Министерство образования Республики Беларусь

Учреждение образования «Институт информационных технологий Белорусского государственного университета информатики и

радиоэлектроники»

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Дисциплина: Разработка программного обеспечения для мобильных платформ.

ОТЧЁТ

по лабораторной работе №4

Вариант № 6

Выполнил: студент гр. 981064 Ефименко П. В.

Проверил: Калитеня И. Л.

Минск 2021

**Цель:** закрепить полученные знания.

**Задание:** реализовать мобильное приложения «Парсер из соцсети»:

1. Парсинг пользователя через id.
2. Вывод фамилии и имени пользователя.
3. Вывод фото пользователя.
4. Вывод телефона пользователя.
5. Вывод места проживания

Код программы:

using LiteDB;

using Laba\_4.DataCore;

using Laba\_4.Presentation.Views.Contacts;

using Laba\_4.Services;

using Xamarin.Forms;

namespace Laba\_4

{

    public partial class App : Application

    {

        private readonly IContainer \_container;

        private readonly INavigationService \_navigationService;

        public App()

        {

            InitializeComponent();

            \_container = new Container(this);

            \_navigationService = \_container.GetNavigationService();

            \_navigationService.PushContactsPageAsync();

        }

        protected override void OnStart()

        {

            // Handle when your app starts

        }

        protected override void OnSleep()

        {

            // Handle when your app sleeps

        }

        protected override void OnResume()

        {

            // Handle when your app resumes

        }

    }

}

using Xamarin.Forms.Xaml;

[assembly: XamlCompilation(XamlCompilationOptions.Compile)]

using System;

using System.ComponentModel;

using Laba\_4.DataCore;

using Laba\_4.Presentation.ViewModels;

using Xamarin.Forms;

namespace Laba\_4

{

    // Learn more about making custom code visible in the Xamarin.Forms previewer

    // by visiting https://aka.ms/xamarinforms-previewer

    [DesignTimeVisible(true)]

    public partial class MainPage : ContentPage

    {

        public MainPage()

        {

            InitializeComponent();

        }

    }

}

using Laba\_4.Models;

using Xamarin.Forms;

namespace Laba\_4.ControlsBehavior

{

    public class ContactDataSelector : DataTemplateSelector

    {

        public DataTemplate DefaultTemplate { get; set; }

        public DataTemplate WorkTemplate { get; set; }

        protected override DataTemplate OnSelectTemplate(object item, BindableObject container)

        {

            var obj = (Contact)item;

            return obj.PhoneType == ContactType.None ? DefaultTemplate : WorkTemplate;

        }

    }

}

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Input;

using Xamarin.Forms;

namespace Laba\_4.ControlsBehavior

{

    public class ListViewTappedBehavior : BindableObject

    {

        public static readonly BindableProperty CommandProperty =

            BindableProperty.CreateAttached(

                nameof(Command),

                typeof(ICommand),

                typeof(ListViewTappedBehavior),

                null,

                propertyChanged: OnCommandChanged);

        public ICommand Command

        {

            get => (ICommand)GetValue(CommandProperty);

            set => SetValue(CommandProperty, value);

        }

        static void OnCommandChanged(BindableObject view, object oldValue, object newValue)

        {

            var entry = view as ListView;

            if (entry == null)

                return;

            entry.ItemTapped += (sender, e) =>

            {

                var command = (newValue as ICommand);

                if (command == null)

                    return;

                if (command.CanExecute(e.Item))

                {

                    command.Execute(e.Item);

                }

            };

        }

    }

}

using System.Collections.Generic;

using System.Threading.Tasks;

using Laba\_4.Models;

namespace Laba\_4.DataCore

{

    public interface IRepository

    {

        Task AddOrUpdateAllAsync<T>(IEnumerable<T> entities) where T : class, IBaseEntity;

        Task<IList<T>> GetAllAsync<T>() where T : class, IBaseEntity;

        Task RemoveAllAsync<T>() where T : class, IBaseEntity;

