypeError { stack: 'TypeError: Failed
};
```



mongoDb atlas:

QUERY RESULTS: **1-3 OF 3**

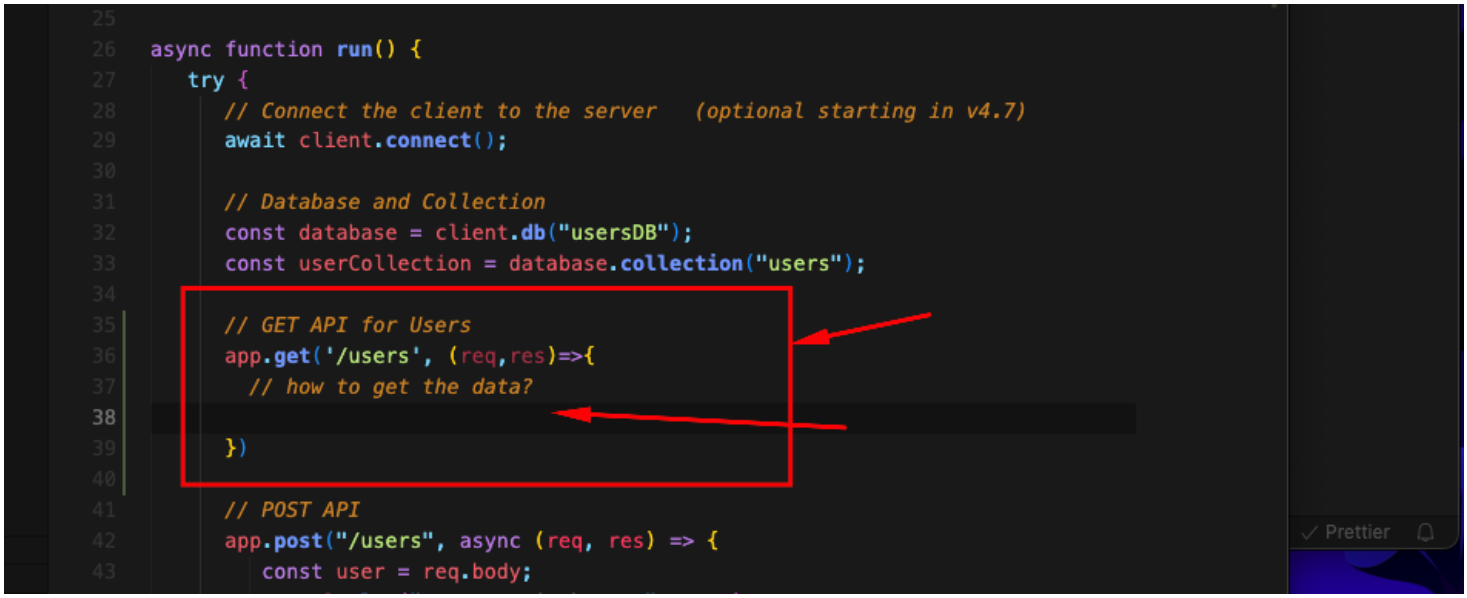
```
_id: ObjectId('6623b2cebb966590264ed0e7')
name : ""
email : ""
```

```
_id: ObjectId('6623b2e4bb966590264ed0e8')
name : "Babu Mia"
email : "babu@babu.com"
```

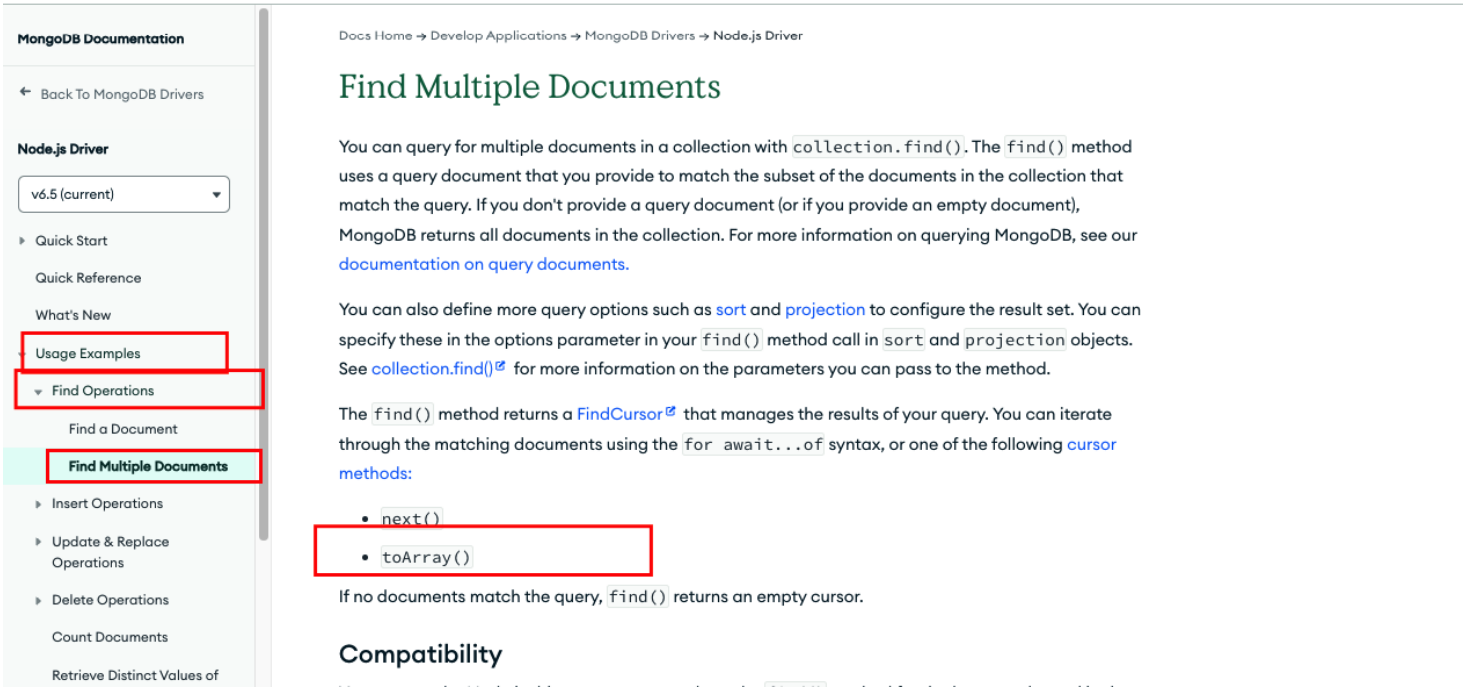
```
_id: ObjectId('6623ba6108e28a0828f6f18e')
name : "naruto"
email : "naruto@gmail.com"
```

Now our aim is to show multiple users when we go to the route "/users"

so first e amra GET users route ta kore feli server side e.



mongo r documentation:

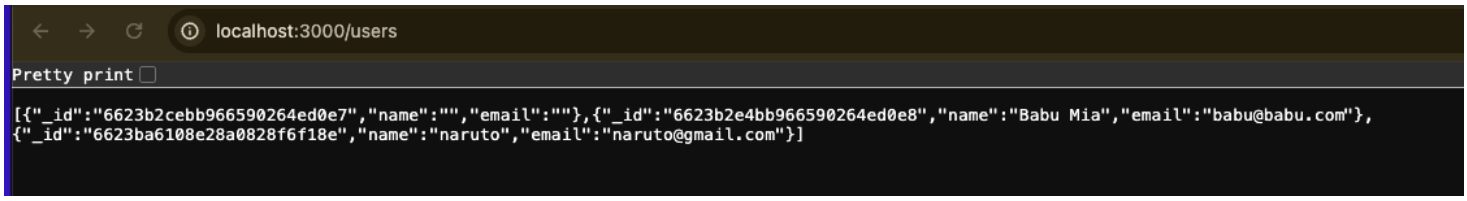



```
async function run() {
  try {

    // Get the database and collection on which to run the operation
    const database = client.db("sample_mflix");
    const movies = database.collection("movies");

    // Query for movies that have a runtime less than 15 minutes
    const query = { runtime: { $lt: 15 } };

    const options = {
      // Sort returned documents in ascending order by title (A->Z)
      sort: { title: 1 },
      // Include only the `title` and `imdb` fields in each returned document
      projection: { _id: 0, title: 1, imdb: 1 },
    };

