Core Functionalities
Phase-1

Project Name: Where to Eat
Deliverable: 3
Date: October 23, 2023

Introduction

This document outlines the review of the codebase for the Where to Eat App. The purpose of this inspection is to ensure the code adheres to coding standards, is free from critical issues, and follows best practices.

Code Location

The code is hosted on GitHub at: https://github.com/brofcb/software-engineering , and the inspection covers the following files:

- Forgot-password-page
- Home-page
- Login-page
- Signup-page
- Services
- Auth files and more.

1: forgot-password-page

Location: WhereToEatApp/src/app/pages/forgot-password-page

```
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="card">
        <h4 class="card-header">Forgot Password</h4>
        <div class="card-body">
          <app-loading-spinner *ngIf="loading"></app-loading-spinner>
          <form *ngIf="!loading" [formGroup]="passwordForm" (ngSubmit)="resetPassword()">
```

```
          <div class="form-group m-3">
            <label for="username" class="mb-1">Email</label>
            <input placeholder="Enter your email" type="email" formControlName="username"
class="form-control"/>
            <div *ngIf="f.username.touched && f.username?.invalid" class="text-danger">
               <div *ngIf="f?.username?.errors?.required">Email is required</div>
               <div *ngIf="f?.username?.errors?.email">Invalid email format</div>
            </div>
          </div>
          <button type="submit" class="btn btn-dark btn-block mx-3 mb-3">Reset Password</button>
        </form>
       </div>
     </div>
     <p class="text-center mt-3">Remember your password? <a routerLink="/login">Login</a></p>
   </div>
  </div>
</div>

import { Component } from '@angular/core';
import { FormGroup, UntypedFormBuilder, Validators } from '@angular/forms';
import { AuthService } from 'src/app/auth/auth.service';

@Component({
  selector: 'app-forgot-password-page',
  templateUrl: './forgot-password-page.component.html',
  styleUrls: ['./forgot-password-page.component.scss']
})
export class ForgotPasswordPageComponent {
  passwordForm: FormGroup;
  loading = false;
  constructor(
    public authService: AuthService,
    private formBuilder: UntypedFormBuilder,
  ) { }

  ngOnInit() {
```

```
    this.passwordForm = this.formBuilder.group({
      username: ['', [Validators.required, Validators.email]],
    });
  }
  get f() { return this.passwordForm?.controls; }

  resetPassword(){
    this.loading = true;
    this.authService.ForgotPassword(this.f?.['username']?.value)
    .finally(() =>{
      this.loading = false;
    });
  }
}
```

2: Home Page

Location: WhereToEatApp/src/app/pages/home-page

```
<div class="row justify-content-center">
    <div class="col">
        <div class="row m-3">
            <div class="col">
                <div class="card">
                    <div class="card-header">
                        <div class="d-flex align-items-center justify-content-between">
                            <h5>New Hangout</h5>
                            <button class="btn btn-dark float-right" [disabled]="!this.hangoutForm.valid"
(click)="createNewHangout()">Add <i class="bi bi-plus-lg"></i></button>
                        </div>
                    </div>
                    <div class="card-body ">
                        <form *ngIf="!loading" [formGroup]="hangoutForm">
                            <div class="row m-3">
                                <div class="col-6">
                                    <label for="name" class="mb-1">Name</label>
```

```
                        <input placeholder="Enter your hangout name" type="text" formControlName="name"
class="form-control"/>
                    </div>
                    <div class="col-6">
                        <label for="location" class="mb-1">Location</label>
                        <input type="text" placeholder="Enter location of the hangout"
formControlName="location" class="form-control"/>
                    </div>
                    <div class="col-12 mt-2">
                        <label for="description" class="mb-1">Description</label>
                        <textarea type="text" formControlName="description" class="form-control"> </textarea>
                    </div>
                    <div class="col-6 m-3">
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>
<div class="row m-3">
    <div class="col-md-6">
        <div class="card">
            <div class="card-header">
                Current Hangouts
            </div>
            <div class="card-body">
                <div class="text-center">
                    <p  *ngIf="!(activeHangouts$|async)?.length">No past hangouts</p>
                </div>
                <ul class="list-group list-group-flush" *ngIf="(activeHangouts$|async)?.length">
                    <li *ngFor="let hangout of (activeHangouts$|async)" class="list-group-item">
                      <a href="/" class="btn btn-outline-dark btn-block">{{hangout?.name}}</a>
                    </li>
                </ul>
            </div>
```

```
          </div>
        </div>
        <div class="col-md-6">
          <div class="card">
            <div class="card-header">
              Past Hangouts
            </div>
            <div class="card-body">
              <div class="text-center">
                <p  *ngIf="!(pastHangouts$|async)?.length">No past hangouts</p>
              </div>
              <ul class="list-group list-group-flush" *ngIf="(pastHangouts$|async)?.length">
                <li *ngFor="let hangout of (pastHangouts$|async)" class="list-group-item">
                  <a href="/" class="btn btn-outline-dark btn-block">{{hangout?.name}}</a>
                </li>
              </ul>
            </div>
          </div>
        </div>
      </div>
    </div>
</div>
```

