THE STATE UNIVERSITY OF ZANZIBAR (SUZA)

SCHOOL OF BUSINESS (SOB)

BARCHELOR DEGREE IN ICT WITH ACCOUNTING

LECTURE NAMES : **Mr. Masoud**

TYPE OF ASSIGNMENT: **SuzaConnect Project Final Report**

Registration NUMBER: **BITA/6/22/079/TZ**

SUBJECT CODES: **WT822**

SUZACONNECT

PROJECT FINAL REPORT

Contents

1. Project Overview

1.1 Introduction

SuzaConnect is an Angular-based web application designed to facilitate communication and connectivity among students and staff at the National University of Zanzibar (SUZA). The platform aims to provide a seamless interface for users to sign up, log in, and access various features like viewing contact information, learning about the university, and navigating to different sections of the site such as the dashboard and home page.

1.2 Objectives

• To provide a user-friendly interface for SUZA students and staff.

• To enable seamless navigation and access to different sections of the website.

• To ensure secure sign-up and login processes.

2. Technologies Used

2.1 Angular

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. It is maintained by Google and a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS.

Key Features:

- • Component-Based Architecture: Angular structures the application into reusable and maintainable components.
- • Dependency Injection: Angular's built-in dependency injection helps in managing the service instances efficiently.
- • Two-Way Data Binding: This allows automatic synchronization of data between the model and view components.
- • Routing: Angular's RouterModule provides a robust way to handle navigation and URL manipulation.

2.2 TypeScript

TypeScript is a strongly typed superset of JavaScript which compiles to plain JavaScript. It provides optional static typing, classes, and interfaces, making it suitable for large-scale applications.

2.3 HTML & CSS

HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are the foundational technologies for building web pages. HTML provides the structure, while CSS is used for presentation and layout.

2.4 Node.js and npm

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. npm (Node Package Manager) is used for managing the dependencies and libraries required by the project.

2.5 Angular CLI

The Angular CLI (Command Line Interface) is a powerful tool that simplifies the development, building, and testing processes of Angular applications.

3. Key Concepts

3.1 Components

Components are the building blocks of an Angular application. Each component consists of:

- • HTML Template: Defines the view for the component.
- • TypeScript Class: Contains the logic and data bindings.
- • CSS Styles: Defines the appearance of the component.
- Example of a Component Definition:

SuzaConnect Front End > src > app > about-us > TS about-us.component.ts > AboutUsComponent

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-about-us',
5     templateUrl: './about-us.component.html',
6     styleUrls: ['./about-us.component.css']
7   })
8   export class AboutUsComponent {
9
10  }
```

3.2 Modules

Modules are containers for a cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities. Angular modules consolidate components, directives, pipes, and services that belong together, enabling reusability.

- Example of a Module Definition:

```
SuzaConnect Front End > src > app > TS app.component.spec.ts > ...
import { TestBed } from '@angular/core/testing';
import { RouterTestingModule } from '@angular/router/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [
        RouterTestingModule
      ],
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  });

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });
```

3.3 Routing

Angular's RouterModule provides a sophisticated mechanism for navigating through the application. It supports both declarative and programmatic navigation and can be configured to handle different paths, parameters, and guards for route protection.

Example of Routing Configuration:

```
SuzaConnect Front End > src > app > TS app.module.ts > AppModule
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { SignUpComponent } from './sign-up/sign-up.component';
import { UserService } from './services/user.service';
import { HomeComponent } from './home/home.component';
import { AboutUsComponent } from './about-us/about-us.component';
import { ContactsComponent } from './contacts/contacts.component';
import { DashboardComponent } from './Dashboard/Dashboard.component';
import { LoginComponent } from './Login/Login.component';

@NgModule({
  declarations: [
    AppComponent,
    SignUpComponent,
    HomeComponent,
```

3.4 Forms and Data Binding

Angular provides powerful form handling capabilities through its FormsModule. It supports both template-driven and reactive forms.

- Example of Template-Driven Form:

```
<form (ngSubmit)="onSubmit()" *ngIf="!registrationSuccessful">
  <div class="form-group">
    <label for="first-name">First Name</label>
    <input type="text" id="first-name" name="firstName" [(ngModel)]="user.firstName" required>
  </div>
  <div class="form-group">
    <label for="last-name">Last Name</label>
    <input type="text" id="last-name" name="lastName" [(ngModel)]="user.lastName" required>
  </div>
  <div class="form-group">
    <label for="email">Email</label>
    <input type="email" id="email" name="email" [(ngModel)]="user.email" required>
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" id="password" name="password" [(ngModel)]="user.password" required>
  </div>
  <button type="submit" class="register-btn">Register Now</button>
  <p class="login-link">Already registered? <a routerLink="/login" class="login" style="cursor: pointer; t
</form>
```

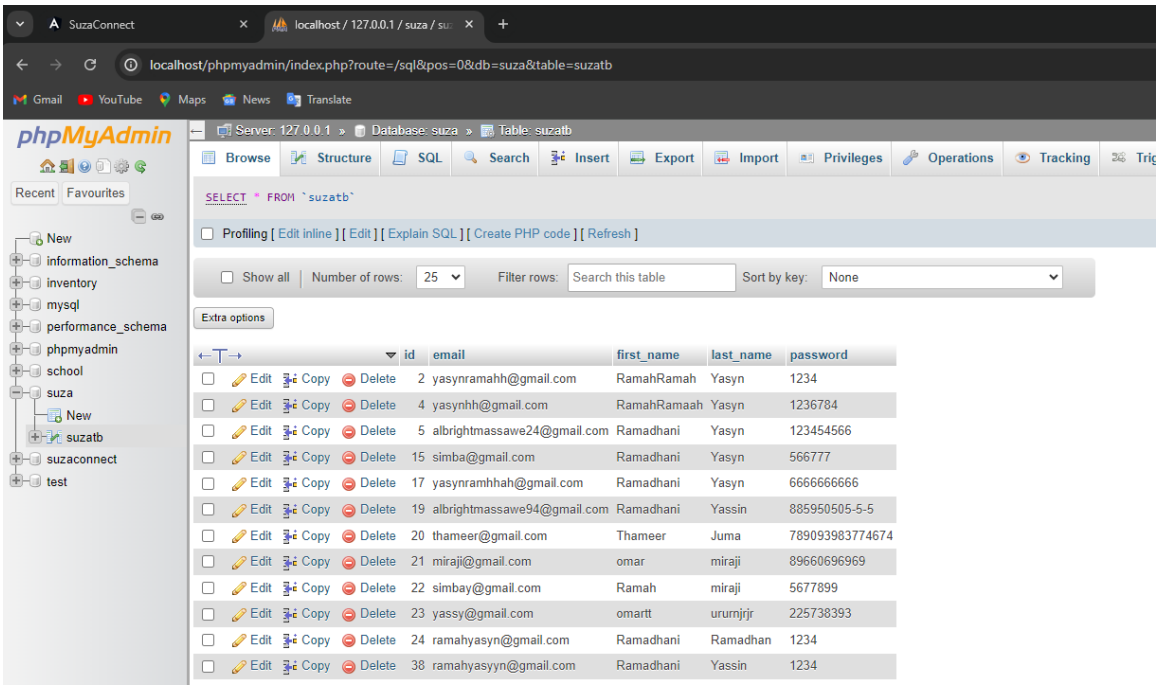
Application Program Interfaces (API's)

```
@RestController
@RequestMapping("/api/users")
public class UserController {
    @Autowired
    private UserRepository userRepository;

