Integrated Application Development Project

Module Name	DSE204/03, MCDSE204/03-FT Integrated Application Development
Course Name	Bachelor in Software Engineering (Honours) (Application Development)
Start Date	25 June 2025
Submission Date	6 Jul 2025
Course Lead/Tutor	Tok Bee Choo (Grace)
Student ID:	041240323
Student Name:	Benjamin Tan Zhi Hong

Table of Content

Table of Figure

1.0 Introduction

The Know-Your-Neighbourhood (KYN) application is a community-focused platform that allows users to register, log in, and interact through forums. This report documents the integration of Google OAuth2 API for authentication and Google Weather API for location-based weather services, addressing the project tasks outlined in the brief.

1.1 Tool Used in Development

The tools used during the project development process are:

- Java with Spring Boot
- Apache Tomcat
- MySQL Database
- phpMyAdmin / MySQL Client
- Eclipse IDE/ STS
- Microsoft Word
- Microsoft PowerPoint
- Visual Studio Code
- NodeJS
- React
- Draw.io

2.0 Apply the Knowledge of API Research to Design an Application

2.1 Analyze the Scenario and Requirement

- **Scenario:** Enhance KYN to support login via Google OAuth2 and display weather data based on user location.
- **Requirements:**
    - Secure user authentication using an existing API (Google OAuth2).
    - Fetch user details (name, email) post-authentication.
    - Provide location-based weather updates using Google Weather API.
    - Ensure compatibility with existing Spring Boot backend and React frontend.
- **Selected APIs:**
    - Google OAuth2 API: For secure login and user data retrieval.
    - Google Weather API: For real-time weather data based on geolocation.

2.2 Wireframe for API Utilization

2.2.1 Login Page (Login.js):

- **API Usage:** Google OAuth2 API redirects to Google's authentication page, returns an access token, and fetches user info via /api/user/info.

Figure 1 Google OAuth2 API Wireframe

2.2.2 Home Page (Home.js):

- **API Usage:** Google Weather API fetches data using user's geolocation (latitude/longitude).

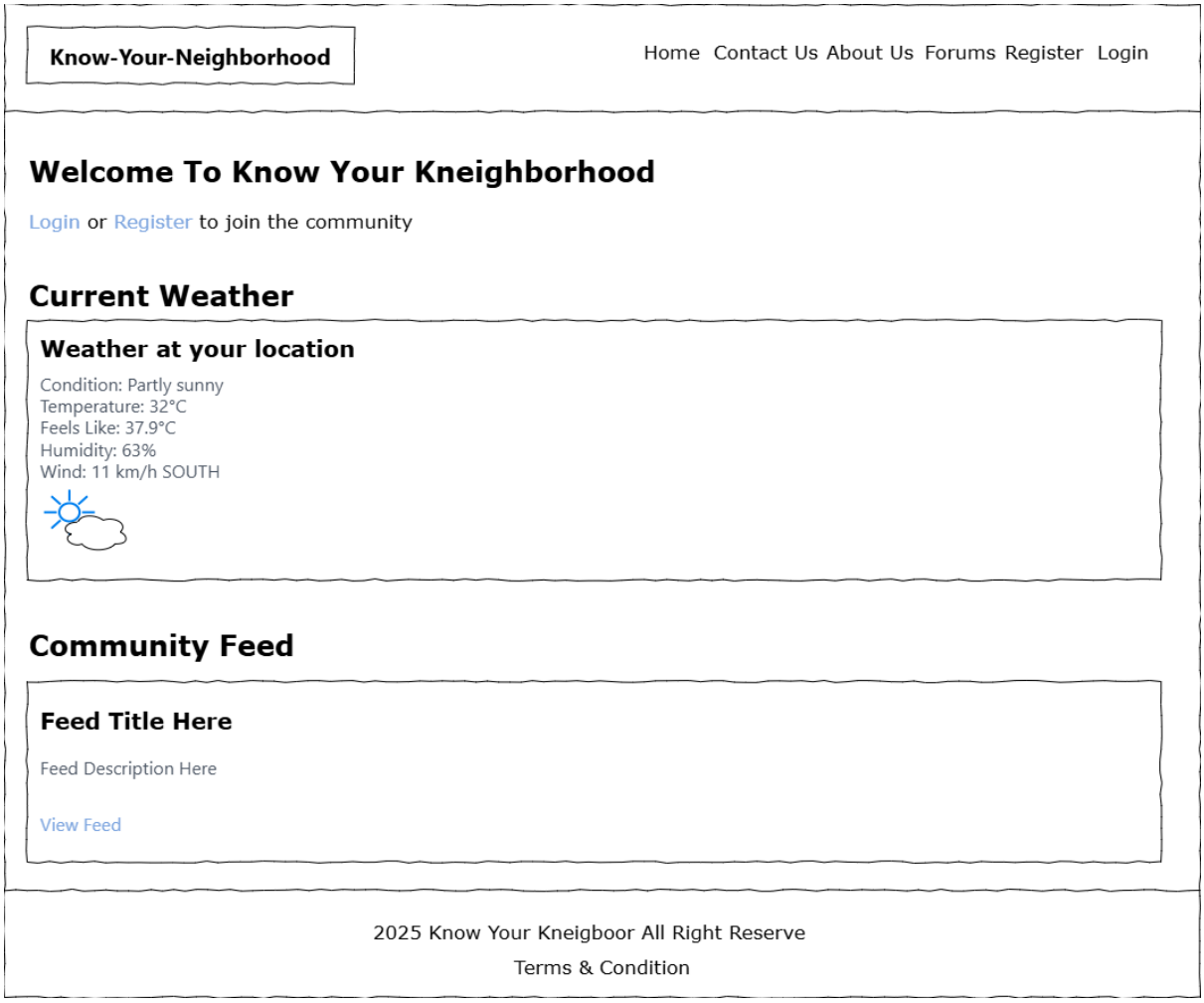

Figure 2 Weather API Wireframe

2.3 Scope and Target Platforms

1. **Scope:**
   - Implement secure login with Google OAuth2.
   - Display weather data on the Home page.
   - Maintain existing functionality (forums, user profiles, etc.).
2. **Target Platforms:**
   - Web browsers (Chrome, Firefox, Safari) via React frontend.
   - Backend hosted on Apache Tomcat with MySQL database.

2.4 Evaluation and Justification of API Selection

1. **Google OAuth2 API:**
    - **Justification:** Industry-standard for SSO, widely adopted, supports secure token-based authentication.
    - **Security:** Uses OAuth 2.0 protocol, HTTPS, and limited scopes (openid, profile, email).
    - **Suitability**: Seamless integration with Spring Security in SecurityConfig.java.
2. **Google Weather API:**
    - **Justification:** Provides hyperlocal, real-time weather data, ideal for community apps.
    - **Security:** API key-based access, restricted to specific domains, and HTTPS enforced.
    - **Suitability:** Integrated in Home.js using user geolocation for relevant updates.

