**Project: Mystelio - Social Media Application**

**Introduction:**

Welcome to the kickoff of our Social Media Application project. This document outlines the key aspects of our project, including the chosen tech stack, modules, and initial tasks for Day 1.

**Tech Stack:**

* **Frontend:** React
* **Backend:** Express, Node.js
* **Database:** MySQL
* **Styling:** Custom CSS with Styled Components

**Modules:**

1. **User Authentication**
2. **Direct Messaging**
3. **User Profiles**
4. **Feed/Posts**
5. **Notifications**

**Database Schema:**

* Users Table
* Messages Table
* Posts Table
* Notifications Table

**Additional Points:**

* Utilize Styled Components for dynamic styling.
* Implement token-based authentication for security.
* Explore real-time communication libraries for instant messaging.

**Day 2 Tasks:**

1. Set up the project structure.
2. Initialize React frontend and Express backend.
3. Create basic UI components for login and signup.
4. Establish the MySQL database connection.

**Day 2: Logo Design**

**Logo Concept:**

Our application, Mystelio, needs a unique and representative logo that captures the essence of social interaction and mystery. The chosen color palette sets the tone for a vibrant and engaging visual identity.

**Color Palette:**

:root {

--thistle: #cdb4dbff;

--fairy-tale: #ffc8ddff;

--carnation-pink: #ffafccff;

--uranian-blue: #bde0feff;

--light-sky-blue: #a2d2ffff;

}

**Logo Elements:**



1. **Circle Background:**

.circle {

width: 120px;

height: 120px;

background-color: var(--uranian-blue);

border-radius: 50%;

display: flex;

align-items: center;

justify-content: center;

margin-right: 16px; /\* Adjust as needed \*/

}

**Logo Composition**

<div class="logo-container">

<div class="circle">

<span class="logo">M</span>

</div>

<div class="tagline">Mystelio - Connect in Style</div>

</div>

**Day 2: Frontend Setup and Basic Pages**

**Objective:**

1. Set up the frontend project.
2. Implement a responsive sidebar.
3. Design login and register pages.
4. Apply basic styling using the provided color palette.

**Tasks Completed:**

**1. Frontend Project Setup:**

* Initialized a React project using Create React App.
* Organized project structure with folders for components, assets, and styles.
* Integrated Font Awesome for icons.
* Set up basic CSS reset and color palette.

**2. Responsive Sidebar:**

* Created a responsive sidebar with a toggle button.
* Implemented a smooth transition for the sidebar.

**3. Login and Register Pages:**

* Designed login and register pages with basic form elements.
* Applied styling using the provided color palette.

**4. Styling:**

* Created a global CSS file with color variables and basic styles.
* Styled components using the color palette for a consistent look.

**Next Steps:**

1. **Authentication Flow:**
   * Implement authentication logic on the frontend.
   * Connect the frontend authentication with the backend.
2. **Direct Messaging Module:**
   * Design and implement the direct messaging module.
   * Connect frontend messaging with backend APIs.
3. **Feed Module (Twitter-like):**
   * Plan and design a feed module.
   * Implement basic feed functionalities.
4. **Documentation:**
   * Maintain documentation for code and project structure.
   * Update documentation regularly.

**Challenges Faced:**

* None so far.

**Day-03: Handling User Authentication in a Web Application**

**Introduction**

In today's session, we will focus on implementing user authentication in a web application using React on the frontend and Node.js with Express on the backend. Authentication is a crucial aspect of many applications to ensure that only authorized users can access certain features and data.

**Topics Covered**

1. **Backend Setup**
   * Initialize a Node.js project with Express.
   * Set up routes for user registration and login.
   * Use bcrypt to hash and secure user passwords.
   * Store user data in a database using Sequelize.
2. **Frontend Integration**
   * Create React components for user registration and login forms.
   * Use Axios to send HTTP requests from the frontend to the backend.
   * Implement form validation on the client-side.
3. **Token-Based Authentication**
   * Generate JWT (JSON Web Tokens) for authenticated users.
   * Secure routes on the backend that require authentication.
   * Store and manage JWT on the client-side.
4. **User Profile and Logout**
   * Display user information on the frontend after successful login.
   * Implement a logout feature to terminate the user session.

**Assignment**

Build a simple registration and login system for your web application. Use the concepts covered in today's session to secure user data and implement a smooth authentication flow.