Home-page.component.ts:

```
import { Component } from '@angular/core';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { FormGroup, UntypedFormBuilder, Validators } from '@angular/forms';
import { Observable } from 'rxjs';
import { AuthService } from 'src/app/auth/auth.service';
import { Hangout } from 'src/app/models/hangout.model';
import { HangoutService } from 'src/app/services/hangout.service';

@Component({
  selector: 'app-home-page',
  templateUrl: './home-page.component.html',
```

```
  styleUrls: ['./home-page.component.scss']
})
export class HomePageComponent {

  hangoutForm: FormGroup;
  loading = false;
  activeHangouts$: Observable<Hangout[]>
  pastHangouts$: Observable<Hangout[]>
  constructor(public afAuth: AngularFireAuth,
              public authService: AuthService,
              private formBuilder: UntypedFormBuilder,
              private hangoutService: HangoutService){}

  ngOnInit(): void {
    this.hangoutForm = this.formBuilder.group({
      name: ['', Validators.required],
      location: ['', Validators.required],
      description: ['', Validators.required]
    });

    this.authService.user$.subscribe(user => {
      this.activeHangouts$ = this.hangoutService.getActiveHangoutsForUser(user?.uid);
      this.pastHangouts$ = this.hangoutService.getPastHangoutsForUser(user?.uid);
    });
  }
  createNewHangout(){
    console.log('Form values:', this.hangoutForm.value);
    if (this.hangoutForm.valid) {
      this.loading = true;
      const hangout = {
        ...this.hangoutForm?.value,
        createdBy: this.authService?.userData?.uid,
        active: true,
        participants: [this.authService?.userData?.uid],
        restaurants: []
      } as Hangout;
```

```
    this.hangoutService.createHangout(hangout)
    this.loading = false;
   }
  }
}
```

3: Login Page

For logging in the application

Location: WhereToEatApp/src/app/pages/login-page

```
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="card">
        <h4 class="card-header">Login</h4>
        <div class="card-body">
          <app-loading-spinner *ngIf="loading"></app-loading-spinner>
          <form *ngIf="!loading" [formGroup]="loginForm" (ngSubmit)="login()">
            <div class="form-group m-3">
              <label for="username" class="mb-1">Username</label>
              <input placeholder="Enter your email" type="text" formControlName="username"
class="form-control"/>
              <div *ngIf="f.username.touched && f.username?.invalid" class="text-danger">
                <div *ngIf="f?.username?.errors?.required">Username is required</div>
                <div *ngIf="f?.username?.errors?.email">Invalid email format</div>
              </div>
            </div>
            <div class="form-group mt-3 mx-3 mb-2">
              <label for="password" class="mb-1">Password</label>
              <input placeholder="Enter your password" type="password" formControlName="password"
class="form-control"/>
              <div *ngIf="f.password.touched && f.password?.invalid" class="text-danger">
                <div *ngIf="f?.password?.errors?.required">Password is required</div>
              </div>
            </div>
```

```
            <div class="d-flex justify-content-between align-items-center">
              <div class="form-group m-3">
                <button [disabled]="!loginForm.valid" class="btn btn-dark">
                    <span *ngIf="loading" class="spinner-border spinner-border-sm mr-1"></span>
                    Login
                </button>
              </div>

              <a routerLink="/forgot-password" class="forgot-password-link m-3">Forgot Password?</a>
            </div>
          </form>
        </div>
      </div>
      <p class="text-center mt-3">Don't have an account? <a routerLink="/signup">Sign Up</a></p>
    </div>
  </div>
</div>
```

