**Back-end Q & A:**

1.To create a MongoDB collection from a CSV file and populate it with data for a REST API backend using Express, follow these steps:

**Steps to Create MongoDB Collection**

**Prerequisites:**

1. **Node.js and npm** installed.
2. **MongoDB server** running locally or remotely.
3. **CSV file** with the required data.

**Steps:**

### ****1. Create MongoDB Collection****

1. Install **MongoDB tools** if not installed:

**npm install -g mongo-tools**

**Convert the CSV file into MongoDB collection**: Assume the CSV file is named football\_data.csv and is in the same directory. Use the mongoimport tool:

**mongoimport --uri "mongodb://localhost:5000/<DATABASE\_NAME>" --collection FootBallData --type csv --headerline --file football\_data.csv**

Replace <DATABASE\_NAME> with the name of your database.

Verify the import in the Mongo shell or Compass:

**use <DATABASE\_NAME>**

**db.FootBallData.find()**

### ****2. Node.js Backend (Express REST API):****

#### Install Required Modules

Run the following command:

**npm install express mongoose body-parser**

#### Create server.js File

Here’s an example of a simple Express app connected to MongoDB:

**const express = require("express");**

**const mongoose = require("mongoose");**

**const bodyParser = require("body-parser");**

**// Initialize Express app**

**const app = express();**

**app.use(bodyParser.json());**

**// MongoDB Connection**

**const mongoURI = "mongodb://localhost:27017/<DATABASE\_NAME>"; // Replace <DATABASE\_NAME> with your DB name**

**mongoose**

**.connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })**

**.then(() => console.log("MongoDB Connected"))**

**.catch((err) => console.error(err));**

**// Define Mongoose Schema and Model**

**const footballSchema = new mongoose.Schema({}, { strict: false }); // Flexible schema for CSV data**

**const FootBallData = mongoose.model("FootBallData", footballSchema);**

**// Routes**

**// Fetch all records**

**app.get("/api/football-data", async (req, res) => {**

**try {**

**const data = await FootBallData.find();**

**res.json(data);**

**} catch (err) {**

**res.status(500).json({ error: err.message });**

**}**

**});**

**// Insert a record**

**app.post("/api/football-data", async (req, res) => {**

**try {**

**const newData = new FootBallData(req.body);**

**const savedData = await newData.save();**

**res.json(savedData);**

**} catch (err) {**

**res.status(500).json({ error: err.message });**

**}**

**});**

**// Start Server**

**const PORT = process.env.PORT || 5000;**

**app.listen(PORT, () => console.log(`Server running on port ${PORT}`));**

**3. Test API**

* Use tools like Postman or curl to test the endpoints:
  + **GET /api/football-data**: Retrieve all records.
  + **POST /api/football-data**: Add a new record.

**Notes:**

* If the CSV structure is complex, specify fields in footballSchema explicitly.
* Adjust the MongoDB URI if using a remote database (e.g., MongoDB Atlas).

2.Install Dependencies

**npm install express mongoose body-parser**

**Create a Mongoose Schema and Model**

File: models/YourModel.js

const mongoose = require('mongoose');

// Define the Schema

const yourSchema = new mongoose.Schema({

name: { type: String, required: true },

age: { type: Number, required: true },

email: { type: String, required: true, unique: true },

address: {

street: String,

city: String,

country: String

},

createdAt: { type: Date, default: Date.now }

});

// Create the Model

const YourModel = mongoose.model('YourModel', yourSchema);

module.exports = YourModel;

**Set up the Express Application**

File: server.js

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const YourModel = require('./models/YourModel');

const app = express();

const PORT = 3000;

// Middleware

app.use(bodyParser.json());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/yourDatabase', {

useNewUrlParser: true,

useUnifiedTopology: true

}).then(() => {

console.log("Connected to MongoDB");

}).catch(err => {

console.error("Error connecting to MongoDB", err);

});

// REST API Endpoints

// Add a new document

app.post('/add', async (req, res) => {

try {

const newDoc = new YourModel(req.body);

await newDoc.save();

res.status(201).send(newDoc);

} catch (error) {

res.status(400).send(error);

}

});

// Delete a document by ID

app.delete('/delete/:id', async (req, res) => {

try {

const deletedDoc = await YourModel.findByIdAndDelete(req.params.id);

if (!deletedDoc) {

return res.status(404).send({ message: "Document not found" });

}

res.status(200).send(deletedDoc);

} catch (error) {

res.status(500).send(error);

}

});

// Update a document by ID

app.put('/update/:id', async (req, res) => {

try {

const updatedDoc = await YourModel.findByIdAndUpdate(req.params.id, req.body, {

new: true,

runValidators: true

});

if (!updatedDoc) {

return res.status(404).send({ message: "Document not found" });

}

res.status(200).send(updatedDoc);

} catch (error) {

res.status(400).send(error);

}

});

// Find documents

app.get('/find', async (req, res) => {

try {

const docs = await YourModel.find(req.query);

res.status(200).send(docs);

} catch (error) {

res.status(500).send(error);

}

});

// Start the server

app.listen(PORT, () => {

console.log(`Server running on http://localhost:${PORT}`);

});

**Run the Application**

Start the server:

node server.js

**Test the Endpoints**

Use tools like [Postman](https://www.postman.com/) or [cURL](https://curl.se/) to test the endpoints.

