

# MOBILE PROGRAMMERS MANUAL



University of Mindanao Engineering Consultation App

CPE223/L (7599)

Author: Aimee Rose B. Vallente

[a.vallente.544816@umindanao.edu.ph](mailto:a.vallente.544816@umindanao.edu.ph)

Engr. Jay Al Gallenero

October 10, 2025

## B. Table of Contents

C. INTRODUCTION .....	3
I. Purpose of the manual.....	3
II. System overview.....	3
III. Technologies Used.....	3
i. Languages .....	3
ii. Frameworks.....	3
iii. RDBMS.....	4
iv. Tools .....	4
D. SYSTEM ARCHITECTURE .....	4
I. Overview of Software Architecture .....	4
E. INSTALLATION INSTRUCTIONS.....	7
I. Environment Setup.....	7
II. Software Requirements.....	7
III. Step-by-Step Installation Process .....	7
F. SOURCE CODE STRUCTURE .....	9
I. Project Folder Layout.....	9
II. Key Packages/Modules/Classes.....	12
III. Naming Conventions .....	15
G. FUNCTION DOCUMENTATION .....	17
I. Method Signatures and Purpose.....	17
II. INPUTS .....	24
III. OUTPUTS .....	26
• Side Effects:.....	27
H. DATABASE DESIGN .....	32
I. ENTITY RELATIONSHIP DIAGRAM .....	32
II. TABLE DEFINITIONS .....	34
III. SAMPLE DATA.....	41
I. TROUBLESHOOTING & DEBUGGING TIPS .....	44
COMMON ISSUES .....	44
DEBUGGING STRATEGIES.....	50
UNIT TEST LOCATION AND USAGE .....	54
J. VERSION CONTROL & BUILD INSTRUCTIONS.....	58
GIT USAGE GUIDELINES.....	58
BUILD TOOLS AND COMMANDS .....	61

## C. INTRODUCTION

### I. Purpose of the manual

This programming manual serves as a comprehensive technical guide for developers working with the University of Mindanao Engineering Consultation App (UMECA). The manual provides detailed documentation of the system's architecture, codebase structure, installation procedures, and development guidelines. It is designed to facilitate understanding of the application's technical implementation, support efficient onboarding of new developers, and establish standardized practices for maintenance and future development of the consultation management system.

### II. System overview

The University of Mindanao Engineering Consultation App (UMECA) is a crossplatform mobile application developed to streamline and enhance the academic consultation process between engineering students and faculty members. The app serves as a centralized system where students can schedule consultations and receive schedule approvals from their professors. UMECA aims to improve communication, reduce scheduling conflicts, and ensure transparency in tracking academic support sessions. It is accessible via mobile devices on both iOS and Android, providing convenience for both students and faculty.

UMECA features user authentication, role-based access (student, faculty, admin), consultation request management, scheduling tools, a bulletin board, and notification systems. Faculty members can manage consultation approvals, review consultation histories, and provide feedback through the platform, while students can monitor upcoming appointments and receive real-time updates. The system supports documentation and reporting functions to aid academic monitoring and institutional assessment. Overall, UMECA is designed to foster a more efficient, organized, and responsive academic environment within the University of Mindanao specifically College of Engineering Education. The application implements an MVVM (ModelView-ViewModel) architectural pattern and utilizes Azure MySQL Database for data persistence, ensuring scalability and reliability for the institution's consultation management needs.

### III. Technologies Used

#### i. Languages

- **C#** - Primary programming language for application logic and business rules
- **XAML (Extensible Application Markup Language)** - Used for defining the user interface layout and visual structure of the application pages and components.

#### ii. Frameworks

- **.NET MAUI (Multi-platform App UI)** - Cross-platform application framework for building native mobile and desktop applications
- **.NET 8.0** - The foundational framework providing the runtime environment and base class libraries for the application.

- **Entity Framework Core** - The ORM (Object-Relational Mapping) framework used for database operations and data access layer implementation.
- **ASP.NET Core Identity** - Handles user authentication, authorization, and account management functionality.
- **Shiny Framework (Version 2.7.3)** - Provides background services and push notification capabilities.
- **Microsoft Azure Notification Hubs (Version 4.2.0)** - Enables push notification delivery across multiple platforms.

### iii. RDBMS

**MySQL 8.0.36** - The relational database management system hosted on Azure, accessible via:

- Server: consultationdb.mysql.database.azure.com
- Port: 3306
- Database: consultationdatabase

**Azure Database for MySQL - Cloud-hosted database service providing high availability, automated backups, and enterprise-grade security with SSL encryption.**

### iv. Tools

- **Visual Studio 2022** - Primary integrated development environment (IDE)
- **Git** - Version control system for source code management
- **Azure Cloud Services** - Cloud infrastructure for database hosting and notification services.
- **NuGet Package Manager** - Dependency management tool for .NET packages.
- **MySQL Workbench** - Database design and administration tool for schema management and query execution.
- **MAUI (.NET Multi-platform App UI)** - Framework dependencies and build tools
- **Android SDK** - Android platform development and emulation tools

## D. SYSTEM ARCHITECTURE

### I. Overview of Software Architecture

The UM Consultation App MAUI implements a layered architecture pattern with clear separation of concerns, organized into the following layers:

#### Presentation Layer:

- Views XAML - based user interface pages organized by user role (StudentView, FacultyView, Common)
- ViewModels: Business logic controllers implementing the MVVM pattern using CommunityToolkit.Mvvm
- Models: UI-specific data transfer objects for view binding
- MvvmHelper: Utility classes providing common UI operations (loading screens, message displays, helpers)

### **Service Layer (Consultation.Services Project)**

- IService interfaces defining service contracts
- Service implementations containing business logic
- AuthService: Handles authentication and user management
- StudentServices: Manages student-specific operations
- FacultyServices: Manages faculty-specific operations
- ConsultationRequestServices: Processes consultation requests and status updates

### **Repository Layer (Consultation.Repository Project)**

- IRepository interfaces defining data access contracts
- Repository implementations providing data access logic
- UserRepository: User account data operations
- StudentRepository: Student entity data operations
- FacultyRepository: Faculty entity data operations
- EditConsultationRequestRepository: Consultation modification operations

### **Domain Layer (Consultation.Domain Project)**

- Entity models representing database tables:
  - o Users (base authentication entity)
  - o Student (student profile and relationships)
  - o Faculty (faculty profile and relationships)
  - o ConsultationRequest (consultation records)
  - o EnrolledCourse (student course enrollments)
  - o Department, Program, SchoolYear (supporting entities)
- Enumerations (Status, Semester, YearLevel, NotificationType, UserType)
- Domain-specific business rules and validation

### **Infrastructure Layer (Consultation.Infrastructure Project)**

- ApplicationDbContext: Entity Framework Core database context
- Database configuration and connection management
- Migrations for schema versioning
- CreateDbContext and DatabaseSeeder for initialization

### **Cross-Cutting Concerns**

- Dependency Injection: Configured in MauiProgram.cs using Microsoft.Extensions.DependencyInjection
- Navigation: Shell-based navigation system defined in AppShell.xaml
- Authentication: ASP.NET Core Identity integration with custom user entities
- Error Handling: Centralized through Helper.DisplayMessage utility

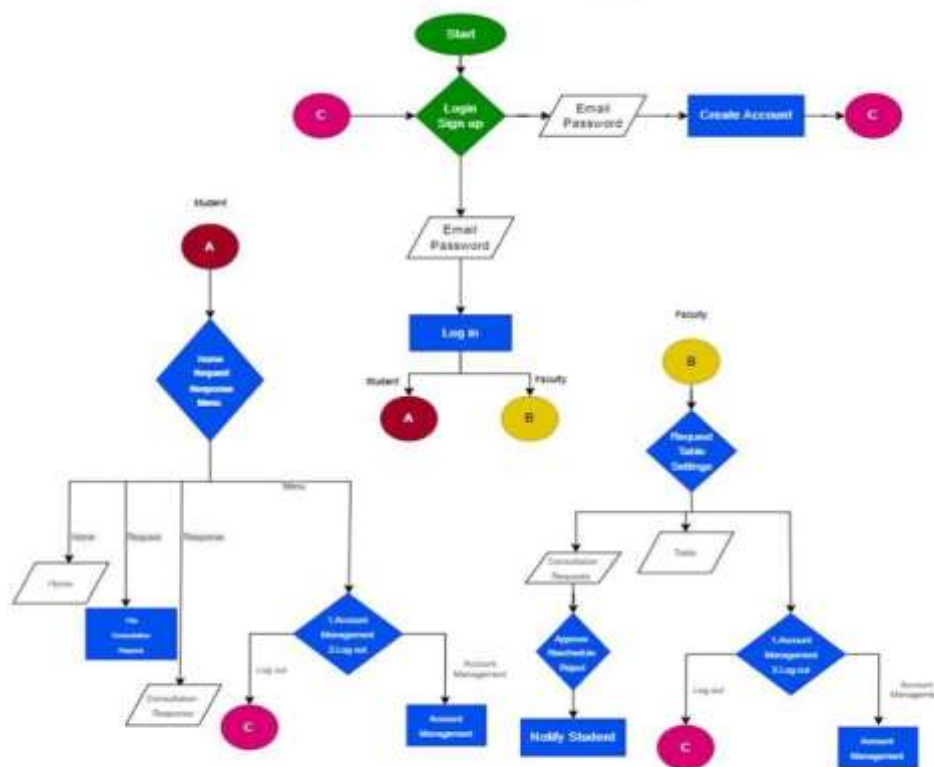
### **Data Flow**

1. View triggers user action → ViewModel receives command

2. ViewModel invokes Service layer method
3. Service layer processes business logic and calls Repository
4. Repository executes database operations through Entity Framework Core
5. Data flows back through layers with appropriate transformations
6. ViewModel updates observable properties
7. View reflects changes through data binding

This architecture ensures maintainability, testability, and scalability while supporting cross-platform deployment through .NET MAUI.

## II. Flow Diagram



### Key Application Flow:

**Authentication Flow:** LoginPage → Authentication Service → Database Verification → Role-based Navigation (Student/Faculty TabBar)

**Student Consultation Request Flow:** HomePage → RequestPage (Select Semester) → RequestConsultationPage (Fill Details) → Service Layer → Repository → Database → ResponsePage (View Status)

**Faculty Consultation Management Flow:** RequestListPage (View Pending) → Approve/Disapprove Action → Service Layer → Repository → Database Update → ConsultationListPage (View History)

**Navigation Flow:** The application uses Shell-based navigation with predefined routes defined in AppShell.xaml, supporting both TabBar navigation for rolespecific interfaces and programmatic navigation for detailed pages.

