**Phase 1: Documentation & Planning**

**1. Project Setup**

* **Technology Stack:** MEAN (MongoDB, Express, Angular, Node.js) along with Socket.io and Peer.js.
* **Initial Setup:**
  + Create a new GitHub repository for the project.
  + Set up a basic Angular frontend and Node.js backend.

**2. Git Repository Organization**

* **Branching Strategy:**
  + main branch: Stable, production-ready code.
  + development branch: Active development.
  + Feature branches (e.g., feature/login, feature/user-management): Specific features.
* **Commit Frequency:**
  + Frequent commits with descriptive messages.
  + Push to the repository regularly to track progress.
* **Server and Frontend Separation:**
  + Separate directories for server-side (Node.js) and client-side (Angular) code.

**3. Data Structures**

* **User Model:**
  + Attributes: username, email, id, roles[], groups[]
  + Stored in local storage (Phase 1).
* **Group Model:**
  + Attributes: groupId, groupName, adminIds[], userIds[]
* **Channel Model:**
  + Attributes: channelId, channelName, groupId, userIds[]

**4. Angular Architecture**

* **Components:**
  + LoginComponent: Handles user authentication.
  + DashboardComponent: Displays user-specific data after login.
  + GroupComponent: Manages groups and channels.
  + AdminComponent: UI for super admin and group admin functionalities.
* **Services:**
  + AuthService: Handles authentication logic.
  + UserService: Manages user data.
  + GroupService: Manages group and channel data.
* **Models:**
  + User: Represents the user entity.
  + Group: Represents the group entity.
  + Channel: Represents the channel entity.
* **Routes:**
  + "/login": Displays login page.
  + "/dashboard": Displays user dashboard.
  + "/admin": Admin functionalities.
  + "/group/:id": Group and channel management.

**5. Node Server Architecture**

* **Modules:**
  + auth.js: Handles user authentication routes.
  + user.js: Manages user-related routes.
  + group.js: Manages group and channel routes.
* **Functions:**
  + login(): Validates user credentials.
  + createUser(): Registers a new user.
  + createGroup(): Creates a new group.
  + createChannel(): Creates a new channel.
* **Files:**
  + server.js: Entry point for the Node.js server.
  + routes/: Contains all route handlers.
  + models/: Defines data models (users, groups, channels).
* **Global Variables:**
  + const users = [];: In-memory storage of users (Phase 1).
  + const groups = [];: In-memory storage of groups and channels.

**6. Server-Side Routes**

* POST /login: Authenticates a user.
* POST /user: Registers a new user.
* POST /group: Creates a new group.
* POST /channel: Creates a new channel.
* **Parameters:** User credentials, group details, channel details.
* **Return Values:** Success/failure messages, user data, group data.
* **Purpose:** Facilitates user management, group and channel creation.

**7. Client-Server Interaction**

* **User Authentication:**
  + Upon login, Angular sends user credentials to Node.js server.
  + Server validates and returns a token if successful.
  + Angular stores the token and displays the appropriate UI based on user role.
* **Group and Channel Management:**
  + Angular sends group/channel creation requests to the server.
  + Server updates in-memory storage and returns updated data.
  + Angular updates the UI to reflect changes in real-time.

**8. User Interface Implementation**

* **Login Page:**
  + Form to enter username and password.
  + Simple validation and submission to backend.
* **Dashboard:**
  + Displays a list of groups the user is part of.
  + Allows navigation to specific groups and channels.
* **Admin Page:**
  + Only accessible to super admins and group admins.
  + UI components to manage users, groups, and channels.

**9. Local Storage Usage**

* **Temporary Data Storage:**
  + Use local storage to hold user, group, and channel data.
  + Structure data in JSON format.
* **Transition to MongoDB in Phase 2.**

**10. GitHub Submission**

* **GitHub Repository:**
  + Ensure all code and documentation are committed and pushed.
  + Share repository with the tutor.
* **README.md:**
  + Detailed documentation of the project.
  + Use markdown syntax.
* **Canvas Submission:**
  + Submit a copy of the README.md as a Word document.