    // Execute query
    const cursor = movies.find(query, options);

    // Print a message if no documents were found
    if ((await movies.countDocuments(query)) === 0) {
      console.log("No documents found!");
    }

    // Print returned documents
    for await (const doc of cursor) {
      console.dir(doc);
    }

  } finally {
    await client.close();
```

so amra pura mil na korleo amra amader moto code lekhbo:

```
async function run() {
  try {
    // Connect the client to the server   (optional starting in v4.7)
    await client.connect();

    // Database and Collection
    const database = client.db("usersDB");
    const userCollection = database.collection("users");

    // GET API for Users
    app.get("/users", async (req, res) => {
      // how to get the data?
      const cursor = userCollection.find();
      const result = await cursor.toArray();
      res.send(result);
    });

    // POST API
```

apatoto amra find e kono argument pass na korei try kortesi



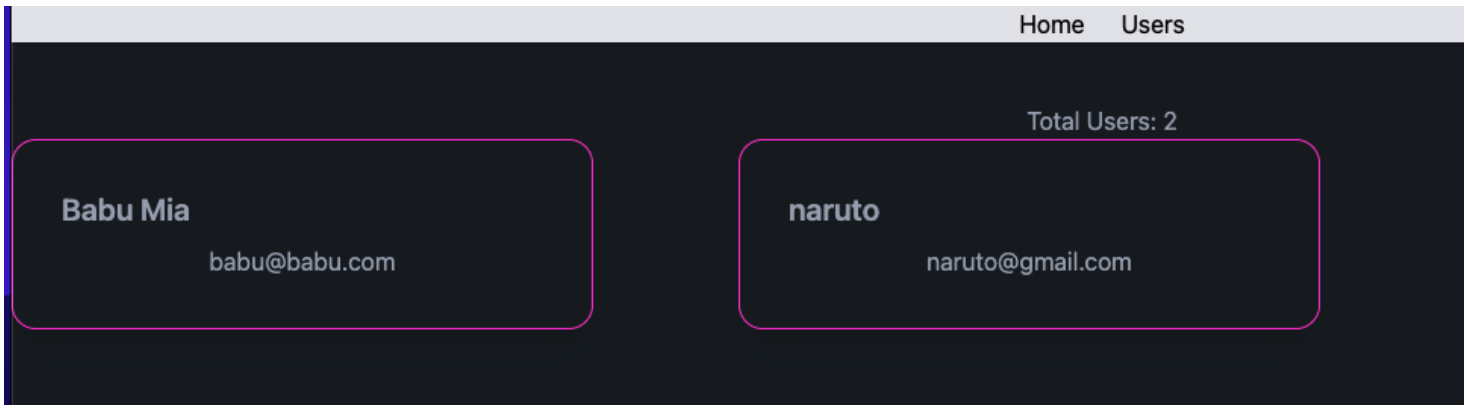
on client side:

```
  {
    path: "/users",
    element: <Users></Users>,
    loader: () => {
      return fetch("http://localhost:3000/users");
    },
  },
]
```

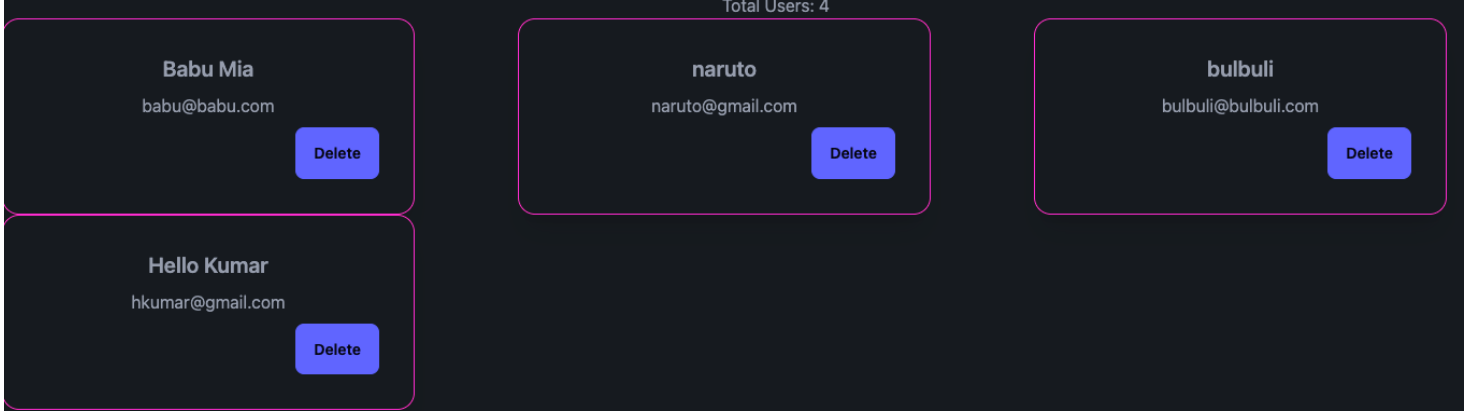
users.jsx:

```jsx
import { useLoaderData } from "react-router-dom";

const Users = () => {
    const users = useLoaderData();
    return (
        <div className="text-center py-[40px]">
            <h3>Total Users: {users.length}</h3>
            <div className="grid grid-cols-3">
                {users.map((user) => {
                    return (
                        <div
                            key={user._id}
                            className="card w-96 bg-base-100
                            shadow-xl border border-secondary"
                        >
                            <div className="card-body">
                                <h2 className="card-title">{user.
                                name}</h2>
                                <p>{user.email}</p>
                            </div>
                        </div>
                    );
                })}
            </div>
        </div>
    );
```

UI:



UI: Client Side:

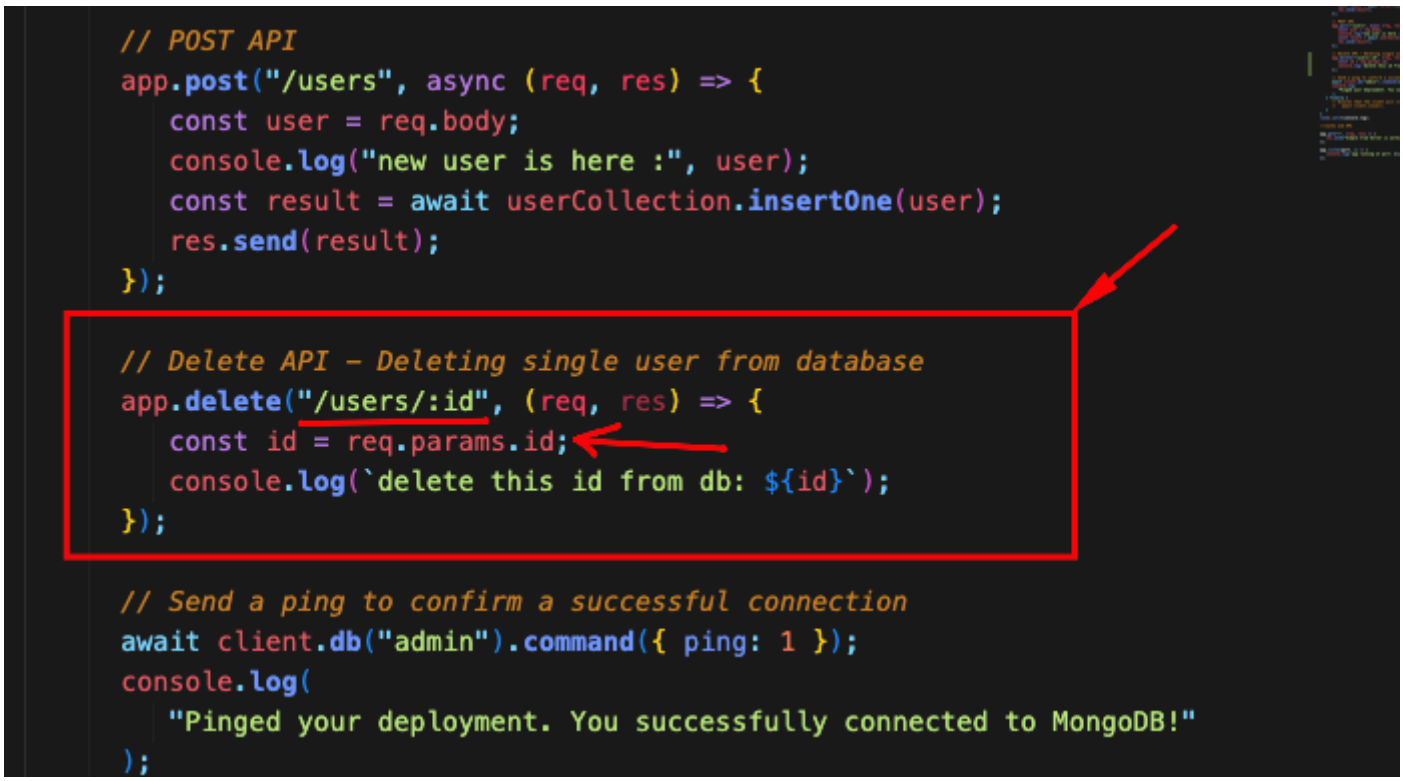


Amra delete button create korbo and click korle amader `id` ta kintu pathaite hobe jei id dhore pore delete ta hobe.

