Lab 3 Wiki

CSEE 5590

Assignment: Lab 3

Team Details:

Name: Duy Hoang Ho Class ID: 7

Name: Tonderai Kambarami Class ID: 8

Name: Cameron Knight Class ID: 12

Name: Evan Wike Class ID: 32

Projects:

User Management Application (Angular)

User Management Appplication (Mobile)

Demo Video:

Languages/Frameworks

- HTML, CSS, SCSS, JavaScript
- MongoDB, Express.js, Angular2 (TypeScript), Node.js (MEAN Stack)

- Material2

User Management Application (MEAN Stack)

I. Introduction

The entry point to our web app is a login screen. This screen contains a simple form, containing fields for the user's username and password. Once the form is filled out with the user's account details, they can then hit the 'Login' button which will submit the form to the backend, where it is then authenticated. If the authentication is successful, the user will then be taken to their account page. If the authentication fails, the user will not be able to login. If the user hasn't registered an account yet, they can click the 'Register' button, where they will be taken to the user registration page.

The user registration page contains another form, with fields for their username, email, first name, last name, password, an image URL for their profile picture, and a description of the user. All of the fields are subject to some form of validation. For example, the email field will check to make sure the specified email is a valid email address, and the username and password fields will ensure that the user's username and password are at least 5 characters in length. All of the fields are required. The last step in the validation process is to check to make sure that the specified email is not already registered to another user in the database. Invalid fields will display an error message below the field in question. Once all the fields are valid, the user may proceed to register their account. Once registered, their account details are hashed and stored in the database, and the user is sent to their account page.

On the account page, users can see their account information and edit it if they wish by clicking the edit button. The edit button will take the user to an edit screen. The edit screen contains a pre-populated form with their account details. Here, users can change their account information and update the database, or cancel.

Both the account and edit pages are exclusive to logged in users. All user information is stored securely in our MongoDB database. The backend portion of this project was implemented as a separate project, and handles all requests from the client.

The final component in our app is the navigation bar. The navigation bar allows users to navigate to different pages and behaves differently depending on whether or not the user is currently logged in. For example, if the user is logged in, it will contain a 'Logout'

button and a link to the user's profile page - neither of which will not render if the user is not logged in.

II. Objective

We were tasked with developing a User Management Application with the following characteristics:

- Allow users to register
- Allow users to save their data
- Allow returning users to login and logout
- Keep a logged in user's session alive between page visits
- Create exclusive pages for logged in users
- A UI that responds differently depending on whether or not the user is logged in
- Use the MEAN stack

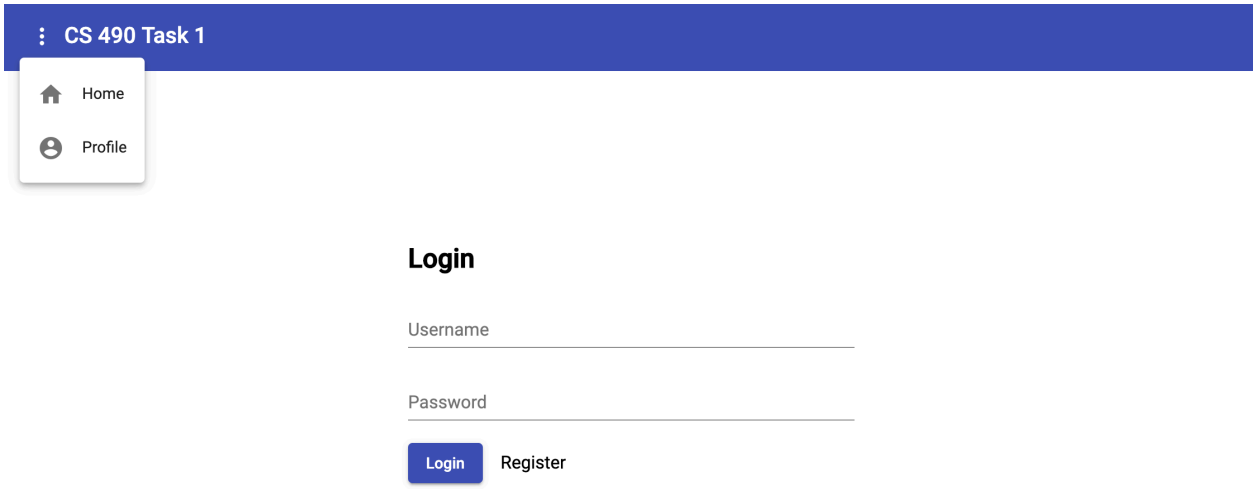
III. Approach

We separated this task into two separate projects, the client and the backend.

