

מטלת סיכום ב- MAUI

PhoneBook

מגיש : אחמד אגבאריה

תז : 034240168

מייל : ahmad081177@gmail.com

מדריך : טל סימון

השתלמות קיץ יולי 2024

Contents

4.....	על הפרוייקט
4.....	טכנולוגיות
4.....	MVVM (Model-View-ViewModel)
4.....	1. Model (מודל)
4.....	2. View (תצוגה)
4.....	3. ViewModel (מודל-תצוגה)
5.....	כיצד זה עובד:
5.....	יתרונות MVVM:
5.....	Dependency Injection (DI)
5.....	מטרות DI:
5.....	כיצד זה עובד:
6.....	שימוש ב-DI ב:MAUI -
7.....	יתרונות DI:
8.....	הררכית הפרוייקט:
9.....	הסבר כללי על מבנה הפרוייקט:
9.....	תיקיות עיקריות:
11.....	מודלים
11.....	AppUser
12.....	AppContact
14.....	מסכים של האפלקציה
14.....	Register הרשמה
16.....	Login רישום כניסה
17.....	עדכון פרטים
18.....	רשימת אנשי הקשר:
19.....	יצירת איש קשר
21.....	אנשי הקשר המעודכנים
24.....	איש קשר
25.....	הודעות שגיאה:
25.....	כניסה עם פרטים לא תקינים:
26.....	רישום פעמיים – אותו user
27.....	מחלקות
27.....	Services
27.....	ContactService
28.....	UserServices
30.....	Storage

30.....	ContactsStorage
32.....	SecureUserStorageService
34.....	ViewModels
34.....	ViewModelBase
35.....	AllContactsViewModel
37.....	AppViewModel
39.....	ContactViewModel
41.....	PhoneBook
42.....	NewContactViewModel
46.....	ProfileViewModel
49.....	RegisterViewModel
54.....	Main Classes:
54.....	App.xaml
55.....	App.xaml.cs
56.....	AppShell.xaml
57.....	MauiProgram
59.....	Views
59.....	LoginPage.xaml
60.....	Login.xaml.cs
61.....	RegistrationPage.xaml
65.....	RegistrationPage
66.....	ProfilePage.xaml
68.....	ProfilePage.xaml.cs
69.....	AllContactsPage
71.....	AllContactsPage.xaml.cs
73.....	ContactPage
74.....	ContactPage.xaml.cs
75.....	NewContactPage.xaml
77.....	NewContactPage.xaml.cs
78.....	References

על הפרויקט

הפרויקט "Phone Book" הוא יישום לניהול אנשי קשר שנבנה באמצעות C# ו-.NET MAUI (Multi-platform App UI) המאפשר פיתוח יישומים חוצי-פלטפורמות. היישום עושה שימוש בתבניות ארכיטקטורה מודרניות כגון Dependency Injection (DI) ו-Model-View-ViewModel (MVVM) על מנת להבטיח הפרדה ברורה בין הלוגיקה העסקית לממשק המשתמש, דבר שתורם לקוד נקי, מודולרי וקל לתחזוקה.

מטרת היישום היא לספק פתרון מקיף לניהול אנשי קשר, בו המשתמשים יכולים לאחסן, לצפות, לערוך ולמחוק אנשי קשר בקלות. הפרויקט משתמש ב-DI לניהול התלות בין רכיבי התוכנה, מה שמאפשר גמישות גבוהה ובדיקות יעילות יותר. שימוש במודל MVVM מאפשר תקשורת ברורה בין ממשק המשתמש (View) ללוגיקה העסקית (ViewModel), תוך שמירה על ממשק תגובתי וקל לתחזוקה.

תכונות עיקריות של היישום כוללות עיצוב ממשק רספונסיבי התומך במספר פלטפורמות, סנכרון נתונים בזמן אמת, וניווט חלק בין מסכי היישום השונים. הפרויקט מהווה דוגמה מעשית ליישום של C# MAUI בפתרון בעיות מהעולם האמיתי, תוך הדגשת היתרונות בשימוש בשיטות פיתוח מודרניות.

טכנולוגיות

MVVM (Model-View-ViewModel)

היא תבנית ארכיטקטונית שנועדה להפריד בין ממשק המשתמש (UI) לבין הלוגיקה העסקית ביישומים. התבנית מחלקת את הקוד לשלושה חלקים עיקריים:

1. Model (מודל)

מייצג את הנתונים ואת הלוגיקה של היישום. הוא אחראי על ניהול המידע ועל השכבה העסקית, כמו קבלת נתונים ממסד נתונים, APIs, או כל מקור חיצוני אחר. במילים פשוטות, ה-Model אחראי על המידע עצמו ואיך הוא מתנהל.

2. View (תצוגה)

החלק שמוצג למשתמש – ממשק המשתמש. כאן מוצגים הנתונים בצורה ויזואלית, כולל כפתורים, טקסטים, טבלאות וכו'. ה-View בדרך כלל מקושר ל-ViewModel כדי להציג את המידע בצורה דינמית.

3. ViewModel (מודל-תצוגה)

גשר בין ה-Model לבין ה-View. הוא מכיל את הלוגיקה שנדרשת כדי להציג את הנתונים ב-View. ה-ViewModel מקבל מידע מה-Model ומעדכן את ה-View בהתאם.

התקשורת נעשית בצורה דו-כיוונית, כלומר כשהמשתמש מבצע פעולה כלשהי, ה-ViewModel מעדכן את ה-Model, ולהיפך.

כיצד זה עובד:

- ה-View מאזין לשינויים ב-ViewModel, שמספק את הנתונים. לדוגמה, אם נוספו אנשי קשר חדשים, ה-View ישתנה אוטומטית כדי להציג אותם.
- ה-ViewModel אחראי על קבלת הנתונים מה-Model, ועיבודם לפני העברתם ל-View. כמו כן, הוא מקבל פעולות מה-View (כמו לחיצה על כפתור) ושולח את המידע ל-Model.

יתרונות MVVM:

- הפרדה ברורה בין ממשק המשתמש לבין הלוגיקה העסקית.
- תחזוקה קלה יותר של הקוד, מכיוון שכל חלק מופרד מהשני.
- בדיקות יחידה קלות יותר – מאחר שה-ViewModel מופרד מה-View, ניתן לבדוק את הלוגיקה של היישום בלי לבדוק את ממשק המשתמש.

Dependency Injection (DI)

הזרקת תלויות בעברית, היא טכניקה בארכיטקטורת תוכנה שמאפשרת ניהול והזרקת של אובייקטים שנדרשים למחלקות (תלויות) בצורה חכמה ומבוזרת. ב-DI, במקום שמחלקה תיצור אובייקטים של התלויות שלה בעצמה, הן מוזרקות לה מבחוץ, לרוב דרך הקונסטרוקטור, באמצעות ממשקים.

מטרות DI:

1. הפרדה בין מחלקות: DI מאפשר למחלקות להיות פחות תלויות אחת בשנייה. בכך מתאפשרת גמישות ושימוש חוזר בקוד, ומפחיתים את הצימוד (tight coupling).
2. בדיקות קוד קלות יותר: מכיוון שהמחלקות מקבלות את התלויות שלהן מבחוץ, אפשר בקלות להחליף את התלויות במימושים מזויפים (mocking) לצורך בדיקות יחידה.
3. ניהול חכם של תלותיות: מונע יצירת אובייקטים מיותרים ומפחית עומס על המערכת בכך שמנהלים את מחזור חיי האובייקטים דרך המערכת.

כיצד זה עובד:

במקום שמחלקה תייצר בעצמה אובייקטים של תלויות לה (כמו שירותי גישה לנתונים או שירותים חיצוניים), היא מקבלת אותם מהמנגנון של DI. למשל, במערכת C# MAUI, DI מיושם באופן מובנה דרך הוספת השירותים לאוסף השירותים של היישום.

דוגמה פשוטה :

נניח שיש לנו מחלקה שמנהלת רשימת אנשי קשר והיא תלויה בשירות גישה לנתונים.

10 references

```
public class ContactService
{
    private readonly IContactRepository _contactRepository;

    1 reference
    public ContactService(IContactRepository contactRepository)
    {
        _contactRepository = contactRepository;
    }

    0 references
    public List<Contact> GetContacts()
    {
        return _contactRepository.GetAll();
    }
}
```

2 references

```
public interface IContactRepository
{
    1 reference
    List<Contact> GetAll();
}
```

כאן, ה ContactService-תלוי ב- IContactRepository, שהוא ממשק לאחסון או שליפה של אנשי קשר. במקום שהמחלקה תיצור אובייקט של ContactRepository היא מקבלת את המימוש שלו דרך הבנאי.

שימוש ב DI-ב:MAUI-

בפרויקטים של MAUI, מוגדר בקובץ MauiProgram.cs, שבו מוסיפים את השירותים הנדרשים לקונפיגורציה :

```
// DI-הוספת שירותים ל
builder.Services.AddSingleton<IContactRepository, ContactRepository>();
builder.Services.AddSingleton<ContactService>();

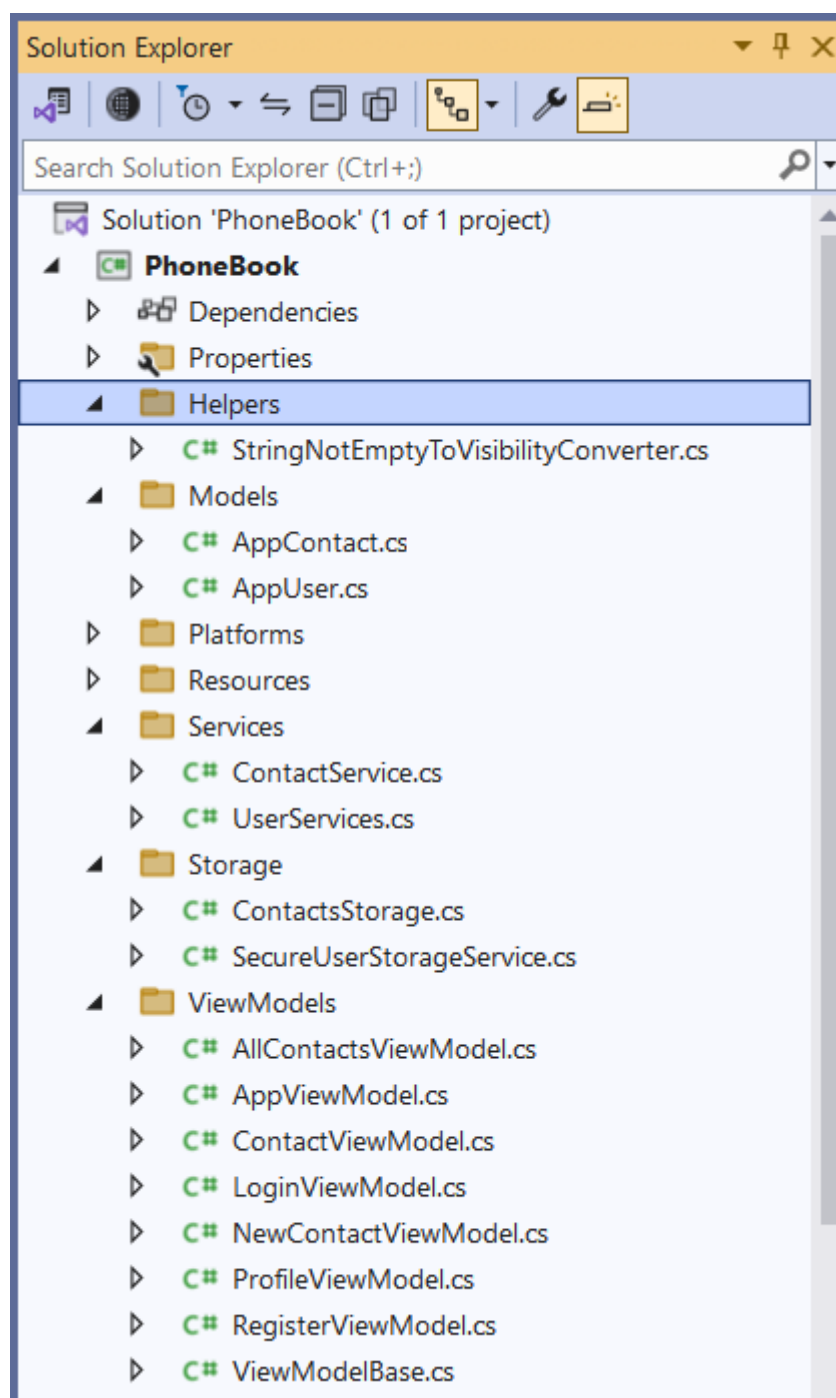
builder.RegisterViews().RegisterServices().RegisterViewModels();
return builder.Build();
```

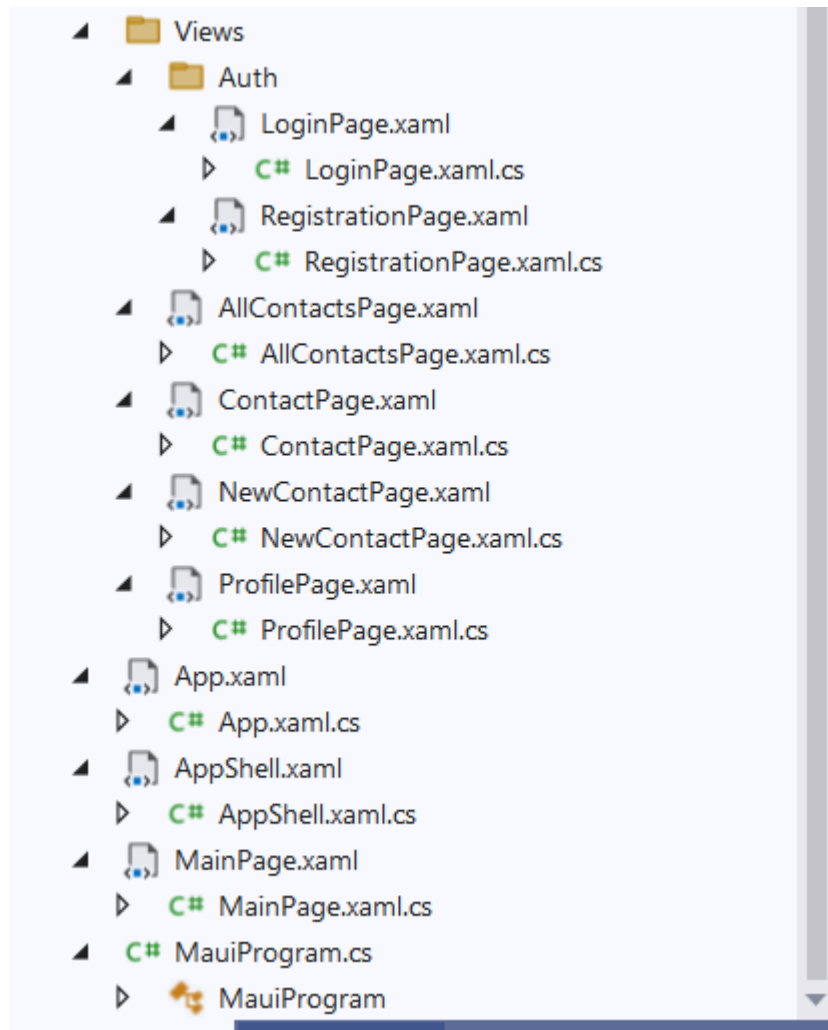
יתרונות DI:

- קוד גמיש וניתן לתחזוקה : הקוד אינו תלוי במימושים ספציפיים אלא בממשקים, מה שמאפשר החלפת מימושים בקלות.
- קלות בבדיקות יחידה : מאפשר להזרים אובייקטים חלופיים בעת בדיקות (למשל מימושים מדומים או מזויפים).
- צמצום צימוד : המחלקות אינן תלויות במימושים קונקרטיים אלא בממשקים, מה שמפחית צימוד ומקל על הרחבת הקוד.

