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 54320.008

Requirements for Phase 1

Scope Update

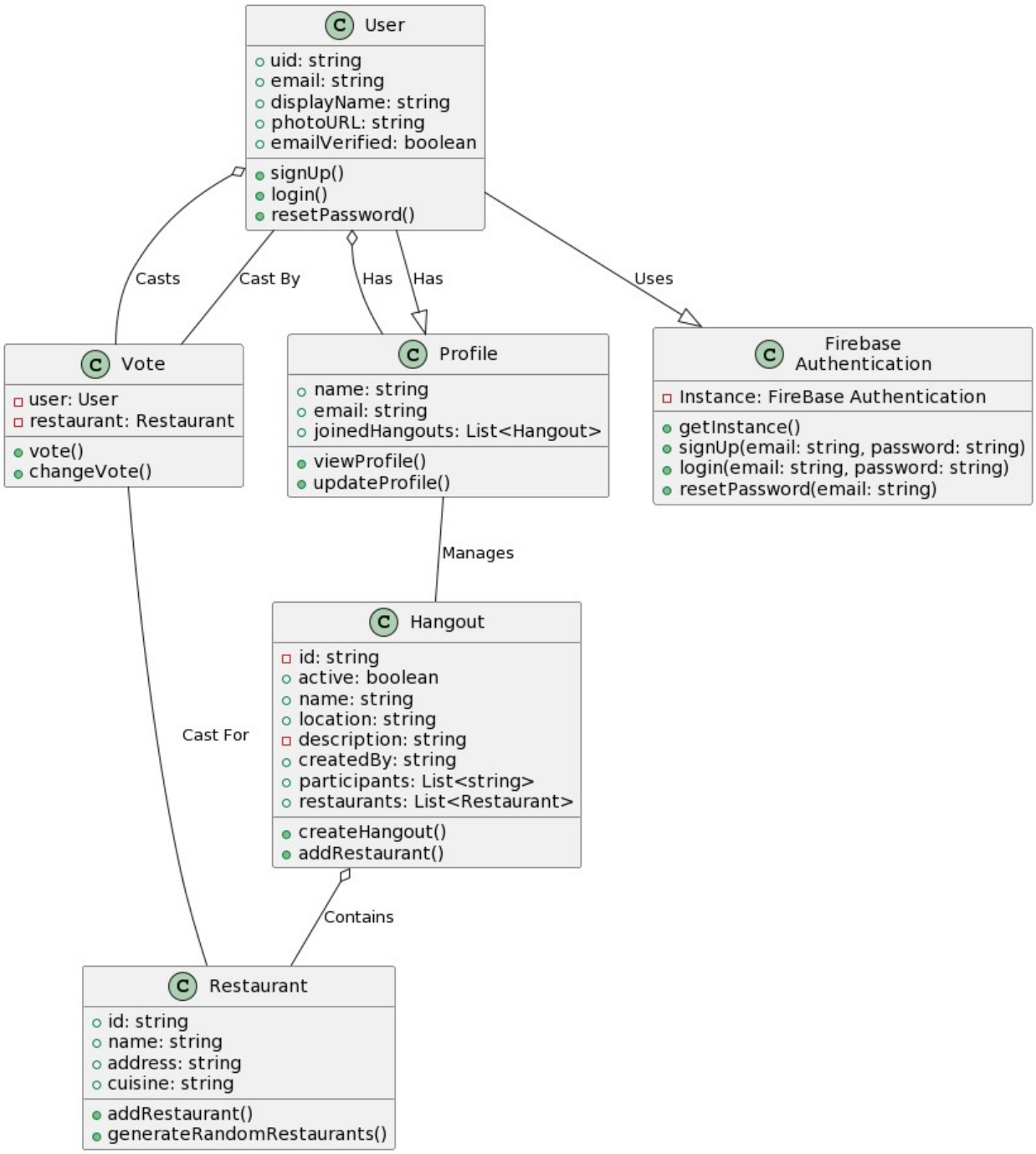
The scope for Phase 1 remains mostly unchanged from Deliverable 2. Below are the more detailed descriptions of the requirements specified for Phase 1.

- 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 (80%)
  - 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.