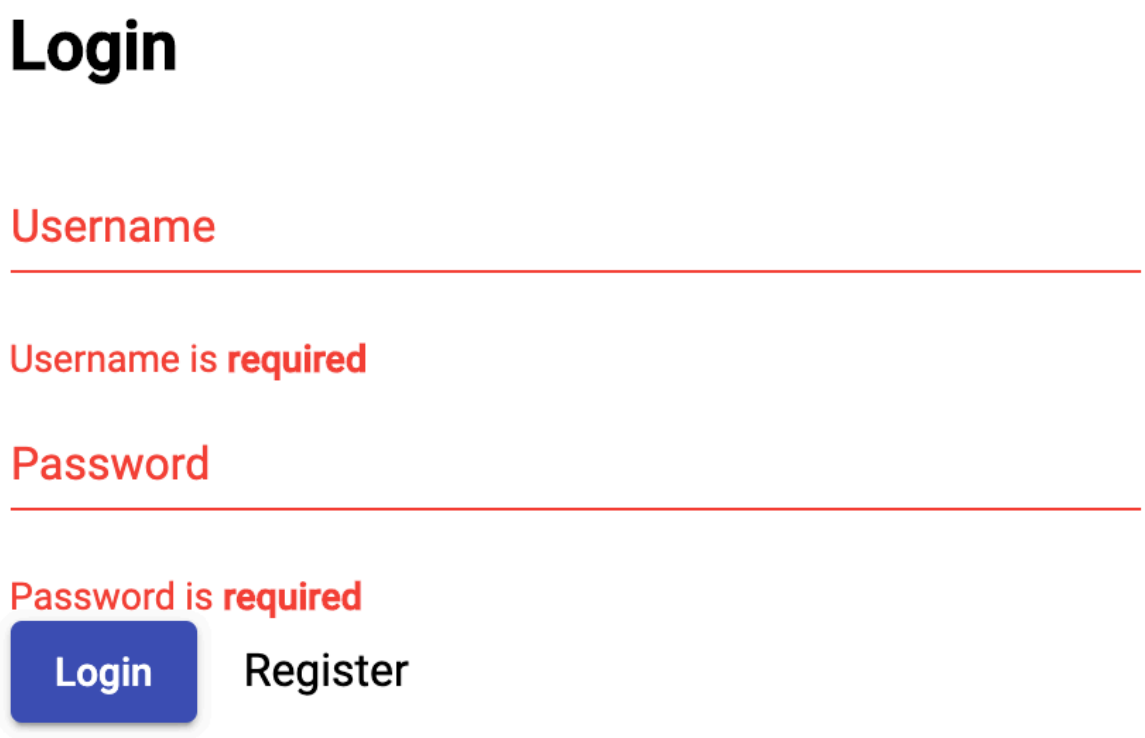
For the client, we chose to use Material2, Angular's component infrastructure utilizing Google's Material Design standards. This made developing the front end significantly faster. Next, we decided on what components we would need for the client, and drew up a flowchart detailing how all of the components would interact with one another and the backend. We also decided to use Angular Router to allow for multiple pages. This, along with the form validation, was probably the hardest part of the project. This is because, with routing, you have to create a service to allow components to share data between one another.

For the backend, we created an Express app running on NodeJS with Mongoose used to interface with MongoDB. The backend uses JSON Web Tokens (JWT) to manage user sessions, and Passport with the passport-local strategy for user authentication.

IV. Screenshots



Login screen, showing navigation for users before they login.



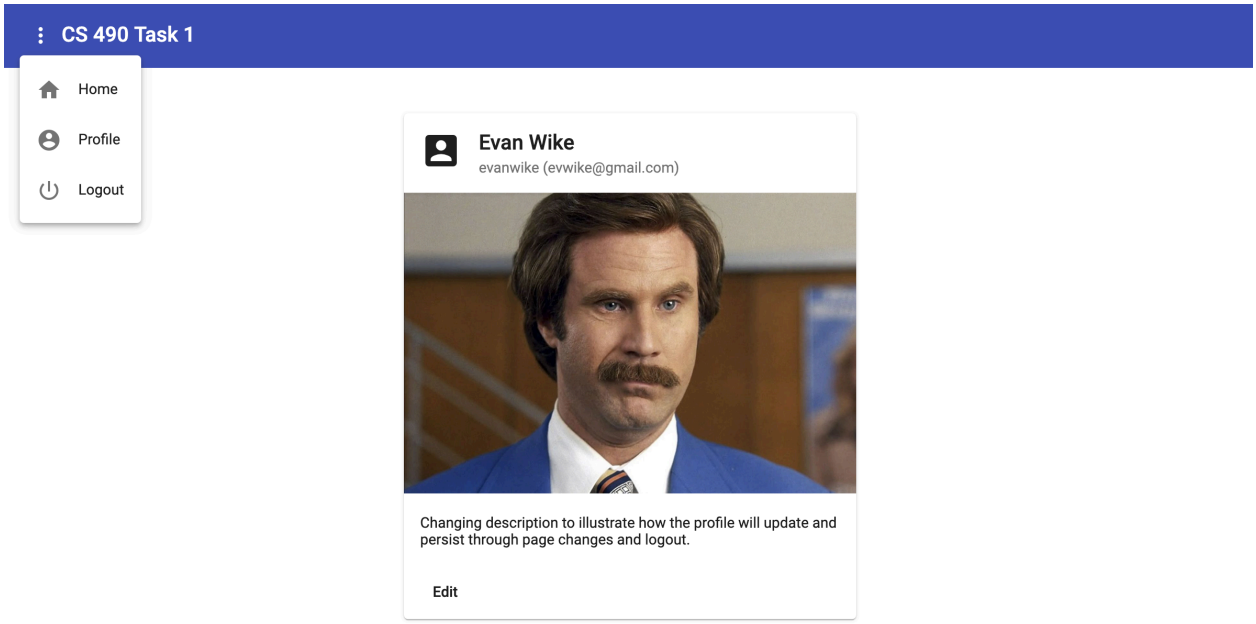
Login validation

```
<div class="wrap center">
  <h2>Edit Profile</h2>

  <form [formGroup]="profileForm" (ngSubmit)="update()">
    <div class="form-group">
      <mat-form-field floatLabel="auto">
        <input matInput id="firstName" type="text" formControlName="firstName" class="form-control"
               placeholder="First Name">
      </mat-form-field>
    </div>
    <div class="form-group">
      <mat-form-field floatLabel="auto">
        <input matInput id="lastName" type="text" formControlName="lastName" class="form-control"
               placeholder="Last Name">
      </mat-form-field>
    </div>
    <div class="form-group">
      <mat-form-field floatLabel="auto">
        <input matInput id="imageUrl" type="imageUrl" formControlName="imageUrl" class="form-control"
               placeholder="Image URL">
      </mat-form-field>
    </div>
    <div class="form-group">
      <mat-form-field floatLabel="auto">
        <textarea matInput id="description" type="description" formControlName="description" class="form-control"
                  placeholder="Description"></textarea>
      </mat-form-field>
    </div>
    <div class="form-group">
      <button mat-raised-button color="primary" type="submit">Update</button>
      <span><a routerLink="/profile">Cancel</a></span>
    </div>
  </form>
</div>
```

