MERN EMG

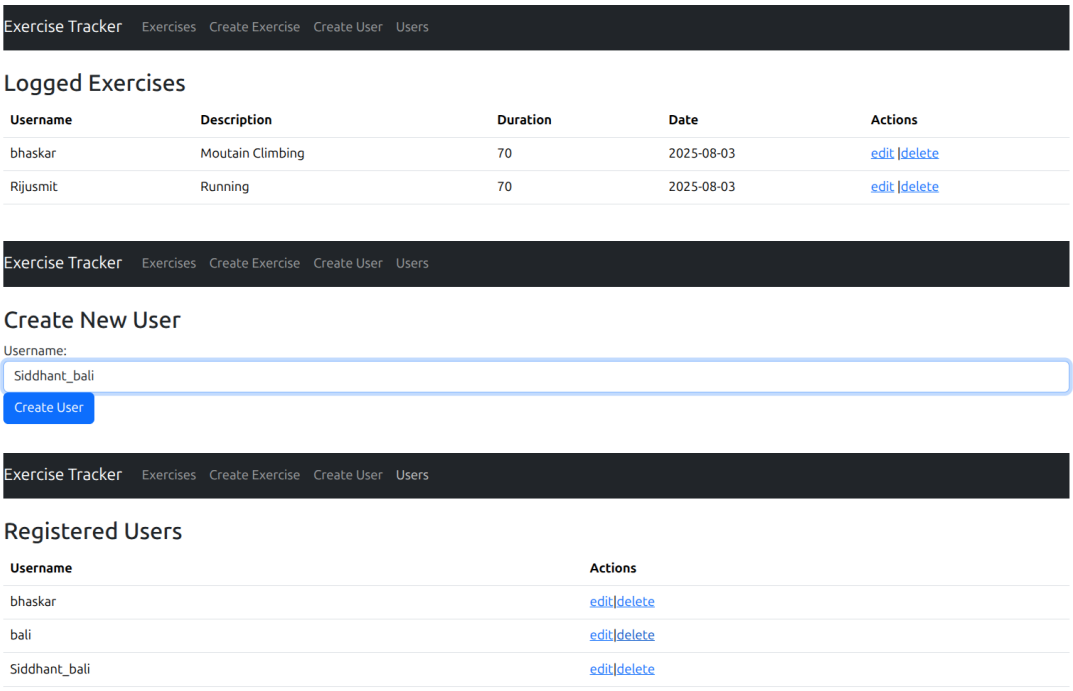
- MERN Stack with project Exercise Tracker App

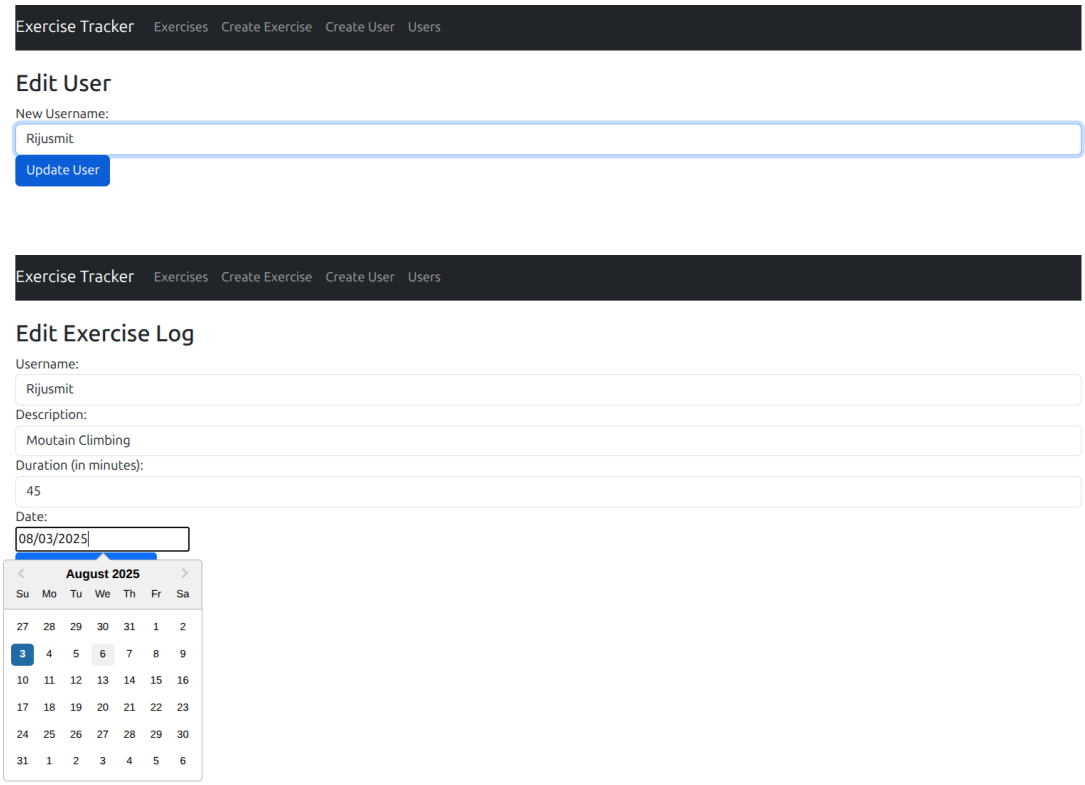
1. Introduction

Welcome to building web apps with the MERN stack (MongoDB, Express, React, Node.js), using MongoDB Atlas and Google Cloud Platform. In this tutorial, we will create an **Exercise Tracker** app.

Outline:

1. Introduction to the MERN stack
2. Database concepts (MongoDB vs. relational)
3. MongoDB Atlas setup
4. Backend (Node.js + Express + Mongoose)
5. Testing API with Insomnia/Postman
6. Frontend (React + React Router + Axios)
7. Connecting frontend to backend





---

2. Technology Stack

- **MongoDB**: Document-based open-source database
- **Express**: Web application framework for Node.js, help to create server, lightweight, fast
- **React**: Front-end JavaScript library for building UIs
- **Node.js**: JavaScript runtime environment that executes JS code outside of the browser (server)
- **Mongoose**: Simple, schema based solution to model application data. ODM (Object Data Modeling) library for MongoDB + Node.js
- **CORS**: Cross-Origin Resource Sharing middleware, helps to AJAX request to skip same origin policy and access resources from remote hosts. interlinked with Express middleware
- **dotenv**: Load environment variables from `.env` file into process.env, makes dev easier

Table of Contents

- MERN EMG
  - 1. Introduction
  - 2. Technology Stack
  - Table of Contents
  - 3. Database Concepts
    - 📚 MongoDB vs Mongoose — Explained Simply
      - 🛠️ Mongoose Essentials
      - 🔄 CRUD Operations (Actions on Data)
      - ⚙️ Example in Code
      - 🧠 TL;DR Quick Summary Table
    - 🧠 MongoDB + Mongoose Essentials
      - 🧩 What is Mongoose?

        - 🛠️ Setup (Install)
        - 🧠 MongoDB + Mongoose Cheatsheet
            - 1️⃣ Setup and Connect to MongoDB
            - 2️⃣ Create a Schema and Model
            - 3️⃣ Basic CRUD Operations
                - ✅ Create a New User
                - 🔍 Read / Find Users
                - ✏️ Update User Data
                - ❌ Delete User
            - 4️⃣ Additional Concepts
                - 🧪 Validation and Default Values
                - 🔗 Relationships using `ref` and `populate`
                - 🔁 Middleware (Hooks)
            - 📌 Common Mistakes to Avoid
            - 🧪 Sample Test Script
- | Relationships | Manual `$lookup` | `ref + populate()` |
- 4. MongoDB Atlas Setup
    - ObjectIds in MongoDB
- 7. Frontend Setup (React) (Just Initialize)
    - 7.1 Initialize React App
- Remove unused files (logo, serviceWorker) and CSS imports.
- 5. Backend Setup (Node.js + Express + Mongoose)
    - 5.1 Prerequisites
    - 5.2 Initialize Backend Project
    - nodemon
    - 5.3 Create `backend/server.js` [BASIC, to Append on the way]
    - 5.4 Environment Variable
    - Integrate Mongoose in `backend/server.js` to connect mongodb atlas with server
    - 5.5 Mongoose Models
        - `models/user.model.js`
        - `models/exercise.model.js`
    - 5.6 Routes
        - importing and using `routes/users.js` & `routes/exercises.js` in `server.js`
        - `routes/users.js`
            - 📦 `req` (Request)
            - 📩 `res` (Response)
        - `routes/exercises.js`
        - Params
- 6. Testing API (Insomnia / Postman)
- 7. Frontend Setup (React) (Actual Code Start)
    - React
    - 7.2 `public/index.html`
    - 7.3 `src/index.js`
    - 7.4 `src/App.js`
    - 7.5 Components

        - BootStrap Essentials
            - 1. Getting Started
            - 2. Layout: Grid System
            - 3. Buttons
            - 4. Form Input
            - 5. Navbar
            - 6. Card
            - 7. Utility Classes
            - 8. Responsive Visibility
            - 9. Container Types
        - 7.5.1 `src/components/Navbar.component.js`
        - 7.5.2 `src/components/CreateUser.component.js`
        - What the heck was that
            - 🧠 GOAL: A simple React app that:
                - 📦 Step 1: Simple `CreateUser.js`
                - 🧠 EXPLANATION: What the hell is happening
                - 🔄 How this flows in real life
                - 🔧 Your Backend Should Handle This Route
                - 🧪 How to test it:
        - New Way
        - 7.5.3 `src/components/CreateExercise.component.js`
        - 7.5.4 `src/components/ExerciseList.component.js`
        - 7.5.5 `src/components/EditExercise.component.js`
    - 8. Running the App
    - 9. Conclusion
    - Others
        - Bonus: User Upgrades
        - `this.props.match.params.id` rejected, now use `const { id } = useParams();` or `this.props.match.params.id`
        - 🧠 Node.js Essentials
            - 🚀 What is Node.js?
            - 📦 1. Setup and Initialize
            - 📂 2. File Structure (Basic Project)
            - 🛠️ 3. Running a File
            - 📚 4. Modules & `require()`
                - Built-in Modules
                - File System Module
                - Create Your Own Module
            - 🌐 5. Build a Simple HTTP Server
            - ⚙️ 6. `npm` and External Packages
                - Install Packages
                - Use in Your Project
            - 📩 7. Handling Requests in Express
            - 📂 8. Organizing Your Code (MVC Style)
            - 🧪 9. Debugging & Logging
            - 🔐 10. Environment Variables

    - 📌 11. Common Built-In Modules
    - 🧠 12. Key Concepts to Know
- 🚀 Express.js Essentials for Beginners
    - 📦 1. What is Express.js?
    - 🔧 2. Installation & Setup
    - 📬 3. HTTP Methods (GET, POST, PUT, DELETE)
    - 🧱 4. Route Parameters & Query Params
    - 🧪 5. Middleware Explained Simply
    - 📂 6. Organize Routes in Separate File
        - Step 1: Create `routes/userRoutes.js`
        - Step 2: Import in `index.js`
    - 🗒️ 7. Send JSON, Status, Headers
    - 🚨 8. Error Handling
    - ✔️ 9. Dev Tip: Use `nodemon` for Auto-Restart
    - 📦 10. Example JSON API Response
    - 💡 BONUS: Build This Mini Project
    - 📌 Summary Table
- Axios Essentials
    - What is Axios?
        - Installation
    - Common Use Cases
    - Basic Syntax
        - Using Promises
        - Using async/await
    - POST Request Example
    - Axios Configuration
        - Global Configuration
        - Per-request Configuration
    - Common Patterns
        - GET with Query Parameters
        - PUT to Update Data
        - DELETE a Resource
    - Axios Interceptors
        - Request Interceptor
        - Response Interceptor
    - Uploading Files
    - Using Axios in React
    - Common Errors and Solutions
    - Summary
- Getting Started with Create React App
    - Available Scripts
        - `npm start`
        - `npm test`
        - `npm run build`
        - `npm run eject`
    - Learn More

    - Code Splitting
    - Analyzing the Bundle Size
    - Making a Progressive Web App
    - Advanced Configuration
    - Deployment
    - `npm run build` fails to minify
- Postman APIs v2.2 load

---

3. Database Concepts

Relational (SQL)	**MongoDB (NoSQL)**
Database	Database
Tables	Collections
Rows	Documents
Columns	Fields
Foreign keys	References (using ObjectIDs)
JOINs	`$lookup` operator
Data on disk: tables	Data on disk: **BSON** (binary JSON) — supports strings, integers, dates, etc.

> BSON

- looks like json
- store in disc as bson format
- wide variety datatype supports
- allows nesting documents
- improves data integrity instead of splitting into different tables

> Example of a MongoDB document(row) Model and labels the different BSON data types used

```
{
  name: "Beau Carnes",                       // String
  title: "Developer & Teacher",             // String
  address: {                                // Nested Document
    address_1: "123 Main Street",           // String
    city: "Grand Rapids",                   // String
    state: "Michigan",                      // String
    postal_code: "49503"                    // String
  },
  topics: ["MongoDB", "Python", "JavaScript", "Robots"],  // Array of
Strings
  employee_number: 1234,                    // Integer
  location: [44.9901, 123.0262]             // Geo-Spatial Coordinates (Array
of Numbers)
```

```
}
```

> **Subdocuments** and **arrays** allow nesting related data together for faster access.

