WhereToEat

Team Noobs

Ibrahim Alkuwaifi

Mohammad Khaja Moinuddin

Sai Kumar Gurram

Teja Peddi

Ashesh Kumar Pinisetti

Vishal Kandunuri

Mohammed Abdul Wahid

Gouthami Gongidi

Mallikarjun Narra

CSCE 5430.008

Requirements:

Phase 1: Core Requirements

- Authentication (100%)
  - User Sign-up, Login, and Password Reset
    - Implement a user sign-up, login, and password reset functionality using Firebase Authentication.
    - Utilize Firebase's SDK to handle the backend logic for authentication. Integrate this into the UI to provide a seamless experience for users.
  - Secure Access and Restrictions
    - Ensure that access to the app is secure and that only authenticated users can create and join hangouts.
    - Use Firebase rules to limit database reads/writes to authenticated users. Implement middleware to check authentication status before rendering sensitive components in the app.
- Hangout Creation (100%)
  - Create Hangouts
    - Allow users to create hangouts, specifying a name, location, and an optional description.
    - Implement a form capturing the hangout name, location, and optional description. Use geolocation APIs to suggest locations.
  - Hangout Options
    - Enable the hangout creator to choose whether users can add their own restaurants or if the system should generate random restaurants.
    - Add toggles or checkboxes in the hangout creation form to allow this selection. Implement logic to handle either scenario.
- Adding Restaurants (100%)
  - User-Added Restaurants
    - Enable users to add their own restaurants, capturing relevant information like name, address, cuisine, etc.
    - Create a form that captures the necessary details. Validate the information before saving it to the database.

Scope Update (Phase 2)

The scope for Phase 2 changed from Deliverable 2. We are using yelp fusion instead of Google Places since it is free. Below are the more detailed descriptions of the requirements specified for Phase 2.

Phase 2: Enhanced User Experience and Integration

1. Yelp Fusion Integration (100%):
    a. Integrate Yelp Fusion API to access restaurant data and enhance the restaurant suggestion feature.
    b. Implement autocomplete functionality for restaurant search and location input.
2. User Interface Development (100%):
    a. Utilize Bootstrap and Angular Material to create a unified and user-friendly interface.
    b. Develop responsive design for optimal viewing on various devices (desktop, tablet, mobile).
3. Hangout Management(100%):
    a. Implement features to edit, delete, and manage hangouts.
    b. Allow users to invite friends to join hangouts and track RSVPs.

Scope Update (Phase 3)

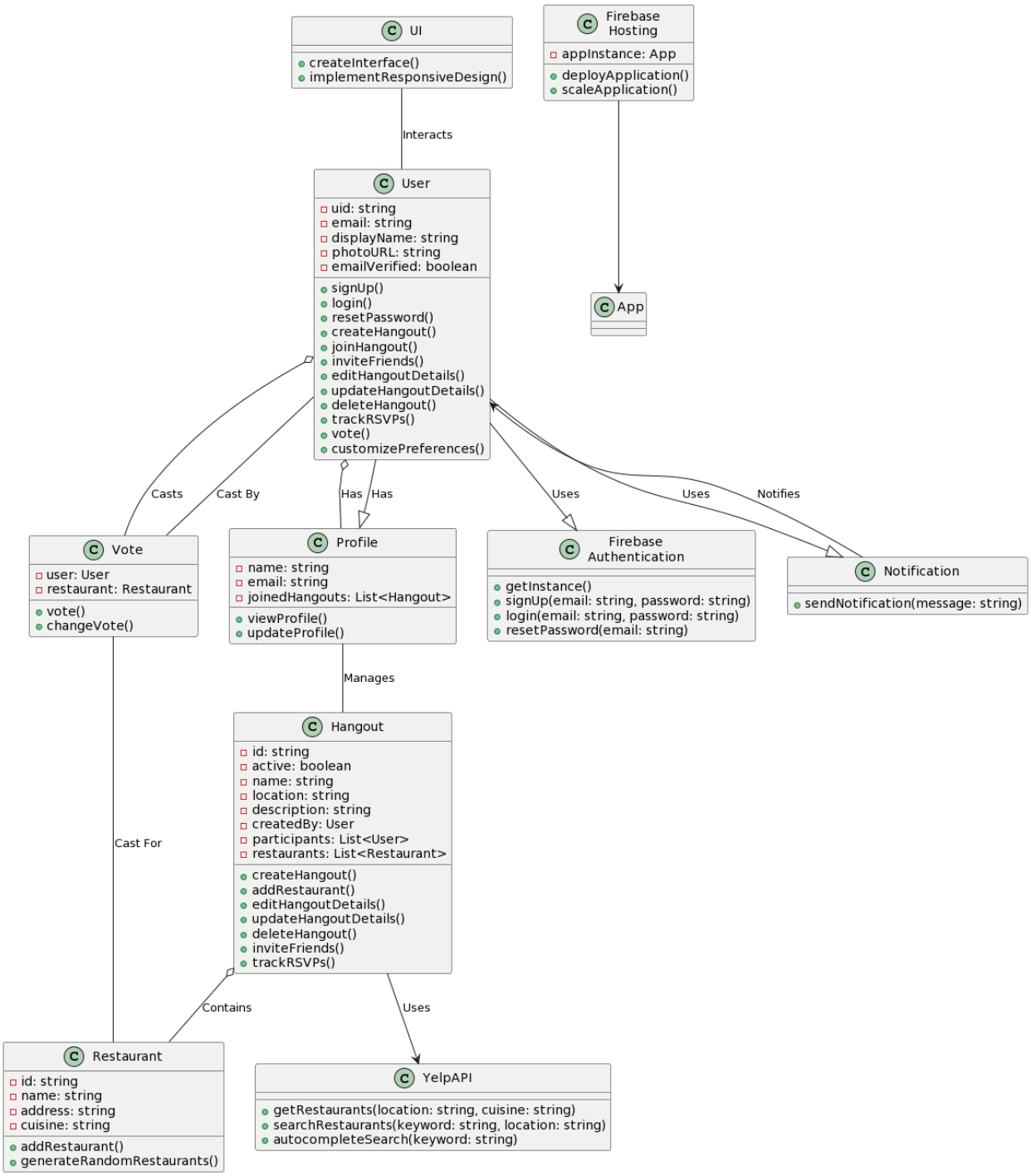
The scope for Phase 3 changed from Deliverable 3. We are using Vercel instead of Firebase to deploy our app.

Phase 3: Deployment

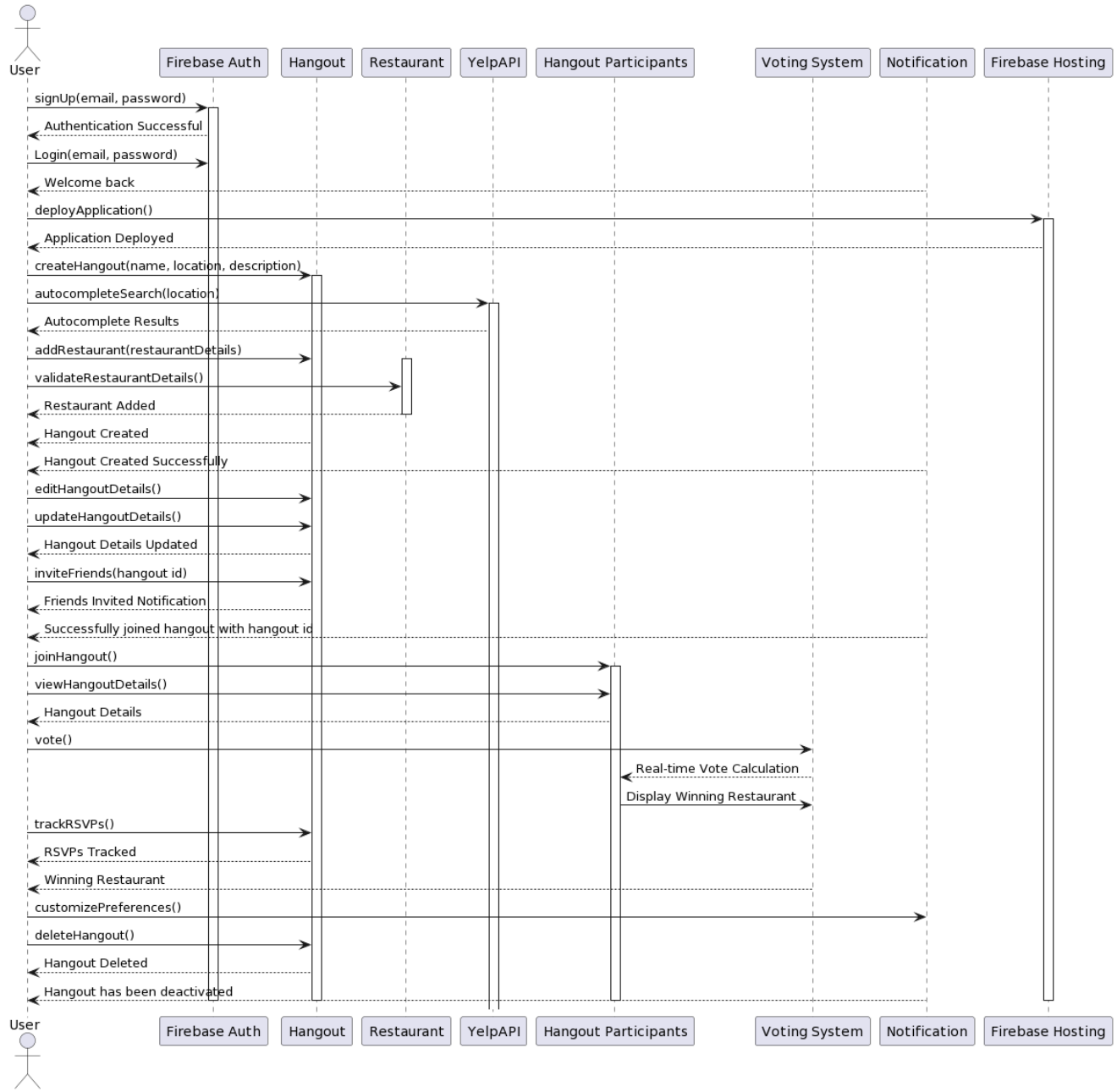
4. Deployment (100%):
    a. Deploy the application using Vercel hosting, making it accessible to users over the internet.
    b. Ensure the application is scalable and can handle increased user traffic.
5. Voting System (100%):
    a. Develop a voting system for users to vote on their preferred restaurants within a hangout.
    b. Implement a feature to display voting results and determine the winning restaurant.