HTML for the Login component

```typescript
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  public loginForm: FormGroup;
  submitted = false;

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

  ngOnInit() {
    this.loginForm = this.fb.group( controlsConfig: {
      username: ['', [Validators.required, Validators.minLength( minLength: 4)]],
      password: ['', [Validators.required, Validators.minLength( minLength: 4)]]
    });
  }

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

  login() {
    this.submitted = true;

    if (this.loginForm.invalid) {
      return;
    }

    const credentials = this.loginForm.value;
    console.log('credentials ', credentials);
    this.userService.login(credentials).subscribe(
      next: data => this.router.navigateByUrl( url: '/'),
      error: err => {
        console.log('Login Error');
        console.error(err);
      }
    );
  }

}
```

TypeScript for the Login component



The navigation bar for logged in users

Sign Up

Username

Evan

Username must be **at least 5 characters in length**

First Name

Evan

Last Name

Wike

Email

abc.com

Password

•••

Password must be **at least 5 characters in length**

Image URL

Description

Sign Up

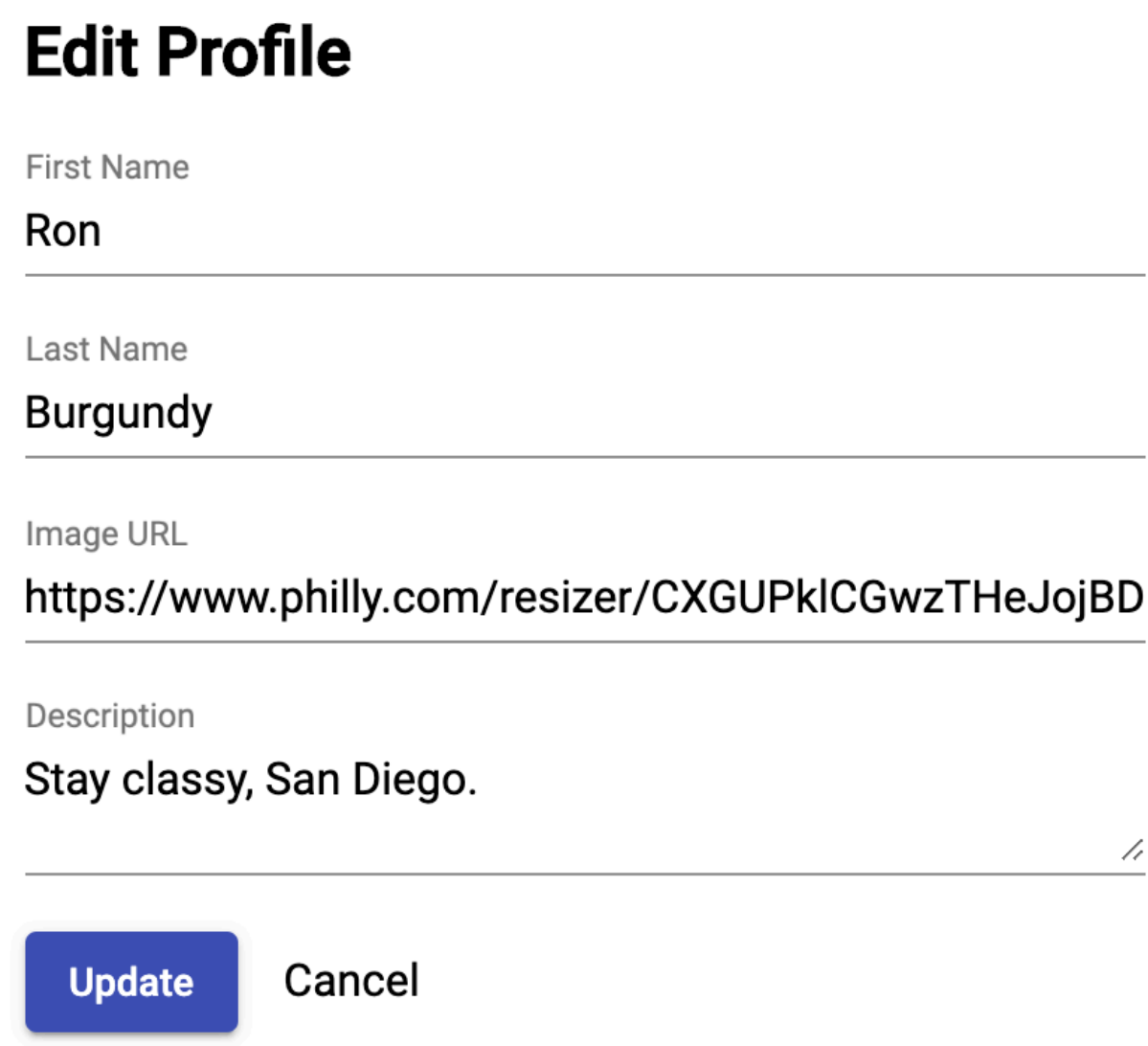
The register component with validation. Displayed when the users click the register button on the login screen

