CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342-G1 SYSTEMS INTEGRATION AND ARCHITECTURE 1

FUNCTIONAL REQUIREMENTS SPECIFICATION (FRS)

Project Title: Mini App - User Registration & Authentication

Prepared By: Xavier John A. Sabornido

Date of Submission: January 31, 2026

Version: 1

Table of Contents

● Introduction

●.1. Purpose

The purpose of this system is to allow users to create an account, log in securely, access protected pages such as a profile or dashboard, and log out safely. It ensures that only authenticated users can access private features of the system.

●.2. Scope

The system provides basic authentication features including user registration, login, profile/dashboard viewing, and logout. It includes account validation, password protection, and session control. The system only focuses on authentication and access control and does not include advanced features such as payments, messaging, or external integrations.

●.3. Definitions, Acronyms, and Abbreviations

- **Authentication** — Process of verifying a user's identity
- **Authorization** — Process of granting access to protected resources
- **User Account** — A registered identity in the system
- **Session** — A temporary login state after successful authentication
- **Password Hash** — Encrypted version of a password stored in database
- **Dashboard** — Main protected page shown after login
- **ERD** — Entity Relationship Diagram
- **UML** — Unified Modeling Language

● Overall Description

●.1. System Perspective

The system is a standalone authentication module that can be integrated into a larger web or mobile application. It acts as the access control layer that manages user identity and protects restricted pages.

●.2. User Classes and Characteristics

Guest User

- Not logged in
- Can register an account
- Can log in
- Cannot access protected pages

Authenticated User

- Logged in with valid credentials

- Can access dashboard/profile
- Can log out
- Can view protected content

●.1. Operating Environment

The system can operate in the following environment:

- Web browser or mobile browser
- Application server (React, Spring Boot, Kotlin, etc.)
- Database server (MySQL, PostgreSQL, Supabase, etc.)
- Development tools such as draw.io and IDEs
- Runs on Windows

●.1. Assumptions and Dependencies

- Users have internet access
- Users provide valid email and password data
- Database server is available and running
- Password encryption library is available
- Session/token mechanism is implemented
- Browser supports cookies or tokens for session handling

● System Features and Functional Requirements

Describe each major feature of the system and its functional requirements.

●.1. Feature 1: **User Registration**

Description: Allows a new user to create an account by submitting required information.
Functional Requirements:

- User can enter username, email, and password
- System validates required fields and format
- System checks for duplicate username or email
- System encrypts password before storing
- System saves user record to database
- System confirms successful registration

●.2. Feature 2: User Login

Description: Allows a registered user to log in using valid credentials.
Functional Requirements:

- User can enter username/email and password
- System verifies credentials against database
- System compares encrypted passwords
- System creates authenticated session/token
- System redirects user to dashboard on success
- System shows error message on failure

●.3. Feature 3: View Dashboard / Profile

Description: Allows authenticated users to view protected pages.
Functional Requirements:

- System checks authentication before access
- System loads user profile data
- System blocks access if not logged in
- System redirects guest users to login page

●.4. Feature 4: **Logout**

Description: Allows a logged-in user to safely end their session.
Functional Requirements:

- User can click logout button
- System destroys session/token
- System clears authentication state
- System redirects to login page
- Protected pages become inaccessible

● Non-Functional Requirements

Performance

- Login and registration responses should complete within 2 seconds
- Database queries should be optimized

Security

- Passwords must be hashed
- Sessions must be securely stored
- Protected pages require authentication check
- Input validation must be enforced

Usability

- Forms should be simple and easy to use
- Error messages should be clear
- Navigation should be straightforward

Reliability

- System should handle invalid inputs safely
- System should not crash on failed login attempts

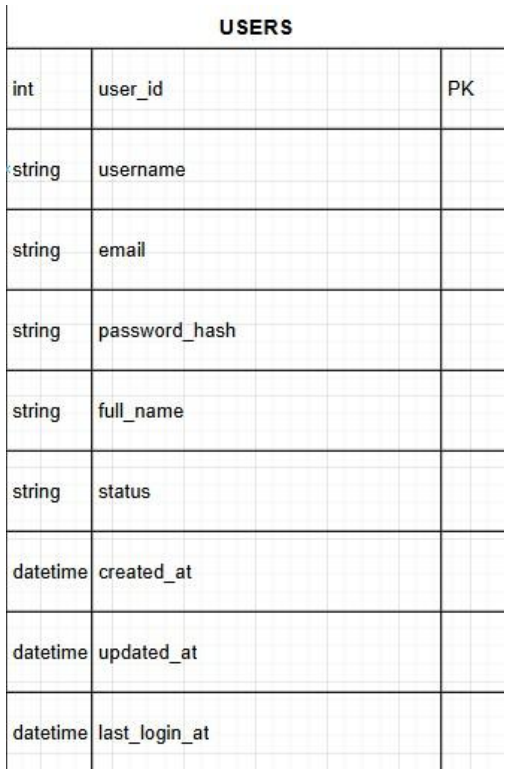
Availability

- System should be available whenever the server and database are running

● System Models (Diagrams)