UML Diagrams:

Class Diagram:

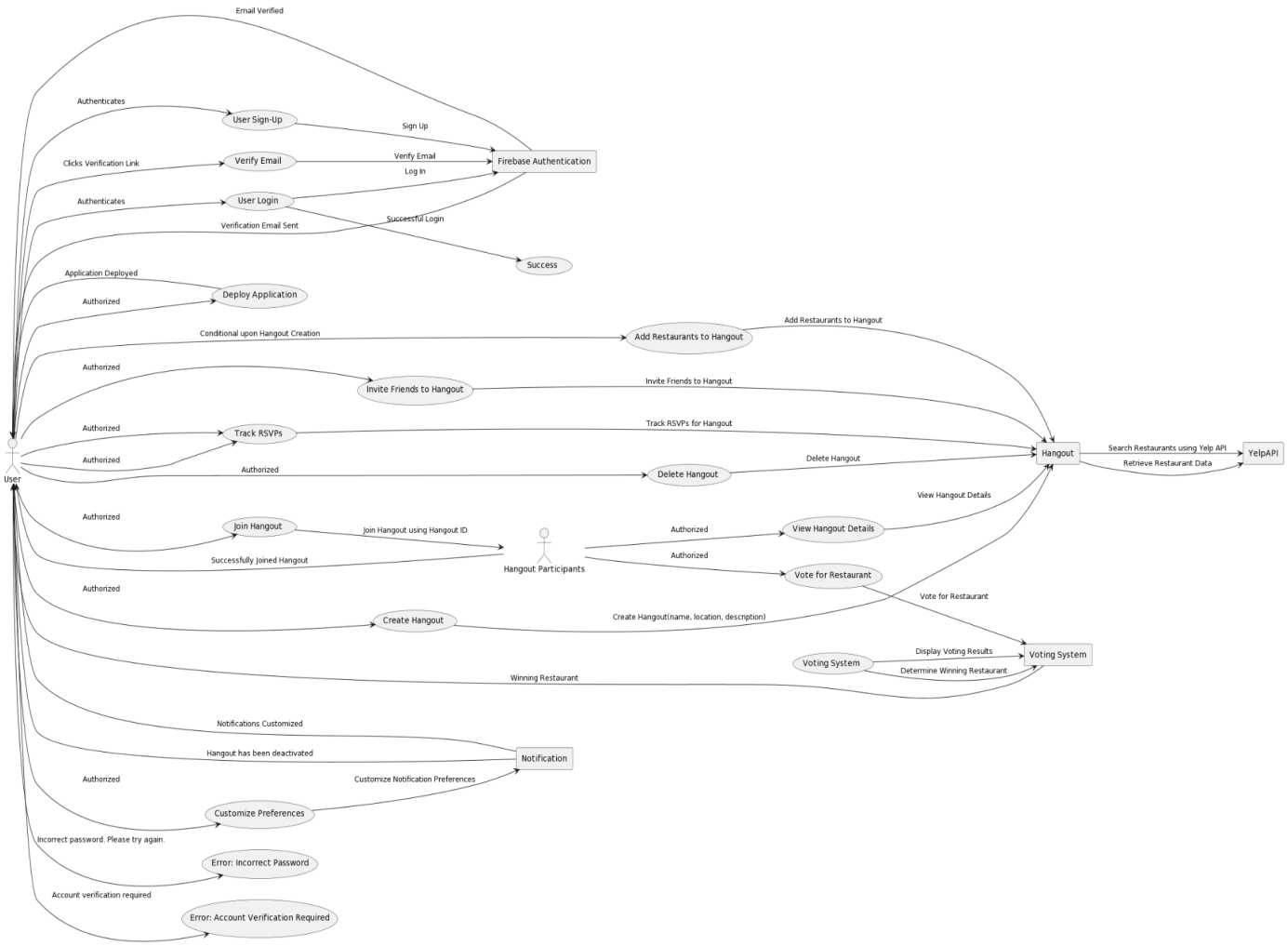


Sequence Diagram:



Use case diagram:

Added one normal case and one error case



Test Cases:

Unit tests:

Unit tests for Forgot Password page:

Code:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { FormGroup, ReactiveFormsModule } from '@angular/forms';
import { of } from 'rxjs';
import { AuthService } from 'src/app/auth/auth.service';
import { ForgotPasswordPageComponent } from './forgot-password-page.component';
fdescribe('ForgotPasswordPageComponent', () => {
let component: ForgotPasswordPageComponent;
let fixture: ComponentFixture<ForgotPasswordPageComponent>;
let authService: AuthService;
beforeEach(() => {
TestBed.configureTestingModule({
declarations: [ForgotPasswordPageComponent],
imports: [ReactiveFormsModule],
providers: [
{
provide: AuthService,
useValue: {
ForgotPassword: () => of(true), // Mock the AuthService's ForgotPassword
method
},
},],
});
fixture = TestBed.createComponent(ForgotPasswordPageComponent);
component = fixture.componentInstance;
authService = TestBed.inject(AuthService);
});
it('should create the component', () => {
expect(component).toBeTruthy();
```

```
});
it('should initialize the form with username control', () => {
fixture.detectChanges(); // Trigger ngOnInit
const usernameControl = component.passwordForm.get('username');
expect(usernameControl).toBeTruthy();
});
it('should reset password and set loading to false', () => {
fixture.detectChanges();
const username = 'test@example.com';
// Spy on AuthService's ForgotPassword method to check if it was called
const authServiceSpy = spyOn(authService, 'ForgotPassword').and.returnValue(Promise.resolve());
// Set the username value
component.passwordForm.get('username')?.setValue(username);
// Call the resetPassword method
component.resetPassword();
// Expect that AuthService's ForgotPassword was called with the username
expect(authServiceSpy).toHaveBeenCalledWith(username);
// Expect loading to be set to false after completing
expect(component.loading).toBeTruthy();
});
```

Description:

Three test cases have been written in this example:

The first test case determines whether the component was correctly constructed.

When ngOnInit is triggered, the second test case determines whether the form has an empty username field upon initialization.

Third test scenario determines whether the resetWhen a password is reset, the password function changes the loading flag to true.After the process is finished, return to false.

Unit tests for Hangout Page:

```
import { ComponentFixture, TestBed, async, fakeAsync, tick } from '@angular/core/testing';
import { ActivatedRoute, Router } from '@angular/router';
import { AuthService } from 'src/app/auth/auth.service';
import { NotificationService } from 'src/app/core/services/notification.service';
import { HangoutPageComponent } from './hangout-page.component';
```

```
import { HangoutService } from 'src/app/services/hangout.service';
import { of } from 'rxjs';
import { Hangout, HangoutWithUsers } from 'src/app/models/hangout.model';
import { Restaurant } from 'src/app/models/restaurant.model';
import { User } from 'src/app/models/user.model';

describe('HangoutPageComponent', () => {
  let component: HangoutPageComponent;
  let fixture: ComponentFixture<HangoutPageComponent>;
  let activatedRoute: jasmine.SpyObj<ActivatedRoute>;
  let hangoutService: jasmine.SpyObj<HangoutService>;
  let authService: jasmine.SpyObj<AuthService>;
  let notificationService: jasmine.SpyObj<NotificationService>;
  let router: jasmine.SpyObj<Router>;

  beforeEach(async(() => {
    const activatedRouteSpy = jasmine.createSpyObj('ActivatedRoute', ['params']);
    const hangoutServiceSpy = jasmine.createSpyObj('HangoutService', ['getHangout', 'castVote',
'deactivateHangout']);
    const authServiceSpy = jasmine.createSpyObj('AuthService', ['user$']);
    const notificationServiceSpy = jasmine.createSpyObj('NotificationService', ['showNotification']);
    const routerSpy = jasmine.createSpyObj('Router', ['navigate']);

    TestBed.configureTestingModule({
      declarations: [HangoutPageComponent],
      providers: [
        { provide: ActivatedRoute, useValue: activatedRouteSpy },
        { provide: HangoutService, useValue: hangoutServiceSpy },
        { provide: AuthService, useValue: authServiceSpy },
        { provide: NotificationService, useValue: notificationServiceSpy },
        { provide: Router, useValue: routerSpy },
      ],
    }).compileComponents();

    activatedRoute = TestBed.inject(ActivatedRoute) as jasmine.SpyObj<ActivatedRoute>;
    hangoutService = TestBed.inject(HangoutService) as jasmine.SpyObj<HangoutService>;
```

```
    authService = TestBed.inject(AuthService) as jasmine.SpyObj<AuthService>;
    notificationService = TestBed.inject(NotificationService) as jasmine.SpyObj<NotificationService>;
    router = TestBed.inject(Router) as jasmine.SpyObj<Router>;
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(HangoutPageComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should set loading to true and fetch hangout details on init', fakeAsync(() => {
    const testHangout: HangoutWithUsers = {test1, 75025, test1};
    activatedRoute.params.and.returnValue(of({ id: 'testHangoutId' }));
    hangoutService.getHangout.and.returnValue(of(testHangout));

    component.ngOnInit();
    tick(); // simulate the passage of time for async operations

    expect(component.loading).toBeFalse();
    expect(component.hangout).toEqual(testHangout);
    expect(component.participantsNames).toBe(testHangout.participants.map((user: User) =>
user.displayName).join(', '));
  }));

  it('should subscribe to authService.user$ and update user on init', fakeAsync(() => {
    const testUser: User = {TestID1};
    authService.user$.and.returnValue(of(testUser));

    component.ngOnInit();
    tick(); // simulate the passage of time for async operations
```

```
  expect(component.user).toEqual(testUser);
}));

it('should check if the current user is the creator', () => {
  const testUser: User = {TestID1} ;
  const testHangout: HangoutWithUsers = {test1, 75025, test1};
  authService.user$.and.returnValue(of(testUser));
  hangoutService.getHangout.and.returnValue(of(testHangout));

  component.ngOnInit();

  expect(component.isCreator()).toBe(testUser.uid === testHangout.createdBy.uid);
});

it('should cast vote when vote is called', () => {
  const testUser: User = {TestID1};
  const testHangout: HangoutWithUsers = {test1, 75025, test1};
  const testRestaurant: Restaurant = { testkitchen, Frisco texas 75609, Chinese };
  authService.user$.and.returnValue(of(testUser));
  hangoutService.getHangout.and.returnValue(of(testHangout));

  component.ngOnInit();
  component.vote(testRestaurant);

  expect(hangoutService.castVote).toHaveBeenCalledWith(testUser.uid, testHangout.id, testRestaurant.id);
});

it('should navigate to home page after successfully deactivating hangout', fakeAsync(() => {
  const testHangout: HangoutWithUsers = {test1, 75025, test1} ;
  hangoutService.getHangout.and.returnValue(of(testHangout));
  hangoutService.deactivateHangout.and.returnValue(Promise.resolve(true));

  component.ngOnInit();
  component.endHangout();
  tick(); // simulate the passage of time for async operations
```