Login.component.ts

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup, UntypedFormBuilder, Validators } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { AuthService } from 'src/app/auth/auth.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginPageComponent {


  loginForm: FormGroup;
  loading = false;
```

```
  constructor(
    private authService: AuthService,
    private formBuilder: UntypedFormBuilder,
  ) { }

  ngOnInit() {
    this.loginForm = this.formBuilder.group({
        username: ['', [Validators.required, Validators.email]],
        password: ['', Validators.required]
    });
  }

  get f() { return this.loginForm?.controls; }

  login() {
    this.loading = true;
    this.authService.SignIn(this.f?.['username']?.value, this.f?.['password']?.value)
      .finally(() => {
        this.loading = false;
      });
  }
}
```

4. Sign Up

Used for signing up for the application.

Location: WhereToEatApp/src/app/pages/signup-page

```
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-md-6">
      <div class="card">
        <h4 class="card-header">Register</h4>
        <div class="card-body">
          <app-loading-spinner *ngIf="loading"></app-loading-spinner>
          <form [formGroup]="registerForm" *ngIf="!loading" (ngSubmit)="register()">
```

```
          <div class="form-group m-3">
            <label for="username" class="mb-1">Username</label>
            <input placeholder="Enter your email" type="text" formControlName="username"
class="form-control"/>
            <div *ngIf="f.username.touched && f.username?.invalid" class="text-danger">
              <div *ngIf="f?.username?.errors?.required">Username is required</div>
              <div *ngIf="f?.username?.errors?.email">Invalid email format</div>
            </div>
          </div>
          <div class="form-group mt-3 mx-3 mb-2">
            <label for="password" class="mb-1">Password</label>
            <input placeholder="Enter your password" type="password" formControlName="password"
class="form-control"/>
            <div *ngIf="f.password.touched && f.password?.invalid" class="text-danger">
              <div *ngIf="f?.password?.errors?.required">Password is required</div>
              <div *ngIf="f?.password?.errors?.minlength">Password must be at least 8 characters long</div>
            </div>
          </div>
          <div class="form-group m-3">
           <button [disabled]="!registerForm.valid" class="btn btn-dark">
              <span *ngIf="loading" class="spinner-border spinner-border-sm mr-1"></span>
              Register
           </button>
          </div>
        </form>
       </div>
     </div>
     <p class="text-center mt-3">Already have an account? <a routerLink="/login">Login</a></p>
   </div>
  </div>
</div>
```

Signup-page.component.ts

```
import { Component } from '@angular/core';
import { FormGroup, UntypedFormBuilder, Validators } from '@angular/forms';
```

```
import { Router } from '@angular/router';
import { AuthService } from 'src/app/auth/auth.service';

@Component({
  selector: 'app-signup-page',
  templateUrl: './signup-page.component.html',
  styleUrls: ['./signup-page.component.scss']
})
export class SignupPageComponent {
  registerForm: FormGroup;
  loading = false;

  constructor(
    private authService: AuthService,
    private formBuilder: UntypedFormBuilder,
  ) { }
  ngOnInit() {
    this.registerForm = this.formBuilder.group({
      username: ['', [Validators.required, Validators.email]],
      password: ['', [Validators.required, Validators.minLength(8)]]
  });
  }

  get f() { return this.registerForm?.controls; }
  register(){
    this.loading = true;
    this.authService.SignUp(this.f?.['username']?.value, this.f?.['password']?.value)
      .finally(() => {
        this.loading = false;
      });
  }
}
```

5: Hangout service

Location: WhereToEatApp/src/app/services/hangout.service.ts

```
import { Injectable } from '@angular/core';
import { AngularFirestore } from '@angular/fire/compat/firestore';
import { Hangout } from '../models/hangout.model';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class HangoutService {

  constructor(private firestore: AngularFirestore,) { }

  // Create a new hangout
  public async createHangout(hangout: Hangout) {

    try {
      await this.firestore.collection('hangouts').add(hangout);
    } catch (error) {
      console.error('Error creating hangout:', error);
    }
  }