```
const handleDeleteUser = (id) => {
   console.log(id);
};
```

```
<button
   onClick={() => {
      handleDeleteUser(user._id);
   }}
   className="btn btn-primary"
>
   Delete
</button>
```

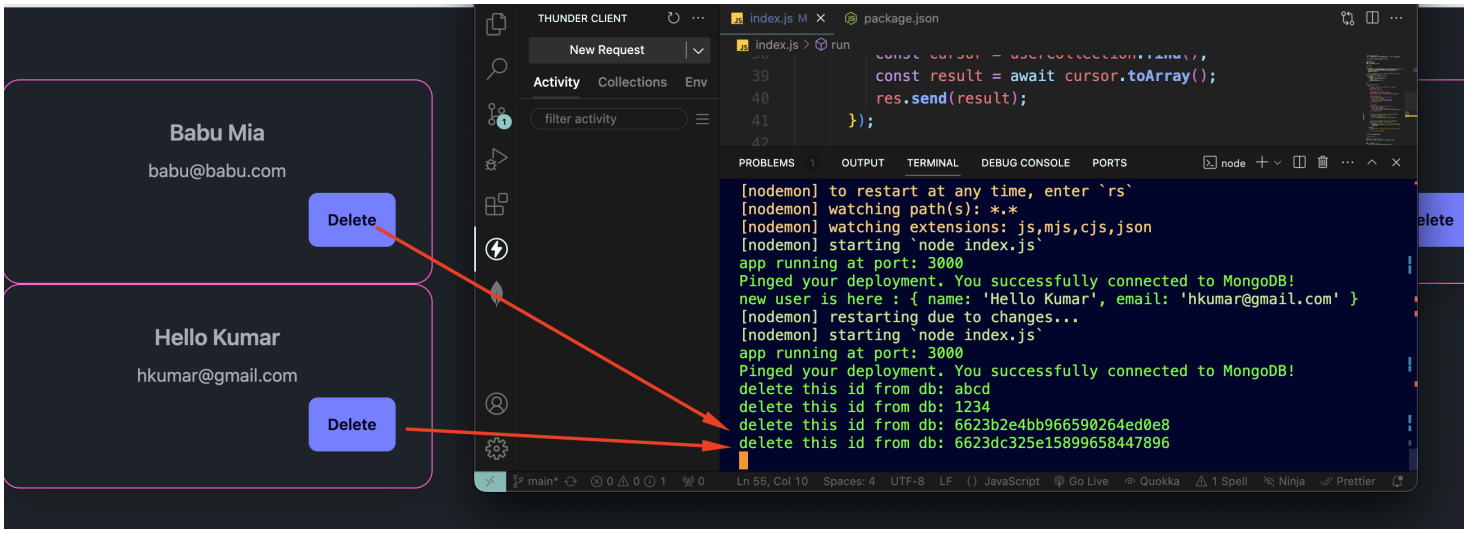
Now delete operation to backend:



so going back to the handleDeleteUser button:

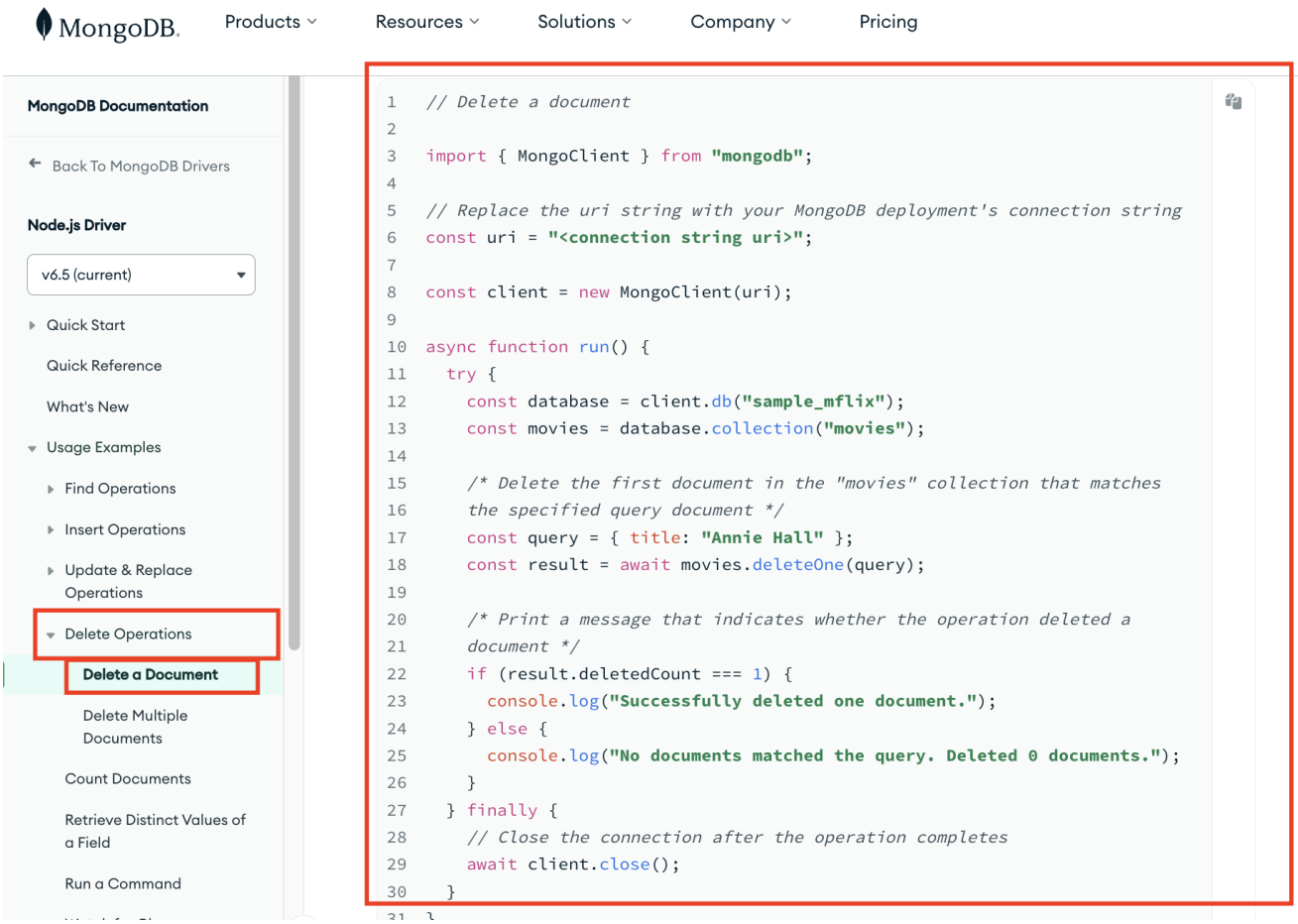
```
const handleDeleteUser = (id) => {
    console.log(id);  ×2 '6623dc325e15899658447896' ⏳ '6
    fetch(`http://localhost:3000/users/${id}`, {
        method: "DELETE",
    })