DI היא טכניקה חשובה בארכיטקטורות מודרניות, המאפשרת ניהול חכם של התלויות במערכת.

הרכית הפרוייקט:





הסבר כללי על מבנה הפרויקט:

תיקיות עיקריות:

- **Models** תיקיה זו מכילה את המודלים של היישום, כמו `AppContact` ו-`AppUser`. מודלים אלו מייצגים את הנתונים הבסיסיים שבאפליקציה, כמו משתמשים ופרטי הקשר.
- **ViewModels** תיקיה זו כוללת את ה-`ViewModels`, כמו `LoginViewModel`, `ProfileViewModel`, `ContactViewModel` ו-`AppContact`. כל `ViewModel` אחראי על הלוגיקה העסקית ומייצג את הנתונים שמשויכים למסכים שונים באפליקציה. ה-`ViewModels` מחברים בין ה-`Views` ל-`Models` בהתאם ל-MVVM.
- **Views** כאן נמצאים כל הקבצים הקשורים לממשק המשתמש (XAML). כל קובץ XAML מייצג מסך מסוים באפליקציה, לדוגמה `LoginPage.xaml`, `ProfilePage.xaml`, `ContactPage.xaml` ומאחורי

כל קובץ XAML יש קובץ C# שמטפל בקוד התנהגותי של המסך-code-behind).

- **Services:** כאן נמצאים השירותים באפליקציה, כמו `ContactService` ו-`UserService`. שירותים אלו אחראים על התקשורת עם מסדי הנתונים, קבצים, או API חיצוניים, והם מוזרקים באמצעות DI.
- **Storage:** תיקיה זו כוללת שירותים המטפלים באחסון נתונים, כמו `ContactsStorage` ו-`SecureUserStorageService`.
- **Helpers:** תיקיה זו מכילה עוזרים (helpers) כמו `StringNotEmptyToVisibilityConverter`, או סיוע בלוגיקה מסוימת בממשק המשתמש.

2. **קבצי XAML + Code-behind:** קבצי XAML מגדירים את הממשק הגרפי של האפליקציה (כפתורים, טקסט, שדות קלט), בעוד שהקבצים שמאחוריהם (המסומנים ב-`.xaml.cs`-מכילים את הלוגיקה שמבוצעת במסך זה, כמו לחיצה על כפתור.

MVVM:

- **Model:** מייצג את הנתונים (כמו פרטי משתמשים או אנשי קשר).
- **View:** הממשק הגרפי של האפליקציה, מה שמשתמשים רואים.
- **ViewModel:** מחבר בין ה-Model ל-View. ה-`ViewModel`-מטפל בכל הלוגיקה של האפליקציה ומעדכן את ה-View בהתאם לנתונים במודל.

DI (Dependency Injection):

המשמעות של DI היא שהיישומים אינם תלויים במימוש ישיר של אובייקטים, אלא מקבלים אותם מבחוץ. זה מאפשר גמישות, בדיקות קוד טובות יותר, והפרדה ברורה יותר של אחריות.

בתמונה שהעלית, רואים בבירור שהיישום בנוי בצורה שמאפשרת הרחבה ושיפור, תוך שימוש ב-MVVM ו-DI-שמפרידים בין החלקים השונים של האפליקציה בצורה מודולרית ומסודרת

```

2  namespace PhoneBook.Models
3  {
4      23 references
5      public class AppUser
6      {
7          0 references
8          public int Id { get; set; }
9          10 references
10         public string Username { get; set; } = string.Empty;
11         4 references
12         public string Email { get; set; } = string.Empty;
13         4 references
14         public string FirstName { get; set; } = string.Empty;
15         4 references
16         public string LastName { get; set; } = string.Empty;
17         4 references
18         public string Phone { get; set; } = string.Empty;
19         2 references
20         public string Password { get; set; } = string.Empty;
21         5 references
22         public DateTime Birthdate { get; set; } = DateTime.Now;
23         4 references
24         public bool IsMale { get; set; } = true;
25
26         0 references
27         public int CalculateAge()
28         {
29             var today = DateTime.Today;
30             var age = today.Year - Birthdate.Year;
31             return age;
32         }
33     }
34 }

```

מחלקת AppUser היא אחת מהמחלקות המרכזיות במודול המודלים של אפליקציית ה-Phone Book. המחלקה מייצגת משתמש באפליקציה ומשמשת לצורך רישום וכניסה של משתמשים. המחלקה מכילה מידע אישי על המשתמשים, כמו שם, כתובת דוא"ל, סיסמה, טלפון ותאריך לידה. בנוסף, המחלקה כוללת פונקציונליות לחישוב גיל המשתמש בהתבסס על תאריך הלידה שלו.

תכונות המחלקה (Properties):

- **Id:** מזהה ייחודי של המשתמש במערכת, משמש למעקב פנימי.
- **Username:** שם המשתמש שנבחר על ידי המשתמש.
- **Email:** כתובת הדוא"ל של המשתמש.
- **FirstName:** שם פרטי של המשתמש.

- **LastName:** שם משפחה של המשתמש.
- **Phone:** מספר טלפון של המשתמש.
- **Password:** סיסמה להגנה על חשבון המשתמש.
- **Birthdate:** תאריך הלידה של המשתמש, אשר גם משמש לחישוב הגיל.
- **IsMale:** מציינ האם המשתמש הוא גבר, כאשר ערך ברירת המחדל הוא true.

AppContact

```

8      {
9      23 references
10     public class AppContact
11     {
12         7 references
13         public string Id { get; set; } = Guid.NewGuid().ToString();
14         4 references
15         public string Email { get; set; } = string.Empty;
16         5 references
17         public string FirstName { get; set; } = string.Empty;
18         5 references
19         public string LastName { get; set; } = string.Empty;
20         4 references
21         public string Phone { get; set; } = string.Empty;
22         4 references
23         public DateTime Birthdate { get; set; } = DateTime.Now;
24         11 references
25         public string ProfileImagePath { get; set; }
26         0 references
27         public string DisplayedProfileImage
28         {
29             get
30             {
31                 if(string.IsNullOrEmpty(ProfileImagePath))
32                 {
33                     if (IsMale) return "male.svg";
34                     else return "female.svg";
35                 }
36                 return ProfileImagePath;
37             }
38         }
39         6 references
40         public bool IsMale { get; set; } = true;
41     }
42 }

```

מחלקת AppContact באפליקציית ה Phone Book-מייצגת איש קשר בספר הטלפונים האישי של המשתמש. כל איש קשר מכיל את המידע הבסיסי כגון שם פרטי, שם משפחה, דוא"ל, טלפון, תאריך לידה ועוד. בנוסף, המחלקה כוללת מאפיינים לניהול תמונת פרופיל של איש הקשר, עם טיפול אוטומטי בתמונת ברירת מחדל במידה ולא קיימת תמונה אישית.

תכונות המחלקה: (Properties)

- **Id:** מזהה ייחודי של איש הקשר, מבוסס על Guid שנוצר באופן אוטומטי.
- **Email:** כתובת הדוא"ל של איש הקשר.
- **FirstName:** שם פרטי של איש הקשר.
- **LastName:** שם משפחה של איש הקשר.
- **Phone:** מספר טלפון של איש הקשר.
- **Birthdate:** תאריך הלידה של איש הקשר, המוגדר כברירת מחדל לתאריך הנוכחי.
- **ProfileImagePath:** הנתיב לתמונת הפרופיל של איש הקשר, אם קיימת.
- **IsMale:** מציינ האם איש הקשר הוא גבר (true) או אישה (false).

מאפיין מחושב: (Computed Property)

- **DisplayedProfileImage:** מאפיין המחזיר את הנתיב לתמונת הפרופיל של איש הקשר. אם לא קיימת תמונה אישית) כלומר, הנתיב ריק או (null יוחזר קובץ ברירת מחדל. עבור גברים יוחזר, "male.svg" ועבור נשים יוחזר. "female.svg". במקרה שקיים נתיב לתמונה אישית, הנתיב יוחזר.

מסכים של האפליקציה

הרשמה Register

Android Emulator - pixel_5_-_api_34:5554

1:14

Register Logout

FirstName

First Name is required

LastName

Last Name is required

Username

Username is either short or includes invalid chars

Email

Email is not valid

Password

Password should be at least 8 chars with digit, lower char, uppercase char and special char

Phone

Phone is required

Gender

Birthdate

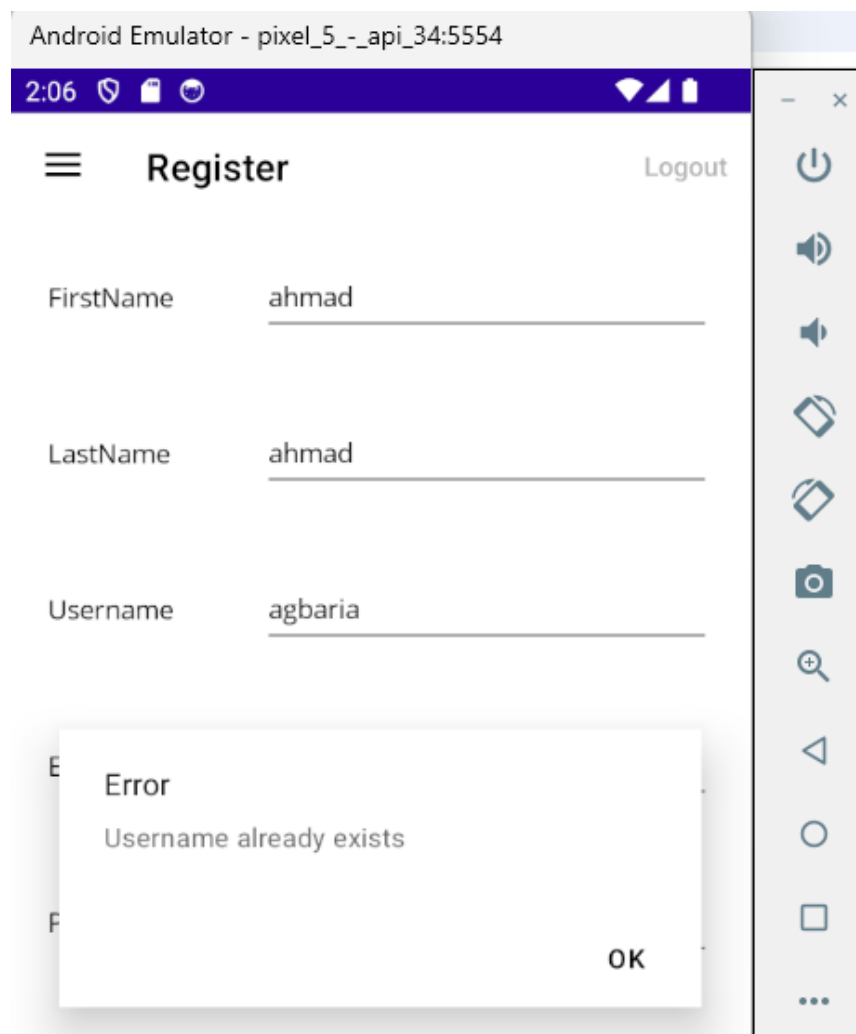
Calculated Age: 18

Register Login

מסך ההרשמה של אפליקציית **Phone Book** כאשר המשתמש מנסה להירשם בפעם הראשונה. זהו טופס המבקש מהמשתמש להזין מידע כמו שם פרטי, שם משפחה, שם משתמש, אימייל, סיסמה, טלפון, מגדר ותאריך לידה.

