**1️ API Development Guidelines**

Each API endpoint should follow this **standard structure**:

**📌 Endpoint Format**

Each API route must:

* Use **RESTful** naming conventions.
* Return **JSON responses**.
* Be structured as follows:

app.method('/endpoint', middleware, async (req, res) => {

try {

// Logic here

res.status(200).json({ message: 'Success' });

} catch (error) {

res.status(500).json({ error: error.message });

}

});

* Example:

app.post('/login', async (req, res) => {

try {

const { username, password } = req.body;

const user = await db.getUserByUsername(username);

if (!user) return res.status(401).json({ message: 'Invalid username' });

const match = await bcrypt.compare(password, user.PasswordHash);

if (!match) return res.status(401).json({ message: 'Invalid password' });

req.session.user = { id: user.UserID, username: user.Username };

res.json({ message: 'Login successful', user: req.session.user });

} catch (error) {

res.status(500).json({ error: error.message });

}

});

**📌 API Endpoint Structure**

| **Functionality** | **Method** | **Endpoint** | **Description** |
| --- | --- | --- | --- |
| **User Login** | POST | /login | Authenticate users and create a session. |
| **User Logout** | POST | /logout | Ends the user session. |
| **Session Check** | GET | /me | Returns logged-in user details. |
| **Register User** | POST | /register | Creates a new user account. |
| **Get Teams** | GET | /teams | Retrieves all teams. |
| **Get Games** | GET | /games | Retrieves all scheduled games. |
| **Chat Messages** | GET | /chat/:roomId | Fetches messages for a chatroom. |
| **Post Chat Message** | POST | /chat/:roomId | Sends a new chat message. |
| **Place a Bet** | POST | /bets | Allows users to place a bet. |
| **Check Bet Status** | GET | /bets/:betId | Retrieves bet details. |

**2️ Backend Module Organization**

Each module (e.g., authentication, game management) should be separated into **controllers** and **routes**.

**📌 Folder Structure**

sportschat-backend/

│── server.js

│── .env

│── routes/

│ ├── authRoutes.js

│ ├── chatRoutes.js

│ ├── gameRoutes.js

│ ├── userRoutes.js

│── controllers/

│ ├── authController.js

│ ├── chatController.js

│ ├── gameController.js

│ ├── userController.js

│── models/

│ ├── User.js

│ ├── Team.js

│ ├── Game.js

│── middleware/

│ ├── authMiddleware.js

│── config/

│ ├── db.js

│── package.json

**📌 Example: Authentication Route**

📍 routes/authRoutes.js

const express = require('express');

const { login, logout, getCurrentUser } = require('../controllers/authController');

const router = express.Router();

router.post('/login', login);

router.post('/logout', logout);

router.get('/me', getCurrentUser);

module.exports = router;

📍 controllers/authController.js

const bcrypt = require('bcrypt');

const db = require('../config/db');

exports.login = async (req, res) => {

try {

const { username, password } = req.body;

const user = await db.getUserByUsername(username);

if (!user) return res.status(401).json({ message: 'Invalid username' });

const match = await bcrypt.compare(password, user.PasswordHash);

if (!match) return res.status(401).json({ message: 'Invalid password' });

req.session.user = { id: user.UserID, username: user.Username };

res.json({ message: 'Login successful', user: req.session.user });

} catch (error) {

res.status(500).json({ error: error.message });

}

};

exports.logout = (req, res) => {

req.session.destroy(err => {

if (err) return res.status(500).json({ message: 'Logout failed' });

res.json({ message: 'Logged out successfully' });

});

};

exports.getCurrentUser = (req, res) => {

if (!req.session.user) {

return res.status(401).json({ message: 'Not logged in' });

}

res.json({ user: req.session.user });

};

**3️ Frontend-Backend Interaction**

All API calls from the frontend should use api.js to ensure **consistency**.

📍 src/services/api.js

import axios from 'axios';

const API = axios.create({

baseURL: 'http://localhost:5000',

withCredentials: true, // Ensures sessions are handled correctly

});

export const login = async (username, password) => {

try {

const response = await API.post('/login', { username, password });

return response.data;

} catch (error) {

console.error('Login error:', error.response?.data || error.message);

throw error;

}

};

export const logout = async () => {

try {

await API.post('/logout');

} catch (error) {

console.error('Logout error:', error.response?.data || error.message);

throw error;

}

};

export const getCurrentUser = async () => {

try {

const response = await API.get('/me');

return response.data;

} catch (error) {

console.error('Session check failed:', error.response?.data || error.message);

throw error;

}

};

export default API;

**4️ UI Implementation for a Guest**

Each UI element should be structured properly to interact with the backend.

📍 **Login Component**

import React, { useState } from 'react';

import { login } from '../services/api';

const Login = () => {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const [message, setMessage] = useState('');

const handleLogin = async (e) => {

e.preventDefault();

try {

const response = await login(username, password);

setMessage(`Welcome, ${response.user.username}!`);

} catch (error) {

setMessage('Login failed. Check console for details.');

}

};

return (

<div>

<h2>Login</h2>

<form onSubmit={handleLogin}>

<input type="text" placeholder="Username" value={username} onChange={(e) => setUsername(e.target.value)} />

<input type="password" placeholder="Password" value={password} onChange={(e) => setPassword(e.target.value)} />

<button type="submit">Login</button>

</form>

<p>{message}</p>

</div>

);

};

export default Login;