## E. INSTALLATION INSTRUCTIONS

### I. Environment Setup

#### Prerequisites:

- Windows 10/11 (version 1809 or higher)
- Visual Studio 2022 (version 17.3 or later)
- .NET 7.2 SDK or higher
- Android SDK (API level 21 or higher)
- 8GB RAM minimum (16GB recommended)
- 10GB available disk space
- Azure MySQL Database
- MySQL Workbench (optional, for database administration)
- Git version control system installed and configured

### II. Software Requirements

#### Required Workloads in Visual Studio:

- .NET Multi-platform App UI development
- Mobile development with .NET
- Universal Windows Platform development (for Windows deployment)

#### Optional Tools:

- Android Emulator or physical Android device
- iOS Simulator (macOS only) or physical iOS device
- Git command-line tools

### III. Step-by-Step Installation Process

#### 1. Clone the Repository: bash

```
git clone <repository-url> cd  
proto-sd-project
```

#### 2. Open the Solution:

- Launch Visual Studio 2022
- Open UM-Consultation-App-MAUI.sln from the cloned repository • Wait for Visual Studio to restore NuGet packages automatically

#### 3. Restore Dependencies: If automatic restoration fails:

- Right-click on the solution in Solution Explorer
  - Select "Restore NuGet Packages" ○ Wait for all packages to download and install ○
4. **Configure Database Connection (if needed):** The database connection string is configured in MauiProgram.cs. The default configuration connects to the Azure MySQL instance:

```
csharp
var cs = "Server=consultationdb.mysql.database.azure.com;" +
        "Port=3306;" +
        "Database=consultationdatabase;" +
        "User Id=ConsultationDb;" +
        "Password=MyServerAdmin123!;" +
        "SslMode=Required;"
```

#### 5. **Build the Project:**

Build → Build Solution (Ctrl+Shift+B)

#### 6. **Run the Application:**

Debug → Start Debugging (F5)

The application will deploy to the selected target and launch First launch may take several minutes as the application is deployed

#### 7. **Verify Installation:**

- The application should display the login page (LoginPage.xaml)
- Test basic navigation and authentication functionality
- Verify database connectivity by attempting to log in with valid credentials

#### 8. **Troubleshooting Initial Setup:**

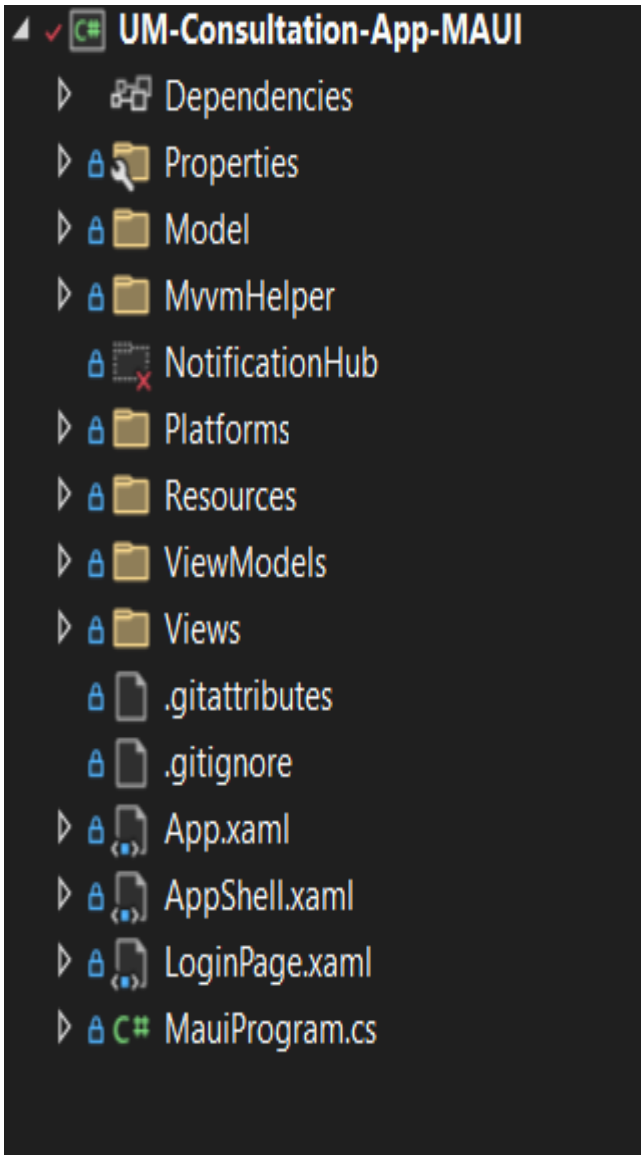
- If build fails, verify all workloads are installed
- Ensure Android SDK is properly configured in Visual Studio
- Check that .NET SDK version matches project requirements



## F. SOURCE CODE STRUCTURE

### I. Project Folder Layout

The solution follows a multi-project architecture organized as follows:



The screenshot shows the solution explorer for the project 'UM-Consultation-App-MAUI'. The project is a C# project. The folder structure includes: Dependencies, Properties, Model, MvvmHelper, NotificationHub, Platforms, Resources, ViewModels, Views, .gitattributes, .gitignore, App.xaml, AppShell.xaml, LoginPage.xaml, and MauiProgram.cs.

Project Name: UM-Consultation-App-MAUI

Architectural Role: Presentation Layer (User Interface)

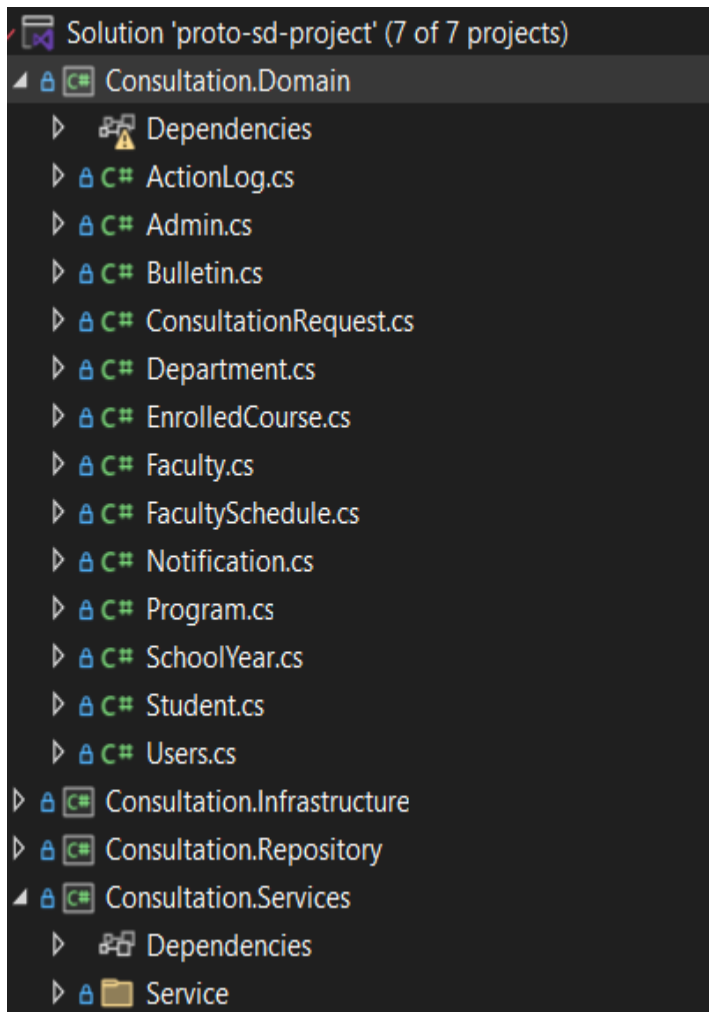
Folder Structure:

- Dependencies
- Properties
- Model folder
- MvvmHelper folder
- NotificationHub folder
- Platforms folder
- Resources folder
- ViewModels folder
- Views folder

Root Files:

- .gitattributes
- .gitignore

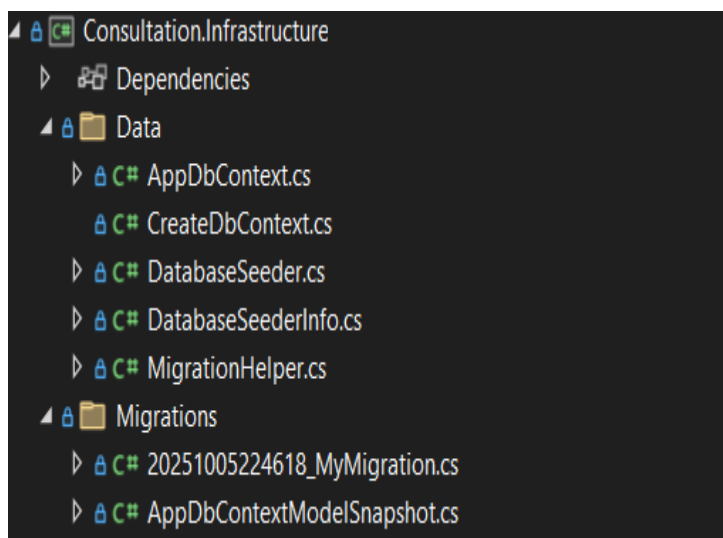
- App.xaml (XAML file)
- AppShell.xaml (XAML file)
- LoginPage.xaml (XAML file)
- MauiProgram.cs (C# file)



Architectural Role: Domain Layer  
(Core Business Entities)

- Dependencies
- C# Entity Classes
- ActionLog.cs
- Admin.cs
- Bulletin.cs
- ConsultationRequest.cs
- Department.cs
- EnrolledCourse.cs
- Faculty.cs
- FacultySchedule.cs
- Notification.cs
- Program.cs
- SchoolYear.cs

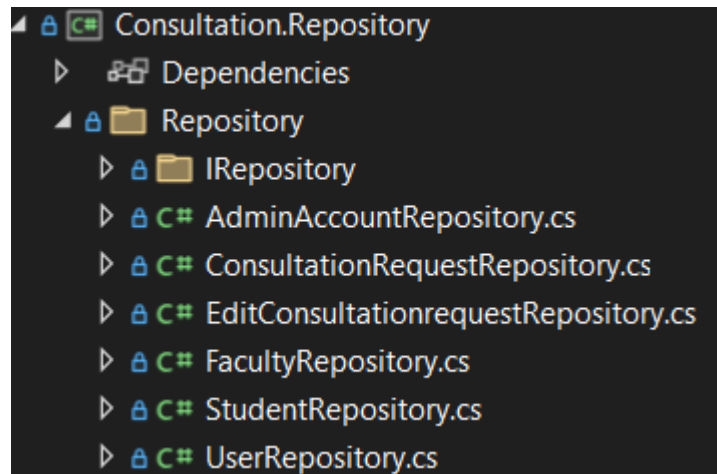
- Student.cs
- Users.cs



Architectural Role: Infrastructure  
Layer (Data Persistence)

- **Data** folder (expanded)
- AppDbContext.cs
- CreateDbContext.cs
- DatabaseSeeder.cs
- DatabaseSeederInfo.cs
- **Migrations** folder (expanded)

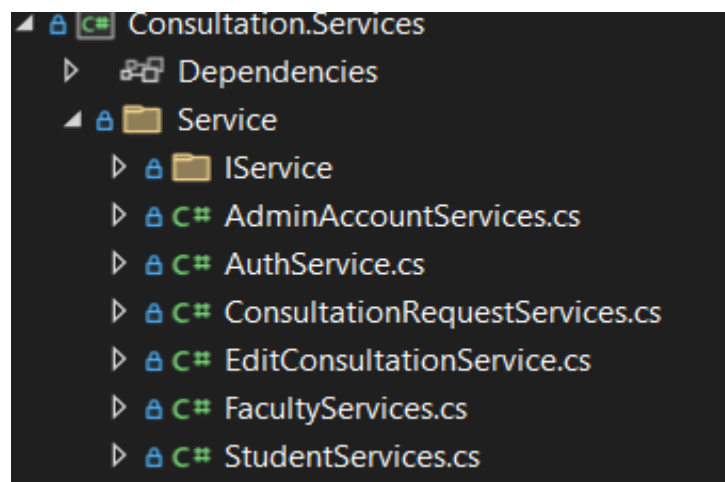
- 20251005224618\_MyMigration.cs
- AppDbContextModelSnapshot.cs