        .then((res) => res.json())
        .then((data) => console.log(data))
        .catch((err) => console.error(err));
};
```

server log



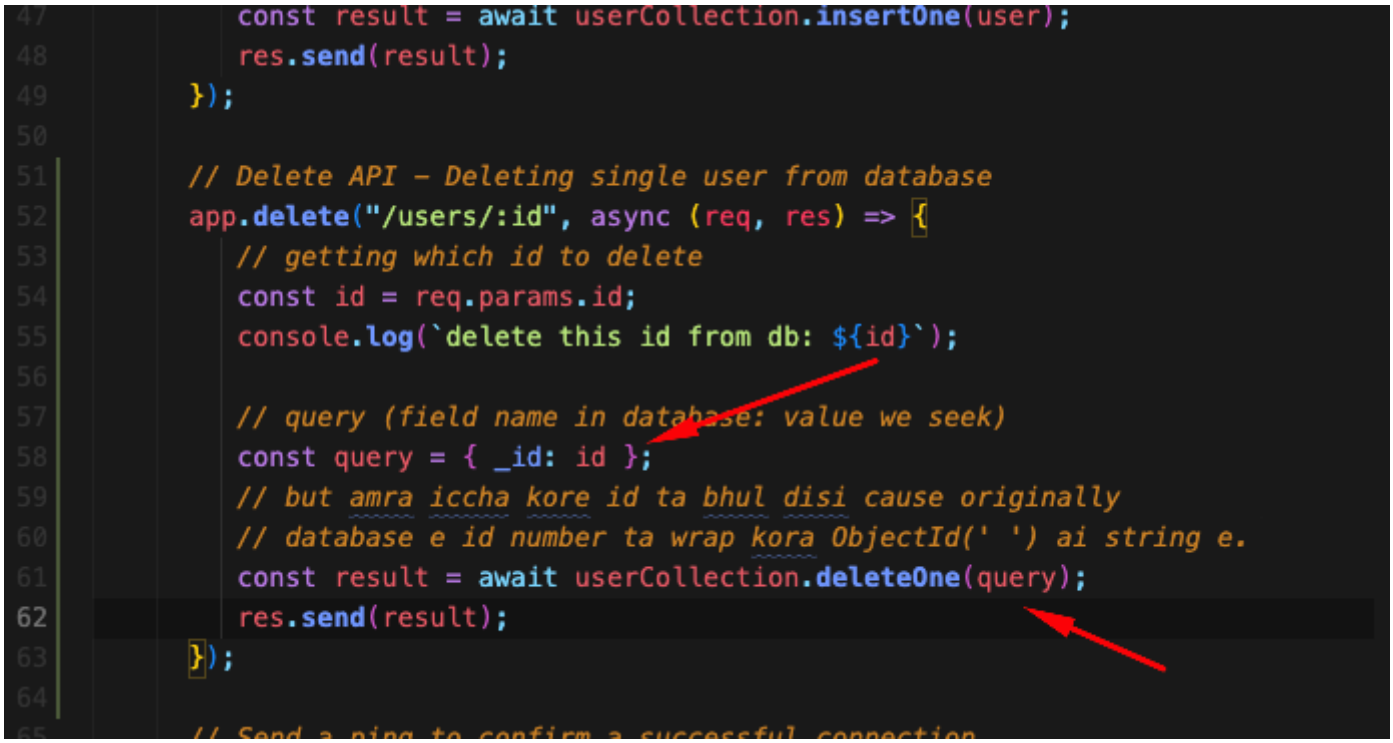
so amra kintu id ta pathaite partesi client thike server e. nice!

now mongo

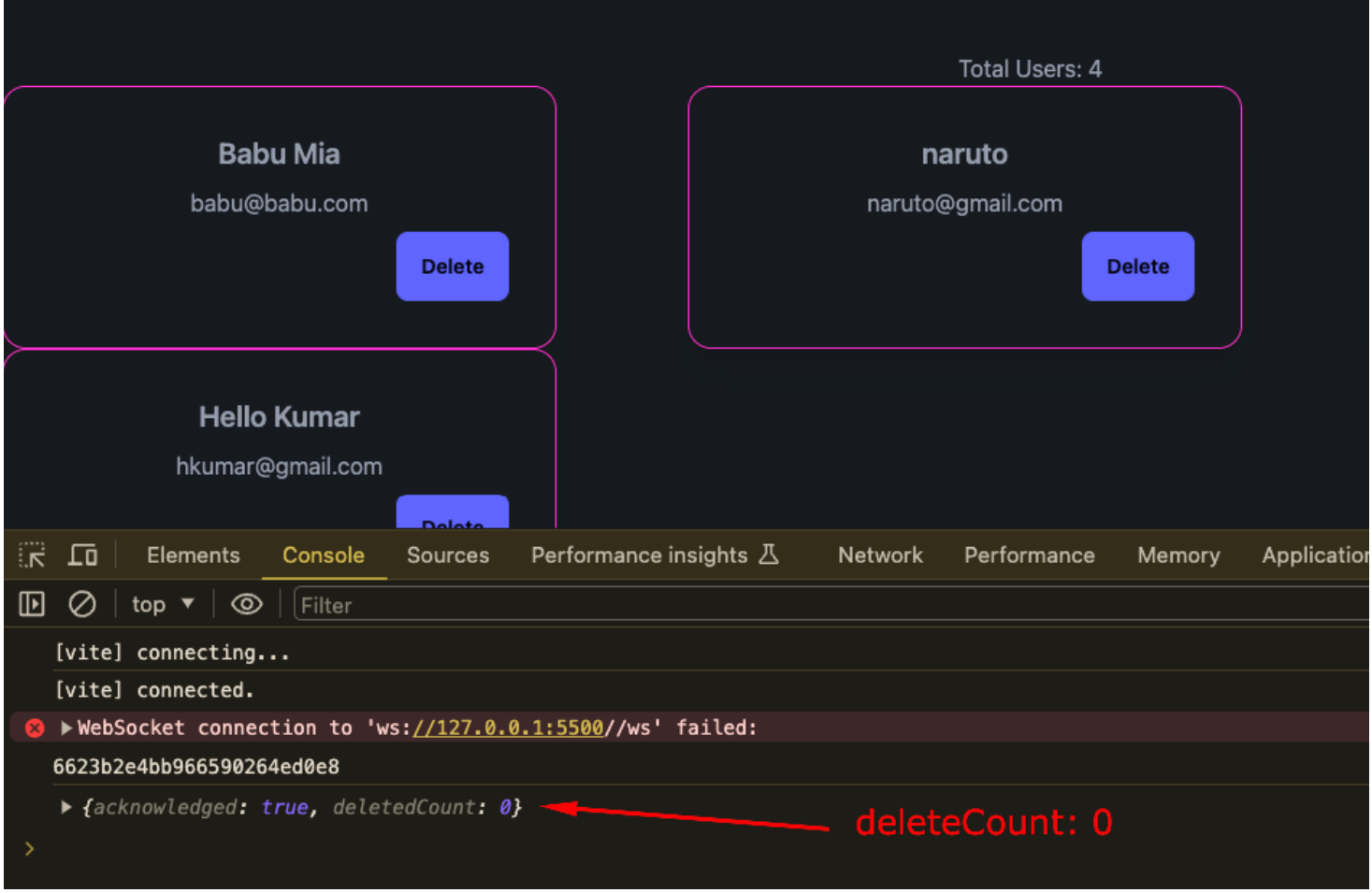


code:

but amra iccha kore id ta bhul disi cause originally database e id number ta wrap kora `ObjectId(' ')` ai string e.



and front end e akhon jodi dekhi je ki response pacchi jodi delete button click kori:



we can show an alert system:

```
const handleDeleteUser = (id) => {
    console.log(id);
    fetch(`http://localhost:3000/users/${id}`, {
        method: "DELETE",
    })
        .then((res) => res.json())
        .then((data) => {
            console.log(data);
            if (data.deletedCount > 0) {
                alert("Data removed from database");
            }
        })
        .catch((err) => console.error(err));
};
```

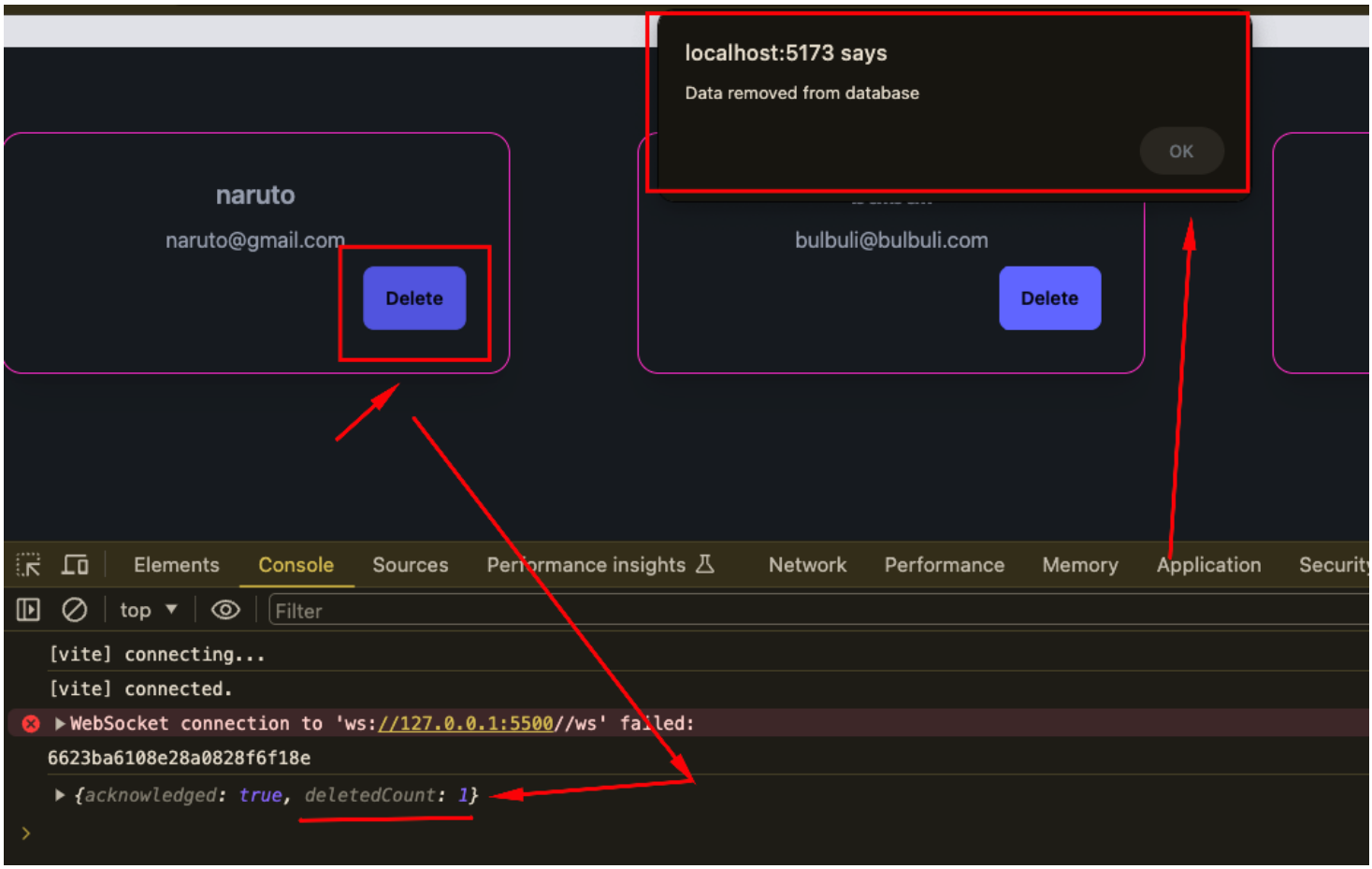
so akhon solve korbo kibhabe ObjectId r issue ta? By importing `ObjectId` and using the `new` keyword

backend:

```
const app = express();
const { MongoClient, ServerApiVersion, ObjectId } = require("mongodb");
```

```
// Delete API - Deleting single user from database
app.delete("/users/:id", async (req, res) => {
    // getting which id to delete
    const id = req.params.id;
    console.log(`delete this id from db: ${id}`);

    // query (field name in database: value we seek)

    const query = { _id: new ObjectId(id) };

    const result = await userCollection.deleteOne(query);
    res.send(result);
});
```

now:

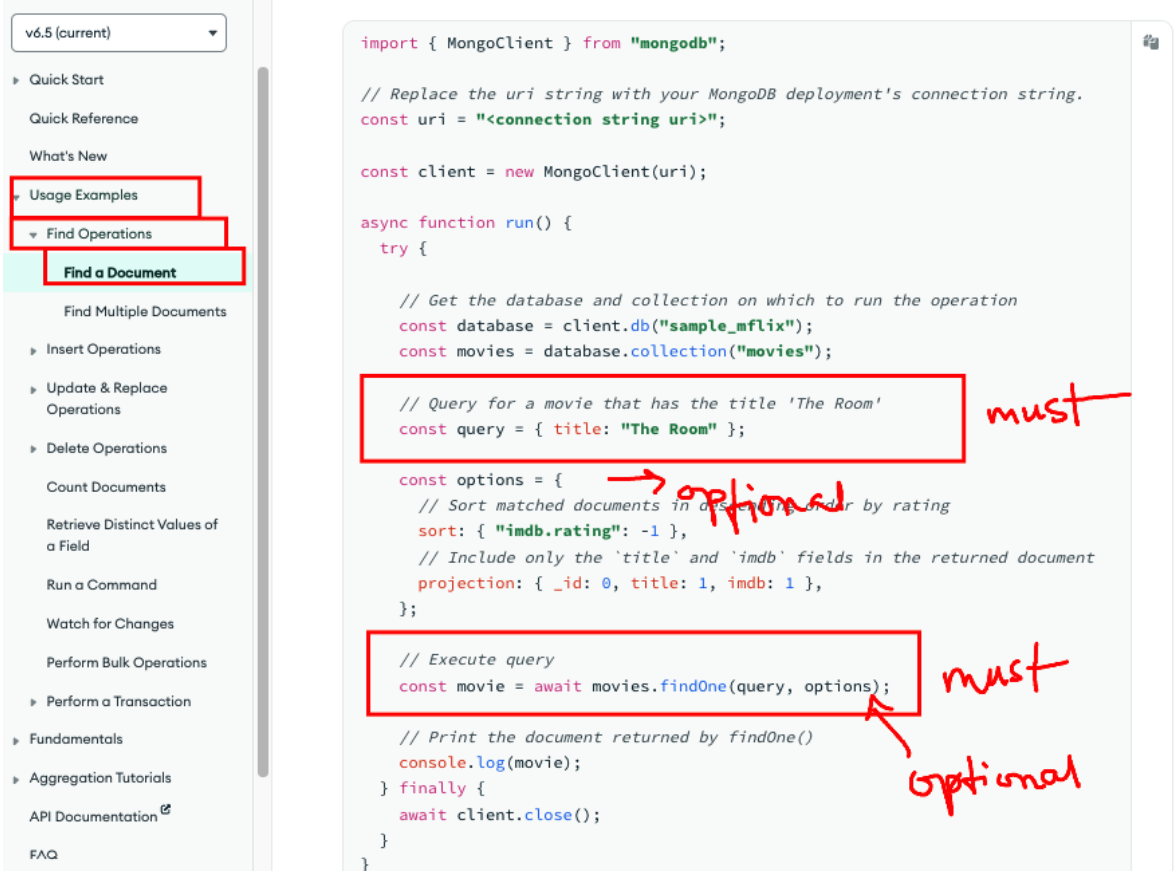


Great, but User is showing in UI till we refresh,
To Solve this: set a state

Perfect.

Now interesting part -> `CREATING A SINGLE DATA API in the backend`

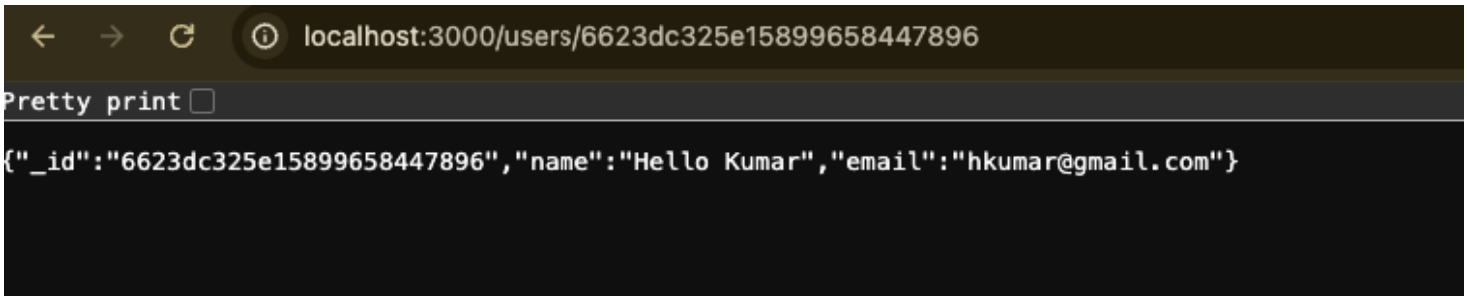
then loading that data in the client side and updating it.



our code backend:

