Purwadhika
Digital Technology School

Full Stack AI Software Development

Full-Stack Application Integration

Job Connector Program

Outline

- Set up a full-stack project
- Configure Tailwind
- Configure Postgres + Prisma with migrations
- Implement REST API endpoints in Express.js
- Integrate frontend with Axios and manage state with Zustand

The Tech Stack

Frontend

Vite + React
Tailwind CSS
Zustand
Axios

Backend

Node.js
Express.js
REST API

Database

PostgreSQL
Prisma ORM

High-Level Architecture

This integration follows a classic **Client-Server** model. The frontend handles UI and state, while the backend manages business logic and data persistence.

- **Client**: React app sends HTTP requests via Axios.
- **Server**: Express receives requests and uses Prisma Client.
- **Database**: Postgres stores data, managed by Prisma schema.

Project Initialization (Backend)

Initialize Node

Start by creating a dedicated server directory and initializing a standard Node.js project.

```
mkdir backend && cd backend
npm init -y
```

Setup TypeScript

Install TypeScript and generate the configuration file to define compilation rules.

```
npm i -D typescript ts-node @type/node nodemon
npx tsx init
```

Project Structure (Backend)

- **routes, controllers, services, and validators.**
- **config/db.ts** initializes Prisma.
- **server.ts** starts the Express server.
- **app.ts** configures middleware and routing.

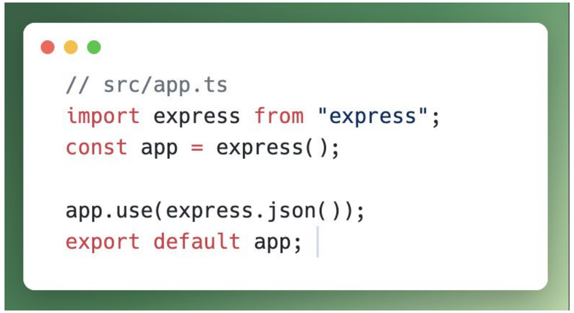
```
backend/
├─ src/
│   ├─ routes/
│   │   ├─ user.routes.ts
│   │   └─ index.ts
│   ├─ controllers/
│   │   └─ user.controller.ts
│   ├─ services/
│   │   └─ user.service.ts
│   └─ validators/
│   │       └─ user.validator.ts
│   ├─ config/
│   │   ├─ env.ts
│   │   └─ db.ts              # Prisma client
│   ├─ middlewares/
│   ├─ utils/
│   ├─ types/
│   ├─ app.ts
│   └─ server.ts
├─ prisma/
│   ├─ schema.prisma
│   └─ migrations/
├─ package.json
├─ tsconfig.json
├─ tsconfig.build.json
├─ .env
└─ .gitignore
```

Using ExpressJS

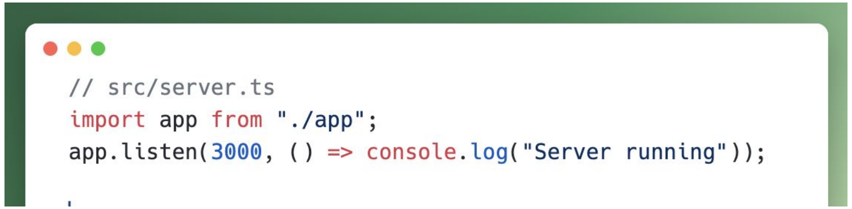
Installation

```
npm install express
npm install -D @types/express
```