    }

}

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using LiteDB;

using Laba\_4.Models;

namespace Laba\_4.DataCore

{

    public class LiteDbRepository : IRepository

    {

        private readonly LiteRepository \_liteRepository;

        public LiteDbRepository(LiteRepository liteRepository)

        {

            \_liteRepository = liteRepository ?? throw new ArgumentNullException(nameof(liteRepository));

        }

        public Task AddOrUpdateAllAsync<T>(IEnumerable<T> entities) where T : class, IBaseEntity

        {

            return Task.Run(() => \_liteRepository.Upsert<T>(entities));

        }

        public async Task<IList<T>> GetAllAsync<T>() where T : class, IBaseEntity

        {

            return await Task.Run(() => \_liteRepository.Fetch<T>());

        }

        public Task RemoveAllAsync<T>() where T : class, IBaseEntity

        {

            return Task.Run(() => \_liteRepository.Delete<T>(LiteDB.Query.All()));

        }

    }

}

using System;

using System.IO;

using System.Threading.Tasks;

using LiteDB;

using Xamarin.Essentials;

namespace Laba\_4.DataCore

{

    public class LiteDbRepositoryFactory

    {

        private const string DbDir = "DataStorage";

        private readonly string \_dbFilePath;

        public LiteDbRepositoryFactory(string fileName)

        {

            if (string.IsNullOrWhiteSpace(fileName)) throw new ArgumentException(nameof(fileName));

            var dirPath = Path.Combine(FileSystem.CacheDirectory, DbDir);

            Directory.CreateDirectory(dirPath);

            \_dbFilePath = Path.Combine(dirPath, fileName);

        }

        public LiteRepository GetRepository()

        {

            return new LiteRepository(\_dbFilePath);

        }

    }

}

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Laba\_4.Models;

namespace Laba\_4.DataCore

{

    public class MemoryRepository : IRepository

    {

        private readonly IList<object> \_dbList;

        public MemoryRepository()

        {

            \_dbList = new List<object>();

        }

        public Task AddOrUpdateAllAsync<T>(IEnumerable<T> entities) where T : class, IBaseEntity

        {

            RemoveAllAsync<T>();

            return Task.Run(() =>

            {

                entities = AddIndexToEntities<T>(entities);

                foreach (var entity in entities)

                {

                    \_dbList.Add(entity);

                }

                return true;

            });

        }

        public async Task<IList<T>> GetAllAsync<T>() where T : class, IBaseEntity

        {

            return await Task.Run(() => \_dbList.OfType<T>().ToList());

        }

        public Task RemoveAllAsync<T>() where T : class, IBaseEntity

        {

            return Task.Run(() =>

            {

                \_dbList.Clear();

                return true;

            });

        }

        private IEnumerable<T> AddIndexToEntities<T>(IEnumerable<T> entities) where T : class, IBaseEntity

        {

            var list= entities.ToList();

            for (int i = 0; i < list.Count; i++)

            {

                var item = list[i];

                item.Id = i;

            }

            return list;

        }

    }

}

using System.ComponentModel;

namespace Laba\_4.Helpers

{

    public class ObservableObject<T> : INotifyPropertyChanged

    {

        private T \_property;

        public T Property

        {

            get => \_property;

            set

            {

                if (\_property != null && \_property.Equals(value)) return;

                \_property = value;

                OnPropertyChanged();

            }

        }

        public event PropertyChangedEventHandler PropertyChanged;

        public void OnPropertyChanged()

        {

            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Property"));

        }

    }

}

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Collections.Specialized;

using System.ComponentModel;

namespace Laba\_4.Helpers

{

    public class ObservableRangeCollection<T> : ObservableCollection<T>

    {

        public ObservableRangeCollection() : base()

        {

        }

        public ObservableRangeCollection(IEnumerable<T> collection) : base(collection)

        {

        }

        public void AddRange(IEnumerable<T> collection)

        {

            if (collection == null)

                throw new ArgumentNullException(nameof(collection));

            CheckReentrancy();

            int startIndex = Count;

            var changedItems = collection is List<T> ? (List<T>)collection : new List<T>(collection);

            foreach (var i in changedItems)

                Items.Add(i);