3.0 Application Implementation

3.1 Types of Backend, Frontend and API Implementation

1. **Backend Types:**
   - **Monolithic Architecture:** Used in KYN with Spring Boot, where all services (authentication, forums, comments) are managed in a single codebase. This simplifies development and deployment for the project's scope.
   - **Microservices:** Considered for scalability but not implemented due to the application's moderate complexity and single-server deployment.
   - **Serverless:** Evaluated but deemed unsuitable as it requires cloud-specific configurations (e.g., AWS Lambda), which are outside the project's technical environment.
2. **Frontend Types:**
   - **Single-Page Application (SPA):** Implemented using React, enabling dynamic content updates without full page reloads, as seen in App.js with react-router-dom for routing.
   - **Static Sites:** Not used due to the need for dynamic interactions (e.g., forum posts, comments).
   - **Progressive Web Apps (PWA):** Considered for offline capabilities but not implemented due to time constraints.
3. **API Implementation Types:**
   - **REST APIs:** Used for internal backend services (e.g., /api/forums, /api/comments) and external integrations (Google OAuth2, Google Weather API).
   - **OAuth APIs:** Specifically used for Google OAuth2 to handle authentication flows.
   - **WebSocket APIs:** Not used but noted for potential real-time features like live forum updates.

3.2 Development Environments

1. **Backend Environment:**
    - **Spring Boot:** Framework for building RESTful APIs, configured in pom.xml with dependencies like spring-boot-starter-web, spring-boot-starter-security, and spring-boot-starter-data-jpa.
    - **Eclipse IDE/Spring Tool Suite (STS):** Used for coding, debugging, and running the backend application. The mvn spring-boot:run command launches the application on Apache Tomcat.
    - **MySQL/phpMyAdmin:** Manages the database schema (users, forums, comments) with spring.jpa.hibernate.ddl-auto=update in application.properties for automatic schema updates.
2. **Frontend Environment:**
    - **React with Node.js:** Development environment set up using create-react-app, with dependencies like axios for API calls and react-router-dom for routing, managed via package.json.
    - **Visual Studio Code:** Primary IDE for frontend development, with extensions for ESLint and Prettier to ensure code quality.
    - **Tailwind CSS:** Configured in index.css for responsive styling across components like Navbar.js, Home.js, and ForumPost.js.
    - **Axure RP:** Used for prototyping wireframes, ensuring UI/UX consistency.
3. **API Integration Tools:**
    - **Google Cloud Console:** Configured API keys and OAuth 2.0 credentials for Google OAuth2 and Weather APIs.
    - **Axios:** Integrated in React components (Home.js, Login.js) for HTTP requests to external APIs.

3.3 Backend and Web Service Development

3.3.1 Backend Setup:

1. Spring Boot Configuration:

The application is initialized in KnowYourNeighborhoodApplication.java with @SpringBootApplication. The application.properties file configures:

- **Database:** spring.datasource.url=jdbc:mysql://localhost:3306/kyn_app with MySQL connection.
- **Security:** spring.security.oauth2.client.registration.google.client-id for Google OAuth2.
- **Session:** server.servlet.session.timeout=30m and server.servlet.session.cookie.same-site=lax.

2. Database Schema:

- **users table:** Stores email (primary key), name, and password (BCrypt-encoded).
- **forums table:** Stores id, title, description, created_by, and created_at.
- **comments table:** Stores id, content, forum_id (foreign key), created_by, and created_at.

3. REST Endpoints:

User Management (UserController.java):

- POST /api/user/register: Registers a user with BCrypt password encoding.
- POST /api/user/login: Authenticates users and sets JSESSIONID cookie.
- GET /api/user/info: Fetches user details post-authentication.
- POST /api/user/logout: Clears session.

Forum Management (ForumController.java):

- GET /api/forums: Retrieves all forums.
- GET /api/forums/{id}: Retrieves a specific forum.
- POST /api/forums: Creates a new forum (authenticated users only).
- GET /api/community/feed: Fetches the top 5 recent forums.

Comment Management (CommentController.java):

- GET /api/comments/forum/{forumId}: Retrieves comments for a forum.
- POST /api/comments/forum/{forumId}: Creates a new comment (authenticated users only).
- Security Configuration (SecurityConfig.java):
- Configures Spring Security with Google OAuth2 using spring.security.oauth2.client.registration.google.
- Session management with SessionCreationPolicy.IF_REQUIRED and maximumSessions(1) to prevent concurrent logins.
- CORS enabled for http://localhost:3000 to allow frontend communication.
- CSRF disabled for stateless REST APIs, with HttpSessionSecurityContextRepository for session persistence.

3.3.2 Web Service Implementation:

- RESTful services built using Spring Boot's @RestController annotation.
- Secured with OAuth2 login (oauth2Login in SecurityConfig.java) and custom successHandler to set JSESSIONID and redirect to http://localhost:3000/auth-callback.
- Logging implemented using SLF4J in controllers (logger.debug) for request tracking.

3.4 Application Development with APIs

1. **Google OAuth2 API Integration:**
    - **Backend**:
        - Configured in application.properties with client-id, client-secret, scope, and redirect-uri.
        - SecurityConfig.java defines OAuth2 login flow with defaultSuccessUrl("http://localhost:3000/auth-callback").
        - UserController.java handles /api/user/info to fetch user details post-authentication, supporting both OAuth2 and form-based login.
    - **Frontend**:
        - Login.js includes a "Login with Google" button that redirects to http://localhost:8080/oauth2/authorization/google.
        - AuthCallback.js uses useEffect to call fetchUser from AuthContext.js, updating the user state and redirecting to the Home page.
    - **Flow**:
        1. User clicks "Login with Google" in Login.js.
        2. Redirects to Google's authentication page.
        3. Post-authentication, Google redirects to /login/oauth2/code/google, handled by Spring Security.
        4. Backend sets JSESSIONID and redirects to /auth-callback.
        5. Frontend fetches user info and updates AuthContext.