HTML for the register component

```typescript
@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
  public registerForm: FormGroup;
  submitted = false;

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

  ngOnInit() {
    this.registerForm = this.fb.group( controlsConfig: {
      username: ['', [Validators.required, Validators.minLength( minLength: 5)]],
      firstName: ['', [Validators.required]],
      lastName: ['', [Validators.required]],
      email: ['', [Validators.required, Validators.email]],
      password: ['', [Validators.required, Validators.minLength( minLength: 5)]],
      description: ['', [Validators.required]],
      imageUrl: ['', [Validators.required]]
    });
  }

  get f() { return this.registerForm.controls; }

  register() {
    this.submitted = true;

    if (this.registerForm.invalid) {
      return;
    }

    const user = this.registerForm.value;
    console.log('user', user);
    this.userService.register( user: {user}).subscribe(
      next: data => this.router.navigateByUrl( url: '/'),
      error: err => {
        console.error(err);
      }
    );
  }
```

TypeScript for the register component

The user profile component after login



The edit profile component, populated with user's current account information

Edit Profile

First Name

Evan

Last Name

Wike

Image URL

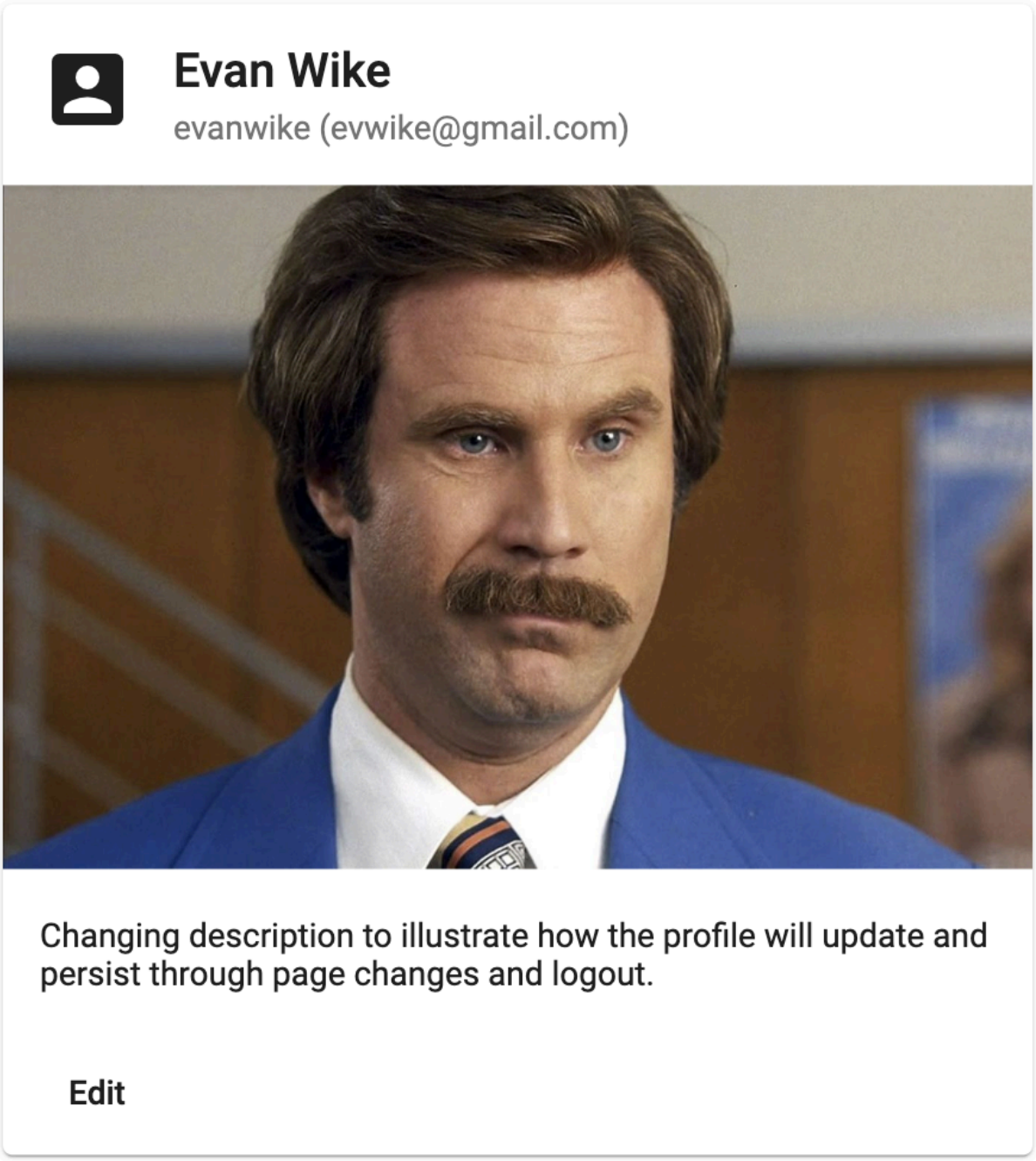
https://www.philly.com/resizer/CXGUPklCGwzTHeJojBD

Description

Changing description to illustrate how the profile will update and persist through page changes and logout.