Basic Express App



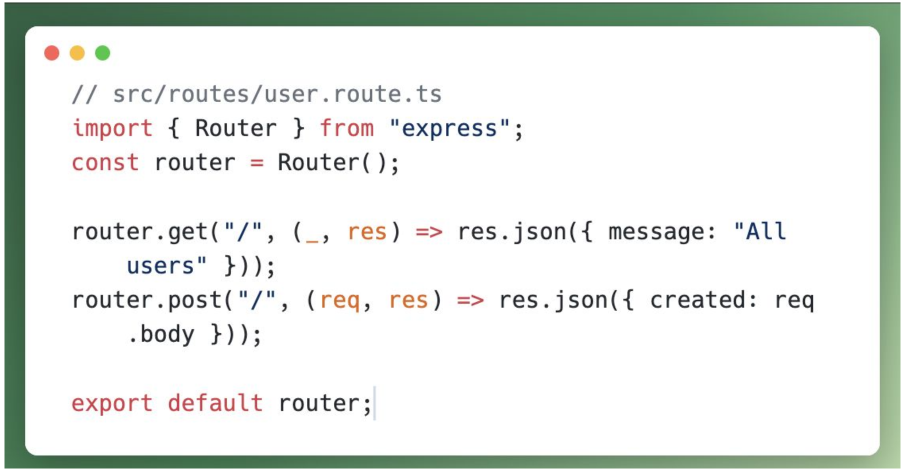
```
// src/app.ts
import express from "express";
const app = express();

app.use(express.json());
export default app;
```



```
// src/server.ts
import app from "./app";
app.listen(3000, () => console.log("Server running"));
```

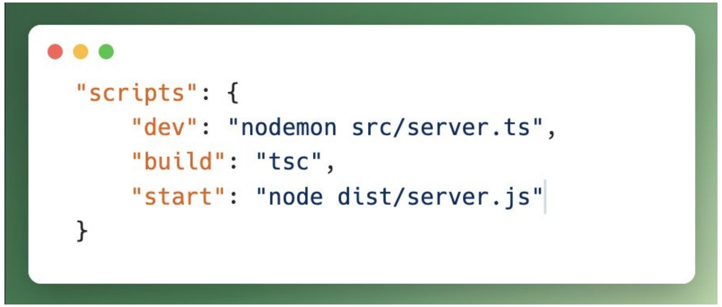
Router Example & NPM Scripts

User Router



```
// src/routes/user.route.ts
import { Router } from "express";
const router = Router();

router.get("/", (_, res) => res.json({ message: "All
    users" }));
router.post("/", (req, res) => res.json({ created: req
    .body }));

export default router;
```

NPM Scripts



```
"scripts": {
    "dev": "nodemon src/server.ts",
    "build": "tsc",
    "start": "node dist/server.js"
}
```

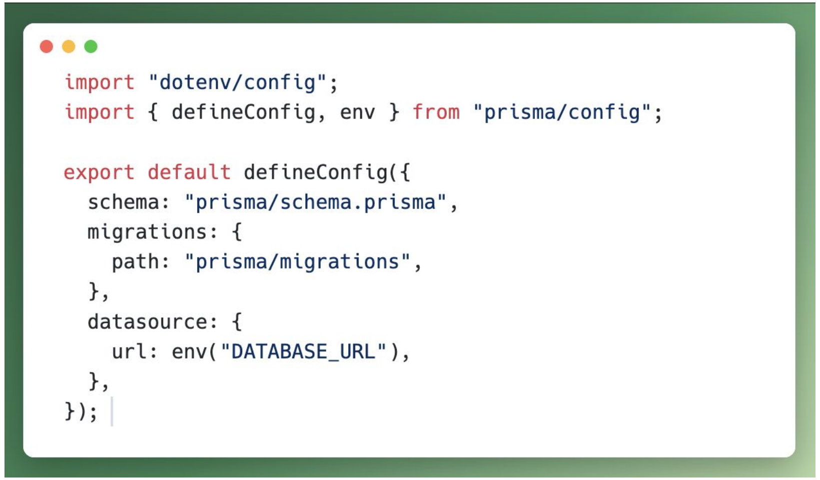
Connecting to Database

Install Prisma & PostgreSQL Client

```
npm install prisma --D
npm install @prisma/client
npx prisma init
```

https://www.prisma.io/docs/getting-started

Setup Prisma Config



```
import "dotenv/config";
import { defineConfig, env } from "prisma/config";

export default defineConfig({
  schema: "prisma/schema.prisma",
  migrations: {
    path: "prisma/migrations",
  },
  datasource: {
    url: env("DATABASE_URL"),
  },
});
```