**Storing Image and retrieving**

**SIGNUP.JS**

import React, { useState } from "react";

import { Link } from "react-router-dom";

import axios from "./../UrlHelper";

import { toast } from "react-toastify";

*// Countries data in JSON format*

const countriesData = [

  { value: "IND", label: "India" },

  { value: "USA", label: "United States" },

  { value: "UK", label: "United Kingdom" },

  { value: "Germany", label: "Germany" },

  { value: "Japan", label: "Japan" },

];

const Signup = () => {

  const [formData, setFormData] = useState({

    fullName: "",

    phoneNumber: "",

    birthDate: "",

    password: "",

    email: "",

    country: "",

    city: "",

    profileImage: null,

  });

  const handleInputChange = (e) => {

    const { name, value } = e.target;

    setFormData((prevData) => ({ ...prevData, [name]: value }));

  };

  const handleImageChange = (e) => {

    const file = e.target.files[0];

    const reader = new FileReader();

    reader.onloadend = () => {

*// Convert the base64 data to a Uint8Array*

      const uint8Array = new Uint8Array(atob(reader.result.split(",")[1]).split("").map((char) => char.charCodeAt(0)));

*// Set the Uint8Array in the component's state*

      setFormData((prevData) => ({ ...prevData, profileImage: uint8Array }));

    };

    if (file) {

      reader.readAsDataURL(file);

    }

  };

  const handleFormSubmit = async (e) => {

    e.preventDefault();

    console.log(formData)

    try {

      const response = await axios.post("/auth/register", formData);

      toast.success(

        <>

          <i *className*="fa-solid fa-handshake"></i> {"Successfully Registered"}

        </>

      );

      console.log(response.data);

*// Clear form data after successful submission*

      setFormData({

        fullName: "",

        phoneNumber: "",

        birthDate: "",

        password: "",

        email: "",

        country: "",

        city: "",

      });

    } catch (error) {

*// Handle errors, you can console.log them for now*

      console.error("Registration Error:", error.message);

    }

  };

  return (

    <div *className*="login register">

      <div *className*="container">

        <div *className*="heading">

          <h1 *className*="title">

            <i *className*="fa-solid fa-user-plus"></i>&nbsp;Register to Mystelio

          </h1>

        </div>

        <form *className*="form" *onSubmit*={handleFormSubmit}>

          <div *className*="input-box">

            <label>

              <i *className*="fa-solid fa-signature"></i>&nbsp;Full Name

            </label>

            <input

*required*=""

*placeholder*="Enter full name"

*type*="text"

*name*="fullName"

*value*={formData.fullName}

*onChange*={handleInputChange}

            />

          </div>

          <div *className*="column">

            <div *className*="input-box">

              <label>

                <i *className*="fa-solid fa-phone"></i>&nbsp;Phone Number

              </label>

              <input

*required*=""

*placeholder*="Enter phone number"

*type*="telephone"

*name*="phoneNumber"

*value*={formData.phoneNumber}

*onChange*={handleInputChange}

              />

            </div>

            <div *className*="input-box">

              <label>

                <i *className*="fa-solid fa-envelope-open-text"></i>&nbsp;Enter

                Email

              </label>

              <input

*required*=""

*placeholder*="Enter Email address"

*type*="email"

*name*="email"

*value*={formData.email}

*onChange*={handleInputChange}

              />

            </div>

          </div>

          <div *className*="column">

            <div *className*="input-box">

              <label>

                <i *className*="fa-solid fa-cake-candles"></i>&nbsp;Birth Date

              </label>

              <input

*required*=""

*placeholder*="Enter birth date"

*type*="date"

*name*="birthDate"

*value*={formData.birthDate}

*onChange*={handleInputChange}

              />

            </div>

            <div *className*="input-box">

              <label>

                <i *className*="fa-solid fa-key"></i>&nbsp;Enter Password

              </label>

              <input

*required*=""

*placeholder*="Enter password"

*type*="password"

*name*="password"

*value*={formData.password}

*onChange*={handleInputChange}

              />

            </div>

          </div>

          <div *className*="input-box address">

            <label>

              <i *className*="fa-solid fa-folder-open"></i>&nbsp;Additional

              Details

            </label>

            <div *className*="column">

              <div *className*="select-box">

                <select

*name*="country"

*value*={formData.country}

*onChange*={handleInputChange}

                >

                  <option *value*="">Select</option>

                  {countriesData.map((country) => (

                    <option *key*={country.value} *value*={country.value}>

                      {country.label}

                    </option>

                  ))}

                </select>

              </div>

              <input

*required*=""

*placeholder*="Enter your city"

