📁 Project Folder Structure

bash Copy Edit

```bash
/my-app
├── /node_modules
├── /prisma
│   ├── schema.prisma          # Defines data models and database schema
│   ├── seed.js                # Populates the database with initial data
│   └── migrations/            # Auto-generated migration files
│
├── /src
│   ├── /config                # Configuration files (e.g., environment variable
│   │   └── index.js
│   │
│   ├── /controllers           # Request handlers (business logic)
│   │   └── userController.js
│   │
│   ├── /middlewares           # Express middleware functions
│   │   └── errorHandler.js
│   │
│   ├── /routes                # Route definitions
│   │   └── userRoutes.js
│   │
│   ├── /services              # Business logic and interactions with Prisma
│   │   └── userService.js
│   │
│   ├── /repositories          # Data access layer (Prisma Client interactions)
│   │   └── userRepository.js
│   │
│   ├── /validators            # Input validation schemas
│   │   └── userValidator.js
│   │
│   ├── /utils                 # Utility functions
│   │   └── logger.js
│   │
│   ├── /types                 # Type definitions (if using TypeScript)
│   │   └── user.d.ts
│   │
│   ├── app.js                 # Express application setup
│   └── server.js              # Server entry point
│
├── .env                       # Environment variables
├── .gitignore
├── package.json
└── README.md
```

📂 Folder Purpose Overview

- `/prisma`: Contains Prisma schema, migrations, and seeding scripts.
- `/src`: Main application logic.
  - `/config`: Configuration settings (e.g., environment variables).
  - `/controllers`: Functions that handle HTTP requests and responses.
  - `/middlewares`: Functions that process requests before reaching route handlers.
  - `/routes`: Defines API endpoints and associates them with controllers.
  - `/services`: Contains business logic and interactions with the database.
  - `/repositories`: Manages data access and interactions with the database.
  - `/validators`: Validates incoming request data.
  - `/utils`: Helper functions and utilities.
  - `/types`: Type definitions (if using TypeScript).
  - `app.js`: Sets up the Express application.
  - `server.js`: Starts the server and listens for requests.

📄 Sample Code for Key Files

`prisma/schema.prisma`

prisma Copy Edit

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

`prisma/seed.js`

javascript Copy Edit

```javascript
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

async function main() {
  await prisma.user.create({
    data: {
      email: 'admin@example.com',
      name: 'Admin User',
    },
  });
  console.log('Database seeded');
}

main()
  .catch(e => {
    console.error(e);
    process.exit(1);
  })
  .finally(async () => {
    await prisma.$disconnect();
  });
```

src/controllers/userController.js

javascript Copy Edit

```javascript
const userService = require('../services/userService');

async function createUser(req, res) {
  try {
    const user = await userService.createUser(req.body);
    res.status(201).json(user);
  } catch (error) {
    res.status(500).json({ error: 'Failed to create user' });
  }
}

module.exports = { createUser };
```

src/services/userService.js

javascript Copy Edit

```javascript
const userRepository = require('../repositories/userRepository');

async function createUser(data) {
  return userRepository.create(data);
}

module.exports = { createUser };
```

src/repositories/userRepository.js

javascript Copy Edit

```javascript
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

async function create(data) {
  return prisma.user.create({ data });
}

module.exports = { create };
```

`src/routes/userRoutes.js`

javascript Copy Edit

```javascript
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.post('/users', userController.createUser);

module.exports = router;
```

`src/app.js` **(Updated)**

javascript Copy Edit

```javascript
const express = require('express');
const userRoutes = require('./routes/userRoutes');
const app = express();
app.use(express.json());
app.use('/api', userRoutes);
module.exports = app;
```

`src/middlewares/errorHandler.js`

javascript Copy Edit

```javascript
function errorHandler(err, req, res, next) {
  console.error(err.stack);
  res.status(500).send('Something went wrong!');
}

module.exports = errorHandler;
```

`src/server.js` **(Updated)**

javascript Copy Edit

```javascript
const app = require('./app');
const errorHandler = require('./middlewares/errorHandler');
const port = process.env.PORT || 3000;

app.use(errorHandler);

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

- **You reuse repository functions** in many services — DRY code.
- Easier to **mock** database operations in unit tests.
- If Prisma ever changes or you switch to another DB, you only update `repositories/`, not every service.