User Registration and Login

1. **Homepage**: Displays a welcome message.
2. **Login and Registration Forms**: Simple forms for user authentication.
3. **Dashboard**: Displays a personalized welcome message with the logged-in user's name.

Steps to Create the App:

1. Setup the Angular Project

```
ng new SimpleApp
cd SimpleApp
ng serve
```

2. Install Angular Routing

Make sure routing is enabled when creating the project or add it manually:

```
ng generate module app-routing --flat --module=app
```

3. Generate Components

Create the necessary components for the homepage, login, registration, and dashboard:

```
ng generate component home
ng generate component login
ng generate component register
ng generate component dashboard
```

4. Set Up Routing

Update `app-routing.module.ts` to define routes for the components:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
```

```
import { LoginComponent } from './login/login.component';
import { RegisterComponent } from './register/register.component';
import { DashboardComponent } from './dashboard/dashboard.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'dashboard', component: DashboardComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

5. Create a Simple Service for User Data

Generate a service to manage user data:

```
ng generate service user
```

Update `user.service.ts`:

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  private userName: string = '';

  setUserName(name: string) {
    this.userName = name;
  }
```

```
  getUserName(): string {
    return this.userName;
  }
}
```

6. Implement Login and Registration

Add basic forms in `login.component.html` and `register.component.html`:

`login.component.html`

```
<h2>Login</h2>
<form (submit)="login()">
  <label for="name">Name:</label>
  <input type="text" [(ngModel)]="name" name="name" id="name" required
/>
  <button type="submit">Login</button>
</form>
```

`login.component.ts`

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { UserService } from '../user.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
})
export class LoginComponent {
  name: string = '';

  constructor(private userService: UserService, private router:
Router) {}

  login() {
    if (this.name.trim()) {
```

```
      this.userService.setUserName(this.name);
      this.router.navigate(['/dashboard']);
    } else {
      alert('Please enter your name');
    }
  }
}
```

`register.component.html` Similar to `login.component.html`. Change `login()` to `register()` and implement basic functionality.

7. Implement the Dashboard

Use the `UserService` to display the user's name:

`dashboard.component.html`

```
<h2>Welcome to Your Dashboard</h2>
<p>Hello, {{ userName }}!</p>
<a routerLink="/">Go to Homepage</a>
```

`dashboard.component.ts`

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../user.service';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
})
export class DashboardComponent implements OnInit {
  userName: string = '';

  constructor(private userService: UserService) {}

  ngOnInit() {
    this.userName = this.userService.getUserName();
```

```
  }
}
```

8. Set Up the Homepage

`home.component.html`

```
<h1>Welcome to SimpleApp</h1>
<a routerLink="/login">Login</a> | <a
routerLink="/register">Register</a>
```

9. Run the App

Start the development server:

```
ng serve
```

Full Implementation Steps with NgRx with state management

1. Install NgRx

Install the necessary NgRx packages for state management:

```
ng add @ngrx/store
ng add @ngrx/effects
```

2. Create State for User Management

Generate a new NgRx feature for user state:

```
ng generate feature users/User --module=app
```

Inside the generated files (`user.actions.ts`, `user.reducer.ts`, etc.), define the state for storing user data.

`user.actions.ts`

```
import { createAction, props } from '@ngrx/store';

export const registerUser = createAction(
  '[User] Register User',
  props<{ user: any }>()
);

export const loginUser = createAction(
  '[User] Login User',
  props<{ email: string; password: string }>()
);

export const loginFailed = createAction('[User] Login Failed');
export const logoutUser = createAction('[User] Logout User');
```

user.reducer.ts

```
import { createReducer, on } from '@ngrx/store';
import { registerUser, loginUser, loginFailed, logoutUser } from
'./user.actions';

export interface UserState {
  users: any[];
  loggedInUser: any | null;
  loginError: string | null;
}

export const initialState: UserState = {
  users: [],
  loggedInUser: null,
  loginError: null,
};

export const userReducer = createReducer(
  initialState,
  on(registerUser, (state, { user }) => ({
    ...state,
    users: [...state.users, user],
  })),
  on(loginUser, (state, { email, password }) => {
    const foundUser = state.users.find(
      (user) => user.email === email && user.password === password
    );
    if (foundUser) {
      return { ...state, loggedInUser: foundUser, loginError: null };
    } else {
      return { ...state, loginError: 'Invalid email or password' };
    }
  }),
  on(loginFailed, (state) => ({
    ...state,
```