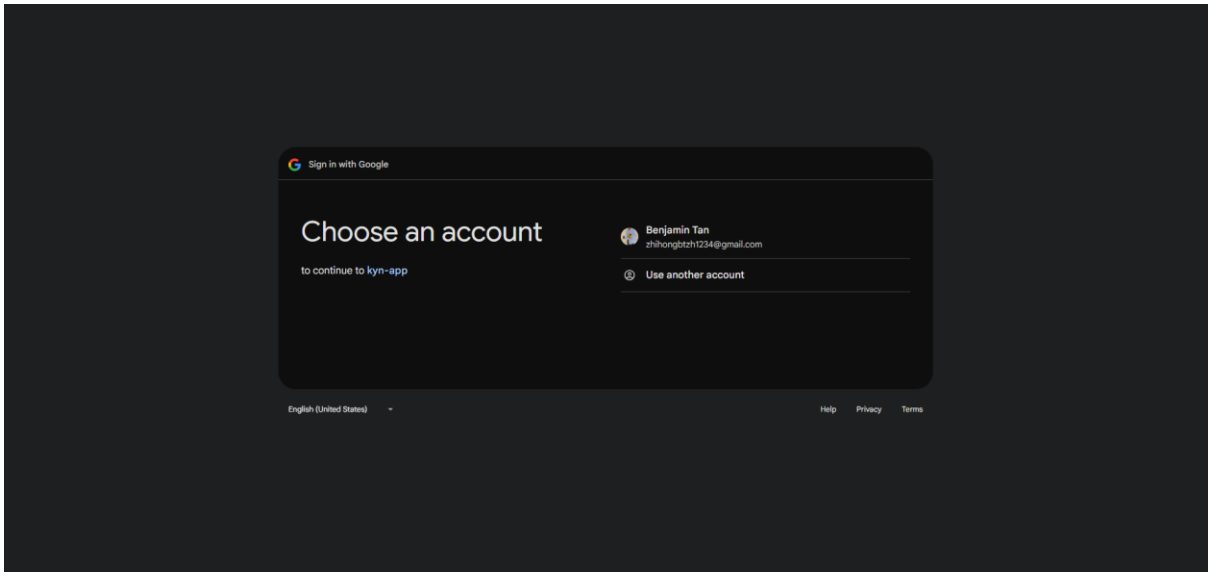

Figure 3 Google Login

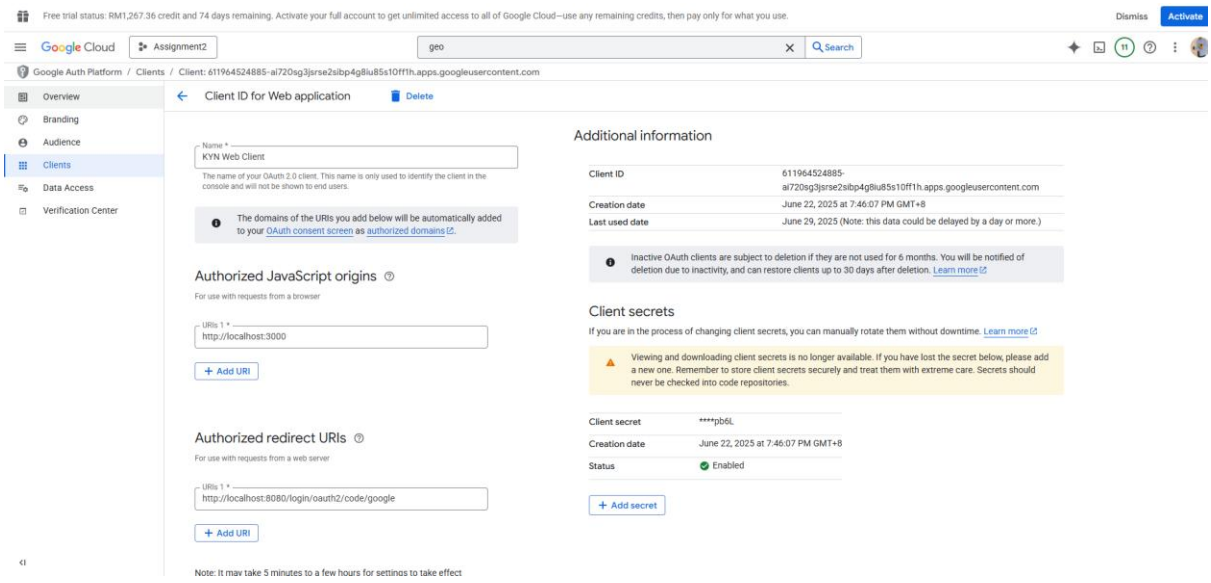

Figure 4 OAuth 2.0 Client IDs Setting

```
spring.security.oauth2.client.registration.google.client-id=611964524885-ai720sg3jsrse2sibp4g8iu85s10ff1h.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=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
spring.security.oauth2.client.registration.google.scope=openid,profile,email
spring.security.oauth2.client.registration.google.redirect-uri={baseUrl}/login/oauth2/code/google
```

Figure 5 Application.propertiese setting in backend

```
.oauth2Login(oauth2 -> oauth2
    .defaultSuccessUrl("http://localhost:3000/auth-callback", true)
    .userInfoEndpoint(userInfo -> userInfo
        .userAuthoritiesMapper(authorities -> authorities)
    )
    .successHandler((request, response, authentication) -> {
        Logger.debug("OAuth2 login success for user: {}, Session ID: {}",
            authentication.getName(), request.getSession().getId());
        request.getSession().setAttribute("SPRING_SECURITY_CONTEXT", SecurityContextHolder.getContext());
        response.addHeader("Set-Cookie", "JSESSIONID=" + request.getSession().getId() + "; Path=/; HttpOnly; SameSite=Lax");
        response.sendRedirect("http://localhost:3000/auth-callback");
    })
)
```

Figure 6 OAuth2 Service Setup in Security.Config

```
@Bean
public OAuth2AuthorizationRequestResolver authorizationRequestResolver(
        ClientRegistrationRepository clientRegistrationRepository) {
    final String authorizationRequestBaseUri = "/oauth2/authorization";
    DefaultOAuth2AuthorizationRequestResolver resolver =
        new DefaultOAuth2AuthorizationRequestResolver(clientRegistrationRepository, authorizationRequestBaseUri);

    resolver.setAuthorizationRequestCustomizer(customizer ->
        customizer.additionalParameters(params ->
            params.put("prompt", "select_account consent")
        )
    );
    return resolver;
}
```

Figure 7 Force Select Google Account Code

2. **Google Weather API Integration:**
    - **Frontend** (Home.js):
        - Uses navigator.geolocation.getCurrentPosition to obtain user's latitude and longitude.
        - Makes an Axios GET request to https://weather.googleapis.com/v1/currentConditions:lookup?key=API_KEY&location.latitude={lat}&location.longitude={lon}.
        - Displays weather data (e.g., temperature, humidity, precipitation) in a widget.
    - **Security Considerations**:
        - API key restricted to http://localhost:3000 in Google Cloud Console.
        - HTTPS enforced for API calls to prevent interception.
    - **Flow**:
        1. On page load, Home.js requests geolocation permission.
        2. If granted, sends coordinates to Weather API.
        3. Renders weather data in a styled card component using Tailwind CSS.