  // Generate a unique join ID for hangout
  generateJoinId() {
    // This is a simplistic implementation; consider a more robust solution
    return Math.random().toString(36).substr(2, 8).toUpperCase();
  }

  // Add a restaurant to a hangout
  async addRestaurant(hangoutId: string, restaurantId: string) {
    const hangoutRef = this.firestore.collection('hangouts').doc(hangoutId);
```

```
  try {
    await hangoutRef.update({
      // restaurants: this.firestore.firestore.FieldValue.arrayUnion(restaurantId)
    });
  } catch (error) {
    console.error('Error adding restaurant:', error);
  }
}

// Join a hangout
async joinHangout(joinId: string, userId: string) {
  const hangoutRef = this.firestore.collection('hangouts').doc(joinId);

  try {
    await hangoutRef.update({
      // participants: this.firestore.firestore.FieldValue.arrayUnion(userId)
    });
  } catch (error) {
    console.error('Error joining hangout:', error);
  }
}

// Cast or update a vote
async castVote(userId: string, hangoutId: string, restaurantId: string) {
  const voteRef = this.firestore.collection('votes').doc(`${userId}_${hangoutId}`);

  try {
    const voteSnapshot = await voteRef.get().toPromise();

    if (voteSnapshot?.exists) {
      // Update existing vote
      await voteRef.update({ restaurantId });
    } else {
      // Cast new vote
      const voteData = { userId, hangoutId, restaurantId };
```

```
        await voteRef.set(voteData);
      }
    } catch (error) {
      console.error('Error casting vote:', error);
    }
  }

  getActiveHangoutsForUser(userId: string | undefined): Observable<Hangout[]> {
    return this.firestore.collection<Hangout>('hangouts', ref =>
      ref.where('active', '==', true).where('participants', 'array-contains', userId)
    ).valueChanges();
  }

  getPastHangoutsForUser(userId: string | undefined): Observable<Hangout[]> {
    return this.firestore.collection<Hangout>('hangouts', ref =>
      ref.where('active', '==', false).where('participants', 'array-contains', userId)
    ).valueChanges();
  }
}
```

6: Auth-guard

It is used to check if a user is logged in and, if not, redirect them to the login page.

Location: WhereToEatApp/src/app/auth/auth-guard.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router } from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from './auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard  {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(
```

```
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    if (this.authService.isLoggedIn !== true) {
      this.router.navigate(['login']);
      return false;
    }
    return true;
  }
}
```

