DOCS

# Backend Setup Documentation

## 1. Project Initialization

### Install Dependencies

Run the following command inside the `backend/` folder:

```sh

npm init -y

npm install express mongoose dotenv cors jsonwebtoken bcryptjs cookie-parser express-validator redis socket.io

npm install --save-dev nodemon jest supertest

```

### Project Structure

```

backend/

│── src/

│ ├── config/ # Database connection

│ ├── controllers/ # Business logic

│ ├── models/ # Mongoose schemas

│ ├── routes/ # API endpoints

│ ├── middleware/ # Auth, validation, logging

│ ├── services/ # External API logic (Plaid, notifications)

│ ├── utils/ # Helpers, error handling

│ ├── app.js # Express setup

│ ├── server.js # Entry point

│── .gitignore

│── package.json

│── README.md

```

---

## 2. Express Server Setup

### `server.js`

This is the entry point for our backend.

```javascript

import express from 'express';

import dotenv from 'dotenv';

import cors from 'cors';

import cookieParser from 'cookie-parser';

import connectDB from './config/db.js';

dotenv.config();

const app = express();

// Middleware

app.use(cors());

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

// Connect Database

connectDB();

// Test Route

app.get('/', (req, res) => {

res.send('API is running...');

});

const PORT = process.env.PORT || 5000;

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

```

---

## 3. Database Connection

### `db.js`

This file connects MongoDB Atlas to our application.

```javascript

import mongoose from 'mongoose';

const connectDB = async () => {

try {

const conn = await mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

});

console.log(`MongoDB Connected: ${conn.connection.host}`);

} catch (error) {

console.error(`Error: ${error.message}`);

process.exit(1);

}

};

export default connectDB;

```

---

## 4. Environment Variables

Create a `.env` file in the `backend/` folder:

```

PORT=5000

MONGO\_URI=mongodb+srv://<your-cluster>.mongodb.net/fintechDB

JWT\_SECRET=your\_secret\_key

```

---

## 5. Running the Server

Start the backend server:

```sh

node server.js

```

OR (for hot-reloading):

```sh

npx nodemon server.js

```

Visit `http://localhost:5000/` – You should see `API is running...`

---

# Authentication Module Documentation

## Overview

This module handles user authentication, including registration and login. It uses JWT-based authentication to secure API endpoints.

## Endpoints

### 1. User Registration

\*\*Endpoint:\*\* `POST /api/auth/register`

\*\*Description:\*\* Registers a new user with email, password, and role.

#### Request Body:

```json

{

"name": "John Doe",

"email": "johndoe@example.com",

"password": "password123",

"role": "customer"

}

```

#### Response:

```json

{

"\_id": "user\_id",

"name": "John Doe",

"email": "johndoe@example.com",

"role": "customer",

"token": "jwt\_token"

}

```

#### Possible Errors:

- \*\*400 Bad Request\*\* - Missing fields or invalid data.

- \*\*409 Conflict\*\* - Email already registered.

- \*\*500 Internal Server Error\*\* - Server issues.

---

### 2. User Login

\*\*Endpoint:\*\* `POST /api/auth/login`

\*\*Description:\*\* Authenticates a user and returns a JWT token.

#### Request Body:

```json

{

"email": "johndoe@example.com",

"password": "password123"

}

```

#### Response:

```json

{

"\_id": "user\_id",

"name": "John Doe",

"email": "johndoe@example.com",

"role": "customer",

"token": "jwt\_token"

}

```

#### Possible Errors:

- \*\*400 Bad Request\*\* - Invalid email or password.

- \*\*500 Internal Server Error\*\* - Server issues.

---

## Middleware

### Authentication Middleware

Verifies JWT tokens for protected routes.

```javascript

import jwt from 'jsonwebtoken';

import User from '../models/User.js';

export const authMiddleware = async (req, res, next) => {

const token = req.header('Authorization');

if (!token) return res.status(401).json({ message: 'Access denied' });

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.user = await User.findById(decoded.id).select('-password');

next();

} catch (err) {

res.status(401).json({ message: 'Invalid token' });

}

};

```

---

## Testing

We used \*\*Jest\*\* and \*\*Supertest\*\* to validate authentication functionality.

### \*\*Test Cases:\*\*

- ✅ Should register a new user (`201 Created`)

- ✅ Should login an existing user (`200 OK`)

- ✅ Should return errors for invalid credentials

#### Example Test:

```javascript

test('Should register a new user', async () => {

const res = await request(app).post('/api/auth/register').send({

name: 'Test User',

email: 'test@example.com',

password: 'password123',

role: 'customer'

});

expect(res.statusCode).toBe(201);

expect(res.body).toHaveProperty('token');

});

```

---