Define Prisma Model

Define Prisma Model

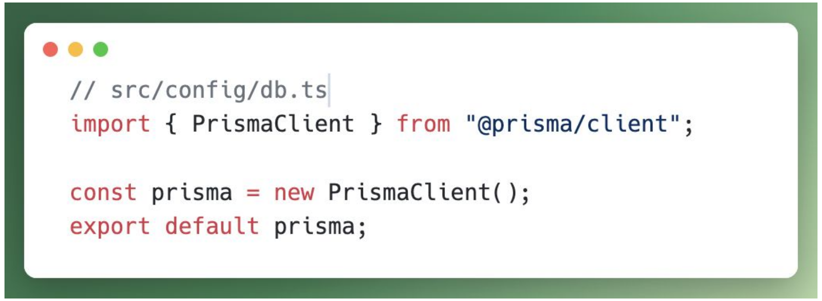
```
generator client {
  provider = "prisma-client"
  output   = "./generated"
}

datasource db {
  provider = "postgresql"
}

model User {
  id    Int     @id @default(autoincrement())
  name  String
  email String  @unique
}
```

Run Migration

```
npx prisma migrate dev --name init
```

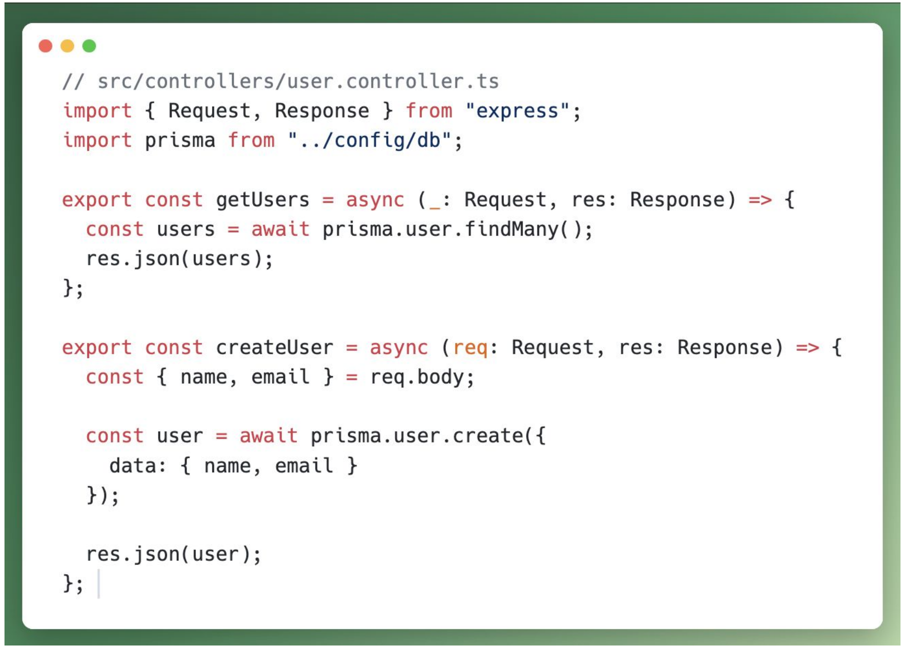
Using Prisma in Controller

Setup Prisma Client



```ts
// src/config/db.ts
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();
export default prisma;
```