```
    expect(notificationService.showNotification).toHaveBeenCalledWith('success', 'Hangout has been deactivated.');
    expect(router.navigate).toHaveBeenCalledWith(['home']);
  }));

  it('should show error notification if deactivating hangout fails', fakeAsync(() => {
    const testHangout: HangoutWithUsers = {test1, 75025, test1} ;
    hangoutService.getHangout.and.returnValue(of(testHangout));
    hangoutService.deactivateHangout.and.returnValue(Promise.resolve(false));

    component.ngOnInit();
    component.endHangout();
    tick(); // simulate the passage of time for async operations

    expect(notificationService.showNotification).toHaveBeenCalledWith('error', 'Failed to deactivate hangout.');
  }));
});
```

Unit test for Home Page:

Code:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { ReactiveFormsModule } from '@angular/forms';
import { of } from 'rxjs';
import { AuthService } from 'src/app/auth/auth.service';
import { HomePageComponent } from './home-page.component';
import { HangoutService } from 'src/app/services/hangout.service';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { Hangout } from 'src/app/models/hangout.model';
fdescribe('HomePageComponent', () => {
let component: HomePageComponent;
let fixture: ComponentFixture<HomePageComponent>;
let authService: AuthService;
let hangoutService: HangoutService;
beforeEach(() => {
TestBed.configureTestingModule({
declarations: [HomePageComponent],
```

```
imports: [ReactiveFormsModule],
providers: [
{
provide: AngularFireAuth,
useValue: {
authState: of({ uid: 'user-uid' }), // Mock AngularFireAuth's authState
},
},
{
provide: AuthService,
useValue: {
user$: of({ uid: 'user-uid' }), // Mock AuthService's user$
userData: { uid: 'user-uid' }, // Mock AuthService's userData
},
},
{
provide: HangoutService,
useValue: {
getActiveHangoutsForUser: (uid: string) => of([]), // Mock HangoutService's
getActiveHangoutsForUser
getPastHangoutsForUser: (uid: string) => of([]), // Mock HangoutService's getPastHangoutsForUser
createHangout: (hangout: Hangout) => of(hangout), // Mock HangoutService's createHangout
},
},
],
});
fixture = TestBed.createComponent(HomePageComponent);
component = fixture.componentInstance;
authService = TestBed.inject(AuthService);
hangoutService = TestBed.inject(HangoutService);
});
it('should create the component', () => {
expect(component).toBeTruthy();
});
it('should initialize the hangoutForm with the necessary controls', () => {
fixture.detectChanges(); // Trigger ngOnInit
```

```
const nameControl = component.hangoutForm.get('name');
const locationControl = component.hangoutForm.get('location');
const descriptionControl = component.hangoutForm.get('description');
expect(nameControl).toBeTruthy();
expect(locationControl).toBeTruthy();
expect(descriptionControl).toBeTruthy();
});
it('should create a new hangout', () => {
fixture.detectChanges(); // Trigger ngOnInit
component.hangoutForm.setValue({
name: 'Test Hangout',
location: 'Test Location',
description: 'Test Description',
});
const createHangoutSpy = spyOn(hangoutService, 'createHangout').and.returnValue(Promise.resolve());
component.createNewHangout();
expect(createHangoutSpy).toHaveBeenCalled();
expect(component.loading).toBeFalse();
});
});
```

Description:

The first test case determines whether the component was correctly constructed.
The accuracy of the form's initialization with empty values is confirmed by the second test case.
In the third test scenario, form values are set and it is verified whether the anticipated data is passed to the createHangout method call.

Unit test for Login page:

Code:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { ReactiveFormsModule } from '@angular/forms';
import { of } from 'rxjs';
import { AuthService } from 'src/app/auth/auth.service';
import { LoginPageComponent } from './login.component';
fdescribe('LoginPageComponent', () => {
```

```
let component: LoginPageComponent;
let fixture: ComponentFixture<LoginPageComponent>;
let authService: AuthService;
beforeEach(() => {
TestBed.configureTestingModule({
declarations: [LoginPageComponent],
imports: [ReactiveFormsModule],
providers: [
{
provide: AuthService,
useValue: {
SignIn: () => of(true), // Mock the AuthService's SignIn method
},
},
],
});
fixture = TestBed.createComponent(LoginPageComponent);
component = fixture.componentInstance;
authService = TestBed.inject(AuthService);
});
it('should create the component', () => {
expect(component).toBeTruthy();
});
it('should initialize the loginForm with the necessary controls', () => {
fixture.detectChanges(); // Trigger ngOnInit
const usernameControl = component.loginForm.get('username');
const passwordControl = component.loginForm.get('password');
expect(usernameControl).toBeTruthy();
expect(passwordControl).toBeTruthy();
});
it('should call the AuthService SignIn method and set loading to false', () => {
fixture.detectChanges();
const username = 'test@example.com';
const password = 'password123';
// Spy on AuthService's SignIn method to check if it was called
const signInSpy = spyOn(authService, 'SignIn').and.returnValue(Promise.resolve());
```

```
// Set the username and password values
component.loginForm.get('username')?.setValue(username);
component.loginForm.get('password')?.setValue(password);
// Call the login method
component.login();
// Expect that AuthService's SignIn was called with the username and password
expect(signInSpy).toHaveBeenCalledWith(username, password);
// Expect loading to be set to false after completing the request
expect(component.loading).toBeTruthy();
});
});
```

Description:

The first test case determines whether the component was correctly constructed.
The accuracy of the form's initialization with empty values is confirmed by the second test case.
In order to replicate a login activity, the third test case mocks the AuthService and confirms that the SignIn method is called.

Unit test for Sign up page:

Code:

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { ReactiveFormsModule } from '@angular/forms';
import { of } from 'rxjs';
import { AuthService } from 'src/app/auth/auth.service';
import { SignupPageComponent } from './signup-page.component';
fdescribe('SignupPageComponent', () => {
let component: SignupPageComponent;
let fixture: ComponentFixture<SignupPageComponent>;
let authService: AuthService;
beforeEach(() => {
TestBed.configureTestingModule({
declarations: [SignupPageComponent],
imports: [ReactiveFormsModule],
providers: [
{
```

```
provide: AuthService,
useValue: {
SignUp: () => of(true), // Mock the AuthService's SignUp method
},
},
],
});
fixture = TestBed.createComponent(SignupPageComponent);
component = fixture.componentInstance;
authService = TestBed.inject(AuthService);
});
it('should create the component', () => {
expect(component).toBeTruthy();
});
it('should initialize the registerForm with the necessary controls', () => {
fixture.detectChanges(); // Trigger ngOnInit
const usernameControl = component.registerForm.get('username');
const passwordControl = component.registerForm.get('password');
expect(usernameControl).toBeTruthy();
expect(passwordControl).toBeTruthy();
});
it('should call the AuthService SignUp method and set loading to false', () => {
fixture.detectChanges();
const username = 'test@example.com';
const password = 'password123';
// Spy on AuthService's SignUp method to check if it was called
const signUpSpy = spyOn(authService, 'SignUp').and.returnValue(Promise.resolve());
// Set the username and password values
component.registerForm.get('username')?.setValue(username);
component.registerForm.get('password')?.setValue(password);
// Call the register method
component.register();
// Expect that AuthService's SignUp was called with the username and password
expect(signUpSpy).toHaveBeenCalledWith(username, password);
// Expect loading to be set to false after completing the request
expect(component.loading).toBeTruthy();
```

```
});
});
```

Description:
We have created test cases on the Sign Up page to see if the user can log in with valid credentials and if the information they have entered is correct.
Additionally, it verifies that all fields have data entered and nothing is left blank.
We verified the user's registration by providing inaccurate information. and made an attempt to register with previously used data, which shouldn't permit the user to register once more.

Functional Tests:

Forgot Password Page:

```
import { ComponentFixture, TestBed, async } from '@angular/core/testing';
import { ReactiveFormsModule } from '@angular/forms';
import { ForgotPasswordPageComponent } from './forgot-password-page.component';
import { AuthService } from 'src/app/auth/auth.service';
import { of } from 'rxjs';

describe('ForgotPasswordPageComponent', () => {
  let component: ForgotPasswordPageComponent;
  let fixture: ComponentFixture<ForgotPasswordPageComponent>;
  let authService: jasmine.SpyObj<AuthService>;

  beforeEach(async(() => {
    const authServiceSpy = jasmine.createSpyObj('AuthService', ['ForgotPassword']);

    TestBed.configureTestingModule({
      declarations: [ForgotPasswordPageComponent],
      imports: [ReactiveFormsModule],
      providers: [
        { provide: AuthService, useValue: authServiceSpy }
      ]
    }).compileComponents();
```

```
  authService = TestBed.inject(AuthService) as jasmine.SpyObj<AuthService>;
}));

beforeEach(() => {
  fixture = TestBed.createComponent(ForgotPasswordPageComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});

it('should create', () => {
  expect(component).toBeTruthy();
});

it('should initialize the form with an empty username', () => {
  expect(component.passwordForm.get('username').value).toEqual('');
});

it('should set loading to true during password reset', () => {
  authService.ForgotPassword.and.returnValue(of({}));

  component.f['username'].setValue('test@example.com');
  component.resetPassword();

  expect(component.loading).toBeTrue();
});

it('should call authService.ForgotPassword with the correct username', () => {
  authService.ForgotPassword.and.returnValue(of({}));

  const testUsername = 'test@example.com';
  component.f['username'].setValue(testUsername);
  component.resetPassword();

  expect(authService.ForgotPassword).toHaveBeenCalledWith(testUsername);
});
```

```
  it('should set loading to false after password reset', () => {
    authService.ForgotPassword.and.returnValue(of({}));

    component.f['username'].setValue('test@example.com');
    component.resetPassword();

    fixture.whenStable().then(() => {
      expect(component.loading).toBeFalse();
    });
  });
});
```

Hangout Page:

```
import { ComponentFixture, TestBed, async } from '@angular/core/testing';
import { ActivatedRoute, Router } from '@angular/router';
import { AuthService } from 'src/app/auth/auth.service';
import { NotificationService } from 'src/app/core/services/notification.service';
import { HangoutPageComponent } from './hangout-page.component';
import { HangoutService } from 'src/app/services/hangout.service';
import { of } from 'rxjs';
import { HangoutWithUsers } from 'src/app/models/hangout.model';
import { User } from 'src/app/models/user.model';
import { Restaurant } from 'src/app/models/restaurant.model';

describe('HangoutPageComponent', () => {
  let component: HangoutPageComponent;
  let fixture: ComponentFixture<HangoutPageComponent>;
  let hangoutService: jasmine.SpyObj<HangoutService>;
  let authService: jasmine.SpyObj<AuthService>;
  let notificationService: jasmine.SpyObj<NotificationService>;
  let router: jasmine.SpyObj<Router>;
  let activatedRoute: jasmine.SpyObj<ActivatedRoute>;

  beforeEach(async(() => {
```

```
  const hangoutServiceSpy = jasmine.createSpyObj('HangoutService', [
    'getHangout',
    'castVote',
    'deactivateHangout',
  ]);
  const authServiceSpy = jasmine.createSpyObj('AuthService', ['user$']);
  const notificationServiceSpy = jasmine.createSpyObj('NotificationService', ['showNotification']);
  const routerSpy = jasmine.createSpyObj('Router', ['navigate']);
  const activatedRouteSpy = jasmine.createSpyObj('ActivatedRoute', ['params']);

  TestBed.configureTestingModule({
    declarations: [HangoutPageComponent],
    providers: [
      { provide: HangoutService, useValue: hangoutServiceSpy },
      { provide: AuthService, useValue: authServiceSpy },
      { provide: NotificationService, useValue: notificationServiceSpy },
      { provide: Router, useValue: routerSpy },
      { provide: ActivatedRoute, useValue: activatedRouteSpy },
    ],
  }).compileComponents();

  hangoutService = TestBed.inject(HangoutService) as jasmine.SpyObj<HangoutService>;
  authService = TestBed.inject(AuthService) as jasmine.SpyObj<AuthService>;
  notificationService = TestBed.inject(NotificationService) as jasmine.SpyObj<NotificationService>;
  router = TestBed.inject(Router) as jasmine.SpyObj<Router>;
  activatedRoute = TestBed.inject(ActivatedRoute) as jasmine.SpyObj<ActivatedRoute>;
}));

beforeEach(() => {
  fixture = TestBed.createComponent(HangoutPageComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});

it('should create', () => {
  expect(component).toBeTruthy();
```

```
  });

  it('should initialize loading to true', () => {
    expect(component.loading).toBeTrue();
  });

  it('should fetch hangout details and update loading on init', () => {
    const testHangout: HangoutWithUsers = { TEST01, true, Test1, 75088, test1, User1 };
    hangoutService.getHangout.and.returnValue(of(testHangout));

    component.ngOnInit();

    expect(component.loading).toBeFalse();
    expect(component.hangout).toEqual(testHangout);
    expect(component.participantsNames).toBe(testHangout.participants.map((user: User) =>
user.displayName).join(', '));
  });

  it('should subscribe to authService.user$ and update user on init', () => {
    const testUser: User = { userID1, test@gmail.com, Test1, http://testimage, false };
    authService.user$.and.returnValue(of(testUser));

    component.ngOnInit();

    expect(component.user).toEqual(testUser);
  });

  it('should check if the current user is the creator', () => {
    const testUser: User = { userID1, test@gmail.com, Test1, http://testimage, false };
    const testHangout: HangoutWithUsers = { TEST01, true, Test1, 75088, test1, User1 } ;
    authService.user$.and.returnValue(of(testUser));
    hangoutService.getHangout.and.returnValue(of(testHangout));

    component.ngOnInit();

    expect(component.isCreator()).toBe(testUser.uid === testHangout.createdBy.uid);
```

```
  });

  it('should cast vote when vote is called', () => {
    const testUser: User = { userID1, test@gmail.com, Test1, http://testimage, false };
    const testHangout: HangoutWithUsers = { TEST01, true, Test1, 75088, test1, User1 };
    const testRestaurant: Restaurant = { Test01, TestTaste, Denton Texas 76025, Indo-Asian};
    authService.user$.and.returnValue(of(testUser));
    hangoutService.getHangout.and.returnValue(of(testHangout));

    component.ngOnInit();
    component.vote(testRestaurant);

    expect(hangoutService.castVote).toHaveBeenCalledWith(testUser.uid, testHangout.id, testRestaurant.id);
  });

  it('should navigate to home page after successfully deactivating hangout', async () => {
    const testHangout: HangoutWithUsers = { TEST01, true, Test1, 75088, test1, User1 } ;
    hangoutService.getHangout.and.returnValue(of(testHangout));
    hangoutService.deactivateHangout.and.returnValue(Promise.resolve(true));

    await component.endHangout();

    expect(notificationService.showNotification).toHaveBeenCalledWith('success', 'Hangout has been deactivated.');
    expect(router.navigate).toHaveBeenCalledWith(['home']);
  });

  it('should show error notification if deactivating hangout fails', async () => {
    const testHangout: HangoutWithUsers = { TEST01, true, Test1, 75088, test1, User1 } ;
    hangoutService.getHangout.and.returnValue(of(testHangout));
    hangoutService.deactivateHangout.and.returnValue(Promise.resolve(false));

    await component.endHangout();

    expect(notificationService.showNotification).toHaveBeenCalledWith('error', 'Failed to deactivate hangout.');
  });
});
```