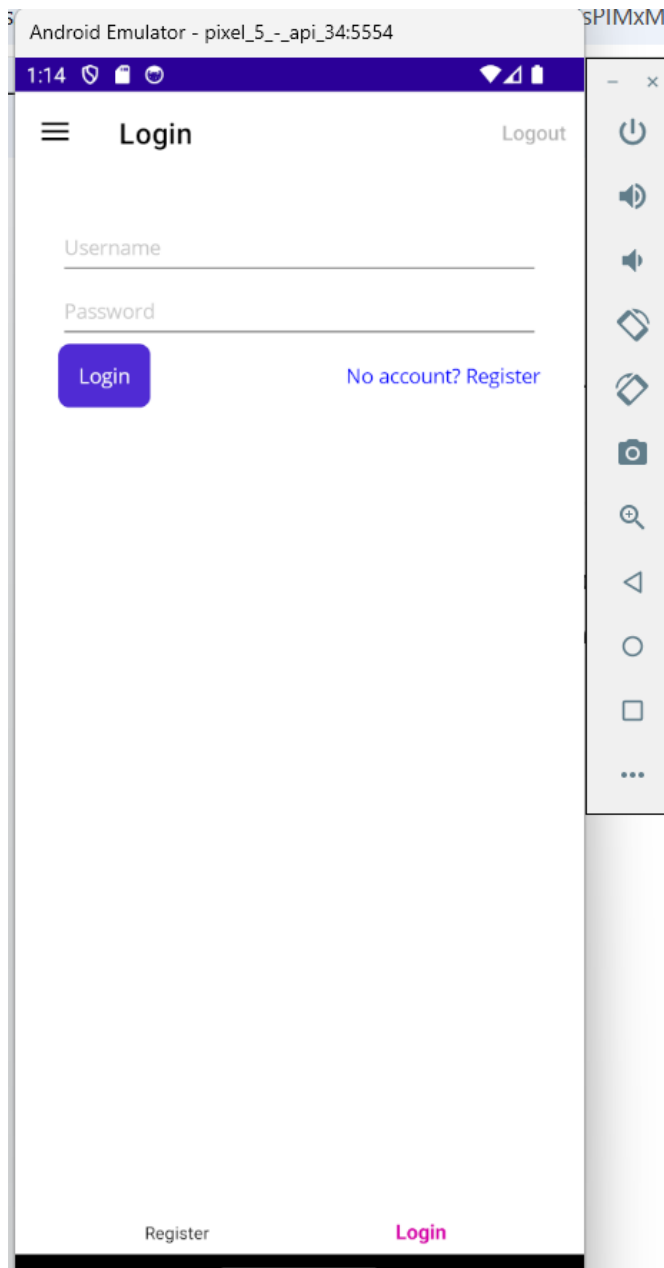
- כל שדה בטופס מלווה בהודעת שגיאה בצבע אדום במידה והנתונים אינם תקינים או חסרים.
- השדות החסרים מופיעים עם הודעת שגיאה כמו "First Name is required" או "Password should be at least 8 chars with digit, lower char, uppercase char and special char", במידה ובמילוי נכון של כל השדות.
- בתחתית הטופס יש כפתורי רישום ("Register") וכניסה ("Login"), המאפשרים למשתמש לבחור את הפעולה המתאימה.

אם יירשם פעם שנייה – עם אותו שם משתמש, תקפוץ לו הודעת השגיאה הבאה :



רישום כניסה Login

ברגע שהמשתמש מלא את הפרטים של ההרשמה, יועבר למסך הכניסה הבא:




עדכון פרטים

אחרי שהשתמש נכנס למערכת, יוכל לצפות ולעדכן את פרטיו במסך : Profile

Android Emulator - pixel_5_-_api_34:5554

2:19

 **Edit Profile** Logout

First Name

Last Name












Username

Email

Phone

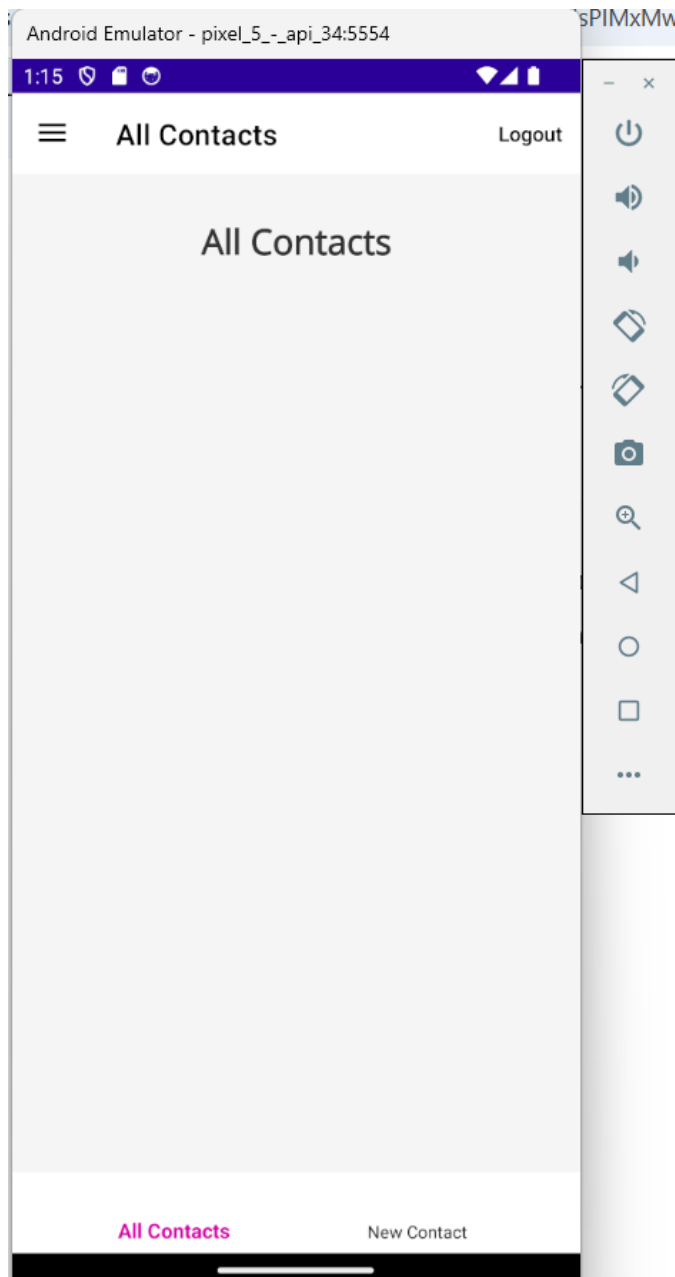
Gender

Birthdate

רשימת אנשי הקשר:

אחרי כניסה למערכת, יועבר המשתמש למסך: רשימת אנשי הקשר, בהתחלה תהיה ריקה.




תהיה לו אופציה ללחוץ על: New Contact ואז יועבר למסך "יצירת איש קשר"