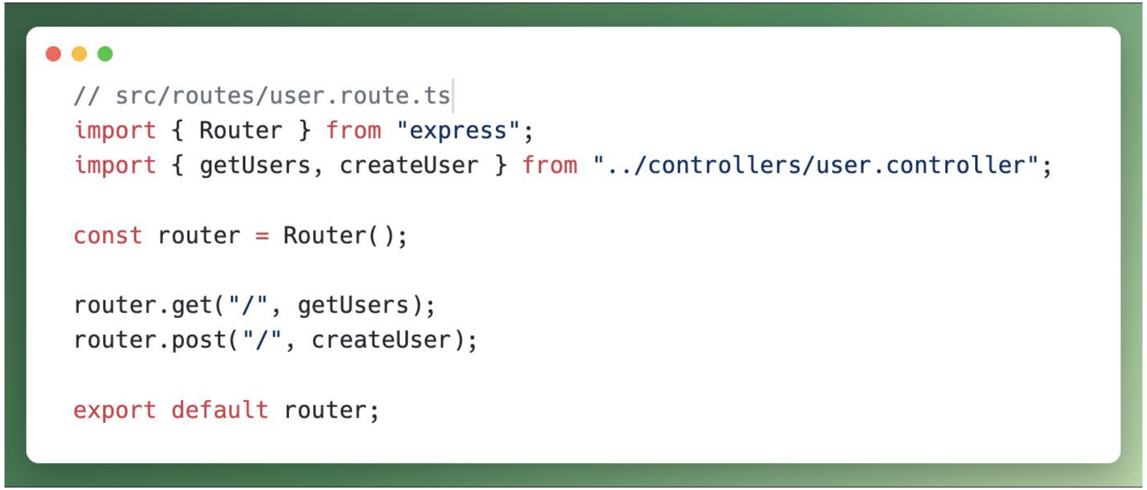
Using Prisma in Controller / Service



```ts
// src/controllers/user.controller.ts
import { Request, Response } from "express";
import prisma from "../config/db";

export const getUsers = async (_: Request, res: Response) => {
  const users = await prisma.user.findMany();
  res.json(users);
};

export const createUser = async (req: Request, res: Response) => {
  const { name, email } = req.body;

  const user = await prisma.user.create({
    data: { name, email }
  });

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

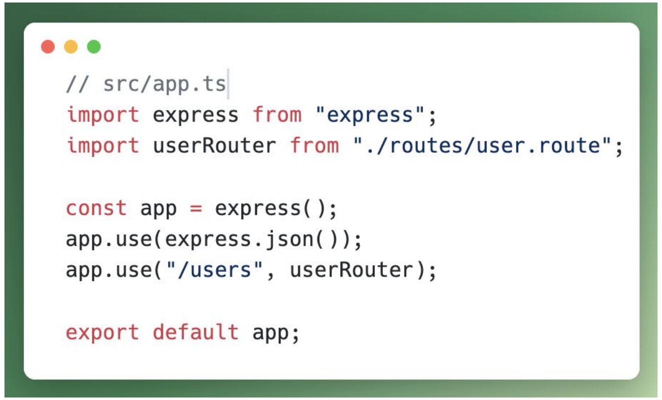
Setup Router for API

Routes for User API



```
// src/routes/user.route.ts
import { Router } from "express";
import { getUsers, createUser } from "../controllers/user.controller";

const router = Router();

router.get("/", getUsers);
router.post("/", createUser);

export default router;
```

Add Routes to Express App



```
// src/app.ts
import express from "express";
import userRouter from "./routes/user.route";

const app = express();
app.use(express.json());
app.use("/users", userRouter);

export default app;
```

Project Initialization (Frontend)

Scaffold with Vite

Use Vite's creation tool to bootstrap a new React application. We explicitly select the React-TS template.

```
npm create vite@latest frontend -- --template react-ts
cd frontend
```

Install & Run

Install the default dependencies and start the development server to verify the setup.

```
npm install
npm run dev
```

https://vite.dev/guide/

Project Structure (Frontend)

- **src/pages:** Route-level views.
- **src/components:** Reusable UI elements.
- **src/store:** Global Zustand state.
- **src/hooks:** Custom React hooks.

```
frontend/
├─ src/
│   ├─ assets/
│   ├─ components/
│   ├─ hooks/
│   ├─ pages/
│   ├─ store/             # zustand
│   ├─ services/          # axios API calls
│   ├─ utils/
│   ├─ types/
│   ├─ App.tsx
│   ├─ main.tsx
│   └─ vite-env.d.ts
├─ public/
├─ index.html
├─ tsconfig.json
├─ vite.config.ts
├─ package.json
└─ .env
```

Styling with Tailwind

Installation

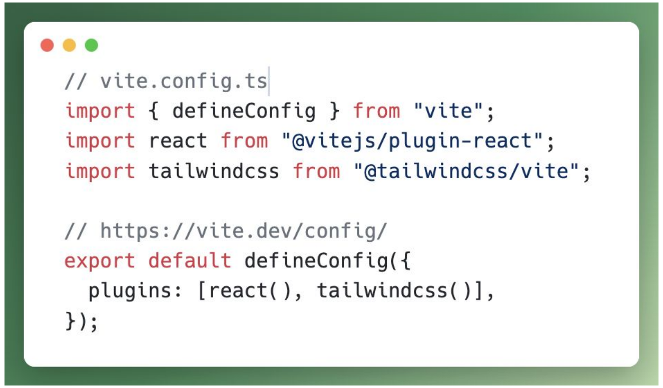
Install Tailwind CSS and its peer dependencies via npm, then generate the configuration files.