Home Page:

```
import { ComponentFixture, TestBed, async } from '@angular/core/testing';
import { HomePageComponent } from './home-page.component';
import { AuthService } from 'src/app/auth/auth.service';
import { HangoutService } from 'src/app/services/hangout.service';
import { NotificationService } from 'src/app/core/services/notification.service';
import { of } from 'rxjs';
import { Hangout } from 'src/app/models/hangout.model';

describe('HomePageComponent', () => {
  let component: HomePageComponent;
  let fixture: ComponentFixture<HomePageComponent>;
  let authService: jasmine.SpyObj<AuthService>;
  let hangoutService: jasmine.SpyObj<HangoutService>;
  let notificationService: jasmine.SpyObj<NotificationService>;

  beforeEach(async(() => {
    const authServiceSpy = jasmine.createSpyObj('AuthService', ['user$']);
    const hangoutServiceSpy = jasmine.createSpyObj('HangoutService', [
      'getActiveHangoutsForUser',
      'getPastHangoutsForUser',
    ]);
    const notificationServiceSpy = jasmine.createSpyObj('NotificationService', ['']);

    TestBed.configureTestingModule({
      declarations: [HomePageComponent],
      providers: [
        { provide: AuthService, useValue: authServiceSpy },
        { provide: HangoutService, useValue: hangoutServiceSpy },
        { provide: NotificationService, useValue: notificationServiceSpy },
      ],
    }).compileComponents();
```

```
  authService = TestBed.inject(AuthService) as jasmine.SpyObj<AuthService>;
  hangoutService = TestBed.inject(HangoutService) as jasmine.SpyObj<HangoutService>;
  notificationService = TestBed.inject(NotificationService) as jasmine.SpyObj<NotificationService>;
}));

beforeEach(() => {
  fixture = TestBed.createComponent(HomePageComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});

it('should create', () => {
  expect(component).toBeTruthy();
});

it('should set loading to true initially', () => {
  expect(component.loading).toBeFalse();
});

it('should subscribe to authService.user$ and load active and past hangouts for the user', () => {
  const testUser = { uid: 'testUserId' };
  const testActiveHangouts: Hangout[] = [{ test1, 75093, test1 }];
  const testPastHangouts: Hangout[] = [{ test2, 75093, test2 }];

  authService.user$.and.returnValue(of(testUser));
  hangoutService.getActiveHangoutsForUser.and.returnValue(of(testActiveHangouts));
  hangoutService.getPastHangoutsForUser.and.returnValue(of(testPastHangouts));

  component.ngOnInit();

  expect(component.activeHangouts$).toBeDefined();
  expect(component.pastHangouts$).toBeDefined();

  component.activeHangouts$.subscribe((hangouts) => {
    expect(hangouts).toEqual(testActiveHangouts);
  });
```

```
    component.pastHangouts$.subscribe((hangouts) => {
      expect(hangouts).toEqual(testPastHangouts);
    });

    expect(component.loading).toBeFalse();
  });

  it('should unsubscribe from subscriptions on ngOnDestroy', () => {
    const unsubscribeSpy = spyOn(component['subs'][0], 'unsubscribe');

    component.ngOnDestroy();

    expect(unsubscribeSpy).toHaveBeenCalled();
  });
});
```

Login Page:

```
import { ComponentFixture, TestBed, async, fakeAsync, tick } from '@angular/core/testing';
import { ReactiveFormsModule } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { LoginPageComponent } from './login.component';
import { AuthService } from 'src/app/auth/auth.service';
import { of, throwError } from 'rxjs';

describe('LoginPageComponent', () => {
  let component: LoginPageComponent;
  let fixture: ComponentFixture<LoginPageComponent>;
  let authService: jasmine.SpyObj<AuthService>;
  let router: jasmine.SpyObj<Router>;

  beforeEach(async(() => {
    const authServiceSpy = jasmine.createSpyObj('AuthService', ['SignIn']);
    const routerSpy = jasmine.createSpyObj('Router', ['navigate']);
```

```
  TestBed.configureTestingModule({
    declarations: [LoginPageComponent],
    imports: [ReactiveFormsModule],
    providers: [
      { provide: AuthService, useValue: authServiceSpy },
      { provide: Router, useValue: routerSpy },
    ],
  }).compileComponents();

  authService = TestBed.inject(AuthService) as jasmine.SpyObj<AuthService>;
  router = TestBed.inject(Router) as jasmine.SpyObj<Router>;
}));

beforeEach(() => {
  fixture = TestBed.createComponent(LoginPageComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});

it('should create', () => {
  expect(component).toBeTruthy();
});

it('should initialize the form with empty username and password fields', () => {
  expect(component.loginForm.get('username').value).toEqual('');
  expect(component.loginForm.get('password').value).toEqual('');
});

it('should set loading to true during login', () => {
  authService.SignIn.and.returnValue(of({}));

  component.f['username'].setValue('test@example.com');
  component.f['password'].setValue('testPassword');
  component.login();
```

```
  expect(component.loading).toBeTrue();
});

it('should call authService.SignIn with the correct username and password', () => {
  authService.SignIn.and.returnValue(of({}));

  const testUsername = 'test@example.com';
  const testPassword = 'testPassword';
  component.f['username'].setValue(testUsername);
  component.f['password'].setValue(testPassword);
  component.login();

  expect(authService.SignIn).toHaveBeenCalledWith(testUsername, testPassword);
});

it('should navigate to the home page after successful login', fakeAsync(() => {
  authService.SignIn.and.returnValue(of({}));

  component.f['username'].setValue('test@example.com');
  component.f['password'].setValue('testPassword');
  component.login();

  tick(); // simulate the passage of time

  expect(router.navigate).toHaveBeenCalledWith(['/home']);
}));

it('should set loading to false after login', fakeAsync(() => {
  authService.SignIn.and.returnValue(of({}));

  component.f['username'].setValue('test@example.com');
  component.f['password'].setValue('testPassword');
  component.login();

  tick(); // simulate the passage of time
```

```
    expect(component.loading).toBeFalse();
  }));

  it('should show an error notification on login failure', fakeAsync(() => {
    const errorMessage = 'Invalid credentials';
    authService.SignIn.and.returnValue(throwError(errorMessage));

    component.f['username'].setValue('test@example.com');
    component.f['password'].setValue('testPassword');
    component.login();

    tick(); // simulate the passage of time

    expect(component.loading).toBeFalse();
    // Add your own notification service method verification here
  }));
});
```

Sign Up page:

```
import { ComponentFixture, TestBed, async, fakeAsync, tick } from '@angular/core/testing';
import { ReactiveFormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { SignupPageComponent } from './signup-page.component';
import { AuthService } from 'src/app/auth/auth.service';
import { of, throwError } from 'rxjs';

describe('SignupPageComponent', () => {
  let component: SignupPageComponent;
  let fixture: ComponentFixture<SignupPageComponent>;
  let authService: jasmine.SpyObj<AuthService>;
  let router: jasmine.SpyObj<Router>;

  beforeEach(async(() => {
    const authServiceSpy = jasmine.createSpyObj('AuthService', ['SignUp']);
```

```
  const routerSpy = jasmine.createSpyObj('Router', ['navigate']);

  TestBed.configureTestingModule({
    declarations: [SignupPageComponent],
    imports: [ReactiveFormsModule],
    providers: [
      { provide: AuthService, useValue: authServiceSpy },
      { provide: Router, useValue: routerSpy },
    ],
  }).compileComponents();

  authService = TestBed.inject(AuthService) as jasmine.SpyObj<AuthService>;
  router = TestBed.inject(Router) as jasmine.SpyObj<Router>;
}));

beforeEach(() => {
  fixture = TestBed.createComponent(SignupPageComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});

it('should create', () => {
  expect(component).toBeTruthy();
});

it('should initialize the form with empty username, password, and name fields', () => {
  expect(component.registerForm.get('username').value).toEqual('');
  expect(component.registerForm.get('password').value).toEqual('');
  expect(component.registerForm.get('name').value).toEqual('');
});

it('should set loading to true during registration', () => {
  authService.SignUp.and.returnValue(of({}));

  component.f['username'].setValue('test@example.com');
  component.f['password'].setValue('testPassword');
```

```
  component.f['name'].setValue('John Doe');
  component.register();

  expect(component.loading).toBeTrue();
});

it('should call authService.SignUp with the correct username, password, and name', () => {
  authService.SignUp.and.returnValue(of({}));

  const testUsername = 'test@example.com';
  const testPassword = 'testPassword';
  const testName = 'John Doe';
  component.f['username'].setValue(testUsername);
  component.f['password'].setValue(testPassword);
  component.f['name'].setValue(testName);
  component.register();

  expect(authService.SignUp).toHaveBeenCalledWith(testUsername, testPassword, testName);
});

it('should navigate to the login page after successful registration', fakeAsync(() => {
  authService.SignUp.and.returnValue(of({}));

  component.f['username'].setValue('test@example.com');
  component.f['password'].setValue('testPassword');
  component.f['name'].setValue('John Doe');
  component.register();

  tick(); // simulate the passage of time

  expect(router.navigate).toHaveBeenCalledWith(['/login']);
}));

it('should set loading to false after registration', fakeAsync(() => {
  authService.SignUp.and.returnValue(of({}));
```

```
    component.f['username'].setValue('test@example.com');
    component.f['password'].setValue('testPassword');
    component.f['name'].setValue('John Doe');
    component.register();

    tick(); // simulate the passage of time

    expect(component.loading).toBeFalse();
  }));

  it('should show an error notification on registration failure', fakeAsync(() => {
    const errorMessage = 'Registration failed';
    authService.SignUp.and.returnValue(throwError(errorMessage));

    component.f['username'].setValue('test@example.com');
    component.f['password'].setValue('testPassword');
    component.f['name'].setValue('John Doe');
    component.register();

    tick(); // simulate the passage of time

    expect(component.loading).toBeFalse();
  }));
});
```

Implemented Features and Limitations:

We successfully incorporated several features into our "Where to Eat" project based on the valuable feedback received during the peer review sessions. These enhancements aimed to address critical aspects identified through rigorous testing, improved documentation, enhanced security measures, and scalability considerations.