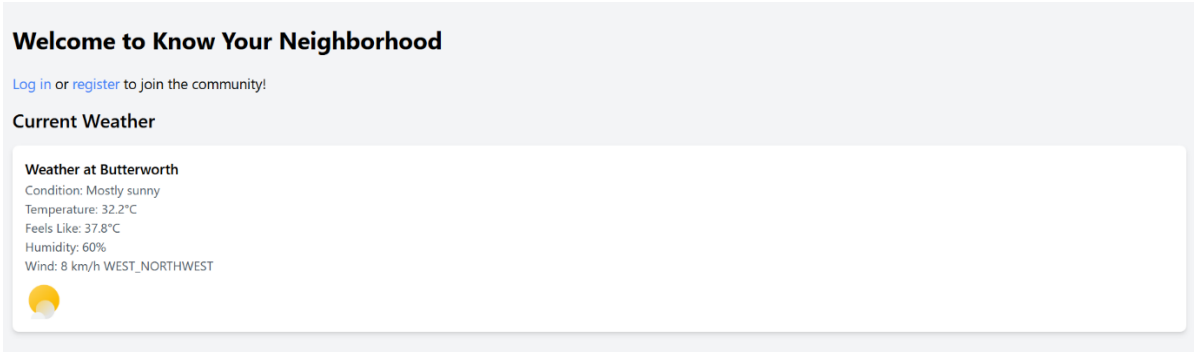

Figure 8 Show Weather in HomePage

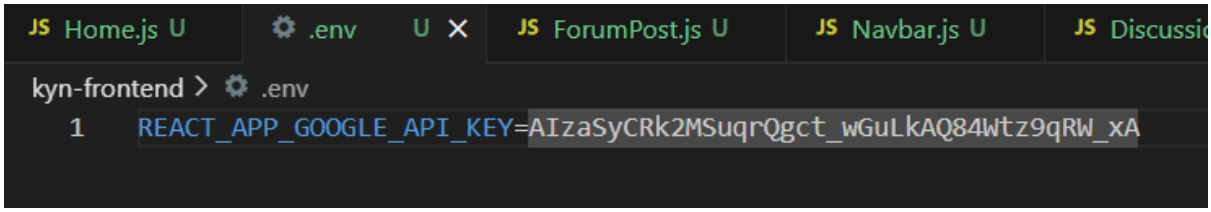

Figure 9 env file setup

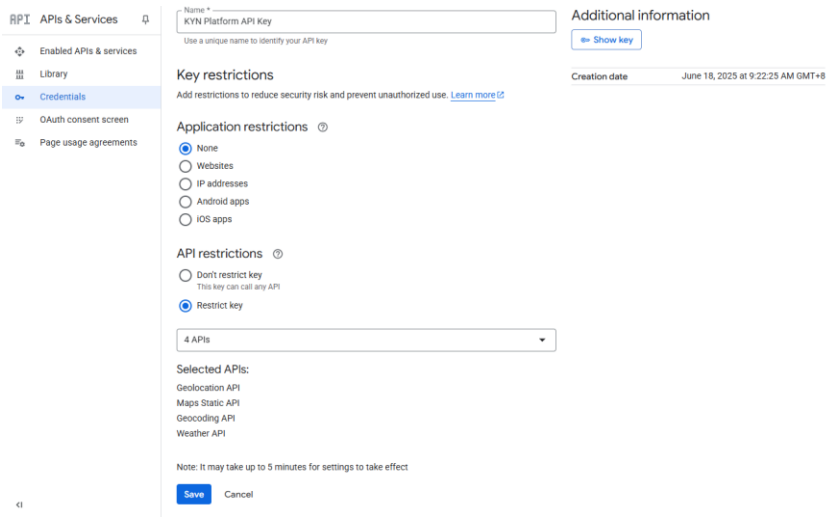

Figure 10 API Restriction Setting in Google Cloud Console

```
// Fetch user's location, city, and weather data
useEffect(() => {
  if (!process.env.REACT_APP_GOOGLE_API_KEY) {
    setWeatherError('API key is missing. Please check environment configuration.');
    setLoadingWeather(false);
    return;
  }
  if (navigator.geolocation) {
    setLoadingWeather(true);
    navigator.geolocation.getCurrentPosition(
      async (position) => {
        const { latitude, longitude } = position.coords;
        try {
          // Fetch city name using Google Maps Geocoding API
          const geocodeResponse = await axios.get(
            `https://maps.googleapis.com/maps/api/geocode/json?latlng=${latitude},${longitude}&key=${process.env.REACT_APP_GOOGLE_API_KEY}`
          );
          const results = geocodeResponse.data.results;
          let cityName = 'Unknown Location';
          if (results.length > 0) {
            // Look for locality or administrative_area_level_2 in address components
            for (const result of results) {
              const locality = result.address_components.find((component) =>
                component.types.includes('locality') ||
                component.types.includes('administrative_area_level_2')
              );
              if (locality) {
                cityName = locality.long_name;
                break;
              }
            }
          }
          setCity(cityName);
```

Figure 11 Fetch user's location, city, and weather data

3.5 Constructed Application

- **Architecture**:
    - **Backend**: Monolithic Spring Boot application running on Apache Tomcat, with MySQL as the database.
    - **Frontend**: React SPA with react-router-dom for navigation, styled with Tailwind CSS.
    - **APIs**: Internal REST APIs (/api/forums, /api/comments) and external APIs (Google OAuth2, Google Weather).
- **Key Components**:
    - **Backend**:
        - SecurityConfig.java: Configures OAuth2, CORS, and session management.
        - UserController.java: Manages user authentication and info retrieval.
        - ForumController.java and CommentController.java: Handle forum and comment CRUD operations.
        - GlobalExceptionHandler.java: Handles errors like ResourceNotFoundException.
    - **Frontend**:
        - App.js: Defines routes for all pages (/, /login, /forums, etc.).
        - AuthContext.js: Manages user state and session persistence.
        - Home.js: Integrates weather API and community feed.
        - Login.js and AuthCallback.js: Handle OAuth2 login flow.
        - DiscussionForum.js and ForumPost.js: Manage forum interactions.
    - **Database**:

        - MySQL tables created via Hibernate (spring.jpa.hibernate.ddl-auto=update).
        - Entity classes: User.java, Forum.java, Comment.java.

- **Deployment**:
    - Backend deployed on localhost:8080 using Apache Tomcat via STS.
    - Frontend deployed on localhost:3000 using npm start in Node.js.
- **Integration**:
    - Backend and frontend communicate via REST APIs with CORS enabled.
    - JSESSIONID cookies ensure session persistence across requests.
    - Google APIs integrated securely with credentials managed in Google Cloud Console.

3.6 Improvement of Application

- Improvement are been apply based on the suggestion of lecturer after project demonstration.