Update Cancel

Edited user profile



Updated profile component

```
import {NgModule} from '@angular/core';
import {CommonModule} from '@angular/common';
import {RouterModule, Routes} from '@angular/router';

import {LoginComponent} from './login/login.component';
import {ProfileComponent} from './profile/profile.component';
import {RegisterComponent} from './register/register.component';
import {AuthGuardService} from './core/services/auth-guard.service';
import {EditProfileComponent} from './edit-profile/edit-profile.component';

const appRoutes: Routes = [
  {path: '', redirectTo: '/profile', pathMatch: 'full'},
  {path: 'login', component: LoginComponent},
  {path: 'profile', component: ProfileComponent, canActivate: [AuthGuardService]},
  {path: 'edit-profile', component: EditProfileComponent, canActivate: [AuthGuardService]},
  {path: 'register', component: RegisterComponent}
];

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

Routing


export class User {
username: string;
email: string;
firstName: string;
lastName: string;
description: string;
imageUrl: string;
token: string;
}

User model

```
@Injectable({
  providedIn: 'root'
})
export class UserService {
  private currentUserSubject = new BehaviorSubject<User>(new User());
  public currentUser = this.currentUserSubject.asObservable().pipe(distinctUntilChanged());

  private isAuthenticatedSubject = new ReplaySubject<boolean>( bufferSize: 1);
  public isAuthenticated = this.isAuthenticatedSubject.asObservable();

  constructor(
    private apiService: ApiService,
    private http: HttpClient,
    private jwtService: JwtService
  ) {
  }

  init() {
    if (this.jwtService.getToken()) {
      this.apiService.get( path: '/user').subscribe(
        next: data => this.setAuth(data.user),
        error: err => this.reset()
      );
    } else {
      this.reset();
    }
  }

  setAuth(user: User) {
    console.log('user', user);
    this.jwtService.saveToken(user.token);

    // Set current user data into observable
    this.currentUserSubject.next(user);
    // Set isAuthenticated to true
    this.isAuthenticatedSubject.next( value: true);
  }
```

User service

```
  reset() {
    this.jwtService.destroyToken();
    this.currentUserSubject.next({} as User);
    this.isAuthenticatedSubject.next( value: false);
  }

  login(credentials): Observable<User> {
    return this.apiService.post( path: '/users/login', credentials)
      .pipe(map(
        project: data => {
          this.setAuth(data.user);
          return data;
        }
      ));
  }

  register(user): Observable<User> {
    return this.apiService.post( path: '/users', user)
      .pipe(map(
        project: data => {
          this.setAuth(data.user);
          return data;
        }
      ));
  }

  getCurrentUser(): User {
    return this.currentUserSubject.value;
  }

  update(user): Observable<User> {
    return this.apiService
      .put( path: '/user',  body: {user})
      .pipe(map( project: data => {
        // Update the currentUser observable
        this.currentUserSubject.next(data.user);
        return data.user;
      }));
  }

}
```

User service continued

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

import {HttpClient, HttpParams} from '@angular/common/http';
import {Observable, throwError} from 'rxjs';
import {catchError} from 'rxjs/operators';
import {environment} from '../../../environments/environment';

@Injectable()
export class ApiService {

  constructor(private http: HttpClient) {
  }

  private static formatErrors(error: any) {
    return throwError(error.error);
  }

  get(path: string, params: HttpParams = new HttpParams()): Observable<any> {
    return this.http.get( url: `${environment.api_url}${path}`, options: {params})
      .pipe(catchError(ApiService.formatErrors));
  }

  put(path: string, body: object = {}): Observable<any> {
    return this.http.put( url: `${environment.api_url}${path}`, body)
      .pipe(catchError(ApiService.formatErrors));
  }

  post(path: string, body: object = {}): Observable<any> {
    return this.http.post( url: `${environment.api_url}${path}`, body)
      .pipe(catchError(ApiService.formatErrors));
  }

  delete(path): Observable<any> {
    return this.http.delete( url: `${environment.api_url}${path}`)
      .pipe(catchError(ApiService.formatErrors));
  }

}
```

API service

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuardService implements CanActivate {

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

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Promise<boolean> {
    return this.userService.isAuthenticated.pipe(take( count: 1)).toPromise().then( onfulfilled: isAuthenticated => {
      if (!isAuthenticated) {
        return this.router.navigate( commands: ['login']).then( onfulfilled: () => false);
      }
      return true;
    });
  }
```

Authentication service

V. References

User Management Application (Mobile)

I. Introduction

This is also a mobile user management application that replicates a university student account login application where students can sign up and sign in with their personal information. Moreover, they also have a choice of logging in with their current Google Login username.

Afterwards, users are capable of editing their personal information (such as name, age, university, major, phone number) that they first signed up with, or the information that is missing from the Google Profile.

II. Objective

The purpose of this application is to build a basic user authentication system on Android. All authentication should be accurate and straightforward and there should be a mechanism for checking whether the credentials such as the username/email address/Google account are valid or not (validation process).

If conditions for a proper account registration are not satisfied, the validation process should affirmatively halt the signup process and thereby prevent the user from

continuing with the inappropriate credentials. Furthermore, user's entered information is expected to be stored properly in a database in order to be retrieved and displayed whenever the user logs in at another point in time. Along the same line, all the edits of the information should be updated accordingly and accurately reflected once the user is redirected to the main page.

This application will contain two main features:

- Sign Up/Authenticate a new account via an email address.
- Sign Up/Authenticate an existing Google username.

III. Approach

Both authentication mechanisms are implemented with Google Firebase authentication feature (Auth object) but with different options. The first feature needs activating the email/password feature of Firebase authentication while the latter needs the Google Social Login feature instead.

The email/password feature contains the signup and signin methods that will help authenticate and register users conveniently.

The Google Social Login will help expedite the sign up process because users are allowed to proceed with their existing/current Google account without having to go through the registration process again. All the user's entered information (such as name, age, university, major, and phone number) is stored in the Firestore database, the latest version of Google Firebase's real-time databases. Before Firestore, our other applications tended towards the initial version of the Firebase database, which was called Realtime database. However, Firestore is more intuitive, object-oriented, and performance-oriented. Getting familiar with Firestore will help in the long run. Records are stored in a document under the user account's ID, which is uniquely created for every single user, regardless of which login mechanism (regular/social) is activated.

For example, when user bobsmith@yahoo.com signs up successfully, his account will be given an ID "*f12saZsow0*", which will then be used as an entry to the database document. Each document will contain JSON-formatted list of objects such as name, age, university, major, and phone number. This mechanism of storing information under user's ID instead of user's username will make sure records are indeed unique and generally avoid any faulty behaviors from the false-positive authentication situations.

VI. Screenshots

Separate Activity for Google Login (a separate pop-up screen with the default google login screen will ask user to sign in with the current credentials)

