Node.js + Express — RESTful Services

1. Basic Setup

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

Create Server

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

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

app.listen(3000, () => console.log("Server running"));
```

---

2. Routing Basics

Route Structure

```
app.METHOD(PATH, HANDLER);
```

HTTP Methods

- GET – fetch data
- POST – create
- PUT/PATCH – update
- DELETE – remove

---

3. CRUD REST API Example

```
let items = [{ id: 1, name: "Item 1" }];

// CREATE
app.post("/items", (req, res) => {
  const item = req.body;
  items.push(item);
  res.status(201).json(item);
```

```
});

// READ all
app.get("/items", (req, res) => res.json(items));

// READ one
app.get("/items/:id", (req, res) => {
  const found = items.find((i) => i.id == req.params.id);
  if (!found) return res.status(404).json({ error: "Not found" });
  res.json(found);
});

// UPDATE
app.put("/items/:id", (req, res) => {
  items = items.map((i) => (i.id == req.params.id ? { ...i, ...req.body } : i
  res.json(req.body);
});

// DELETE
app.delete("/items/:id", (req, res) => {
  items = items.filter((i) => i.id != req.params.id);
  res.json({ deleted: true });
});
```

---

4. Express Concepts

Middleware

```
app.use((req, res, next) => {
  console.log("Request received");
  next();
});
```

Error Handling

```
app.use((err, req, res, next) => {
  res.status(500).json({ error: err.message });
});
```

Sequelize ORM

1. Installation

```
npm install sequelize sqlite3
```

2. Setup

```
const { Sequelize, DataTypes } = require("sequelize");
const sequelize = new Sequelize("sqlite:db.sqlite");
```

---

3. Defining Models

```
const User = sequelize.define("User", {
  name: { type: DataTypes.STRING, allowNull: false },
  email: { type: DataTypes.STRING, unique: true },
});
```

Common Data Types

- STRING
- INTEGER
- BOOLEAN
- DATE
- TEXT

---

4. Sync Database

```
sequelize.sync({ alter: true });
```

Creates the tables in your database if they don’t exist

Adds columns if missing (ONLY if using sync({ alter: true }))

sync({ force: true }) = Drops the table then recreates it.

---

5. CRUD Queries

Create

```
await User.create({ name: "Aloy", email: "a@mail.com" });
```

Read

```
const all = await User.findAll();
const one = await User.findByPk(1);
const filtered = await User.findOne({ where: { email: "a@mail.com" } });
```

Update

```
await User.update({ name: "Updated" }, { where: { id: 1 } });
```

Delete

```
await User.destroy({ where: { id: 1 } });
```

---

6. Associations

Associations in Sequelize are used so Sequelize knows how your tables relate, allowing it to:

- Automatically create foreign keys
- Let you fetch related data with .include (automatic JOINs)
- Give helper methods like user.getPosts()
- Keep your code cleaner and easier to maintain
- Without associations, Sequelize treats all tables as unrelated.

One-to-Many

```
User.hasMany(Post);
Post.belongsTo(User);
```

Many-to-Many

```
User.belongsToMany(Role, { through: "UserRoles" });
```

Eager Loading

```
await User.findAll({ include: Post });
```

Lazy Loading

Fetch only when needed

React Basics

1. Component Basics

```
function App() {
  return <h1>Hello</h1>;
}
```

---

2. useState

```
const [count, setCount] = useState(0);

return (
  <div>
    <p>{count}</p>
    <button onClick={() => setCount(count + 1)}>Add</button>
  </div>
);
```

Notes:

- State update triggers re-render
- Do not mutate state directly

---

3. useEffect

Run on Mount

```
useEffect(() => {
  console.log("Mounted");
}, []);
```

Run When Dependencies Change

```
useEffect(() => {
  console.log("Count changed");
}, [count]);
```

Cleanup

```
useEffect(() => {
  const timer = setInterval(() => console.log("tick"), 1000);
  return () => clearInterval(timer);
}, []);
```

---

4. useParams (React Router)

```
import { useParams } from "react-router-dom";

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

---

5. Context API

Create Context

```
import { createContext } from "react";
export const ThemeContext = createContext();
```

Provider

```
function App() {
  const [theme, setTheme] = useState("light");

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      <Home />
    </ThemeContext.Provider>
  );
}
```

Consume Context

```
import { useContext } from "react";
const { theme, setTheme } = useContext(ThemeContext);
```

---

6. Rendering Notes

- Re-renders when state or props change
- Virtual DOM diffs updates
- Components must be pure functions

Summary

Express

- Routes: GET, POST, PUT, DELETE
- Middleware and error handling
- `req.params`, `req.body`

Sequelize

- Model definitions
- CRUD
- Associations
- `sequelize.sync`

React

- Components
- useState
- useEffect
- useParams
- useContext