            OnPropertyChanged(new PropertyChangedEventArgs("Count"));

            OnPropertyChanged(new PropertyChangedEventArgs("Item[]"));

            OnCollectionChanged(new NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Add, changedItems, startIndex));

        }

    }

}

using System.Collections.Generic;

namespace Laba\_4.Models

{

    public class Contact : IBaseEntity

    {

        public string PhotoPath { get; set; }

        public string Name { get; set; }

        public string PhoneNumber { get; set; }

        public string Region { get; set; }

        public string ImageUrl { get; set; }

        public ContactType PhoneType { get; set; }

        public int Id { get; set; }

        public override bool Equals(object obj)

        {

            return obj is Contact contact &&

                   PhotoPath == contact.PhotoPath &&

                   Name == contact.Name &&

                   PhoneNumber == contact.PhoneNumber &&

                   PhoneType == contact.PhoneType &&

                   Id == contact.Id;

        }

        public override int GetHashCode()

        {

            var hashCode = 1018803031;

            hashCode = hashCode \* -1521134295 + EqualityComparer<string>.Default.GetHashCode(PhotoPath);

            hashCode = hashCode \* -1521134295 + EqualityComparer<string>.Default.GetHashCode(Name);

            hashCode = hashCode \* -1521134295 + EqualityComparer<string>.Default.GetHashCode(PhoneNumber);

            hashCode = hashCode \* -1521134295 + PhoneType.GetHashCode();

            hashCode = hashCode \* -1521134295 + Id.GetHashCode();

            return hashCode;

        }

        public static bool operator ==(Contact left, Contact right)

        {

            return EqualityComparer<Contact>.Default.Equals(left, right);

        }

        public static bool operator !=(Contact left, Contact right)

        {

            return !(left == right);

        }

    }

    public class ContactIdComparer : IEqualityComparer<Contact>

    {

        public bool Equals(Contact x, Contact y)

        {

            return x.Id == y.Id;

        }

        public int GetHashCode(Contact obj)

        {

            return obj.GetHashCode();

        }

    }

    public enum ContactType

    {

        None,

        WorkPhone

    }

}

using System;

using System.Collections.Generic;

using System.Text;

namespace Laba\_4.Models

{

    public interface IBaseEntity

    {

        int Id { get; set; }

    }

}

using System;

using Laba\_4.DataCore;

using Xamarin.Forms;

namespace Laba\_4.Services

{

    public class Container : IContainer

    {

        private readonly Application \_application;

        private readonly IRepository \_repositoryLazy;

        private readonly INavigationService \_navigationServiceLazy;

        public Container(Application application)

        {

            //var liteDbRepositoryFactory = new LiteDbRepositoryFactory("contact.db");

            //var repository = new LiteDbRepository(liteDbRepositoryFactory.GetRepository());

            var repository = new MemoryRepository();

            \_application = application;

            \_repositoryLazy = repository;

            \_navigationServiceLazy = new NavigationService(\_application, this);

        }

        public IRepository GetRepository()

        {

            return \_repositoryLazy;

        }

        public INavigationService GetNavigationService()

        {

            return \_navigationServiceLazy;

        }

    }

}

using Laba\_4.DataCore;

namespace Laba\_4.Services

{

    public interface IContainer

    {

        IRepository GetRepository();

        INavigationService GetNavigationService();

    }

}

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

using Laba\_4.Models;

namespace Laba\_4.Services

{

    public interface INavigationService

    {

        Task PopAsync();

        Task PushContactDetailsPageAsync(Contact contact);

        Task PushContactsPageAsync();

        Task PushEditContactAsync();

        Task PushEditContactAsync(Contact contact);

    }

}

using System;

using System.Collections.Generic;

using System.Text;

namespace Laba\_4.Services

{

    public interface IToast

    {

        void Show(string message);