📚 MongoDB vs Mongoose — Explained Simply

Term	Excel Analogy	Simple Explanation
Database	A workbook (Excel file)	A container holding many collections (like multiple sheets in Excel)
Collection	A single sheet in Excel	A group of similar data (e.g., `users`, `products`)
Document	A single row in Excel	A single data entry (e.g., one user) stored in JSON format
Field	A cell or column label	A key/value pair inside a document (e.g., `username: "robin"`)
_id	Row number (auto)	Unique ID automatically given to every document
JSON / BSON	Structured data	Format MongoDB uses to store and exchange data (BSON = Binary JSON)

🛠 Mongoose Essentials

Term	Excel Analogy	Simple Explanation
Schema	Column layout & rules	Defines structure of documents (what fields, what type, required or not)
Model	Excel operations tool	A reusable object to create, read, update, delete documents in a collection
Instance / Object	A new row	A new document created from a model
Validator	Data rule checker	Ensures data is correct (e.g., username must be at least 3 characters)
Middleware	Auto-checklist before saving	Code that runs before or after saving documents (like pre-checks)
Population	VLOOKUP	Replace reference IDs with real documents (like showing full user info in an order)

🔄 CRUD Operations (Actions on Data)

Operation	Excel Action	MongoDB Method	What It Does
Create	Add new row	`insertOne`, `Model.create()`	Add a new document
Read	View rows	`find()`, `findOne()`	Fetch documents
Update	Edit a row	`updateOne`, `findByIdAndUpdate()`	Change data in a document
Delete	Remove row	`deleteOne`, `findByIdAndDelete()`	Remove a document

⚙️ Example in Code

```
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, minlength: 3 }
});

const User = mongoose.model('User', userSchema);

const newUser = new User({ username: 'robin' });
newUser.save(); // Adds new document to MongoDB
```

🧠 TL;DR Quick Summary Table

Concept	MongoDB	Mongoose
Database	✅	❌
Collection	✅	❌
Document	✅	✅
Schema	❌	✅
Model	❌	✅
CRUD operations	✅	✅ (simplified)
Validation	Basic	Powerful & easy
Relations	Manual refs	Population feature

Sure! Here's a beginner-friendly guide to **MongoDB + Mongoose** essentials with clear explanations, examples, and common use cases.

🧠 MongoDB + Mongoose Essentials

MongoDB is a **NoSQL database** that stores data in **flexible, JSON-like documents**.

- 🗒️ Document-based (not tables & rows like SQL)

- 🧵 Schema-less (but structure can be enforced with Mongoose)
- 📦 Stores data in BSON (Binary JSON)

```
{
  "_id": "abc123",
  "name": "Robin",
  "age": 22,
  "interests": ["coding", "music"]
}
```

🧩 What is Mongoose?

Mongoose is an **ODM (Object Data Modeling)** library for **Node.js** that makes it easier to interact with MongoDB:

- 🚧 Enforces schema
- 🔄 Provides CRUD methods
- ⚙️ Handles relationships, validation, middleware

🛠 Setup (Install)

```
npm install mongoose
```

```
// db.js
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/testdb', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

mongoose.connection.once('open', () => {
  console.log('✅ MongoDB connected');
});
```

Perfect! Here's your **final MongoDB + Mongoose Essentials cheat sheet** with a focus on:

✅ `find()` ✅ All CRUD operations ✅ Fully commented in **simple language** ✅ Beginner-friendly format

---

🧠 MongoDB + Mongoose Cheatsheet

1️⃣ Setup and Connect to MongoDB

```
// db.js
const mongoose = require('mongoose');

// Connect to MongoDB (change `testdb` to your DB name)
mongoose.connect('mongodb://localhost:27017/testdb', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

// Event listener for successful connection
mongoose.connection.once('open', () => {
  console.log('✅ Connected to MongoDB');
});
```

2️⃣ Create a Schema and Model

```
// models/User.js
const mongoose = require('mongoose');

// Create a schema to define structure of your "User" documents
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },     // name must be a string and
required
  age: Number,                                // age is optional and should
be a number
  email: { type: String, required: true },    // email is required
  isAdmin: { type: Boolean, default: false }, // if not provided, isAdmin
will be false
  createdAt: { type: Date, default: Date.now } // automatically stores
current time
});

// Create a model (acts like a class)
const User = mongoose.model('User', userSchema);

// Export the model to use in other files
module.exports = User;
```

3️⃣ Basic CRUD Operations

✅ Create a New User

```
const User = require('./models/User');

// Create a new user document
const newUser = new User({
  name: 'Alice',
  age: 25,
  email: 'alice@example.com',
});

// Save the user to MongoDB
await newUser.save(); // This actually writes the document into the
collection
```

🔍 Read / Find Users

```
// Find ALL users
const allUsers = await User.find(); // No filter = get everything
console.log(allUsers);

// Find users who are admins
const admins = await User.find({ isAdmin: true });

// Find one user by a specific field (returns first match)
const alice = await User.findOne({ name: 'Alice' });

// Find by ID (very common for updates/deletes)
const userById = await User.findById('64bc1234abc987xyz');

// Use query operators ($gte = greater than or equal)
const adults = await User.find({ age: { $gte: 18 } });
```

📝 **Important**: `find()` always returns an array; `findOne()` returns a single object or `null`.

✏️ Update User Data

```
// Update user by a field (name)
await User.updateOne(
  { name: 'Alice' },       // Filter
  { $set: { age: 26 } }    // What to update
);

// Find by ID and update directly
await User.findByIdAndUpdate('64bc1234abc987xyz', {
  $set: { isAdmin: true }
});
```

📝 Use `$set` to modify only selected fields.

❌ Delete User

```
// Delete by filter
await User.deleteOne({ name: 'Alice' });

// Delete by ID
await User.findByIdAndDelete('64bc1234abc987xyz');
```

4️⃣ Additional Concepts

🧪 Validation and Default Values

```
const schema = new mongoose.Schema({
  name: { type: String, required: true },  // Must be present
  age: { type: Number, min: 0 },           // Cannot be negative
  email: { type: String, unique: true },   // No duplicate emails
  isAdmin: { type: Boolean, default: false },
  createdAt: { type: Date, default: Date.now },
});
```

🔗 Relationships using `ref` and `populate`

```
// Post.js
const postSchema = new mongoose.Schema({
  title: String,
  content: String,
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' }
});
```

```
// To get post with full author info (populate = join)
const posts = await Post.find().populate('author');
```

🔁 Middleware (Hooks)

```
// Runs BEFORE saving a user
userSchema.pre('save', function (next) {
  console.log('About to save user:', this.name);
  next(); // continue saving
});
```

📌 Common Mistakes to Avoid

Mistake	Fix
Not using `await`	Use `await` with all async functions
Using `find()` but expecting object	Use `findOne()` or `findById()` if you want one document
Missing `.connect()`	Always call `mongoose.connect(...)` before using models
Forgetting `module.exports`	Always export your model so it can be used elsewhere

---

🧪 Sample Test Script

```
// test.js
const mongoose = require('mongoose');
const User = require('./models/User');

async function run() {
  try {
    await mongoose.connect('mongodb://localhost:27017/testdb');

    // Create a new user
    const user = new User({ name: 'Bob', age: 30, email: 'bob@example.com'
});
    await user.save();

    // Find users
    const users = await User.find();
    console.log('All Users:', users);

    // Update user
    await User.updateOne({ name: 'Bob' }, { $set: { age: 31 } });

    // Delete user
    await User.deleteOne({ name: 'Bob' });

  } catch (err) {
    console.error('❌ Error:', err.message);
  } finally {
    await mongoose.disconnect();
    console.log('🛑 Disconnected');
  }
```

```
}

run();
```

####### 📌 Summary

Concept	MongoDB	Mongoose
DB Type	NoSQL (Document-based)	ODM (Library for Node.js)
Data Format	BSON (like JSON)	Schema + Models
Connection	`mongodb://localhost:27017/dbname`	`mongoose.connect(...)`
CRUD	Native Mongo Shell or Driver	`Model.find()`, `Model.save()` etc
Relationships	Manual `$lookup`	`ref + populate()`

---

4. MongoDB Atlas Setup

1. **Sign in** to MongoDB Atlas at `https://cloud.mongodb.com`
2. Click **"New Project"** → name your project → **"Create Project"**
3. Click **"Build a Cluster"** → choose **Google Cloud Platform**, **Free Tier (M0 sandbox)**, select a region → **"Create Cluster"**
4. **Whitelist IP & Create DB User**:
    - In **Network Access**, add your current IP.
    - In **Database Access**, create a user & password.
5. **Get Connection String**:
    - Click **"Connect"** → **"Connect your application"** → copy the connection string.
    - It will look like:

      ```
      mongodb+srv://<username>:
      <password>@cluster0.mongodb.net/myFirstDatabase?
      retryWrites=true&w=majority
      ```

6. **Database & Collections** for this Projectapp:
    - **Collections**:
        - `users`,
        - `exercises`
        - (each exercise references one user)

ObjectIds in MongoDB

Here's the breakdown of a MongoDB ObjectId as shown:

```
ObjectId("5c2fc4b3 e52f37b7ee a58d00")

  5c2fc4b3   ← 4-byte UNIX timestamp
  e52f37b7ee ← 5-byte random value
  a58d00     ← 3-byte incremental counter
```

- auto generated by Mongodb driver
- is guranteed to be unique across each document in collection
- different part of object id represent different things

---

7. Frontend Setup (React) (Just Initialize)

7.1 Initialize React App

```
# from project root:
npx create-react-app mern-exercise-tracker
cd mern-exercise-tracker
npm install axios react-router-dom bootstrap react-datepicker
```

- `create-react-app` will generate default react project with default dependencies installed
- do `/node_modules` to `node_modules` in `.gitignore`

Remove unused files (logo, serviceWorker) and CSS imports.

5. Backend Setup (Node.js + Express + Mongoose)

5.1 Prerequisites

- **Node.js** installed (`node -v`)
- **npm** available
- **MongoDB Atlas** cluster ready

5.2 Initialize Backend Project

```
# you can either create backend folder inside the frontend directory or
make it seperate adjacent to it
mkdir backend && cd backend
npm init -y #create package.json
npm install express mongoose cors dotenv #backend packages
npm install -g nodemon #makes dev easier, tool to make NodeJS Applications
by automatically restarting the node application when files changes in
directory/detective
```

```
copy .gitignore of frontend to backend
```

nodemon

- makes dev easier,
- tool to make NodeJS Applications
- by automatically restarting the node application when files changes in directory/detective
- SO WHENEVER WE UPDATE OUR SERVER FILE, IT AUTOMATICALLY RESTARTS THE SERVER

> whenever to install globally ,use sudo

5.3 Create `backend/server.js` [BASIC, to Append on the way]

```
// server.js
// body parser not needed in new version of express

// import express framework and core middleware to enable cross-origin
requests
const express = require('express')
const cors = require('cors')

// load env var form .env into process.env
require('dotenv').config();


// create a new express app
const app = express();
// get port no. from environment or default 5000
const port = process.env.PORT || 5000;


// setup middleware
// load cors, now app allows requests from any origin
app.use(cors());

// load express, now app auto parse json payloads in incomin requests
// as our server is gonna send&receive JSONs
app.use(express.json()); // bodyparser is included in express

// start the server and listen on specific port
app.listen(port, ()=> {console.log(`Server Running SUCCESS at Port :
${port}`)})

// basic server ready :)
```

- run it

```

```

```
nodemon server
```

```
bali-king@war-machine:~/BaliGit/fullstack-webdev-essentials/MERN/backend$
nodemon server
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
[dotenv@17.2.1] injecting env (1) from .env -- tip: ⚙  suppress all logs
with { quiet: true }
Server Running SUCCESS at Port : 5000
```

- now we are ready to connect mongodb atlas with server

5.4 Environment Variable

Create `.env` in `/backend`:

```
ATLAS_URI=mongodb+srv://<username>:
<password>@cluster0.mongodb.net/exercise-tracker?
retryWrites=true&w=majority
```

> type in password and username :0 on : never store `.env` file in git repo or getfkd. put them in `.gitignore`

```
.env
.env.local
.env.development.local
.env.test.local
.env.production.local
```