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### Phase 1 Overview

#### **1. Project Planning**

* **Objective**: Create a basic chat system that includes user creation, login functionality, and assignment to groups and channels. The system will include three roles: Super Admin, Group Admin, and User. The frontend will be implemented using Angular, and the backend will be implemented using Node.js and Express. Data will be stored in the browser's local storage for now, and full functionality will be implemented in Phase 2.

#### **2. Git Repository Setup**

* **Repository Structure**:
  + **Main Branches**:
    - main: Production-ready code.
    - development: Active development branch.
  + **Feature Branches**:
    - feature/login: Implementation of user login.
    - feature/group-management: Management of groups and channels.
  + **Commit Practices**:
    - Frequent commits with clear messages.
    - Use of pull requests to merge feature branches into development.

#### **3. Data Structures**

* **User Model**:
  + Attributes: username, email, id, roles[], groups[]
  + Example:

json

Copy code

{

"username": "john\_doe",

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

"id": 1,

"roles": ["User"],

"groups": [1, 2]

}

* **Group Model**:
  + Attributes: groupId, groupName, adminIds[], userIds[]
  + Example:

json

Copy code

{

"groupId": 1,

"groupName": "Project Team",

"adminIds": [1],

"userIds": [1, 2, 3]

}

* **Channel Model**:
  + Attributes: channelId, channelName, groupId, userIds[]
  + Example:

json

Copy code

{

"channelId": 1,

"channelName": "General",

"groupId": 1,

"userIds": [1, 2]

}

#### **4. Angular Architecture**

* **Components**:
  + **LoginComponent**: Handles user login.
  + **DashboardComponent**: Displays groups and channels for logged-in users.
  + **GroupManagementComponent**: Allows admins to create and manage groups and channels.
* **Services**:
  + **AuthService**: Manages authentication and session management.
  + **UserService**: Manages user data retrieval and updates.
  + **GroupService**: Manages group and channel data retrieval and updates.
* **Routing**:
  + /login: Login page.
  + /dashboard: Dashboard showing groups and channels.
  + /group-management: Page for group and channel management (only for admins).

#### **5. Node.js Server Architecture**

* **Modules**:
  + **auth.js**: Handles authentication routes.
  + **user.js**: Manages user-related routes.
  + **group.js**: Manages group and channel routes.
* **Routes**:
  + POST /login: Authenticates a user.
  + POST /user: Registers a new user.
  + POST /group: Creates a new group.
  + POST /channel: Creates a new channel.
* **Global Variables**:
  + const users = [];: In-memory storage of users (to be replaced by MongoDB in Phase 2).
  + const groups = [];: In-memory storage of groups and channels.

#### **6. Client-Server Interaction**

* **User Login**:
  + User submits credentials via Angular form.
  + Angular sends credentials to Node.js server for validation.
  + On success, server responds with user details and roles.
  + Angular updates the view based on the user’s role.
* **Group and Channel Management**:
  + Admins can create groups and channels via Angular forms.
  + Angular sends data to Node.js server.
  + Server updates in-memory data structures and responds with the updated list.
  + Angular updates the UI.

### ****Step-by-Step Implementation****

#### **Step 1: Setup Angular Project**

1. Create a new Angular project:

bash

Copy code

npx @angular/cli new chat-system-frontend

cd chat-system-frontend

1. Generate required components and services:

bash

Copy code

ng generate component login

ng generate component dashboard

ng generate component group-management

ng generate service auth

ng generate service user

ng generate service group

1. Set up routing in app-routing.module.ts:

typescript

Copy code

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { LoginComponent } from './login/login.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { GroupManagementComponent } from './group-management/group-management.component';

const routes: Routes = [

{ path: 'login', component: LoginComponent },

{ path: 'dashboard', component: DashboardComponent },

{ path: 'group-management', component: GroupManagementComponent },

{ path: '', redirectTo: '/login', pathMatch: 'full' }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