Architectural Role: Repository Layer (Data Access Abstraction)

- Dependencies
- Repository folder (expanded)
- IRepository subfolder
- C# Repository Implementation Files:
- AdminAccountRepository.cs

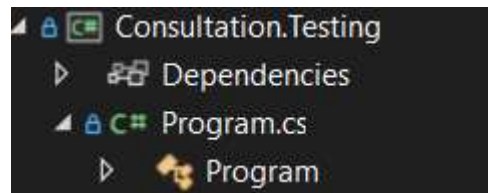
- ConsultationRequestRepository.cs
- EditConsultationrequestRepository.cs (note: lowercase 'r' in filename)
- FacultyRepository.cs
- StudentRepository.cs
- UserRepository.cs



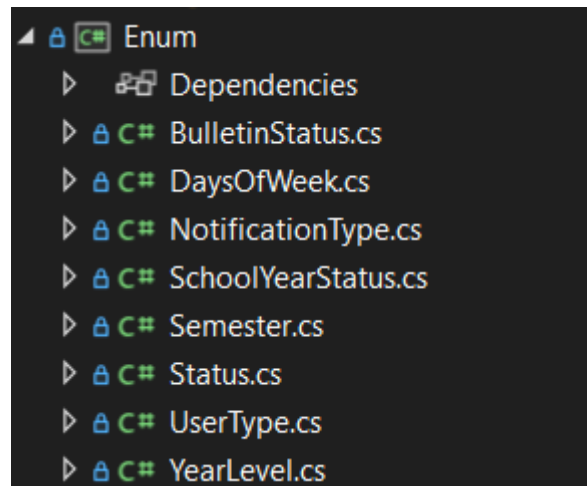
Architectural Role: Service Layer (Business Logic)

- Dependencies
- Service folder (expanded)
- IService subfolder
- C# Service Implementation Files:
- AdminAccountServices.cs

- AuthService.cs
- ConsultationRequestServices.cs
- EditConsultationServices.cs
- FacultyServices.cs
- StudentServices.cs



- Architectural Role: Testing/Utility Layer
- Dependencies
- Program.cs (C# file)
- Program node (appears to be a project reference or configuration item)



Architectural Role: Cross-cutting concerns layer providing shared type definitions

- Dependencies folder
- C# Files:

- BulletinStatus.cs
- DaysOfWeek.cs
- NotificationType.cs
- SchoolYearStatus.cs

- Semester.cs
- Status.cs
- UserType.cs
- YearLevel.cs

### Configuration Files:

.gitattributes - Git line ending configuration

.gitignore - Git ignore rules for build artifacts

UM-Consultation-App-MAUI.sln - Visual Studio solution file

launchSettings.json - Debug launch profiles

## II. Key Packages/Modules/Classes

### Core Application Classes:

#### App.xaml.cs

- Entry point for the MAUI application
- Initializes the main page and application shell
- Configures application-level resources

## **MauiProgram.cs**

- Configures dependency injection container
- Registers services, repositories, ViewModels, and pages
- Configures Entity Framework Core with Azure MySQL
- Sets up ASP.NET Core Identity
- Registers Shiny services for notifications

## **AppShell.xaml/AppShell.xaml.cs**

- Defines application navigation structure
- Configures TabBar navigation for Student and Faculty roles
- Registers navigation routes using Shell routing system

## **Key ViewModels:**

### **LoginViewModel.cs**

- Handles user authentication logic
- Manages password visibility toggling
- Routes users based on their role (Student/Faculty)
- Stores current user session information in static properties (Student, Faculty)

### **HomeViewModel.cs**

- Displays student profile information
- Shows pending consultation count
- Formats and presents school year and program data

### **RequestViewModel.cs**

- Manages student course enrollment display
- Handles school year/semester selection
- Navigates to consultation request form

### **RequestConsultationViewModel.cs**

- Handles consultation request submission
- Manages course selection and faculty assignment
- Validates consultation time slots

### **ResponseViewModel.cs**

- Displays student consultation request history
- Shows consultation status with color-coded indicators
- Filters consultations by semester

### **FacultyRequestViewModel.cs**

- Displays pending consultation requests for faculty
- Handles consultation approval/disapproval logic

- Updates consultation status in the database

### **FacultyCLPViewModel.cs**

- Displays faculty consultation history (Consultation List Page)
- Filters consultations excluding pending status

### **MenuViewModel.cs**

- Displays user profile information
- Handles password change functionality
- Manages user logout

#### **Key Services:**

### **IAuthService / AuthService**

- User authentication and registration
- Password hashing using ASP.NET Core Identity
- Role-based user creation

### **IStudentServices / StudentServices**

- Retrieves student information
- Fetches enrolled courses by semester
- Manages student consultation requests

### **IFacultyServices / FacultyServices**

- Retrieves faculty information
- Manages faculty-specific operations

### **IConsultationRequestServices / ConsultationRequestServices**

- Creates and manages consultation requests
- Updates consultation status • Handles consultation scheduling

#### **Key Repositories:**

### **IAuthRepository / UserRepository**

- User account data access
- Implements authentication queries

### **IStudentRepository / StudentRepository**

- Student entity data operations
- Enrolled course retrieval

### **IFacultyRepository / FacultyRepository**

- Faculty entity data operations
- Faculty consultation list retrieval

- Consultation status updates

## **Helper Utilities:**

### **Helper.cs (MvvmHelper)**

- GetSemesterName(): Converts Semester enum to display string
- GetYearLevelName(): Converts YearLevel enum to display string
- DisplayMessage(): Shows alert dialogs
- DisplayOption(): Shows confirmation dialogs
- StringSplitter(): Utility for string parsing

### **LoadingServices.cs**

- Implements ILoadingServices interface
- Shows/hides loading popup during async operations

## **Domain Entities:**

**Users** - Base authentication entity with Identity integration

**Student** - Student profile with relationships:

- Program, SchoolYear, EnrolledCourses, ConsultationRequests

**Faculty** - Faculty profile with relationships:

- Program, SchoolYear, EnrolledCourses, FacultySchedules, ConsultationRequests

**ConsultationRequest** - Consultation record with:

- Student, Faculty relationships
- Schedule details (DateSchedule, StartedTime, EndedTime)
- Status tracking (Pending, Approved, Declined)

**EnrolledCourse** - Links students/faculty to courses with:

- CourseCode, CourseName, SchoolYear, Program relationships

## **III. Naming Conventions**

### **C# Code Conventions:**

The project follows C# and .NET MAUI standard naming conventions:

## General Conventions:

- PascalCase for class names, method names, public properties, and namespaces o Example: LoginViewModel, DisplayStudentUserInformation(), StudentName
- camelCase for private fields, local variables, and method parameters o Example: \_authService, studentInfo, userUMIDNumber
- Private field prefix with underscore (\_) for readonly service dependencies o Example: private readonly IAuthService \_authService;

## File Naming:

- XAML view files: PascalCase.xaml (e.g., LoginPage.xaml, HomePage.xaml)
- Code-behind files: PascalCase.xaml.cs (e.g., LoginPage.xaml.cs)
- ViewModel files: PascalCaseViewModel.cs (e.g., LoginViewModel.cs)
- Service interfaces: IPascalCaseService.cs (e.g., IAuthService.cs)
- Repository interfaces: IPascalCaseRepository.cs (e.g., IStudentRepository.cs)
- Model classes: PascalCase.cs (e.g., StudentModel.cs)

## XAML Naming:

- Control names use x:Name attribute with camelCase prefixed by control type o Example: txtboxEmail, btnLogin, lblPasswordMatch
- Command bindings: {Binding CommandNameCommand} o Example: {Binding ClickLogInCommand}, {Binding CreateAccountClickCommand}

## Observable Properties (CommunityToolkit.Mvvm):

- Decorated with [ObservableProperty] attribute
- Field name in camelCase, generates PascalCase property o Example: [ObservableProperty] private string studentname; generates Studentname property

## Relay Commands (CommunityToolkit.Mvvm):

- Methods decorated with [RelayCommand] attribute
- Method name describes action, generates MethodNameCommand property o Example: [RelayCommand] private async Task ClickLogIn() generates ClickLogInCommand

## Navigation Routes:

- Defined in AppShell with PascalCase page names o Example: Route="LoginPage", Route="HomePage"

## Database Naming:

- Table names: PascalCase matching entity class names (e.g., Users, Student, ConsultationRequest)
- Column names: PascalCase matching property names (e.g., StudentID, FacultyName)
- Foreign key relationships follow convention: EntityNameID (e.g., StudentID, FacultyID)



### Constants and Enums:

- Enum types: PascalCase (e.g., Status, Semester, YearLevel) • Enum values: PascalCase (e.g., Pending, Approved, Declined)

### Namespace Organization:

- Root namespace: UM\_Consultation\_App\_MAUI • Sub-namespaces follow folder structure:
  - UM\_Consultation\_App\_MAUI.ViewModels
  - UM\_Consultation\_App\_MAUI.Views.StudentView
  - UM\_Consultation\_App\_MAUI.MvvmHelper
  - Consultation.Domain
  - Consultation.Services.Service.IService

### Resource Naming:

- Images: lowercase with underscores (e.g., email\_icon.png, homeicon.png)
- Colors and styles: PascalCase in XAML (e.g., <Color x:Key="Primary">#b61c1c</Color>)

## G. FUNCTION DOCUMENTATION

### I. Method Signatures and Purpose

#### LoginViewModel Methods:

*csharp [RelayCommand] private*

*async Task ClickLogIn()*

**Purpose:** Authenticates user credentials, determines user role (Student/Faculty), retrieves user information, and navigates to the appropriate dashboard.

*csharp [RelayCommand] private async Task*

*CreateAccountClick()*

**Purpose:** Navigates to the account creation page.

*csharp [RelayCommand]*

*private void TogglePasswordVisibility()*

**Purpose:** Toggles the password field visibility between masked and visible text. *csharp*

*private async Task StudentInformation(string userUMIDNumber)*

**Purpose:** Retrieves and stores student profile information including enrolled courses and consultation requests.

*csharp*

*private async Task FacultyInformation(string userUMIDNumber)*

**Purpose:** Retrieves and stores faculty profile information including assigned courses and consultations.

---

### **HomeViewModel Methods:**

*csharp private async void*

*DisplayStudentUserInformation()*

**Purpose:** Loads and formats student profile data for display on the home page, including name formatting, school year, program, and pending consultation count.

*csharp private void UserInformation(Student  
studentInfo)*

**Purpose:** Formats student information with proper name ordering (Last, First Middle) and prepares display strings for UI binding.

---

### **RequestViewModel Methods: *csharp private***

*async void DisplayEnrolledCourses()*

**Purpose:** Retrieves student's enrolled courses and populates the UI collection with course details and available school years.

*csharp [RelayCommand] private async Task*

*RequestConsultationClick()*

**Purpose:** Validates semester selection, converts selected semester string to enum value, and navigates to the consultation request form with the selected semester parameter.

### **RequestConsultationViewModel Methods: *csharp***

*[RelayCommand] private async void*

*DisplayUserInformationOption()*

**Purpose:** Loads student's enrolled courses for the selected semester and populates the course selection dropdown. *csharp private void DisplayUserInformation()*

**Purpose:** Formats and displays student name and UMID in the consultation request form header.

```
csharp public ICommand CourseCodeCommand => new Command<object>(ec => { ...  
    })
```

**Purpose:** Handles course selection event, updates the instructor name and course code fields based on the selected enrolled course.