*type*="text"

*name*="city"

*value*={formData.city}

*onChange*={handleInputChange}

              />

            </div>

          </div>

          <div *className*="input-box">

            <label>

              <i *className*="fa-solid fa-file-image"></i>&nbsp;Profile Image

            </label>

            <input *type*="file" *accept*="image/\*" *onChange*={handleImageChange} />

          </div>

          <button *className*="login-button" *type*="submit">

            Submit

          </button>

          <span *className*="agreement">

            <Link *to*="/login">Already a user? Login Here!</Link>

          </span>

        </form>

      </div>

    </div>

  );

};

export default Signup;

**BACKEND AUTHURLS.JS:**

const bcrypt = require("bcrypt");

const Joi = require("joi");

const jwt = require("jsonwebtoken");

const User = require("../models/userModel");

const multer = require('multer');

const express = require("express");

const router = express.Router();

require('dotenv').config();

*// Joi is a powerful validation library for JavaScript and Node.js. It's commonly used for:*

*// Input Validation: Joi helps you validate and sanitize user input to ensure that it meets the expected criteria. This is crucial for preventing security vulnerabilities such as SQL injection or other forms of injection attacks.*

*// Schema Validation: Joi allows you to define a schema that specifies the expected shape and types of data. This helps ensure that your data adheres to a predefined structure.*

*// Error Handling: Joi provides detailed error messages when validation fails, making it easier to identify and address issues. These error messages can be sent back to clients for informative feedback.*

const userSchema = Joi.object({

  fullName: Joi.string().required(),

  phoneNumber: Joi.string().required(),

  birthDate: Joi.date().required(),

  password: Joi.string().required(),

  email: Joi.string().email().required(),

  country: Joi.string(),

  city: Joi.string(),

  profileImage: Joi.allow(),

});

router.post('/register', async (req, res) => {

  try {

    const { error } = userSchema.validate(req.body);

    if (error) {

      console.log(error)

      return res.status(400).json({ error: error.details[0].message });

    }

    const hashedPassword = await bcrypt.hash(req.body.password, 10);

*// Create a new user in the database*

    const imageBytes = Object.values(req.body.profileImage);

    const imageBuffer = Buffer.from(imageBytes);

    const newUser = await User.create({

      fullName : req.body.fullName,

      phoneNumber: req.body.phoneNumber,

      birthDate: req.body.birthDate,

      password: hashedPassword,

      email: req.body.email,

      country: req.body.country,

      city: req.body.city,

      profileImage: imageBuffer,

    });

    res.status(201).json({ message: 'Success' });

  } catch (error) {

    console.error('Error registering user:', error);

    res.status(500).send('Error registering user');

  }

});

*// Validation schema for login*

const loginSchema = Joi.object({

  email: Joi.string().email().required(),

  password: Joi.string().required(),

});

router.post("/login", async (req, res) => {

  try {

*// Validate input*

    const { error } = loginSchema.validate(req.body);

    if (error) {

      return res.status(400).json({ error: error.details[0].message });

    }

    const { email, password } = req.body;

*// Find user by email*

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

    if (!user) {

      return res.status(404).json({ error: "User not found" });

    }

*// Compare passwords*

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

    if (!passwordMatch) {

      return res.status(401).json({ error: "Invalid password" });

    }

*// Generate JWT token*

    const token = jwt.sign(

      { userId: user.id, email: user.email },

      process.env.JWT\_SECRET,

      { expiresIn: "1h" } *// Token expires in 1 hour*

    );

*// Return JWT token and user data (excluding password)*

    res.status(200).json({

      token,

      user: {

        id: user.id,

        fullName: user.fullName,

        phoneNumber: user.phoneNumber,

        birthDate: user.birthDate,

        email: user.email,

        country: user.country,

        city: user.city,

        creted\_at: user.createdAt,

        updated\_at: user.updatedAt,

        profileImage: user.profileImage

      },

    });

  } catch (error) {

    console.error("Error during login:", error);

    res.status(500).send("Error during login");

  }

});

module.exports = router;

**SIGNIN Retrieving img:**

import React, { useState, useEffect } from "react";

import { Link } from "react-router-dom";

import axios from "./../UrlHelper";

import { toast } from "react-toastify";

const Login = () => {

  const [user, setUser] = useState(null);

  const [formData, setFormData] = useState({

    email: "",

    password: "",

  });

  const handleInputChange = (e) => {

    const { name, value } = e.target;

    setFormData((prevData) => ({ ...prevData, [name]: value }));

