Init Node/Express/Prisma project

Prerequisites

First, make sure you have the following installed in your local development environment:

Node.js version >= 12.x.x installed (`node --version`)

Access to one package manager such as npm or yarn

Init project

`mdkir <project_name> && cd ./<project_name> && npm init -y`

Install express

`yarn add express`

Add Typescript

`yarn add -D typescript ts-node`

Then create a `tsconfig.json` and fill it with the following code:

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "rootDir": "./src/",
    "outDir": "./build",
    "esModuleInterop": true,
    "strict": true
  }
}
```

Next you have to create `src/` folder, wich will contain all the logic of your back-end.

Add type definitions for Node & Express

`yarn add -D @types/node @types/express`

Create root file to launch server

`touch ./src/index.ts`

And fill it with :

```
import express from "express";

const app = express();
const indexRouter = require("./routes/index");

app.use(express.json());
app.use("/", indexRouter);
const PORT = 8000;
app.listen(PORT, () => {
  console.log(`⚡[server]: Server is running at http://localhost:${PORT}`);
});
```

Let's explain this code:
First we have import statement of express wich will permit us to initialize an express server via `const app = express()`.
Next we import all the routes we will implement for our API.
Let's create it.

Initialize the root route

Create a `./src/routes/` folder with an `index.ts` file inside. The folder will contain all the routes you will make through your project and the `index.ts` will centralize all of them in one file to avoid multiple imports.

Write these 4 lines of code to initialize the root route :

```
 import express from "express";
const router = express.Router();
router.use("/", (req, res)=> {
    res.send("Hello World")
});
module.exports = router;
```

Here, when you'll call your localhost:8000, it will respond whis an "Hello World" string. All other routes will respond a 404 Not Found.

Let's return to our `./src/index.ts`. The last line here make the server listen to the port define just above: 8000. Each request to localhost:8000 will pass through the indexRouter and resolve or not the urls to send responses.

Install Nodemon to watch file changes

To avoid kill and start the server at every change you made, let's install Nodemon. It will listen to every file of the project and when one of them is updated, it will reload automaticaly your server.

`yarn add -D nodemon`

Update `package.json` file by adding in `script` : `"start": "nodemon src/index.ts",`

In order to build the project, add too : `"build": "tsc --project ./",`

Now you will have a basic node+express server running with the command `yarn start` or `npm start`

Install prisma

`npm install prisma @types/node --save-dev`

Update `tsconfig.json` to match this:

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "rootDir": "./src/",
    "outDir": "./build",
    "esModuleInterop": true,
    "strict": true,
    "sourceMap": true,
    "lib": ["esnext"]
  }
}
```

Next, run `npx prisma init` to create the schema of your database.
It will create a `prisma/` folder with a schema.prisma file wich will contain the models of your database.

Next step is to set the DATABASE_URL in your .env file.
`DATABASE_URL=postgresql://<username>:<password>@localhost:5432/<database_name>`

Once it's done, run `npx prisma generate` to install prisma client and create a `/utils/prisma.ts` filled with:

```
import { PrismaClient } from "@prisma/client";

let prisma: any;

declare global {
    namespace NodeJS {
      interface Global {
        prisma: any;
```

```
    }
  }
}

if (process.env.NODE_ENV === "production") {
  prisma = new PrismaClient();
} else {
  if (!global.prisma) {
    global.prisma = new PrismaClient();
  }

  prisma = global.prisma;
}

export default prisma;
```

This file will permit you to use the prisma client object with all his functions. Just think about import it when needed.

Database already exists

Run `npx prisma db pull` to automaticaly generate your prisma.schema file with all your database models. Works too with `npx prisma introspect`.

Database doesn't exist

In `/prisma/schema.prisma` you have to define your different tables your database will contain.
For example to create a users table :

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

Run the command `npx prisma db push` to forward this update to your database.

First prisma request

We will use the route/controller/model to split the logic and make the project updatable easier.

So first, create in your `./src` folder 2 subfolders: `controllers` and `models`.

Let's start with the route folder now. Create a new file called `users.route.ts` and fill it :

```
import express from "express";
import { getAllUsers } from "../controllers/notes.controller";

const router = express.Router();
router.get("/", getAllUsers);

module.exports = router;
```

And in your `routes/index.ts` replace the router.use() by this one : `router.use("/users", require("./users.route"));`
Like this we link the "/users" route with our `/routes/users.route.ts`.

You probably see that there is an import in the users.route file. Let's make it work.

In the `src/controllers/` folder, create a `users.controller.ts` file. It will contain the logic of the request.

```
import { Request, Response } from "express";
import { getAllUsersModel } from "../models/users.model";

export const getAllUsers = async (req: Request, res: Response) => {
  const users = await getAllUsersModel();
  res.json(users);
};
```

In that case, we just call a function that make the request, but if you need to transform some data before return them in response, we can do it here.
Don't forget to return the result via `res.json()`.

Finally, the last function is imported from a model. The requests are exported like this to permit other function to use them without trigger an entire process.

Let's create the `/models/users.model.ts` file and implement it:

```
import prisma from "../utils/prisma";

export const getAllUsersModel = async () => {
  const users = await prisma.users.findMany();
  return users;
};
```

Do you remember the prisma client we initialized before ? Let's use it.
getAllUsersModel use findMany() function of prisma client. We just need to provide the table where we want to request.

findMany return an array to the controller, who can transform data if needed and return it to the client with res.json().