```
// GET API for a single User:
app.get("/users/:id", async (req, res) => {
    const id = req.params.id;
    const query = { _id: new ObjectId(id) };
    const user = await userCollection.findOne(query);

    res.send(user);
});
```

so now we got an API which can show single data



Now let us make this route and get dynamic id in the client side:

root.jsx:

```
            },
        },
        {
            path: "/users/:id",
            element: <Update></Update>,
            loader: ({ params }) => {
                return fetch(`http://localhost:3000/users/$
                {params.id}`);
            },
        },
    ],
},
```

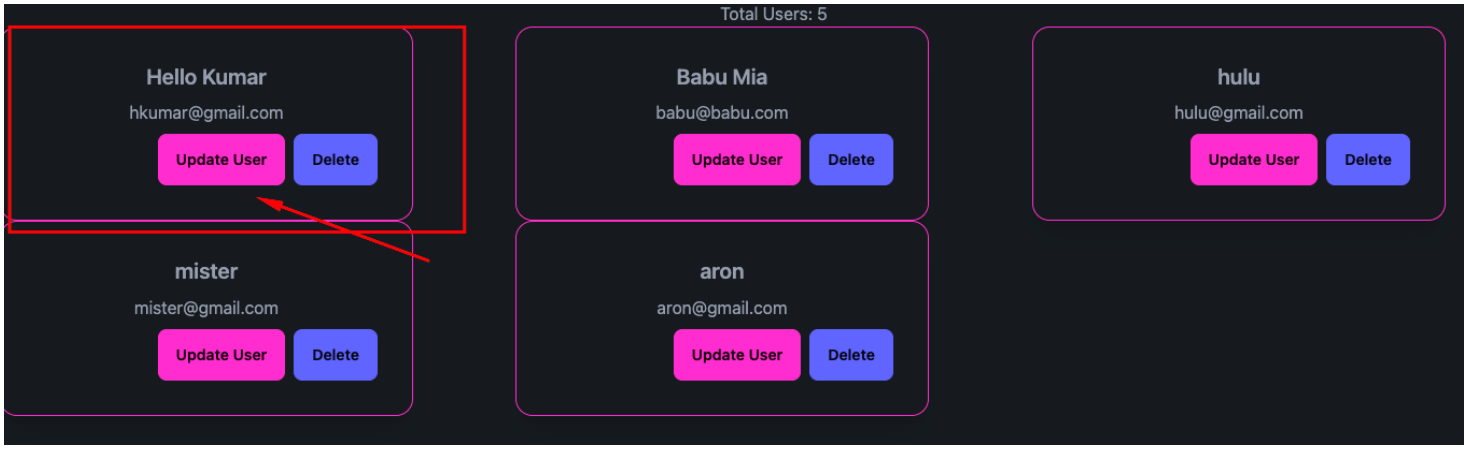
update.jsx:

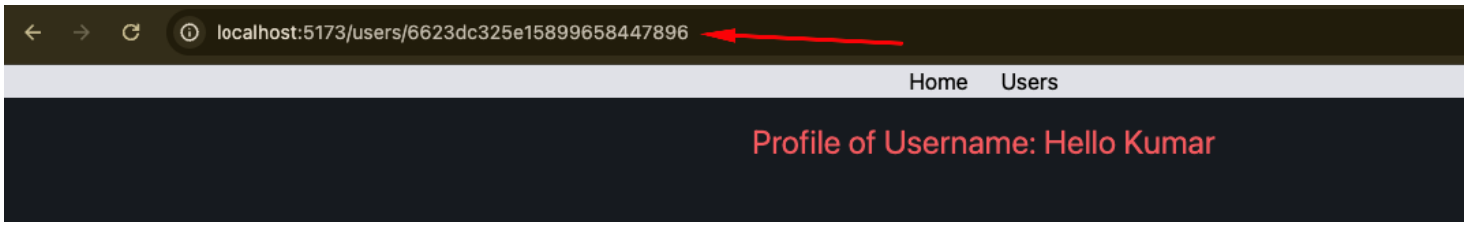
```
const Update = () => {
    const user = useLoaderData();

    return (
        <div>
            <h3 className="text-2xl mt-4 text-red-400
            text-center">
                Profile of Username: {user.name}
            </h3>
        </div>
    );
};

export default Update;
```

users.jsx:

```
                <p>{user.email}</p>
                <div className="card-actions
                justify-end">
                    <Link to={`/users/${user._id}`}>
                        <button className="btn
                        btn-secondary">
                            Update User
                        </button>
                    </Link>
                    <button
                        onClick={() => {
                            handleDeleteUser(user._id);
```

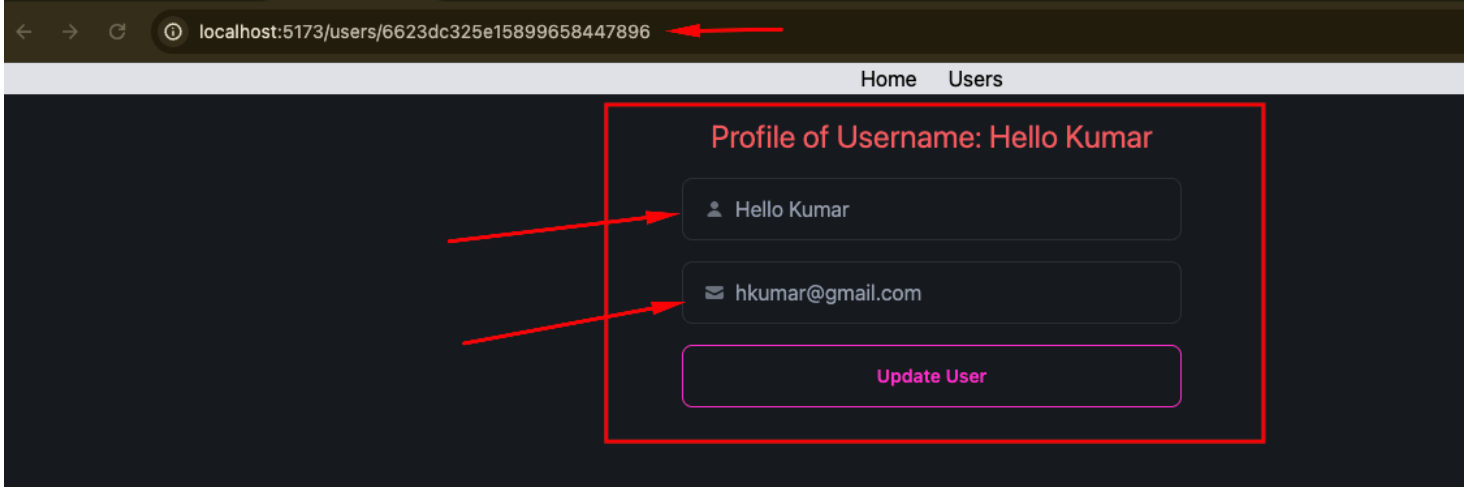
UI:





Now let us show the user info in a form and using default values.