  };

  const handleFormSubmit = async (e) => {

    e.preventDefault();

    try {

      const response = await axios.post("/auth/login", formData);

*// Assume the server sends a JWT token in the response*

      const token = response.data.token;

      const userData = response.data.user;

      setUser(userData)

      handleImageFetch(userData);

*// Handle the token as needed (e.g., store it in local storage or a state variable)*

      toast.success("Login Successful");

      console.log("Login Successful:", response.data);

*// Clear form data after successful submission*

      setFormData({

        email: "",

        password: "",

      });

    } catch (error) {

*// Handle errors, you can console.log them for now*

      console.error("Login Error:", error.message);

      toast.error("Login Failed");

    }

  };

  const handleImageFetch = (userData) => {

    if (userData && userData.profileImage && userData.profileImage.data) {

      const bufferToBase64 = (buffer) => {

        const binary = buffer.reduce(

          (acc, byte) => acc + String.fromCharCode(byte),

          ""

        );

        return btoa(binary);

      };

      const base64Image = bufferToBase64(userData.profileImage.data);

      setUser((prevUser) => ({ ...prevUser, profileImage: base64Image }));

    }

  };

  return (

    <div *className*="login">

      <div *className*="container">

        <div *className*="heading">

          <h1 *className*="title">

            <i *className*="fa-solid fa-right-to-bracket"></i>&nbsp;LogIn to

            Mystelio

          </h1>

        </div>

        <form *className*="form" *onSubmit*={handleFormSubmit}>

          <div *className*="input-box">

            <label>

              <i *className*="fa-solid fa-envelope-open-text"></i>&nbsp;Enter Your

              email

            </label>

            <input

*required*=""

*placeholder*="example@mystelio.com"

*type*="email"

*name*="email"

*value*={formData.email}

*onChange*={handleInputChange}

            />

          </div>

          <div *className*="input-box">

            <label>

              <i *className*="fa-solid fa-unlock-keyhole"></i>&nbsp;Enter Password

            </label>

            <input

*required*=""

*placeholder*="\*\*\*\*\*\*\*\*\*\*"

*type*="password"

*name*="password"

*value*={formData.password}

*onChange*={handleInputChange}

            />

          </div>

          <span *className*="forgot-password">

            <Link *to*="/">Forgot Password ?</Link>

          </span>

          <input *value*="Sign In" *type*="submit" *className*="login-button" />

        </form>

        <span *className*="agreement">

          <Link *to*="/register">New to Mystelio? Signup Here!</Link>

        </span>

      </div>

      {user && user.profileImage && (

        <img *src*={`data:image/png;base64,${user.profileImage}`} *alt*="Profile" />

      )}

      {user && user.fullName}

    </div>

  );

};

export default Login;

**Day-04 Summary:**

1. **File Upload and Image Handling:**
   * Updated the backend to store image URLs instead of the actual image in the database.
   * Implemented image upload functionality for user profiles.
   * Saved images in the server's **uploads/profilepics/** directory.
2. **Frontend Integration:**
   * Modified the frontend registration form to handle file uploads and send the image URL to the backend.
   * Updated the login component to display user information, including the profile image fetched from the URL.
3. **User Authentication:**
   * Enhanced user registration route to store the image URL in the database.
   * Implemented user login functionality with JWT token generation.
4. **Error Handling:**
   * Implemented error handling for various scenarios during registration and login processes.
5. **Bug Fixes:**
   * Resolved issues related to file uploads, undefined file requests, and image display.
6. **Future Considerations:**
   * Discussed considerations for organizing uploads into different folders for various features like user profiles and posts.

**Updated backend:**

**userModel:**

const { DataTypes } = require("sequelize");

const db = require("../config/database");

const User = db.define("User", {

  fullName: {

    type: DataTypes.STRING,

    allowNull: false,

  },

  phoneNumber: {

    type: DataTypes.STRING,

    allowNull: false,

  },

  birthDate: {

    type: DataTypes.DATE,

    allowNull: false,

  },

  password: {

    type: DataTypes.STRING,

    allowNull: false,

  },

  email: {

    type: DataTypes.STRING,

    allowNull: false,

    unique: true,

    validate: {

      isEmail: true,

    },

  },

  country: {

    type: DataTypes.STRING,

  },

  city: {

    type: DataTypes.STRING,

  },

  profileImagePath: {

    type: DataTypes.STRING, *// Store the path to the image file*

  },

});

module.exports = User;

