Node Express Mongoose TS

Alright — let's go step-by-step and actually *build* something with **Node.js**, **Express**, and **Mongoose** in **TypeScript** so you understand both **how** and **why** each piece works.

We'll go in three phases:

1. **Setup & TypeScript configuration**
2. **Express server with routing**
3. **Mongoose integration (CRUD)**

---

1. Setup & Project Structure

📁 **Folder structure:**

```pgsql
backend-ts/
├── src/
│   ├── config/
│   │   └── db.ts
│   ├── models/
│   │   └── User.ts
│   ├── routes/
│   │   └── userRoutes.ts
│   ├── controllers/
│   │   └── userController.ts
│   ├── server.ts
├── package.json
├── tsconfig.json
├── .env
```

Step 1: Initialize project

```bash
```

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

Step 2: Install dependencies

```bash
npm install express mongoose dotenv
npm install --save-dev typescript ts-node-dev @types/node @types/express @types/mongoose
```

- `express` → Web framework
- `mongoose` → MongoDB ODM (Object Data Modeling)
- `dotenv` → Load environment variables
- `typescript` → TypeScript compiler
- `ts-node-dev` → Run TS directly in dev mode (auto restart)
- `@types/...` → Type definitions for TS

Step 3: Create `tsconfig.json`

```json
{
  "compilerOptions": {
    "target": "ESNext",
    "module": "CommonJS",
    "rootDir": "./src",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "skipLibCheck": true
  }
}
```

Explanation:

- `rootDir` → Where TypeScript looks for `.ts` files

- `outDir` → Where compiled JS goes
- `strict` → Better type safety
- `esModuleInterop` → Allows `import express from 'express'`

Step 4: Update `package.json` scripts

```
"scripts": {
  "dev": "ts-node-dev --respawn --transpile-only src/server.ts",
  "build": "tsc",
  "start": "node dist/server.js"
}
```

---

2. Express Server Setup

📄 `src/server.ts`

```ts
import express, { Application } from "express";
import dotenv from "dotenv";
import connectDB from "./config/db";
import userRoutes from "./routes/userRoutes";

dotenv.config();

const app: Application = express();

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

// Routes
app.use("/api/users", userRoutes);

// Start server
const PORT = process.env.PORT || 5000;
app.listen(PORT, async () => {
  await connectDB(); // Connect to MongoDB before accepting requests
```

```
  console.log(`Server running on port ${PORT}`);
});
```

---

3. MongoDB Connection with Mongoose

📄 `src/config/db.ts`

```ts
import mongoose from "mongoose";

const connectDB = async (): Promise<void> => {
  try {
    if (!process.env.MONGO_URI) {
      throw new Error("MONGO_URI is not defined in .env");
    }
    await mongoose.connect(process.env.MONGO_URI);
    console.log("MongoDB connected successfully");
  } catch (error) {
    console.error("MongoDB connection error:", error);
    process.exit(1); // Exit process if DB connection fails
  }
};

export default connectDB;
```

---

4. Create a Model in TypeScript

📄 `src/models/User.ts`

```ts
import mongoose, { Schema, Document } from "mongoose";

// Interface for TypeScript type safety
export interface IUser extends Document {
  name: string;
  email: string;
  password: string;
}
```

```ts
// Mongoose schema
const userSchema: Schema<IUser> = new Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true }
  },
  { timestamps: true }
);

export default mongoose.model<IUser>("User", userSchema);
```

---

5. Controller Functions

📄 `src/controllers/userController.ts`

```ts
import { Request, Response } from "express";
import User, { IUser } from "../models/User";

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

    const newUser: IUser = new User({ name, email, password });
    await newUser.save();

    res.status(201).json(newUser);
  } catch (error) {
    res.status(500).json({ message: "Error creating user", error });
  }
};

export const getUsers = async (_req: Request, res: Response) => {
  try {
    const users = await User.find();
    res.json(users);
  } catch (error) {
    res.status(500).json({ message: "Error fetching users", error });
```

```
  }
};
```

---

6. Routes

📄 `src/routes/userRoutes.ts`

```ts
import { Router } from "express";
import { createUser, getUsers } from "../controllers/userController";

const router: Router = Router();

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

export default router;
```

---

7. `.env` File

```ini
PORT=5000
MONGO_URI=mongodb://localhost:27017/backend_ts
```

---

8. Run the project

```bash
npm run dev
```

Visit:

- **POST** → `http://localhost:5000/api/users` with JSON body:

```json
```

```
{ "name": "John Doe", "email": "john@example.com", "password": "123456" }
```

- **GET** → `http://localhost:5000/api/users`

Key TypeScript Benefits in Backend

1. **Type safety** → Request/Response, Model shapes
2. **Interfaces** → Guarantee data consistency
3. **Better IntelliSense** → Autocomplete & error catching before runtime
4. **Cleaner code** → Modular structure + TS enforcing best practices

---

If you want, I can now **extend this example** to include:

- Password hashing with **bcrypt**
- JWT authentication
- DTO (Data Transfer Object) pattern for stricter request validation
- Middleware for error handling

That would make this a **production-ready TypeScript backend**.

Do you want me to continue with that advanced version?