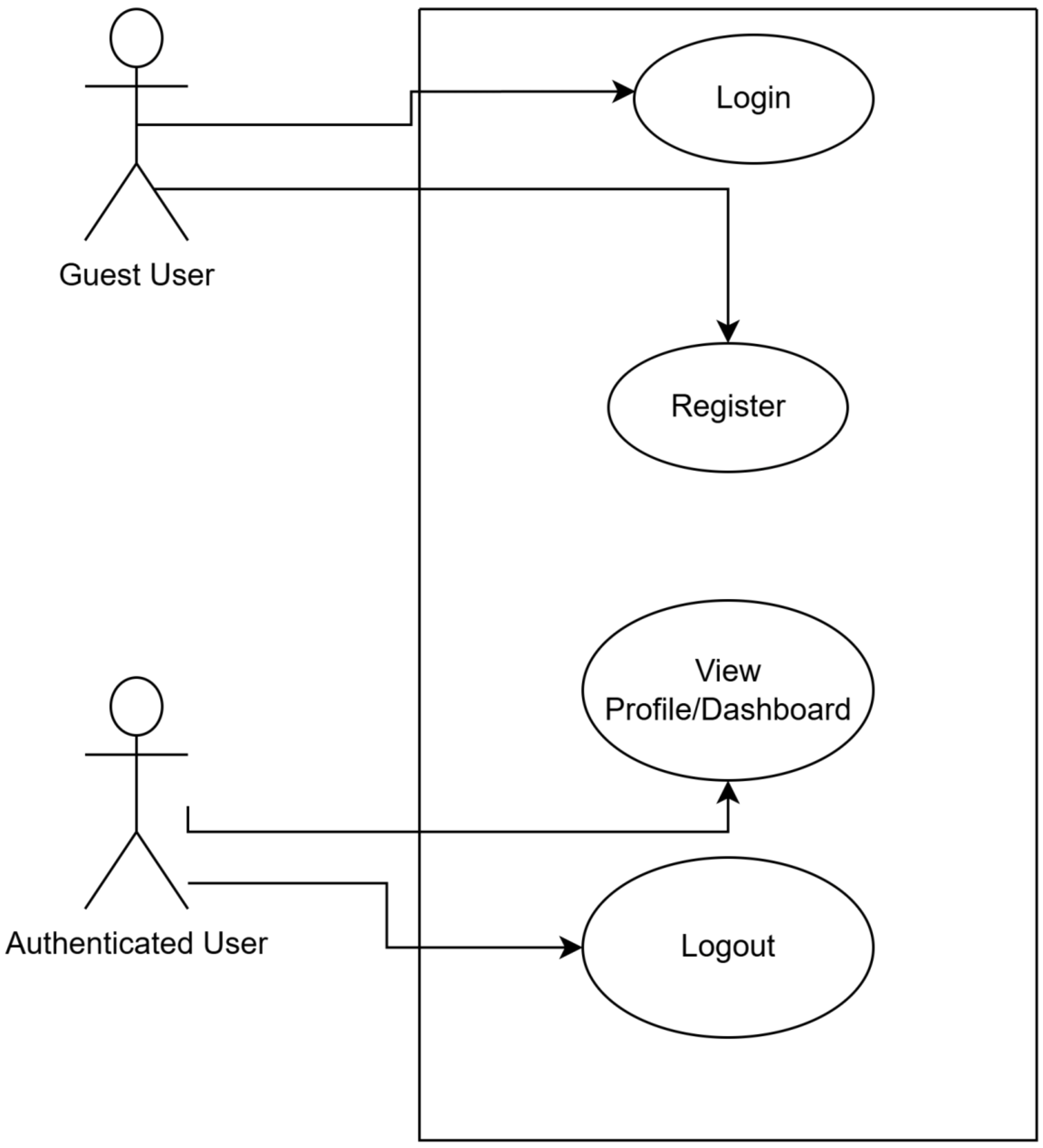
Insert the necessary diagrams for the system:

●.1. ERD

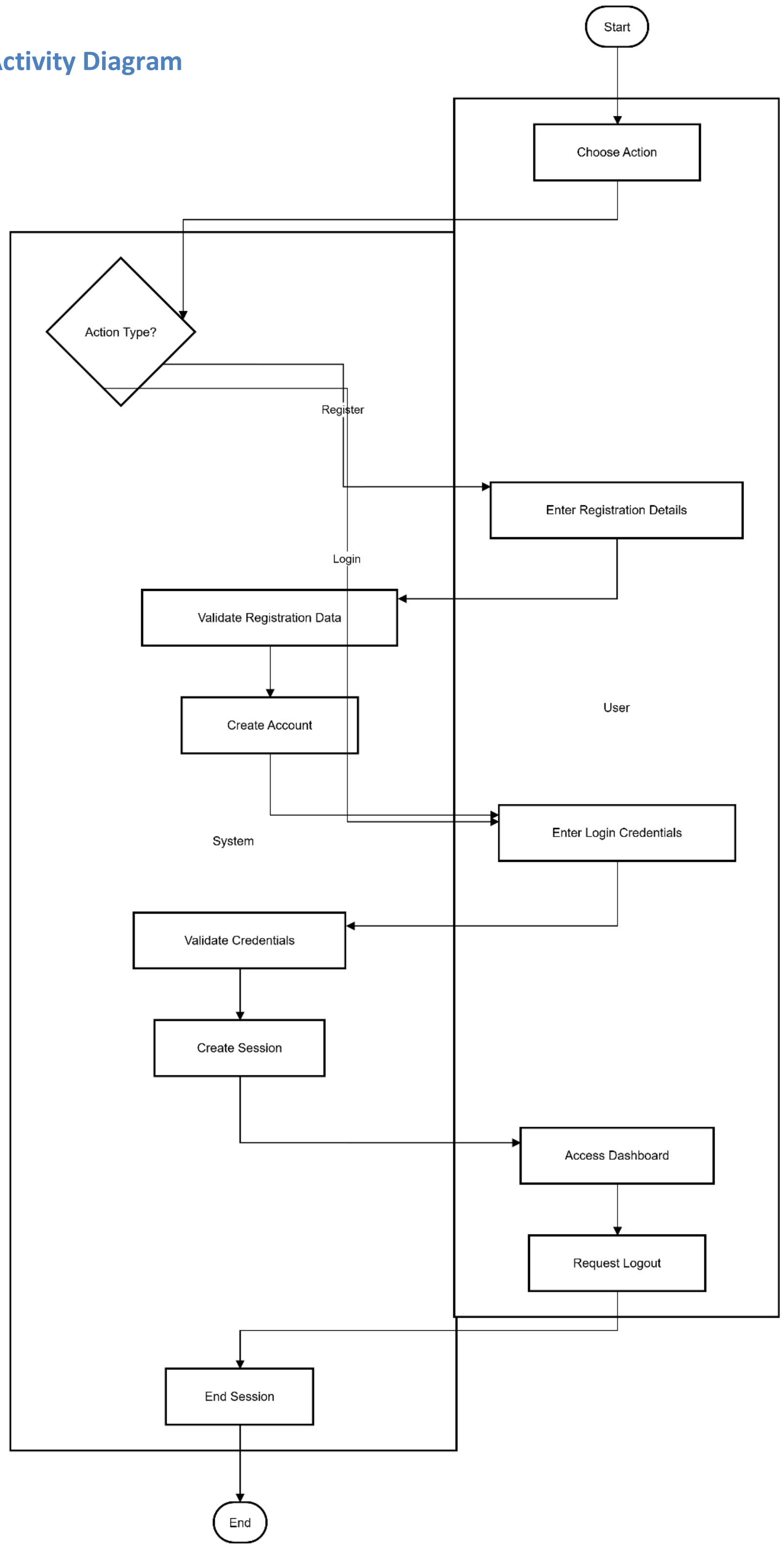


USERS		
int	user_id	PK
string	username	
string	email	
string	password_hash	
string	full_name	
string	status	
datetime	created_at	
datetime	updated_at	
datetime	last_login_at	