```
    loginError: 'Invalid email or password',
  })),
  on(logoutUser, (state) => ({
    ...state,
    loggedInUser: null,
  }))
);
```

Add the `userReducer` to the root store in `app.module.ts`:

```
import { StoreModule } from '@ngrx/store';
import { userReducer } from './users/user.reducer';

@NgModule({
  imports: [
    StoreModule.forRoot({ user: userReducer }),
    // other imports...
  ],
})
```

3. Update Registration Form

Add additional fields with validation in the registration form.

`register.component.html`

```
<h2>Register</h2>
<form (ngSubmit)="register()" #form="ngForm">
  <div>
    <label for="firstName">First Name:</label>
    <input type="text" id="firstName" [(ngModel)]="firstName"
name="firstName" required />
  </div>
  <div>
    <label for="lastName">Last Name:</label>
    <input type="text" id="lastName" [(ngModel)]="lastName"
name="lastName" required />
```

```
  </div>
  <div>
    <label for="dob">Date of Birth:</label>
    <input type="date" id="dob" [(ngModel)]="dob" name="dob" required
/>
  </div>
  <div>
    <label for="country">Country:</label>
    <input type="text" id="country" [(ngModel)]="country"
name="country" required />
  </div>
  <div>
    <label for="email">Email:</label>
    <input type="email" id="email" [(ngModel)]="email" name="email"
required />
  </div>
  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" [(ngModel)]="password"
name="password" required />
  </div>
  <button type="submit">Register</button>
</form>
```

register.component.ts

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { registerUser } from '../users/user.actions';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
})
export class RegisterComponent {
  firstName = '';
  lastName = '';
```

```
  dob = '';
  country = '';
  email = '';
  password = '';

  constructor(private store: Store) {}

  register() {
    const user = {
      firstName: this.firstName,
      lastName: this.lastName,
      dob: this.dob,
      country: this.country,
      email: this.email,
      password: this.password,
    };
    this.store.dispatch(registerUser({ user }));
    alert('Registration Successful!');
  }
}
```

4. Update Login Form

Add validation and error messages.

`login.component.html`

```
<h2>Login</h2>
<form (ngSubmit)="login()" #form="ngForm">
  <div>
    <label for="email">Email:</label>
    <input type="email" id="email" [(ngModel)]="email" name="email"
required />
  </div>
  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" [(ngModel)]="password"
name="password" required />
```

```
  </div>
  <button type="submit">Login</button>
</form>
<p *ngIf="loginError" style="color: red;">{{ loginError }}</p>
```

login.component.ts

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { loginUser } from '../users/user.actions';
import { Observable } from 'rxjs';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
})
export class LoginComponent {
  email = '';
  password = '';
  loginError$: Observable<string | null>;

  constructor(private store: Store<{ user: any }>) {
    this.loginError$ = this.store.select((state) =>
state.user.loginError);
  }

  login() {
    this.store.dispatch(loginUser({ email: this.email, password:
this.password }));
  }
}
```

5. Dashboard

Display logged-in user details in the dashboard.

dashboard.component.ts

```
import { Component } from '@angular/core';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
})
export class DashboardComponent {
  loggedInUser$: Observable<any>;

  constructor(private store: Store<{ user: any }>) {
    this.loggedInUser$ = this.store.select((state) =>
state.user.loggedInUser);
  }
}
```

dashboard.component.html

```
<h2>Welcome to Your Dashboard</h2>
<div *ngIf="loggedInUser$ | async as user">
  <p>Hello, {{ user.firstName }} {{ user.lastName }}!</p>
  <p>Email: {{ user.email }}</p>
  <p>Country: {{ user.country }}</p>
</div>
<a routerLink="/">Go to Homepage</a>
```