#### **Step 2: Setup Node.js Server**

1. Initialize the Node.js project:

bash

Copy code

mkdir chat-system-backend

cd chat-system-backend

npm init -y

npm install express cors body-parser

npm install --save-dev nodemon

1. Create server.js:

javascript

Copy code

const express = require('express');

const cors = require('cors');

const bodyParser = require('body-parser');

const app = express();

app.use(cors());

app.use(bodyParser.json());

const users = [];

const groups = [];

const channels = [];

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

const { username, password } = req.body;

const user = users.find(u => u.username === username && u.password === password);

if (user) {

res.status(200).send({ message: 'Login successful', user });

} else {

res.status(401).send({ message: 'Invalid credentials' });

}

});

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

const { username, email, password } = req.body;

const newUser = { username, email, password, id: users.length + 1, roles: ['User'], groups: [] };

users.push(newUser);

res.status(201).send({ message: 'User created', user: newUser });

});

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

const { groupName, adminId } = req.body;

const newGroup = { groupId: groups.length + 1, groupName, adminIds: [adminId], userIds: [adminId] };

groups.push(newGroup);

res.status(201).send({ message: 'Group created', group: newGroup });

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

1. Start the server:

bash

Copy code

npm start

#### **Step 3: Implement Angular Components**

* **LoginComponent**: Handle user login with a form that submits to the backend.
* **DashboardComponent**: Display available groups and channels for the logged-in user.
* **GroupManagementComponent**: Allow admins to create and manage groups and channels.

#### **Step 4: Store Data in Local Storage**

* In Phase 1, use local storage to simulate backend data persistence.
* Replace this with MongoDB in Phase 2.

### ****Step 5: Documentation****

1. **README.md**: Document your project’s structure, design decisions, and implementation details.
2. **GitHub Repository**: Ensure that your code is pushed regularly and that the repository is well-organized.

### ****Submission****

* Push your final code to GitHub.
* Submit the README.md as both a markdown file in GitHub and a Word document on Canvas.

This should provide a comprehensive start for Phase 1 of your assignment. Let me know if you need help wit**Phase 1 Overview**

**1. Project Planning**

* **Objective**: Create a basic chat system that includes user creation, login functionality, and assignment to groups and channels. The system will include three roles: Super Admin, Group Admin, and User. The frontend will be implemented using Angular, and the backend will be implemented using Node.js and Express. Data will be stored in the browser's local storage for now, and full functionality will be implemented in Phase 2.

**2. Git Repository Setup**

* **Repository Structure**:
  + **Main Branches**:
    - main: Production-ready code.
    - development: Active development branch.
  + **Feature Branches**:
    - feature/login: Implementation of user login.
    - feature/group-management: Management of groups and channels.
  + **Commit Practices**:
    - Frequent commits with clear messages.
    - Use of pull requests to merge feature branches into development.

**3. Data Structures**

* **User Model**:
  + Attributes: username, email, id, roles[], groups[]
  + Example:

json

Copy code

{

"username": "john\_doe",

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

"id": 1,

"roles": ["User"],

"groups": [1, 2]

}

* **Group Model**:
  + Attributes: groupId, groupName, adminIds[], userIds[]
  + Example:

json

Copy code

{

"groupId": 1,

"groupName": "Project Team",

"adminIds": [1],

"userIds": [1, 2, 3]

}

* **Channel Model**:
  + Attributes: channelId, channelName, groupId, userIds[]
  + Example:

json

Copy code

{

"channelId": 1,

"channelName": "General",

"groupId": 1,

"userIds": [1, 2]

}

**4. Angular Architecture**

* **Components**:
  + **LoginComponent**: Handles user login.
  + **DashboardComponent**: Displays groups and channels for logged-in users.
  + **GroupManagementComponent**: Allows admins to create and manage groups and channels.
* **Services**:
  + **AuthService**: Manages authentication and session management.
  + **UserService**: Manages user data retrieval and updates.
  + **GroupService**: Manages group and channel data retrieval and updates.
* **Routing**:
  + /login: Login page.
  + /dashboard: Dashboard showing groups and channels.
  + /group-management: Page for group and channel management (only for admins).