    }

}

using System;

using System.Threading.Tasks;

using Laba\_4.Models;

using Laba\_4.Presentation.ViewModels;

using Laba\_4.Presentation.Views.ContactDetails;

using Laba\_4.Presentation.Views.Contacts;

using Laba\_4.Presentation.Views.EditContact;

using Xamarin.Forms;

namespace Laba\_4.Services

{

    public class NavigationService : INavigationService

    {

        private readonly Application \_application;

        private readonly IContainer \_container;

        private INavigation Navigation => \_application.MainPage.Navigation;

        public NavigationService(Application application, IContainer container)

        {

            \_application = application ?? throw new ArgumentNullException(nameof(application));

            \_container = container ?? throw new ArgumentNullException(nameof(container));

            var page = new MainPage();

            CreateNewNavigation(page);

        }

        public Task PopAsync()

        {

            return Navigation.PopModalAsync();

        }

        public Task PushContactDetailsPageAsync(Contact contact)

        {

            var viewModel = new ContactDetailsViewModel(\_container.GetNavigationService(),

                \_container.GetRepository(),

                contact);

            var page = new ContactDetailsViewPage(viewModel);

            return Navigation.PushModalAsync(page);

        }

        public Task PushContactsPageAsync()

        {

            var viewModel = new ContactsViewModel(\_container.GetRepository(),

                \_container.GetNavigationService());

            var page = new ContactsViewPage(viewModel);

            return Navigation.PushModalAsync(page);

        }

        public Task PushEditContactAsync()

        {

            var viewModel = new EditContactViewModel(\_container.GetRepository(),

                \_container.GetNavigationService());

            var page = new EditContactPage(viewModel);

            return Navigation.PushModalAsync(page);

        }

        public Task PushEditContactAsync(Contact contact)

        {

            var viewModel = new EditContactViewModel(\_container.GetRepository(),

                \_container.GetNavigationService(),

                contact);

            var page = new EditContactPage(viewModel);

            return Navigation.PushModalAsync(page);

        }

        private void CreateNewNavigation(Page page)

        {

            var navigationPage = new NavigationPage(page);

            \_application.MainPage = navigationPage.RootPage;

        }

    }

}

using System;

using System.Collections.Generic;

using System.Text;

namespace Laba\_4.Validations

{

    public interface IBaseValidationRule<in T>

    {

        IReadOnlyList<string> Properties { get; }

        string ValidationMessage { get; set; }

        bool Check(T value);

    }

}

namespace Laba\_4.Validations

{

    public interface IValidity

    {

        bool IsValid { get; set; }

    }

}

using System.Collections.Generic;

using System.Linq;

namespace Laba\_4.Validations

{

    public class ValidableObject<T> : IValidity

    {

        private readonly IReadOnlyList<IBaseValidationRule<T>> \_rules;

        public bool IsValid { get; set; }

        public ValidableObject(IReadOnlyList<IBaseValidationRule<T>> rules)

        {

            IsValid = false;

            \_rules = rules;

        }

        public ValidationResult Validate(T value)

        {

            var errors = \_rules.Where(item => !item.Check(value))

                .ToDictionary(x => x.Properties, y => y.ValidationMessage);

            var validationResult = new ValidationResult(errors);

            return validationResult;

        }

    }

}

using System.Collections.Generic;

using System.Linq;

namespace Laba\_4.Validations

{

    public class ValidationResult

    {

        public bool IsValid => !Errors.Any();

        public IDictionary<IReadOnlyList<string>, string> Errors { get; }

        public ValidationResult(Dictionary<IReadOnlyList<string>, string> errors)

        {

            if (errors == null)

            {

                errors = new Dictionary<IReadOnlyList<string>, string>();

            }

            Errors = errors;

        }

    }

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using Laba\_4.DataCore;

namespace Laba\_4.Presentation.ViewModels

{

    public class BaseViewModel : INotifyPropertyChanged

    {

        private bool \_isBusy;

        public bool IsBusy

        {

            get => \_isBusy;

            set => SetProperty(ref \_isBusy, value);

        }

        private string \_title = string.Empty;

        public string Title

        {

            get => \_title;

            set => SetProperty(ref \_title, value);

        }

        protected bool SetProperty<T>(ref T backingStore, T value,

            [CallerMemberName]string propertyName = "",

            Action onChanged = null)