```
npm install tailwindcss @tailwindcss/vite
```

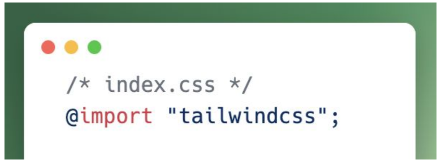
https://tailwindcss.com/docs/installation/using-vite

Configuration

Point Tailwind to your template files in **vite.config.ts** and add the directives to your main CSS file.



```ts
// vite.config.ts
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import tailwindcss from "@tailwindcss/vite";

// https://vite.dev/config/
export default defineConfig({
  plugins: [react(), tailwindcss()],
});
```



```css
/* index.css */
@import "tailwindcss";
```

State with Zustand

Zustand provides a minimal API to manage global state. Create a store hook that can be accessed from any component.

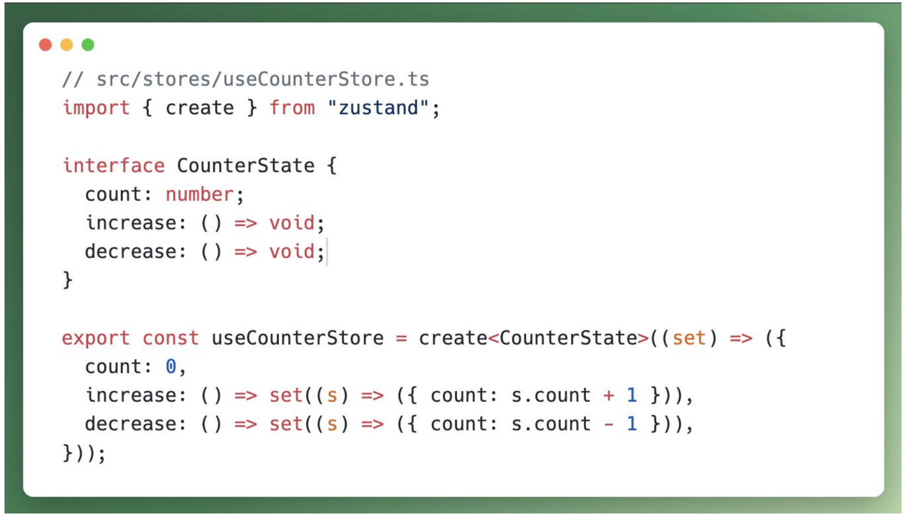
This eliminates **"prop drilling"** and makes managing user sessions or fetched data significantly cleaner.

Installation

```
npm install zustand
```

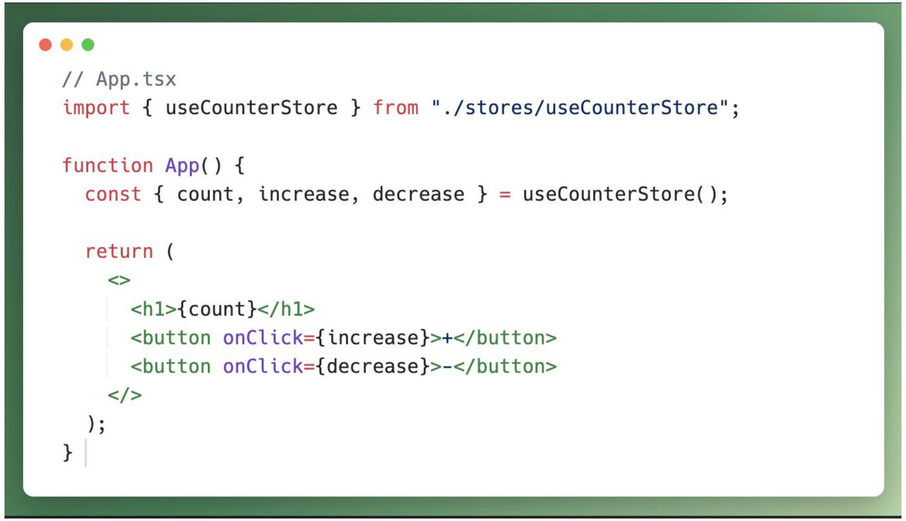
State with Zustand

Create Store



