📝 Node.js Express Backend with MongoDB

This project is a simple backend built using **Node.js**, **Express.js**, and **MongoDB**, featuring user authentication and post creation functionality. Below is a breakdown of the file structure and an in-depth explanation of how everything works together.

📁 Project Structure

```
├── .env                # 🔐 Environment variables (like database URI)
├── models              # 📦 MongoDB data models
│   ├── Post.js         # 📝 Schema for storing posts
│   └── User.js         # 👤 Schema for storing users
├── node_modules        # 📚 Installed dependencies (auto-generated)
├── package.json        # 📦 Project metadata and dependencies
├── package-lock.json   # 🔒 Lock file for exact versions of dependencies
├── routes
│   └── auth.js         # 🚪 API routes for authentication and post creation
└── server.js           # 🚀 Entry point to run the server
```

🔧 Environment Variables (.env)

```
PORT=5000
MONGO_URI=mongodb+srv://<username>:<password>@cluster.mongodb.net/myDatabase
```

- `PORT` : Port number your server runs on.
- `MONGO_URI` : MongoDB connection string.

📦 Models

👤 models/User.js

```
const mongoose = require("mongoose"); // 🍃 Importing mongoose for MongoDB

// 🧱 Define the user schema
const userSchema = new mongoose.Schema({
  name: { type: String, required: true }, // 🧍 User's name
  email: { type: String, required: true, unique: true }, // 📧 Unique email
```

```
  password: { type: String, required: true }, // 🔐 Password (not hashed yet)
});

// 📦 Exporting the model
module.exports = mongoose.model("User", userSchema);
```

📝 models/Post.js

```
const mongoose = require("mongoose"); // 🍃 Mongoose to create schema

// 🧱 Define the post schema
const postSchema = new mongoose.Schema({
  postType: { type: String, required: true }, // 🏷️ Post category/type
  content: { type: String, required: true }, // 🖊️ Post content
  dateCreated: { type: Date, default: Date.now }, // 🗓️ Auto timestamp
});

// 📦 Export the Post model
module.exports = mongoose.model("Post", postSchema);
```

🚀 Server Setup: server.js

```
const express = require("express"); // 🌐 Framework for API building
const mongoose = require("mongoose"); // 🍃 MongoDB object modeling
const cors = require("cors"); // 🔓 Handle cross-origin requests
require("dotenv").config(); // 🔐 Load environment variables

const app = express(); // 🎉 Initialize Express app
const PORT = process.env.PORT || 5000; // 🚦 Use .env PORT or fallback to 5000

app.use(cors()); // 🔓 Allow CORS (frontend can call backend)
app.use(express.json()); // 📦 Parse JSON bodies

// 🛠️ Connect to MongoDB
mongoose
  .connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => console.log("MongoDB connected")) // ✅ Success message
  .catch((err) => console.log(err)); // ❌ Error handler

// 📌 Set up route middleware
app.use("/api", require("./routes/auth")); // 🔗 Routes defined in routes/auth.js
```

```
// 🚀 Start the server
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

🚪 Routes: `routes/auth.js`

```
const express = require("express"); // 🌐 Express router
const router = express.Router();
const User = require("../models/User"); // 👤 Import User model
const Post = require("../models/Post"); // 📝 Import Post model

// 📝 Sign Up Route
router.post("/sign-up", async (req, res) => {
  const { name, email, password, confirmPassword } = req.body;

  // ❗ Check if passwords match
  if (password !== confirmPassword) {
    return res.status(400).json({ error: "Passwords do not match" });
  }

  try {
    // 🔍 Check for existing user
    const existingUser = await User.findOne({ email });
    if (existingUser)
      return res.status(400).json({ error: "Email already exists" });

    // 💾 Create and save new user
    const user = new User({ name, email, password }); // NOTE: No hashing yet
    await user.save();

    res.status(201).json({ message: "User created successfully" });
  } catch (err) {
    res.status(500).json({ error: "Server error" });
  }
});

// 🔐 Sign In Route
router.post("/sign-in", async (req, res) => {
  const { email, password } = req.body;

  try {
    const user = await User.findOne({ email });

    // ❌ No user found
    if (!user) return res.status(400).json({ error: "User not found" });

    // ❌ Wrong password
    if (user.password !== password) {
      return res.status(400).json({ error: "Incorrect password" });
    }
```

```
    // ✅ Login successful
    res.status(200).json({ message: "Login successful", user });
  } catch (err) {
    console.error("Sign-in error:", err);
    res.status(500).json({ error: "Server error" });
  }
});