        {

            if (EqualityComparer<T>.Default.Equals(backingStore, value))

                return false;

            backingStore = value;

            onChanged?.Invoke();

            OnPropertyChanged(propertyName);

            return true;

        }

        public event PropertyChangedEventHandler PropertyChanged;

        public void OnPropertyChanged([CallerMemberName] string propertyName = "")

        {

            var changed = PropertyChanged;

            changed?.Invoke(this, new PropertyChangedEventArgs(propertyName));

        }

    }

}

using System;

using System.Diagnostics;

using System.Linq;

using System.Windows.Input;

using Laba\_4.DataCore;

using Laba\_4.Helpers;

using Laba\_4.Services;

using Xamarin.Essentials;

using Xamarin.Forms;

namespace Laba\_4.Presentation.ViewModels

{

    public class ContactDetailsViewModel

    {

        private readonly INavigationService \_navigationService;

        private readonly IRepository \_repository;

        public ContactDetailsViewModel(INavigationService navigationService, IRepository repository, Models.Contact contact)

        {

            \_navigationService = navigationService ?? throw  new ArgumentNullException(nameof(navigationService));

            \_repository = repository ?? throw new ArgumentNullException(nameof(repository));

            Contact = new ObservableObject<Models.Contact> { Property = contact };

            CallCommand = new Command(OnCall);

            EditContactCommand = new Command(OnEditContact);

            RemoveContactCommand = new Command(OnRemoved);

            BackCommand = new Command(OnBack);

        }

        private async void OnRemoved()

        {

            var contacts = await \_repository.GetAllAsync<Models.Contact>();

            var contact = contacts.First(item => item.Equals(Contact.Property));

            contacts.Remove(contact);

            await \_repository.RemoveAllAsync<Models.Contact>();

            await \_repository.AddOrUpdateAllAsync(contacts);

            await \_navigationService.PopAsync();

        }

        public ObservableObject<Models.Contact> Contact { get; set; }

        public ICommand CallCommand { get; set; }

        public ICommand EditContactCommand { get; set; }

        public ICommand RemoveContactCommand { get; set; }

        public ICommand BackCommand { get; set; }

        private void OnEditContact()

        {

            var contact = Contact.Property;

            \_navigationService.PushEditContactAsync(contact);

        }

        private void OnBack()

        {

            \_navigationService.PopAsync();

        }

        private void OnCall()

        {

            var phoneNumber = Contact.Property.PhoneNumber;

            Device.BeginInvokeOnMainThread(() => Call(phoneNumber));

        }

        private void Call(string phoneNumber)

        {

            try

            {

                if (Device.Idiom == TargetIdiom.Desktop)

                {

                    Device.OpenUri(new Uri("tel:" + phoneNumber));

                }

                else

                {

                    PhoneDialer.Open(phoneNumber);

                }

            }

            catch (FeatureNotSupportedException ex)

            {

                DependencyService.Get<IToast>().Show("Phone Dialer is not supported on this device!");

            }

            catch (Exception ex)

            {

                Debug.WriteLine(ex.Message);

            }

        }

    }

}

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Threading.Tasks;

using System.Windows.Input;

using LiteDB;

using Laba\_4.DataCore;

using Laba\_4.Helpers;

using Laba\_4.Models;

using Laba\_4.Services;

using Xamarin.Forms;

namespace Laba\_4.Presentation.ViewModels

{

    public class ContactsViewModel : BaseViewModel

    {

        private readonly IRepository \_repository;

        private readonly INavigationService \_navigationService;

        public ContactsViewModel(IRepository repository, INavigationService navigationService)

        {

            \_repository = repository ?? throw new ArgumentNullException(nameof(repository)); ;

            \_navigationService = navigationService ?? throw new ArgumentNullException(nameof(navigationService)); ;

            LoadCommand = new Command(async () => await OnExecuteLoadCommand());

            TappedCommand = new Command(OnItemTapped);

            AddContactCommand = new Command(OnAddContact);

            ContactItems = new ObservableRangeCollection<Contact>();