Android Emulator - pixel_5_-_api_34:5554

1:15

☰ **New Contact** Logout

Select



First Name

Last Name

Email

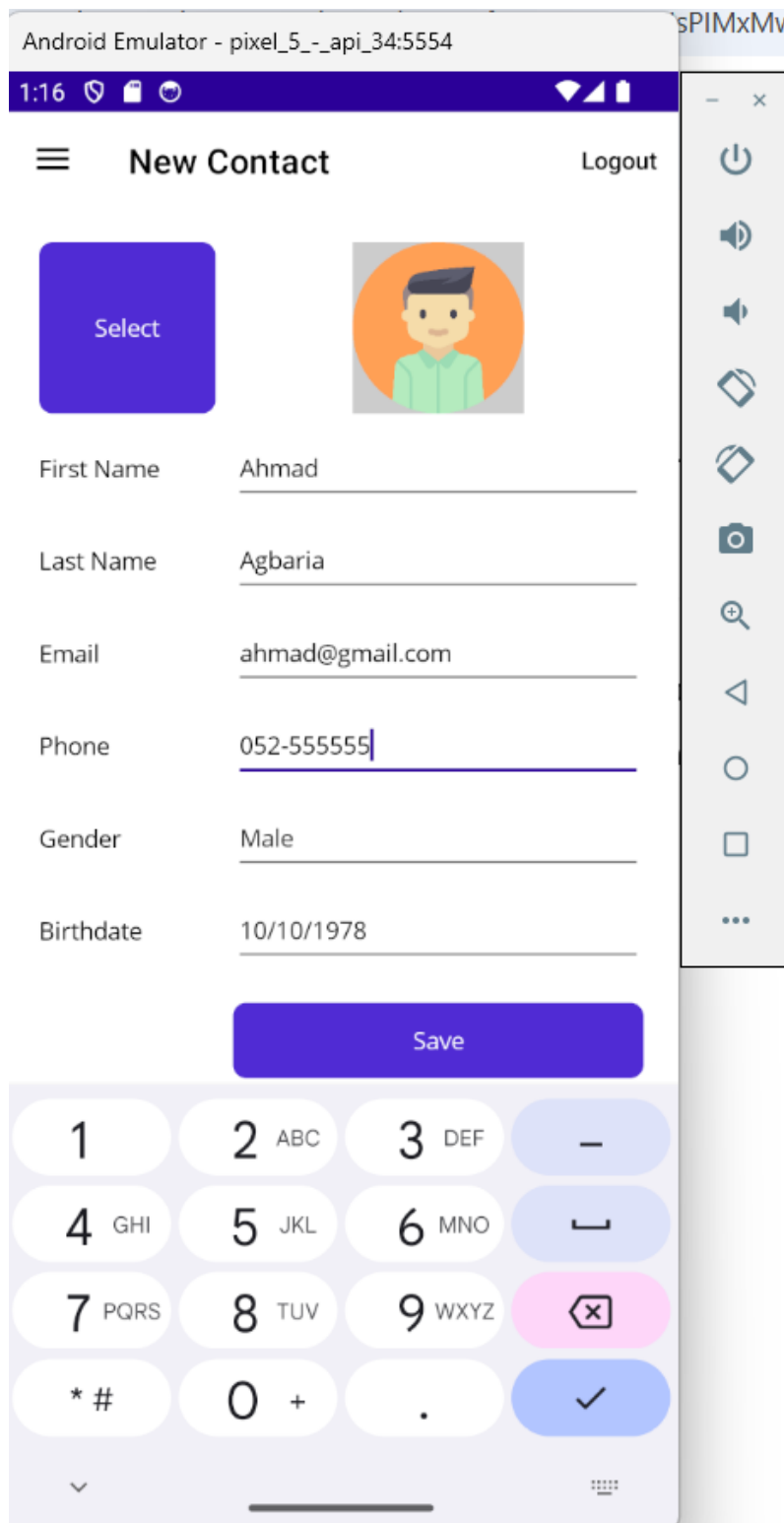
Phone

Gender

Birthdate

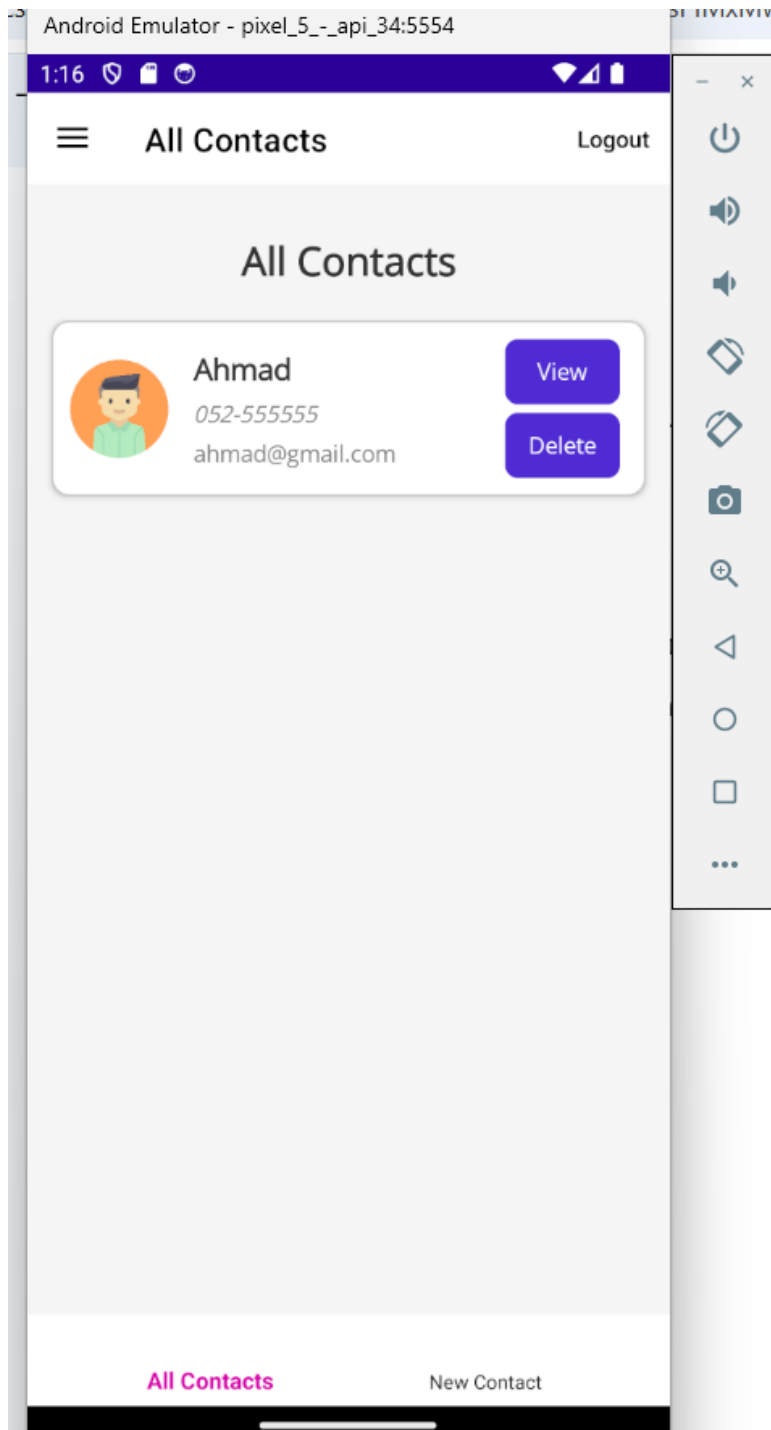
Save

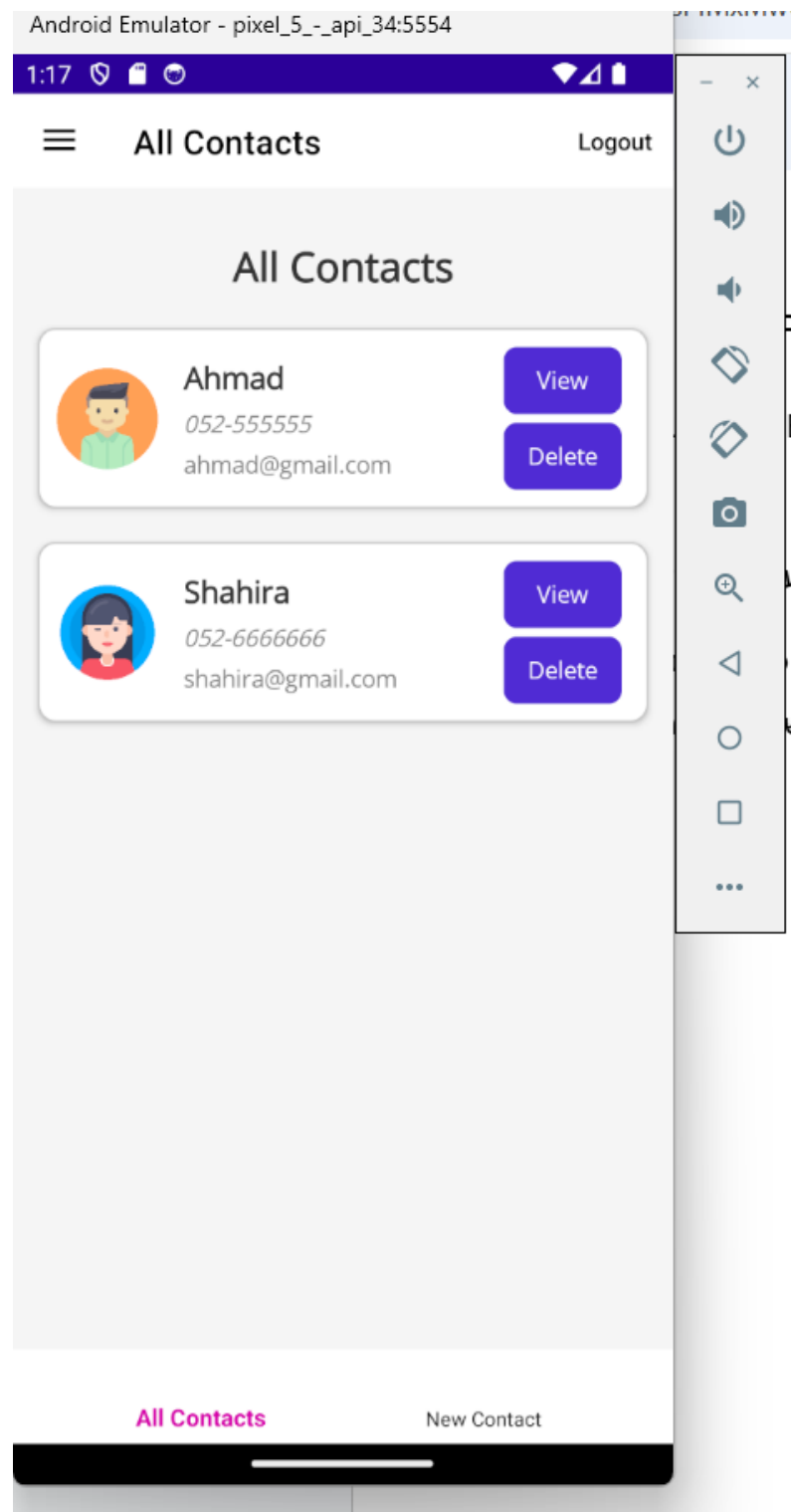
All Contacts **New Contact**



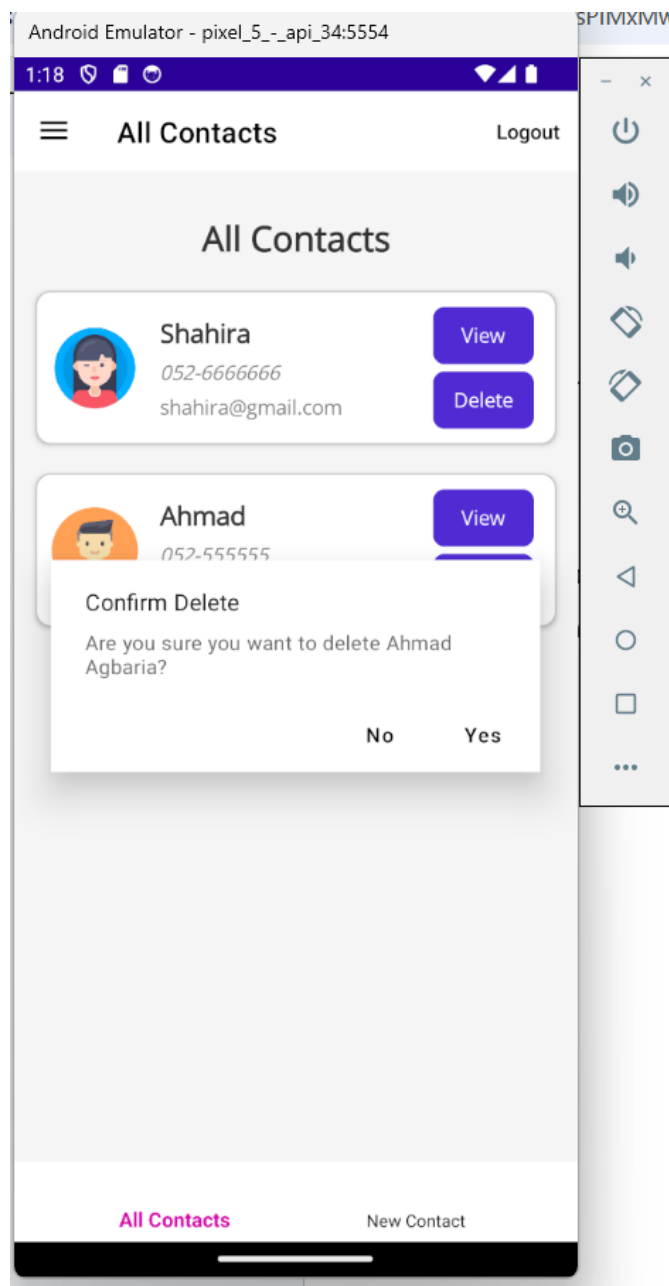
ברגע שמסיים את מילוי הפרטים ושמירת איש הקשר, איש הקשר יישמר ב-storage של המכשיר והמשתמש יועבר למסך "אנשי הקשר" המעודכנים

אנשי הקשר המעודכנים



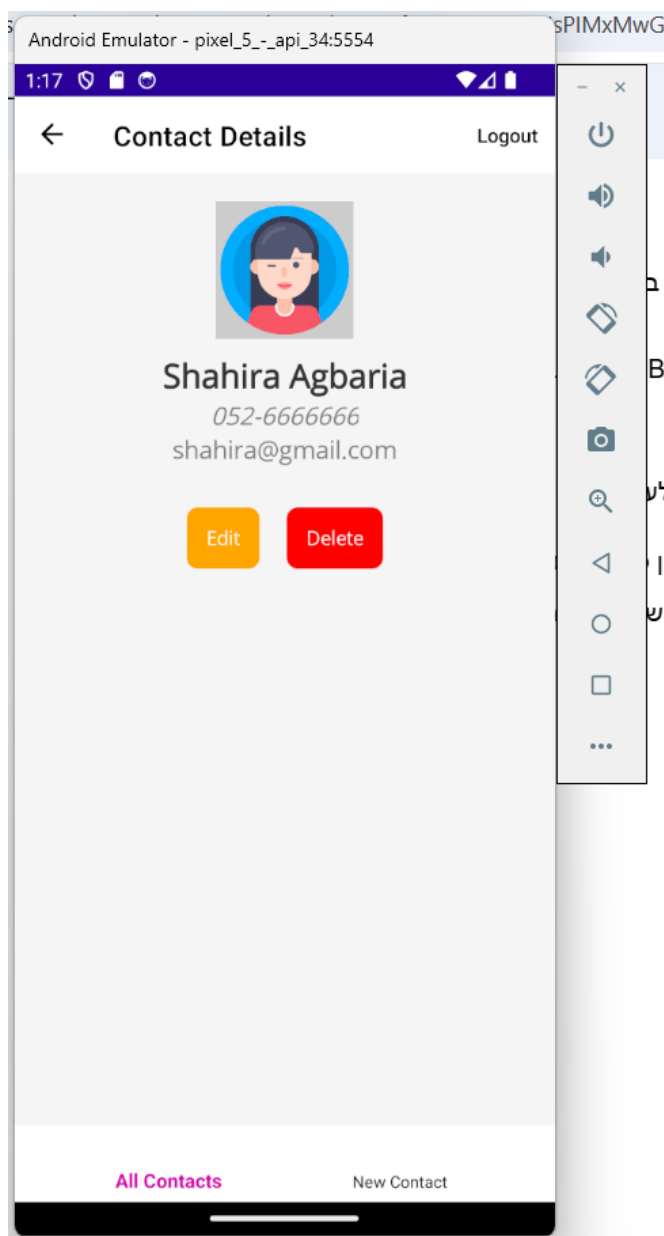


למשתמש את האופציה למחוק איש קשר – לחיצה על Delete ואז יופיע לו חלון Popup לאישור



אחרי זה, דף אנשי הקשר יעודכן מחדש – בלי איש הקשר שנמחק, הכל שמור ב- storage
גם את האופציה לצפות בפרטי איש קשר בדף נפרד:

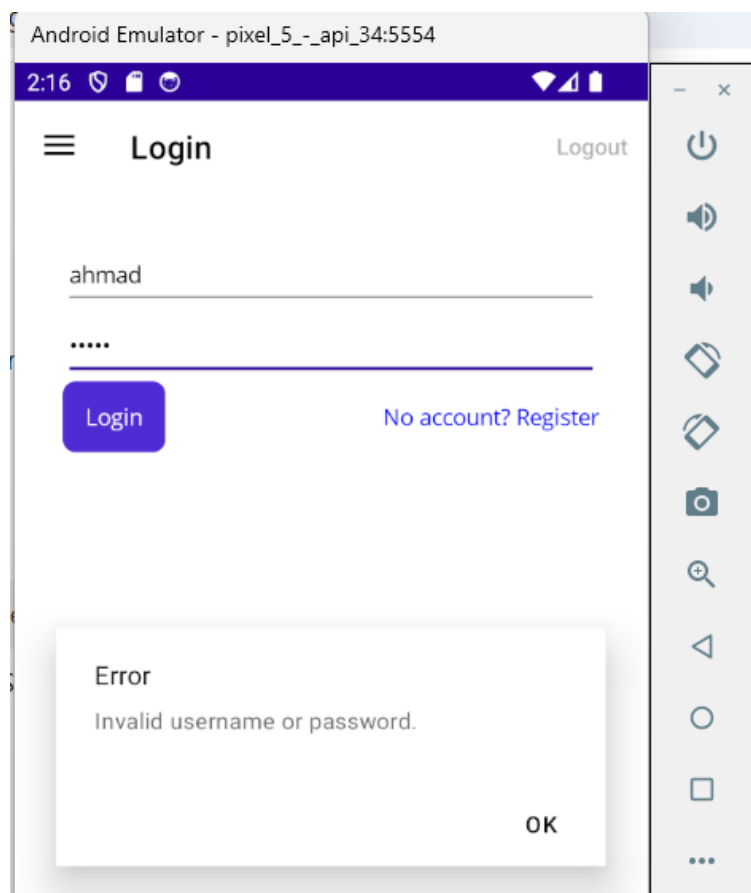
איש קשר



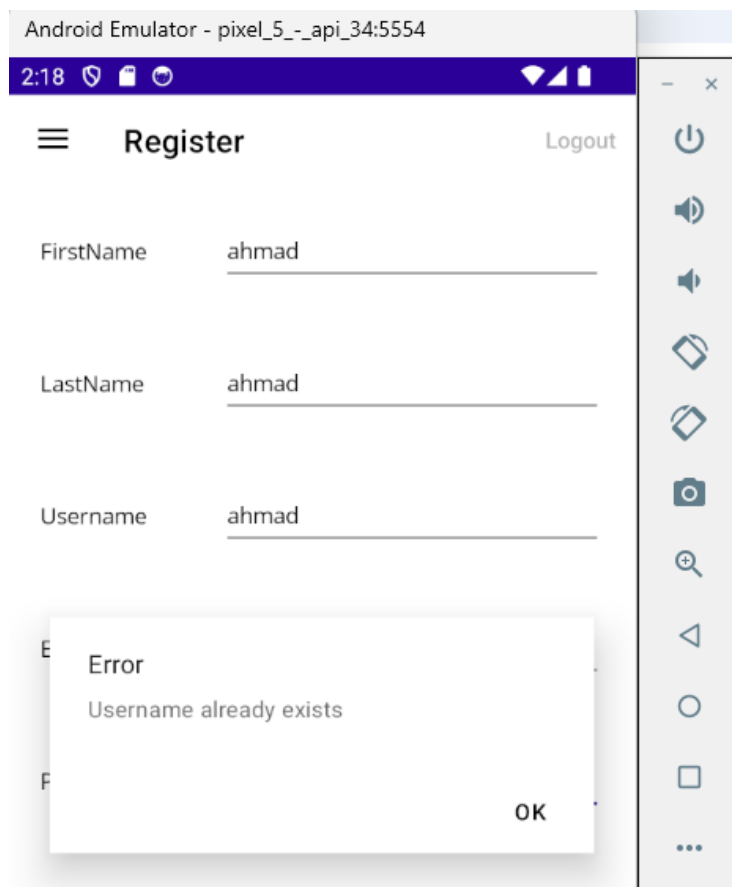
דף שמכיל את כל המידע על איש קשר ספציפי, עם אופציה לעדכן או למחוק אותו

הודעות שגיאה:

כניסה עם פרטים לא תקינים:



רישום פעמיים – אותו user



Services

ContactService

```
using PhoneBook.Models;
using PhoneBook.Storage;

namespace PhoneBook.Services
{
    public class ContactService
    {
        // Simulate fetching contacts from the device (For iOS/Android,
        // you'd use platform-specific APIs)
        public async Task<List<AppContact>> GetAllContactsAsync()
        {
            var contacts = new List<AppContact>();

            // Ensure we have permissions to read contacts
            var hasPermission = await RequestContactsPermissionAsync();
            if (!hasPermission)
            {
                return contacts; // Return empty list if no permission
            }

            contacts = await
ContactsStorageService.Instance.GetContactsAsync();

            return contacts;
        }

        public async Task DeleteContactAsync(string id)
        {
            await ContactsStorageService.Instance.DeleteContactAsync(id);
        }

        public async Task NewContactAsync(AppContact contact)
        {
            await
ContactsStorageService.Instance.AddNewContactAsync(contact);

            // Request permissions for reading contacts
            private async Task<bool> RequestContactsPermissionAsync()
            {
                var status = await
Permissions.CheckStatusAsync<Permissions.ContactsRead>();
                if (status != PermissionStatus.Granted)
                {
                    status = await
Permissions.RequestAsync<Permissions.ContactsRead>();
                }

                return status == PermissionStatus.Granted;
            }
        }
    }
}
```