3.6.1 Pagination for Forum Page

- Add Pagination Login to display forum by page each page display 5 fourm.

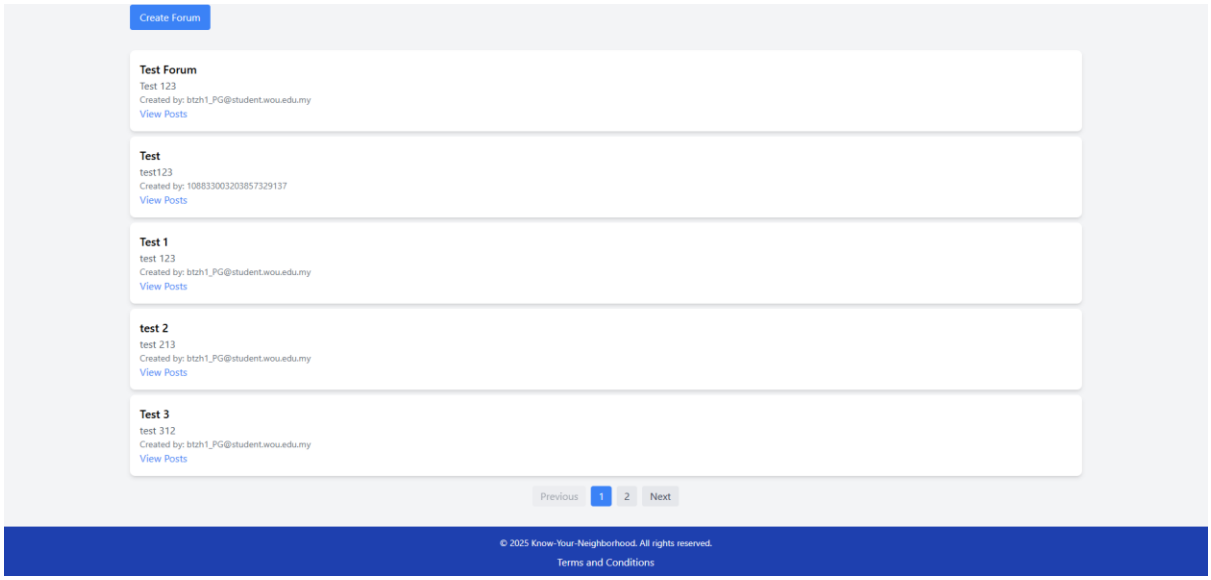

Figure 12 Output of Pagination Implementation in Forum Page

```
// Pagination logic
const indexOfLastForum = currentPage * forumsPerPage;
const indexOfFirstForum = indexOfLastForum - forumsPerPage;
const currentForums = forums.slice(indexOfFirstForum, indexOfLastForum);
const totalPages = Math.ceil(forums.length / forumsPerPage);

const handlePageChange = (pageNumber) => {
  setCurrentPage(pageNumber);
};

const handleNextPage = () => {
  if (currentPage < totalPages) {
    setCurrentPage(currentPage + 1);
  }
};

const handlePrevPage = () => {
  if (currentPage > 1) {
    setCurrentPage(currentPage - 1);
  }
};
```

Figure 13 Pagination Logic Code

3.6.2 Homepage Weather Display by City Name

- Modified the Home.js code to retrieve City Address and display it.

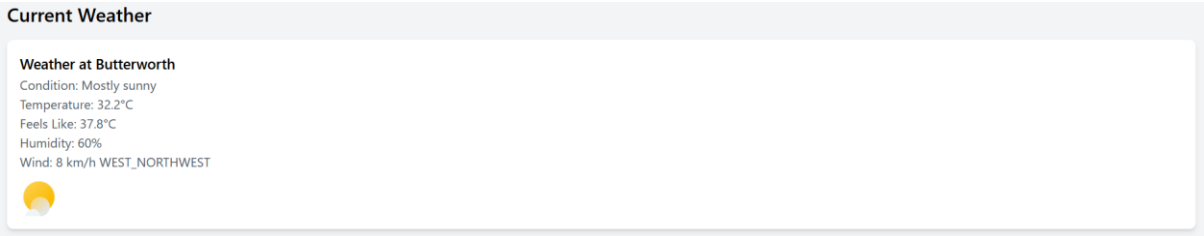

Figure 14 Output of Display Weather by City Name in Home Page

```
// Fetch user's location, city, and weather data
useEffect(() => {
  if (!process.env.REACT_APP_GOOGLE_API_KEY) {
    setWeatherError('API key is missing. Please check environment configuration.');
    setLoadingWeather(false);
    return;
  }
  if (navigator.geolocation) {
    setLoadingWeather(true);
    navigator.geolocation.getCurrentPosition(
      async (position) => {
        const { latitude, longitude } = position.coords;
        try {
          // Fetch city name using Google Maps Geocoding API
          const geocodeResponse = await axios.get(
            `https://maps.googleapis.com/maps/api/geocode/json?latlng=${latitude},${longitude}&key=${process.env.REACT_APP_GOOGLE_API_KEY}`
          );
          const results = geocodeResponse.data.results;
          let cityName = 'Unknown Location';
          if (results.length > 0) {
            // Look for locality or administrative_area_level_2 in address components
            for (const result of results) {
              const locality = result.address_components.find((component) =>
                component.types.includes('locality') ||
                component.types.includes('administrative_area_level_2')
              );
              if (locality) {
                cityName = locality.long_name;
                break;
              }
            }
          }
          setCity(cityName);
```

Figure 15 Fetch Address and Weather Information Code Logic

4.0 Application Testing

4.1 White Box Testing

White box testing was performed to verify the internal logic and structure of the backend APIs in UserController, ForumController, and CommentController. The tests focused on key functionalities: user registration, login, forum retrieval, and comment creation.

4.1.1 Test W001 - User Registration Success:

- Description: Test successful user registration with valid input.
- Input: JSON payload with email="new@example.com", name="New User", password="password".
- Expected Output: HTTP 200, JSON response with user details (email, name).
- Result: PASS

```
@Test
public void testUserRegistrationSuccess() throws Exception {
    User user = new User("new@example.com", "New User", "$2a$10$exampleHash");
    when(userRepository.findById("new@example.com")).thenReturn(Optional.empty());
    when(userRepository.save(any(User.class))).thenReturn(user);

    MvcResult result = mockMvc.perform(post("/api/user/register")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content("{\"email\":\"new@example.com\",\"name\":\"New User\",\"password\":\"password\"}"))
            .andExpect(status().isOk())
            .andReturn();

    System.out.println("Response: " + result.getResponse().getContentAsString());
    mockMvc.perform(post("/api/user/register")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content("{\"email\":\"new@example.com\",\"name\":\"New User\",\"password\":\"password\"}"))
            .andExpect(jsonPath("$.email").value("new@example.com"))
            .andExpect(jsonPath("$.name").value("New User"));
}
```