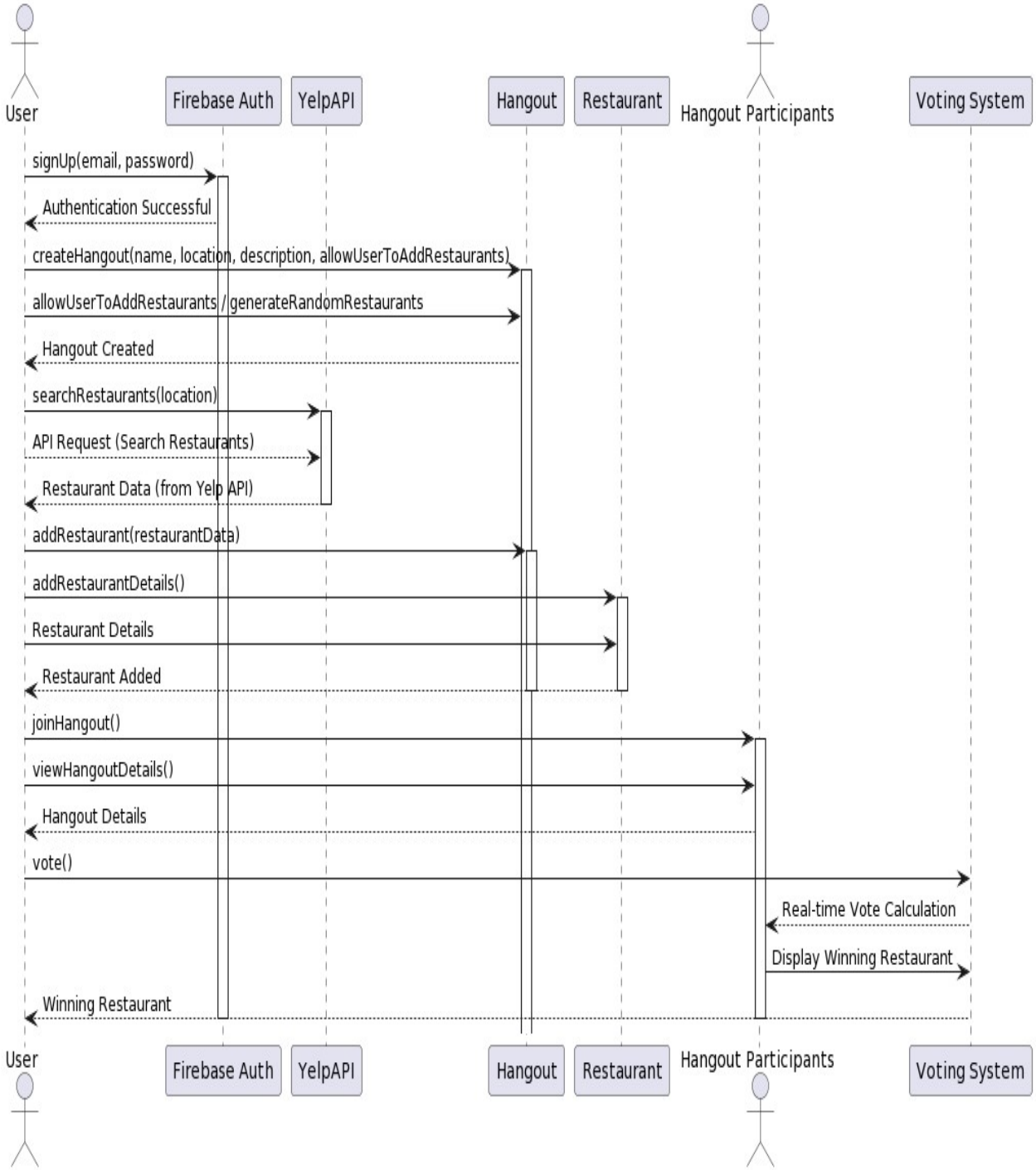
        - Add toggles or checkboxes in the hangout creation form to allow this selection. Implement logic to handle either scenario.
- Adding Restaurants (50%)
    - User-Added Restaurants
        - Enable users to add their own restaurants, capturing relevant information like name, address, cuisine, etc.
        - Create a form that captures the necessary details. Validate the information before saving it to the database.
    - Random Restaurant Generation
        - Implement a feature to generate random restaurants for users who don't want to add their own.
        - Use an algorithm to generate random restaurants based on predefined or user-defined criteria. This could include pulling data from a local database or using machine learning to suggest options based on user preferences.