        }

        public ObservableRangeCollection<Contact> ContactItems { get; set; }

        public ICommand LoadCommand { get; private set; }

        public ICommand TappedCommand { get; private set; }

        public ICommand AddContactCommand { get; private set; }

        private void OnItemTapped(object obj)

        {

            var contact = (Contact) obj;

            Device.BeginInvokeOnMainThread(async () =>  await \_navigationService.PushContactDetailsPageAsync(contact));

        }

        private void OnAddContact()

        {

            Device.BeginInvokeOnMainThread(async () => await \_navigationService.PushEditContactAsync());

        }

        private async Task OnExecuteLoadCommand()

        {

            //if (IsBusy) return;

            IsBusy = true;

            try

            {

                ContactItems.Clear();

                var contacts = await \_repository.GetAllAsync<Contact>();

                ContactItems.AddRange(contacts);

            }

            catch (Exception ex)

            {

                Debug.WriteLine(ex);

            }

            finally

            {

                IsBusy = false;

            }

        }

    }

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Input;

using AngleSharp;

using AngleSharp.XPath;

using CsQuery;

using Laba\_4.DataCore;

using Laba\_4.Models;

using Laba\_4.Services;

using Xamarin.Forms;

using static System.String;

namespace Laba\_4.Presentation.ViewModels

{

    public class EditContactViewModel : BaseViewModel

    {

        private readonly IRepository \_repository;

        private readonly INavigationService \_navigationService;

        private string \_name;

        private string \_phoneNumber;

        private string \_phoneType;

        private string \_imageUrl;

        private string \_region;

        private string[] \_phoneTypes;

        private bool \_isValid;

        private int \_contactId;

        private string \_vkUrl;

        public EditContactViewModel(IRepository repository, INavigationService navigationService, Contact contact = null)

        {

            \_repository = repository ?? throw new ArgumentNullException(nameof(repository));

            \_navigationService = navigationService ?? throw new ArgumentNullException(nameof(navigationService));

            SubmitCommand = new Command(OnSubmit);

            BackCommand = new Command(OnBack);

            ImportCommand = new Command(async() => await OnImportAsync());

            Name = contact?.Name;

            PhoneNumber = contact?.PhoneNumber;

            PhoneType = contact?.PhoneType.ToString();

            PhoneTypes = Enum.GetNames(typeof(ContactType));

            Region = contact?.Region;

            ImageUrl = contact?.ImageUrl;

            \_contactId = contact?.Id ?? -1;

            IsValid = true;

        }

        private async Task OnImportAsync()

        {

            try

            {

                if (!string.IsNullOrWhiteSpace(VkUrl))

                {

                    var config = Configuration.Default.WithDefaultLoader().WithXPath();

                    var address = $"https://vkfaces.com/vk/user/{VkUrl}";

                    var context = BrowsingContext.New(config);

                    var document = await context.OpenAsync(address);

                    var name = document.Body.QuerySelectorAll("div[class=\"list-info\"]>div[class=\"list-info\_\_title\"]")?.First().TextContent;

                    var region = document.Body.QuerySelectorAll("div[class=\"list-info\"]>div[class=\"list-info\_\_subtitle\"]")?.First().TextContent.Replace("\n", "");

                    var img = document.Body.QuerySelectorAll("div[class=\"avatar \_elastic\_down js-popup-gallery-trigger \_pointer\"]>img[class=\"avatar\_\_img\"]")?.First().GetAttribute("src");

                    Name = name;

                    Region = region;

                    ImageUrl = img;

                }

            }

            catch (Exception)

            {

            }

        }

        public ICommand SubmitCommand { get; set; }

        public ICommand BackCommand { get; set; }

        public ICommand ImportCommand { get; set; }

        public string Name

        {

            get => \_name;

            set => SetProperty(ref \_name, value);

        }

        public string VkUrl

        {

            get => \_vkUrl;

            set => SetProperty(ref \_vkUrl, value);

        }

        public string Region

        {

            get => \_region;

            set => SetProperty(ref \_region, value);

        }

        public string ImageUrl

        {

            get => \_imageUrl;

            set => SetProperty(ref \_imageUrl, value);