7: Auth-service

AuthService will take the users to the page they intended to access before login

Location: WhereToEatApp/src/app/auth/auth.service.ts

```
import { Injectable, NgZone } from '@angular/core';
import {
  AngularFirestore,
  AngularFirestoreDocument,
} from '@angular/fire/compat/firestore';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { Router } from '@angular/router';
import { User } from '../models/user.model';
import { NotificationService } from '../core/services/notification.service';
import { Observable, Subscription, of, switchMap } from 'rxjs';
import { FirebaseError } from 'firebase/app';

@Injectable({
    providedIn: 'root'
  })
  export class AuthService {

    userData: any; // Save logged in user data
    user$: Observable<User| null | undefined>;
    authStateSubscription: Subscription | undefined;
```

```
constructor(
  public afs: AngularFirestore, // Inject Firestore service
  public afAuth: AngularFireAuth, // Inject Firebase auth service
  public router: Router,
  public ngZone: NgZone,
  public notificationService: NotificationService
) {

  this.afAuth.authState.subscribe((user) => {
    if (user) {
      this.userData = user;
      localStorage.setItem('user', JSON.stringify(this.userData));
      JSON.parse(localStorage.getItem('user')!);
    } else {
      localStorage.setItem('user', 'null');
      JSON.parse(localStorage.getItem('user')!);
    }
  });

  this.user$ = this.afAuth.authState.pipe(
    switchMap(user => {
      if(user) {
        return this.afs.doc<User>(`users/${user.uid}`).valueChanges();
      }
      else{
        return of(null);
      }
    })
  );
}

// Sign in with email/password
SignIn(email: string, password: string) {
  return this.afAuth
    .signInWithEmailAndPassword(email, password)
    .then((result) => {
```

```
        if(!result.user?.emailVerified){
          const customError = new FirebaseError('auth/user-disabled', 'This account has not been verified.');
          throw customError;
        }
        // Unsubscribe from previous authState subscription
        if (this.authStateSubscription) {
          this.authStateSubscription.unsubscribe();
        }
        // Subscribe to authState once and handle login logic
        this.authStateSubscription = this.afAuth.authState.subscribe((user) => {
          if (user) {
            this.notificationService.showNotification('success', 'Welcome back!');
            this.router.navigate(['home']);
          }
        });
      })
      .catch((error) => {
        const errorMessage = this.getErrorMessageForCode(error.code);
        this.SignOut();
        this.notificationService.showNotification('error', errorMessage);
      });
  }
  // Sign up with email/password
  SignUp(email: string, password: string) {
    return this.afAuth
      .createUserWithEmailAndPassword(email, password)
      .then((result) => {
        if (this.authStateSubscription) {
          this.authStateSubscription.unsubscribe();
        }
        this.notificationService.showNotification('success', 'Account registered successfully. Check your inbox to
verify your email.');
        this.SendVerificationEmail();
        this.SetUserData(result.user);
      })
      .catch((error) => {
```

```
      const errorMessage = this.getErrorMessageForCode(error.code);
      this.notificationService.showNotification('error', errorMessage);
    });
}
// Send email verification when new user sign up
SendVerificationEmail() {
  return this.afAuth.currentUser
    .then((u: any) => u.sendEmailVerification())
    .then(() => {
      this.router.navigate(['verify-email-address']);
    });
}
// Forgot password
ForgotPassword(passwordResetEmail: string) {
  return this.afAuth
    .sendPasswordResetEmail(passwordResetEmail)
    .then(() => {
      this.notificationService.showNotification('success', 'Password reset email sent, check your inbox.');
    })
    .catch((error) => {
      this.notificationService.showNotification('error', error?.message);
    });
}
// Returns true when user is looged in and email is verified
get isLoggedIn(): boolean {
  const user = JSON.parse(localStorage.getItem('user')!);
  return user !== null && user.emailVerified !== false ? true : false;
}
/* Setting up user data when sign in with username/password,
sign up with username/password and sign in with social auth
provider in Firestore database using AngularFirestore + AngularFirestoreDocument service */
SetUserData(user: any) {
  const userRef: AngularFirestoreDocument<any> = this.afs.doc(
    `users/${user.uid}`
  );
  const userData: User = {
```

```
    uid: user.uid,
    email: user.email,
    displayName: user.displayName,
    photoURL: user.photoURL,
    emailVerified: user.emailVerified,
  };
  return userRef.set(userData, {
    merge: true,
  });
}
// Sign out
SignOut() {
  return this.afAuth.signOut().then(() => {
    localStorage.removeItem('user');
    this.router.navigate(['sign-in']);
  });
}

private getErrorMessageForCode(errorCode: string): string {
  switch (errorCode) {
    case 'auth/invalid-email':
      return 'Invalid email address.';
    case 'auth/user-disabled':
      return 'This account has not been verified.';
    case 'auth/user-not-found':
      return 'User not found. Please check your email and password.';
    case 'auth/wrong-password':
      return 'Incorrect password. Please try again.';
    case 'auth/weak-password':
      return 'Password is too weak. Please choose a stronger password.';
    case 'auth/email-already-in-use':
      return 'The email address is already in use by another account.'
    // Add more cases for other error codes as needed
    default:
      return 'An error occurred. Please try again later.';
  }
```

```
  }
 }
```

8: Auto login:

It allows users to automatically log in to an application without having to enter their username and password

Location: WhereToEatApp/src/app/auth/auto-login.ts

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router } from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from './auth.service';

@Injectable({
  providedIn: 'root'
})
export class AutoLogin  {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    if (this.authService.isLoggedIn) {
      this.router.navigate(['home']);
      return false;
    }
    return true;
  }
}
```