// 📝 Create Post Route
router.post("/create-post", async (req, res) => {
  const { postType, content } = req.body;

  try {
    // 💾 Save new post
    const post = new Post({ postType, content });
    await post.save();

    res.status(201).json({ message: "Post created successfully!" });
  } catch (err) {
    console.error("Post creation error:", err);
    res.status(500).json({ error: "Server error" });
  }
});

module.exports = router; // 🔗 Export router
```

🔁 Workflow Summary

1. **Start the server** with `node server.js` or `nodemon`.
2. **Connects to MongoDB** using the `.env` MONGO_URI.
3. Routes are set up at `/api`:
   - POST `/api/sign-up` - Register a new user 👤
   - POST `/api/sign-in` - Log in existing user 🔐
   - POST `/api/create-post` - Create a new post 📝

🔗 How This Node.js + Express + MongoDB App Works

This project is a basic web backend with user authentication and post creation, organized into multiple files that talk to each other. Let's break it all down 👇

What this app does:

- Accepts user registration (`sign-up`)

- Allows users to log in (`sign-in`)
- Lets users create posts (`create-post`)
- Stores all data in **MongoDB**
- Runs using a server built with **Express.js**

📁 Folder & File Connections

```
Root Directory
│
├── .env                🔐 Secrets like your MongoDB URL
├── server.js           🚀 Main server file that connects everything
├── routes
│   └── auth.js         📬 API endpoints for sign-up, sign-in, and post
├── models
│   ├── User.js         👤 Mongoose model for users
│   └── Post.js         📝 Mongoose model for posts
├── package.json        📦 Lists dependencies and scripts
└── node_modules        📚 All packages installed (auto-generated)
```

🔌 File-by-File Breakdown

1️⃣ `.env` – 🔐 Environment Settings

- Stores sensitive info like MongoDB URI.
- Not shared publicly (listed in `.gitignore` usually).

Example:

```
PORT=5000
MONGO_URI=your-mongodb-connection-string
```

2️⃣ `server.js` – 🚀 Main Server File

- This is the heart of the backend.
- Loads all dependencies, sets up Express, connects to MongoDB, and uses routes.

What it does:

- Loads environment variables from `.env`

- Connects to the MongoDB database
- Uses JSON and CORS middleware
- Registers `/api` routes from `auth.js`
- Starts the server

➡️ **Connected to:**

- `.env` for the MongoDB URI
- `routes/auth.js` for handling user/post routes

3️⃣ `models/User.js` – 👤 User Data Schema

- Describes how a user is stored in MongoDB.
- Uses `mongoose.Schema` to define fields like name, email, password.

Why it's important:

- Keeps data structured.
- Automatically creates a `users` collection in MongoDB.

➡️ **Used in:**

- `auth.js` when registering or logging in a user.

4️⃣ `models/Post.js` – 📝 Post Data Schema

- Describes how posts are saved.
- Has fields for `postType`, `content`, and `dateCreated`.

➡️ **Used in:**

- `auth.js` when a user creates a new post.

5️⃣ `routes/auth.js` – 🚪 Authentication and Post Routes

This file contains all the API endpoints:

- `POST /api/sign-up` – Registers a new user
- `POST /api/sign-in` – Logs in a user
- `POST /api/create-post` – Saves a post to MongoDB

➡️ **Connected to:**

- `User.js` and `Post.js` for data models
- Used in `server.js` via `app.use("/api", require("./routes/auth"));`

6️⃣ `package.json` – 📦 Project Setup

- Lists all your packages (like Express, Mongoose, CORS, dotenv).
- Defines scripts (like `start`) to run the server easily.

⚙️ How Everything Works Together (Step-by-Step)

1. You run the server with a command (see below ⬇️).
2. `server.js`:
   - Loads your `.env` file
   - Connects to MongoDB
   - Prepares the API using `auth.js`
3. When you hit an API like `/api/sign-up`, it:
   - Goes to `routes/auth.js`
   - Uses `User.js` or `Post.js` models to talk to MongoDB
   - Responds with success/failure message
4. MongoDB saves and returns data via those models.

🖥️ Start the Server

✅ First, install all dependencies (only once):

```
npm install
```

▶️ To start the server:

```
node server.js
```

Or if you've set this in `package.json` under `"scripts"` like this:

```
"scripts": {
  "start": "node server.js"
}
```

You can simply run:

```
npm start
```

🚀 API Endpoints Overview

Endpoint	Method	Description
`/api/sign-up`	POST	Create new user
`/api/sign-in`	POST	Login existing user
`/api/create-post`	POST	Create a new post

✅ Why This Setup is Good

- **Separation of concerns**: Models, routes, and server config are kept clean and separate.
- **Reusable**: You can build on this easily — like adding JWT or bcrypt later.
- **Scalable**: You can add more routes, models, or features without clutter.