*csharp [RelayCommand]*

*public async Task FileConsultationClick()*

**Purpose:** Validates consultation request data, submits the consultation request to the service layer, and provides user feedback. (Currently shows "under development" message) *csharp [RelayCommand] public async Task BackNavigation()*

**Purpose:** Clears the course selection list and navigates back to the student home page.

---

### **ResponseViewModel Methods: *csharp***

*private async void LoadResponses()*

**Purpose:** Retrieves student's consultation request history, formats the data with color-coded status indicators, and populates both the consultation list and semester filter options.

---

### **FacultyRequestViewModel Methods:**

*csharp private async void*

*DisplayConsultation()*

**Purpose:** Loads pending consultation requests for the logged-in faculty member and populates the pending requests list.

*csharp [RelayCommand] private async Task ApproveRequest(Consultations  
selectedConsultation)*

**Purpose:** Displays confirmation dialog, updates the consultation status to "Approved" in the database, and removes the request from the pending list.

*csharp [RelayCommand] private async Task DisApproveRequest(Consultations  
selectedConsultation)*

**Purpose:** Displays confirmation dialog, prompts for disapproval reason, updates the consultation status to "Declined" with the reason in the database, and removes the request from the pending list.

---

### **FacultyCLPViewModel Methods: *csharp***

*private async void DisplayConsultation()*

**Purpose:** Retrieves faculty's consultation history excluding pending requests and populates the consultation list with approved and declined consultations.

---

### **MenuViewModel Methods:**

*csharp private async void*

*DisplayStudentInformation()*

**Purpose:** Determines user role (Student/Faculty) and displays appropriate profile information (name, UMID, email, program) in the menu page.

*csharp [RelayCommand] private async*

*Task ChangePassword()*

**Purpose:** Displays a popup dialog for users to change their password.

*csharp [RelayCommand] public*

*async void LogoutButton()*

**Purpose:** Resets the application shell, clears user session, and navigates back to the login page.

---

#### **CreateAccountViewModel Methods:**

*csharp [RelayCommand] private async*

*Task CreateAccount()*

**Purpose:** Validates account creation form (password match, required fields, email format), extracts UMID from email, and creates a new user account with the specified role.

*csharp private Consultation.Domain.Enum.UserType*

*UserTypeChecker()* **Purpose:** Converts the selected user type string to

the corresponding UserType enum value. *csharp private void*

*AddUserType()*

*Purpose:* Populates the user type dropdown with "Student" and "Faculty" options.

---

#### **Helper Utility Methods (MvvmHelper.Helper): csharp**

*public static string GetSemesterName(Semester value)*

**Purpose:** Retrieves the display name attribute of a Semester enum value for UI presentation.

*csharp public static string GetYearLevelName(YearLevel value)*

**Purpose:** Retrieves the display name attribute of a YearLevel enum value for UI presentation.

*csharp public static void DisplayMessage(string message)*

**Purpose:** Shows a modal alert dialog with the specified message and an "OK" button.

*csharp*

*public static async Task<bool> DisplayOption(string message, string FirstOption, string SecondOption)*

**Purpose:** Shows a confirmation dialog with two custom button options and returns the user's choice as a boolean.

***csharp***

*public static List<string> StringSplitter(char splitter, string word)*

**Purpose:** Splits a string by the specified character delimiter and returns a list of substrings.

---

### **LoadingServices Methods:**

***csharp*** *public void Show()*

**Purpose:** Displays a loading screen popup to indicate ongoing background operations and prevent user interaction.

***csharp*** *public void*

*Hide()*

**Purpose:** Closes and disposes of the loading screen popup, restoring user interaction capability.

---

### **Service Layer Methods (Consultation.Services):**

#### **IAuthService / AuthService: *csharp***

*Task<Users> Login(string email, string password, string role)*

**Purpose:** Authenticates user credentials against the database, verifies the specified role, and returns the user entity if successful.

***csharp***

*Task CreateAccount(string email, string umid, string password, UserType userType, string phoneNumber, string userId)*

**Purpose:** Creates a new user account with hashed password, assigns the specified role, and initializes the user entity in the database.

#### **IStudentServices / StudentServices: *csharp***

*Task<Student> GetStudentInformation(string studentUMID)*

**Purpose:** Retrieves comprehensive student information including program, school year, enrolled courses, and consultation requests using the student's UMID. ***csharp***

*Task<List<EnrolledCourse>> GetStudentEnrolledCourses(int studentId)* **Purpose:**

Retrieves all enrolled courses for a student across all semesters.

*csharp*

*Task<List<EnrolledCourse>> GetStudentEnrolledCourses(int studentId, Semester semester)*

**Purpose:** Retrieves enrolled courses for a student filtered by the specified semester.

*csharp*

*Task<List<ConsultationRequest>> GetStudentConsultationRequests(int studentId)*

**Purpose:** Retrieves all consultation requests submitted by the specified student.

**IFacultyServices / FacultyServices:** *csharp*

*Task<Faculty> GetFacultyInformation(string facultyUMID)*

**Purpose:** Retrieves comprehensive faculty information including program, enrolled courses, faculty schedules, and consultation requests using the faculty's UMID.

**IConsultationRequestServices / ConsultationRequestServices:** *csharp*

*Task AddConsultation(int studentId, string courseCode, DateTime dateSchedule, TimeSpan startTime, TimeSpan endTime, Status status, string disapprovedReason)*

**Purpose:** Creates a new consultation request record with the specified details and persists it to the database.

---

## **Repository Layer Methods:**

**IFacultyRepository / FacultyRepository:**

*csharp*

*Task<List<ConsultationRequest>> FacultyConsultation(int facultyId)*

**Purpose:** Retrieves all consultation requests associated with the specified faculty member, including student details.

*csharp*

*Task ChangeConsultationByID(int consultationId, Status status, string reason)*

**Purpose:** Updates the status and disapproval reason of a consultation request identified by its ID.

### **IStudentRepository / StudentRepository: *csharp***

*Task<Student> GetStudentByUMID(string umid)*

**Purpose:** Retrieves a student entity by UMID, including related program, school year, enrolled courses, and consultation requests.

### **IAuthRepository / UserRepository: *csharp***

*Task<Users> GetUserByEmailAsync(string email)*

**Purpose:** Retrieves a user entity by email address for authentication purposes.

*csharp*

*Task CreateUserAsync(Users user, string password)*

**Purpose:** Creates a new user account with a hashed password using Identity framework.

*csharp*

*Task AddToRoleAsync(Users user, string role)*

**Purpose:** Assigns a role to a user account using Identity framework.

## **II. INPUTS**

### **LoginViewModel.ClickLogIn:**

- Email (string): User's university email address (format: [firstinitial.lastname@umindanao.edu.ph](mailto:firstinitial.lastname@umindanao.edu.ph))
- Password (string): User's account password
- Returns user role determination and navigation result



**CreateAccountViewModel.CreateAccount:**

- Email (string): University email in required format
- Password (string): Desired account password
- ConfirmPassword (string): Password confirmation for validation
- SelectedUserType (string): "Student" or "Faculty"
- PhoneNumber (string, optional): Contact phone number • Extracts UMID from email (6-digit substring after second period)

**RequestViewModel.RequestConsultationClick:**

- SelectedSchoolYear (string): Semester selection in format "Semester YYYY - YYYY" (e.g., "First Semester 2024 - 2025")
- Converts to Semester enum value

**RequestConsultationViewModel.FileConsultationClick:**

- Studentname (string): Student's full name
- Studentumid (string): Student's UMID
- Coursecode (string): Selected course code
- Courseinstructor (string): Faculty member name
- Starttime (string): Consultation start time in TimeSpan format
- Endtime (string): Consultation end time in TimeSpan format
- Selectedcourses (string): Selected course name

**FacultyRequestViewModel.ApproveRequest:**

- selectedConsultation (Consultations): The consultation object to approve
- Id (int): Consultation request ID

**FacultyRequestViewModel.DisApproveRequest:**

- selectedConsultation (Consultations): The consultation object to disapprove
- Id (int): Consultation request ID
- Reason (string): User-entered reason for disapproval via prompt dialog

**AuthService.Login:**

- email (string): User's email address
- password (string): User's password (plain text, hashed during verification)
- role (string): "Student" or "Faculty" role identifier

**AuthService.CreateAccount:**

- email (string): User's email address
- umid (string): User's UMID
- password (string): Plain text password (hashed by service)
-

- userType (UserType enum): Student, Faculty, or Admin
  - phoneNumber (string): Contact phone number
  - userId (string): User identifier (typically UMID)
- StudentServices.GetStudentEnrolledCourses:**
- studentId (int): Student's database ID
  - semester (Semester enum, optional): Filter by specific semester

**ConsultationRequestServices.AddConsultation:**

- studentId (int): Student's database ID
- courseCode (string): Course identifier
- dateSchedule (DateTime): Scheduled consultation date
- startTime (TimeSpan): Consultation start time
- endTime (TimeSpan): Consultation end time
- status (Status enum): Initial status (typically Pending)
- disapprovedReason (string): Reason field (empty for new requests)

**Helper.StringSplitter:**

- splitter (char): Character delimiter for splitting
- word (string): String to be split

**Helper.DisplayOption:**

- message (string): Dialog message text
- FirstOption (string): Text for first button (typically "Yes")
- SecondOption (string): Text for second button (typically "No")

---

### III. OUTPUTS

**LoginViewModel.ClickLogIn:**

- **Returns:** Task (async void via navigation)
- **Side Effects:**
  - Sets LoginViewModel.Student or LoginViewModel.Faculty static property
  - Sets LoginViewModel.AccountVerification boolean flag
  - Navigates to ///Student or ///FacultyHomePage route
  - Displays loading screen during authentication
  - Shows error alert on invalid credentials

**CreateAccountViewModel.CreateAccount:**

- **Returns:** Task (async void)
- **Side Effects:**
  - Creates new user account in database via AuthService

- Displays success or error message via `Helper.DisplayMessage`
- Validates email format and shows error if invalid

#### **RequestViewModel.RequestConsultationClick:**

- **Returns:** Task (async void) •
- **Side Effects:**
  - Sets static `RequestViewModel.Semester` property
  - Navigates to `//RequestConsultationPage?Semester={Semester}` route
  - Shows loading screen during navigation
  - Displays error message if no semester selected

#### **RequestConsultationViewModel.FileConsultationClick:**

- **Returns:** Task (async void) •
- **Side Effects:**
  - Currently displays "under development" message
  - Intended to create consultation request in database
  - Shows loading screen during operation
  - Clears `AvailableCourses` collection

#### **ResponseViewModel.LoadResponses:**

- **Returns:** void
- **Side Effects:**
  - Populates `Responses ObservableCollection` with consultation history
  - Populates `Options ObservableCollection` with semester filters
  - Each Response object includes color-coded `StatusColor` property

#### **FacultyRequestViewModel.ApproveRequest:**

- **Returns:** Task (async void) • **Side Effects:**
  - Updates consultation status to `Approved` in database
  - Removes consultation from `PendingRequests` collection
  - Displays confirmation dialog before action
  - Shows cancellation if user selects "No"

#### **FacultyRequestViewModel.DisApproveRequest:**

- **Returns:** Task (async void) • **Side Effects:**
  - Updates consultation status to `Declined` in database with reason
  - Removes consultation from `PendingRequests` collection
  - Displays confirmation dialog and reason prompt
  - Shows cancellation if user cancels

#### **MenuViewModel.ChangePassword:**

**Returns:** Task (async void)

•

- **Side Effects:**
  - Displays ChangePassword popup dialog
  - Awaits user interaction with popup