```
public class GoogleSignInActivity extends AppCompatActivity {
    private static final String TAG = "GoogleSignInActivity";
    private GoogleSignInClient mGoogleSignInClient;
    private static final int RC_SIGN_IN = 9001;
    private FirebaseAuth mAuth;
    private FirebaseFirestore db;
    DocumentReference docRef;
    Map<String, Object> data;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_google_sign_in);
        mAuth = FirebaseAuth.getInstance();
        db = FirebaseFirestore.getInstance();
        ///Google Sign In Implementation
        // Configure Google Sign In
        GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                .requestIdToken(getString(R.string.default_web_client_id))
                .requestEmail()
                .build();
        mGoogleSignInClient = GoogleSignIn.getClient( activity: this, gso);
        mAuth = FirebaseAuth.getInstance();
        signIn();
    }
    private void signIn() {
        Intent signInIntent = mGoogleSignInClient.getSignInIntent();
        startActivityForResult(signInIntent, RC_SIGN_IN);
    }
```

Main function that's in charge of signing the user in with Google login. If the authentication is successful, a callback function is then executed -> updateDB(user) where the user is the returned user object.

```
private void firebaseAuthWithGoogle(GoogleSignInAccount acct) {
    Log.d(TAG,  msg: "firebaseAuthWithGoogle:" + acct.getId());

    AuthCredential credential = GoogleAuthProvider.getCredential(acct.getIdToken(),  s1: null);
    mAuth.signInWithCredential(credential)
            .addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    if (task.isSuccessful()) {
                        // Sign in success, update UI with the signed-in user's information
                        Log.d(TAG,  msg: "signInWithCredential:success");
                        FirebaseUser user = mAuth.getCurrentUser();
                        Pass user object to this method
                        updateDB(user);
                    } else {
                        // If sign in fails, display a message to the user.
                        Log.w(TAG,  msg: "signInWithCredential:failure", task.getException());
                        Snackbar.make(findViewById(R.id.gsignin_layout),  text: "Authentication Failed.", Snackbar.LENGTH_SHORT).
                        updateDB( user: null);
                    }

                    // ...
                }
            });
}
```

updateDB function that either updates current user information to the document in Firestore, or create a new document in the Firestore if the user just signs in for the first time

```
private void updateDB(FirebaseUser user) {
      hideProgressDialog();
    if (user != null){
        String user_email = user.getEmail();
        String user_id = user.getUid();
        String user_name = user.getDisplayName();
        ///Using FireStore to update document with id = user_id which is passed as a parameter
        docRef = db.collection( collectionPath: "users").document(user_id);
        // Add document data  with string user_id using a hashmap
        data = new HashMap<>();
        data.put("email", user_email);
        data.put("name", user_name);
        //asynchronously write data
        docRef.update(data)
                .addOnSuccessListener(new OnSuccessListener<Void>() {
                    @Override
                    public void onSuccess(Void aVoid) {
                        Log.d(TAG,  msg: "Personal Details Updated Successfully!");
                        Intent intent = new Intent( packageContext: GoogleSignInActivity.this, MainActivity.class);
                        startActivity(intent);
                        finish();
                    }
                })
                .addOnFailureListener(new OnFailureListener() {
                    @Override
                    public void onFailure(@NonNull Exception e) {
                        Log.w(TAG,  msg: "Error Updating Details! Please Try Again Later!", e);
                        docRef.set(data);
                        Intent intent = new Intent( packageContext: GoogleSignInActivity.this, MainActivity.class);
                        startActivity(intent);
                    }
                });
    }
```

Profile image and User profile display

```
    ///Retrieve User Info
    user_id = mAuth.getCurrentUser().getUid();
    getUserInfo(user_id);
    listView = (ListView) findViewById(R.id.listView);
    items = new ArrayList<String>();
    ArrayAdapter<String> itemsAdapter =
            new ArrayAdapter<~>( context: this, android.R.layout.simple_list_item_1, items);
    listView.setAdapter(itemsAdapter);

    //Profile Image
    mProfileImage = (ImageView) findViewById(R.id.profileImg);
    //Click a picture
    mProfileImage.setOnClickListener((v) -> {
            Intent intent = new Intent(Intent.ACTION_PICK);
            intent.setType("image/*"); //restricts selection
            startActivityForResult(intent,  requestCode: 1);
    });
```

LogOut() that is used in both authentication cases

```
public void LogOut(View v){
    FirebaseAuth.getInstance().signOut();
    Intent intent = new Intent( packageContext: MainActivity.this, WelcomeActivity.class);
    startActivity(intent);
    finish();
    Toast.makeText( context: MainActivity.this,  text: "logout", Toast.LENGTH_SHORT);
}
```

getUserInfo() function will take care of retrieving user object from the Firestore database. If successfully, it will pass the returned object to another function for display and presentation

```
public void getUserInfo(final String ID) {
    db.collection( collectionPath: "users")
            .get()
            .addOnCompleteListener((task) → {
                    if (task.isSuccessful()) {
                        for (QueryDocumentSnapshot document : task.getResult()) {
                            if (document.getId().equals(ID)) {

                                Map<String, Object> user;
                                user = document.getData();
                                displayUserInfo(user);

                            }
                            //Log.d(TAG, document.getId() + " => " + document.getData());
                        }
                    } else {
                        //Log.w(TAG, "Error getting documents.", task.getException());
                    }
            });
}
```

Main function that displays the forwarded user object onto a list view on the front end.