        }

        public string PhoneNumber

        {

            get => \_phoneNumber;

            set => SetProperty(ref \_phoneNumber, value);

        }

        public string PhoneType

        {

            get => \_phoneType;

            set => SetProperty(ref \_phoneType, value);

        }

        public string[] PhoneTypes

        {

            get => \_phoneTypes;

            set => SetProperty(ref \_phoneTypes, value);

        }

        public bool IsValid

        {

            get => \_isValid;

            set => SetProperty(ref \_isValid, value);

        }

        private async void OnSubmit()

        {

            Enum.TryParse(PhoneType, out ContactType contactType);

            var contact = new Contact

            {

                PhoneType = contactType,

                Name = Name,

                PhoneNumber = PhoneNumber,

                PhotoPath = string.Empty,

                ImageUrl = ImageUrl,

                Region = Region,

                Id = \_contactId

            };

            var contacts = await \_repository.GetAllAsync<Contact>();

            if (contacts.Contains(contact, new ContactIdComparer()))

            {

                var entity = contacts.First(item => item.Id == contact.Id);

                entity.Id = contact.Id;

                entity.Name = contact.Name;

                entity.PhoneNumber = contact.PhoneNumber;

                entity.PhoneType = contact.PhoneType;

                entity.PhotoPath = contact.PhotoPath;

                entity.ImageUrl = contact.ImageUrl;

                entity.Region = contact.Region;

            }

            else

            {

                contacts.Add(contact);

            }

            await \_repository.AddOrUpdateAllAsync(contacts);

        }

        private async void OnBack()

        {

            await \_navigationService.PopAsync();

        }

    }

}

using System;

using System.Collections.Generic;

using System.Text;

using Laba\_4.Models;

namespace Laba\_4.Validations.Rules

{

    public class IsNameNotNullOrWhiteSpace : IBaseValidationRule<Contact>

    {

        public IReadOnlyList<string> Properties { get; } = new List<string>() { nameof(Contact.Name) };

        public string ValidationMessage { get; set; }

        public bool Check(Contact value)

        {

            return !string.IsNullOrWhiteSpace(value.Name);

        }

    }

    public class IsPhoneNumberNotNullOrWhiteSpace : IBaseValidationRule<Contact>

    {

        public IReadOnlyList<string> Properties { get; } = new List<string>() { nameof(Contact.PhoneNumber) };

        public string ValidationMessage { get; set; }

        public bool Check(Contact value)

        {

            return !string.IsNullOrWhiteSpace(value.PhoneNumber);

        }

    }

    public class IsPhotoPathNotNullOrWhiteSpace : IBaseValidationRule<Contact>

    {

        public IReadOnlyList<string> Properties { get; } = new List<string>() { nameof(Contact.PhotoPath) };

        public string ValidationMessage { get; set; }

        public bool Check(Contact value)

        {

            return !string.IsNullOrWhiteSpace(value.PhotoPath);

        }

    }

}

Результат выполнения программы представлен на рисунке ниже.