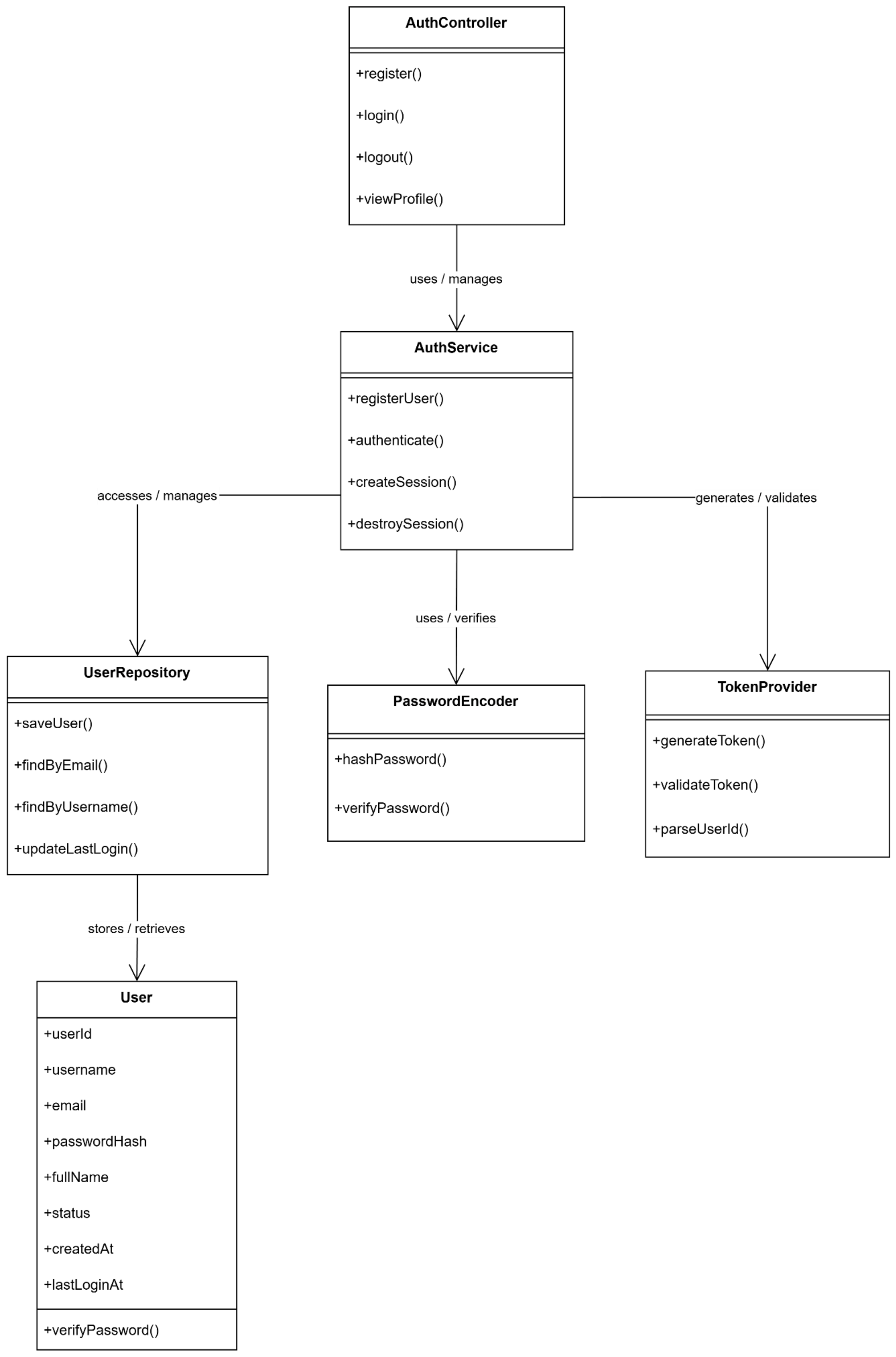
●.2. Use Case Diagram



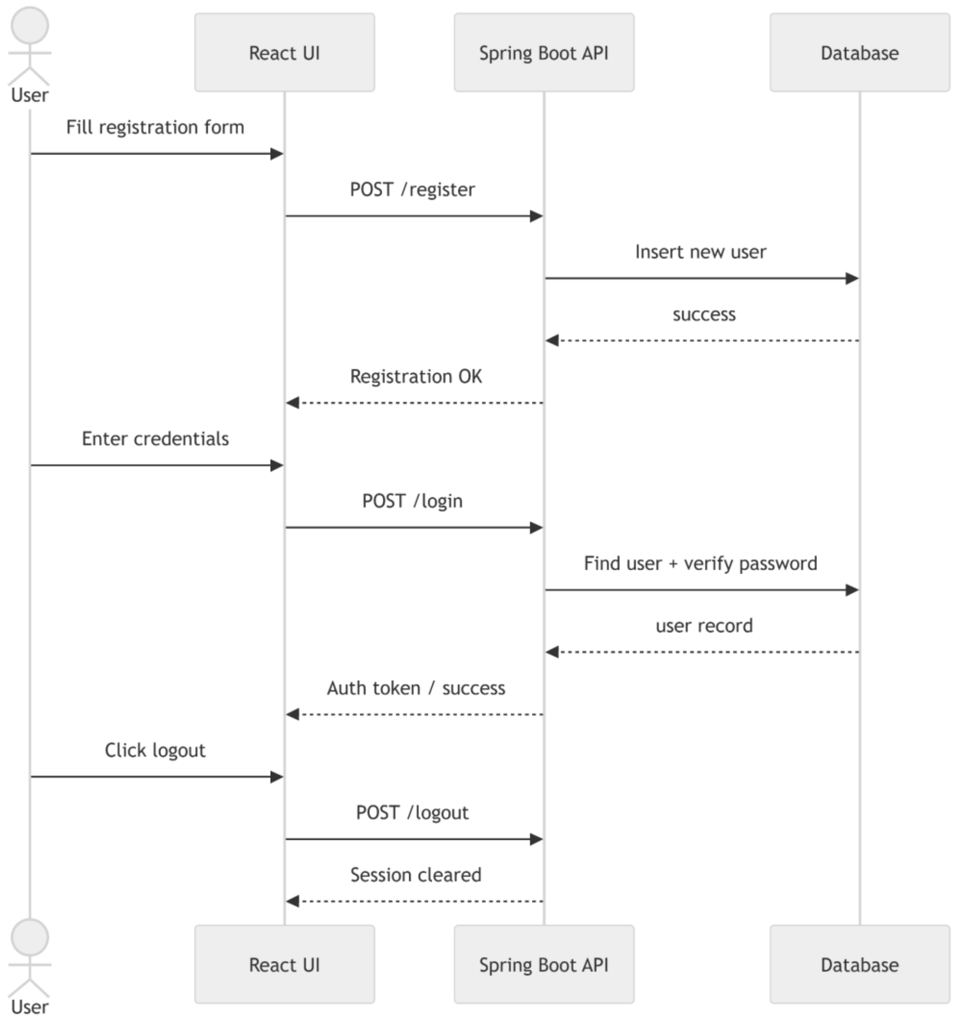
●.3. Activity Diagram



●.4. Class Diagram



●.5. Sequence Diagram



- **Appendices**

Appendix A — Diagrams Included

The following system design diagrams are attached to support this document:

- Entity Relationship Diagram (ERD) — Users table schema
- Use Case Diagram — Guest and Authenticated user actions
- Activity Diagram — Registration, Login, and Logout process flow
- Class Diagram — Backend authentication structure
- Sequence Diagram — Interaction flow for authentication processes

These diagrams visually describe the system behavior, structure, and data design.

Appendix B — Tools Used

The following tools were used in preparing the documentation and diagrams:

- draw.io — Diagram creation
- Mermaid — Diagram code generation
- UML modeling standards — Diagram structure guide
- Word/Google Docs — Documentation formatting

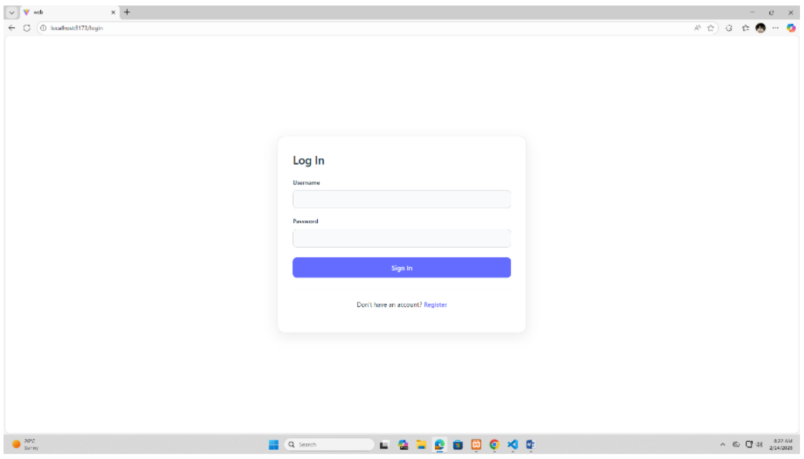
Appendix C — Authentication Rules Summary