Integrate Mongoose in `backend/server.js` to connect mongodb atlas with server

```
// server.js
// body parser not needed in new version of express

// import express framework and core middleware to enable cross-origin
requests
const express = require('express')
const cors = require('cors')

// ### Integrate Mongoose in `backend/server.js` to connect mongodb atlas
with server
```

```
// import mongoose for connecting to mongodb
const mongoose = require('mongoose')

// load env var form .env into process.env
require('dotenv').config();


// create a new express app
const app = express();
// get port no. from environment or default 5000
const port = process.env.PORT || 5000;


// setup middleware
// load cors, now app allows requests from any origin
app.use(cors());

// load express, now app auto parse json payloads in incomin requests
// as our server is gonna send&receive JSONs
app.use(express.json()); // bodyparser is included in express

// after setting up middleware
// load mongodb conn. string from .env
// we will get uri from mongodb atlas dashboard
// uri, where our db is stored
const uri = process.env.ATLAS_URI;

// tell mongoose to connect mongodb using that uri
// As of Mongoose 6.x, many connection options (including useCreateIndex,
useNewUrlParser, useUnifiedTopology, etc.) are now set by default or
deprecated.
mongoose.connect(uri)  .then(() => console.log("MongoDB db-connection est.
SUCCESS"))
  .catch((err) => console.error("MongoDB connection ERROR:", err));



// old way
// // tell mongoose to connect mongodb using that uri
// mongoose.connect(uri, { // passing uri

//     // extra flags, due to internal mongodb update guidelines
//     useNewUrlParser: true, // use the new URL parser instead of the
deprecated one
//     useCreateIndex: true // use createIndex() instead of ensureIndex()
// })

// grab default connection obj
const connection = mongoose.connection;


// old way to detect
// replaced by .then(() => console.log("MongoDB db-connection est.
SUCCESS")) .catch((err) => console.error("MongoDB connection ERROR:",
```

```
err));
// // listen,1st time the connection opens, log est success
// connection.once('open',
//     ()=>{
//         console.log("MongoDB db-connection Est. SUCCESS")
//     }
// )

// start the server and listen on specific port
app.listen(port, ()=> {console.log(`Server Running SUCCESS at Port :
${port}`)})

// server+mongoose server ready :)
```

```
bali-king@war-machine:~/BaliGit/fullstack-webdev-essentials/MERN/backend$
nodemon server
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
[dotenv@17.2.1] injecting env (1) from .env -- tip: ⚙  load multiple .env
files with { path: ['.env.local', '.env'] }
Server Running SUCCESS at Port : 5000
MongoDB db-connection est. SUCCESS
^Cbali-king@war-machine:~/BaliGit/fullstack-webdev-essentials/MERN/backend$
```

5.5 Mongoose Models

- now let's setup db
- let's make schema using mongoose
- let's make `backend/model` folder ,where we have `user.model.js`,`exercise.model.js`

> In Mongoose, a model is a blueprint for a collection in MongoDB.Once you define a model, you can use it to: Create, Read, Update, and Delete documents in that collection (CRUD) If MongoDB is like a database of Excel sheets, then a Mongoose model is like defining the columns and rules for one sheet.

`models/user.model.js`

- 1 field
- multiple validations

```
const mongoose = require('mongoose');// Import the mongoose library to work
with MongoDB
const Schema = mongoose.Schema;// Get the Schema constructor from mongoose
to define data structure
```

```
const userSchema = new Schema( // Define a new schema (structure) for a
"User" document
    // single field `username`

    {
        username :   // "username" field must be a string
        {

            // validators
            type: String, // Data type is String
            required: true, // This field is mandatory
            unique:true, // No two users can have the same username
            trim: true,  // Removes extra spaces at the beginning or end
            minlength: 3 // Must be at least 3 characters long
        }

    }
    ,
    {
        timestamps:true  // Automatically adds "createdAt" and "updatedAt"
fields
    }
);

const User = mongoose.model('User',userSchema); //mongoose.model() creates
a model named 'User' using the schema userSchema.
module.exports = User; //This line exports the User model so you can import
and use it in other files (e.g., routes or controllers).
```

models/exercise.model.js

- now lets exercise :0
- 4 fields
- less validations
- same as user.model.js

```
const mongoose = require('mongoose')
const Schema =  mongoose.Schema;

const exerciseSchema = new Schema(
    {
        username : {type: String, required: true},
        description: {type: String, required: true},
        duration: {type: Number, required:true},
        date:{type:Date, required:true},

    }
    ,
    {
        timestamps: true,
    }
```

```
);

const Exercise = mongoose.model('Exercise',exerciseSchema);
module.exports = Exercise;
```

- now models created

5.6 Routes

- now we have to add API endpoint routes
- so that server could do CRUD operations
- mkdir routes && cd routes
- touch exercises.js users.js
- before making it
- we are doing server work

importing and using `routes/users.js` & `routes/exercises.js` in `server.js`

```
// server.js
// body parser not needed in new version of express

// import express framework and core middleware to enable cross-origin
requests
const express = require('express')
const cors = require('cors')



// ### Integrate Mongoose in `backend/server.js` to connect mongodb atlas
with server
// import mongoose for connecting to mongodb
const mongoose = require('mongoose')

// load env var form .env into process.env
require('dotenv').config();


// create a new express app
const app = express();
// get port no. from environment or default 5000
const port = process.env.PORT || 5000;


// setup middleware
// load cors, now app allows requests from any origin
```

```
app.use(cors());

// load express, now app auto parse json payloads in incomin requests
// as our server is gonna send&receive JSONs
app.use(express.json()); // bodyparser is included in express

// after setting up middleware
// load mongodb conn. string from .env
// we will get uri from mongodb atlas dashboard
// uri, where our db is stored
const uri = process.env.ATLAS_URI;

// tell mongoose to connect mongodb using that uri
// As of Mongoose 6.x, many connection options (including useCreateIndex,
useNewUrlParser, useUnifiedTopology, etc.) are now set by default or
deprecated.
mongoose.connect(uri)  .then(() => console.log("MongoDB db-connection est.
SUCCESS"))
  .catch((err) => console.error("MongoDB connection ERROR:", err));



// old way
// // tell mongoose to connect mongodb using that uri
// mongoose.connect(uri, { // passing uri

//     // extra flags, due to internal mongodb update guidelines
//     useNewUrlParser: true, // use the new URL parser instead of the
deprecated one
//     useCreateIndex: true // use createIndex() instead of ensureIndex()
// })

// grab default connection obj
const connection = mongoose.connection;


// old way to detect
// replaced by .then(() => console.log("MongoDB db-connection est.
SUCCESS")) .catch((err) => console.error("MongoDB connection ERROR:",
err));
// // listen,1st time the connection opens, log est success
// connection.once('open',
//     ()=>{
//         console.log("MongoDB db-connection Est. SUCCESS")
//     }
// )



// #### importing and using `routes/users.js` & `routes/exercises.js` in
`server.js`
// just before app.listen()
// import routes
const exercisesRouter = require('./routes/exercises');
```

```
//exercisesRouter is now an Express router containing endpoints like POST,
GET, etc., for exercises.
const userRouter = require('./routes/users'); //usersRouter will handle
routes related to user operations like registration or listing users

app.use('/exercises', exercisesRouter); // Mount exercise routes at
/exercises
// This tells the Express app to use all routes from exercisesRouter, and
prefix them with /exercises.
// For example, if exercisesRouter has a GET / route, it will be available
at GET /exercises/.
app.use('/users',userRouter) // Mount user routes at /users


// start the server and listen on specific port
app.listen(port, ()=> {console.log(`Server Running SUCCESS at Port :
${port}`)})

// server+mongoose server ready :)
```

- don't run it till you makes routes :0

routes/users.js

```
const Express = require('express'); // import express

const router = Express.Router();
// NOT // const Router = Express.Router; as Router var is conflict Router
stuff // use router func to create route handlers
let User = require('../models/user.model'); // import model/mongoose schema

// 1st Route/endpoint that handles http get req/
// Route to GET Req. to fetch all users
router.route('/').get( //get request at index
    (req, res) =>
    {
        User.find()   // Find all user records from the database, method of
mongoose, it returns a promise
        .then(users => res.json(users))    // If successful, return the
users in JSON format
        .catch(err=> res.status(400).json('ERROR: '+err));     // If error
occurs, send a 400 status with the error message
    }
);

// req ,from Express, This object represents the incoming HTTP request.
// res ,from Express, This object is used to send the response back to the
client.

// Define a route for POST request to add a new user
```

```
router.route('/add').post(
    (req,res) => {
        const username = req.body.username;   // Extract username from the
request body
        // req.body.username: This gets the "username" field from the data
sent by the client (like a form or Postman)
        // For example: if you send { "username": "robin" }, this line will
store "robin" in the variable `username`
        const newUser = new User({username});   // Create a new User object
using the username
        // This creates a new object (document) using the User model
        // The new object looks like: { username: 'robin' }

        newUser.save()   // Save the new user to the database
        // .save(): This is a Mongoose method that saves the new user to
the MongoDB database

        .then(()=>res.json('User added!'))     // If successful, respond
with a success message
        .catch(err=> res.status(400).json('ERROR: '+err));     // If error
occurs, send a 400 status with the error message
    }


);

module.exports = router;
// Export the router so it can be used in other parts of the app
```

> req ,from Express, This object represents the incoming HTTP request. res ,from Express, This object is used to send the response back to the client. An HTTP request is a message sent by a client (like a browser, Postman, or frontend app) to a server, asking it to do something. Great question!

📦 **`req` (Request)**

- Contains information about the HTTP request made by the **client (browser, frontend app, etc)**.
- Includes data like:
  - `req.body` → data sent by the client (POST/PUT)
  - `req.params` → URL parameters (e.g., `/user/:id`)
  - `req.query` → query string (e.g., `?search=apple`)
  - `req.headers` → metadata about the request

👉 Example:

```
const username = req.body.username;
// This gets the 'username' field sent in the body of a POST request
```

- common req methods | Syntax | Purpose | | ---------- | ---------- | | `req.body` | Data sent in the body (POST, PUT) | | `req.params` | Route parameters (e.g., `/user/:id`) | | `req.query` | Query string (e.g., `/search?term=car`) | | `req.headers` | All the headers from the client | | `req.method` | HTTP method (GET, POST, etc.) | | `req.url` | Requested URL |

📤 `res` **(Response)**

- Used to **send back a response** to the client.
- Can be a:
  - JSON object → `res.json({ message: "Done" })`
  - Status code → `res.status(400)`
  - Plain text → `res.send("Hello")`

👉 Example:

```
res.json("User added!");
// This sends a JSON response back to the client
```

- common res methods

Syntax	Purpose
`res.send(data)`	Send a plain text or HTML response
`res.json(data)`	Send a JSON response
`res.status(code)`	Set HTTP status code (e.g., 200, 404, 500)
`res.redirect(url)`	Redirect to another URL
`res.end()`	End the response process

> arrow function is just shortcut to write typical function

```
// // Long Way (Traditional Function)
// function(req, res) {
//   // code
// }

// // Short Way (Arrow Function)
// (req, res) => {
//   // code
// }
```

routes/exercises.js

- similar theory to users.js
- but slight different code

```
// Import express and create a router object
const Express = require('express');

const router = Express.Router(); //() is imp as This will assign the Router
function itself, not an instance.
// NOT // const Router = Express.Router; as Router var is conflict Router
stuff
// Import the Exercise model
const Exercise = require('../models/exercise.model');

// MyChull :)
const INDEX = router.route('/');
const ADD = router.route('/add');
const ID = router.route('/:id');
const UPDATE_ID = router.route('/update/:id');

// GET all exercises
// Route: GET '/'
// Find all exercises and send as JSON
INDEX.get(function(req,res){
    Exercise.find()
    // Non Arrow function MyChull :)
    .then(function(exercises){
        return res.json(exercises)
    })
    .catch(err=> res.status(400).json('ERROR: '+err));     // If error
occurs, send a 400 status with the error message

});

// POST: Add a new exercise
// Route: POST '/add'
// Extract username, description, duration, date from req.body
// Create a new Exercise object
// Save it to database
// Send success or error response
ADD.post(function(req,res){

    // MyCHull :)
    const REQ = req.body;
    const username = REQ.username,
    description = REQ.description,
    duration = Number(REQ.duration), // Number() Convert duration to a
number

    date = Date.parse(REQ.date); // Date.parse() Convert date string to
```

```
Date format
    // cleaner way
    // const {username,description,duration,date} = req.body;

    // Destructure fields from the request body (sent by the client)
    const newExercise = new Exercise (
        {
            username,
            description,
            duration,
            date
        }

    );

    newExercise.save() //save new exercise to mongodb
    .then(function(){res.json("Exercise added !")})
    .catch(function(err){res.status(400).json('Error: '+err)})
    ;

});


// GET: Fetch single exercise by ID
// Route: GET '/:id'
// Find exercise by ID from URL
// Return exercise or send error
ID.get(function(req,res){ //function(res, req) { ... },no swap req,res ✖
ORDER MATTERS, TYPICAL READ OF ARGUEMENTS
    Exercise.findById(req.params.id)// Exercise is mongoose model
representin Exercise colleciton in MongoDB db
    // findById is a Mongoose method It searches for a document by its
unique _id field (the default MongoDB ID for every document).
    .then(function(Exercise){res.json(Exercise)})
    .catch(function(err){res.status(400).json('Error:'+ err)})
    ;

});


// DELETE: Remove exercise by ID
// Route: DELETE '/:id'
// Delete the exercise by ID
// Send success or error response
ID.delete(function(
    req,res
){
    Exercise.findByIdAndDelete(
        req
        .params
        .id
    )// Delete exercise with matching ID
    .then(() => res.json('Exercise deleted !'))  // Respond with success
message
```

```
    .catch(err => res.status(400).json('Error: ' + err));  // Handle error


}
);


// POST: Update an existing exercise
// Route: POST '/update/:id'
// Find exercise by ID
// Update its fields with values from req.body
// Save the updated exercise
// Send success or error response
UPDATE_ID.post((req,res)=>{
    Exercise.findById(req.params.id)
    .then(
        function(exercise){
            exercise.username = req.body.username;
            exercise.description = req.body.description;
            exercise.duration = Number(req.body.duration);
            exercise.date = Date.parse(req.body.date);

            exercise.save()
            .then(() => res.json('Exercise Updated !'))  // Respond with
success message
            .catch(err => res.status(400).json('Error: ' + err));  //
Handle error

        }
    )
    .catch(err => res.status(400).json('Error: ' + err));  // Handle error

});



// Export the router so it can be used in server.js
module.exports = router;
```