```ts
// src/stores/useCounterStore.ts
import { create } from "zustand";

interface CounterState {
  count: number;
  increase: () => void;
  decrease: () => void;
}

export const useCounterStore = create<CounterState>((set) => ({
  count: 0,
  increase: () => set((s) => ({ count: s.count + 1 })),
  decrease: () => set((s) => ({ count: s.count - 1 })),
}));
```

Use in Component



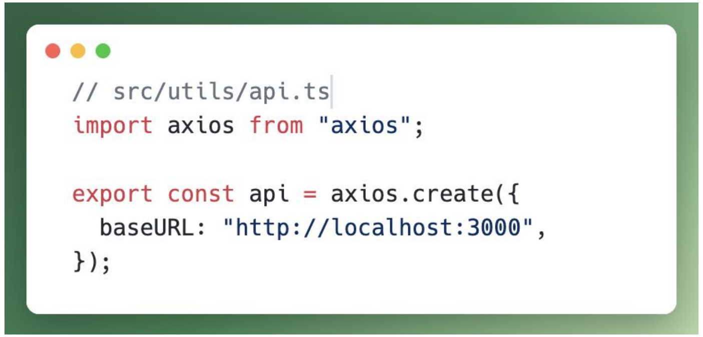
```tsx
// App.tsx
import { useCounterStore } from "./stores/useCounterStore";

function App() {
  const { count, increase, decrease } = useCounterStore();

  return (
    <>
      <h1>{count}</h1>
      <button onClick={increase}>+</button>
      <button onClick={decrease}>-</button>
    </>
  );
}
```

Connecting Frontend to Backend

Install Axios in Frontend Project

```
npm install axios
```

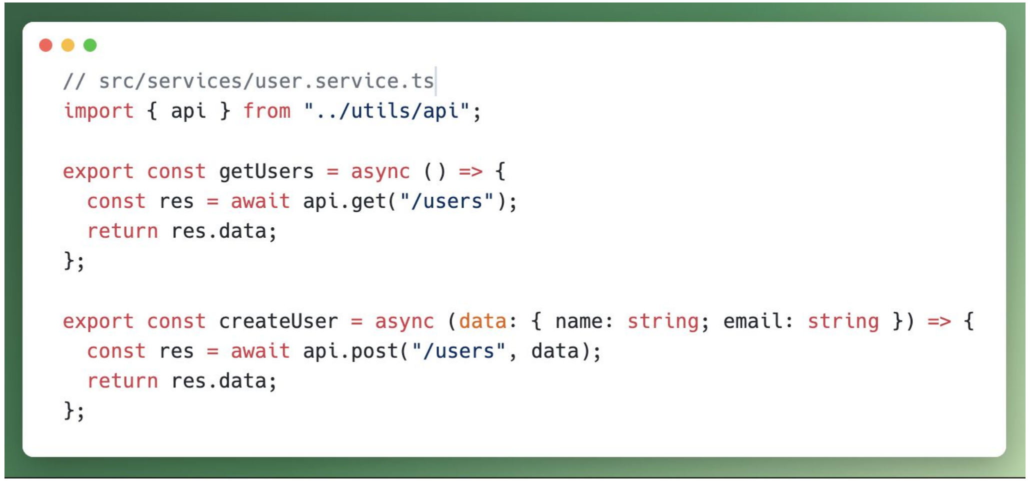
Create API Wrapper



```ts
// src/utils/api.ts
import axios from "axios";