UserServices

```
using PhoneBook.Models;
using PhoneBook.Storage;

namespace PhoneBook.Services
{
    public class UserServices
    {
        private AppUser? _loggedInUser;
        public UserServices()
        {
        }
        // Simulate getting the logged-in user
        public async Task<AppUser?> GetLoggedInUser()
        {
            if (_loggedInUser == null)
            {
                if (IsUserLoggedIn())
                {
                    var username = Preferences.Default.Get("Username", "");
                    if (!string.IsNullOrEmpty(username))
                    {
                        AppUser? u = await
SecureUserStorageService.Instance.GetUserAsync(username);
                        _loggedInUser = u;
                    }
                }
            }
            return _loggedInUser;
        }
        public async Task<bool> Register(AppUser user)
        {
            if (user == null) return false;
            //Search for user with same user name in AllUsers
            AppUser? u = await
SecureUserStorageService.Instance.GetUserAsync(user.Username);
            if (u != null)
            {
                return false;
            }
            await SecureUserStorageService.Instance.AddNewUserAsync(user);
            return true;
        }
        public async Task<bool> UpdateUserProfile(AppUser user)
        {
            AppUser? u = await
SecureUserStorageService.Instance.GetUserAsync(user.Username);
            if (u == null) return false;

            return await
SecureUserStorageService.Instance.UpdateUserAsync(user);
        }
        public async Task<AppUser?> Login(string username, string
password)
        {
            if (username == null || password == null)
                return null;
            AppUser? u = await
SecureUserStorageService.Instance.GetUserAsync(username);
            if (u == null) return null;
            if (u.Password != password) return null;
        }
    }
}
```

```

        _loggedInUser = u;
        Preferences.Default.Set("IsLoggedIn", true);
        Preferences.Default.Set("Username", u.Username);
        return u;
    }
    public bool IsUserLoggedIn()
    {
        bool f1 = Preferences.Default.Get("IsLoggedIn", false);
        bool f2 =
!string.IsNullOrEmpty(Preferences.Default.Get("Username", ""));
        return f1 && f2;
    }

    public void Logout()
    {
        _loggedInUser = null;
        Preferences.Default.Clear();
    }
}

```

Storage

ContactsStorage

```
using System.Text.Json;
using Microsoft.Maui.Storage;
using System.Threading.Tasks;
using PhoneBook.Models;

namespace PhoneBook.Storage
{
    public class ContactsStorageService
    {
        // Singleton instance
        private static ContactsStorageService _instance;
        public static ContactsStorageService Instance => _instance ??= new
ContactsStorageService();

        // Private constructor to prevent external instantiation
        private ContactsStorageService() { }

        private const string ContactKey = "AppContacts";
        //get all contacts

        public async Task AddNewContactAsync(AppContact contact)
        {
            var contacts = await GetContactsAsync();
            contacts.Add(contact);
            await SaveAsync(contacts);
        }

        public async Task UpdateContactAsync(AppContact contact)
        {
            var contacts = await GetContactsAsync();
            var index = contacts.FindIndex(c => c.Id == contact.Id);
            if (index >= 0)
            {
                contacts[index] = contact; // Update existing contact
                await SaveAsync(contacts);
            }
        }

        public async Task DeleteContactAsync(string id)
        {
            var contacts = await GetContactsAsync();
            contacts.RemoveAll(c => c.Id == id);
            await SaveAsync(contacts);
        }

        public async Task<AppContact> GetContactAsync(string id)
        {
            var contacts = await GetContactsAsync();
            return contacts.FirstOrDefault(c => c.Id == id);
        }

        public bool Clear()
        {
            return SecureStorage.Remove(ContactKey);
        }

        public async Task SaveAsync(List<AppContact> contacts)
        {

```

```

        var json = JsonSerializer.Serialize(contacts);
        await SecureStorage.SetAsync(ContactKey, json);
    }

    public async Task<List<AppContact>> GetContactsAsync()
    {
        try
        {
            var json = await SecureStorage.GetAsync(ContactKey);
            if (string.IsNullOrEmpty(json))
            {
                return new List<AppContact>();
            }
            else
            {
                return
JsonSerializer.Deserialize<List<AppContact>>>(json);
            }
        }
        catch (Exception ex) // Catch specific exception
        {
            Console.WriteLine($"Error in GetContactsAsync:
{ex.Message}");
            // Optionally log the stack trace for more detail
            Console.WriteLine(ex.StackTrace);
            return new List<AppContact>();
        }
    }
}

```

SecureUserStorageService

```
using System.Text.Json;
using Microsoft.Maui.Storage;
using System.Threading.Tasks;
using PhoneBook.Models;
namespace PhoneBook.Storage
{
    public class SecureUserStorageService
    {
        // Singleton instance
        private static SecureUserStorageService? _instance = null;
        public static SecureUserStorageService Instance => _instance ??=
new SecureUserStorageService();
        // Private constructor to prevent external instantiation
        private SecureUserStorageService() { }

        private const string UserKey = "AppUsers";
        private List<AppUser> _users = new List<AppUser>();

        public async Task AddNewUserAsync(AppUser user)
        {
            var users = await GetUsersAsync();
            users.Add(user);
            await SaveAsync(users);
        }

        public async Task<bool> UpdateUserAsync(AppUser user)
        {
            var users = await GetUsersAsync();
            var index = users.FindIndex(u => u.Username == user.Username);
            if (index >= 0)
            {
                users[index] = user; // Update existing user
                await SaveAsync(users);
                return true;
            }
            return false;
        }

        public async Task DeleteUserAsync(AppUser user)
        {
            var users = await GetUsersAsync();
            users.RemoveAll(u => u.Username == user.Username);
            await SaveAsync(users);
        }

        public async Task<AppUser?> GetUserAsync(string username)
        {
            var users = await GetUsersAsync();
            return users.FirstOrDefault(u => u.Username == username);
        }

        public bool Clear()
        {
            return SecureStorage.Default.Remove(UserKey);
        }

        public async Task SaveAsync(List<AppUser> users)
        {
            var json = JsonSerializer.Serialize(users);
            await SecureStorage.Default.SetAsync(UserKey, json);
        }
    }
}
```



```

    }

    private async Task<List<AppUser>> GetUsersAsync()
    {
        try
        {
            string? json = await
SecureStorage.Default.GetAsync(UserKey);
            if (string.IsNullOrEmpty(json))
            {
                return [];
            }
            return JsonSerializer.Deserialize<List<AppUser>>(json) ??
[];
        }
        catch
        {
            return [];
        }
    }
}

```

ViewModels

ViewModelBase

```
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace PhoneBook.ViewModels
{
    public class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged([CallerMemberName] string
propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }

        protected bool SetProperty<T>(ref T storage, T value,
[CallerMemberName] string propertyName = null)
        {
            if (EqualityComparer<T>.Default.Equals(storage, value))
                return false;

            storage = value;
            OnPropertyChanged(propertyName);
            return true;
        }
    }
}
```

AllContactsViewModel

```
using PhoneBook.Models;
using PhoneBook.Services;
using System.Collections.ObjectModel;
using System.Windows.Input;

namespace PhoneBook.ViewModels
{
    public class AllContactsViewModel : ViewModelBase
    {
        private readonly ContactService _contactService;
        private ObservableCollection<AppContact> _contacts;
        private bool _isBusy;

        public ObservableCollection<AppContact> Contacts
        {
            get => _contacts;
            set
            {
                _contacts = value;
                OnPropertyChanged(nameof(Contacts));
            }
        }

        public bool IsBusy
        {
            get => _isBusy;
            set
            {
                _isBusy = value;
                OnPropertyChanged(nameof(IsBusy));
            }
        }

        public ICommand LoadContactsCommand { get; }

        public AllContactsViewModel()
        {
            _contactService = new ContactService();
            Contacts = new ObservableCollection<AppContact>();

            // Command to load contacts
            LoadContactsCommand = new Command(async () => await
LoadContacts());

            // Load contacts initially
            LoadContactsCommand.Execute(null);
        }

        public async Task LoadContacts()
        {
            if (IsBusy) return;

            IsBusy = true;

            try
            {
                var contacts = await
_contactService.GetAllContactsAsync();
                Contacts.Clear();
            }
        }
    }
}
```

```

        foreach (var contact in contacts)
        {
            Contacts.Add(contact);
        }
    }
    catch (Exception ex)
    {
        // Handle exceptions (e.g., permission denied or unable to
fetch contacts)
        Console.WriteLine($"Error fetching contacts:
{ex.Message}");
    }
    finally
    {
        {
            IsBusy = false;
        }
    }
}
}

```

AppViewModel

```
using PhoneBook.Services;
using System.Diagnostics;
using System.Windows.Input;

namespace PhoneBook.ViewModels
{
    public class AppViewModel : ViewModelBase
    {
        public ICommand LogoutCommand { get; private set; }
        public ICommand NavigateToProfileCommand { get; private set; }
        public ICommand NavigateToAllContactsCommand { get; private set; }

        public bool IsLoggedIn
        {
            get => userServices.IsUserLoggedIn();
            set
            {
                OnPropertyChanged(nameof(IsLoggedIn));
                OnPropertyChanged(nameof(IsLoggedOut)); // To manage the
visibility of buttons
            }
        }

        public bool IsLoggedOut => !IsLoggedIn;
        public void NotifyLoginStatusChanged()
        {
            OnPropertyChanged(nameof(IsLoggedIn));
            OnPropertyChanged(nameof(IsLoggedOut));
        }
        private UserServices userServices;
        public AppViewModel()
        {
            userServices =
IPlatformApplication.Current.Services.GetService<UserServices>();
            NotifyLoginStatusChanged();

            LogoutCommand = new Command(async () => await Logout());
            NavigateToProfileCommand = new Command(async () => await
Shell.Current.GoToAsync("//ProfilePage"));
            NavigateToAllContactsCommand = new Command(async () => await
Shell.Current.GoToAsync("//AllContactsPage"));
        }

        private async Task Logout()
        {
            try
            {
                userServices.Logout();

                // Optionally, log the current shell section for debugging
                Debug.WriteLine($"Current Shell Section:
{Shell.Current.CurrentState}");

                // Navigate to the login page
                await Shell.Current.GoToAsync("//LoginPage");
                NotifyLoginStatusChanged();
            }
            catch (Exception ex)
            {
                // Log the exception
                Debug.WriteLine($"Error during logout: {ex.Message}");
            }
        }
    }
}
```

```
        await Application.Current.MainPage.DisplayAlert("Error",  
"Logout failed. Please try again.", "OK");  
    }  
}  
}
```

ContactViewModel

```
using PhoneBook.Models;
using PhoneBook.Services;
using System.Windows.Input;

namespace PhoneBook.ViewModels
{
    public class ContactViewModel : ViewModelBase
    {
        private AppContact _contact;

        public string FullName => $"{_contact.FirstName}
{_contact.LastName}";
        public string Phone => _contact.Phone;
        public string Email => _contact.Email;
        public DateTime Birthdate => _contact.Birthdate;
        public string ProfileImagePath => _contact.ProfileImagePath;

        public ICommand EditCommand { get; private set; }
        public ICommand DeleteCommand { get; private set; }

        private ContactService _contactService;

        public string DisplayedProfileImage
        {
            get
            {
                // If the user has uploaded a profile image, use it
                if (!string.IsNullOrEmpty(_contact.ProfileImagePath))
                {
                    return _contact.ProfileImagePath;
                }

                // If no image is uploaded, return the default avatar
                based on gender
                return _contact.IsMale
                    ? "male.svg" // Use male avatar from
Resources/Images/male.png
                    : "female.svg"; // Use female avatar from
Resources/Images/female.png
            }
        }

        public ContactViewModel() {
            _contactService =
IPlatformApplication.Current.Services.GetService<ContactService>();
        }

        public ContactViewModel(AppContact contact): this()
        {
            _contact = contact;
            RefreshMe();
            EditCommand = new Command(EditContact);
            DeleteCommand = new Command(DeleteContact);
        }

        void RefreshMe()
        {
            OnPropertyChanged(nameof(FullName));
            OnPropertyChanged(nameof(Email));
            OnPropertyChanged(nameof(Phone));
            OnPropertyChanged(nameof(Birthdate));
            OnPropertyChanged(nameof(DisplayedProfileImage));
        }

        private async void EditContact()
```

```

    {
        //Display not implemented yet
        await Application.Current.MainPage.DisplayAlert("Coming
Soon.", "Function is not implemente yet!", "OK");
    }

    private async void DeleteContact()
    {
        // Delete the contact
        await _contactService.DeleteContactAsync(_contact.Id);
        //Display success message
        await Application.Current.MainPage.DisplayAlert("Success",
"Contact deleted successfully", "OK");

        // Navigate back to AllContactsPage after deletion
        await Shell.Current.GoToAsync("..");
    }
}

```


PhoneBook

```
using PhoneBook.Models;
using PhoneBook.Services;
using System.Windows.Input;

namespace PhoneBook.ViewModels
{
    public class LoginViewModel : ViewModelBase
    {
        public ICommand LoginCommand { get; }
        UserServices userServices;
        public LoginViewModel()
        {
            userServices =
IPlatformApplication.Current.Services.GetService<UserServices>();
            LoginCommand = new Command(OnLogin);
        }

        private string username;
        public string Username
        {
            get => username;
            set
            {
                username = value;
                OnPropertyChanged(nameof(Username));
            }
        }

        private string password;
        public string Password
        {
            get => password;
            set
            {
                password = value;
                OnPropertyChanged(nameof(Password));
            }
        }
        private async void OnLogin()
        {
            AppUser? user = await userServices.Login(Username, Password);
            if (user!=null)
            {
                // Ensure navigation on the main thread
                await MainThread.InvokeOnMainThreadAsync(async () =>
                {
                    await Shell.Current.GoToAsync("//AllContactsPage");
                    (Shell.Current as AppShell)?.UpdateTabVisibility();
                });
            }
            else
            {
                await Application.Current.MainPage.DisplayAlert("Error",
"Invalid username or password.", "OK");
            }
        }
    }
}
```