* **Add**: POST <http://localhost:5000/add>

{

"name": "John Doe",

"age": 30,

"email": "johndoe@example.com",

"address": {

"street": "123 Elm St",

"city": "Springfield",

"country": "USA"

}

}

 **Find**: GET http://localhost:5000/find

* Query Example: http://localhost:5000/find?name=John%20Doe

 **Update**: PUT [http://localhost:5000/update/<id](http://localhost:5000/update/%3cid)>

{

"age": 35

}

**Delete**: DELETE http://localhost:5000/delete/<id>

### Summary

This implementation creates a modular and clean Express application, demonstrating Mongoose queries for CRUD operations. Adjust the schema, routes, and error handling to match your specific needs.

**3.**

**Set Up Project**

**Initialize the project**:

mkdir backend-app

cd backend-app

npm init -y

npm install express mongoose body-parser

Create the Mongoose Schema File:

Create a file named schema.js (or any name of your choice) that will define and export the Mongoose schema.

#### schema.js:

#### const mongoose = require('mongoose');

#### // Define the schema

#### const userSchema = new mongoose.Schema({

#### name: { type: String, required: true },

#### email: { type: String, required: true, unique: true },

#### password: { type: String, required: true }

#### });

#### // Create and export the model

#### module.exports = mongoose.model('User', userSchema);

### Step Create the MongoDB Connection File

Create another file named db.js (or any name you prefer) to connect to the MongoDB database and import the schema.

#### db.js:

#### const mongoose = require('mongoose');

#### const User = require('./schema'); // Import the schema

#### // MongoDB connection URI

#### const mongoURI = 'mongodb://127.0.0.1:27017/mydatabase';

#### // Connect to MongoDB

#### mongoose.connect(mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })

#### .then(() => console.log('MongoDB connected successfully'))

#### .catch(err => console.error('MongoDB connection error:', err));

#### // Export the User model for use in other files

#### module.exports = { User };

### Set Up the Express REST API

Create a file named server.js to set up the Express server and demonstrate CRUD operations using the REST API.

#### server.js:

#### const express = require('express');

#### const bodyParser = require('body-parser');

#### const { User } = require('./db'); // Import the User model from the connection file

#### const app = express();

#### const PORT = 5000;

#### // Middleware

#### app.use(bodyParser.json());

#### // REST API Endpoints

#### // Create a new user

#### app.post('/api/users', async (req, res) => {

#### try {

#### const newUser = new User(req.body);

#### await newUser.save();

#### res.status(201).json(newUser);

#### } catch (error) {

#### res.status(400).json({ error: error.message });

#### }

#### });

#### // Get all users

#### app.get('/api/users', async (req, res) => {

#### try {

#### const users = await User.find();

#### res.json(users);

#### } catch (error) {

#### res.status(500).json({ error: error.message });

#### }

#### });

#### // Update a user

#### app.put('/api/users/:id', async (req, res) => {

#### try {

#### const updatedUser = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });

#### if (!updatedUser) return res.status(404).json({ error: 'User not found' });

#### res.json(updatedUser);

#### } catch (error) {

#### res.status(400).json({ error: error.message });

#### }

#### });

#### // Delete a user

#### app.delete('/api/users/:id', async (req, res) => {

#### try {

#### const deletedUser = await User.findByIdAndDelete(req.params.id);

#### if (!deletedUser) return res.status(404).json({ error: 'User not found' });

#### res.json({ message: 'User deleted successfully' });

#### } catch (error) {

#### res.status(500).json({ error: error.message });

#### }

#### });

#### // Start the server

#### app.listen(PORT, () => {

#### console.log(`Server running on http://localhost:${PORT}`);

#### });

**Run the Application**

1. Start MongoDB on your system:

#### Mongod

#### Run the server:

#### node server.js

#### Test the API endpoints using tools like Postman or curl.

### Example MongoDB CRUD Queries

#### Create a New User

POST /api/users  
Body:

#### {

#### "name": "John Doe",

#### "email": "john.doe@example.com",

#### "password": "securepassword"

#### }

#### Get All Users

GET /api/users

#### Update a User

PUT /api/users/:id  
Body:

#### {

#### "name": "John Smith"

#### }

#### Delete a User

DELETE /api/users/:id

This setup fulfills the requirement to:

1. Use Mongoose to define schemas.
2. Connect to MongoDB in a separate file.
3. Demonstrate MongoDB queries through a REST API.