UML Design For Phase-1:

Class diagram:

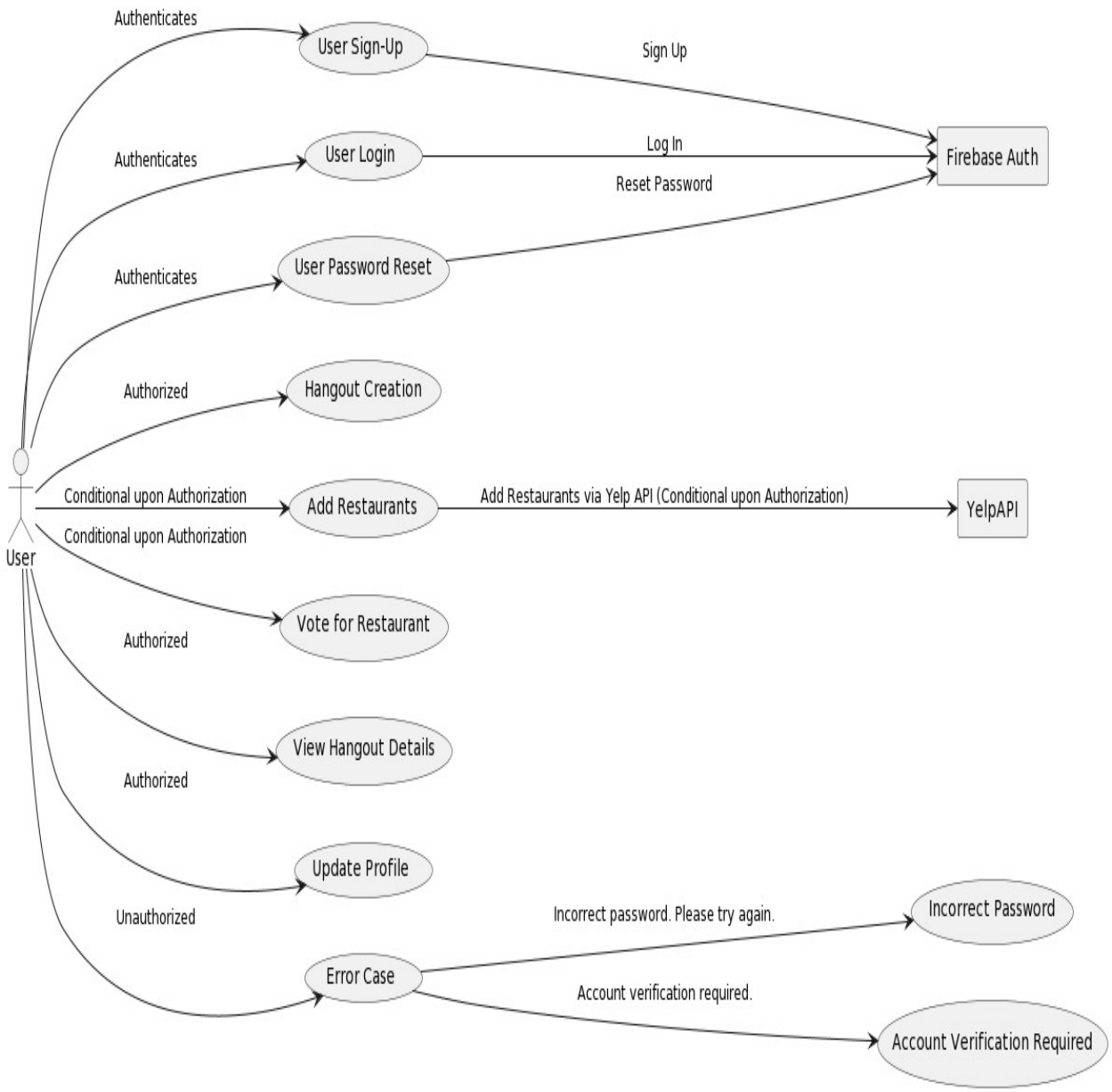


Sequence Diagram:



Use case diagram:

Error case included:



Unit Test cases:

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();
});
```

In this example, we have written three test cases:

The first test case checks if the component is created successfully.

The second test case checks if the form is initialized with an empty username field when ngOnInit is called.

The third test case checks if the resetPassword function sets the loading flag to true during the password reset and then back to false after the operation is complete.

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

The first test case checks if the component is created successfully.

The second test case verifies that the form is initialized correctly with empty values.

The third test case simulates the creation of a new hangout by setting form values and checking that the createHangout method is called with the expected data.

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

The first test case checks if the component is created successfully.

The second test case verifies that the form is initialized correctly with empty values.

The third test case simulates a login action, mocking the AuthService and verifying 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();
  });
});
```

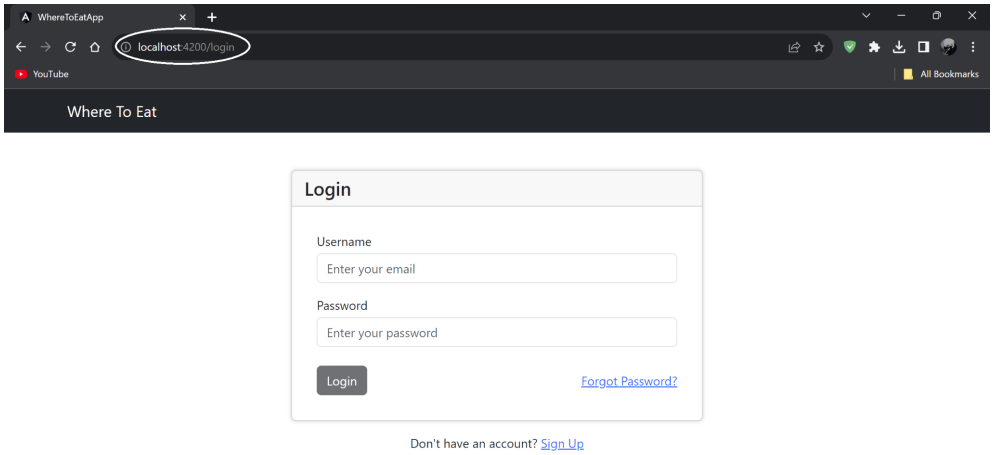
In the Sign up page we have written test cases to check whether the user is able to login with valid credentials and whether the data entered by the user is valid or not. It also checks for the filled in data in order to confirm

that no field is left blank. We also checked the user sign up by entering invalid data. And also tried to register with already used data which it should not allow to register the user again.

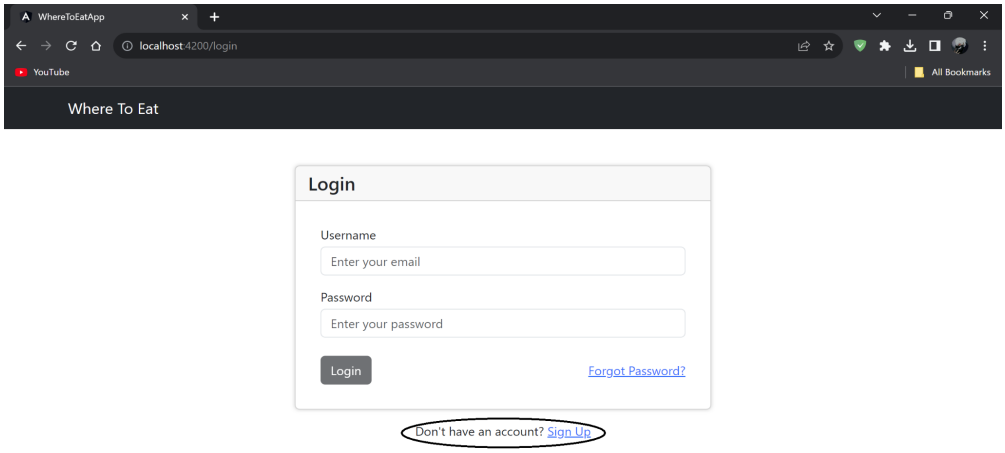
User manual:

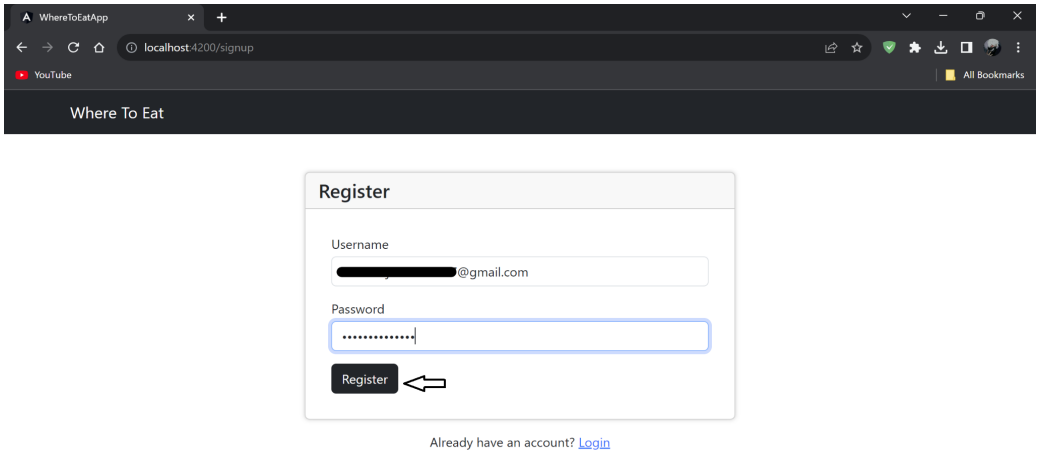
Here is a step-by-step follow-up on accessing our application WHERE TO EAT.

Step 1: Once the user accesses the page using the URL, the User gets redirected to the landing page.

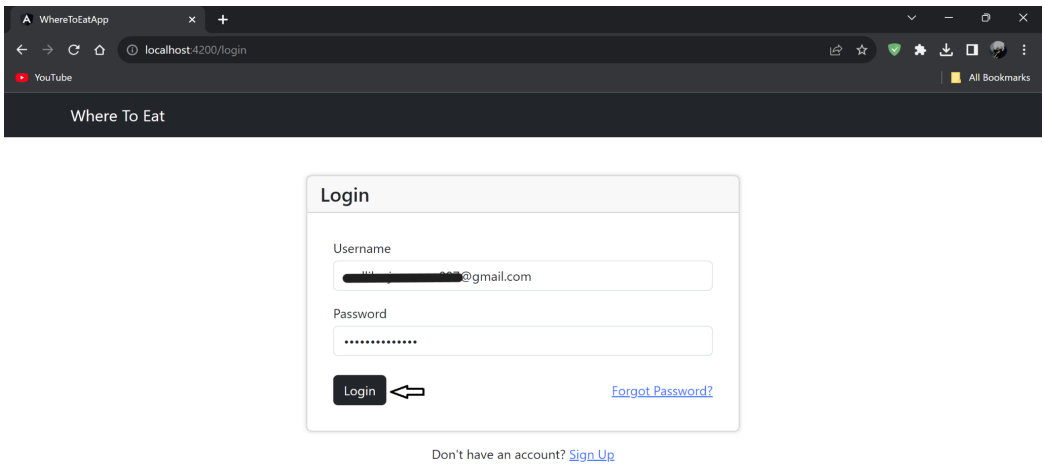


Step 2: If the client is a new user he has the facility to Sign Up.