Params

- parameters
- special rewuest in express request
- stores route parameters from URL Path
- access through `req.params`
- basically access stuff from url :0
- now lets test our APIs

6. Testing API (Insomnia / Postman)

- open API Tester
- creat collection
- make request
- shoot it :0
- SUCCESS MEANS `200`

1. **POST** `http://localhost:5000/users/add`

```
{ "username": "Momotaro" }
```

output : `"User added!"`

2. **GET** `http://localhost:5000/users`

output :


```
[
  {
      "_id": "688a6c7c2ea7acb852ea682a",
      "username": "momotaro",
      "createdAt": "2025-07-30T19:03:24.994Z",
      "updatedAt": "2025-07-30T19:03:24.994Z",
      "__v": 0
  },
  {
      "_id": "688a6cf72ea7acb852ea682c",
      "username": "bali",
      "createdAt": "2025-07-30T19:05:27.299Z",
      "updatedAt": "2025-07-30T19:05:27.299Z",
      "__v": 0
  },
  {
      "_id": "688a6cfb2ea7acb852ea682e",
      "username": "bhati",
      "createdAt": "2025-07-30T19:05:31.711Z",
      "updatedAt": "2025-07-30T19:05:31.711Z",
      "__v": 0
  },
  {
      "_id": "688a6d012ea7acb852ea6830",
      "username": "bhaskar",
      "createdAt": "2025-07-30T19:05:37.677Z",
      "updatedAt": "2025-07-30T19:05:37.677Z",
      "__v": 0
  }
]
```

- mongodb auto created _id, and other stuff
- also we save to db !!
- check from mongodb>atlas>exercise-tracker>collections
- refresh if any lag

3. **POST** `http://localhost:5000/exercises/add`

```
{
  "username":"bali",
  "description":"Jumping Jacks",
  "duration": "9" ,
  "date":"2025-08-30T19:03:24.994Z"
}
```

output : `"Exercise added !"`

4. **GET** `http://localhost:5000/exercises`

output :


```
[
  {
      "_id": "688a6fee71069ae17aa9e424",
      "username": "bali",
      "description": "Mountain Climbing",
      "duration": 9,
      "date": "2025-08-30T19:03:24.994Z",
      "createdAt": "2025-07-30T19:18:06.596Z",
      "updatedAt": "2025-07-30T19:18:06.596Z",
      "__v": 0
  },
  {
      "_id": "688a6ffb71069ae17aa9e426",
      "username": "bali",
      "description": "Jumping Jacks",
      "duration": 9,
      "date": "2025-08-30T19:03:24.994Z",
      "createdAt": "2025-07-30T19:18:19.287Z",
      "updatedAt": "2025-07-30T19:18:19.287Z",
      "__v": 0
  },
  {
      "_id": "688a707071069ae17aa9e428",
      "username": "bhati",
      "description": "Deadlift",
      "duration": 11,
      "date": "2025-08-30T19:03:24.994Z",
      "createdAt": "2025-07-30T19:20:16.402Z",
      "updatedAt": "2025-07-30T19:20:16.402Z",
```

```
        "__v": 0
    }
]
```

5. **GET** `http://localhost:5000/exercises/<id>`

    - object id auto created by mongodb
    - accessing `/:id` object id from database, then it will return that info only, directly access stuff from putting id in url
    - eg: `http://localhost:5000/exercises/688a6fee71069ae17aa9e424`
    - output :

```
{
"_id": "688a6fee71069ae17aa9e424",
"username": "bali",
"description": "Mountain Climbing",
"duration": 9,
"date": "2025-08-30T19:03:24.994Z",
"createdAt": "2025-07-30T19:18:06.596Z",
"updatedAt": "2025-07-30T19:18:06.596Z",
"__v": 0
}
```

6. **POST** `http://localhost:5000/exercises/update/<id>` (with updated fields)

    - eg: `http://localhost:5000/exercises/update/688a6fee71069ae17aa9e424`
    - &

```
        {
    "username": "bhaskar",
    "description": "Mountain Climbing",
    "duration": 9,
    "date": "2025-08-30T19:03:24.994Z"
    }
```

    - Output : `"Exercise Updated !"`
    - YOU ALWAYS HAVE TO SEND PROPER ALL FIELDS,NOT PARTIAL OF ANY KIND, NOTHING LESS, NOTHING MORE , JUST EXACT !!!!!!

7. **DELETE** `http://localhost:5000/exercises/<id>`

    - literally same as update , just `POST -> DELETE`
    - eg: `http://localhost:5000/exercises/688a6fee71069ae17aa9e424`, not exercises/delete/:id
    - &

```
        {
      "username": "bhati",
      "description": "Deadlift",
      "duration": 11,
      "date": "2025-08-30T19:03:24.994Z"
  }
```

  - Output : `"Exercise Deleated !"`
  - YOU ALWAYS HAVE TO SEND PROPER ALL FIELDS,NOT PARTIAL OF ANY KIND, NOTHING LESS, NOTHING MORE , JUST EXACT !!!!!!

- all stuff sync with mongodb database !!
- you can directly do changes in MongoDB Atlas Panel !!
- now let's jump to frontend.
- good job :0

---

7. Frontend Setup (React) (Actual Code Start)

React

- Declarative, efficient, flexible JS Lib. for building UIs.
- It lets you compose complex UIs from small and isolated pieces of code called COMPONENTs.
- we use components to tell react, what we want to see the screen.
- when our data change, react will effiently update and re-render our components.
- components takes in parametet called PROPs(Properties)
- and it return a hierarchy of views to display throught the render method
- RENDER METHOD returns a description of what you want to see on screen as JSX(React Element, Special Syntax)
- JSX converted to Html before it goes throught preprocessing
- JSX Comes with full power of JS under {}
- JSX have className ,not class in HTML
- REACT SAMPLE COMPONENT CODE :

```
// Define a class component called ShoppingList which inherits from
React.Component
class ShoppingList extends React.Component {

  // Define the render method - it returns JSX to be displayed
  render() {
    return (
```

```
      // Main wrapper div with a class name for styling
      <div className="shopping-list">

        {/* Display a heading with dynamic name passed via props */}
        <h1>Shopping List for {this.props.name}</h1>

        {/* Unordered list of shopping items */}
        <ul>
          <li>Cereal</li>    // First item in the list
          <li>Milk</li>      // Second item in the list
          <li>Bananas</li>   // Third item in the list
        </ul>
      </div>
    );
  }
}

// Example usage: <ShoppingList name="Beau" />
// This renders: Shopping List for Beau with 3 items
```

7.2 public/index.html

- this HTML Loads when we go to website
- at here `<div id="root"></div>` our react app Loads.

- Change `<title>` to **Exercise Tracker**
- Ensure `<div id="root"></div>` remains

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed
on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the
```

```
build.
      Only files inside the `public` folder can be referenced from the
HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico"
will
      work correctly both with client-side routing and a non-root public
URL.
      Learn how to configure a non-root public URL by running `npm run
build`.
    -->
    <title>Exercise Tracker</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

7.3 src/index.js

- it's get loaded when we go to website
- imports React,ReactDOM,./index.css,App,etc.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));// from
public/index.html <div id="root"></div>

root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
```

```
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

7.4 src/App.js

- main react app where we're gonna put all of our code that will display on the page
- SAMPLE :

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

- run it by `npm start`

```
Compiled successfully!

You can now view mern-exercise-tracker in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.9:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

- Edit src/App.js and save to reload.
- its Loaded on `http://localhost:3000`
- bootstrap: UI Lib
- react-router-dom: makes easy to route specific UR: paths to different react components that will load on the page.
- make sandwich of `<Router> </Router>`
- `<Route/>`elemenent is used to route each element in application ,eg: in this `<Route path="/edit/:id" component= {CreateExercise}>` if we go to this path`/edit/:id` then it will LOAD this component `CreateExercise`
- MAIN CODE

```
import React from "react";// Importing the React library to use JSX and
component features
import 'bootstrap/dist/css/bootstrap.min.css';// Importing Bootstrap CSS
for styling (you get buttons, layouts, etc. pre-designed)
import {BrowserRouter, Route,Routes} from 'react-router-dom';// Importing
tools from React Router for handling navigation between pages
// Importing the components used in the app
import Navbar from './components/Navbar.component'; // Top navigation bar
import ExerciseList from './components/ExerciseList.component'; // Homepage
- shows list of exercises
import CreateExercise from './components/CreateExercise.component'; //
Pageto create a new exercise
import EditExercise from './components/EditExercise.component'; // Page
toedit an existing exercise
import CreateUser from './components/CreateUser.component'; // Page
tocreate a new user
import EditUserComponent from "./components/EditUser.component";
import UserList from "./components/User.component";
function App()// Main App component
{
  return(// Wraps the entire application inside the BrowserRouter so
routing can work
    <BrowserRouter>
      <div className="container">{/* This div gives some padding and
centers content using Bootstrap's'container' class */}
        <Navbar />{/* The Navbar is always shown at the top of the page */}
        <br/>{/* Just a line break for spacing */}
        {/* Define routes for different URLs and their
correspondingcomponents */}
        <Routes>
        <Route path="/users" Component={UserList} />
        <Route path="/users/update/:username" Component={EditUserComponent}
/>


        <Route path="/" exact Component={ExerciseList} />{/* If the URL is
```

```
exactly '/', show the ExerciseList component(i.e., the homepage) */}
        <Route path="/edit/:id" Component={EditExercise} />{/* If the URL
is '/edit/:id', show the EditExercise component. The ':id' is a dynamic
part - it will be replaced by an actual exercise ID */}
        <Route path="/create" Component={CreateExercise}/>{/* If the URL is
'/create', show the CreateExercise component */}
        <Route path="/user"Component={CreateUser}/>{/* If the URL is
'/user', show the CreateUser component */}
        </Routes>
      </div>
    </BrowserRouter>
  );
}

export default App;// Export the App component so it can be used in
index.js to render the whole app
```

- now we will make components in `src/components`

```
cd src
mkdir components
cd components
touch Navbar.component.js CreateUser.component.js
CreateExercise.component.js ExerciseList.component.js
EditExercise.component.js
```

7.5 Components

- inside `src/components` folder
- naming convention `componentname.component.js`
- `rafce` is snippet for ES7+ React Dev, its `React Arrow Function Component Export`, just type and tab

```
import React from 'react';

const ComponentName = () => {
  return ( // render func. inbuilt
    <div>
      ComponentName
    </div>
  );
};

export default ComponentName;
```

- old/classic way of creating a React component using a class and render() function

```
import React, { Component } from 'react';

class ComponentName extends Component {
  render() {
    return (
      <div>
        ComponentName
      </div>
    );
  }
}

export default ComponentName;
```

BootStrap Essentials

Bootstrap is a popular open-source CSS framework developed by Twitter. It helps you build responsive, mobile-first websites quickly with pre-designed components and utility classes.

- Current version: Bootstrap 5.x
- No jQuery required in Bootstrap 5+
- Includes: Grid system, Components, Utilities, JavaScript plugins

1. Getting Started

Include Bootstrap via CDN:

```
<!-- In the <head> -->
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.c
ss" rel="stylesheet">

<!-- Before </body> -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.
min.js"></script>
```

2. Layout: Grid System

```
<div class="container">
  <div class="row">
    <div class="col-6">Left</div>
    <div class="col-6">Right</div>
```

```
  </div>
</div>
```

3. Buttons

```
<button class="btn btn-primary">Primary</button>
<button class="btn btn-secondary">Secondary</button>
<button class="btn btn-outline-danger">Outline Danger</button>
```

4. Form Input

```
<form>
  <div class="mb-3">
    <label for="exampleInput" class="form-label">Name</label>
    <input type="text" class="form-control" id="exampleInput"
placeholder="Enter your name">
  </div>
</form>
```

5. Navbar

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">MySite</a>
  </div>
</nav>
```

6. Card

```
<div class="card" style="width: 18rem;">
  <img src="image.jpg" class="card-img-top" alt="...">
  <div class="card-body">
    <h5 class="card-title">Card Title</h5>
    <p class="card-text">Some quick example text.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
</div>
```