1. **Documentation Enhancement:** Our team meticulously revised the documentation, ensuring clarity and comprehensiveness. The updated user documentation now encompasses all essential aspects of the application, making it more user-friendly and easier to comprehend.

2. **Testing and Quality Assurance:** Following the feedback's emphasis on testing, we implemented a series of test cases to detect and rectify potential bugs or issues, thereby enhancing the overall quality of our application.

3. **Security and Privacy Measures:** Our team prioritized the implementation of Firebase Authentication, adhering to the latest best practices. This fortification significantly bolsters the protection of user data within the application.

4. **Scalability Improvements:** To address concerns regarding scalability, we made significant architectural adjustments to ensure our application can seamlessly accommodate potential growth in the user base. These changes were made in both the database and infrastructure to enable smooth scalability as needed.

Limitations and Known Issues:

While striving to enhance our project, certain limitations and known issues were identified. These include:

Feature Completion: Due to time constraints, certain planned features were not fully implemented. This includes [mention specific features]. However, these features remain part of our future development plans.

Plans for the Next Phase:

Moving forward, our team is committed to further elevating the "Where to Eat" project in the upcoming semester. Our focus will be on:

1. **Feature Completion:** We aim to finalize the pending features that were previously identified but not completed due to time limitations. This involves [specific features or functionalities].

2. **Enhancements Based on Feedback:** We will continuously track feedback and user suggestions to implement further improvements to the application's usability, performance, and overall user experience.

3. **Advanced Functionality Integration:** Exploring the integration of additional functionalities to enrich the application's capabilities and make it more versatile for our users.

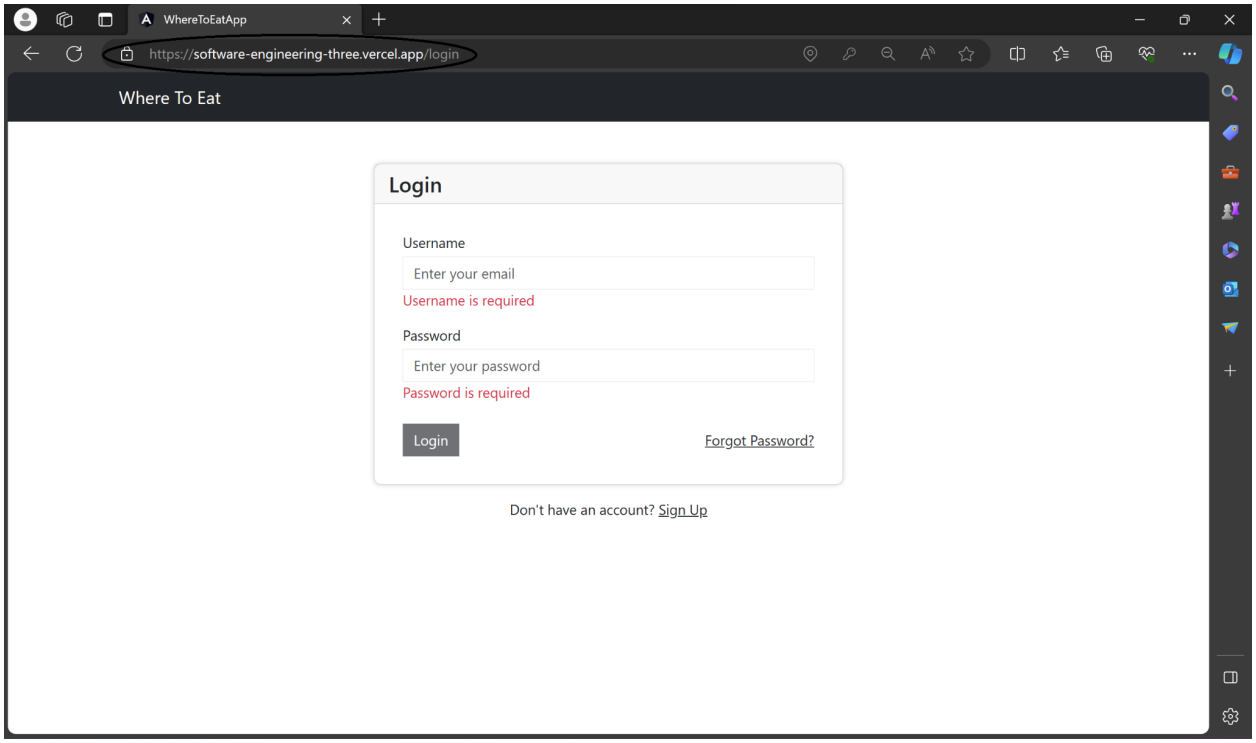
4. **Optimization for Efficiency:** Continuously optimizing the codebase and backend infrastructure to ensure efficiency, speed, and reliability, allowing for a seamless user experience.

User Manual:

Here is a step-by-step follow-up on accessing our application WHERE TO EAT.

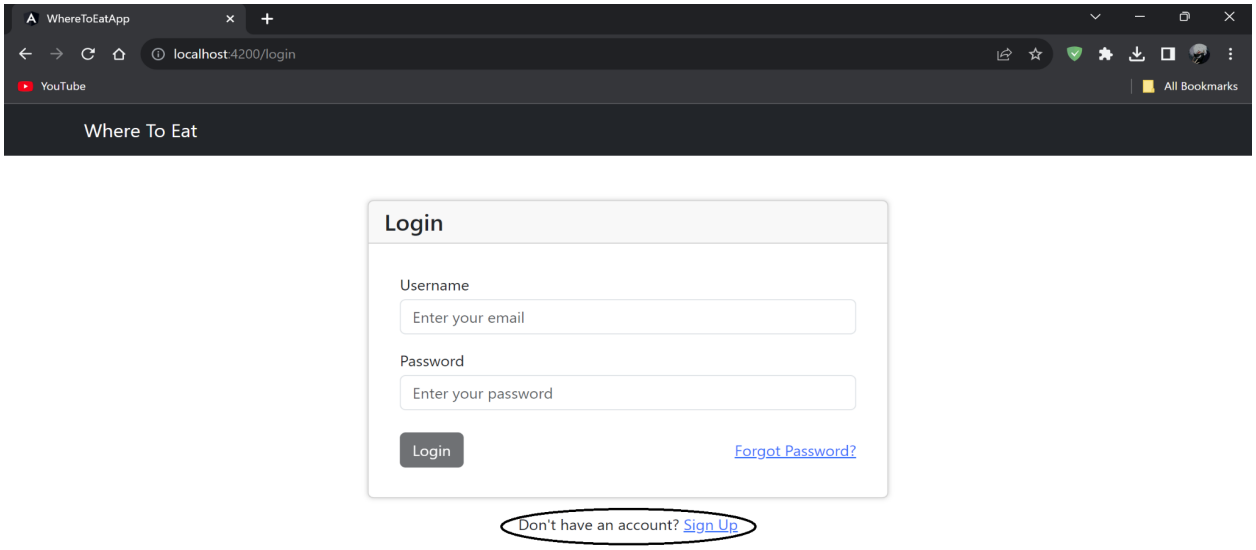
Step 1:

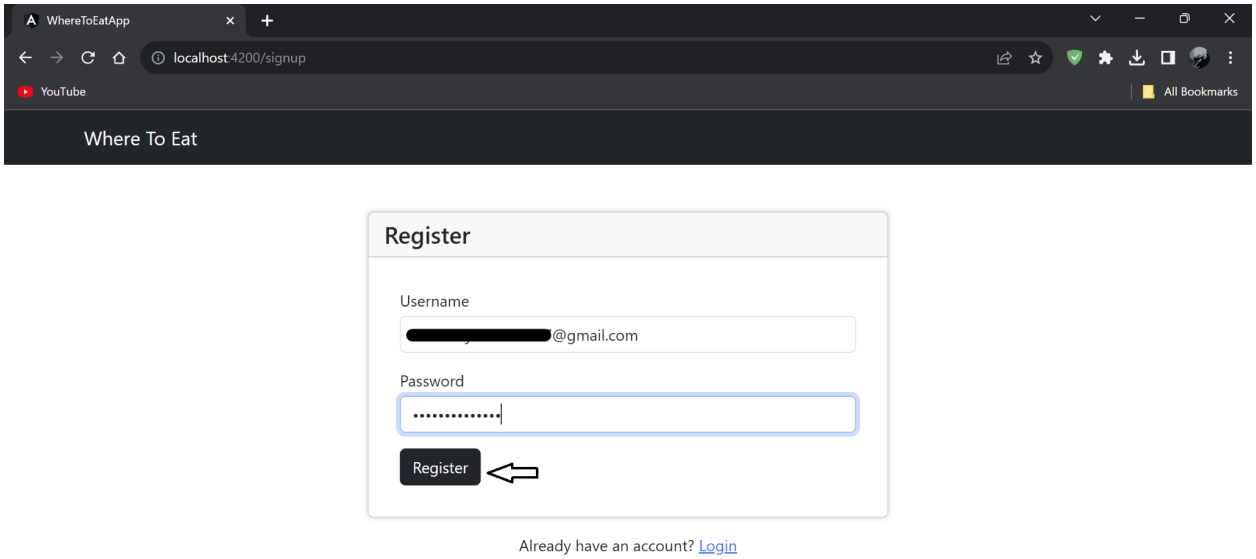
Once the user accesses the page using the URL, the User gets redirected to the landing page.