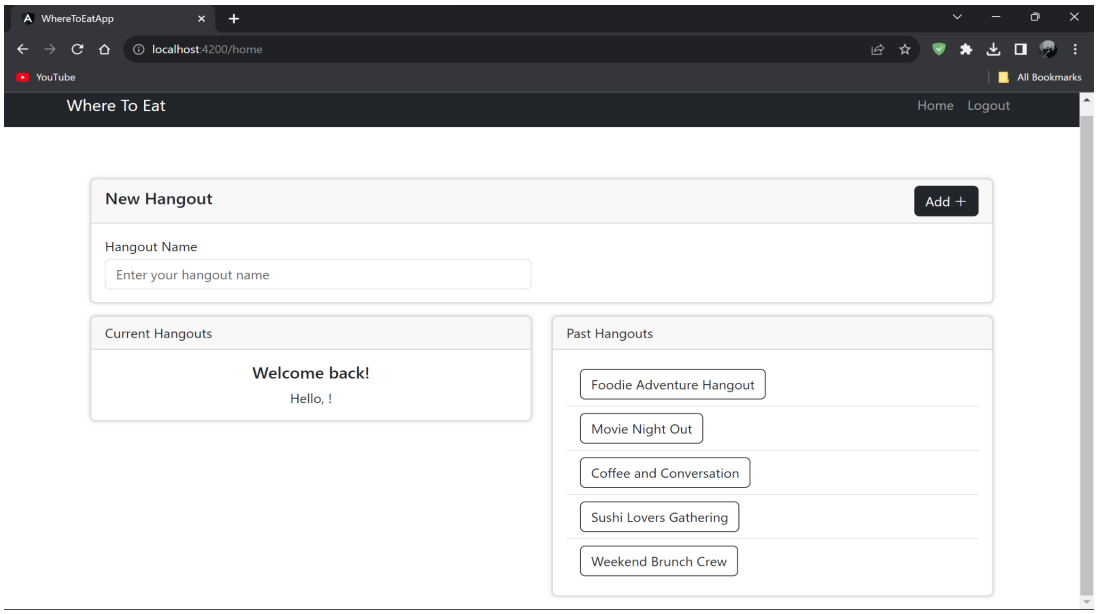




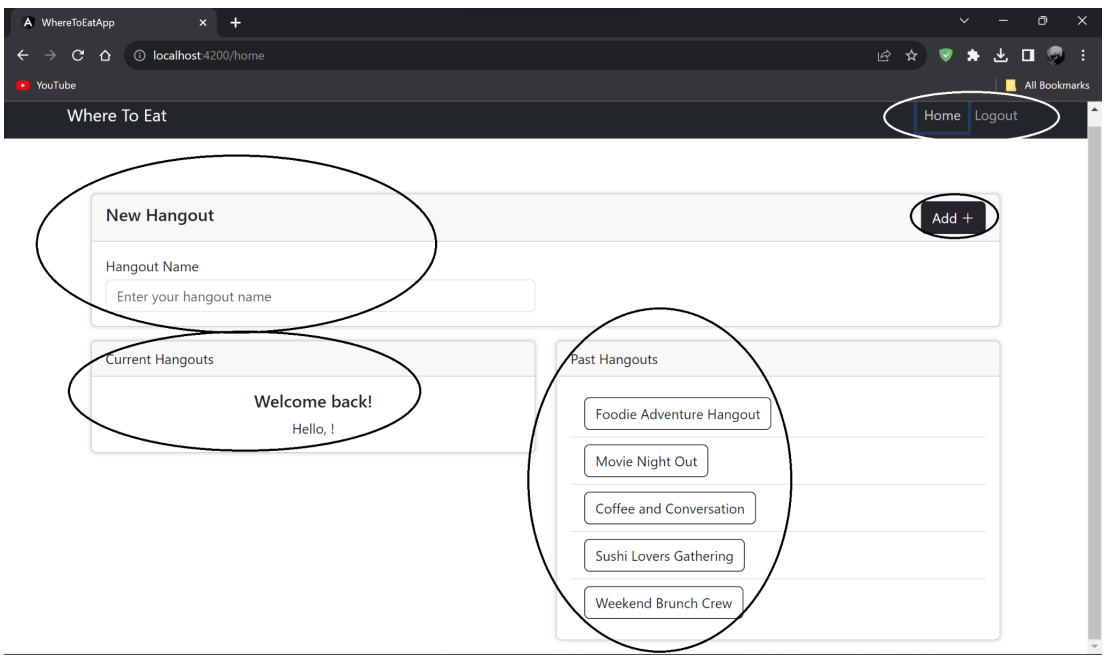
Step 3: Once the user gets to register he is redirected to the Login page where he gets to enter the same credentials to Login.



Step 4: 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.
- 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 **Hangout Name** for your hangout so that your group can join.

- Current hangouts are displayed in the **Current Hangouts** section, where all the current sessions are displayed. Users can hop from hangout to hangout.
- Our application also has a **Past Hangouts** where users can filter the hangouts category wise i.e (Foodie Adventure Hangout, Movie Night Out, Coffee and Conversation, etc.)