7. Utility Classes

```

```

```
<p class="text-center fw-bold text-success">Centered bold green text</p>
<div class="m-4 p-2 bg-light">Box with margin and padding</div>
```

8. Responsive Visibility

```
<div class="d-none d-md-block">Visible on md and larger</div>
<div class="d-block d-md-none">Visible only on small screens</div>
```

9. Container Types

```
<div class="container">Fixed-width container</div>
<div class="container-fluid">Full-width container</div>
```

7.5.1 src/components/Navbar.component.js

```
// Import React and Component class from 'react' so we can define a class
component
import React, { Component } from 'react';

// Import Link from react-router-dom for client-side navigation (without
reloading the page)
import { Link } from 'react-router-dom';

// Define and export a Navbar class component
export default class Navbar extends Component {
  // The render method returns the JSX that will be displayed
  render() {
    return (
      // This is a Bootstrap-styled navigation bar (dark background and
expands on larger screens)
      <nav className="navbar navbar-dark bg-dark navbar-expand-lg">

        {/* The brand/logo part of the navbar, clicking it takes user to
the home page ("/") */}
        <Link to="/" className="navbar-brand">Exercise Tracker</Link>

        {/* Container for the navigation links - Note: small typo in
'collapse' fixed below */}
        <div className="collapse navbar-collapse">

          {/* Unordered list to group navigation items */}
          <ul className="navbar-nav mr-auto">

            {/* First navigation item: Link to home page, shows "Exercises"
*/}
```

```
            <li className="navbar-item">
              <Link to="/" className="nav-link">Exercises</Link>
            </li>

            {/* Second navigation item: Link to create a new exercise */}
            <li className="navbar-item">
              <Link to="/create" className="nav-link">Create
Exercise</Link>
            </li>

            {/* Third navigation item: Link to create a new user */}
            <li className="navbar-item">
              <Link to="/user" className="nav-link">Create User</Link>
            </li>

          </ul>
        </div>
      </nav>
    );
  }
}
```

7.5.2 src/components/CreateUser.component.js

```
// Importing React and Component so we can use class-based components
import React, { Component } from 'react';

// Import axios for sending HTTP requests (used to talk to the backend
server)
import axios from 'axios';

// CreateUser component: lets users create a new username and save it to
the backend
export default class CreateUser extends Component {
  // Constructor: initializes the component with default state and binds
methods
  constructor(props) {
    super(props);

    // Component state to store the current value of the username input
field
    this.state = { username: '' };

    // Bind 'this' to event handler methods so they work correctly inside
the class
    this.onChangeUsername = this.onChangeUsername.bind(this);
    this.onSubmit = this.onSubmit.bind(this);
  }

  // Event handler: updates state when the input value changes
```

```
  onChangeUsername(e) {
    this.setState({ username: e.target.value }); // e.target.value is the
typed input
  }

  // Event handler: runs when the form is submitted
  onSubmit(e) {
    e.preventDefault(); // Prevents default page reload behavior of a form

    // Create a user object from the current state
    const user = { username: this.state.username };

    // Log the user to the console (for debugging)
    console.log(user);

    // Send a POST request to the backend server to add this user
    axios.post('http://localhost:5000/users/add', user)
      .then(res => console.log(res.data)); // Logs the server response to
console

    // Reset the form by clearing the username from the state
    this.setState({ username: '' });
  }

  // The render() method defines the UI that gets displayed on the screen
  render() {
    return (
      <div>
        <h3>Create New User</h3>

        {/* Form element to capture the username */}
        <form onSubmit={this.onSubmit}>

          {/* Bootstrap form group for styling */}
          <div className="form-group">
            <label>Username:</label>
            <input
              type="text"                 // Input field type
              required                    // Makes the field mandatory
              className="form-control"   // Bootstrap class for styling
input
              value={this.state.username} // Controlled input value
              onChange={this.onChangeUsername} // Update state when user
types
            />
          </div>

          {/* Submit button to send form data */}
          <div className="form-group">
            <input
              type="submit"
              value="Create User"
              className="btn btn-primary" // Bootstrap button style
            />
```

```
          </div>

        </form>
      </div>
    );
  }
}
```

What the heck was that

Alright — let's **strip it down to the absolute basics** and **explain line-by-line** what's happening in a very beginner-friendly way.

🧠 **GOAL: A simple React app that:**

✅ Lets you type a username in a box ✅ Click a button to send that to the backend server ✅ Shows what you typed in the console ✅ And explains **WTF is happening step-by-step**

📦 **Step 1: Simple `CreateUser.js`**

```
import React, { Component } from 'react';
import axios from 'axios';

export default class CreateUser extends Component {
  constructor(props) {
    super(props);

    // We want to keep track of the username that user types
    this.state = {
      username: ''
    };

    // These lines make sure that `this` inside methods refers to the class
    this.onChangeUsername = this.onChangeUsername.bind(this);
    this.onSubmit = this.onSubmit.bind(this);
  }

  // This function runs whenever you type something in the input box
  onChangeUsername(e) {
    // e.target.value is the text you typed
    this.setState({
      username: e.target.value
    });
  }

  // This function runs when you click the submit button
```

```
  onSubmit(e) {
    e.preventDefault(); // This stops the page from reloading

    const user = {
      username: this.state.username
    };

    console.log('Submitting:', user);

    // Sends that username to the backend at
http://localhost:5000/users/add
    axios.post('http://localhost:5000/users/add', user)
      .then(res => console.log('Server response:', res.data));

    // After submit, clear the input box
    this.setState({
      username: ''
    });
  }

  // This is what shows up on the webpage
  render() {
    return (
      <div>
        <h3>Create New User</h3>
        <form onSubmit={this.onSubmit}>
          <div>
            <label>Username: </label>
            <input
              type="text"
              required
              value={this.state.username}
              onChange={this.onChangeUsername}
            />
          </div>
          <div>
            <input type="submit" value="Create User" />
          </div>
        </form>
      </div>
    );
  }
}
```

🧠 **EXPLANATION: What the hell is happening**

####### 🛠 1. React Component Structure

```
export default class CreateUser extends Component
```

You are **creating a UI component** called `CreateUser`. It's a "class component" — like a small machine that:

- Maintains state (like memory)
- Handles user interaction (like typing or clicking)
- Renders stuff to the screen

####### 🔒 2. `constructor()`

```
constructor(props) {
  super(props);
  this.state = { username: '' };
  this.onChangeUsername = this.onChangeUsername.bind(this);
  this.onSubmit = this.onSubmit.bind(this);
}
```

- `this.state` = memory. It remembers what the user typed.
- `bind()` = glue functions to this class so they don't lose context

---

####### ⌨️ 3. `onChangeUsername(e)`

```
this.setState({ username: e.target.value });
```

- Whenever the user types something in the input box:
    - `e.target.value` is the typed text
    - We save that text in `this.state.username`

####### 📨 4. `onSubmit(e)`

```
e.preventDefault();
const user = { username: this.state.username };
axios.post('http://localhost:5000/users/add', user)
```

- When user **clicks the button**, this function:
    - Stops the form from refreshing
    - Creates an object like `{ username: "yourname" }`
    - Sends it to your backend using `axios`

####### 📺 5. `render()`

```
```

```
<form onSubmit={this.onSubmit}>
  <input type="text" value={this.state.username} onChange=
{this.onChangeUsername} />
```

- This is what **shows up on your webpage**
- Input box shows current value from state
- When user types → triggers `onChangeUsername`
- When user submits → triggers `onSubmit`

🔄 How this flows in real life

1. 🧍 User opens the page
2. 💬 User types "hello"
3. ✅ `onChangeUsername()` updates state to `{ username: "hello" }`
4. 👆 User clicks "Create User"
5. 🎯 `onSubmit()` runs
    - It reads `this.state.username` → makes `{ username: "hello" }`
    - Sends it to backend (`POST /users/add`)
6. 🧽 Resets the input box

🔧 Your Backend Should Handle This Route

```
// Example Express backend (Node.js)
app.post('/users/add', (req, res) => {
  const username = req.body.username;
  console.log("New user:", username);
  res.send("User added!");
});
```

🧪 How to test it:

1. Run your backend on `localhost:5000`
2. Run your React frontend
3. Go to `/create-user` page
4. Type any name

5. Click submit
6. Check:
    - Browser console: logs `{ username: "..." }`
    - Backend console: logs `"New user: ..."`

New Way

```
// Import necessary modules
import React, { useState } from 'react';
import axios from 'axios';

const CreateUser = () => {
  // Declare state for username
  const [username, setUsername] = useState('');

  // Handle input change
  const handleUsernameChange = (e) => {
    setUsername(e.target.value);
  };

  // Handle form submit
  const handleSubmit = async (e) => {
    e.preventDefault();

    const user = { username };
    console.log(user); // For debugging

    try {
      const res = await axios.post('http://localhost:5000/users/add',
user);
      console.log(res.data); // Log response
      setUsername(''); // Clear input after successful post
    } catch (err) {
      console.error('Error adding user:', err);
    }
  };

  return (
    <div>
      <h3>Create New User</h3>
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label>Username:</label>
          <input
            type="text"
            required
            className="form-control"
            value={username}
            onChange={handleUsernameChange}
```

```
          />
        </div>
        <div className="form-group">
          <input
            type="submit"
            value="Create User"
            className="btn btn-primary"
          />
        </div>
      </form>
    </div>
  );
};

export default CreateUser;
```

7.5.3 src/components/CreateExercise.component.js

```
// Import React and Component for class-based component
import React, { Component } from 'react';

// Import a calendar date-picker component and its CSS for selecting dates
import DatePicker from 'react-datepicker';
import 'react-datepicker/dist/react-datepicker.css';

// Import axios to send HTTP requests (e.g., to fetch or save data)
import axios from 'axios';

// Define and export the CreateExercise component
export default class CreateExercise extends Component {
  constructor(props) {
    super(props);

    // Initialize the component's state
    this.state = {
      username: '',       // Selected username from dropdown
      description: '',    // Text about the exercise
      duration: 0,        // Duration in minutes
      date: new Date(),   // Date of exercise
      users: []           // List of all usernames fetched from server
    };

    // Bind the 'this' keyword to all methods so they work correctly in
class component
    this.onChangeUsername = this.onChangeUsername.bind(this);
    this.onChangeDescription = this.onChangeDescription.bind(this);
    this.onChangeDuration = this.onChangeDuration.bind(this);
    this.onChangeDate = this.onChangeDate.bind(this);
    this.onSubmit = this.onSubmit.bind(this);
  }
```

```
  // Lifecycle method: runs after the component is first added to the page
  componentDidMount() {
    // Fetch the list of users from the backend
    axios.get('http://localhost:5000/users/')
      .then(res => {
        // If users exist, update the state with usernames
        if (res.data.length > 0) {
          this.setState({
            users: res.data.map(user => user.username), // Get only
usernames
            username: res.data[0].username // Set default selected username
          });
        }
      })
      .catch(err => console.log(err)); // Log any error
  }

  // Method to update state when username is changed from dropdown
  onChangeUsername(e) {
    this.setState({ username: e.target.value });
  }

  // Method to update description text
  onChangeDescription(e) {
    this.setState({ description: e.target.value });
  }

  // Method to update duration
  onChangeDuration(e) {
    this.setState({ duration: e.target.value });
  }

  // Method to update the selected date
  onChangeDate(date) {
    this.setState({ date }); // Shorthand for { date: date }
  }

  // Method called when the form is submitted
  onSubmit(e) {
    e.preventDefault(); // Prevent the form from refreshing the page

    // Create an object with current form values
    const exercise = {
      username: this.state.username,
      description: this.state.description,
      duration: this.state.duration,
      date: this.state.date
    };

    console.log(exercise); // Log it to the browser console for debugging

    // Send this data to the backend to store the new exercise log
    axios.post('http://localhost:5000/exercises/add', exercise)
```

```
        .then(res => console.log(res.data)); // Log the response

      // Redirect to home page after submission
      window.location = '/';
    }

    render() {
      return (
        <div>
          <h3>Create New Exercise Log</h3>

          {/* Form for adding a new exercise */}
          <form onSubmit={this.onSubmit}>

            {/* Dropdown to choose a user */}
            <div className="form-group">
              <label>Username: </label>
              <select required className="form-control"
                value={this.state.username}
                onChange={this.onChangeUsername}>
                {/* Loop through the users list and show each as an option
*/}
                {this.state.users.map(user => (
                  <option key={user} value={user}>{user}</option>
                ))}
              </select>
            </div>

            {/* Input for description */}
            <div className="form-group">
              <label>Description: </label>
              <input
                type="text"
                required
                className="form-control"
                value={this.state.description}
                onChange={this.onChangeDescription}
              />
            </div>

            {/* Input for duration */}
            <div className="form-group">
              <label>Duration (in minutes): </label>
              <input
                type="number"
                className="form-control"
                value={this.state.duration}
                onChange={this.onChangeDuration}
              />
            </div>

            {/* Date picker for selecting the date */}
            <div className="form-group">
              <label>Date: </label>
```

```
            <div>
              <DatePicker
                selected={this.state.date}
                onChange={this.onChangeDate}
              />
            </div>
          </div>

          {/* Submit button */}
          <div className="form-group">
            <input
              type="submit"
              value="Create Exercise Log"
              className="btn btn-primary"
            />
          </div>

        </form>
      </div>
    );
  }
}
```

- created constructer
- super props call
- `this.state` is like initialize a var in react{no let name!!!}
- whenever you update state ,then it auto relods webpage with updated values
- for `users:[]`, to show drop-down property
- now create methods
- `this.state.username` = "bali" NOOO, use `this.setState()` method for update in react, it updates specific property of state
- `e` The event object (e.g., input change event), it can be named anything. e is the event object automatically passed by the browser when an event occurs.
- `onSubmit` method last ;0
- `window.location='/';` used to direct the page ;0
- now we are using `this` inside functions
- and now we have to map like `this.onChangeDescription = this.onChangeDescription.bind(this);`
- `componentDidMount` is React's life cycle method that react will automatically call at different points, it will first called at the link right before anything displays on page.