Figure 16 WOO1 - User Registration Success

4.1.2 Test W002 - User Registration with Duplicate Email:

- Description: Test registration with an existing email.
- Input: JSON payload with email="test@example.com", name="Test User", password="password".
- Expected Output: HTTP 409, error message "User already exists".
- Result: PASS

```
@Test
public void testUserRegistrationDuplicateEmail() throws Exception {
    User user = new User("test@example.com", "Test User", "$2a$10$exampleHash");
    when(userRepository.findById("test@example.com")).thenReturn(Optional.of(user));

    mockMvc.perform(post("/api/user/register")
                .contentType(MediaType.APPLICATION_JSON)
                .content("{\"email\":\"test@example.com\",\"name\":\"Test User\",\"password\":\"password\"}"))
            .andExpect(status().isConflict())
            .andExpect(jsonPath("$").value("User already exists"));
}
```

Figure 17 W002 - User Registration with Duplicated Email

4.1.3 Test W003 - User Login Success:

- Description: Test successful login with valid credentials.
- Input: JSON payload with email="test@example.com", password="password".
- Expected Output: HTTP 200, JSON response with user details.
- Result: PASS

```
@Test
public void testUserLoginSuccess() throws Exception {
    User user = new User("test@example.com", "Test User", "$2a$10$exampleHash");
    when(userRepository.findById("test@example.com")).thenReturn(Optional.of(user));
    when(authenticationManager.authenticate(any(UsernamePasswordAuthenticationToken.class)))
            .thenReturn(new UsernamePasswordAuthenticationToken("test@example.com", null));

    mockMvc.perform(post("/api/user/login")
                .contentType(MediaType.APPLICATION_JSON)
                .content("{\"email\":\"test@example.com\",\"password\":\"password\"}"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.email").value("test@example.com"))
            .andExpect(jsonPath("$.name").value("Test User"));
}
```

Figure 18 W003 - User Login Success

4.1.4 Test W004 - User Login Failure:

- Description: Test login with invalid credentials.
- Input: JSON payload with email="test@example.com", password="wrong".
- Expected Output: HTTP 401, error message "Invalid email or password".
- Result: PASS

```
@Test
public void testUserLoginFailure() throws Exception {
    when(authenticationManager.authenticate(any(UsernamePasswordAuthenticationToken.class)))
            .thenThrow(new org.springframework.security.core.AuthenticationException("Invalid credentials") {});

    mockMvc.perform(post("/api/user/login")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content("{\"email\":\"test@example.com\",\"password\":\"wrong\"}"))
            .andExpect(status().isUnauthorized())
            .andExpect(jsonPath("$").value("Invalid email or password"));
}
```

Figure 19 W004 - User Login Failure

4.1.5 Test W005 - Get All Forums:

- Description: Test retrieval of all forums.
- Input: GET request to /api/forums.
- Expected Output: HTTP 200, JSON array of forums.
- Result: PASS

```
@Test
public void testGetAllForums() throws Exception {
    Forum forum = new Forum();
    forum.setId(1L);
    forum.setTitle("Test Forum");
    forum.setDescription("Description");
    forum.setCreatedBy("test@example.com");
    forum.setCreatedAt(LocalDateTime.now());

    when(forumRepository.findAll()).thenReturn(Arrays.asList(forum));

    mockMvc.perform(get("/api/forums")
                    .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].title").value("Test Forum"));
}
```

Figure 20 W005 - Get All Forums

4.1.6 Test W006 - Get Forum by ID(Not Found):

- Description: Test retrieval of a non-existent forum.
- Input: GET request to /api/forums/1.
- Expected Output: HTTP 404, JSON response with message="Forum not found with ID: 1".
- Result: PASS

```
@Test
public void testGetForumByIdNotFound() throws Exception {
    when(forumRepository.findById(1L)).thenReturn(Optional.empty());

    mockMvc.perform(get("/api/forums/1")
                    .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound())
            .andExpect(jsonPath("$.message").value("Forum not found with ID: 1"));
}
```

Figure 21 W006 - Get Forum by ID (Not Found)

4.1.7 Test W007 – Get Comment Success:

- Description: Test comment creation for an existing forum by an authenticated user.
- Input: POST request to /api/comments/forum/1 with content="Test comment".
- Expected Output: HTTP 200, JSON response with comment details (content, createdByName).
- Result: PASS

```
@Test
public void testCreateCommentSuccess() throws Exception {
    Forum forum = new Forum();
    forum.setId(1L);
    Comment comment = new Comment();
    comment.setId(1L);
    comment.setContent("Test comment");
    comment.setForum(forum);
    comment.setCreatedBy("test@example.com");
    comment.setCreatedAt(LocalDateTime.now());
    comment.setCreatedByName("Test User");

    when(forumRepository.findById(1L)).thenReturn(Optional.of(forum));
    when(commentRepository.save(any(Comment.class))).thenReturn(comment);
    when(userRepository.findById("test@example.com")).thenReturn(Optional.of(new User("test@example.com", "Test User", null)));

    mockMvc.perform(post("/api/comments/forum/1")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content("{\"content\":\"Test comment\"}"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.content").value("Test comment"))
            .andExpect(jsonPath("$.createdByName").value("Test User"));
}
```

Figure 22 W007 - Get Comment Success

4.1.8 Test W008 – Create Comment Unauthorized:

- Description: Test comment creation by an unauthenticated user.
- Input: POST request to /api/comments/forum/1 with content="Test comment" and anonymous user.
- Expected Output: HTTP 401, JSON response with message="Not authenticated".
- Result: PASS

```
@Test
public void testCreateCommentUnauthorized() throws Exception {
    Authentication anonymousAuth = new UsernamePasswordAuthenticationToken("anonymousUser", null);
    SecurityContext securityContext = mock(SecurityContext.class);
    when(securityContext.getAuthentication()).thenReturn(anonymousAuth);
    SecurityContextHolder.setContext(securityContext);

    mockMvc.perform(post("/api/comments/forum/1")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content("{\"content\":\"Test comment\"}"))
            .andExpect(status().isUnauthorized())
            .andExpect(jsonPath("$.message").value("Not authenticated"));
}
```

Figure 23 W008 - Create Comment Unauthorized

4.2 Black Box Testing

4.2.1 Test ID: B001

- **Test Case:** User Registration
- **Test Description/Step:**
    1. Navigate to register page
    2. Enter full name, email, password
    3. Click on “Register” Button
- **Test Data:**
    - **Input:**
        - name= New User
        - email=new@example.com
        - password=password123
    - **Output:**
        - Display a success message
- **Expected Result:**
    - Registration Successful message, redirect to login
- **Actual Result:**
    - Success message displayed, redirect to login page
- **PASS/FAIL:** PASS