- Only registered users can log in

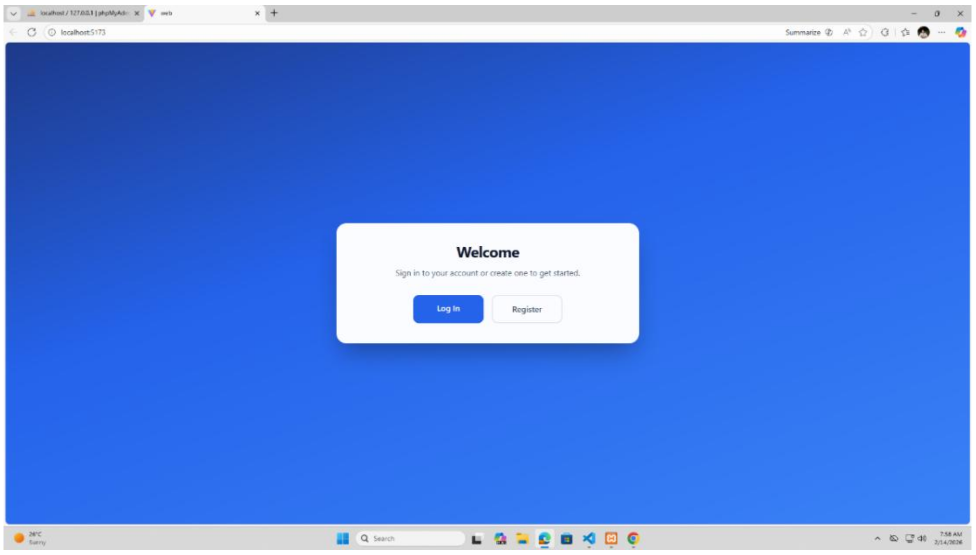
- Passwords are stored in hashed form
- Sessions/tokens are required to access protected pages
- Logged-out users cannot access dashboard pages
- Duplicate email/username is not allowed