```
const Update = () => {
  const user = useLoaderData();

  const handleUpdateUser = (event) => {
    event.preventDefault();
    const form = event.target;
    const name = form.name.value;
    const email = form.email.value;
    console.log(name, email);  ×2 'Hello Kumar' | 'hkumar@gmail.com' ⌛ 'Hello
  };

  return (
    <div>
      <h3 className="text-2xl mt-4 text-red-400 text-center">…
      </h3>

      <form
        onSubmit={handleUpdateUser}
        className="max-w-sm mx-auto space-y-4 mt-4"
      >
        {/* Name field */}
        <label className="input input-bordered flex items-center gap-2">
          <svg …
          </svg>
          <input
            type="text"
            name="name"
            className="grow"
            defaultValue={user.name}
            placeholder="Username"
          />
```



PUT- thakle update korbe nahole purata entry korbe
PATCH - update parts

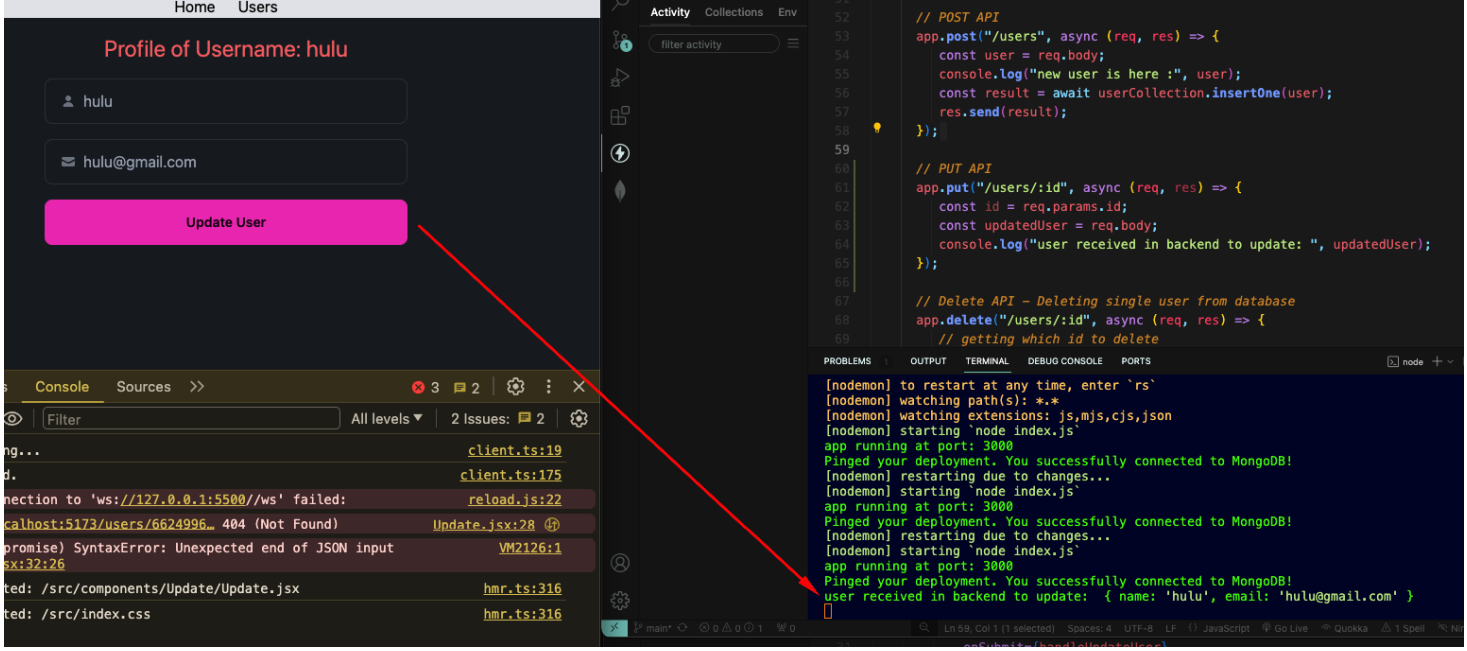
so first e amra backend e PUT API create korbo:

```
// PUT API
app.put("/users/:id", async (req, res) => {
   const id = req.params.id;
   const updatedUser = req.body;
   console.log("user received in backend to update: ", updatedUser);
});
```

client side e akhon code korbo jate backend e user info ta pathano jae:

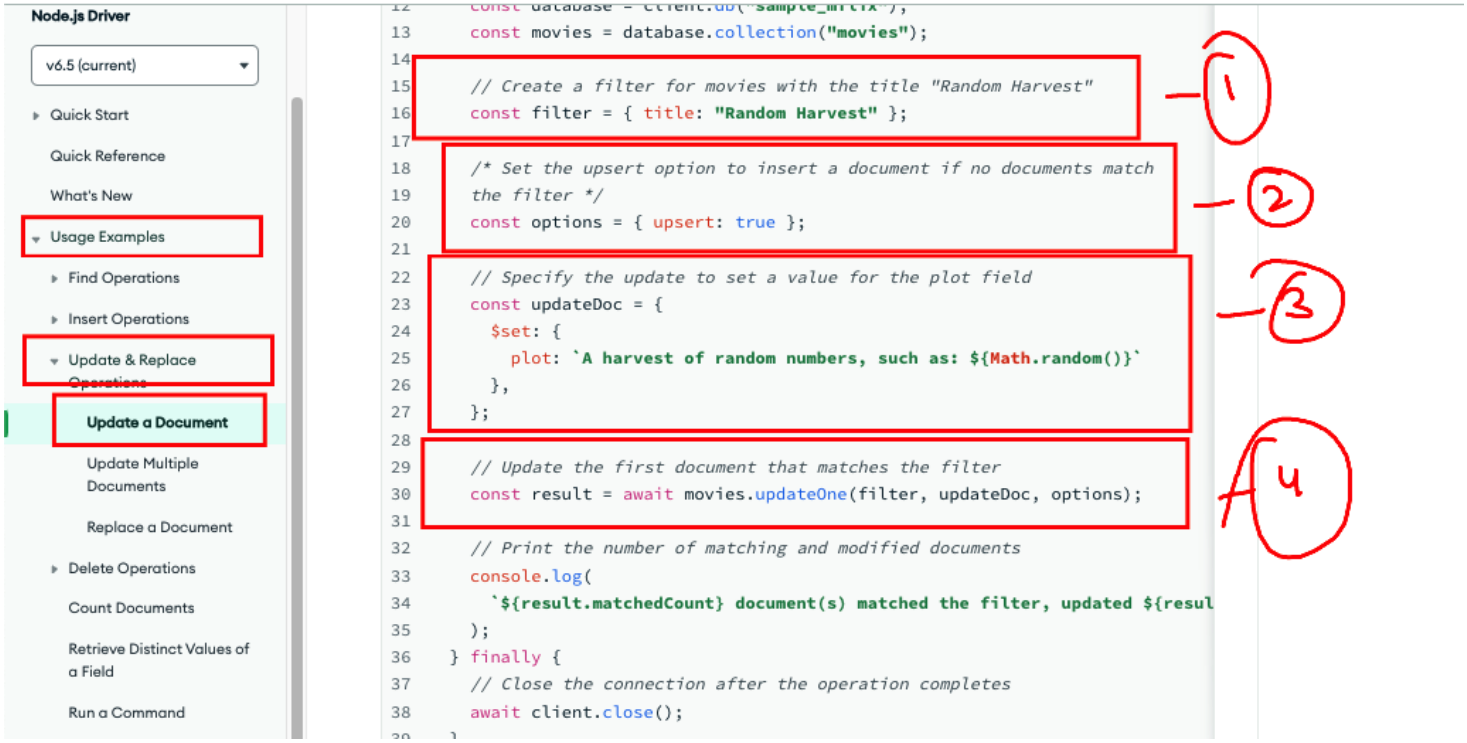
```
const Update = () => {
   const user = useLoaderData();

   const handleUpdateUser = (event) => {
      event.preventDefault();
      const form = event.target;
      const name = form.name.value;
      const email = form.email.value;
      const updatedUser = { name, email };

      fetch(`http://localhost:3000/users/${user._id}`, {
         method: "PUT",
         headers: { "content-type": "application/json" },
         body: JSON.stringify(updatedUser),
      })
         .then((res) => res.json())
         .then((data) => {
            console.log(data);
         });
   };