    @PostMapping("/signup")
    public ResponseEntity<Object> createUser(@RequestBody User user) {
        try {
            userRepository.save(user);
            return
ResponseEntity.status(HttpStatus.CREATED).body(Map.of("message", "User
registered successfully"));
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(Map.of("error
", "Failed to register user"));
        }
    }
    @GetMapping("/{email}")
    public ResponseEntity<User> getUserByEmail(@PathVariable String email)
{
        User user = userRepository.findByEmail(email);
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
```

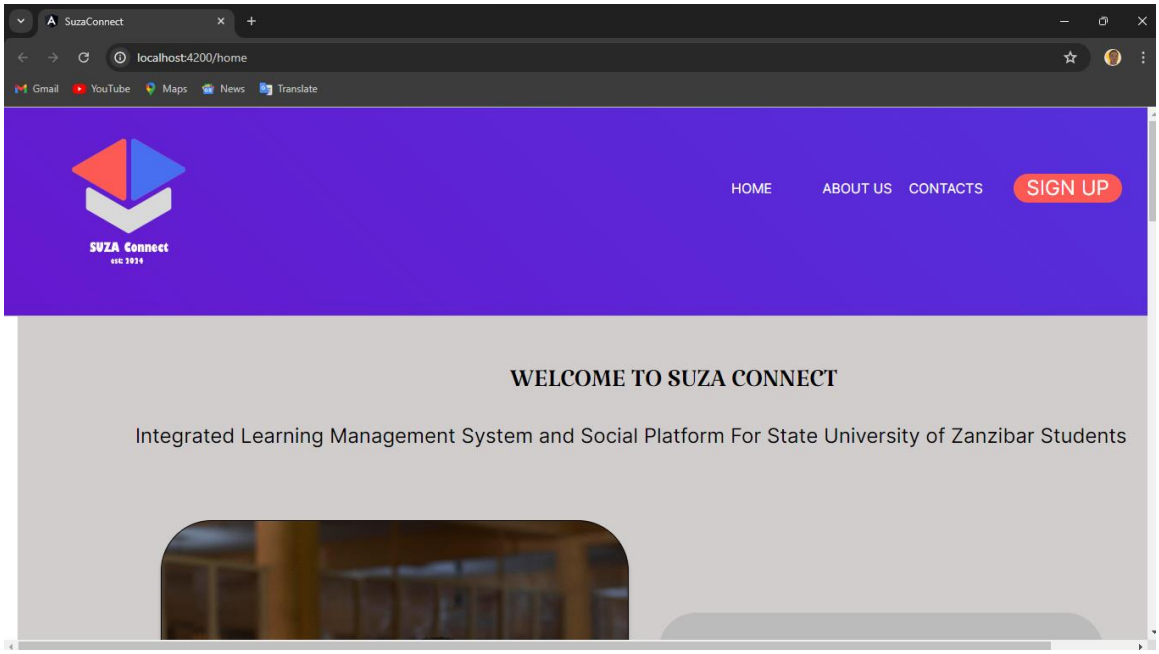
```
        }
        return new ResponseEntity<>(user, HttpStatus.OK);
    }