#### **MenuViewModel.LogoutButton:**

- **Returns:** void (async void) • **Side Effects:**
  - Resets Application.Current.MainPage to new AppShell
  - Navigates to "//LoginPage" route
  - Clears user session data

#### **AuthService.Login:**

- **Returns:** Task<Users> - User entity if authentication succeeds, null otherwise • **Side Effects:**
  - Queries database for user with matching email and role
  - Verifies password hash using PasswordHasher

#### **AuthService.CreateAccount:**

- **Returns:** Task (async void) • **Side Effects:**
  - Creates new Users entity in database
  - Hashes password using Identity PasswordHasher
  - Assigns user to specified role via Identity
  - Displays success or error message

#### **StudentServices.GetStudentInformation:**

- **Returns:** Task<Student> - Student entity with navigation properties loaded • **Includes:**
  - Program (with Department)
  - SchoolYear
  - EnrolledCourses (with Faculty, SchoolYear, Program)
  - ConsultationRequests (with status)
  - Users (base authentication entity)

#### **StudentServices.GetStudentEnrolledCourses:**

- **Returns:** Task<List<EnrolledCourse>> - List of enrolled courses • **Includes:**
  - Faculty information
  - SchoolYear details
  - Program information

#### **FacultyServices.GetFacultyInformation:**

- **Returns:** Task<Faculty> - Faculty entity with navigation properties • **Includes:**
  - Program
  - EnrolledCourses
  - FacultySchedules
  - ConsultationRequests
  - SchoolYear
  - Users

#### **ConsultationRequestServices.AddConsultation:**

-

- **Returns:** Task (async void) • **Side Effects:**
  - Creates new ConsultationRequest entity in database
  - Sets initial status and timestamps
  - Persists to database via SaveChangesAsync

#### **FacultyRepository.FacultyConsultation:**

- **Returns:** Task<List<ConsultationRequest>> - List of consultation requests •  
**Includes:**
  - Student information (StudentName, StudentUMID)
  - Consultation details (DateSchedule, StartedTime, EndedTime, Status)

#### **Helper.GetSemesterName:**

- **Returns:** string - Display name from DisplayAttribute or enum ToString()
- **Example:** Semester.Semester1 → "First Semester"

#### **Helper.GetYearLevelName:**

- **Returns:** string - Display name from DisplayAttribute or enum ToString()
- **Example:** YearLevel.FirstYear → "First Year"

#### **Helper.DisplayMessage:**

- **Returns:** void (async operation via MainPage.DisplayAlert)
- **Side Effects:** Shows modal alert dialog with "Ok" button

#### **Helper.DisplayOption:**

- **Returns:** Task<bool> - true if first option selected, false if second option selected
- **Side Effects:** Shows modal dialog with two custom buttons
- **Helper.StringSplitter:**
  - **Returns:** List<string> - List of substrings split by delimiter

#### **LoadingServices.Show:**

**Returns:** void

- **Side Effects:** Displays loading popup overlay on current page

#### **LoadingServices.Hide:**

- **Returns:** void
- **Side Effects:** Closes and disposes loading popup if active

•

## IV. SIDE EFFECTS

### Database Operations:

#### Login Process:

- Queries Users table with Include for navigation properties
- No database modifications during authentication
- May execute multiple queries if checking both Student and Faculty roles

#### Account Creation:

- Inserts new record into Users table
- Creates corresponding Student or Faculty entity based on role
- Inserts role assignment record via Identity framework
- All operations wrapped in implicit transaction via Entity Framework

#### Consultation Request Submission:

- Inserts new ConsultationRequest record
- Sets foreign key relationships to Student and Faculty
- Initial Status set to Pending

#### Consultation Status Update:

- Updates Status field of ConsultationRequest entity
- Updates DisapprovedReason field if declining
- Modifies DateSchedule, StartedTime, or EndedTime if approved

#### Navigation Side Effects:

##### Shell Navigation:

- Changes current page in visual tree
- May trigger page lifecycle events (OnAppearing, OnDisappearing)
- Updates navigation stack for back button behavior
- Route parameters passed via query string format **TabBar Navigation:**

Switches active tab while maintaining page state

- Does not dispose inactive tab pages • Preserves view state during tab switches

##### UI State Management:

##### ObservableCollection Modifications:

- Adding items triggers CollectionChanged event

•

- UI automatically updates via data binding • Clearing collections removes all items from UI

#### **ObservableProperty Changes:**

- Property setters raise PropertyChanged event
- Bound UI controls update automatically
- No explicit UI refresh required with MVVM binding

#### **Loading Screen Display:**

- Blocks user interaction with underlying page
- Prevents multiple simultaneous operations • Must be manually hidden via LoadingServices.Hide()

#### **Static Property Storage:**

- LoginViewModel.Student and LoginViewModel.Faculty persist throughout app session
- LoginViewModel.AccountVerification determines user role context
- RequestViewModel.Semester maintains selected semester across navigation
- Static properties not cleared on logout (handled by navigation reset)

#### **Resource Management:**

##### **Popup Lifecycle:**

- LoadingScreen popup created on Show(), disposed on Hide()
- ChangePassword popup created and disposed per invocation
- Popups must be explicitly closed to prevent memory leaks

##### **Entity Framework Context:**

- DbContext registered as scoped service, disposed after operation
- Navigation properties loaded via Include() remain in memory • No explicit disposal required due to DI container management

##### **Authentication State:**

- ASP.NET Core Identity maintains authentication context
- Password hashes stored securely, plain text never persisted
- Role assignments cached by Identity framework

•

### Color-Coded Status Display:

- Response and Consultation objects calculate StatusColor in constructor
- Color mapping: Approved → Green, Declined → Red, Pending → Orange, Unknown → Gray
- Color changes require recreating object instances

### Error Handling Side Effects:

- Helper.DisplayMessage shows non-blocking modal dialogs
- Exceptions in service layer propagate to ViewModel
- Try-finally blocks ensure loading screen cleanup
- Database exceptions may leave transactions uncommitted

## H. DATABASE DESIGN

### I. ENTITY RELATIONSHIP DIAGRAM

The database implements a relational structure with the following primary entities and relationships:

#### Core Entities:

##### Users (Base authentication table)

- Primary Key: Id (string)
- Contains: UserName, Email, PasswordHash, PhoneNumber, UMID, UserType
- Relationships: One-to-One with Student, Faculty, or Admin based on UserType

##### Student

- Primary Key: StudentID (int)
- Foreign Keys: ProgramID, SchoolYearID, UsersID • Relationships:
  - o Many-to-One with Program
  - o Many-to-One with SchoolYear
  - o One-to-Many with EnrolledCourse
  - o One-to-Many with ConsultationRequest
  - o One-to-One with Users

##### Faculty

- Primary Key: FacultyID (int)
- Foreign Keys: ProgramID, SchoolYearID, UsersID • Relationships:
  - o Many-to-One with Program
  - o Many-to-One with SchoolYear
  - o One-to-Many with EnrolledCourse
  - o One-to-Many with FacultySchedule
  - o One-to-Many with ConsultationRequest
  - o One-to-One with Users



### **ConsultationRequest**

- Primary Key: ConsultationID (int) • Foreign Keys: StudentID, FacultyID • Relationships:
  - Many-to-One with Student
  - Many-to-One with Faculty
- Contains consultation scheduling and status information

### **EnrolledCourse**

- Primary Key: EnrolledCourseID (int)
- Foreign Keys: StudentID, FacultyID, SchoolYearID, ProgramID (nullable) • Relationships:
  - Many-to-One with Student
  - Many-to-One with Faculty
  - Many-to-One with SchoolYear
  - Many-to-One with Program
- Links students to courses taught by faculty

### **Program**

- Primary Key: ProgramID (int) • Foreign Key: DepartmentID • Relationships:
  - Many-to-One with Department
  - One-to-Many with Student
  - One-to-Many with Faculty
  - One-to-Many with EnrolledCourse

### **Department**

- Primary Key: DepartmentID (int) • Relationships: ◦ One-to-Many with Program

### **SchoolYear**

- Primary Key: SchoolYearID (int)
- Contains: Year1, Year2, Semester, SchoolYearStatus • Relationships:
  - One-to-Many with Student
  - One-to-Many with Faculty
  - One-to-Many with EnrolledCourse

### **Supporting Entities:**

#### **FacultySchedule**

- Primary Key: FacultyScheduleID (int)
- Foreign Key: FacultyID
- Contains: TimeStart, TimeEnd, Day
- Defines faculty availability schedule

## Notification

- Primary Key: NotificationNumber (int)
- Contains: NotificationMessage, NotificationType
- System notification records

## Bulletin

- Primary Key: BulletinID (int)
- Contains: Title, Author, Content, Status, DatePublished, FileCount, IsArchived
- Bulletin board posts

## ActionLog

- Primary Key: ActionLogID (int)
- Foreign Key: UsersID
- Contains: Username, Description, Date, Time
- Audit trail of user actions

## Admin

- Primary Key: AdminID (int)
- Foreign Key: UsersID
- Contains: AdminName, UsersID
- Administrative user profiles

## Key Relationships Summary:

- Users → Student/Faculty/Admin (1:1, discriminated by UserType)
- Student → ConsultationRequest (1:Many)
- Faculty → ConsultationRequest (1:Many)
- Student → EnrolledCourse (1:Many)
- Faculty → EnrolledCourse (1:Many)
- Program → Department (Many:1)
- SchoolYear → EnrolledCourse (1:Many)

## II. TABLE DEFINITIONS

### Users Table: *sql*

#### Table: Users

#### Columns:

- *Id (VARCHAR, Primary Key) - Unique user identifier from Identity framework*

- *UserName (VARCHAR) - Username for authentication*
- *NormalizedUserName (VARCHAR) - Uppercase username for case-insensitive searches*
- *Email (VARCHAR) - User email address*
- *NormalizedEmail (VARCHAR) - Uppercase email for case-insensitive searches*
- *EmailConfirmed (BOOLEAN) - Email verification status*
- *PasswordHash (VARCHAR) - Hashed password using Identity PasswordHasher*
- *SecurityStamp (VARCHAR) - Security token for password changes*
- *ConcurrencyStamp (VARCHAR) - Concurrency token for updates*
- *PhoneNumber (VARCHAR, Nullable) - Contact phone number*
- *PhoneNumberConfirmed (BOOLEAN) - Phone verification status*
- *TwoFactorEnabled (BOOLEAN) - Two-factor authentication status*
- *LockoutEnd (DATETIMEOFFSET, Nullable) - Account lockout expiration*
- *LockoutEnabled (BOOLEAN) - Lockout capability status*
- *AccessFailedCount (INT) - Failed login attempt counter*
- *UMID (VARCHAR) - University ID number*
- *UserType (INT) - Enum: 0=Admin, 1=Student, 2=Faculty*

#### **Student Table: *sql***

#### **Table: Student**

#### **Columns:**

- *StudentID (INT, Primary Key, Auto-increment) - Unique student identifier*
- *StudentUMID (VARCHAR) - Student university ID number*
- *StudentName (VARCHAR) - Full name of student*

-

-

*Email (VARCHAR) - Student email address yearLevel (INT) - Enum: FirstYear,*

*SecondYear, ThirdYear, FourthYear*

- *ProgramID (INT, Foreign Key → Program.ProgramID) - Academic program*
- *SchoolYearID (INT, Foreign Key → SchoolYear.SchoolYearID) - Current school year*
- *UsersID (VARCHAR, Foreign Key → Users.Id) - Authentication link*