- then rest of form/render stuff ;0
- to put js in b/w standard HTML code, use {}
- new way

```
// Import React and hooks needed for managing state and side effects
import React, { useState, useEffect } from 'react';

// Import the DatePicker component and its styles
import DatePicker from 'react-datepicker';
import 'react-datepicker/dist/react-datepicker.css';

// Import axios to make HTTP requests to the backend server
import axios from 'axios';

// Define the CreateExercise functional component
const CreateExercise = () => {
  // useState hook sets up state variables (like this.state in class
components)

  const [username, setUsername] = useState('');        // Selected user
from dropdown
  const [description, setDescription] = useState('');  // Description text
for exercise
  const [duration, setDuration] = useState(0);         // Duration in
minutes
  const [date, setDate] = useState(new Date());        // Selected date
  const [users, setUsers] = useState([]);              // List of usernames
for dropdown

  // useEffect hook runs after the component is mounted (like
componentDidMount)
  useEffect(() => {
    // Fetch list of users from the backend
    axios.get('http://localhost:5000/users/')
      .then(res => {
        // If response has users, set the state for users list and default
username
        if (res.data.length > 0) {
          setUsers(res.data.map(user => user.username));  // Extract only
usernames
          setUsername(res.data[0].username);              // Set first user
as default
        }
      })
      .catch(err => console.error(err)); // Log any error
  }, []); // Empty dependency array means this runs once when component
mounts

  // Function to handle form submission
  const handleSubmit = (e) => {
    e.preventDefault(); // Prevent page refresh on form submit
```

```
    // Create an object with the current form values
    const exercise = {
      username,
      description,
      duration,
      date
    };

    console.log(exercise); // Log for debugging in browser console

    // Send a POST request to backend to add new exercise
    axios.post('http://localhost:5000/exercises/add', exercise)
      .then(res => console.log(res.data))  // Log success response
      .catch(err => console.error(err));   // Log error if any

    // Redirect user to home page (optional)
    window.location = '/';
  };

  // JSX for rendering the form
  return (
    <div>
      <h3>Create New Exercise Log</h3>

      {/* Main form for creating an exercise */}
      <form onSubmit={handleSubmit}>

        {/* Dropdown for selecting username */}
        <div className="form-group">
          <label>Username:</label>
          <select
            required
            className="form-control"
            value={username}
            onChange={(e) => setUsername(e.target.value)}  // Update state
on change
          >
            {/* Loop through users array to show each option */}
            {users.map(user => (
              <option key={user} value={user}>
                {user}
              </option>
            ))}
          </select>
        </div>

        {/* Text input for exercise description */}
        <div className="form-group">
          <label>Description:</label>
          <input
            type="text"
            required
            className="form-control"
```

```
            value={description}
            onChange={(e) => setDescription(e.target.value)} // Update on
change
          />
        </div>

        {/* Number input for duration */}
        <div className="form-group">
          <label>Duration (in minutes):</label>
          <input
            type="number"
            required
            className="form-control"
            value={duration}
            onChange={(e) => setDuration(e.target.value)} // Update on
change
          />
        </div>

        {/* Calendar component for picking date */}
        <div className="form-group">
          <label>Date:</label>
          <DatePicker
            selected={date}              // Current selected date
            onChange={(date) => setDate(date)} // Update on selection
          />
        </div>

        {/* Submit button */}
        <div className="form-group">
          <input
            type="submit"
            value="Create Exercise Log"
            className="btn btn-primary"
          />
        </div>
      </form>
    </div>
  );
};

// Export the component so it can be used in the app
export default CreateExercise;
```

7.5.4 src/components/ExerciseList.component.js

```
// Importing React and Component class to create a class-based component
import React, { Component } from 'react';

// Importing Link from react-router-dom to enable client-side navigation
```

```
(edit links)
import { Link } from 'react-router-dom';

// Importing axios to make HTTP requests (GET, DELETE)
import axios from 'axios';

////////////////////////////////////////////////////////////////////////////
///
// Functional Component: Exercise
// This is a **stateless functional component** used to render each
exercise row.
// It receives props (data + delete function) and displays the fields in a
table row.
////////////////////////////////////////////////////////////////////////////
///
const Exercise = (props) => (
  <tr>
    <td>{props.exercise.username}</td>               {/* Username of person
who logged exercise */}
    <td>{props.exercise.description}</td>            {/* Description of the
exercise (e.g. Running) */}
    <td>{props.exercise.duration}</td>               {/* Duration in
minutes */}
    <td>{props.exercise.date.substring(0, 10)}</td>  {/* Trim date to first
10 chars (YYYY-MM-DD) */}
    <td>
      {/* Link to edit page using the _id from MongoDB document */}
      <Link to={`/edit/${props.exercise._id}`}>edit</Link> |

      {/* Delete button: Calls deleteExercise function passed as prop */}
      <a href="#!" onClick={() => {
props.deleteExercise(props.exercise._id) }}>delete</a>
    </td>
  </tr>
);

////////////////////////////////////////////////////////////////////////////
///
// Class Component: ExerciseList
// This component is responsible for:
// 1. Fetching exercises from backend on mount
// 2. Rendering the table of exercises
// 3. Deleting an exercise and updating UI
////////////////////////////////////////////////////////////////////////////
///
export default class ExerciseList extends Component {
  constructor(props) {
    super(props);

    // Binding 'this' context for the deleteExercise method
    this.deleteExercise = this.deleteExercise.bind(this);

    // Component state holds an array of all exercise objects
    this.state = {
```

```
      exercises: []
    };
  }


////////////////////////////////////////////////////////////////////////////
/
  // Lifecycle Method: componentDidMount()
  // Runs automatically after the component is rendered (mounted)
  // Fetches the list of exercises from the backend API using axios

////////////////////////////////////////////////////////////////////////////
/
  componentDidMount() {
    axios.get('http://localhost:5000/exercises/')
      .then(res => {
        // On success, update the exercises array in state
        this.setState({ exercises: res.data });
      })
      .catch(err => {
        // On error, print the error to the console
        console.log(err);
      });
  }


////////////////////////////////////////////////////////////////////////////
/
  // Method: deleteExercise
  // Deletes an exercise from the database and updates the UI
  // @param {string} id - ID of the exercise to delete

////////////////////////////////////////////////////////////////////////////
/
  deleteExercise(id) {
    // Send DELETE request to backend
    axios.delete(`http://localhost:5000/exercises/${id}`)
      .then(res => console.log(res.data)) // Log server response

    // Update UI by removing the deleted item from state without reloading
    this.setState({
      exercises: this.state.exercises.filter(el => el._id !== id)
    });
  }


////////////////////////////////////////////////////////////////////////////
/
  // Method: exerciseList
  // Returns a list of <Exercise /> components, one for each item in the
state

////////////////////////////////////////////////////////////////////////////
/
```

```
  exerciseList() {
    return this.state.exercises.map(currentexercise => {
      return (
        <Exercise
          exercise={currentexercise}                // Pass data to child
          deleteExercise={this.deleteExercise}      // Pass delete handler
to child
          key={currentexercise._id}                 // Unique key for
efficient rendering
        />
      );
    });
  }


  ////////////////////////////////////////////////////////////////////////////
  /
    // Render Method
    // This returns the JSX that defines the structure of the component UI

  ////////////////////////////////////////////////////////////////////////////
  /
    render() {
      return (
        <div>
          <h3>Logged Exercises</h3>
          <table className="table">
            <thead className="thead-light">
              <tr>
                <th>Username</th>
                <th>Description</th>
                <th>Duration</th>
                <th>Date</th>
                <th>Actions</th> {/* For edit & delete buttons */}
              </tr>
            </thead>
            <tbody>
              { this.exerciseList() } {/* Render list of exercises */}
            </tbody>
          </table>
        </div>
      );
    }
  }
```

7.5.5 src/components/EditExercise.component.js

```
// Import React and Component class for creating a class-based component
import React, { Component } from 'react';
```

```
// Import DatePicker component and its styles
import DatePicker from 'react-datepicker';
import 'react-datepicker/dist/react-datepicker.css';

// Import axios to make HTTP requests
import axios from 'axios';

////////////////////////////////////////////////////////////////////////
////
// CLASS COMPONENT: EditExercise
// This component fetches an existing exercise entry from the server,
// allows the user to edit it using a form, and updates it via a POST
request.
////////////////////////////////////////////////////////////////////////
////
export default class EditExercise extends Component {
  constructor(props) {
    super(props);

    // Initialize the component state with default values
    this.state = {
      username: '',        // Selected user
      description: '',     // Description of exercise
      duration: 0,         // Duration in minutes
      date: new Date(),    // Date of the exercise
      users: []            // List of usernames for dropdown
    };

    // Bind `this` context to event handlers
    this.onChangeUsername = this.onChangeUsername.bind(this);
    this.onChangeDescription = this.onChangeDescription.bind(this);
    this.onChangeDuration = this.onChangeDuration.bind(this);
    this.onChangeDate = this.onChangeDate.bind(this);
    this.onSubmit = this.onSubmit.bind(this);
  }


////////////////////////////////////////////////////////////////////////
//
  // LIFECYCLE METHOD: componentDidMount()
  // Runs after the component is inserted into the DOM
  // Fetches the existing exercise data to populate the form

////////////////////////////////////////////////////////////////////////
//
  componentDidMount() {
    // Fetch the specific exercise using the ID from URL params

axios.get(`http://localhost:5000/exercises/${this.props.router.params.id}`)
      .then(res => {
        // Update state with exercise data
        this.setState({
          username: res.data.username,
          description: res.data.description,
```

```
          duration: res.data.duration,
          date: new Date(res.data.date) // Convert to JS Date object
        });
      })
      .catch(err => console.log(err)); // Handle any error

    // Fetch list of all users to populate the dropdown
    axios.get('http://localhost:5000/users/')
      .then(res => {
        // If user list is not empty, extract usernames into the users
array
        if (res.data.length > 0) {
          this.setState({
            users: res.data.map(user => user.username)
          });
        }
      })
      .catch(err => console.log(err)); // Handle error
  }

  // Event handler: update username in state
  onChangeUsername(e) {
    this.setState({ username: e.target.value });
  }

  // Event handler: update description in state
  onChangeDescription(e) {
    this.setState({ description: e.target.value });
  }

  // Event handler: update duration in state
  onChangeDuration(e) {
    this.setState({ duration: e.target.value });
  }

  // Event handler: update date in state
  onChangeDate(date) {
    this.setState({ date: date });
  }


////////////////////////////////////////////////////////////////////////////
//
  // FORM SUBMIT HANDLER
  // Called when the form is submitted. Sends the updated exercise data
  // to the server using POST request and redirects back to homepage

////////////////////////////////////////////////////////////////////////////
//
  onSubmit(e) {
    e.preventDefault(); // Prevent page refresh

    const exercise = {
      username: this.state.username,
```

```
      description: this.state.description,
      duration: this.state.duration,
      date: this.state.date
    };

    console.log('Submitting updated exercise:', exercise);

    // Send POST request to update the exercise on the backend

axios.post(`http://localhost:5000/exercises/update/${this.props.router.para
ms.id}`, exercise)
      .then(res => console.log(res.data));

    // Redirect to home page after submit
    window.location = '/';
  }


////////////////////////////////////////////////////////////////////////////
//
  // RENDER METHOD
  // Returns JSX that renders the form pre-filled with current values

////////////////////////////////////////////////////////////////////////////
//
  render() {
    return (
      <div>
        <h3>Edit Exercise Log</h3>
        <form onSubmit={this.onSubmit}>
          {/* USERNAME DROPDOWN */}
          <div className="form-group">
            <label>Username: </label>
            <select
              required
              className="form-control"
              value={this.state.username}
              onChange={this.onChangeUsername}
            >
              {this.state.users.map(user => (
                <option key={user} value={user}>{user}</option>
              ))}
            </select>
          </div>

          {/* DESCRIPTION INPUT */}
          <div className="form-group">
            <label>Description: </label>
            <input
              type="text"
              required
              className="form-control"
              value={this.state.description}
              onChange={this.onChangeDescription}
```

```
            />
          </div>

          {/* DURATION INPUT */}
          <div className="form-group">
            <label>Duration (in minutes): </label>
            <input
              type="number"
              className="form-control"
              value={this.state.duration}
              onChange={this.onChangeDuration}
            />
          </div>

          {/* DATE PICKER */}
          <div className="form-group">
            <label>Date: </label>
            <div>
              <DatePicker
                selected={this.state.date}
                onChange={this.onChangeDate}
              />
            </div>
          </div>

          {/* SUBMIT BUTTON */}
          <div className="form-group">
            <input type="submit" value="Update Exercise Log" className="btn
btn-primary" />
          </div>
        </form>
      </div>
    );
  }
}
```