NewContactViewModel

```
using PhoneBook.Models;
using PhoneBook.Services;
using System.ComponentModel;
using System.Windows.Input;

namespace PhoneBook.ViewModels
{
    public class NewContactViewModel : ViewModelBase
    {
        private AppContact _contact;

        public string FirstName
        {
            get => _contact.FirstName;
            set
            {
                if (_contact.FirstName != value)
                {
                    _contact.FirstName = value;
                    OnPropertyChanged(nameof(FirstName));
                    OnPropertyChanged(nameof(CanSave)); // Update CanSave
                    when the input changes
                }
            }
        }

        public string LastName
        {
            get => _contact.LastName;
            set
            {
                if (_contact.LastName != value)
                {
                    _contact.LastName = value;
                    OnPropertyChanged(nameof(LastName));
                    OnPropertyChanged(nameof(CanSave));
                }
            }
        }

        public string Email
        {
            get => _contact.Email;
            set
            {
                if (_contact.Email != value)
                {
                    _contact.Email = value;
                    OnPropertyChanged(nameof(Email));
                    OnPropertyChanged(nameof(CanSave));
                }
            }
        }

        public string Phone
        {
            get => _contact.Phone;
            set
            {
                if (_contact.Phone != value)
                {
```

```

        _contact.Phone = value;
        OnPropertyChanged(nameof(Phone));
        OnPropertyChanged(nameof(CanSave));
    }
}

public DateTime Birthdate
{
    get => _contact.Birthdate;
    set
    {
        if (_contact.Birthdate != value)
        {
            _contact.Birthdate = value;
            OnPropertyChanged(nameof(Birthdate));
        }
    }
}

public string ProfileImagePath
{
    get => _contact.ProfileImagePath;
    set
    {
        if (_contact.ProfileImagePath != value)
        {
            _contact.ProfileImagePath = value;
            OnPropertyChanged(nameof(ProfileImagePath));
        }
    }
}

public bool IsMale
{
    get { return _contact.IsMale; }
    set
    {
        if (_contact.IsMale != value)
        {
            _contact.IsMale = value;
            OnPropertyChanged(nameof(IsMale));
        }
    }
}

// Dropdown property for Gender
private string _selectedGender;
public string SelectedGender
{
    get => _selectedGender;
    set
    {
        _selectedGender = value;
        if (_selectedGender == "Male")
        {
            IsMale = true;
        }
        else
        {
            IsMale = false;
        }
        OnPropertyChanged();
        OnPropertyChanged(nameof(DisplayedProfileImage));
    }
}

```

```

    }
}

// Command to save the contact
public ICommand SaveContactCommand { get; }
public ICommand SelectImageCommand { get; }

// Determines if the form can be saved
public bool CanSave => !string.IsNullOrEmpty(FirstName) &&
    !string.IsNullOrEmpty(LastName) &&
    !string.IsNullOrEmpty(Email) &&
    !string.IsNullOrEmpty(Phone);

private readonly ContactService contactService;
public NewContactViewModel(ContactService contactService)
{
    this.contactService = contactService;
    // Initialize the contact object
    _contact = new AppContact();

    // Initialize the SaveContactCommand
    SaveContactCommand = new Command(async () => await
SaveContact(), () => CanSave);
    SelectImageCommand = new Command(async () => await
SelectImageAsync());
}
public string DisplayedProfileImage
{
    get
    {
        // If the user has uploaded a profile image, use it
        if (!string.IsNullOrEmpty(_contact.ProfileImagePath))
        {
            return _contact.ProfileImagePath;
        }

        // If no image is uploaded, return the default avatar
based on gender
        return _contact.IsMale
            ? "male.svg" // Use male avatar from
Resources\Images\male.png
            : "female.svg"; // Use female avatar from
Resources\Images\female.png
    }
}
private async Task SelectImageAsync()
{
    // MediaPlayer logic for selecting and saving profile image
    var result = await MediaPlayer.PickPhotoAsync();
    if (result != null)
    {
        var filePath = Path.Combine(FileSystem.AppDataDirectory,
result.FileName);
        using (var stream = await result.OpenReadAsync())
        using (var fileStream = File.Create(filePath))
        {
            await stream.CopyToAsync(fileStream);
        }
        _contact.ProfileImagePath = filePath;
        OnPropertyChanged(nameof(DisplayedProfileImage)); //
Update the image
    }
}

```

```

// Method to save the contact
private async Task SaveContact()
{
    _contact.Id = Guid.NewGuid().ToString();
    // Save the contact using the ContactService
    await this.contactService.NewContactAsync(_contact);

    // Navigate back to the contact list
    await Shell.Current.GoToAsync("//AllContactsPage");
}
}
}

```

ProfileViewModel

```
using PhoneBook.Models;
using PhoneBook.Services;
using System.Windows.Input;

namespace PhoneBook.ViewModels
{
    public class ProfileViewModel : ViewModelBase
    {
        private AppUser _user=new AppUser();

        public string FirstName
        {
            get => _user.FirstName;
            set
            {
                if (_user.FirstName != value)
                {
                    _user.FirstName = value;
                    OnPropertyChanged(nameof(FirstName));
                }
            }
        }

        public string LastName
        {
            get => _user.LastName;
            set
            {
                if (_user.LastName != value)
                {
                    _user.LastName = value;
                    OnPropertyChanged(nameof(LastName));
                }
            }
        }

        public string Username => _user.Username; // Username is usually
read-only

        public string Email
        {
            get => _user.Email;
            set
            {
                if (_user.Email != value)
                {
                    _user.Email = value;
                    OnPropertyChanged(nameof(Email));
                }
            }
        }

        public string Phone
        {
            get => _user.Phone;
            set
            {
                if (_user.Phone != value)
                {
                    _user.Phone = value;
                    OnPropertyChanged(nameof(Phone));
                }
            }
        }
    }
}
```

```

    }
}

public DateTime Birthdate
{
    get => _user.Birthdate;
    set
    {
        if (_user.Birthdate != value)
        {
            _user.Birthdate = value;
            OnPropertyChanged(nameof(Birthdate));
        }
    }
}

public bool IsMale
{
    get => _user.IsMale;
    set
    {
        if (_user.IsMale != value)
        {
            _user.IsMale = value;
            OnPropertyChanged(nameof(IsMale));
        }
    }
}

public string SelectedGender
{
    get => IsMale ? "Male" : "Female";
    set
    {
        IsMale = value == "Male";
        OnPropertyChanged(nameof(SelectedGender));
    }
}

// Command to save the updated profile
public ICommand SaveProfileCommand { get; }

private readonly UserServices userServices;

public ProfileViewModel()
{
    userServices =
        IPlatformApplication.Current.Services.GetService<UserServices>();
    // Load the logged-in user details
    //LoadUserProfile();

    // Command to save profile updates
    SaveProfileCommand = new Command(SaveProfile);
}

public async Task LoadUserProfile()
{
    // Fetch the current logged-in user (from a service)
    _user = await userServices.GetLoggedInUser();

    // Notify UI to refresh with loaded data

```

```

        OnPropertyChanged(nameof(FirstName));
        OnPropertyChanged(nameof(LastName));
        OnPropertyChanged(nameof(Username));
        OnPropertyChanged(nameof(Email));
        OnPropertyChanged(nameof(Phone));
        OnPropertyChanged(nameof(Birthdate));
        OnPropertyChanged(nameof(SelectedGender));
    }

    private async void SaveProfile()
    {
        // Save the updated user profile
        await userServices.UpdateUserProfile(_user);
        //Show success message

        await Application.Current.MainPage.DisplayAlert("Success",
        "Your profile has been updated successfully!", "OK");
    }
}

```


RegisterViewModel

```
using PhoneBook.Models;
using PhoneBook.Services;
using PhoneBook.Views.Auth;
using System.Text.RegularExpressions;
using System.Windows.Input;

namespace PhoneBook.ViewModels
{
    public class RegisterViewModel : ViewModelBase
    {
        AppUser user;
        private string username;
        private string password;
        private string email;
        private string phone;
        private string firstName;
        private string lastName;
        private DateTime birthdate = DateTime.Now.AddYears(-18);
        private bool isMale = false;
        private string errorMessage = "";
        public string UsernameErrorLabel { get; set; } = "";
        public string PasswordErrorLabel { get; set; } = "";
        public string EmailErrorLabel { get; set; } = "";
        public string PhoneErrorLabel { get; set; } = "";
        public string FirstNameErrorLabel { get; set; } = "";
        public string LastNameErrorLabel { get; set; } = "";
        //public string BirthdateErrorLabel { get; set; } = "";
        public string ErrorMessageErrorLabel { get; set; } = "";
        public string Username
        {
            get { return username; }
            set
            {
                if (username != value)
                {
                    username = value;
                    OnPropertyChanged(nameof(Username));
                    HandleError();
                }
            }
        }
        public string Email
        {
            get { return email; }
            set
            {
                if (email != value)
                {
                    email = value;
                    OnPropertyChanged(nameof(Email));
                    HandleError();
                }
            }
        }
        public string FirstName
        {
            get { return firstName; }
            set
            {
                if (firstName != value)
                {
```

```

        firstName = value;
        OnPropertyChanged(nameof(FirstName));
        HandleError();
    }
}
}
public string LastName
{
    get { return lastName; }
    set
    {
        if (lastName != value)
        {
            lastName = value;
            OnPropertyChanged(nameof(LastName));
            HandleError();
        }
    }
}
public string Phone
{
    get { return phone; }
    set
    {
        if (phone != value)
        {
            phone = value;
            OnPropertyChanged(nameof(Phone));
            HandleError();
        }
    }
}
public string Password
{
    get { return password; }
    set
    {
        if (password != value)
        {
            password = value;
            OnPropertyChanged(nameof(Password));
            HandleError();
        }
    }
}
public DateTime Birthdate
{
    get { return birthdate; }
    set
    {
        if (birthdate != value)
        {
            birthdate = value;
            OnPropertyChanged(nameof(Birthdate));
            HandleError();
        }
    }
}

public bool IsMale
{
    get { return isMale; }
    set

```

```

        {
            if (isMale != value)
            {
                isMale = value;
                OnPropertyChanged(nameof(IsMale));
                HandleError();
            }
        }
    }
    // Dropdown property for Gender
    private string _selectedGender;
    public string SelectedGender
    {
        get => _selectedGender;
        set
        {
            _selectedGender = value;
            OnPropertyChanged();
            if (_selectedGender == "Male")
            {
                IsMale = true;
            }
            else
            {
                IsMale = false;
            }
        }
    }
}

public int CalculatedAge
{
    get
    {
        return DateTime.Now.Year - Birthdate.Year;
    }
}

//PROPERTY to be Bound to the Button Command
public ICommand RegisterCommand
{
    get; private set;
}
private readonly LoginPage loginPage;
private readonly UserServices userServices;
public RegisterViewModel()
{
    userServices =
IPlatformApplication.Current.Services.GetService<UserServices>();
    RegisterCommand = new Command(Register);
    this.loginPage =
IPlatformApplication.Current.Services.GetService<LoginPage>();
}
private async void Register()
{
    user = new AppUser() {
        Username = Username,
        Password = Password,
        Email = Email,
        Phone = Phone,
        FirstName = FirstName,
        LastName = LastName,
        Birthdate = Birthdate,
        IsMale = IsMale,

```

```

    };
    bool r = await userServices.Register(user);
    if (!r)
    {
        ErrorMessage = "Username already exists";
        OnPropertyChanged(nameof(ErrorMessage));
        OnPropertyChanged(nameof(HasError));
        await
Application.Current.MainPage.DisplayAlert("Error", ErrorMessage, "OK");
        return;
    }
    //Switch to the Login Page
    await Shell.Current.GoToAsync("//LoginPage");
    //await
Application.Current.MainPage.Navigation.PushAsync(loginPage);
}

public bool CanRegister
{
    get
    {
        return !HasError;
    }
}
public bool HasError
{
    get
    {
        return !string.IsNullOrEmpty(errorMessage);
    }
}
}
public string ErrorMessage
{
    get
    {
        return errorMessage;
    }
    set
    {
        if (errorMessage != value)
        {
            errorMessage = value;
            OnPropertyChanged(nameof(ErrorMessage));
        }
    }
}
}
private void HandleError()
{
    ErrorMessage = "";
    UsernameErrorLabel = "";
    EmailErrorLabel = "";
    PhoneErrorLabel = "";
    FirstNameErrorLabel = "";
    LastNameErrorLabel = "";
    PasswordErrorLabel = "";
    if (!IsValidName())
    {
        UsernameErrorLabel="Username is either short or includes
invalid chars";
    }
    if (!IsValidEmail())
    {

```

```

        EmailErrorLable= "Email is not valid";
    }
    if (string.IsNullOrEmpty(FirstName))
    {
        FirstNameErrorLable= "First Name is required";
    }
    if (string.IsNullOrEmpty(LastName))
    {
        LastNameErrorLable= "Last Name is required";
    }
    if (string.IsNullOrEmpty(Phone))
    {
        PhoneErrorLable= "Phone is required";
    }
    if (!IsValidPassword())
    {
        PasswordErrorLable = "Password should be at least 8 chars
with digit, lower char, uppercase char and special char";
    }

    OnPropertyChanged(nameof(UsernameErrorLabel));
    OnPropertyChanged(nameof(EmailErrorLable));
    OnPropertyChanged(nameof(FirstNameErrorLable));
    OnPropertyChanged(nameof(LastNameErrorLable));
    OnPropertyChanged(nameof(PhoneErrorLable));
    OnPropertyChanged(nameof(PasswordErrorLable));
    OnPropertyChanged(nameof(HasError));
    OnPropertyChanged(nameof(CanRegister));
}
private bool IsValidName()
{
    if (string.IsNullOrEmpty(Username))
    {
        return false;
    }
    Regex reg = new ("^[a-zA-Z][a-zA-Z0-9]{2,}$");
    return reg.IsMatch(Username);
}
private bool IsValidEmail()
{
    if (string.IsNullOrEmpty(Email))
    {
        return false;
    }
    //Check if email is valid
    Regex reg = new ("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-
Z]{2,}$");
    return reg.IsMatch(Email);
}
private bool IsValidPassword()
{
    if (string.IsNullOrEmpty>Password))
    {
        return false;
    }
    //Password should be at least 8 characters long includes at least one
uppercase letter,one lowercase letter,one number,and one special character
    string pattern = @"^(?=.*[a-z])(?=.*[A-
Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$";
    Regex regex = new (pattern);
    return regex.IsMatch>Password);
}

```

Main Classes:

App.xaml

```
using PhoneBook.Services;

namespace PhoneBook
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            MainPage = new AppShell();
        }
        // OnStart - called when the application starts
        protected override void OnStart()
        {
            base.OnStart();
            var service =
IPlatformApplication.Current?.Services.GetService<UserServices>();
            // Handle logic when the app starts
            if (service != null && service.IsUserLoggedIn())
            {
                // If the user is logged in, go to ProfilePage
                Shell.Current.GoToAsync("//AllContactsPage");
            }
            else
            {
                // If not logged in, go to the RegisterPage or LoginPage
                Shell.Current.GoToAsync("//LoginPage", true);
            }
        }
    }
}
```