### **Faculty Table: *sql***

#### **Table: Faculty**

##### **Columns:**

- *FacultyID (INT, Primary Key, Auto-increment) - Unique faculty identifier*
- *FacultyUMID (VARCHAR) - Faculty university ID number*
- *FacultyName (VARCHAR) - Full name of faculty*
- *FacultyEmail (VARCHAR) - Faculty email address*
- *ProgramID (INT, Foreign Key → Program.ProgramID) - Assigned program*
- *SchoolYearID (INT, Foreign Key → SchoolYear.SchoolYearID) - Current academic year*
- *UsersID (VARCHAR, Foreign Key → Users.Id) - Authentication link*

### **ConsultationRequest Table: *sql***

#### **Table: ConsultationRequest**

##### **Columns:**

- *ConsultationID (INT, Primary Key, Auto-increment) - Unique consultation identifier*
- *DateRequested (DATETIME) - Timestamp when request was created*
- *DateSchedule (DATETIME) - Scheduled consultation date*
- *StartedTime (TIME) - Consultation start time*

- 
- 
- *EndedTime (TIME) - Consultation end time*
- *Concern (VARCHAR) - Description of consultation topic*
- *DisapprovedReason (VARCHAR, Nullable) - Reason if request declined*
- *SubjectCode (VARCHAR) - Course code for consultation*
- *Status (INT) - Enum: 0=Pending, 1=Approved, 2=Declined, 3=Upcoming*
- *ProgramName (VARCHAR) - Program name*
- *StudentID (INT, Foreign Key → Student.StudentID) - Requesting student*
- *FacultyID (INT, Foreign Key → Faculty.FacultyID) - Assigned faculty*

#### **EnrolledCourse Table: *sql***

##### **Table: EnrolledCourse**

##### **Columns:**

- *EnrolledCourseID (INT, Primary Key, Auto-increment) - Unique enrollment record*
- *CourseName (VARCHAR) - Full course name*
- *CourseCode (VARCHAR) - Course identifier code*
- *SchoolYearID (INT, Foreign Key → SchoolYear.SchoolYearID) - Academic year*
- *StudentID (INT, Foreign Key → Student.StudentID) - Enrolled student*
- *FacultyID (INT, Foreign Key → Faculty.FacultyID) - Course instructor -*
- *ProgramCourse (VARCHAR, Nullable) - Program-specific course designation*

#### **Program Table: *sql***

##### **Table: Program**

##### **Columns:**

- *ProgramID (INT, Primary Key, Auto-increment) - Unique program identifier*
- *ProgramName (VARCHAR) - Official program name*

-

-

*Description (VARCHAR) - Program description*

*DepartmentID (INT, Foreign Key → Department.DepartmentID) - Owning department*

### **Department Table: sql**

#### **Table: Department**

##### **Columns:**

- *DepartmentID (INT, Primary Key, Auto-increment) - Unique department identifier*
- *DepartmentName (VARCHAR) - Department name*
- *Description (VARCHAR) - Department description*

### **SchoolYear Table: sql**

#### **Table: SchoolYear**

##### **Columns:**

- *SchoolYearID (INT, Primary Key, Auto-increment) - Unique school year identifier*
- *Year1 (VARCHAR) - Starting year (e.g., "2024")*
- *Year2 (VARCHAR) - Ending year (e.g., "2025")*
- *SchoolYearStatus (INT) - Enum: Active status indicator - Semester (INT) - Enum:*  
*0=Semester1, 1=Semester2, 2=Summer*

### **FacultySchedule Table: sql**

#### **Table: FacultySchedule**

##### **Columns:**

- *FacultyScheduleID (INT, Primary Key, Auto-increment) - Unique schedule entry*
- *TimeStart (TIME) - Availability start time*
- *TimeEnd (TIME) - Availability end time*
- Day (INT) - Enum for day of week*
- FacultyID (INT, Foreign Key → Faculty.FacultyID) - Faculty member*

-

-

### **Notification Table: *sql***

#### **Table: Notification**

##### **Columns:**

- *NotificationNumber (INT, Primary Key, Auto-increment) - Unique notification ID*
- *NotificationMessage (VARCHAR) - Notification content*
- *NotificationType (INT) - Enum: Notification category*

### **Bulletin Table: *sql***

#### **Table: Bulletin**

##### **Columns:**

- *BulletinID (INT, Primary Key, Auto-increment) - Unique bulletin identifier*
- *Title (VARCHAR) - Bulletin title*
- *Author (VARCHAR) - Author name*
- *Content (VARCHAR) - Bulletin body text*
- *Status (INT) - Enum: Publication status*
- *DatePublished (DATETIME) - Publication timestamp*
- *FileCount (INT) - Number of attached files - IsArchived (BOOLEAN) - Archive status flag*

### **ActionLog Table: *sql***

#### **Table: ActionLog**

##### **Columns:**

- ActionLogID (INT, Primary Key, Auto-increment) - Unique log entry identifier*
- Username (VARCHAR) - User who performed action*
- *Description (VARCHAR) - Action description*

- 
- 
- *Date (DATETIME) - Action date*
- *Time (TIME) - Action time*
- *UsersID (VARCHAR, Foreign Key → Users.Id) - User reference*

#### **Admin Table:**

#### **sql Table:**

#### **Admin**

#### **Columns:**

- *AdminID (INT, Primary Key, Auto-increment) - Unique admin identifier*
- *AdminName (VARCHAR) - Administrator name*
- *UsersID (VARCHAR, Foreign Key → Users.Id) - Authentication link*

#### **Enum Definitions:**

#### **Status Enum:**

- 0 = Pending
- 1 = Approved
- 2 = Declined
- 3 = Upcoming

#### **Semester Enum:**

- 0 = Semester1 (First Semester)
- 1 = Semester2 (Second Semester)
- 2 = Summer

#### **YearLevel Enum:**

- 0 = FirstYear
- 1 = SecondYear
- 2 = ThirdYear
- 3 = FourthYear

#### **UserType Enum:**



- 0 = Admin
- 1 = Student
- 2 = Faculty

#### **NotificationType Enum:**

- (Specific values defined in Consultation.Domain.Enum project)

### **III. SAMPLE DATA**

#### **Users Table Sample:**

Id: "550e8400-e29b-41d4-a716-446655440000"

UserName: "juan.dela.202401.cruz@umindanao.edu.ph" Email:

"j.cruz. 202401@umindanao.edu.ph"

PasswordHash: "AQAAAAEAACcQAAAAEH..." (hashed)

UMID: "202401"

UserType: 1 (Student)

PhoneNumber: "09171234567"

#### **Student Table Sample:**

StudentID: 1

StudentUMID: "202401"

StudentName: "Juan Dela Cruz"

Email: " j.cruz. 202401@umindanao.edu.ph " yearLevel: 2

(ThirdYear)

ProgramID: 1

SchoolYearID: 1

UsersID: "550e8400-e29b-41d4-a716-446655440000"

#### **Faculty Table Sample:**

FacultyID: 1

FacultyUMID: "200501"

FacultyName: "Dr. Maria Santos"

FacultyEmail: "m. santos.200501.faculty@umindanao.edu.ph"

ProgramID: 1

SchoolYearID: 1

UsersID: "660f9511-f39c-52e5-b827-557766551111"

**Program Table Sample:**

ProgramID: 1

ProgramName: "BSCpE"

Description: "Bachelor of Science in Computer Engineering"

DepartmentID: 1

**Department Table Sample:**

DepartmentID: 1

DepartmentName: "College of Engineering Education"

Description: "Department Engineering Education"

**SchoolYear Table Sample:**

SchoolYearID: 1

Year1: "2024"

Year2: "2025"

Semester: 0 (Semester1 - First Semester)

SchoolYearStatus: 1 (Active)

**EnrolledCourse Table Sample:**

EnrolledCourseID: 1

CourseName: "Data Structures and Algorithms"

CourseCode: "CS201"

SchoolYearID: 1

StudentID: 1

FacultyID: 1

ProgramCourse: "BSCS Core Course"

**ConsultationRequest Table Sample:**

ConsultationID: 1

DateRequested: "2025-01-15 10:30:00"

DateSchedule: "2025-01-20"

StartedTime: "14:00:00"

EndedTime: "15:00:00"

Concern: "Need clarification on linked lists implementation"

DisapprovedReason: NULL

SubjectCode: "CS201"

Status: 0 (Pending)

ProgramName: "BSCS"

StudentID: 1

FacultyID: 1

**FacultySchedule Table Sample:**

**FacultyScheduleID: 1**

TimeStart: "10:00:00"

TimeEnd: "12:00:00"

Day: 1 (Monday)

FacultyID: 1

**Bulletin Table Sample:**

BulletinID: 1

Title: "Midterm Examination Schedule"

Author: "Academic Affairs"

Content: "Midterm exams will be held from February 1-7, 2025..."

Status: 1 (Published)

DatePublished: "2025-01-10 08:00:00"

FileCount: 0

IsArchived: FALSE

### **ActionLog Table Sample:**

#### **ActionLogID: 1**

Username: "juan.dela.202401.cruz@umindanao.edu.ph"

Description: "Student logged in to the system"

Date: "2025-01-15"

Time: "08:30:00"

UsersID: "550e8400-e29b-41d4-a716-446655440000"

### **Sample Data Relationships:**

- Student (StudentID: 1) enrolled in EnrolledCourse (EnrolledCourseID: 1)
- EnrolledCourse taught by Faculty (FacultyID: 1)
- Student submitted ConsultationRequest (ConsultationID: 1) to Faculty
- ConsultationRequest status is Pending awaiting faculty approval
- Faculty has defined availability in FacultySchedule (Mondays 10:00-12:00)

## **I. TROUBLESHOOTING & DEBUGGING TIPS**

### **COMMON ISSUES**

#### **1. Database Connection Failures:**

Symptom: Application crashes or shows "Unable to connect to database" errors on startup or during data operations.

#### **Causes:**

- Network connectivity issues to Azure MySQL server
- Incorrect connection string configuration
- Firewall rules blocking access to Azure port 3306
- SSL certificate validation failures

### **Solutions:**

- Verify network connectivity: ping `consultationdb.mysql.database.azure.com`
  - Check connection string in `MauiProgram.cs` for correct credentials
  - Ensure Azure MySQL firewall allows your IP address
  - Verify SSL mode is set to "Required" in connection string
  - Test connection using MySQL Workbench with same credentials
  - Check Entity Framework retry logic is functioning (max 5 retries with 10-second delays)
- 

## **2. Navigation Not Working:**

Symptom: Clicking buttons or navigation links does not change pages or throws `ArgumentException: Route not found.`

### **Causes:**

- Route not registered in `AppShell.xaml.cs`
- Incorrect route name in `GoToAsync()` call
- Shell navigation system not initialized • Attempting navigation before `MainPage` is set

### **Solutions:**

- Verify route registration in `AppShell.xaml.cs` Routing
  - Check route name spelling matches exactly between registration and navigation
  - Ensure `AppShell` is set as `MainPage` in `App.xaml.cs`: `MainPage = new AppShell();`
  - Use correct route format: `await Shell.Current.GoToAsync("//RouteName")` for absolute paths
  - Check that navigation is called after page initialization (not in constructor)
  - Clear navigation stack if experiencing stack overflow:  
`Application.Current.MainPage = new AppShell();`
- 

## **3. Data Binding Issues:**

Symptom: UI elements not displaying data, or changes not reflecting in the interface despite data updates.