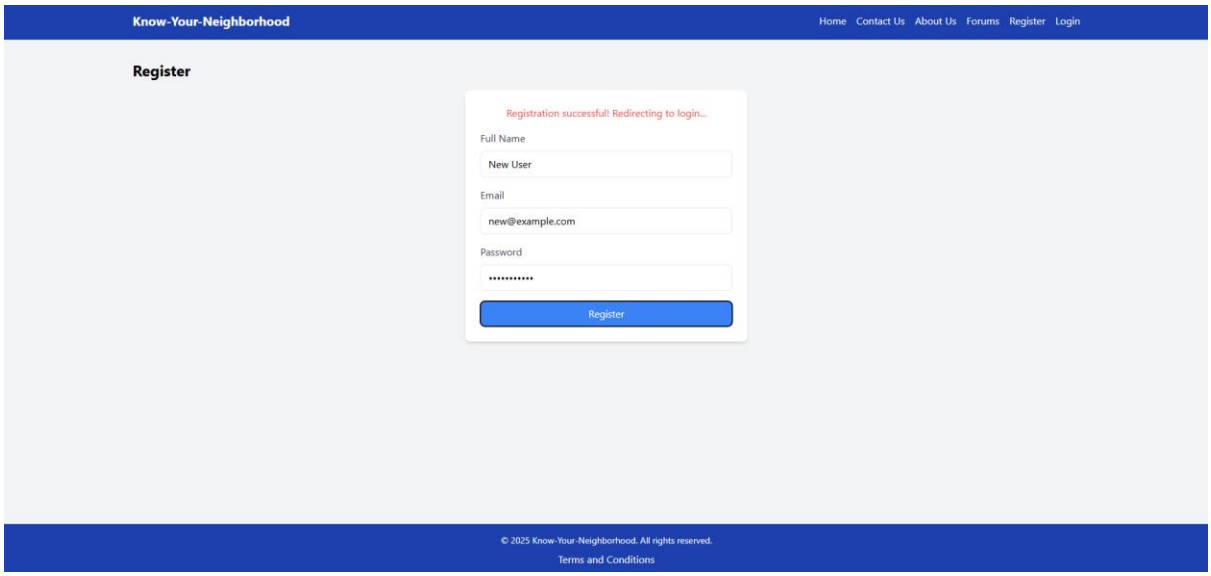

Figure 24 B001 - User Registration

4.2.2 Test ID: B002

- **Test Case:** User sees error on duplicate registration
- **Test Description/Step:**
    1. Navigate to register page
    2. Enter full name, email, password
    3. Click on “Register” Button
- **Test Data:**
    - **Input:**
        - name= New User
        - email=new@example.com
        - password=password123
    - **Output:**
        - Display a error message
- **Expected Result:**
    - Error message: Registration failed: User already exists
- **Actual Result:**
    - Error message displayed
- **PASS/FAIL:** PASS

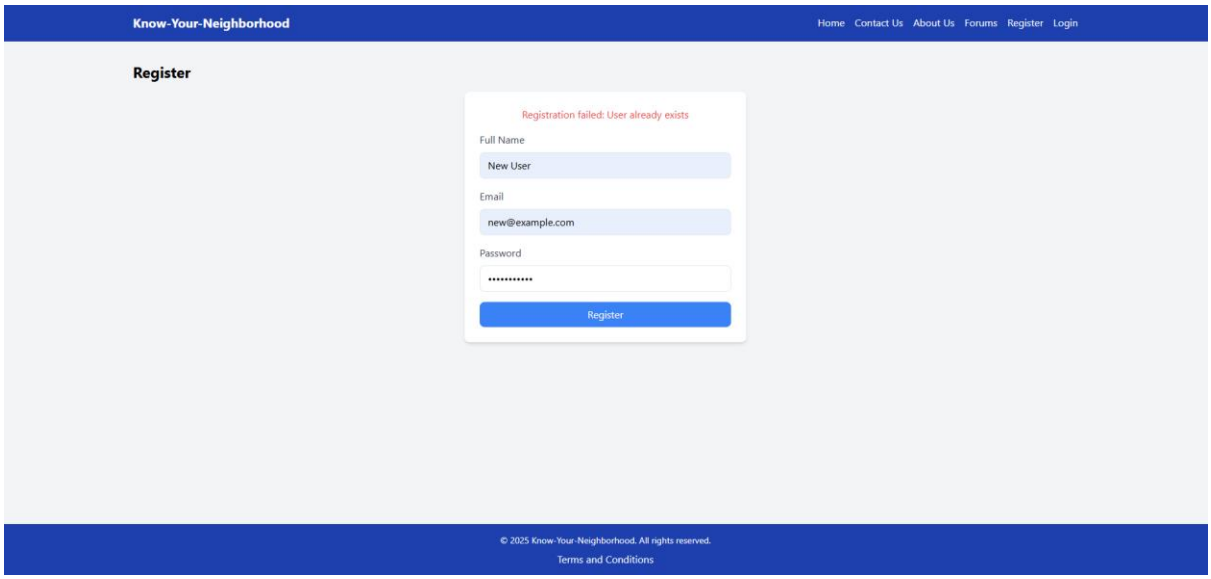

Figure 25 B002 - User sees error on duplicate registration

4.2.3 Test ID: B003

- **Test Case:** User can log in successfully
- **Test Description/Step:**
    1. Navigate to login page
    2. Enter email and password
    3. Click on “Login” Button
- **Test Data:**
    - **Input:**
        - email=new@example.com
        - password=password123
    - **Output:**
        - Display a success message
- **Expected Result:**
    - Login Successful message, redirect to home page
- **Actual Result:**
    - Success message displayed redirect to home page
- **PASS/FAIL:** PASS