```
//    This function takes in a map and display it as a listview
    public void displayUserInfo (Map<String, Object> usermap){
        ArrayList<String> items = new ArrayList<~>();
        String name = ((usermap == null) || (usermap.get("name") == null) ? "N/A" : usermap.get("name").toString());
        String age = ((usermap == null) || (usermap.get("age") == null) ? "N/A" : usermap.get("age").toString());
        String email = ((usermap == null) || (usermap.get("email") == null) ? "N/A" : usermap.get("email").toString());
        String phone = ((usermap == null) || (usermap.get("phone") == null) ? "N/A" : usermap.get("phone").toString());
        String university = ((usermap == null) || (usermap.get("university") == null) ? "N/A" : usermap.get("university").toString());
        String major = ((usermap == null) || (usermap.get("major") == null) ? "N/A" : usermap.get("major").toString());

        items.add("Name: " + name);
        items.add("Email: " + email);
        items.add("Age: " + age);
        items.add("Phone Number: " + phone);
        items.add("Institution/ Organization: " + university);
        items.add("Major: " + major);
        ArrayAdapter<String> itemsAdapter =
                new ArrayAdapter<~>( context: this, android.R.layout.simple_list_item_1, items);
        listView.setAdapter(itemsAdapter);
    }
```

Profile image. If an image is selected from the phone, this will save the URI and upload it to Firebase Storage.

```
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    //See if it returns the same code from the startactivityforresult above
    if (requestCode == 1 && resultCode == Activity.RESULT_OK){
        final Uri imageURI = data.getData();
        resultURI = imageURI;
        mProfileImage.setImageURI(resultURI);
        applyChanges();
    }
}
```

This will process the image, upload it to Firebase storage, and display it in the user's main page.

```
private void applyChanges(){

    if (resultURI != null){
        final StorageReference filePath = FirebaseStorage.getInstance().getReference().child("profile_images").child(user_id);
        Bitmap bitmap = null;
        //Add uri to bitmap
        try {
            bitmap = MediaStore.Images.Media.getBitmap(getApplication().getContentResolver(), resultURI);
        } catch (IOException e) {
            e.printStackTrace();
        }
        ByteArrayOutputStream baos = new ByteArrayOutputStream(); //compress images
        bitmap.compress(Bitmap.CompressFormat.JPEG,  quality: 20, baos);
        byte[] data = baos.toByteArray();
        UploadTask uploadTask = filePath.putBytes(data);
        //Upload task will save the image to Firebase Database
        uploadTask.addOnSuccessListener((OnSuccessListener) (taskSnapshot) → {
                filePath.getDownloadUrl().addOnSuccessListener((OnSuccessListener) (uri) → {
                        Map newImage = new HashMap();
                        newImage.put("profileImageURL", uri.toString());
                        DocumentReference docRef = db.collection( collectionPath: "users").document(user_id);
                        docRef.update( field: "profileImageURL", uri.toString())
                        .addOnSuccessListener((OnSuccessListener) (aVoid) → {
                                Log.d(TAG,  msg: "DocumentSnapshot successfully updated!");
                        });
                        return;
                }).addOnFailureListener((exception) → {
                        finish();
                        return;
                });
        });
    }
    else {
        finish();
    }
```

This function verifies user's credentials

```
//This is where the identity check is. The main variable is the validation flag. If it is false, then the user
//will not go anywhere beyond the login page. Else if the user enters the right username and password, he will be directed
//to the next page. Username and password are determined by the 2 variables "userName" and "password"
public void checkCredentials(View v)
{
    final String email = mEmail.getText().toString();
    final String password = mPassword.getText().toString();
    if(email.isEmpty() ){
        mEmail.setError("Email Address cannot be blank");
    }
    else if(!Patterns.EMAIL_ADDRESS.matcher(email).matches()){
        mEmail.setError("Enter a Valid Email");
    }
    else if(password.isEmpty()){
        mPassword.setError("Password cannot be blank");
    }
    else {
        mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener( activity: LoginActivity.this, (task) → {
                if (!task.isSuccessful()) {
                    Toast.makeText( context: LoginActivity.this,  text: "Authentication Failed!", Toast.LENGTH_SHORT).show();
                }
        });
    }

}
```

This function listens to any state change (not signed in vs signed in) and redirects accordingly.