### **Causes:**

- Missing `[ObservableProperty]` attribute on ViewModel properties
- `BindingContext` not set correctly in code-behind or XAML
- Property names don't match between XAML binding and ViewModel
- `ObservableCollection` not being used for list data

- PropertyChanged events not firing

#### **Solutions:**

- Verify [ObservableProperty] attribute is present on all bindable properties
  - Ensure BindingContext is set in page constructor or XAML: BindingContext = viewModel;
  - Check property name casing matches exactly (remember CommunityToolkit generates PascalCase properties from camelCase fields)
  - Use ObservableCollection<T> instead of List<T> for dynamic lists
  - Inherit ViewModels from ObservableObject base class
  - Check XAML binding paths are correct: Text="{Binding PropertyName}"
  - Enable XAML compilation errors: Right-click XAML file → Properties → Build Action → Embedded resource
- 

### **4. Authentication Failures:**

Symptom: Valid credentials rejected, or incorrect role routing after login.

#### **Causes:**

- Password hashing mismatch between creation and verification
- User role not properly assigned during account creation
- Multiple user records with same email but different roles
- Incorrect UserType enum value stored in database

#### **Solutions:**

- Verify password hashing uses same PasswordHasher instance: IPasswordHasher<Users>
  - Check role assignment in AuthService.CreateAccount() calls AddToRoleAsync()
  - Query database to ensure only one user exists per email
  - Verify UserType enum values: 0=Admin, 1=Student, 2=Faculty
  - Test with known working credentials from database seeder
  - Check LoginViewModel.Student and LoginViewModel.Faculty are being set correctly
  - Verify AccountVerification boolean flag matches user role
- 

### **5. NuGet Package Restore Errors:**

Symptom: Build errors about missing types, namespaces, or package references.

#### **Causes:**

- NuGet packages not fully restored
  - Package version conflicts between projects
  - Corrupted package cache
  - Missing project references to supporting projects **Solutions:**
    - Right-click solution → Restore NuGet Packages
    - Clean solution: Build → Clean Solution
    - Delete bin and obj folders manually from all projects
    - Clear NuGet cache: Tools → Options → NuGet Package Manager → Clear All NuGet Cache(s)
    - Verify all project references exist: Consultation.Domain, Consultation.Repository, Consultation.Services, Consultation.Infrastructure, Enum
    - Check .csproj file for correct PackageReference versions
    - Rebuild solution: Build → Rebuild Solution
- 

## 6. Android Deployment Failures:

Symptom: App fails to deploy to Android emulator or device, or crashes immediately on launch.

### Causes:

- Android SDK version mismatch
- Insufficient device storage
- Missing Android permissions in manifest
- AOT compilation errors

### Solutions:

- Verify Android SDK API Level 23+ installed
  - Check emulator has sufficient storage (minimum 4GB free) • Verify
- AndroidManifest.xml includes required permissions: *xml*

*<uses-permission android:name="android.permission.INTERNET" />*

*<uses-permission  
android:name="android.permission.ACCESS\_NETWORK\_STATE" />*

- Disable AOT compilation for debugging: Project Properties → Android Options → Uncheck "Enable AOT compilation"
  - Try different emulator device configuration
  - Enable USB debugging on physical device
  - Check Android logs: View → Other Windows → Device Log
-

## 7. ObservableCollection Not Updating UI:

Symptom: Adding or removing items from ObservableCollection doesn't update ListView or CollectionView.

### Causes:

- Collection reassigned instead of modified
- Collection not initialized before view binding
- ItemsSource binding pointing to wrong property
- UI thread synchronization issues

### Solutions:

- Modify existing collection instead of reassigning: Use collection.Add() not collection = new ObservableCollection()
  - Initialize collections in constructor or field initializer:  
public ObservableCollection<T> Items { get; } = new();
  - Verify ItemsSource binding: ItemsSource="{Binding CollectionPropertyName}"
  - Ensure collection modifications on UI thread: Wrap in  
MainThread.BeginInvokeOnMainThread(() => { ... })
  - Check that BindingContext is set before populating collection
- 

## 8. Loading Screen Not Dismissing:

Symptom: Loading popup remains visible indefinitely, blocking user interaction.

### Causes:

- Exception thrown before LoadingServices.Hide() called
- Missing finally block ensuring cleanup
- Multiple Show() calls without corresponding Hide() calls
- Popup reference lost or nulled prematurely

### Solutions:

- Wrap async operations in try-finally blocks:

*csharp*

```
try {  
  
    _loadingScreen.Show();  
  
    await LongRunningOperation();  
  
}  
  
finally {
```



```
_loadingScreen.Hide();  
  
}
```

- Verify LoadingServices implements proper cleanup in Hide() method
- Check for multiple Show() calls without intervening Hide()
- Add exception logging to identify where failures occur
- Test loading screen separately to ensure popup lifecycle works

---

## 9. Dependency Injection Failures:

Symptom: `NullReferenceException` when accessing injected services, or "No service for type" errors.

### Causes:

- Service not registered in MauiProgram.cs
- Incorrect service lifetime (Singleton vs Transient vs Scoped)
- Constructor injection parameter name mismatch • Service implementation not implementing interface

### Solutions:

- Verify service registration in MauiProgram.cs:

*csharp*

```
builder.Services.AddTransient<IServiceInterface, ServiceImplementation>();
```

- Check constructor parameters match registered interfaces
- Ensure ViewModels are registered with AddTransient or AddSingleton
- Verify Pages are registered in DI container
- Use correct lifetime: AddTransient for per-use, AddSingleton for app-wide
- Check service implementation inherits/implements the registered interface

---

## 10. Entity Framework Query Failures:

Symptom: Queries return null or incomplete data, or throw "Object reference not set" errors.

### Causes:

- Navigation properties not eager loaded with Include()
- Lazy loading disabled (EF Core default)
- Database seeding not executed
- Connection string pointing to empty database

### **Solutions:**

- Use `.Include()` for navigation properties:

***csharp** **await***

***\_context.Students***

***.Include(s => s.Program)***

***.Include(s => s.EnrolledCourses)***

***.FirstOrDefaultAsync(s => s.StudentUMID == umid);***

- Verify database contains seeded data using MySQL Workbench
  - Check DbContext connection string is correct
  - Run migrations if schema changes made: Tools → NuGet Package Manager → Package Manager Console → Update-Database
  - Test queries directly in MySQL Workbench to verify data exists
- 

## **DEBUGGING STRATEGIES**

### **1. Breakpoint Debugging:**

#### **Setting Breakpoints:**

- Click in left margin of code editor to set breakpoint (red dot appears)
- Set conditional breakpoints: Right-click breakpoint → Conditions
- Use data breakpoints for property changes: Debug → New Breakpoint → Data Breakpoint

#### **Key Debugging Locations:**

- ViewModel command methods (before and after service calls)
- Service layer methods (to verify business logic execution)
- Repository methods (to inspect database queries)
- LoginViewModel.ClickLogin() to trace authentication flow
- Helper.DisplayMessage() to verify message display
- Database context SaveChangesAsync() calls

#### **Inspecting Variables:**

- Hover over variables to see current values
- Use Locals window (Debug → Windows → Locals) to see all local variables

- Watch specific expressions (Debug → Windows → Watch) for complex objects • Use Immediate Window (Debug → Windows → Immediate) to execute code during debugging
- 

## 2. Logging and Output:

### Debug Output:

- Use Debug.WriteLine() for development logging: *csharp*

*Debug.WriteLine(\$"Login attempt: Email={Email}, Role={role}");*

- View output in Output window: View → Output (select "Debug" from dropdown)

### Custom Logging:

- Implement logging in service methods:

*csharp*

*try {*

*var result = await \_repository.GetStudentByUMID(umid);*

*Debug.WriteLine(\$"Student retrieved: {result?.StudentName ?? "Not found"}");*

*return result;*

*}*

*catch (Exception ex) {*

*Debug.WriteLine(\$"Error in GetStudentInformation: {ex.Message}");*

*throw;*

*}*

### Exception Logging:

- *Catch and log exceptions in ViewModels:*

*csharp catch (Exception*

*ex) {*

```

        Helper.DisplayMessage($"Error: {ex.Message}");

        Debug.WriteLine($"Exception details: {ex.ToString()}");
    }

```

---

### 3. Database Query Inspection:

Enable EF Core Logging: Add to MauiProgram.cs during DbContext configuration: *csharp*

```

builder.Services.AddDbContext<AppDbContext>(opt =>
{
    opt.UseMySQL(cs, serverVersion)

    .LogTo(Console.WriteLine, LogLevel.Information) // Log SQL queries

    .EnableSensitiveDataLogging(); // Show parameter values (development only)
});

```

#### View Generated SQL:

- Check Output window for SQL queries when queries execute
- Verify Include() statements generate proper JOIN queries
- Identify N+1 query problems (multiple queries in loops)

#### Direct Database Testing:

- Use MySQL Workbench to run queries directly
  - Verify data exists before testing application queries
  - Check for data type mismatches or null values
- 

### 4. UI Debugging:

#### XAML Live Visual Tree:

- Debug → Windows → Live Visual Tree (while app running)
- Inspect UI element hierarchy
- Verify BindingContext propagation through visual tree

#### XAML Live Property Explorer:

- Debug → Windows → Live Property Explorer
- View current property values of selected UI element
- Check binding sources and data contexts

### Enable XAML Compilation:

- Catches XAML errors at compile time instead of runtime
- Set in .csproj: <XamlCompilationOptions>Compile</XamlCompilationOptions>

### Binding Error Debugging:

- Check Output window for binding failures (search for "Binding:")
- Common errors: "Path not found", "Property not found on object"
- Verify property names match exactly (case-sensitive)

## 5. Network and API Debugging:

### Monitor Database Connections:

- Use Fiddler or Charles Proxy to monitor network traffic
- Verify SSL handshake succeeds
- Check for timeout errors or connection rejections

### Test Connection Separately: *csharp private async*

#### *Task TestDatabaseConnection()*

```
{
    try
    {
        using var context = new AppDbContext();    var canConnect = await
context.Database.CanConnectAsync();

        Debug.WriteLine($"Database connection: {(canConnect ? "Success" :
"Failed")}");
    }

    catch (Exception ex)
    {
```

```
Debug.WriteLine($"Connection error: {ex.Message}");

}

}
```

---

## 6. Memory and Performance Debugging: Memory

### Profiling:

- Debug → Windows → Diagnostic Tools
- Monitor memory usage during navigation and data loading
- Identify memory leaks from unsubscribed events or retained references

### Performance Profiling:

- Analyze → Performance Profiler
- Identify slow database queries or UI rendering bottlenecks
- Profile async operations for deadlocks

### Common Memory Issues:

- Event handlers not unsubscribed (especially in ViewModels)
  - Static properties holding large objects
  - Navigation stack not cleared causing page accumulation
- 

## 7. Platform-Specific Debugging:

### Android Debugging:

- View → Other Windows → Device Log
- Filter by package name: com.companyname.umconsultationappmaui
- Look for Java exceptions or native crashes

### Windows Debugging:

- Standard Visual Studio debugging fully supported
  - Check Event Viewer for application crashes
  - Verify Windows-specific dependencies are installed
- 

## UNIT TEST LOCATION AND USAGE

### Recommended Test Project Structure:

## UM-Consultation-App-MAUI.Tests/

- |— ViewModelTests/
  - | |— LoginViewModelTests.cs
  - | |— HomeViewModelTests.cs
  - | |— RequestViewModelTests.cs
  - | |— FacultyRequestViewModelTests.cs
- |— ServiceTests/
  - | |— AuthServiceTests.cs
  - | |— StudentServicesTests.cs
  - | |— ConsultationRequestServicesTests.cs
- |— RepositoryTests/
  - | |— StudentRepositoryTests.cs
  - | |— FacultyRepositoryTests.cs
- |— Helpers/
  - | |— MockDataHelper.cs