Step 2:

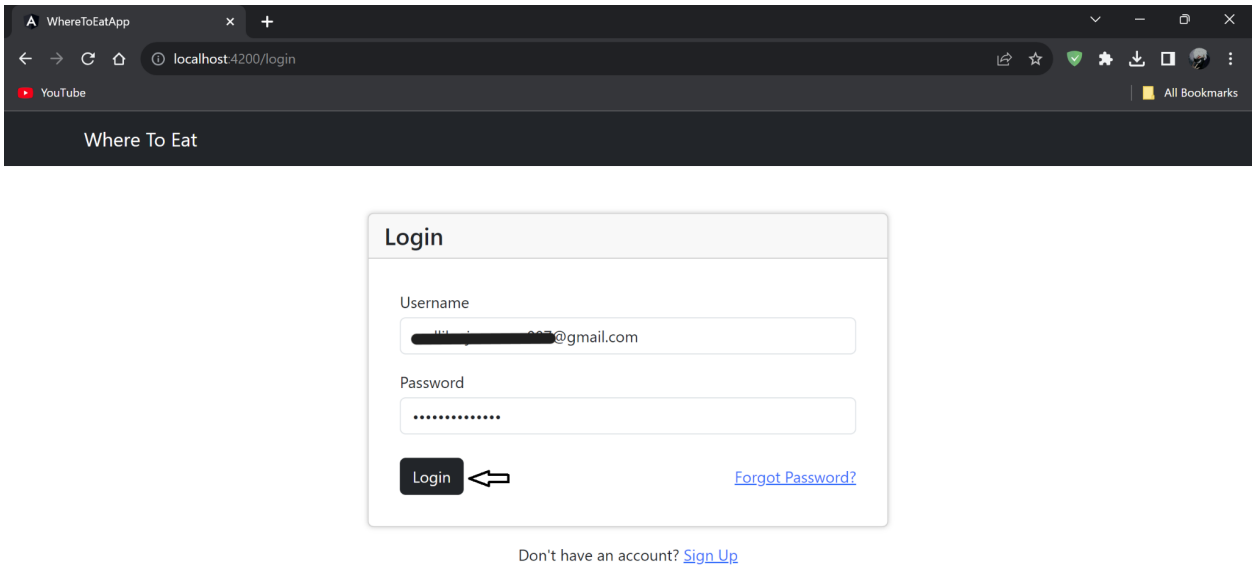
If the client is a new user, he can use the app's Sign Up feature for making a new UserAcc.


WhereToEatApp
localhost:4200/login
YouTube
All Bookmarks
Where To Eat
Login
Username
Enter your email
Password
Enter your password
Login
Forgot Password?
Don't have an account? Sign Up


WhereToEatApp
localhost:4200/signup
YouTube
All Bookmarks
Where To Eat
Register
Username
[redacted]@gmail.com
Password
Register
Already have an account? Login

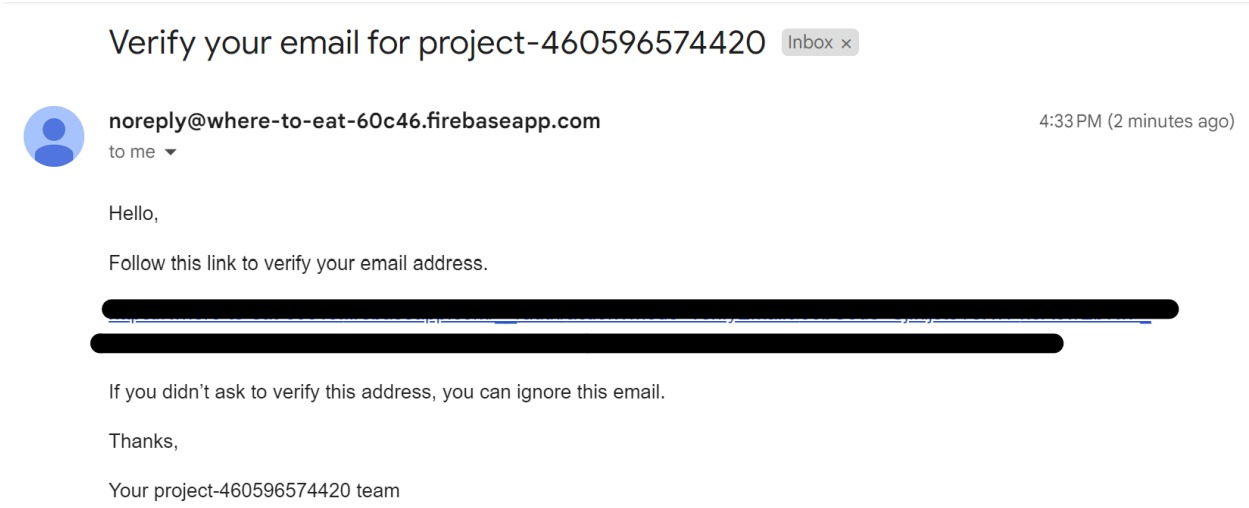
Step 3:

Once the user gets to register he is redirected to the Login page where he gets to enter the same credentials to Login.

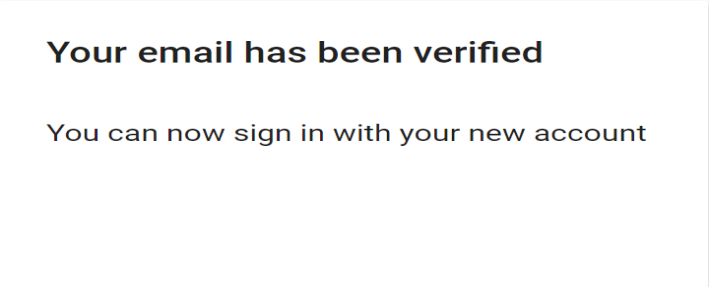


Step 4:

After step 3, the User is done with the registration and login, and the User gets a verification email on the registered email.

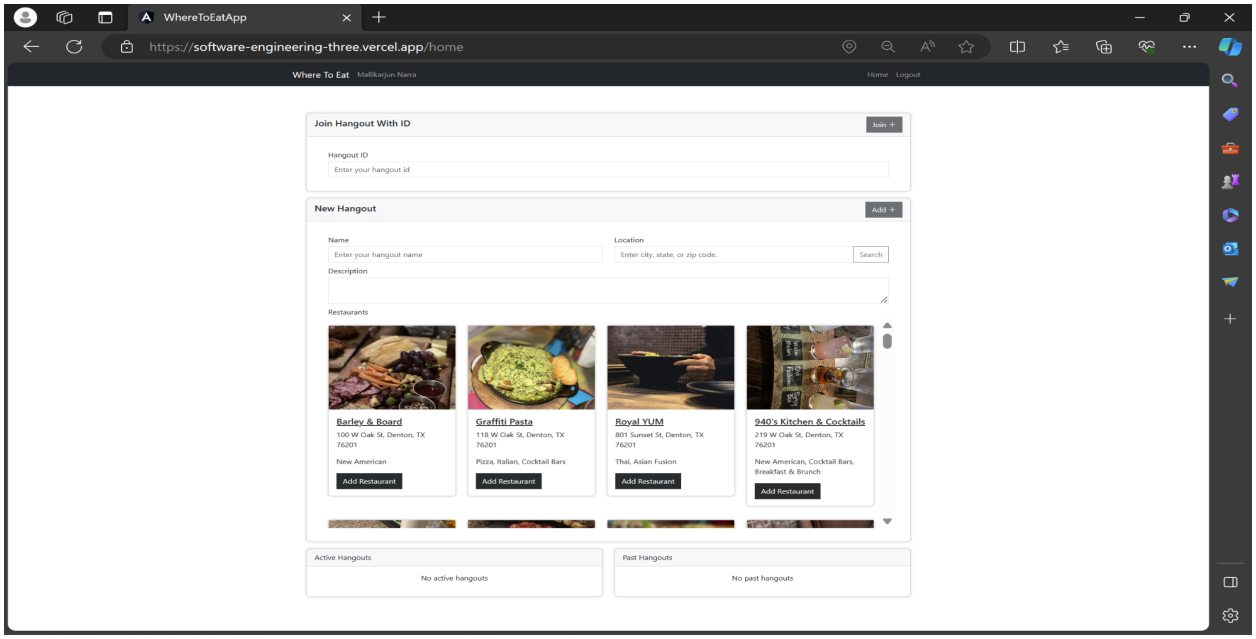


Once verified, the User gets a confirmation that the email is verified.

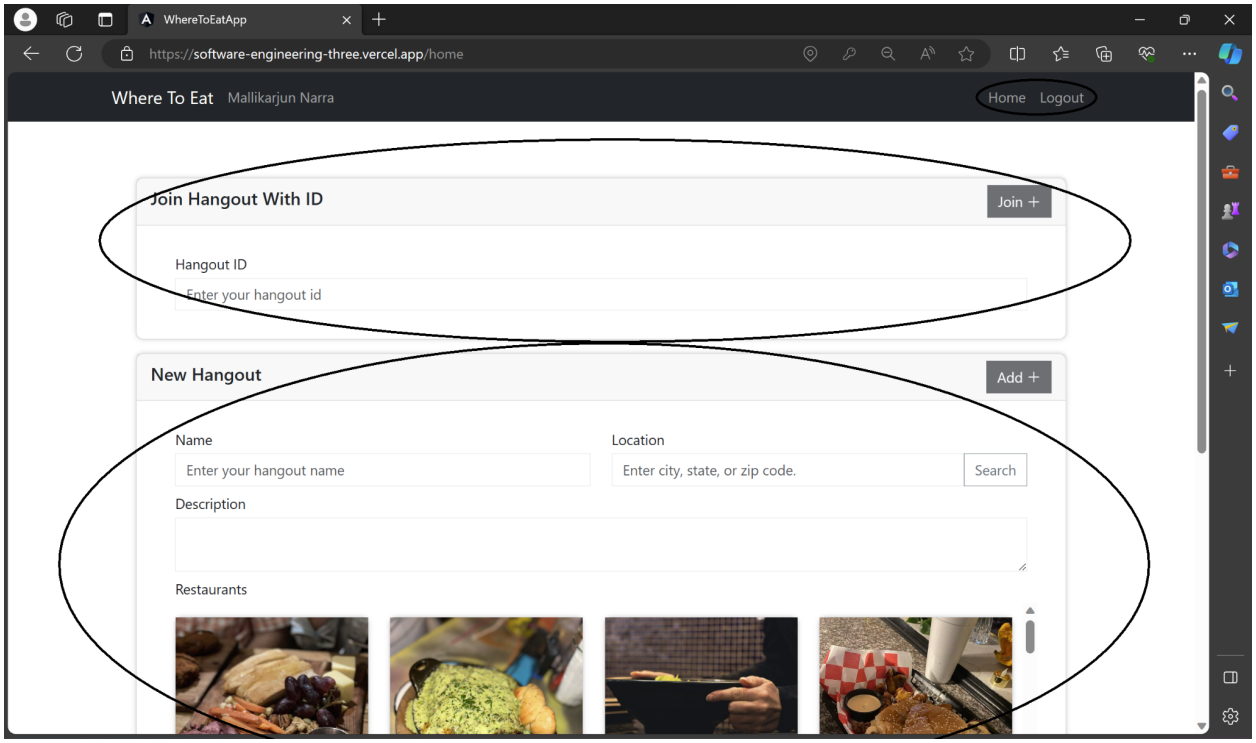


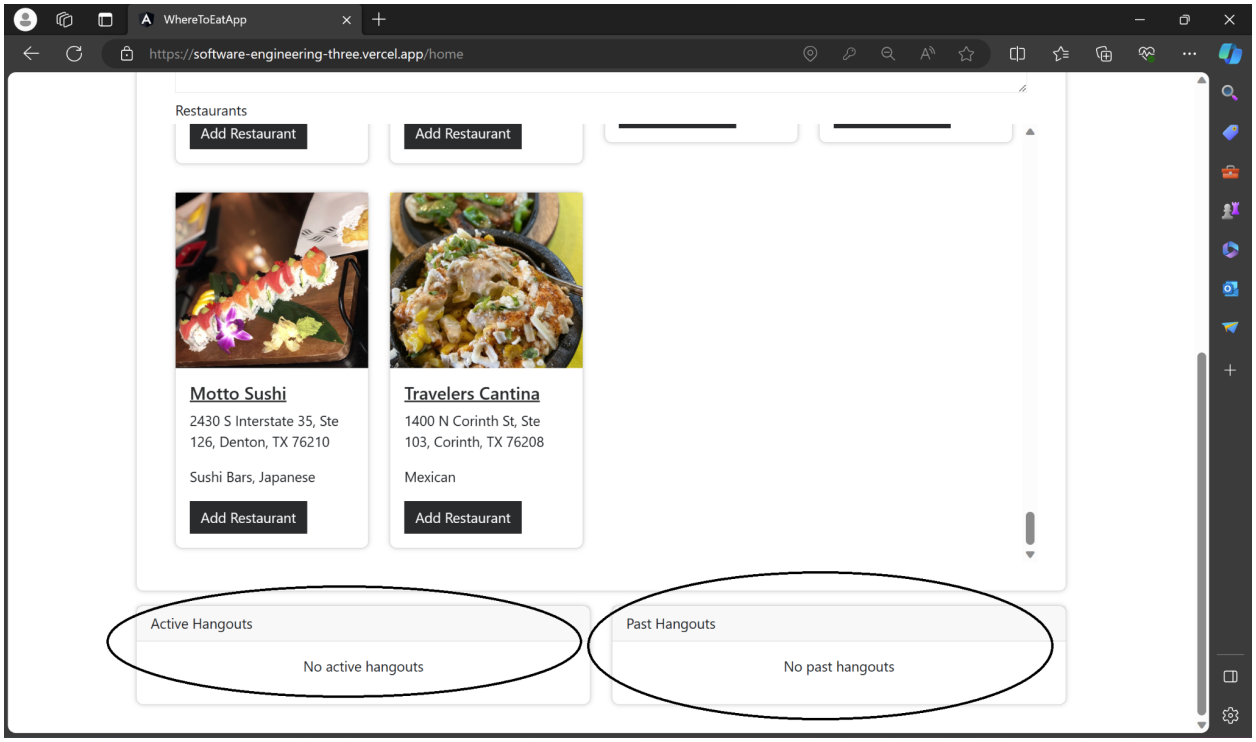
Step 5:

Once the user successfully logs in, the User gets redirected to the home page of our application.

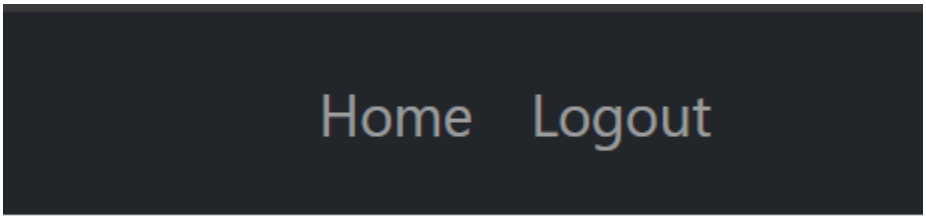


A few key components of the application are

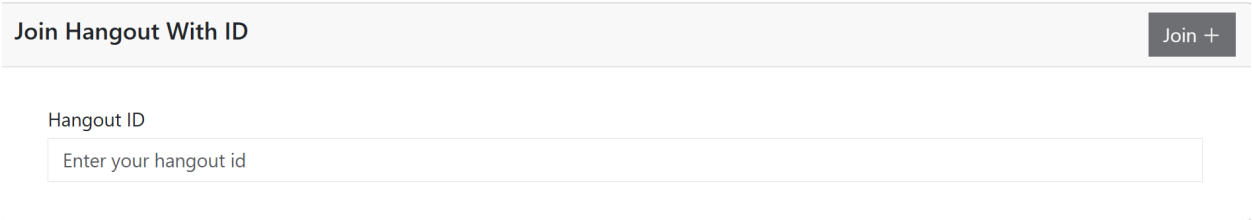




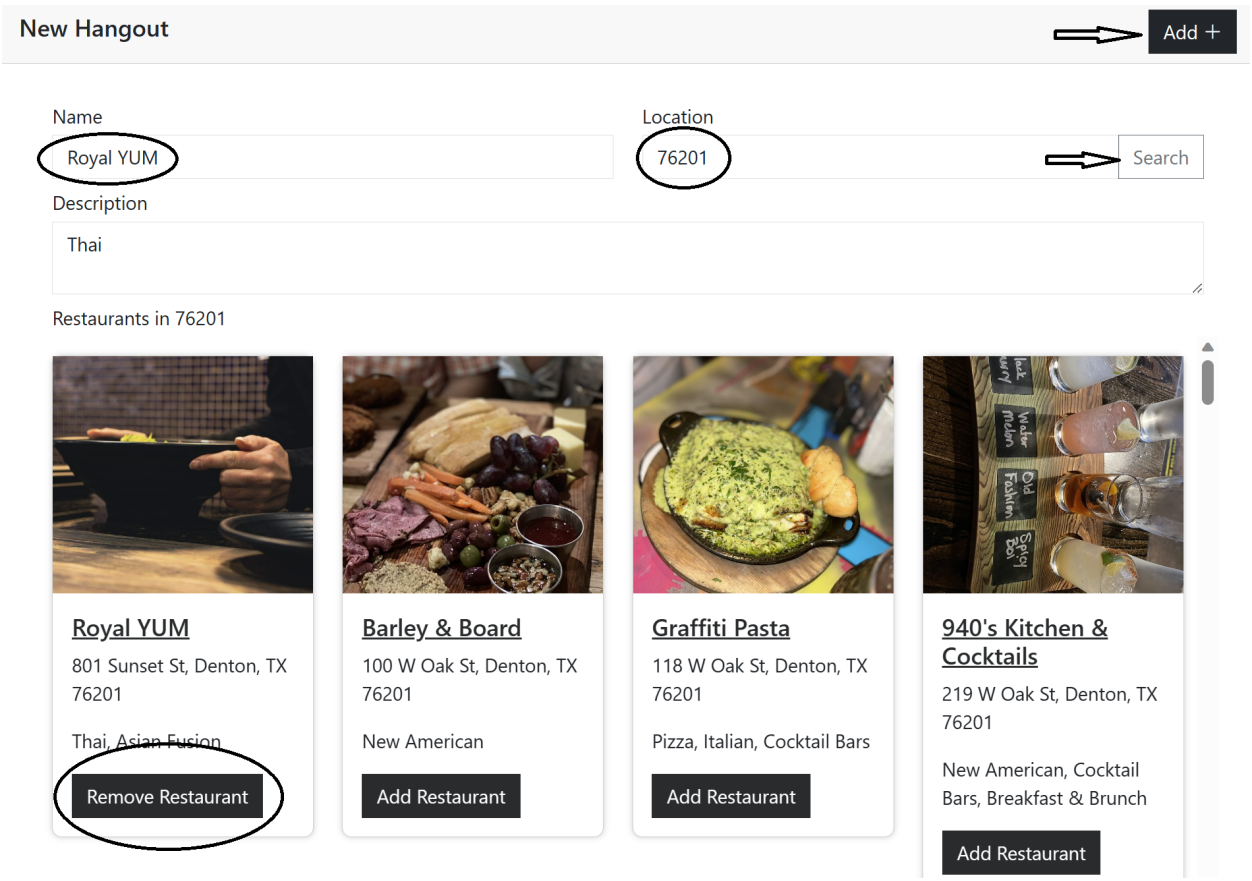
- In the top right corner, we have **Home | Logout** buttons which are responsible for redirecting to the home page when clicked on Home and Logging out of the application when clicked on LogOut.



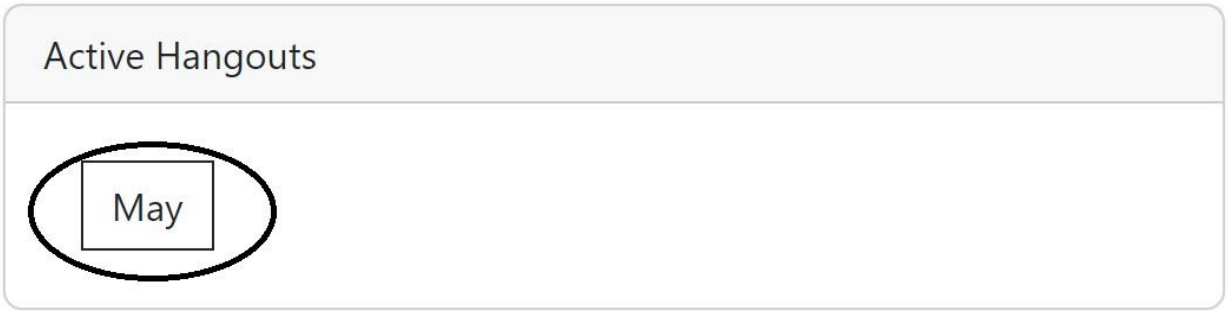
- If the user already has a hangout created and wants to join the **Join Hangout With ID** Section, the User can join the active hangout ID and then Click on **Join +**



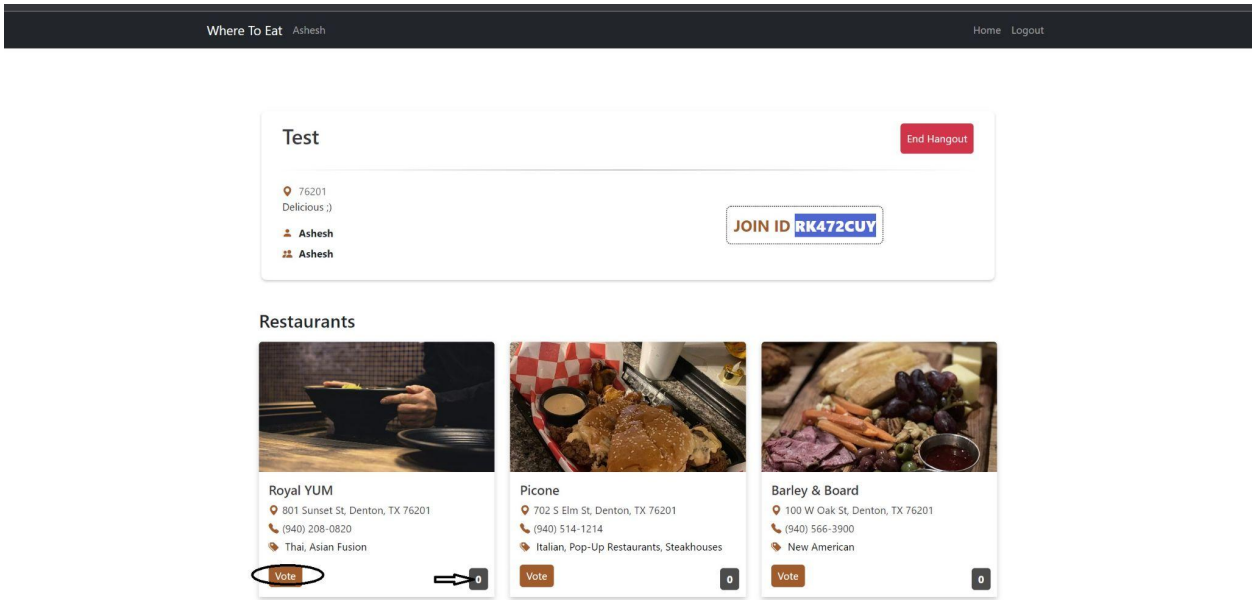
- We also have a feature to create a new Hangout in the **New Hangout** section using the **Add +** button which typically creates a new Hangout when you give a verified **Hangout Name, Zip Code, and Description then search / Select a Hangout from the Dropdown of options available** Such that your hangout group can join.



- Current hangouts are displayed in the **Active Hangouts** section, where all the current sessions are displayed. User/s can hop from hangout to hangout. For demo purposes, we have created a test Hangout known as “May”(Hangout Name) as shown below.