   return (
      <div>
```

akhon jodi amra update click kori, request ta kintu backend e show kore:



akhon backend e jehutu id ta paisi PUT korar jonne, kaaj akhon mongor

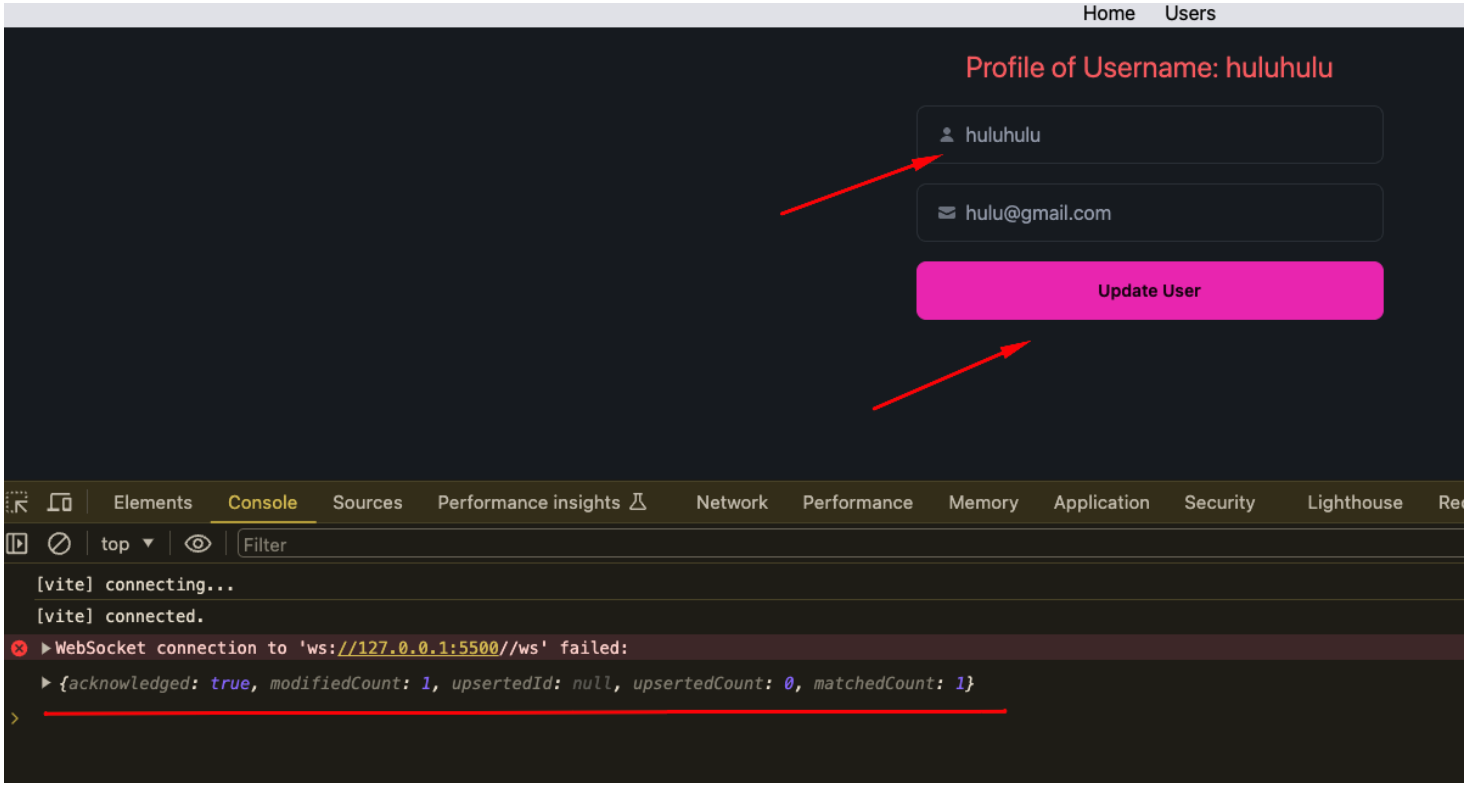
---



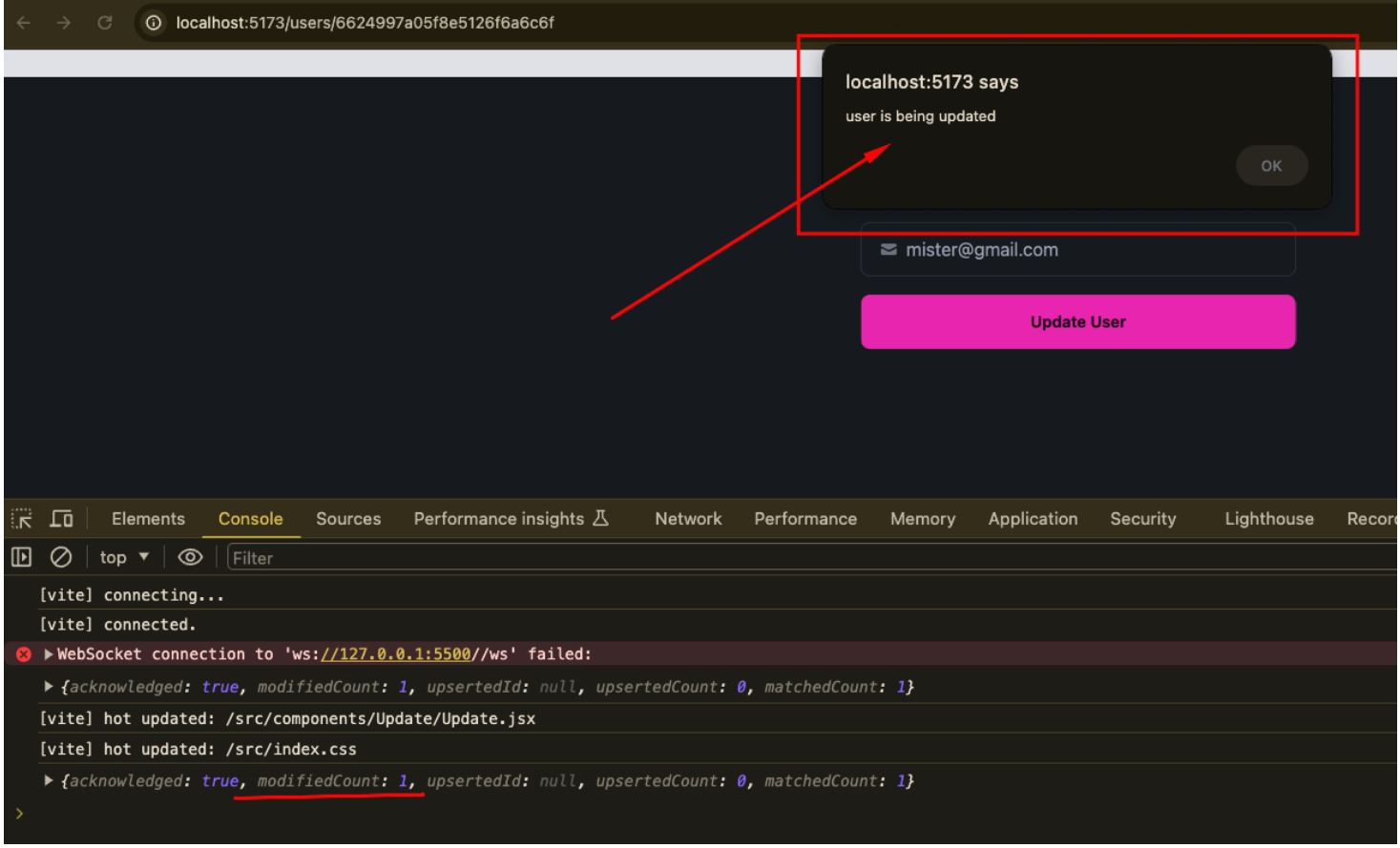
our code:

```
// PUT API
app.put("/users/:id", async (req, res) => {
    const id = req.params.id;
    const user = req.body;
    console.log("user received in backend to update: ", user);

    const filter = { _id: new ObjectId(id) };
    const options = { upsert: true };

    const updatedUser = {
        $set: {
            name: user.name,
            email: user.email,
        },
    };

    const result = await userCollection.updateOne(
        filter,
        updatedUser,
        options
    );

    res.send(result);
});
```

UI:



we can slow alert if we want:



Elements Console Sources Performance insights Network Performance Memory Application Security Lighthouse Recor

top Filter

```
[vite] connecting...
[vite] connected.
▶WebSocket connection to 'ws://127.0.0.1:5500//ws' failed:
▶ {acknowledged: true, modifiedCount: 1, upsertedId: null, upsertedCount: 0, matchedCount: 1}
[vite] hot updated: /src/components/Update/Update.jsx
[vite] hot updated: /src/index.css
▶ {acknowledged: true, modifiedCount: 1, upsertedId: null, upsertedCount: 0, matchedCount: 1}
```