## Setting Up Unit Tests:

### 1. Add Test Project:

- o Right-click Solution → Add → New Project o Select "xUnit Test Project" or "NUnit Test Project" o Name: UM-Consultation-App-MAUI.Tests o Target framework: net8.0

### 2. Install Required Packages: *xml*

*<PackageReference Include="xunit" Version="2.6.0" />*

*<PackageReference Include="Moq" Version="4.20.0" />*

*<PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="8.0.0" />*

### 3. Add Project References:

- o Reference all projects: UM-Consultation-App-MAUI,

Consultation.Domain, Consultation.Services, Consultation.Repository,  
Consultation.Infrastructure

### Sample Test Implementation: csharp

// *LoginViewModelTests.cs* public

```
class LoginViewModelTests
```

```
{
```

```
    private readonly Mock<IAuthService> _mockAuthService;    private  
    readonly Mock<IStudentServices> _mockStudentServices;    private  
    readonly Mock<IFacultyServices> _mockFacultyServices;    private  
    readonly Mock<ILoadingServices> _mockLoadingServices;
```

```
    public LoginViewModelTests()
```

```
    {
```

```
        _mockAuthService = new Mock<IAuthService>();  
        _mockStudentServices = new Mock<IStudentServices>();  
        _mockFacultyServices = new Mock<IFacultyServices>();  
        _mockLoadingServices = new Mock<ILoadingServices>();
```

```
    }
```

```
    [Fact]
```

```
    public async Task  
    ClickLogIn_ValidStudentCredentials_NavigatesToStudentDashboard()  
    {
```

```
        // Arrange
```

```
        var testUser = new Users { UMID = "202401", Email =
```



```

"test@umindanao.edu.ph" };      var testStudent = new Student {

StudentUMID = "202401" };

    _mockAuthService.Setup(x => x.Login(It.IsAny<string>(), It.IsAny<string>(),
"Student"))

        .ReturnsAsync(testUser);

    _mockStudentServices.Setup(x                =>
x.GetStudentInformation(It.IsAny<string>()))

        .ReturnsAsync(testStudent);

var viewModel = new LoginViewModel(_mockAuthService.Object,

        _mockStudentServices.Object,

        _mockFacultyServices.Object,

        _mockLoadingServices.Object)

{

    Email = "test@umindanao.edu.ph",

    Password = "TestPassword123"

};

// Act

await viewModel.ClickLogInCommand.ExecuteAsync(null);

// Assert

Assert.True(LoginViewModel.AccountVerification);

Assert.NotNull(LoginViewModel.Student);

    _mockLoadingServices.Verify(x => x.Show(), Times.Once);
    _mockLoadingServices.Verify(x => x.Hide(), Times.Once);

}

```

}

### **Running Tests:**

- Test → Run All Tests (Ctrl+R, A)
- Test → Test Explorer to view individual test results
- View code coverage: Test → Analyze Code Coverage for All Tests

### **Key Testing Areas:**

- ViewModel command execution and navigation logic
- Service layer business rules and data validation
- Repository database query results (using InMemory database)
- Helper utility methods (GetSemesterName, StringSplitter, etc.)
- Authentication and authorization workflows

### **Mocking Best Practices:**

- Mock external dependencies (IAuthService, database repositories)
- Use InMemoryDatabase for Entity Framework testing
- Test both success and failure scenarios
- Verify method calls using Moq's Verify()
- Test async operations with proper await handling

## **J. VERSION CONTROL & BUILD INSTRUCTIONS**

### **GIT USAGE GUIDELINES**

Repository Structure: The project uses Git for version control with the following branch strategy:

#### **Branch Naming Conventions:**

- master or main - Production-ready code
- develop - Integration branch for features
- feature/feature-name - Individual feature development
- bugfix/bug-description - Bug fixes
- hotfix/critical-fix - Emergency production fixes

#### **Commit Message Format:**

*<type>: <short summary>*

*<detailed description (optional)>*

*<footer (optional)>*

#### **Commit Types:**

- feat: - New feature
- fix: - Bug fix
- docs: - Documentation changes
- style: - Code formatting, no logic change
- refactor: - Code restructuring, no behavior change
- test: - Adding or updating tests
- chore: - Maintenance tasks, dependency updates **Examples:**

feat: Add consultation approval functionality for faculty

Implemented approve/disapprove commands in FacultyRequestViewModel with confirmation dialogs and database status updates. **fix:** Resolve navigation issue after consultation submission

Navigation was failing due to incorrect route name in RequestConsultationViewModel.

Changed route from "HomePage" to "///Student" for absolute navigation.

**docs:** Update installation instructions with Azure database details

### **Git Workflow:**

#### **1. Clone Repository: *bash***

```
git clone https://github.com/[organization]/proto-sd-project.git cd
```

```
proto-sd-project
```

#### **2. Create Feature Branch: *bash* *git checkout -b feature/your-feature-name***

#### **3. Stage and Commit Changes:**

```
bash
```

```
git add .
```

```
git commit -m "feat: Implement new feature description"
```

#### **4. Push to Remote:**

```
bash git push origin feature/your-feature-  
name
```

#### **5. Create Pull Request:**

- Navigate to repository on GitHub/Azure DevOps
- Create pull request from feature branch to develop
- Request code review from team members • Address review comments and push updates

## **6. Merge After Approval: *bash git***

*checkout develop git pull origin develop*

*git merge feature/your-feature-name git*

*push origin develop*

## **7. Delete Feature**

### **Branch:**

***bash** git branch -d feature/your-feature-name git push*

*origin --delete feature/your-feature-name*

### **Handling Merge Conflicts:**

***bash***

***# Update local branch with latest develop git***

*checkout feature/your-feature-name git fetch*

*origin git merge origin/develop*

***# Resolve conflicts in Visual Studio or manually***

***# After resolving, stage and commit***

*git add .*

*git commit -m "fix: Resolve merge conflicts with develop" git*

*push origin feature/your-feature-name*

*.gitignore Configuration: The project includes a comprehensive .gitignore file excluding:*

- Build artifacts (bin/, obj/, [Dd]ebug/, [Rr]elease/)
- User-specific files (\*.suo, \*.user)
- Visual Studio cache (.vs/)
- NuGet packages (\*.nupkg) • Temporary files (\*.tmp, \*.log)

### **Files to Never Commit:**

- Database credentials (use environment variables or secure config)
- API keys or secrets
- Personal development settings
- Large binary files without LFS

### **Large File Storage (if needed):**

*bash*

*# Install Git LFS*

*git lfs install*

*# Track large files git*

*lfs track "\*.png" git lfs*

*track "\*.jpg"*

*# Commit .gitattributes git add .gitattributes git commit -m*

*"chore: Configure Git LFS for image files"*

---

## **BUILD TOOLS AND COMMANDS**

### **Visual Studio Build Commands:**

#### **Build Solution:**

*bash*

*# From Developer Command Prompt for VS 2022 msbuild UM-*

**Consultation-App-MAUI.sln /p:Configuration=Debug**

#### **Rebuild Solution (Clean + Build):**

*bash msbuild UM-Consultation-App-MAUI.sln /t:Rebuild*

*/p:Configuration=Release*

### **Build Specific Project:**

***bash***

```
msbuild UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj  
/p:Configuration=Debug
```

### **Clean Solution:**

***bash msbuild UM-Consultation-App-MAUI.sln***

*/t:Clean*

---

### **.NET CLI Commands: Restore NuGet**

**Packages:** ***bash dotnet restore UM-Consultation-***

*App-MAUI.sln*

### **Build Solution:**

***bash dotnet build UM-Consultation-App-MAUI.sln --configuration***

*Debug*

### **Build for Specific Platform:**

***bash***

***# Android***

*dotnet build UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj -f net8.0-*  
*android # Windows*

*dotnet build UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj -f net8.0-*  
*windows10.0.19041.0*

### **Run Application:**

***bash***

```
dotnet run --project UM-Consultation-App-MAUI/UM-Consultation-  
AppMAUI.csproj
```

## Android-Specific Build Commands:

### Build APK (Debug):

***bash***

```
dotnet build UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj -f net8.0-android -c Debug
```

### Build APK (Release): ***bash***

```
dotnet build UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj -f net8.0-android -c Release
```

### Deploy to Device:

***bash***

```
dotnet build UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj -f net8.0-android -t:Run
```

**Generate Signed APK: *bash***

```
dotnet publish UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj -f net8.0-android -c Release /p:AndroidSigningKeyStore=mykey.keystore /p:AndroidSigningKeyAlias=myalias /p:AndroidSigningStorePass=password /p:AndroidSigningKeyPass=password
```

---

## Windows-Specific Build Commands:

### Build MSIX Package: ***bash***

```
dotnet publish UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj -f net8.0-windows10.0.19041.0 -c Release /p:GenerateAppxPackageOnBuild=true
```

### Build for Windows Machine: ***bash***

```
dotnet build UM-Consultation-App-MAUI/UM-Consultation-App-MAUI.csproj -f net8.0-windows10.0.19041.0
```

---

## Build Configuration Options:

### Configuration Flags:

- -c Debug or /p:Configuration=Debug - Debug build with symbols
- -c Release or /p:Configuration=Release - Optimized production build

## Platform-Specific Symbols: **csharp**

### **#if ANDROID**

*// Android-specific code*

### **#if WINDOWS**

*// Windows-specific code*

### **#endif**

## Conditional Compilation in .csproj: xml

```
<PropertyGroup Condition="''$(TargetFramework)' == 'net8.0-android'">
```

```
<SupportedOSPlatformVersion>23.0</SupportedOSPlatformVersion> </PropertyGroup>
```

---

## Continuous Integration Build Script Example: yml

### *# Azure DevOps Pipeline*

trigger:

branches:

include: -

main -

develop **pool:**

vmImage: 'windows-latest' **steps:**

- task: UseDotNet@2

inputs:

version: '8.0.x'

- task: NuGetCommand@2

**inputs:**



```
command: 'restore'    restoreSolution: 'UM-
Consultation-App-MAUI.sln'

- task: DotNetCoreCLI@2  displayName: 'Build Android'  inputs:

    command: 'build'    projects: 'UM-Consultation-App-MAUI/UM-Consultation-
App-MAUI.csproj'    arguments: '-f net8.0-android -c Release' - task:

PublishBuildArtifacts@1  inputs:

    PathToPublish: '$(Build.ArtifactStagingDirectory)'

    ArtifactName: 'drop'
```

---

## Troubleshooting Build Issues:

### Clear Build Cache:

*bash*

*# Delete bin and obj folders*

```
rmdir /s /q bin rmdir /s /q obj
```

*# Clear NuGet cache*

```
dotnet nuget locals all --clear
```

**Verbose Build Output:** *bash* dotnet build -v detailed UM-

Consultation-App-MAUI.sln

### Check Build Logs:

- Visual Studio: View → Output → Show output from: Build
- Check msbuild.log in project directory for detailed errors

### Common Build Errors:

- "CS0246: The type or namespace name could not be found" - Missing NuGet package or project reference
- "NETSDK1045: The current .NET SDK does not support targeting .NET 8.0" - Update Visual Studio or install .NET 8.0 SDK

- "XA0000: Unsupported API level" - Update Android SDK in Visual Studio Android SDK Manager