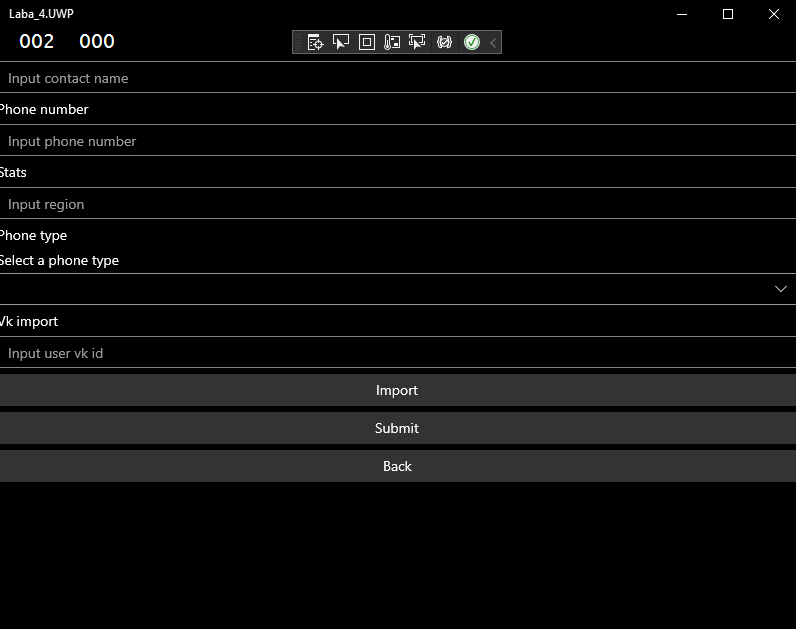


Рисунок 1 – Вход в приложение

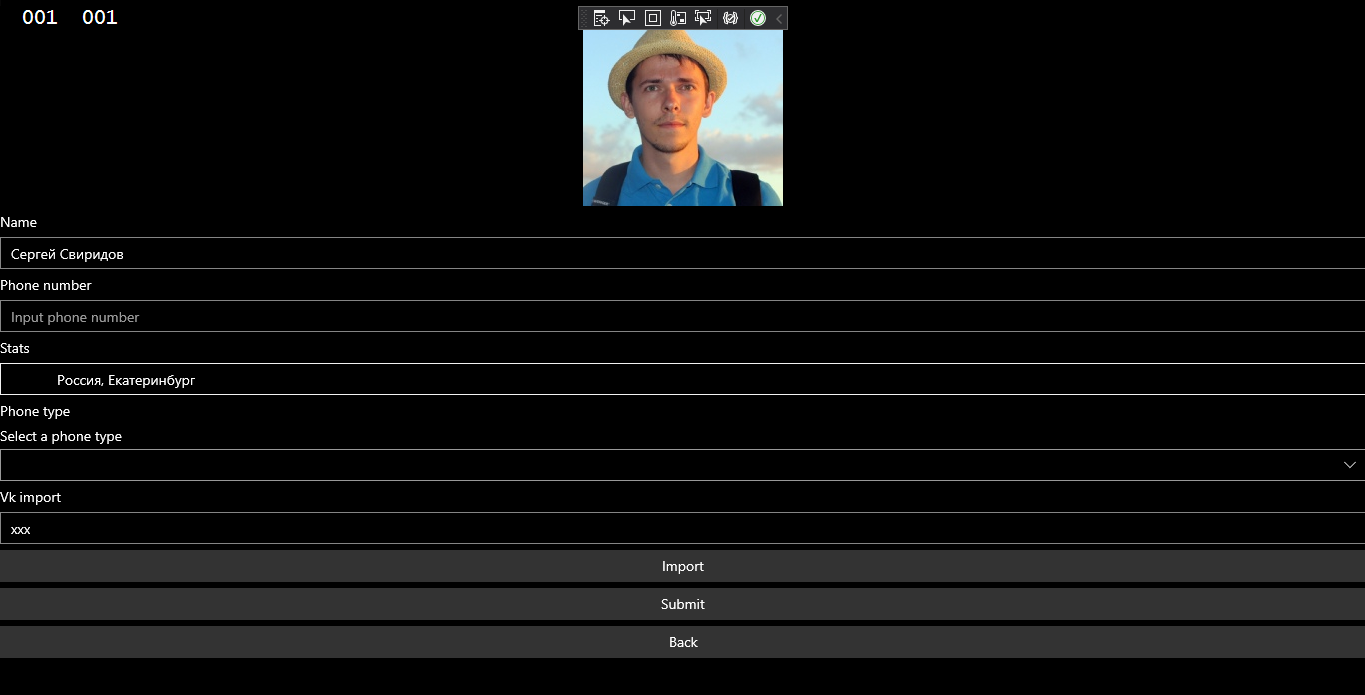


Рисунок 2 – Вывод данных

**Вывод:** в ходе выполнения лабораторной работы мною было изучено мобильное приложения для операционной системы Android, написанное в среде разработки «Xamarin», на языке программирования C#, в ходе разработки была использована облачная база данных «Firebase»: строение, пользование в приложение, принцип работы Firebase. А также закреплены знания прошлых лабораторных работ.