```
mAuth = FirebaseAuth.getInstance();
firebaseAuthListener = (AuthStateListener) (firebaseAuth) → {
        FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser(); //get the information of the current user
        if (user != null){
            //if not null, move to another activity, to be created later.
            //Remember the current context!
            Intent intent = new Intent( packageContext: LoginActivity.this, MainActivity.class);
            intent.putExtra( name: "user_id", user.getUid());
            intent.putExtra( name: "user_email", user.getEmail());
            startActivity(intent);
            finish();
        }
};
```

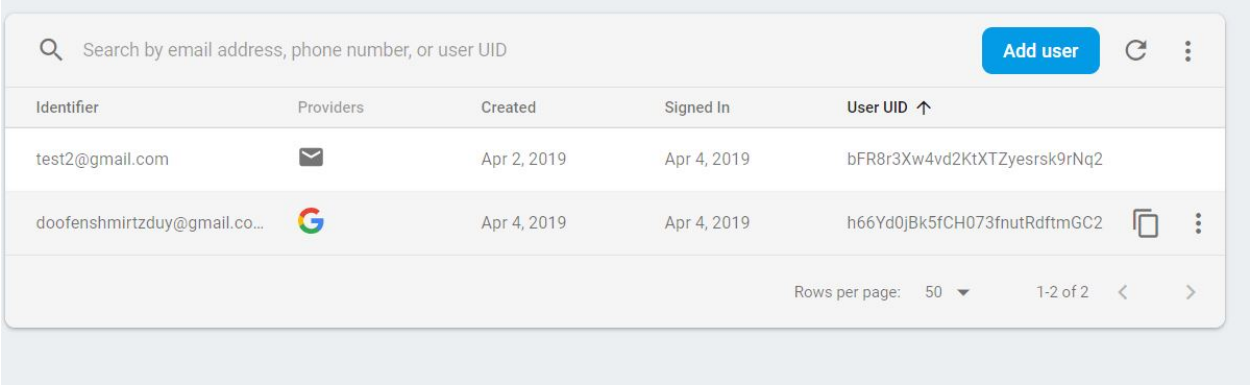
This function (although having the same name as the applychanges for profile image function) is responsible for credential validation process such as username is not in form of an email address, or password is not long enough, there is a blank/unfilled textbox.

```
public void applyChanges(View v)
{
    final String email = mEmail.getText().toString();
    final String age = mAge.getText().toString();
    final String phone = mPhone.getText().toString();
    final String university = mUniversity.getText().toString();
    final String name = mName.getText().toString();
    final String major = mMajor.getText().toString();
    if(email.isEmpty()){
        mEmail.setError("Email Address cannot be blank");
    }
    else if(!Patterns.EMAIL_ADDRESS.matcher(email).matches()){
        mEmail.setError("Enter a Valid Email");
        mEmail.requestFocus();
    }
    else if(name.isEmpty()){
        mName.setError("Name cannot be blank");
        mName.requestFocus();
    }
    else if(age.isEmpty()){
        mAge.setError("Age cannot be blank");
        mAge.requestFocus();
    }
    else if(university.isEmpty()){
        mUniversity.setError("University cannot be blank");
        mUniversity.requestFocus();
    }
    else if(phone.isEmpty()){
        mPhone.setError("Phone cannot be blank");
        mPhone.requestFocus();
    }
    else if(major.isEmpty()){
        mMajor.setError("Major cannot be blank");
        mMajor.requestFocus();
    }
```

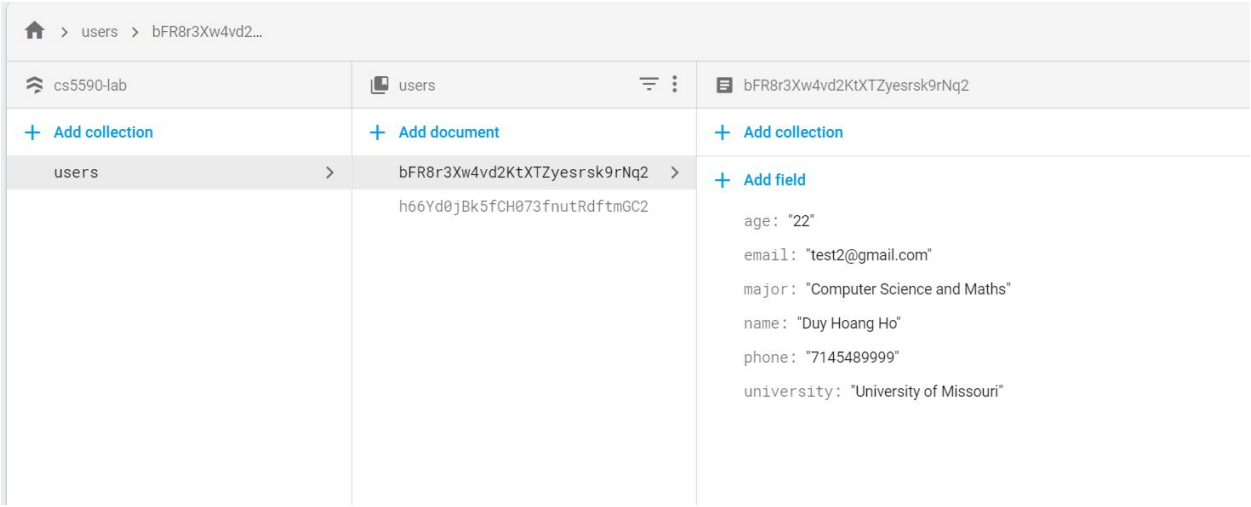
This function ensures that the latest user-entered information is updated and reflected properly in the application

```
else {
        ///Using FireStore
        DocumentReference docRef = db.collection( collectionPath: "users").document(user_id);
        // Add document data  with string user_id using a hashmap
        Map<String, Object> data = new HashMap<>();
        data.put("email", email);
        data.put("name", name);
        data.put("phone", phone);
        data.put("age", age);
        data.put("university", university);
        data.put("major", major);
        //asynchronously write data
        docRef.update(data)
                .addOnSuccessListener((OnSuccessListener) (aVoid) → {
                        Log.d(TAG,  msg: "Personal Details Updated Successfully!");
                        Intent intent = new Intent( packageContext: EditDetailsActivity.this, MainActivity.class);
                        startActivity(intent);
                        finish();
                })
                .addOnFailureListener((e) → {
                        Log.w(TAG,  msg: "Error Updating Details! Please Try Again Later!", e);
                });

}
```

Firebase Authentication Dashboard



Firestore's document database



V. References:

[Firestore Tutorials]

[Glide Library for Profile Image]