Compilation and Run Instructions:

Prerequisites:

- Make sure your development environment has the necessary tools and applications installed.
- a web browser to see the webpage.
- For server - side scripting, use npm and node.js.
- For storage and authentication, use Firebase.
- Key to the Google Maps API(if needed).
- Any more particular requirements that your project may need.

Setup:

- 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>"
};
```

- 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.

Environment.ts

```
export const environment = {
 production: false,
};
```

It specifies an environment configuration object for a non-production (usually development) environment with the production parameter set to false.

Feedback - Peer Review:

During the peer review session, the project received valuable feedback in several areas.

Documentation Clarity:

Feedback indicated that the documentation needed to be clearer and more comprehensive. In response, the team worked on improving the user documentation, ensuring that it was easy to understand and covered all essential aspects of the application.

Testing and Quality Assurance:

Feedback highlighted the importance of testing. As a result, the team implemented few test cases to identify and address any bugs or issues.

Security and Privacy:

The team made sure that Firebase Authentication was configured to the latest best practices to protect user data effectively.

Scalability:

Some feedback addressed the application's scalability, suggesting that it should be designed to handle potential growth in the user base. In response, the team made architectural changes and ensured that the database and infrastructure could scale as needed.

Feedback Implementation Tracking:

A systematic approach to tracking and documenting feedback and its subsequent actions. The team agreed to ensure that all suggestions were addressed and resolved effectively.

Accomplishments:

Where To Eat project has achieved several milestones and key features. Here's a brief reflection of what has been accomplished, what went well, and areas that could be improved:

Accomplishments and What Went Well:

Key Features Implementation: We have successfully implemented key features such as user authentication, hangout creation, restaurant addition, voting, and data management using Firebase and Google Places API.

User-Friendly Interface: Our project focused on creating a user-friendly interface, making it easier for users to navigate, join hangouts, vote for restaurants, and manage their profiles.

Data Management: Firestore was used to efficiently manage data, providing a structured approach to store hangout details, user information, and restaurant data.

Planning and Documentation: We have a well-defined planning phase with clear timelines and objectives. The insertion of user documentation and a code repository on GitHub demonstrates our commitment to transparency and future maintenance.

Technological Stack: Our choice of technology stack, including Angular, Firebase, and Google Places API, was well-suited for the project's requirements and helped us in achieving the desired functionality throughout our project.

Security: Secure user authentication was implemented using Firebase Authentication, ensuring the protection of user data.

Areas That Could Be Improved:

Testing: While our project uses basic functionality for testing and bug fixes, the comprehensive testing, such as unit testing and integration testing, should be more focused on during the development process.

Scalability: As our project grows, there may be considerations for scalability in terms of accommodating a larger user base, as well as managing increased data loads.

Mobile Responsiveness: Given that it's a web application, ensuring mobile responsiveness is important to accommodate users on various devices.

User Feedback and Iteration: Continuous feedback from users and development can help us in refining the application and adding features based on user needs.

By focusing on areas of improvement such as testing, user feedback, scalability, and mobile responsiveness our project can further enhance its value to users and ensure its long-term success.

Project Timeline

Here, we used the - **Gantt chart**: Gantt chart

Members Contribution Table

Member name	Contribution description	Overall Contribution (%)	Note(if applicable)
Ibrahim Alkuwaifi	I implemented firebase authentication (registration, login, forgot password, and email verification). I also implemented hangout creation. I also worked on the report.	80%	
Mohammad Khaja Moinuddin	I carried out Core Functionalities of Phase 1 (created an inspection code document), Updating the project timeline via Gantt chart and Uploading files to Github.	10%	
Teja Peddi	I completed the UML design for phase-1	10%	
Mohammed Abdul Wahid	I gave a brief reflection on what has been accomplished, what went well and could be improved.	10%	
Gouthami Gongidi	I have provided a compilation and run instructions that tells how to compile and perform instructions	10%	
Mallikarjun Narra	I have provided a user manual that tells us how to install/use your program	10%	

Ashesh kumar Pinisetti	I gave a brief description of peer review feedback and updated the folder directory structure and worked on this report.	10%	