App.xaml.cs

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
  x:Class="PhoneBook.AppShell"
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:authviews="clr-namespace:PhoneBook.Views.Auth"
  xmlns:local="clr-namespace:PhoneBook"
  xmlns:views="clr-namespace:PhoneBook.Views"
  xmlns:vm="clr-namespace:PhoneBook.ViewModels"
  Title="Phone Book"
  Shell.FlyoutBehavior="Flyout">

  <FlyoutItem Title="Authentication" IsVisible="{Binding IsLoggedOut}">
    <ShellContent
      x:Name="RegisterTab"
      Title="Register"
      ContentTemplate="{DataTemplate authviews:RegistrationPage}"
      Route="RegistrationPage" />

    <ShellContent
      x:Name="LoginTab"
      Title="Login"
      ContentTemplate="{DataTemplate authviews:LoginPage}"
      Route="LoginPage" />
  </FlyoutItem>

  <FlyoutItem Title="Contacts" IsVisible="{Binding IsLoggedIn}">
    <ShellContent
      x:Name="ContactsTab"
      Title="All Contacts"
      ContentTemplate="{DataTemplate views:AllContactsPage}"
      Route="AllContactsPage" />
    <ShellContent
      x:Name="NewContactTab"
      Title="New Contact"
      ContentTemplate="{DataTemplate views:NewContactPage}"
      Route="NewContactPage" />
  </FlyoutItem>

  <FlyoutItem Title="Profile" IsVisible="{Binding IsLoggedIn}">
    <ShellContent
      x:Name="ProfileTab"
      Title="Profile"
      ContentTemplate="{DataTemplate views:ProfilePage}"
      Route="ProfilePage" />
  </FlyoutItem>

  <!-- Logout Button in Toolbar -->
  <Shell.ToolbarItems>
    <ToolbarItem
      x:Name="LogoutBtn"
      Command="{Binding LogoutCommand}"
      IsEnabled="{Binding IsLoggedIn}"
      Text="Logout" />
  </Shell.ToolbarItems>
</Shell>
```

AppShell.xaml

```
using Microsoft.Extensions.DependencyInjection;
using PhoneBook.ViewModels;
using PhoneBook.Views;
using PhoneBook.Views.Auth;

namespace PhoneBook
{
    public partial class AppShell : Shell
    {
        private AppViewModel? _appViewModel;
        public AppShell()
        {
            InitializeComponent();

            _appViewModel =
                IPlatformApplication.Current?.Services.GetService<AppViewModel>();
            BindingContext = _appViewModel;

            // Register routes
            RegisterRoutes();
        }

        private void RegisterRoutes()
        {
            Routing.RegisterRoute("/{0}/".Format(nameof(RegistrationPage)),
                typeof(RegistrationPage));
            Routing.RegisterRoute("/{0}/".Format(nameof(LoginPage)),
                typeof(LoginPage));
            Routing.RegisterRoute("/{0}/".Format(nameof(AllContactsPage)),
                typeof(AllContactsPage));
            Routing.RegisterRoute("/{0}/".Format(nameof(ProfilePage)),
                typeof(ProfilePage));
            Routing.RegisterRoute("/{0}/".Format(nameof(NewContactPage)),
                typeof(NewContactPage));
            Routing.RegisterRoute(nameof(ContactPage),
                typeof(ContactPage));
        }

        public void UpdateTabVisibility()
        {
            _appViewModel?.NotifyLoginStatusChanged();
        }
    }
}
```


MauiProgram

```
using Microsoft.Extensions.Logging;
using PhoneBook.Services;
using PhoneBook.ViewModels;
using PhoneBook.Views;
using PhoneBook.Views.Auth;

namespace PhoneBook
{
    public static class MauiProgram
    {
        public static MauiApp CreateMauiApp()
        {
            var builder = MauiApp.CreateBuilder();
            builder
                .UseMauiApp<App>()
                .ConfigureFonts(fonts =>
                {
                    fonts.AddFont("OpenSans-Regular.ttf",
"OpenSansRegular");
                    fonts.AddFont("OpenSans-Semibold.ttf",
"OpenSansSemibold");
                    fonts.AddFont("MaterialSymbolsOutlined.ttf",
"MaterialSymbols");
                    fonts.AddFont("gadi-almog-regular-aaa.otf",
"AlmogRegular");
                });

            #if DEBUG
                builder.Logging.AddDebug();
            #endif

            builder.RegisterViews().RegisterServices().RegisterViewModels();
            return builder.Build();
        }

        public static MauiAppBuilder RegisterViews(this MauiAppBuilder
builder)
        {
            builder.Services.AddTransient<LoginPage>();
            builder.Services.AddTransient<RegistrationPage>();
            builder.Services.AddTransient<AllContactsPage>();
            builder.Services.AddTransient<ProfilePage>();
            builder.Services.AddTransient<NewContactPage>();
            builder.Services.AddTransient<ContactPage>();
            return builder;
        }

        public static MauiAppBuilder RegisterViewModels(this
MauiAppBuilder builder)
        {
            builder.Services.AddTransient<AllContactsViewModel>();
            builder.Services.AddTransient<AppViewModel>();
            builder.Services.AddTransient<LoginViewModel>();
            builder.Services.AddTransient<ProfileViewModel>();
            builder.Services.AddTransient<RegisterViewModel>();
            builder.Services.AddTransient<NewContactViewModel>();
            builder.Services.AddTransient<ContactViewModel>();
            return builder;
        }
    }
}
```

```
        public static MauiAppBuilder RegisterServices(this
MauiAppBuilder builder)
        {
            builder.Services.AddSingleton<ContactService>();
            builder.Services.AddSingleton<UserServices>();
            return builder;
        }
    }
}
```

Views

LoginPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    x:Class="PhoneBook.Views.Auth.LoginPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:viewmodels="clr-namespace:PhoneBook.ViewModels"
    Title="Login"
    x:DataType="viewmodels:LoginViewModel">
    <StackLayout Padding="30">
        <!-- Username Entry -->
        <Entry
            Keyboard="Default"
            Placeholder="Username"
            Text="{Binding Username}" />

        <!-- Password Entry -->
        <Entry
            IsPassword="True"
            Placeholder="Password"
            Text="{Binding Password}" />

        <!-- Login Button -->
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <!-- For Login Button -->
                <ColumnDefinition Width="*" />
                <!-- To fill remaining space -->
            </Grid.ColumnDefinitions>

            <!-- Login Button -->
            <Button
                Grid.Column="0"
                Command="{Binding LoginCommand}"
                Text="Login" />

            <!-- Register Label -->
            <Label
                Grid.Column="1"
                HorizontalOptions="End"
                Text="No account? Register"
                TextColor="Blue"
                VerticalOptions="Center">
                <Label.GestureRecognizers>
                    <TapGestureRecognizer
Tapped="TapGestureRecognizer_Tapped" />
                </Label.GestureRecognizers>
            </Label>
        </Grid>
    </StackLayout>
</ContentPage>
```

Login.xaml.cs

```
using PhoneBook.Services;
using PhoneBook.ViewModels;

namespace PhoneBook.Views.Auth;

public partial class LoginPage : ContentPage
{
    public LoginPage(LoginViewModel model)
    {
        InitializeComponent();
        BindingContext = model;
    }
    private async void TapGestureRecognizer_Tapped(object sender,
TappedEventArgs e)
    {
        await Shell.Current.GoToAsync("//RegistrationPage", true);
    }
}
```

RegistrationPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    x:Class="PhoneBook.Views.Auth.RegistrationPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:PhoneBook.Helpers"
    xmlns:viewmodels="clr-namespace:PhoneBook.ViewModels"
    Title="Register"
    x:DataType="viewmodels:RegisterViewModel">
    <ContentPage.Resources>
        <ResourceDictionary>
            <local:StringNotEmptyToVisibilityConverter
x:Key="StringNotEmptyToVisibilityConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>
    <ScrollView>
        <Grid
            Padding="20"
            ColumnSpacing="10"
            RowSpacing="10">
            <Grid.ColumnDefinitions>
                <!-- Label column takes 30% of the width -->
                <ColumnDefinition Width="0.3*" />
                <!-- Control column takes 70% of the width -->
                <ColumnDefinition Width="0.7*" />
            </Grid.ColumnDefinitions>

            <Grid.RowDefinitions>
                <!-- Row definitions for fields -->
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <!-- Row definitions for error messages -->
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <!-- First Name -->
            <Label
                Grid.Row="0"
                Grid.Column="0"/>
```

```

        Text="FirstName"
        VerticalOptions="Center" />
<Entry
    Grid.Row="0"
    Grid.Column="1"
    Placeholder="Enter First Name"
    Text="{Binding FirstName}" />
<Label
    Grid.Row="1"
    Grid.ColumnSpan="2"
    IsVisible="{Binding FirstNameErrorLabel,
Converter={StaticResource StringToBoolConverter}}"
    Text="{Binding FirstNameErrorLabel}"
    TextColor="Red" />

<!-- Last Name -->
<Label
    Grid.Row="2"
    Grid.Column="0"
    Text="LastName"
    VerticalOptions="Center" />
<Entry
    Grid.Row="2"
    Grid.Column="1"
    Placeholder="Enter Last Name"
    Text="{Binding LastName}" />
<Label
    Grid.Row="3"
    Grid.ColumnSpan="2"
    IsVisible="{Binding LastNameErrorLabel,
Converter={StaticResource StringToBoolConverter}}"
    Text="{Binding LastNameErrorLabel}"
    TextColor="Red" />

<!-- Username -->
<Label
    Grid.Row="4"
    Grid.Column="0"
    Text="Username"
    VerticalOptions="Center" />
<Entry
    Grid.Row="4"
    Grid.Column="1"
    Placeholder="Enter Username"
    Text="{Binding Username}" />
<Label
    Grid.Row="5"
    Grid.ColumnSpan="2"
    IsVisible="{Binding UsernameErrorLabel,
Converter={StaticResource StringToBoolConverter}}"
    Text="{Binding UsernameErrorLabel}"
    TextColor="Red" />

<!-- Email -->
<Label
    Grid.Row="6"
    Grid.Column="0"
    Text="Email"
    VerticalOptions="Center" />
<Entry
    Grid.Row="6"
    Grid.Column="1"
    Keyboard="Email"

```

```

        Placeholder="Enter Email"
        Text="{Binding Email}" />
<Label
    Grid.Row="7"
    Grid.ColumnSpan="2"
    IsVisible="{Binding EmailErrorLable,
Converter={StaticResource StringToBoolConverter}}}"
    Text="{Binding EmailErrorLable}"
    TextColor="Red" />

<!-- Password -->
<Label
    Grid.Row="8"
    Grid.Column="0"
    Text="Password"
    VerticalOptions="Center" />
<Entry
    Grid.Row="8"
    Grid.Column="1"
    IsPassword="True"
    Placeholder="Enter Password"
    Text="{Binding Password}" />
<Label
    Grid.Row="9"
    Grid.ColumnSpan="2"
    IsVisible="{Binding PasswordErrorLable,
Converter={StaticResource StringToBoolConverter}}}"
    Text="{Binding PasswordErrorLable}"
    TextColor="Red" />

<!-- Phone -->
<Label
    Grid.Row="10"
    Grid.Column="0"
    Text="Phone"
    VerticalOptions="Center" />
<Entry
    Grid.Row="10"
    Grid.Column="1"
    Keyboard="Telephone"
    Placeholder="Enter Phone"
    Text="{Binding Phone}" />
<Label
    Grid.Row="11"
    Grid.ColumnSpan="2"
    IsVisible="{Binding PhoneErrorLable,
Converter={StaticResource StringToBoolConverter}}}"
    Text="{Binding PhoneErrorLable}"
    TextColor="Red" />

<!-- IsMale: Dropdown: Male/Female -->
<Label
    Grid.Row="12"
    Grid.Column="0"
    Text="Gender"
    VerticalOptions="Center" />
<Picker
    Grid.Row="12"
    Grid.Column="1"
    SelectedItem="{Binding SelectedGender}">
    <Picker.ItemsSource>
        <x:Array Type="{x:Type x:String}">

```

```

        <x:String>Male</x:String>
        <x:String>Female</x:String>
    </x:Array>
    </Picker.ItemsSource>
</Picker>
<!-- Birthdate -->
<Label
    Grid.Row="13"
    Grid.Column="0"
    Text="Birthdate"
    VerticalOptions="Center" />
<DatePicker
    Grid.Row="13"
    Grid.Column="1"
    Date="{Binding Birthdate}" />

<!-- Age -->
<Label
    Grid.Row="14"
    Grid.Column="0"
    Text="Calculated Age:"
    VerticalOptions="Center" />
<Label
    Grid.Row="14"
    Grid.Column="1"
    Text="{Binding CalculatedAge}"
    TextColor="Green" />

<!-- Register Button -->
<Button
    Grid.Row="15"
    Grid.Column="1"
    Command="{Binding RegisterCommand}"
    IsEnabled="{Binding CanRegister}"
    Text="Register" />
<Label
    Grid.Row="15"
    Grid.Column="0"
    Text="Login"
    TextColor="Blue"
    VerticalOptions="Center">
    <Label.GestureRecognizers>
        <TapGestureRecognizer
Tapped="TapGestureRecognizer_Tapped" />
    </Label.GestureRecognizers>
</Label>
<!-- Sow Error Messae - just in case -->
<Label
    Grid.Row="16"
    FontAttributes="Bold"
    Text="{Binding ErrorMessage}"
    TextColor="Red" />
</Grid>
</ScrollView>
</ContentPage>

```


RegistrationPage