    @GetMapping("id")
    public ResponseEntity<List<User>> getAllUsers() {
        List<User> users = userRepository.findAll();
        return new ResponseEntity<>(users, HttpStatus.OK);
    }
    @PostMapping("/login")
    public ResponseEntity<Map<String, String>> loginUser(@RequestBody User
loginRequest) {
        User user = userRepository.findByEmail(loginRequest.getEmail());
        if (user == null) {
            return new ResponseEntity<>(Map.of("error", "User not found"),
HttpStatus.NOT_FOUND);
        }
        if (!user.getPassword().equals(loginRequest.getPassword())) {
            return new ResponseEntity<>(Map.of("error", "Invalid
password"), HttpStatus.UNAUTHORIZED);
        }
        return ResponseEntity.ok().body(Map.of("message", "Login
successful"));
    }
    @DeleteMapping("/{email}")
    public ResponseEntity<List<User>> deleteUser(@PathVariable String
email) {
        User user = userRepository.findByEmail(email);
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        userRepository.delete(user);
        List<User> users = userRepository.findAll(); // Retrieve updated
list
        return new ResponseEntity<>(users, HttpStatus.OK);
    }
    @PutMapping("/{email}")
    public ResponseEntity<String> updateUser(@PathVariable String email,
@RequestBody User updatedUser) {
        User user = userRepository.findByEmail(email);
        if (user == null) {
            return new ResponseEntity<>("User not found",
HttpStatus.NOT_FOUND);
        }
```

4. Project views and Database Schema