Web Screen Shots

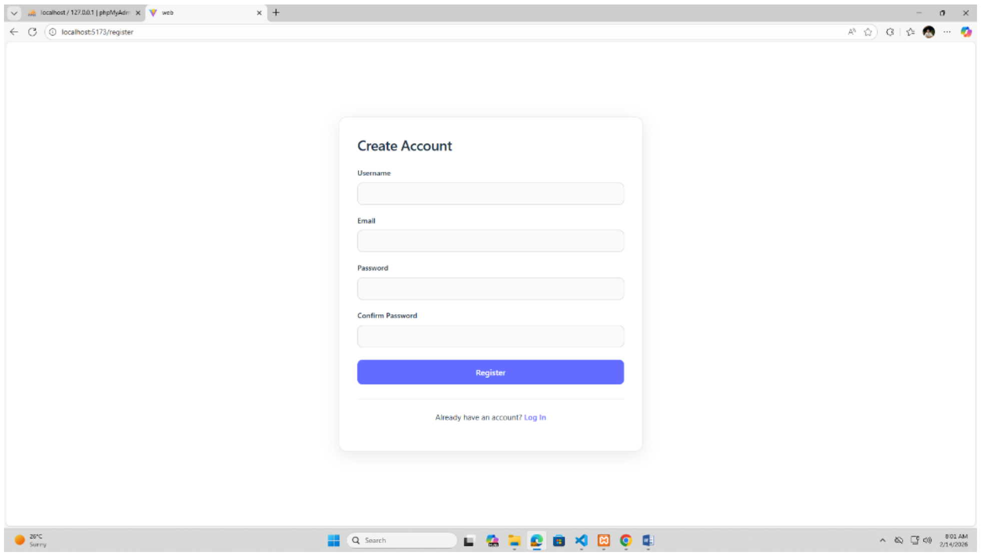
- Login Page



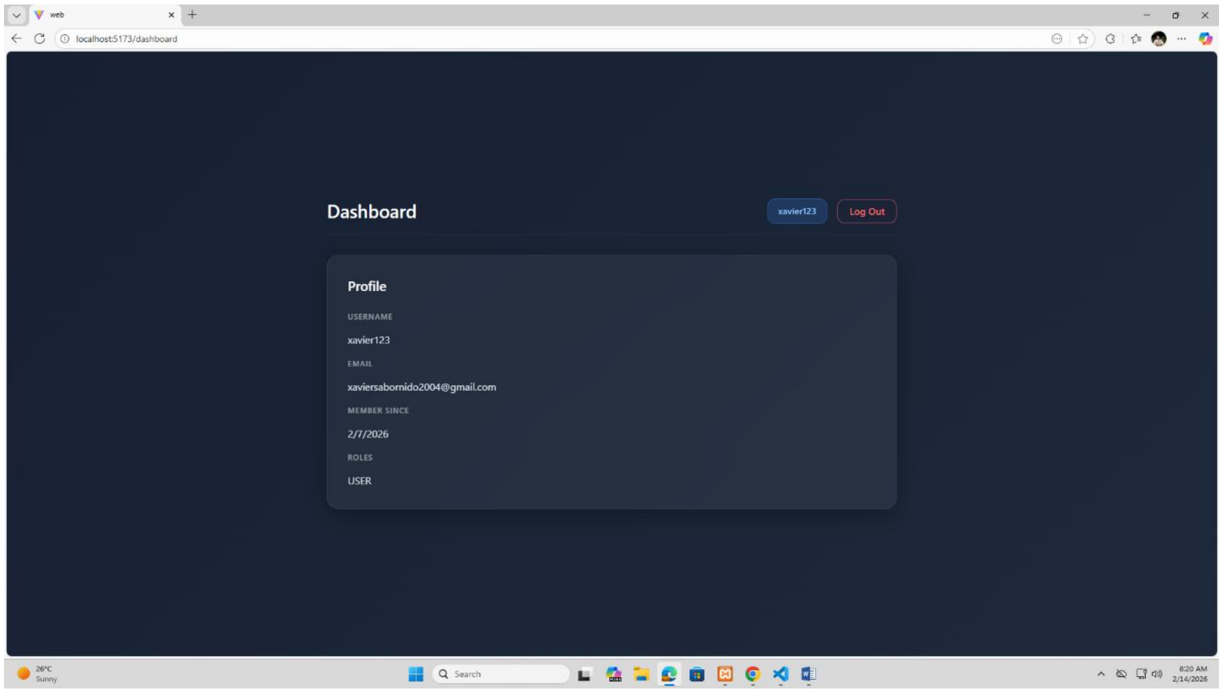
- Home Page



- Registration Page



- Dashboard



USERNAME

xavier123

EMAIL

xaviersabornido2004@gmail.com

MEMBER SINCE

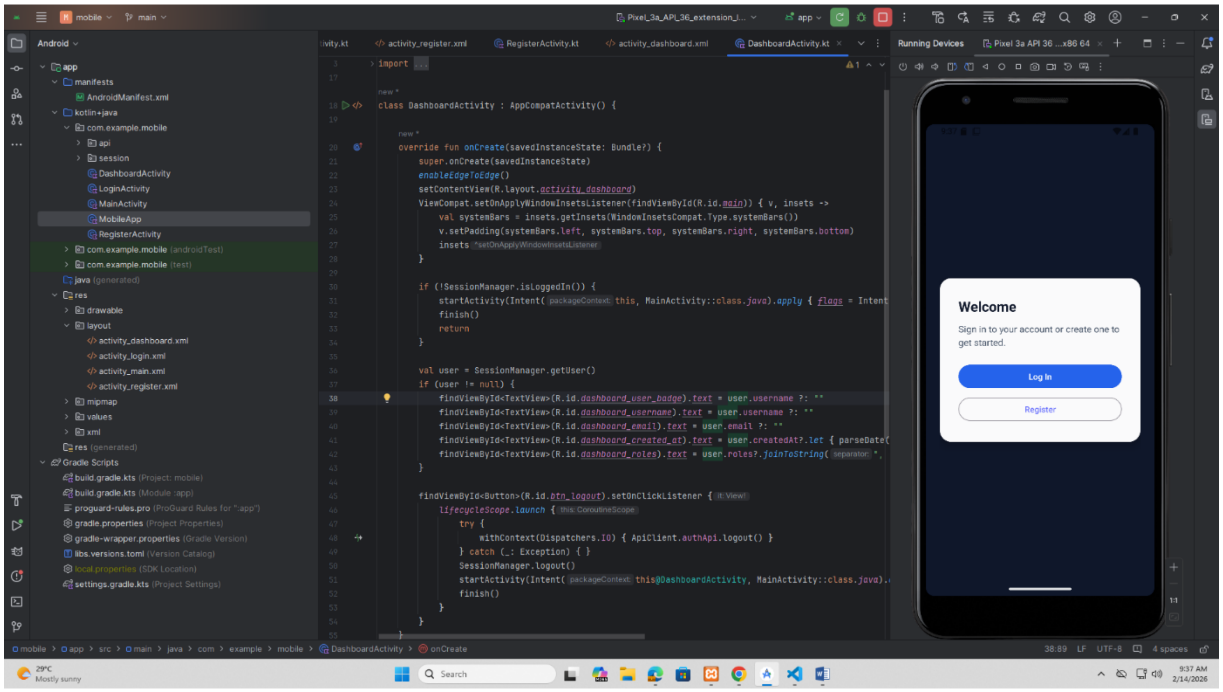
2/7/2026

ROLES

USER

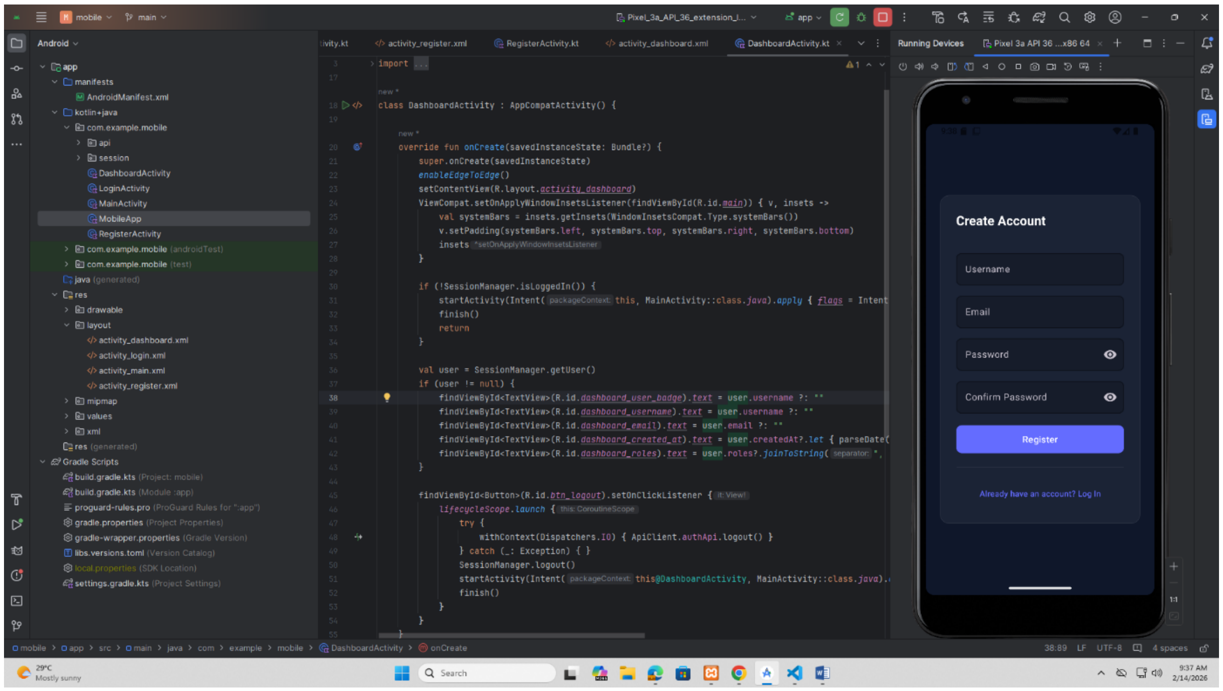
- ## Mobile Screen Shots

- Home Page



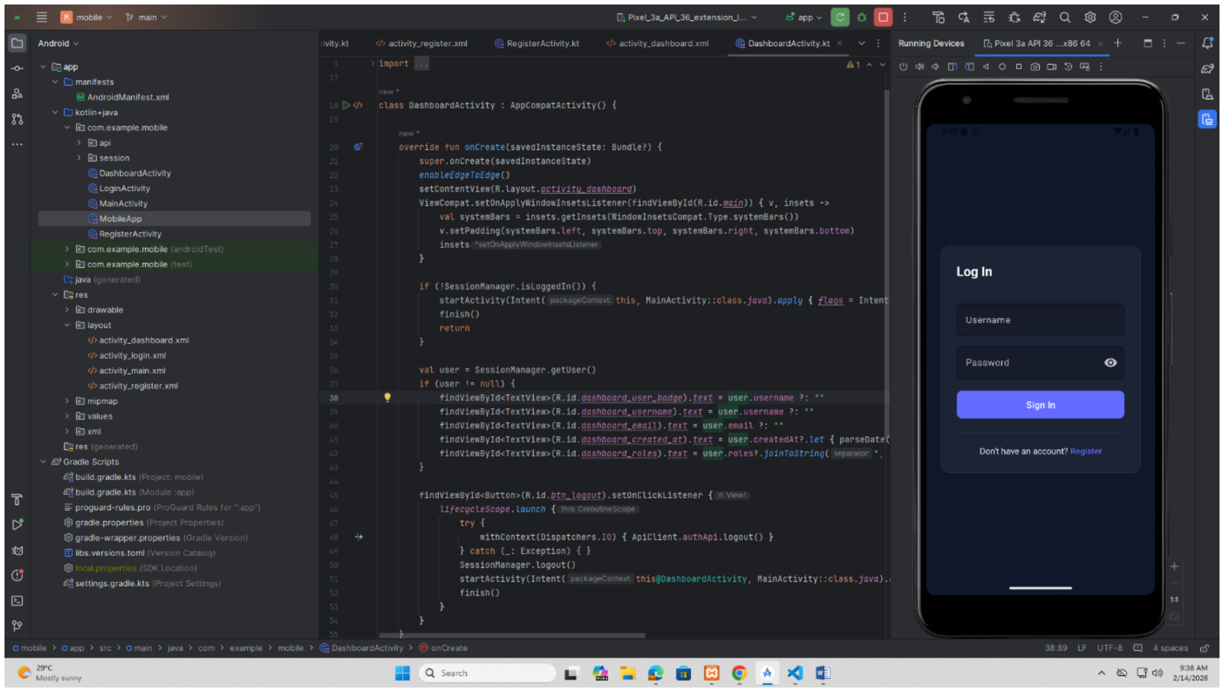
```kotlin
import ...

class DashboardActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_dashboard)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }

        if (!SessionManager.isLoggedIn()) {
            startActivity(Intent(this, MainActivity::class.java).apply { flags = Intent
            finish()
            return
        }

        val user = SessionManager.getUser()
        if (user != null) {
            findViewById<TextView>(R.id.dashboard_user_badge).text = user.username ?: ""
            findViewById<TextView>(R.id.dashboard_username).text = user.username ?: ""
            findViewById<TextView>(R.id.dashboard_email).text = user.email ?: ""
            findViewById<TextView>(R.id.dashboard_created_at).text = user.createdAt?.let { parseDate(
            findViewById<TextView>(R.id.dashboard_roles).text = user.roles?.joinToString( ",
        }

        findViewById<Button>(R.id.btn_logout).setOnClickListener {
            lifecycleScope.launch {
                try {
                    withContext(Dispatchers.IO) { ApiClient.authApi.logout() }
                } catch (_: Exception) { }
                SessionManager.logout()
                startActivity(Intent(this@DashboardActivity, MainActivity::class.java).
                finish()
            }
        }
}
```

