# 🚀 Express.js + TypeScript Backend Development Guide

This note explains how to build a backend API using:

- Express.js
- TypeScript
- MongoDB (Mongoose)
- PostgreSQL (pg)
- Multer (image upload)

At the end, we build a small Blog API with image upload (no authentication).

---

## Why Use TypeScript with Express?

TypeScript gives:

- Static typing
- Better autocomplete
- Safer refactoring
- Clear data contracts
- Improved scalability

Instead of runtime errors, you catch many issues during development.

---

# 🏗️ Project Setup

## Initialize Project

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

## Install Dependencies

```
npm install express mongoose pg multer cors dotenv
npm install -D typescript ts-node-dev @types/express @types/node @types/
multer
```

### Create TypeScript Config

```
npx tsc --init
```

Update tsconfig.json:

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "rootDir": "src",
    "outDir": "dist",
    "strict": true,
    "esModuleInterop": true
  }
}
```

---

## Recommended Folder Structure

```
src/
  app.ts
  server.ts
  config/
    mongo.ts
    postgres.ts
  models/
    Blog.ts
  routes/
    blog.routes.ts
  controllers/
    blog.controller.ts
  types/
    blog.types.ts
  uploads/
```

---

## Connecting Databases

---

# 🛩️ MongoDB with Mongoose

### src/config/mongo.ts

```
import mongoose from "mongoose";

export const connectMongo = async () => {
  await mongoose.connect(process.env.MONGO_URI as string);
  console.log("MongoDB connected");
};
```

---

# 🛫 PostgreSQL with pg

### src/config/postgres.ts

```
import { Pool } from "pg";

export const pool = new Pool({
  connectionString: process.env.POSTGRES_URL
});

export const connectPostgres = async () => {
  await pool.query("SELECT 1");
  console.log("PostgreSQL connected");
};
```

---

# 🛖 Type Definitions

### src/types/blog.types.ts

```
export interface BlogInput {
  title: string;
  content: string;
  image?: string;
}
```

---

# Blog Model (MongoDB)

## src/models/Blog.ts

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

export interface BlogDocument extends Document {
  title: string;
  content: string;
  image?: string;
  createdAt: Date;
}

const BlogSchema = new Schema<BlogDocument>(
  {
    title: { type: String, required: true },
    content: { type: String, required: true },
    image: { type: String }
  },
  { timestamps: true }
);

export const Blog = mongoose.model<BlogDocument>("Blog", BlogSchema);
```

---

# 🗾 Multer Configuration

## src/config/multer.ts

```
import multer from "multer";
import path from "path";

const storage = multer.diskStorage({
  destination: "src/uploads",
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname));
  }
});

export const upload = multer({ storage });
```

---

# 🏠 Blog Controller

### src/controllers/blog.controller.ts

```
import { Request, Response } from "express";
import { Blog } from "../models/Blog";

export const createBlog = async (req: Request, res: Response) => {
  try {
    const { title, content } = req.body;

    const image = req.file ? req.file.filename : undefined;

    const blog = await Blog.create({ title, content, image });

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

export const getBlogs = async (req: Request, res: Response) => {
  const blogs = await Blog.find().sort({ createdAt: -1 });
  res.json(blogs);
};
```

---

# 🛣️ Blog Routes

### src/routes/blog.routes.ts

```
import { Router } from "express";
import { createBlog, getBlogs } from "../controllers/blog.controller";
import { upload } from "../config/multer";

const router = Router();

router.post("/", upload.single("image"), createBlog);
router.get("/", getBlogs);

export default router;
```

---

# Express App Setup

### src/app.ts

```
import express from "express";
import cors from "cors";
import blogRoutes from "./routes/blog.routes";

const app = express();

app.use(cors());
app.use(express.json());
app.use("/uploads", express.static("src/uploads"));

app.use("/api/blogs", blogRoutes);

export default app;
```

---

# 🚀 Server Entry

### src/server.ts

```
import dotenv from "dotenv";
import app from "./app";
import { connectMongo } from "./config/mongo";
import { connectPostgres } from "./config/postgres";

dotenv.config();

const start = async () => {
  await connectMongo();
  await connectPostgres();

  app.listen(5000, () => {
    console.log("Server running on port 5000");
  });
};

start();
```

---

# How to Run

```
npm run dev
```

Add to package.json:

```
"scripts": {
  "dev": "ts-node-dev --respawn src/server.ts"
}
```

---

# Example Request (Postman)

POST → http://localhost:5000/api/blogs

Form-Data:

- title → My Blog
- content → Hello world
- image → (upload file)

---

# Where PostgreSQL Can Be Used

You can use PostgreSQL for:

- Storing analytics
- Storing comments
- Storing structured data

Example raw query:

```
await pool.query(
  "INSERT INTO comments (blog_id, content) VALUES ($1, $2)",
  [blogId, comment]
);
```

---

# 🚢 Final Summary

Using TypeScript with Express gives:

- Strong typing
- Better scalability
- Cleaner architecture

Mongoose handles flexible blog documents. PostgreSQL handles relational data. Multer handles image uploads.

You now have a working blog API with image upload using:

- Express
- TypeScript
- MongoDB
- PostgreSQL
- Multer

---

You are now building production-ready TypeScript backends 🚀