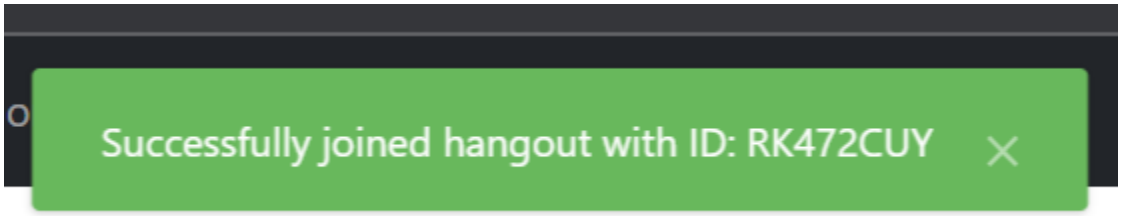


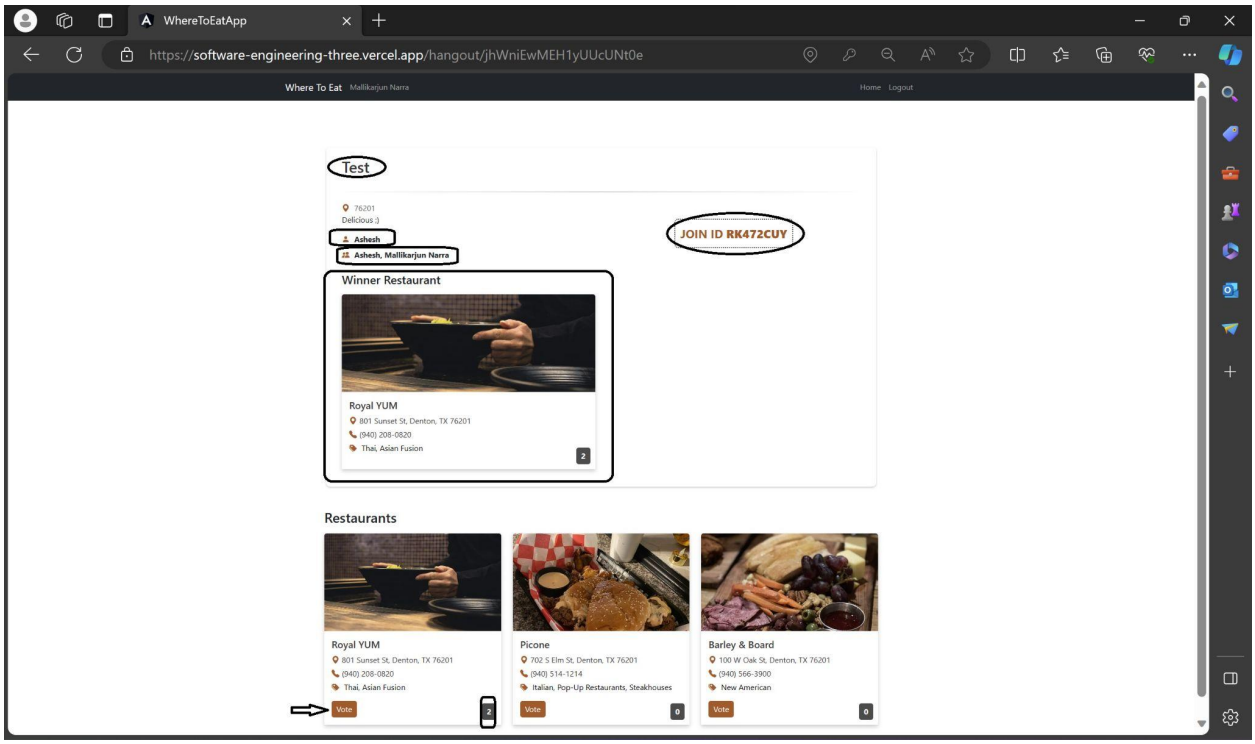
- If we open the Current Active Hangout, we can view all the details about the Hangout, such as the “creator's information”, “participant list”, “invite and zip code”, “Hangout description”, and “added restaurants with a voting system”.



- Other participants can also join using the “Join Hangout With ID” Of the web application to directly join Hangouts

- Once successfully joined we get a success message like this and the user gets navigated to the already existing Active hangout

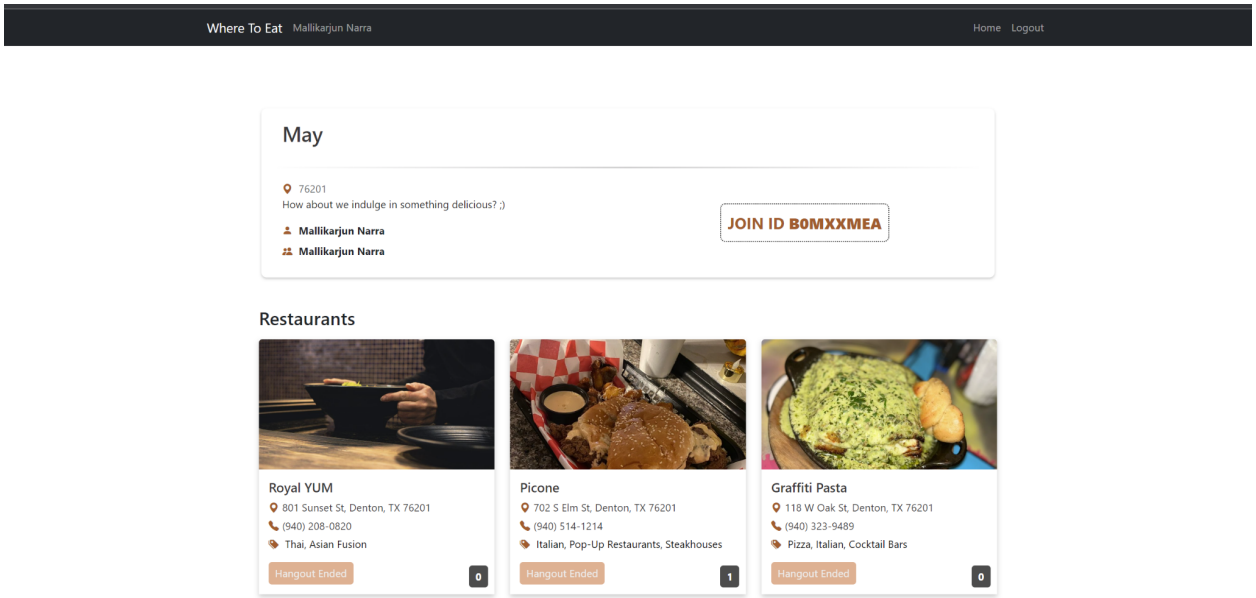




- In the above demonstration, we have the details of the hangouts like “Hangout Name”, “Admin Name”, “Users”, “and “votes” casted restaurant wise and the restaurant with the majority of votes

wins which is represented as the winner restaurant division, once done with the hangout only the admin can end the hangout using the End Hangout button.

- Our web application also has a **Past Hangouts module** where user/s view a list of past hangouts (History) i.e Foodie Adventure Hangout, Movie Night Out, Coffee and Conversation, etc. This Typically displays the Hangouts that are terminated with all details(i.e admin details, participants details, restaurant with the majority of votes etc.,) and all the functional elements of the past hangout are disabled/turned off.



Compilation and Run Instructions:

- Take a clone of the project repository from your Git or other version control system.
    - git clone https://github.com/brofcb/software-engineering.git
    - cd WhereToEatApp
    - Use npm to install project dependencies - npm install
- Configure Firebase
    - Create a Firebase project via the Firebase Console to configure Firebase.
    - Set up Firebase using your Firebase credentials and initialize it in your app.

      ```
      // Firebase configuration setup
      const firebaseConfig = {
      apiKey: "<YOUR_API_KEY>",
      authDomain: "<YOUR_AUTH_DOMAIN>",
      projectId: "<YOUR_PROJECT_ID>",
      storageBucket: "<YOUR_STORAGE_BUCKET>",
      messagingSenderId: "<YOUR_MESSAGING_SENDER_ID>",
      appId: "<YOUR_APP_ID>"
      };
      ```
- Configure Yelp
    - Create an app via the yelp for developers to generate API key
    - Set up the Yelp API key and configure it in your app
    - YELP_API_KEY: process.env.YELP_API_KEY || '<Your API key>'
- Running the Web Application:
    - Start the web application server - npm start
- Access the web application:
    - Open a web browser and go to http://localhost:PORT (insert port number here to indicate the port your server is operating on).

    - You will be sent to the login page, where you may get the four essential elements: registration, password reset, username, and password.
- Test Cases:
    - Make sure your web application's test cases cover a variety of scenarios and features before beginning any testing.You may create and run these test cases using testing frameworks such as Mocha, Jasmine, or Jest.
- Run the test suite:
    - npm test
    - Verify that all features and components are operating as intended by reviewing the test results.

A brief reflection on the "WhereToEat" project:

Accomplishments:

We've made progress in developing our app called "WhereToEat." We have successfully implemented features such as user authentication, hangout creation, adding restaurants, voting, and managing data using Firebase and Google Places API. Our interface is designed to be user friendly making it easy to navigate, create hangouts, vote and manage profiles. Additionally, we have added a feature called "Past Hangouts," which allows users to explore hangouts like the Foodie Adventure Hangout and Movie Night Out.

Our expertise in data management is evident with our use of Firestore for maintaining our database. We prioritize planning and documentation by following a defined timeline and providing user guides on GitHub.

Areas for Improvement:

We are committed to growth and ensuring an experience for the users. We are focusing on enhancing testing procedures, including unit and integration testing to guarantee functionality. Scalability is a priority as more users join our platform. User feedback is incredibly valuable to us as it helps us improve and introduce features based on their preferences.

Project Risks:

While we are excited about the progress we've made so far we recognize challenges that may arise along the way. We understand the risks associated with relying on services addressing technical obstacles promptly and staying ahead of evolving user preferences. We have considered factors such as resources, scalability issues, privacy concerns, security measures, and data protection, in our plans.

No worries we're actively working on preventing and resolving issues.

Regarding project resources:

We have everything planned out. People, funding, and necessary tools. Managing and licensing resources such as APIs and design libraries are vital for achieving success.

Oh by the way our app also has a feature called "Current Active Hangout." It reveals all the details about hangouts, including the creator's information, list of participants, invitation-specific hangout description, and even a fun voting system to help decide on restaurants.

Project Timeline:

Here, we have used **Gantt Chart** to show the project timeline: Gantt chart

Members Contribution Table:

Member name	Contribution description	Overall Contribution (%)	Note(if applicable)
Ibrahim Alkuwaifi	I worked on adding the Yelp fusion API. I also worked on updating the progress for this week. I also worked on hosting the web app.	11.11%	
Mohammad Khaja Moinuddin	Worked on brief reflection on the project and updated the timeline of Gantt chart.	11.11%	
Teja Peddi	Worked on UML,Sequence, Usecase including one normal case and one error case	11.11%	
Mohammed Abdul Wahid	I updated the meeting minutes for Deliverable -5 in the project repository on GitHub mentioning the Agenda Items/Topics Discussed and Action Items/Tasks Assigned.	11.11%	
Gouthami Gongidi	Helped to work on Unit Case diagram	11.11%	
Mallikarjun Narra	Drafted User manual with the summery all key components	11.11%	
Ashesh kumar Pinisetti	Added compilation/run instructions Added note-deliverable-5 in gitlab Updated file directory structure in gitlab	11.11%	

Sai Kumar Gurram	Added Unit test cases and Functional Test Cases	11.11%	
Vishal Kandunuri	Added Feedback-Peer review	11.11%	