---

8. Running the App

1. **Start backend**:

```
cd backend
nodemon server.js
```

2. **Start frontend**:

```
cd mern-exercise-tracker
npm start
```

3. Visit `http://localhost:3000` to interact with your Exercise Tracker!

---

9. Conclusion

You have now built a full-stack **Exercise Tracker** application using the MERN stack:

- **Backend**: Node.js, Express, MongoDB Atlas, Mongoose
- **Frontend**: React, Axios, React Router, Bootstrap

Feel free to extend this app by:

- Adding authentication (JWT)
- Deploying to Heroku / Netlify
- Enhancing UI/UX

Happy coding! 🚀

Others

Bonus: User Upgrades

- User Delete API with `.findOneAndDelete()`& `.deleteMany()`
  - Deletes user exercises
  - Deletes user
- User Update API with `.findOne()`&`.updateMany()`
  - Updates user exercises
  - Updates user

```
// bla bla bla

const Exercise = require('../models/exercise.model');

// DELETE: Remove user and their exercises
router.route('/delete/:username').delete((req, res) => {
  const username = req.params.username;

  // First delete all exercises by this user
  Exercise.deleteMany({ username: username })
    .then(() => {
      // Then delete the user
      return User.findOneAndDelete({ username: username });
    })
    .then(() => {
      res.json('User and all associated exercises deleted!');
    })
    .catch(err => {
      res.status(400).json('Error: ' + err);
    });
});
```

```
router.route('/update/:username').post((req,res)=>{
    let oldUsername = req.params.username;
    let newUsername = req.body.username;

    User.findOne({username:oldUsername})
    .then((user)=>{user.username=newUsername;
        user.save()
            .then(() => {
      // Then update all exercises with old username
      return Exercise.updateMany(
        { username: oldUsername },
        { $set: { username: newUsername } }
      );
    })
    .then(() => {
      res.json('Username and related exercises updated!');
    })
    .catch(err => res.status(400).json('Error: ' + err));


    })
    .catch(err => res.status(400).json('Error: ' + err));  // Handle error
});

// bla bla bla
```

- /srs/components/EditUser.component.js

```
// Import React and Component class
import React, { Component } from 'react';
import withRouter from './withRouter'; // Ensure this is the correct path

// Import axios for HTTP requests
import axios from 'axios';

////////////////////////////////////////////////////////////////////////////
////
// CLASS COMPONENT: EditUser
// This component allows editing a user's username.
// It fetches the user based on the username in the URL and updates it via
POST.
////////////////////////////////////////////////////////////////////////////
////
class EditUser extends Component {
  constructor(props) {
    super(props);

    // Initialize state
```

```
    this.state = {
      username: ''
    };

    // Bind methods
    this.onChangeUsername = this.onChangeUsername.bind(this);
    this.onSubmit = this.onSubmit.bind(this);
  }


  ////////////////////////////////////////////////////////////////////////////
  //
    // LIFECYCLE METHOD: componentDidMount()
    // Fetch the existing user using the username from the URL

  ////////////////////////////////////////////////////////////////////////////
  //
    componentDidMount() {
      const oldUsername = this.props.router.params.username;

      axios.get('http://localhost:5000/users/')
        .then(res => {
          const user = res.data.find(u => u.username === oldUsername);
          if (user) {
            this.setState({ username: user.username });
          } else {
            console.log("User not found");
          }
        })
        .catch(err => console.log(err));
    }


  ////////////////////////////////////////////////////////////////////////////
  //
    // EVENT HANDLER: When input field changes, update state

  ////////////////////////////////////////////////////////////////////////////
  //
    onChangeUsername(e) {
      this.setState({ username: e.target.value });
    }


  ////////////////////////////////////////////////////////////////////////////
  //
    // FORM SUBMIT HANDLER: Update username via POST request

  ////////////////////////////////////////////////////////////////////////////
  //
    onSubmit(e) {
      e.preventDefault();

      const updatedUser = {
```

```
      username: this.state.username
    };

    const oldUsername = this.props.router.params.username;

    axios.post(`http://localhost:5000/users/update/${oldUsername}`,
updatedUser)
      .then(res => console.log(res.data))
      .catch(err => console.log(err));

    // Redirect to user list
    window.location = '/users';
  }


////////////////////////////////////////////////////////////////////////////
//
  // RENDER METHOD: Shows the form with the current username

////////////////////////////////////////////////////////////////////////////
//
  render() {
    return (
      <div>
        <h3>Edit User</h3>
        <form onSubmit={this.onSubmit}>
          {/* USERNAME INPUT FIELD */}
          <div className="form-group">
            <label>New Username: </label>
            <input
              type="text"
              required
              className="form-control"
              value={this.state.username}
              onChange={this.onChangeUsername}
            />
          </div>

          {/* SUBMIT BUTTON */}
          <div className="form-group">
            <input type="submit" value="Update User" className="btn btn-
primary" />
          </div>
        </form>
      </div>
    );
  }
}

export default withRouter(EditUser);
```

- /srs/components/User.component.js

```
import React, { Component } from 'react';
import axios from 'axios';
import { Link } from 'react-router-dom';

////////////////////////////////////////////////////////
// Functional Component: User (for each user row)
////////////////////////////////////////////////////////
const User = (props) => (
  <tr>
    <td>{props.user.username}</td>
    <td>
<Link to={`/users/update/${props.user.username}`}>edit</Link>|
      <a href="#!" onClick={() => { props.deleteUser(props.user.username)
}}>delete</a>
    </td>
  </tr>
);

////////////////////////////////////////////////////////
// Class Component: UserList
// - Fetch users from backend
// - Render users in table
// - Delete user and update UI
////////////////////////////////////////////////////////
export default class UserList extends Component {
  constructor(props) {
    super(props);

    this.deleteUser = this.deleteUser.bind(this);

    this.state = {
      users: []
    };
  }

  componentDidMount() {
    axios.get('http://localhost:5000/users/')
      .then(res => {
        this.setState({ users: res.data });
      })
      .catch(err => {
        console.log(err);
      });
  }

  deleteUser(username) {
    axios.delete(`http://localhost:5000/users/delete/${username}`)
      .then(res => console.log(res.data));

    this.setState({
      users: this.state.users.filter(el => el.username !== username)
    });
```

```
  }

  userList() {
    return this.state.users.map(currentuser => {
      return (
        <User
          user={currentuser}
          deleteUser={this.deleteUser}
          key={currentuser._id}
        />
      );
    });
  }

  render() {
    return (
      <div>
        <h3>Registered Users</h3>
        <table className="table">
          <thead className="thead-light">
            <tr>
              <th>Username</th>
              <th>Actions</th>
            </tr>
          </thead>
          <tbody>
            { this.userList() }
          </tbody>
        </table>
      </div>
    );
  }
}
```

- Routing above in `src/App.js`

```
// bla bla bla
import EditUserComponent from "./components/EditUser.component";
import UserList from "./components/User.component";

// bla bla bla
<Routes>
        <Route path="/users" Component={UserList} />
        <Route path="/users/update/:username" Component={EditUserComponent}
/>
        </Routes>


// bla bla bla
```

`this.props.match.params.id` rejected, now use `const { id } = useParams();` or `this.props.match.params.id`

In short:

- `**match**` → used in **React Router v5** → `this.props.match.params.id`
- `**router**` → custom wrapper for **React Router v6** → `this.props.router.params.id` (after using `withRouter`)

> ✅ Use `router` in v6 because `match` no longer exists.

- jugaad code

```
// withRouter.js
import { useParams, useNavigate, useLocation } from 'react-router-dom';
import React from 'react';

export default function withRouter(Component) {
  function ComponentWithRouterProp(props) {
    const params = useParams();
    const navigate = useNavigate();
    const location = useLocation();

    return (
      <Component
        {...props}
        router={{ params, navigate, location }} // 👈 attach to `router`
      />
    );
  }

  return ComponentWithRouterProp;
}



// REASON
// React Router v6+ does not provide this.props.match directly.
// To fix this, wrap your class component using a small withRouter HOC
(Higher Order Component) to pass the params manually

// old rejected
// this.props.match.params.id

// new

// import { useParams } from 'react-router-dom';

// const MyComponent = () => {
```

```
//   const { id } = useParams();
//   return <div>ID: {id}</div>;
// };
```

- useage

```
// bla bla bla

import withRouter from './withRouter'; // adjust path as needed


class EditExercise extends Component {
  // bla bla bla
}

export default withRouter(EditExercise);// done
```

🧠 Node.js Essentials

🚀 What is Node.js?

> **Node.js** is a runtime environment that lets you **run JavaScript on the server** (outside the browser).

- Built on **Chrome's V8 JavaScript engine**
- Uses **event-driven**, **non-blocking I/O**
- Perfect for building **fast**, **scalable** apps: APIs, real-time apps, microservices

---

📦 1. Setup and Initialize

```
#### Install Node.js (from https://nodejs.org)
#### Then initialize your project
npm init -y
```

Creates a `package.json` file which manages project dependencies and scripts.

📂 2. File Structure (Basic Project)

```
my-app/
├── index.js
├── package.json
└── node_modules/
```

🛠 3. Running a File

```
node index.js
```

```
// index.js
console.log("Hello from Node.js!");
```

📚 4. Modules & `require()`

Built-in Modules

```
const os = require('os');
console.log('CPU Cores:', os.cpus().length);
```

File System Module

```
const fs = require('fs');

// Read file
fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});

// Write file
fs.writeFile('output.txt', 'Hello Node.js!', (err) => {
  if (err) throw err;
  console.log('File written!');
});
```

Create Your Own Module

```
// math.js
function add(a, b) {
  return a + b;
}
module.exports = add;
```

```
// index.js
const add = require('./math');
console.log(add(2, 3)); // 5
```

🌐 5. Build a Simple HTTP Server

```
const http = require('http');

// Create a server
const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello from Node server!');
});

// Start server on port 3000
server.listen(3000, () => {
  console.log('✅ Server running at http://localhost:3000');
});
```

⚙️ 6. npm and External Packages

Install Packages

```
npm install express
```

Use in Your Project

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello from Express!');
});

app.listen(3000, () => {
  console.log('✅ Express server on http://localhost:3000');
});
```

🕹️ 7. Handling Requests in Express

```
// Middleware to parse JSON
app.use(express.json());

// GET
app.get('/users', (req, res) => {
  res.json([{ name: 'Alice' }, { name: 'Bob' }]);
});

// POST
app.post('/users', (req, res) => {
  const user = req.body;
  res.status(201).json({ message: 'User created', user });
});
```

📂 8. Organizing Your Code (MVC Style)

```
project/
├── index.js
├── routes/
│   └── userRoutes.js
├── controllers/
│   └── userController.js
└── models/
    └── userModel.js
```

🧪 9. Debugging & Logging

```
console.log("✅ Everything looks fine");
console.error("❌ Something went wrong");
```

Use tools like:

- `nodemon`: auto-restarts server on changes
- `npm install --save-dev nodemon`
- Add to `package.json` scripts:

```
"scripts": {
  "dev": "nodemon index.js"
}
```

Run with:

```
npm run dev
```

🔐 10. Environment Variables

Use `.env` to hide secrets:

```
#### .env
PORT=5000
```

```
npm install dotenv
```

```
require('dotenv').config();
console.log(process.env.PORT);
```

📌 11. Common Built-In Modules

Module	Use Case
`fs`	File system operations
`path`	File/directory path manipulation
`os`	OS info (CPU, memory, etc.)
`http`	Build server without Express
`events`	Event handling system

🧠 12. Key Concepts to Know

Concept	Meaning / Example
Non-blocking	Code continues running while waiting for I/O
Asynchronous	Uses callbacks, promises, or `async/await`
Event loop	Mechanism behind async behavior
Callback Hell	Too many nested callbacks → use Promises instead

Concept	Meaning / Example
Middleware	Functions between request & response in Express

🚀 Express.js Essentials for Beginners

📦 1. What is Express.js?

Express.js is a minimal and flexible Node.js web application framework.

- Makes building APIs and web apps **fast and easy**
- Simplifies handling **routes**, **requests**, and **responses**
- Supports **middleware** for modular code

🔧 2. Installation & Setup

```
npm init -y
npm install express
```

```
// index.js
const express = require('express');
const app = express(); // Create an Express app

const PORT = 3000;

// Root route
app.get('/', (req, res) => {
  res.send('🎉 Hello from Express!');
});