- Registration Page



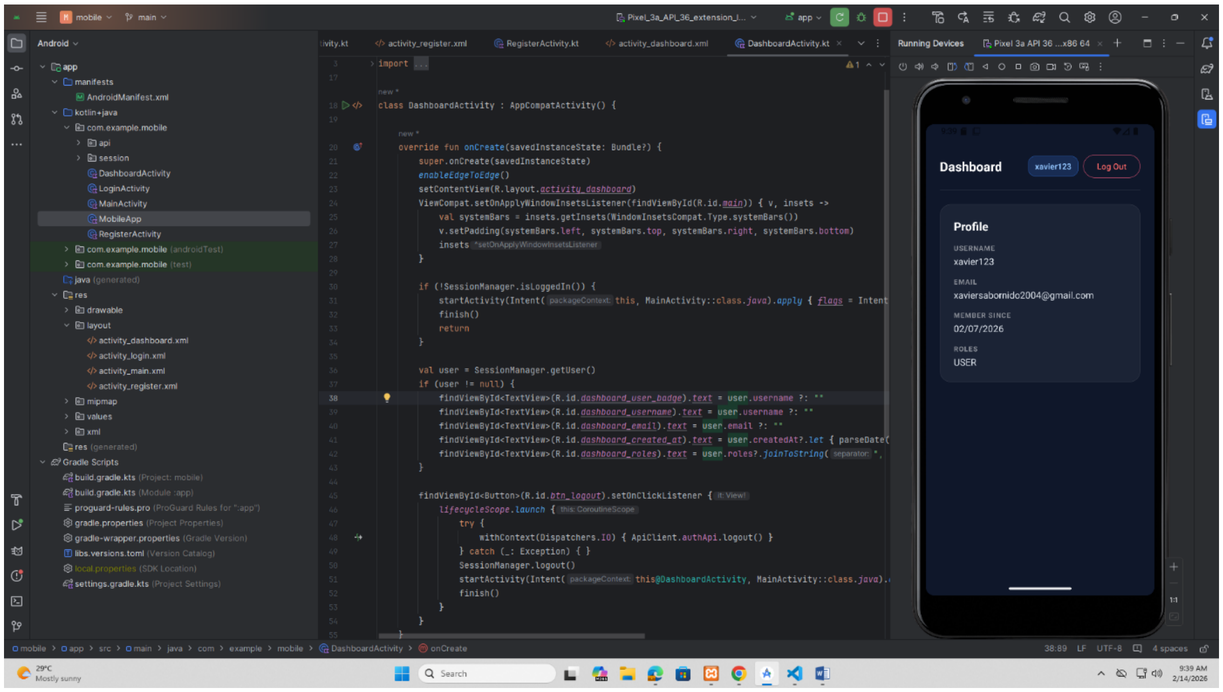
```kotlin
import ...

class DashboardActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_dashboard)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }

        if (!SessionManager.isLoggedIn()) {
            startActivity(Intent(this, MainActivity::class.java).apply { flags = Intent
            finish()
            return
        }

        val user = SessionManager.getUser()
        if (user != null) {
            findViewById<TextView>(R.id.dashboard_user_badge).text = user.username ?: ""
            findViewById<TextView>(R.id.dashboard_username).text = user.username ?: ""
            findViewById<TextView>(R.id.dashboard_email).text = user.email ?: ""
            findViewById<TextView>(R.id.dashboard_created_at).text = user.createdAt?.let { parseDate(
            findViewById<TextView>(R.id.dashboard_roles).text = user.roles?.joinToString( ",
        }

        findViewById<Button>(R.id.btn_logout).setOnClickListener {
            lifecycleScope.launch {
                try {
                    withContext(Dispatchers.IO) { ApiClient.authApi.logout() }
                } catch (_: Exception) { }
                SessionManager.logout()
                startActivity(Intent(this@DashboardActivity, MainActivity::class.java).
                finish()
            }
        }
}
```

- Login Page



```kotlin
import ...

class DashboardActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_dashboard)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }

        if (!SessionManager.isLoggedIn()) {
            startActivity(Intent(this, MainActivity::class.java).apply { flags = Intent
            finish()
            return
        }

        val user = SessionManager.getUser()
        if (user != null) {
            findViewById<TextView>(R.id.dashboard_user_badge).text = user.username ?: ""
            findViewById<TextView>(R.id.dashboard_username).text = user.username ?: ""
            findViewById<TextView>(R.id.dashboard_email).text = user.email ?: ""
            findViewById<TextView>(R.id.dashboard_created_at).text = user.createdAt?.let { parseDate(
            findViewById<TextView>(R.id.dashboard_roles).text = user.roles?.joinToString( ",
        }

        findViewById<Button>(R.id.btn_logout).setOnClickListener {
            lifecycleScope.launch {
                try {
                    withContext(Dispatchers.IO) { ApiClient.authApi.logout() }
                } catch (_: Exception) { }
                SessionManager.logout()
                startActivity(Intent(this@DashboardActivity, MainActivity::class.java).
                finish()
            }
        }
    }
```

- Dashboard



```kotlin
import ...

class DashboardActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_dashboard)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }

        if (!SessionManager.isLoggedIn()) {
            startActivity(Intent(this, MainActivity::class.java).apply { flags = Intent
            finish()
            return
        }

        val user = SessionManager.getUser()
        if (user != null) {
            findViewById<TextView>(R.id.dashboard_user_badge).text = user.username ?: ""
            findViewById<TextView>(R.id.dashboard_username).text = user.username ?: ""
            findViewById<TextView>(R.id.dashboard_email).text = user.email ?: ""
            findViewById<TextView>(R.id.dashboard_created_at).text = user.createdAt?.let { parseDate(
            findViewById<TextView>(R.id.dashboard_roles).text = user.roles?.joinToString( ",
        }

        findViewById<Button>(R.id.btn_logout).setOnClickListener {
            lifecycleScope.launch {
                try {
                    withContext(Dispatchers.IO) { ApiClient.authApi.logout() }
                } catch (_: Exception) { }
                SessionManager.logout()
                startActivity(Intent(this@DashboardActivity, MainActivity::class.java).
                finish()
            }
        }
    }
```