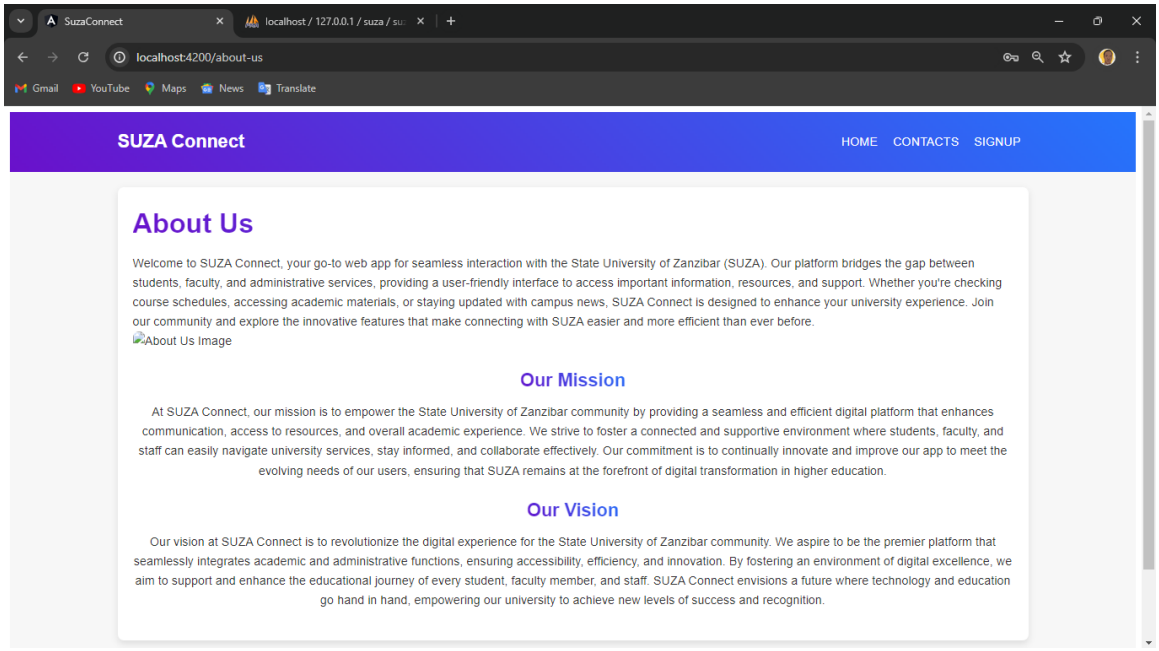
Database Schema



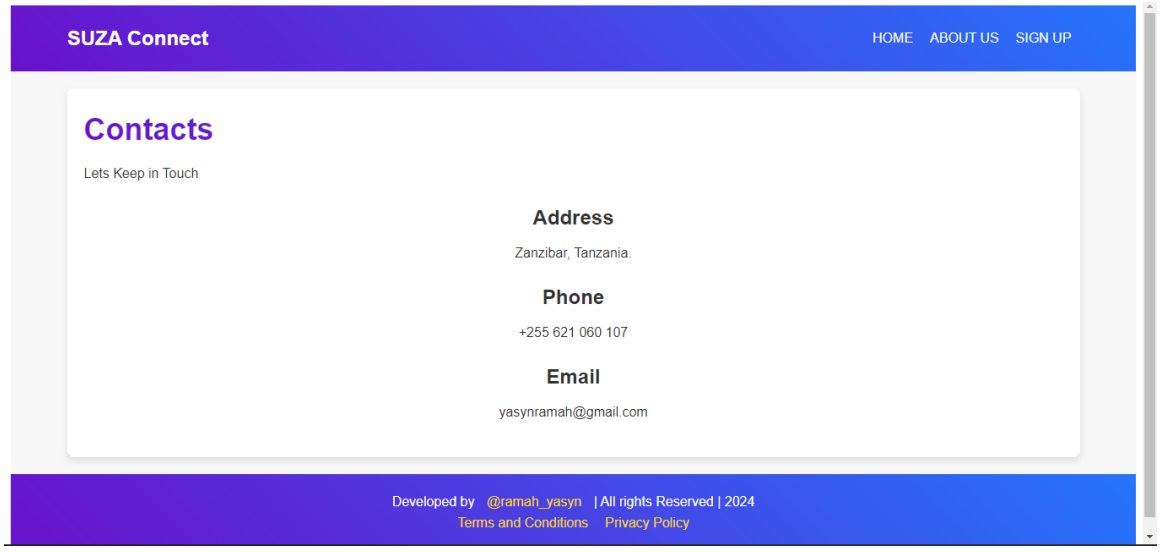
Home Page



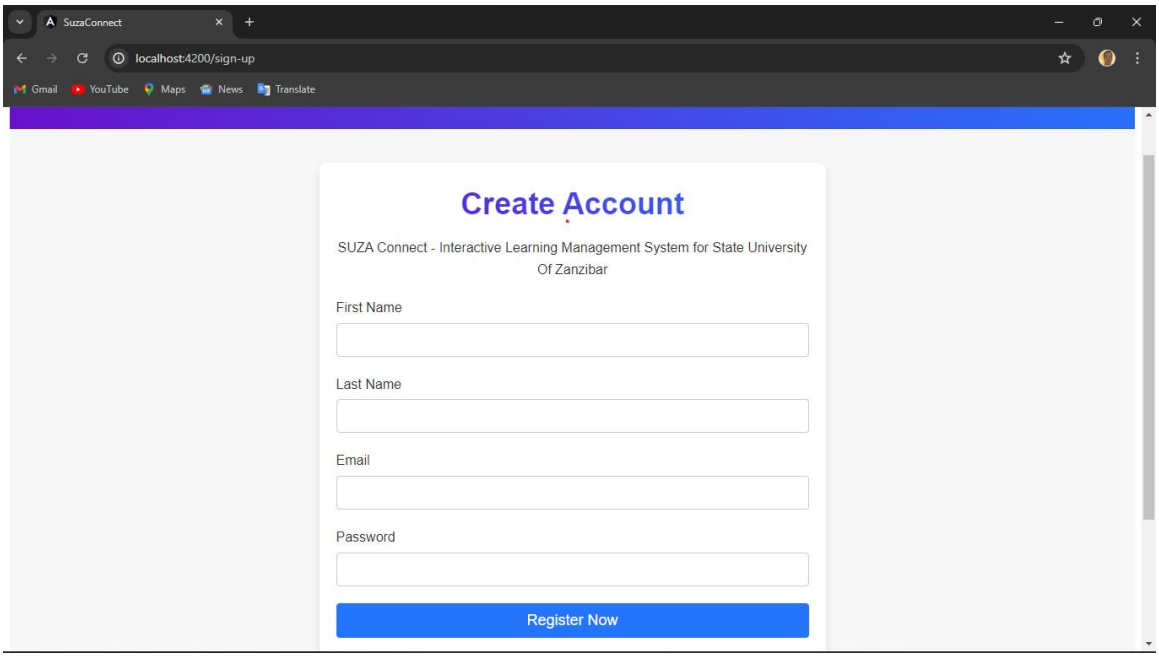
About us page



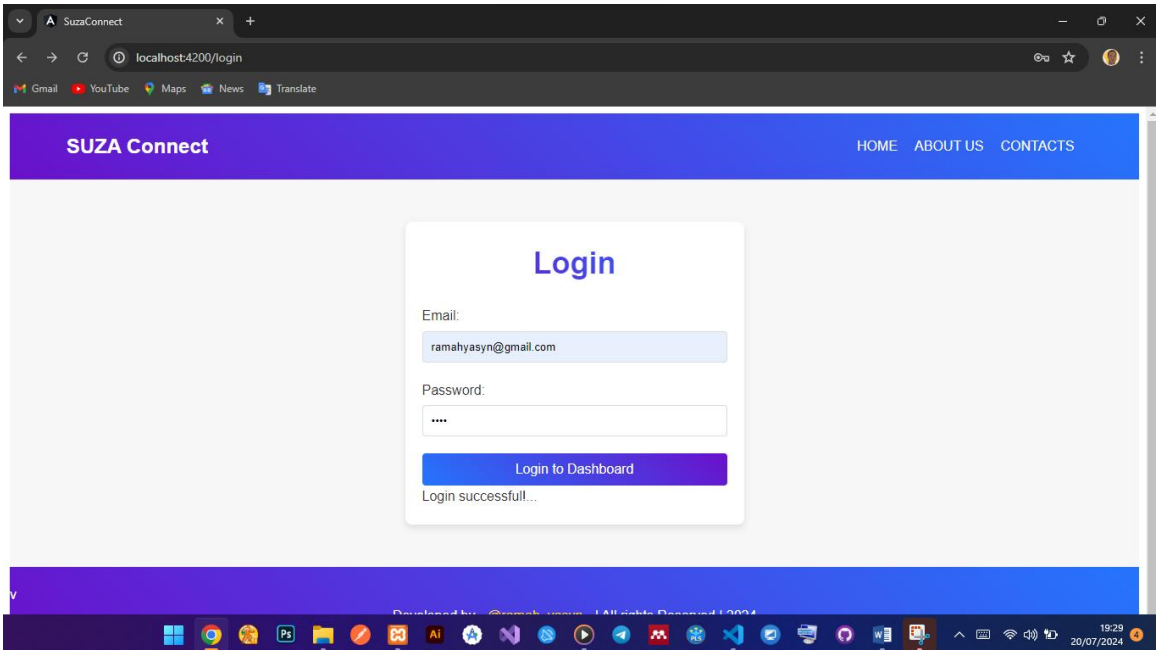
Contacts View



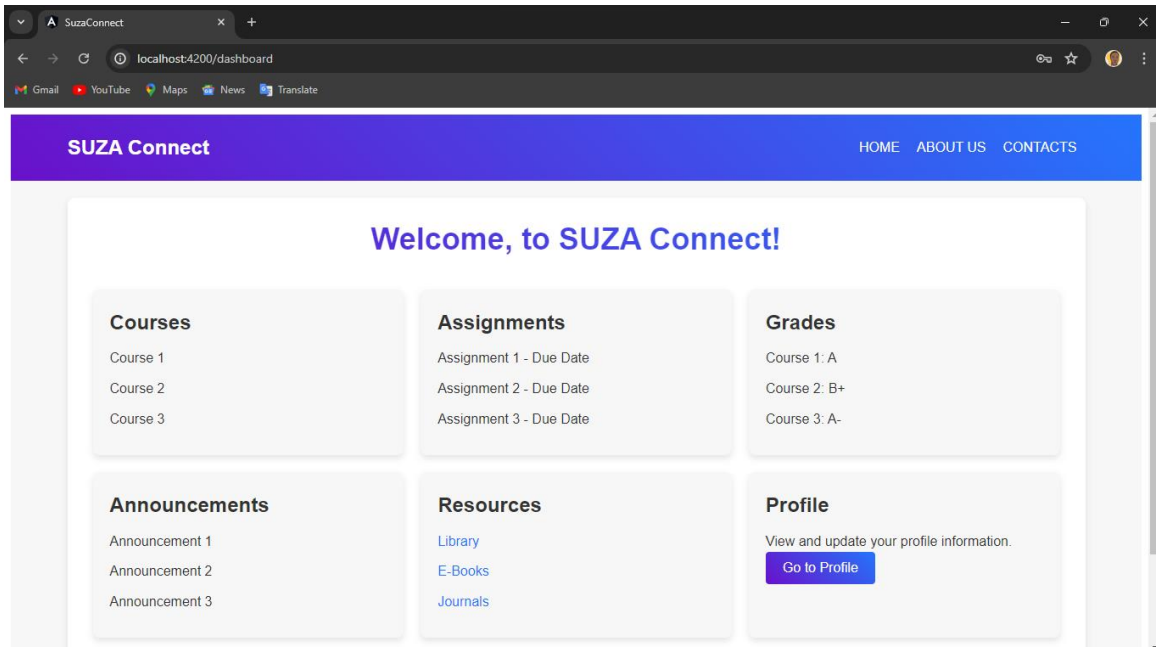
Signup View



Login



Dashboard



5. Conclusion

SuzaConnect leverages the powerful features of Angular and TypeScript to provide a robust and maintainable web application for SUZA students and staff. By utilizing component-based architecture, modular design, and advanced routing mechanisms, the project ensures scalability and ease of use. The project demonstrates a comprehensive understanding of modern web development practices and the effective use of Angular's ecosystem.