**AuthUrls.js in backend:**

const bcrypt = require("bcrypt");

const Joi = require("joi");

const jwt = require("jsonwebtoken");

const User = require("../models/userModel");

const multer = require("multer");

const express = require("express");

const router = express.Router();

const path = require("path");

require("dotenv").config();

*// Joi is a powerful validation library for JavaScript and Node.js. It's commonly used for:*

*// Input Validation: Joi helps you validate and sanitize user input to ensure that it meets the expected criteria. This is crucial for preventing security vulnerabilities such as SQL injection or other forms of injection attacks.*

*// Schema Validation: Joi allows you to define a schema that specifies the expected shape and types of data. This helps ensure that your data adheres to a predefined structure.*

*// Error Handling: Joi provides detailed error messages when validation fails, making it easier to identify and address issues. These error messages can be sent back to clients for informative feedback.*

const userSchema = Joi.object({

  fullName: Joi.string().required(),

  phoneNumber: Joi.string().required(),

  birthDate: Joi.date().required(),

  password: Joi.string().required(),

  email: Joi.string().email().required(),

  country: Joi.string(),

  city: Joi.string(),

  profileImage: Joi.allow(),

});

*// Set up storage for multer*

const profilePicsStorage  = multer.diskStorage({

  destination: (req, file, cb) => {

    cb(null, "uploads/profilepics/"); *// Upload files to the 'uploads' folder*

  },

  filename: (req, file, cb) => {

    const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() \* 1e9);

    const ext = path.extname(file.originalname);

    cb(null, file.fieldname + "-" + uniqueSuffix + ext);

  },

});

const profilePicsUpload  = multer({ storage: profilePicsStorage });

router.post("/register", profilePicsUpload.single("profileImage"), async (req, res) => {

  try {

    const { error } = userSchema.validate(req.body);

    if (error) {

      return res.status(400).json({ error: error.details[0].message });

    }

    const hashedPassword = await bcrypt.hash(req.body.password, 10);

*// Create a new user in the database*

    const newUser = await User.create({

      fullName: req.body.fullName,

      phoneNumber: req.body.phoneNumber,

      birthDate: req.body.birthDate,

      password: hashedPassword,

      email: req.body.email,

      country: req.body.country,

      city: req.body.city,

      profileImagePath: req.file ? req.file.path : null, *// Save the image path*

    });

    res.status(201).json({ message: "Success" });

  } catch (error) {

    console.error("Error registering user:", error);

    res.status(500).send("Error registering user");

  }

});

*// Validation schema for login*

const loginSchema = Joi.object({

  email: Joi.string().email().required(),

  password: Joi.string().required(),

});

router.post("/login", async (req, res) => {

  try {

*// Validate input*

    const { error } = loginSchema.validate(req.body);

    if (error) {

      return res.status(400).json({ error: error.details[0].message });

    }

    const { email, password } = req.body;

*// Find user by email*

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

    if (!user) {

      return res.status(404).json({ error: "User not found" });

    }

*// Compare passwords*

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

    if (!passwordMatch) {

      return res.status(401).json({ error: "Invalid password" });

    }

*// Generate JWT token*

    const token = jwt.sign(

      { userId: user.id, email: user.email },

      process.env.JWT\_SECRET,

      { expiresIn: "1h" } *// Token expires in 1 hour*

    );

*// Return JWT token and user data (excluding password)*

    res.status(200).json({

      token,

      user: {

        id: user.id,

        fullName: user.fullName,

        phoneNumber: user.phoneNumber,

        birthDate: user.birthDate,

        email: user.email,

        country: user.country,

        city: user.city,

        creted\_at: user.createdAt,

        updated\_at: user.updatedAt,

        profileImage: user.profileImagePath,

      },

    });

  } catch (error) {

    console.error("Error during login:", error);

    res.status(500).send("Error during login");

  }

});

module.exports = router;

**To make uploads folder as static and can be accessible by anyone:**

In server.js:

*// Serve static files from the "uploads" directory*

app.use("/uploads", express.static(path.join(\_\_dirname, "uploads")));

**Frontend to add fetch img:**

  const handleImageFetch = (userData) => {

    if (userData && userData.profileImage) {

*// Assuming userData.profileImage contains the file path*

      const imageUrl = `http://localhost:5000/${userData.profileImage.replace(

        "\\",

        "/"

      )}`;

      console.log(imageUrl);

      setUser((prevUser) => ({ ...prevUser, profileImage: imageUrl }));

    }

  };