export const api = axios.create({
  baseURL: "http://localhost:3000",
});
```

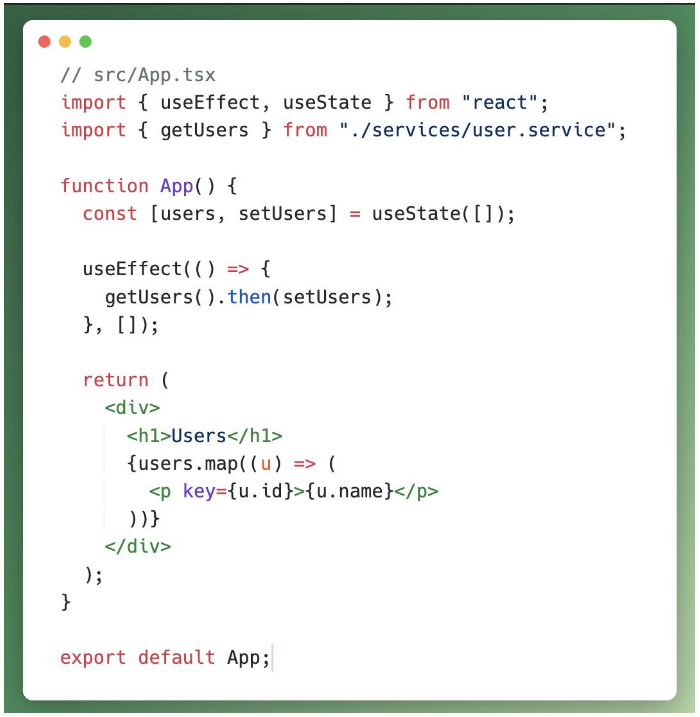
Connecting Frontend to Backend

API Call Service



```
// src/services/user.service.ts
import { api } from "../utils/api";

export const getUsers = async () => {
  const res = await api.get("/users");
  return res.data;
};

export const createUser = async (data: { name: string; email: string }) => {
  const res = await api.post("/users", data);
  return res.data;
};
```

Use in React Component



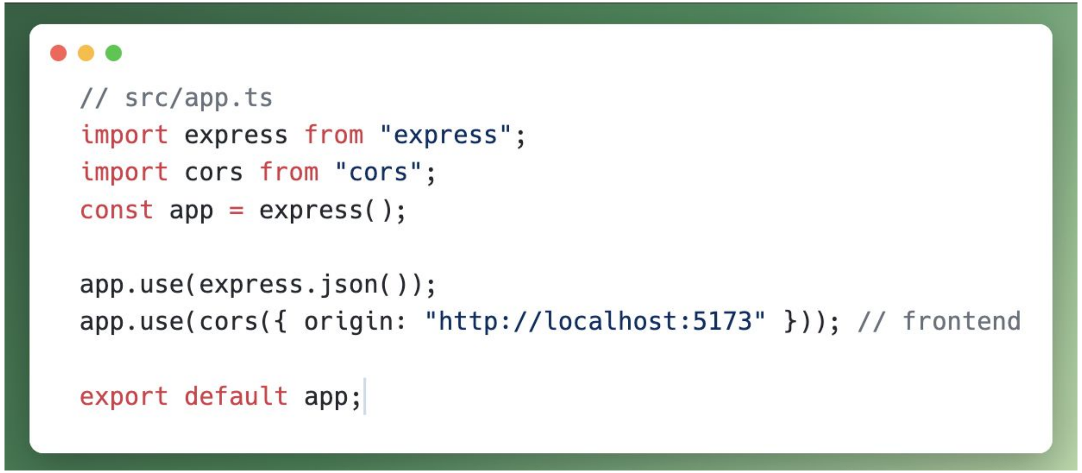
```
// src/App.tsx
import { useEffect, useState } from "react";
import { getUsers } from "./services/user.service";

function App() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    getUsers().then(setUsers);
  }, []);

  return (
    <div>
      <h1>Users</h1>
      {users.map((u) => (
        <p key={u.id}>{u.name}</p>
      ))}
    </div>
  );
}

export default App;
```

CORS Setup on Backend

Install cors

```
npm install cors
npm install @types/cors -D
```

Add in Express



```
// src/app.ts
import express from "express";
import cors from "cors";
const app = express();

app.use(express.json());
app.use(cors({ origin: "http://localhost:5173" })); // frontend

export default app;
```

Exercise

Build a simple User Management App (CRUD) with a connected frontend and backend

- Backend: Express + TypeScript, Prisma ORM, PostgreSQL database, REST API (GET /users, POST /users).
- Frontend: Vite + React + TypeScript, Axios for API calls, UI to list and add users.
- Database: User model with id, name, email (unique).
- Connection: CORS enabled or Vite proxy; Axios baseURL set to backend.
- Output: Working CRUD flow, form submission creates user ➔ saved in Postgres ➔ auto-refresh list on UI.

Thank you