**5. Node.js Server Architecture**

* **Modules**:
  + **auth.js**: Handles authentication routes.
  + **user.js**: Manages user-related routes.
  + **group.js**: Manages group and channel routes.
* **Routes**:
  + POST /login: Authenticates a user.
  + POST /user: Registers a new user.
  + POST /group: Creates a new group.
  + POST /channel: Creates a new channel.
* **Global Variables**:
  + const users = [];: In-memory storage of users (to be replaced by MongoDB in Phase 2).
  + const groups = [];: In-memory storage of groups and channels.

**6. Client-Server Interaction**

* **User Login**:
  + User submits credentials via Angular form.
  + Angular sends credentials to Node.js server for validation.
  + On success, server responds with user details and roles.
  + Angular updates the view based on the user’s role.
* **Group and Channel Management**:
  + Admins can create groups and channels via Angular forms.
  + Angular sends data to Node.js server.
  + Server updates in-memory data structures and responds with the updated list.
  + Angular updates the UI.

**Step-by-Step Implementation**

**Step 1: Setup Angular Project**

1. Create a new Angular project:

bash

Copy code

npx @angular/cli new chat-system-frontend

cd chat-system-frontend

1. Generate required components and services:

bash

Copy code

ng generate component login

ng generate component dashboard

ng generate component group-management

ng generate service auth

ng generate service user

ng generate service group

1. Set up routing in app-routing.module.ts:

typescript

Copy code

import { NgModule } from '@angular/core';

import { RouterModule, Routes } from '@angular/router';

import { LoginComponent } from './login/login.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { GroupManagementComponent } from './group-management/group-management.component';

const routes: Routes = [

{ path: 'login', component: LoginComponent },

{ path: 'dashboard', component: DashboardComponent },

{ path: 'group-management', component: GroupManagementComponent },

{ path: '', redirectTo: '/login', pathMatch: 'full' }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

**Step 2: Setup Node.js Server**

1. Initialize the Node.js project:

bash

Copy code

mkdir chat-system-backend

cd chat-system-backend

npm init -y

npm install express cors body-parser

npm install --save-dev nodemon

1. Create server.js:

javascript

Copy code

const express = require('express');

const cors = require('cors');

const bodyParser = require('body-parser');

const app = express();

app.use(cors());

app.use(bodyParser.json());

const users = [];

const groups = [];

const channels = [];

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

const { username, password } = req.body;

const user = users.find(u => u.username === username && u.password === password);

if (user) {

res.status(200).send({ message: 'Login successful', user });

} else {

res.status(401).send({ message: 'Invalid credentials' });

}

});

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

const { username, email, password } = req.body;

const newUser = { username, email, password, id: users.length + 1, roles: ['User'], groups: [] };

users.push(newUser);

res.status(201).send({ message: 'User created', user: newUser });

});

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

const { groupName, adminId } = req.body;

const newGroup = { groupId: groups.length + 1, groupName, adminIds: [adminId], userIds: [adminId] };

groups.push(newGroup);

res.status(201).send({ message: 'Group created', group: newGroup });

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

1. Start the server:

bash

Copy code

npm start

**Step 3: Implement Angular Components**

* **LoginComponent**: Handle user login with a form that submits to the backend.
* **DashboardComponent**: Display available groups and channels for the logged-in user.
* **GroupManagementComponent**: Allow admins to create and manage groups and channels.

**Step 4: Store Data in Local Storage**

* In Phase 1, use local storage to simulate backend data persistence.
* Replace this with MongoDB in Phase 2.

**Step 5: Documentation**

1. **README.md**: Document your project’s structure, design decisions, and implementation details.
2. **GitHub Repository**: Ensure that your code is pushed regularly and that the repository is well-organized.

**Submission**

* Push your final code to GitHub.
* Submit the README.md as both a markdown file in GitHub and a Word document on Canvas.

This should provide a comprehensive start for Phase 1 of your assignment. Let me know if you need help with any specific code or concepts!

h any specific code or concepts!