// Start the server
app.listen(PORT, () => {
  console.log(`🚀 Server running at http://localhost:${PORT}`);
});
```

Run it using:

```
node index.js
```

📬 3. HTTP Methods (GET, POST, PUT, DELETE)

```
// Middleware to parse JSON request bodies
app.use(express.json());

// GET request (read)
app.get('/users', (req, res) => {
  res.json([{ name: 'Alice' }, { name: 'Bob' }]);
});

// POST request (create)
app.post('/users', (req, res) => {
  const user = req.body; // Get data sent in JSON
  res.status(201).json({ message: 'User added', user });
});

// PUT request (update)
app.put('/users/:id', (req, res) => {
  res.json({ message: `User ${req.params.id} updated` });
});

// DELETE request (delete)
app.delete('/users/:id', (req, res) => {
  res.json({ message: `User ${req.params.id} deleted` });
});
```

🧱 4. Route Parameters & Query Params

```
// Route parameter
app.get('/user/:name', (req, res) => {
  res.send(`Hello, ${req.params.name}!`);
});

// Query parameter
app.get('/search', (req, res) => {
  res.send(`Search query: ${req.query.q}`);
});
```

URL example:

- `/user/robin` → `req.params.name = robin`
- `/search?q=node` → `req.query.q = node`

🧪 5. Middleware Explained Simply

Middleware = functions that run **before** the route handler

```
// Global middleware (runs on every request)
app.use((req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.path}`);
  next(); // Move to next middleware or route
});
```

Other common middleware:

- `express.json()` → to read JSON body
- `cors()` → handle CORS (install with `npm install cors`)

📂 6. Organize Routes in Separate File

Step 1: Create `routes/userRoutes.js`

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.send('Get all users');
});

router.post('/', (req, res) => {
  res.send('Add new user');
});

module.exports = router;
```

Step 2: Import in `index.js`

```
const userRoutes = require('./routes/userRoutes');
app.use('/users', userRoutes);
```

Now:

- `GET /users` → "Get all users"
- `POST /users` → "Add new user"

🧾 7. Send JSON, Status, Headers

```
app.get('/status', (req, res) => {
  res.status(200).json({
```

```
    message: 'Everything is OK',
    timestamp: new Date(),
  });
});
```

🚨 8. Error Handling

```
// 404 Not Found (for unmatched routes)
app.use((req, res) => {
  res.status(404).json({ error: 'Route not found' });
});
```

✅ 9. Dev Tip: Use `nodemon` for Auto-Restart

```
npm install --save-dev nodemon
```

Add to `package.json`:

```
"scripts": {
  "dev": "nodemon index.js"
}
```

Run:

```
npm run dev
```

📦 10. Example JSON API Response

```
app.get('/products', (req, res) => {
  res.json([
    { id: 1, name: 'Tyre', price: 1500 },
    { id: 2, name: 'Battery', price: 2500 }
  ]);
});
```

Use **Postman** or browser to test it.

💡 BONUS: Build This Mini Project

Simple User API using Express:

Route	Method	Description
`/users`	GET	Get all users
`/users/:id`	GET	Get user by ID
`/users`	POST	Add a new user
`/users/:id`	PUT	Update user
`/users/:id`	DELETE	Delete user

✅ Use in-memory array ✅ Use Postman to test ✅ Later connect it to MongoDB with Mongoose

---

📌 Summary Table

Feature	Code
Create server	`express()` + `listen()`
Route	`app.get('/route', fn)`
Body data	`express.json()` middleware
Params	`req.params`, `req.query`
JSON response	`res.json({ ... })`
Error handling	`app.use((req, res) => {})`

Axios Essentials

What is Axios?

Axios is a popular JavaScript library used to make HTTP requests (`GET`, `POST`, `PUT`, `DELETE`, etc.) from:

- The browser (frontend)
- Node.js (backend)

It supports Promises and works well with `async/await`.

Installation

Using npm:

```
npm install axios
```

Or via CDN in HTML:

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js">
</script>
```

Common Use Cases

HTTP Method	Use Case
GET	Fetch data
POST	Send new data
PUT	Update existing data
DELETE	Remove data

Basic Syntax

Using Promises

```
import axios from 'axios';

axios.get('/api/example')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error(error);
  });
```

Using async/await

```
const fetchData = async () => {
  try {
    const res = await axios.get('/api/example');
    console.log(res.data);
  } catch (err) {
    console.error(err);
  }
};
```

POST Request Example

```
const user = {
  name: 'Robin',
  age: 25
};

axios.post('/api/users', user)
  .then(res => console.log(res.data))
  .catch(err => console.error(err));
```

Axios Configuration

Global Configuration

```
axios.defaults.baseURL = 'https://example.com/api';
axios.defaults.headers.common['Authorization'] = 'Bearer token123';
axios.defaults.timeout = 5000;
```

Per-request Configuration

```
axios.get('/users', {
  params: { id: 1 },
  headers: { 'X-Custom-Header': 'hello' }
});
```

Common Patterns

GET with Query Parameters

```
axios.get('/products', { params: { category: 'spares' } });
```

PUT to Update Data

```
axios.put('/users/1', { name: 'Updated Name' });
```

DELETE a Resource

```
axios.delete('/users/1');
```

Axios Interceptors

Request Interceptor

```
axios.interceptors.request.use(config => {
  config.headers.Authorization = 'Bearer your_token';
  return config;
});
```

Response Interceptor

```
axios.interceptors.response.use(
  response => response,
  error => {
    if (error.response.status === 401) {
      console.log("Unauthorized");
    }
    return Promise.reject(error);
  }
);
```

Uploading Files

```
const formData = new FormData();
formData.append('file', selectedFile);

axios.post('/upload', formData, {
  headers: { 'Content-Type': 'multipart/form-data' }
});
```

Using Axios in React

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const UsersList = () => {
  const [users, setUsers] = useState([]);

  useEffect(() => {
```

```
      axios.get('http://localhost:4000/api/users')
        .then(res => setUsers(res.data))
        .catch(err => console.error(err));
    }, []);

    return (
      <div>
        <h2>User List</h2>
        {users.map(u => <div key={u.id}>{u.name}</div>)}
      </div>
    );
  };

  export default UsersList;
```

Common Errors and Solutions

Issue	Solution
CORS errors	Backend must send proper CORS headers
Forgetting `await`	Leads to unresolved promises
Incorrect `Content-Type`	Use `application/json` or `multipart/form-data`
No error handling	Always use `.catch()` or `try/catch`

Summary

- Axios simplifies HTTP requests with a cleaner API than `fetch()`
- Works in both browser and Node.js environments
- Supports async/await, interceptors, configuration, and error handling
- Ideal for integrating REST APIs in React or backend apps

Getting Started with Create React App

This project was bootstrapped with Create React App.

Available Scripts

In the project directory, you can run:

`npm start`

Runs the app in the development mode.
Open http://localhost:3000 to view it in your browser.

The page will reload when you make changes.
You may also see any lint errors in the console.

```
npm test
```

Launches the test runner in the interactive watch mode.
See the section about running tests for more information.

```
npm run build
```

Builds the app for production to the `build` folder.
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.
Your app is ready to be deployed!

See the section about deployment for more information.

```
npm run eject
```

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This command will remove the single build dependency from your project.

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESLint, etc) right into your project so you have full control over them. All of the commands except `eject` will still work, but they will point to the copied scripts so you can tweak them. At this point you're on your own.

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deployments, and you shouldn't feel obligated to use this feature. However we understand that this tool wouldn't be useful if you couldn't customize it when you are ready for it.

Learn More

You can learn more in the Create React App documentation.

To learn React, check out the React documentation.

Code Splitting

This section has moved here: https://facebook.github.io/create-react-app/docs/code-splitting

Analyzing the Bundle Size

This section has moved here: https://facebook.github.io/create-react-app/docs/analyzing-the-bundle-size

Making a Progressive Web App

This section has moved here: https://facebook.github.io/create-react-app/docs/making-a-progressive-web-app

Advanced Configuration

This section has moved here: https://facebook.github.io/create-react-app/docs/advanced-configuration

Deployment

This section has moved here: https://facebook.github.io/create-react-app/docs/deployment

`npm run build` fails to minify

This section has moved here: https://facebook.github.io/create-react-app/docs/troubleshooting#npm-run-build-fails-to-minify

Postman APIs v2.2 load


```
{
    "info": {
        "_postman_id": "9180c15c-bc52-4982-81ed-fe06e53514f8",
        "name": "mern-excercise-tracker",
        "schema":
"https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
        "_exporter_id": "43597602",
        "_collection_link":
"https://siddhantbali.postman.co/workspace/Siddhant-Bali's-
Workspace~aa45828a-f45f-44f3-88be-317b8266fb88/collection/43597602-
9180c15c-bc52-4982-81ed-fe06e53514f8?
action=share&source=collection_link&creator=43597602"
    },
    "item": [
        {
            "name": "/users/add",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"username\":\"bhaskar\"\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "{{baseURL}}/users/add",
                    "host": [
                        "{{baseURL}}"
                    ],
                    "path": [
                        "users",
                        "add"
                    ]
                }
```

```
            },
            "response": []
        },
        {
            "name": "/users/",
            "request": {
                "method": "GET",
                "header": [],
                "url": {
                    "raw": "{{baseURL}}/users/",
                    "host": [
                        "{{baseURL}}"
                    ],
                    "path": [
                        "users",
                        ""
                    ]
                }
            },
            "response": []
        },
        {
            "name": "/exercises/add",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"username\":\"bhati\",\n
\"description\":\"Deadlift\",\n    \"duration\": \"11\" ,\n
\"date\":\"2025-08-30T19:03:24.994Z\"\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "{{baseURL}}/exercises/add",
                    "host": [
                        "{{baseURL}}"
                    ],
                    "path": [
                        "exercises",
                        "add"
                    ]
                }
            },
            "response": []
        },
        {
            "name": "/exercises/",
            "protocolProfileBehavior": {
                "disableBodyPruning": true
```

```
            },
            "request": {
                "method": "GET",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "{\n    \"username\":\"bhati\",\n
\"description\":\"Deadlift\",\n    \"duration\": \"11\" ,\n
\"date\":\"2025-08-30T19:03:24.994Z\"\n}",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "{{baseURL}}/exercises/",
                    "host": [
                        "{{baseURL}}"
                    ],
                    "path": [
                        "exercises",
                        ""
                    ]
                }
            },
            "response": []
        },
        {
            "name": "/exercises/:id",
            "protocolProfileBehavior": {
                "disableBodyPruning": true
            },
            "request": {
                "method": "GET",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "
{{baseURL}}/exercises/688a6fee71069ae17aa9e424",
                    "host": [
                        "{{baseURL}}"
                    ],
                    "path": [
                        "exercises",
                        "688a6fee71069ae17aa9e424"
```

```
                    ]
                }
            },
            "response": []
        },
        {
            "name": "/exercises/update/:id",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "        {\n        \"username\": \"bhaskar\",\n
\"description\": \"Mountain Climbing\",\n        \"duration\": 9,\n
\"date\": \"2025-08-30T19:03:24.994Z\"\n        }",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "
{{baseURL}}/exercises/update/688a6fee71069ae17aa9e424",
                    "host": [
                        "{{baseURL}}"
                    ],
                    "path": [
                        "exercises",
                        "update",
                        "688a6fee71069ae17aa9e424"
                    ]
                }
            },
            "response": []
        },
        {
            "name": "/users/update/:username",
            "request": {
                "method": "POST",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": "        {\n        \"username\": \"Siddhant
Bali\"\n        }",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "{{baseURL}}/username/update/bali",
                    "host": [
```

```
                        "{{baseURL}}"
                    ],
                    "path": [
                        "username",
                        "update",
                        "bali"
                    ]
                }
            },
            "response": []
        },
        {
            "name": "/exercises/delete/:id",
            "request": {
                "method": "DELETE",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": " {\n            \"username\": \"bhati\",\n
\"description\": \"Deadlift\",\n            \"duration\": 11,\n
\"date\": \"2025-08-30T19:03:24.994Z\",\n        }",
                    "options": {
                        "raw": {
                            "language": "json"
                        }
                    }
                },
                "url": {
                    "raw": "
{{baseURL}}/exercises/delete/688a707071069ae17aa9e428",
                    "host": [
                        "{{baseURL}}"
                    ],
                    "path": [
                        "exercises",
                        "delete",
                        "688a707071069ae17aa9e428"
                    ]
                }
            },
            "response": []
        },
        {
            "name": "/users/delete/:username",
            "request": {
                "method": "DELETE",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": " {\n            \"username\": \"bhati\"\n
}",
                    "options": {
                        "raw": {
                            "language": "json"
```

```
                        }
                    }
                },
                "url": {
                    "raw": "{{baseURL}}/users/delete/bhati",
                    "host": [
                        "{{baseURL}}"
                    ],
                    "path": [
                        "users",
                        "delete",
                        "bhati"
                    ]
                }
            },
            "response": []
        }
    ],
    "variable": [
        {
            "key": "baseURL",
            "value": "http://localhost:5000",
            "type": "default"
        }
    ]
}
```