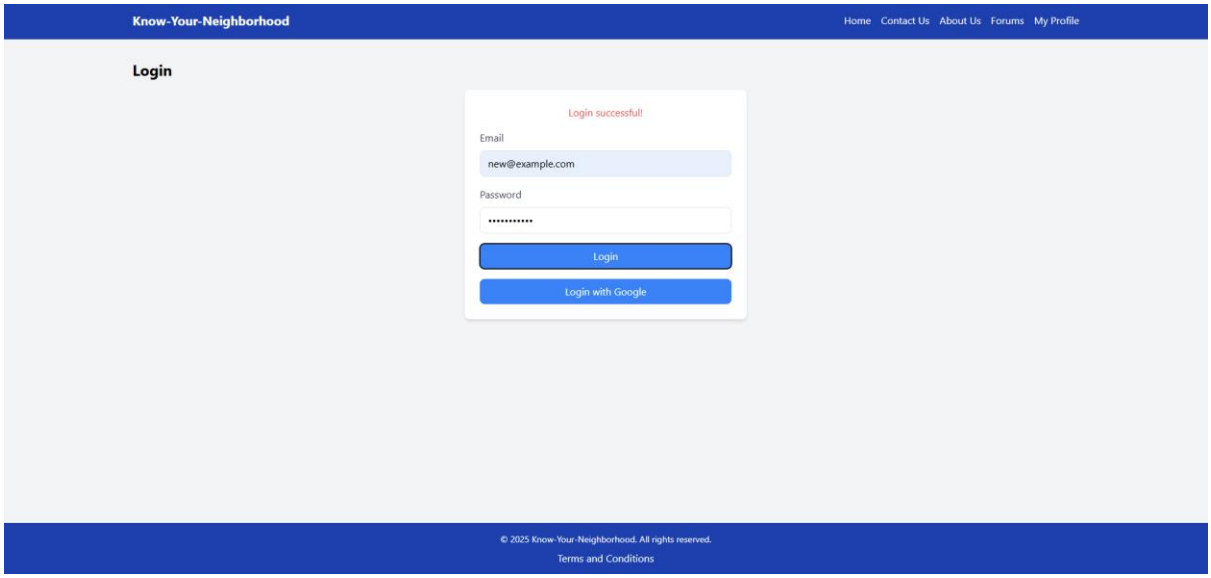

Figure 26 B003 - User can log in successfully

4.2.3 Test ID: B004

- **Test Case:** User can post a forum comment
- **Test Description/Step:**
    1. Log in as registered user
    2. Navigate to forum -> View Posts
    3. Enter comment text
    4. Click “Post” Comment Button
- **Test Data:**
    - **Input:**
        - Comment=Test Comment
    - **Output:**
        - Comment displayed
- **Expected Result:**
    - Comment displayed with user name and timestamp
- **Actual Result:**
    - Comment “Test Comment” display with “Posted by: New User”
- **PASS/FAIL:** PASS

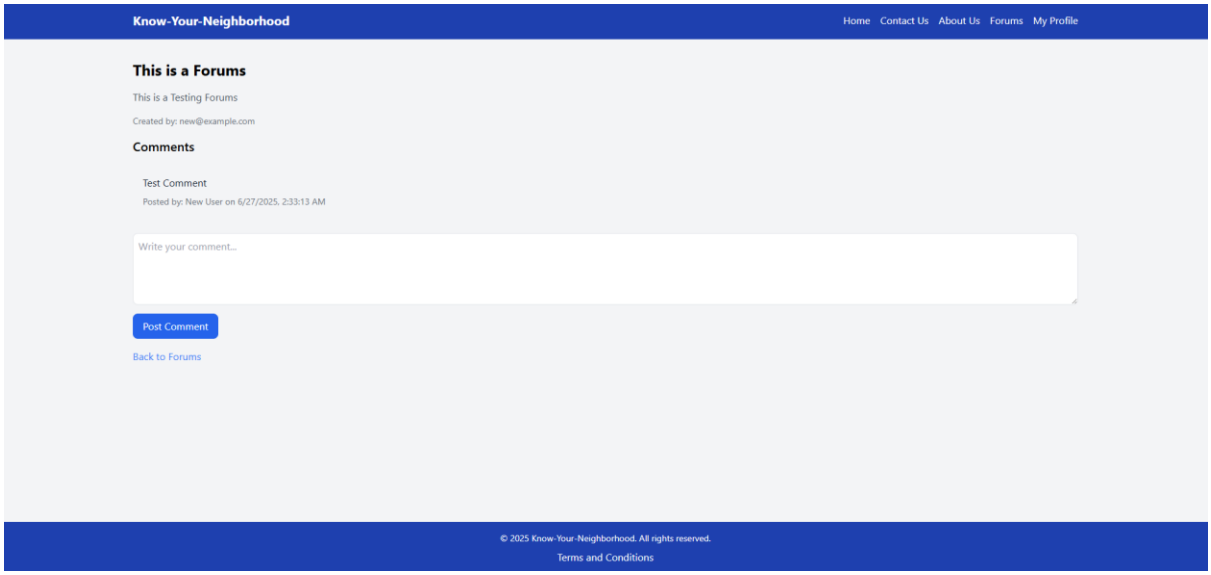

Figure 27 B004 - User can post a forum comment

5.0 Review and Reflect on the APIs Used

5.1 Review of developed APIs

5.1.1 Google OAuth2 API:

- **Strengths:**
  - Secure authentication with minimal user input
  - Industry-standard, reliable, and scalable
- **Weakness:**
  - Dependency on Google Service
  - Complex token management for refresh tokens

5.1.2 Google Weather API:

- **Strengths:**
  - Hyperlocal, accurate weather data
  - Easy integration with geolocation
- **Weakness:**
  - API key exposure risk if not secured
  - Cost implications for high usage

6.0 Conclusion

The development of the Know-Your-Neighbourhood application successfully integrated Google OAuth2 and Google Weather APIs, achieving secure user authentication and location-based weather updates. The use of Spring Boot for the backend and React for the frontend ensured a robust and dynamic single-page application. The project met all requirements, including secure login, user data retrieval, and real-time weather data display, while maintaining compatibility with existing features like forums and user profiles. Despite challenges such as potential vendor lock-in and API key security, the application delivers a seamless user experience across supported web browsers. This project demonstrates the effective application of API integration, Spring Security, and modern frontend frameworks to build a community-oriented platform. Future enhancements could include microservices for scalability or additional APIs for extended functionality.

7.0 References

Axios (2023). Getting Started | Axios Docs. [online] axios-http.com. Available at: https://axios-http.com/docs/intro [Accessed 13 Jun. 2025].

React (n.d.). Built-in React Hooks. [online] react.dev. Available at: https://react.dev/reference/react [Accessed 13 Jun. 2025].

React Router (2024). React Router Home | React Router. [online] Reactrouter.com. Available at: https://reactrouter.com/home [Accessed 13 Jun. 2025].

Tailwind (2025). Installing with Vite - Installation. [online] Tailwindcss.com. Available at: https://tailwindcss.com/docs/installation/using-vite [Accessed 13 Jun. 2025].

NPM (2024). *@greatsumini/react-facebook-login*. [online] npm. Available at: https://www.npmjs.com/package/@greatsumini/react-facebook-login [Accessed 20 Jun. 2025].

Google Cloud Developer (2025). *Weather API overview*. [online] Google for Developers. Available at: https://developers.google.com/maps/documentation/weather/overview [Accessed 23 Jun. 2025].

Mohamed, S. (2024). *Implementing 'Sign in with Google' in Spring Boot Application*. [online] Medium. Available at: https://medium.com/@sallu-salman/implementing-sign-in-with-google-in-spring-boot-application-5f05a34905a8 [Accessed 23 Jun. 2025]