```
using PhoneBook.Services;
using PhoneBook.ViewModels;
namespace PhoneBook.Views.Auth;
public partial class RegistrationPage : ContentPage
{
    public RegistrationPage(RegisterViewModel model)
    {
        InitializeComponent();
        BindingContext = model;
    }

    private async void TapGestureRecognizer_Tapped(object sender,
TappedEventArgs e)
    {
        await Shell.Current.GoToAsync("//LoginPage", true);
    }
}
```

ProfilePage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    x:Class="PhoneBook.Views.ProfilePage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:PhoneBook.Helpers"
    xmlns:viewmodels="clr-namespace:PhoneBook.ViewModels"
    Title="Edit Profile"
    x:DataType="viewmodels:ProfileViewModel">

    <ScrollView>
        <Grid
            Padding="20"
            ColumnSpacing="10"
            RowSpacing="10">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.3*" />
                <ColumnDefinition Width="0.7*" />
            </Grid.ColumnDefinitions>

            <Grid.RowDefinitions>
                <!-- Define rows for each field -->
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <!-- First Name -->
            <Label
                Grid.Row="0"
                Grid.Column="0"
                Text="First Name"
                VerticalOptions="Center" />
            <Entry
                Grid.Row="0"
                Grid.Column="1"
                Placeholder="Enter First Name"
                Text="{Binding FirstName}" />

            <!-- Last Name -->
            <Label
                Grid.Row="1"
                Grid.Column="0"
                Text="Last Name"
                VerticalOptions="Center" />
            <Entry
                Grid.Row="1"
                Grid.Column="1"
                Placeholder="Enter Last Name"
                Text="{Binding LastName}" />

            <!-- Username -->
            <Label
                Grid.Row="2"
                Grid.Column="0"
                Text="Username"
```

```

        VerticalOptions="Center" />
<Entry
    Grid.Row="2"
    Grid.Column="1"
    Placeholder="Enter Username"
    Text="{Binding Username}" />

<!-- Email -->
<Label
    Grid.Row="3"
    Grid.Column="0"
    Text="Email"
    VerticalOptions="Center" />
<Entry
    Grid.Row="3"
    Grid.Column="1"
    Placeholder="Enter Email"
    Text="{Binding Email}" />

<!-- Phone -->
<Label
    Grid.Row="4"
    Grid.Column="0"
    Text="Phone"
    VerticalOptions="Center" />
<Entry
    Grid.Row="4"
    Grid.Column="1"
    Placeholder="Enter Phone"
    Text="{Binding Phone}" />

<!-- Gender -->
<Label
    Grid.Row="5"
    Grid.Column="0"
    Text="Gender"
    VerticalOptions="Center" />
<Picker
    Grid.Row="5"
    Grid.Column="1"
    SelectedItem="{Binding SelectedGender}">
    <Picker.ItemsSource>
        <x:Array Type="{x:Type x:String}">
            <x:String>Male</x:String>
            <x:String>Female</x:String>
        </x:Array>
    </Picker.ItemsSource>
</Picker>

<!-- Birthdate -->
<Label
    Grid.Row="6"
    Grid.Column="0"
    Text="Birthdate"
    VerticalOptions="Center" />
<DatePicker
    Grid.Row="6"
    Grid.Column="1"
    Date="{Binding Birthdate}" />

<!-- Save Button -->

```

```

        <Button
            Grid.Row="7"
            Grid.Column="1"
            Command="{Binding SaveProfileCommand}"
            Text="Save Profile" />
    </Grid>
</ScrollView>
</ContentPage>

```

ProfilePage.xaml.cs

```

using PhoneBook.ViewModels;

namespace PhoneBook.Views;

public partial class ProfilePage : ContentPage
{
    public ProfilePage(ProfileViewModel model)
    {
        InitializeComponent();
        BindingContext = model;
    }
    protected override void OnAppearing()
    {
        base.OnAppearing();
        ((ProfileViewModel)BindingContext).LoadUserProfile();
    }
}

```

AllContactsPage

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
  x:Class="PhoneBook.Views.AllContactsPage"
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  Title="All Contacts"
  BackgroundColor="#f5f5f5">

  <!-- Main Layout -->
  <StackLayout Padding="10">

    <!-- Page Header -->
    <Label
      Margin="0,20,0,10"
      FontAttributes="Bold"
      FontSize="24"
      HorizontalOptions="Center"
      Text="All Contacts"
      TextColor="#333"
      VerticalOptions="Start" />

    <!-- Busy Indicator -->
    <ActivityIndicator
      IsRunning="{Binding IsBusy}"
      IsVisible="{Binding IsBusy}"
      VerticalOptions="CenterAndExpand"
      Color="Gray" />

    <!-- Contacts List -->
    <CollectionView
      ItemsSource="{Binding Contacts}"
      SelectionChanged="OnContactSelected"
      SelectionMode="Single">
      <CollectionView.ItemTemplate>
        <DataTemplate>
          <Frame
            Margin="5,10"
            Padding="10"
            BackgroundColor="White"
            CornerRadius="10"
            HasShadow="True">
            <StackLayout Orientation="Horizontal"

Spacing="15">

              <!-- Contact Profile Image (or avatar) -->
              <Image
                Aspect="AspectFill"
                Clip="EllipseGeometry"
                HeightRequest="60"
                HorizontalOptions="Start"
                Source="{Binding DisplayedProfileImage}"
                VerticalOptions="Center"
                WidthRequest="60" />

              <!-- Contact Details -->
              <StackLayout
                HorizontalOptions="FillAndExpand"
                Orientation="Vertical"
                Spacing="5"
                VerticalOptions="Center">
                <Label
```

```

        FontAttributes="Bold"
        FontSize="18"
        Text="{Binding FirstName} {Binding
LastName}"

        TextColor="#333" />
<Label
    FontAttributes="Italic"
    FontSize="14"
    Text="{Binding Phone}"
    TextColor="Gray" />
<Label
    FontSize="14"
    Text="{Binding Email}"
    TextColor="Gray" />
</StackLayout>
<StackLayout
    HorizontalOptions="End"
    Orientation="Vertical"
    WidthRequest="80">
    <Button
        Margin="0,0,0,5"
        BindingContext="{Binding .}"
        Clicked="OnViewContactClicked"
        HeightRequest="40"
        HorizontalOptions="Fill"
        Text="View"
        VerticalOptions="Start"
        WidthRequest="70" />

    <Button
        Margin="0"
        BindingContext="{Binding .}"
        Clicked="OnDeleteContactClicked"
        HeightRequest="40"
        HorizontalOptions="Fill"
        Text="Delete"
        VerticalOptions="End"
        WidthRequest="70" />
</StackLayout>
</StackLayout>
</Frame>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>
</StackLayout>
</ContentPage>

```

AllContactsPage.xaml.cs

```
using PhoneBook.Models;
using PhoneBook.Services;
using PhoneBook.ViewModels;

namespace PhoneBook.Views;

public partial class AllContactsPage : ContentPage
{
    public AllContactsPage(AllContactsViewModel model)
    {
        InitializeComponent();
        BindingContext = model;
    }
    protected override void OnAppearing()
    {
        base.OnAppearing();
        //Force reload the list - since this could come from after
        "Delete" or "New" contact
        ((AllContactsViewModel)BindingContext).LoadContactsCommand.Execute(null);
    }
    private async void OnViewContactClicked(object sender, EventArgs e)
    {
        // Get the button that was clicked
        var button = sender as Button;

        // Retrieve the contact associated with the clicked button
        var contact = button?.BindingContext as AppContact;

        if (contact != null)
        {
            // Navigate to ContactPage and pass the selected contact
            await Shell.Current.GoToAsync(nameof(ContactPage), new
            Dictionary<string, object>
            {
                { "contact", contact }
            });
        }
    }
    private async void OnContactSelected(object sender,
    SelectionChangedEventArgs e)
    {
        if (e.CurrentSelection.FirstOrDefault() is AppContact
        selectedContact)
        {
            // Navigate to ContactPage and pass the selected contact
            await Shell.Current.GoToAsync(nameof(ContactPage), new
            Dictionary<string, object>
            {
                { "contact", selectedContact }
            });
        }
    }
    private async void OnDeleteContactClicked(object sender, EventArgs e)
    {
        // Get the button that was clicked
        var button = sender as Button;

        // Retrieve the contact associated with the clicked button
        var contact = button?.BindingContext as AppContact;
```

```

        if (contact != null)
        {
            // Confirm deletion
            bool confirm = await
Application.Current.MainPage.DisplayAlert("Confirm Delete", $"Are you sure
you want to delete {contact.FirstName} {contact.LastName}?", "Yes", "No");

            if (confirm)
            {
                var contactService =
IPlatformApplication.Current.Services.GetService<ContactService>();
                // Delete the contact
                await contactService.DeleteContactAsync(contact.Id);
                // Display success message
                await Application.Current.MainPage.DisplayAlert("Success",
"Contact deleted successfully", "OK");
                //refresh the contacts list
                LoadContacts();
            }
        }
    }

    private void LoadContacts()
    {
        // Implement the logic to reload contacts
        ((AllContactsViewModel)BindingContext).LoadContactsCommand.Execute(null);
    }
}

```


ContactPage

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
  x:Class="PhoneBook.Views.ContactPage"
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:vm="clr-namespace:PhoneBook.ViewModels"
  Title="Contact Details"
  x:DataType="vm:ContactViewModel"
  BackgroundColor="#f5f5f5">

  <StackLayout Padding="20">

    <!-- Contact Profile Image (or default avatar) -->
    <Image
      Aspect="AspectFill"
      BackgroundColor="#ccc"
      Clip="EllipseGeometry"
      HeightRequest="100"
      HorizontalOptions="Center"
      Source="{Binding DisplayedProfileImage}"
      WidthRequest="100" />

    <!-- Contact Details -->
    <Label
      Margin="0,10,0,0"
      FontAttributes="Bold"
      FontSize="24"
      HorizontalOptions="Center"
      Text="{Binding FullName}"
      TextColor="#333" />
    <Label
      FontAttributes="Italic"
      FontSize="18"
      HorizontalOptions="Center"
      Text="{Binding Phone}"
      TextColor="Gray" />
    <Label
      FontSize="18"
      HorizontalOptions="Center"
      Text="{Binding Email}"
      TextColor="Gray" />

    <!-- Edit and Delete Buttons -->
    <StackLayout
      Margin="0,30,0,0"
      HorizontalOptions="Center"
      Orientation="Horizontal"
      Spacing="20">
      <Button
        BackgroundColor="Orange"
        Command="{Binding EditCommand}"
        Text="Edit"
        TextColor="White" />
      <Button
        BackgroundColor="Red"
        Command="{Binding DeleteCommand}"
        Text="Delete"
        TextColor="White" />
    </StackLayout>
  </StackLayout>
</ContentPage>
```

ContactPage.xaml.cs

```
using PhoneBook.Models;
using PhoneBook.ViewModels;

namespace PhoneBook.Views;
[QueryProperty(nameof(Contact), "contact")] // Register the query
property
public partial class ContactPage : ContentPage
{
    public ContactPage()
    {
        InitializeComponent();
    }

    private AppContact _contact;
    public AppContact Contact
    {
        get => _contact;
        set
        {
            _contact = value;
            BindingContext = new ContactViewModel(_contact); // Set the
ViewModel when the contact is set
        }
    }
}
```

NewContactPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    x:Class="PhoneBook.Views.NewContactPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:vm="clr-namespace:PhoneBook.ViewModels"
    Title="New Contact"
    x:DataType="vm:NewContactViewModel">
    <ScrollView>
        <Grid
            Padding="20"
            ColumnSpacing="10"
            RowSpacing="10">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.3*" />
                <ColumnDefinition Width="0.7*" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <!-- Row definitions for fields -->
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <!-- Profile Image (URL or Path) -->

            <!-- Profile Image (or default avatar) -->
            <Image
                Grid.Row="0"
                Grid.Column="1"
                Aspect="AspectFill"
                BackgroundColor="#ccc"
                Clip="EllipseGeometry"
                HeightRequest="100"
                HorizontalOptions="Center"
                Source="{Binding DisplayedProfileImage}"
                WidthRequest="100" />

            <!-- Button to Select Image -->
            <Button
                Grid.Row="0"
                Grid.Column="0"
                Command="{Binding SelectImageCommand}"
                Text="Select Profile Image" />

            <!-- Other fields and Save button -->
            <!-- First Name -->
            <Label
                Grid.Row="1"
                Grid.Column="0"
                Text="First Name"
                VerticalOptions="Center" />
            <Entry
                Grid.Row="1"
                Grid.Column="1"
                />
        </Grid>
    </ScrollView>
</ContentPage>
```

```

        Placeholder="Enter First Name"
        Text="{Binding FirstName}" />

<!-- Last Name -->
<Label
    Grid.Row="2"
    Grid.Column="0"
    Text="Last Name"
    VerticalOptions="Center" />
<Entry
    Grid.Row="2"
    Grid.Column="1"
    Placeholder="Enter Last Name"
    Text="{Binding LastName}" />

<!-- Email -->
<Label
    Grid.Row="3"
    Grid.Column="0"
    Text="Email"
    VerticalOptions="Center" />
<Entry
    Grid.Row="3"
    Grid.Column="1"
    Keyboard="Email"
    Placeholder="Enter Email"
    Text="{Binding Email}" />

<!-- Phone -->
<Label
    Grid.Row="4"
    Grid.Column="0"
    Text="Phone"
    VerticalOptions="Center" />
<Entry
    Grid.Row="4"
    Grid.Column="1"
    Keyboard="Telephone"
    Placeholder="Enter Phone"
    Text="{Binding Phone}" />

<!-- IsMale: Dropdown: Male/Female -->
<Label
    Grid.Row="5"
    Grid.Column="0"
    Text="Gender"
    VerticalOptions="Center" />
<Picker
    Grid.Row="5"
    Grid.Column="1"
    SelectedItem="{Binding SelectedGender}">
    <Picker.ItemsSource>
        <x:Array Type="{x:Type x:String}">
            <x:String>Male</x:String>
            <x:String>Female</x:String>
        </x:Array>
    </Picker.ItemsSource>
</Picker>
<!-- Birthdate -->
<Label
    Grid.Row="6"
    Grid.Column="0"
    Text="Birthdate"

```

```

        VerticalOptions="Center" />
<DatePicker
    Grid.Row="6"
    Grid.Column="1"
    Date="{Binding Birthdate}" />

<!-- Save Button -->
<Button
    Grid.Row="8"
    Grid.Column="1"
    Command="{Binding SaveContactCommand}"
    IsEnabled="{Binding CanSave}"
    Text="Save" />
</Grid>
</ScrollView>
</ContentPage>

```

NewContactPage.xaml.cs

```

using PhoneBook.ViewModels;

namespace PhoneBook.Views;

public partial class NewContactPage : ContentPage
{
    public NewContactPage(NewContactViewModel model)
    {
        InitializeComponent();
        BindingContext = model;
    }
}

```

References

Class room:

<https://classroom.google.com/c/Njk2ODY2MTg5NTMw>

Microsoft Learn

<https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-8.0>

Online Resources:

YouTube:

https://www.youtube.com/playlist?list=PLdo4fOcmZ0oVqdnxkgCtMu0m-86lML_IW

Github:

<https://github.com/jsuarezruiz/awesome-dotnet-maui#ui>

<https://github.com/stma1one>

AI Resources

Chat GPT