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 №2

Вариант № 6

Выполнил: студент гр. 981064 Ефименко П. В.

Проверил: Калитеня И. Л.

Минск 2021

**Цель:** изучение и создание прототипов мобильных приложений.

**Задание:** разработать мобильное приложение согласно прототипу, представленного ниже на рисунке 1. Реализовать следующий функционал:

1. Получить информацию по выбранной категории из Интернет-каталога (Через RSS, API ресурса или используя парсер)
2. Предусмотреть сохранение работоспобности при отсутствии Интернет-соединения (БД) и вывести сообщение об отсутствии соединения
3. Реализовать пункт О Разработчике, где указать номер группы, фамилию и номер лабораторной
4. Реализовать поиск по имеющимся позициям
5. Страница Профиль должна содержать ФИО, почту и фотографию

Код программы:

using Laba\_2.Services;

using Xamarin.Forms;

namespace Laba\_2

{

public partial class App : Application

{

public new static App Current => (App)Application.Current;

public IContainer Container { get; private set; }

private readonly IConfig \_config;

public App()

{

\_config = ApplicationConfig.Configuration;

Container = new Container(\_config);

InitializeComponent();

MainPage = new AppShell();

}

protected override void OnStart()

{

}

protected override void OnSleep()

{

}

protected override void OnResume()

{

}

}

}

using Laba\_2.Views.Home;

using Laba\_2.Views.Profile;

using Laba\_2.Views.About;

using Xamarin.Forms;

namespace Laba\_2

{

public partial class AppShell : Shell

{

public AppShell()

{

InitializeComponent();

Routing.RegisterRoute(nameof(HomePage), typeof(HomePage));

Routing.RegisterRoute(nameof(ProfilePage), typeof(ProfilePage));

Routing.RegisterRoute(nameof(AboutPage), typeof(AboutPage));

}

}

}

using Xamarin.Forms.Xaml;

[assembly: XamlCompilation(XamlCompilationOptions.Compile)]

namespace Laba\_2.Models

{

public interface IBaseEntity

{

long Id { get; set; }

}

}

namespace Laba\_2.Models

{

public interface IProductItem : IBaseEntity

{

string Name { get; }

string ExtendedName { get; }

double Price { get; }

string ImageUrl { get; }

string NamePrefix { get; }

string Currency { get; }

}

}

using Laba\_2.Services.Helpers;

using System;

namespace Laba\_2.Models

{

internal interface IProfile

{

ObservableObject<string> AvatarUrl { get; set; }

ObservableObject<DateTime> DateOfBirth { get; set; }

ObservableObject<string> FirstName { get; set; }

ObservableObject<string> LastName { get; set; }

ObservableObject<string> PhoneNumber { get; set; }

}

}

namespace Laba\_2.Models

{

public class ProductItem : IProductItem, IBaseEntity

{

public long Id { get; set; }

public string Name { get; set; }

public string ExtendedName { get; set; }

public double Price { get; set; }

public string ImageUrl { get; set; }

public string NamePrefix { get; set; }

public string Currency { get; set; }

}

}

using Laba\_2.Services.Helpers;

using System;

namespace Laba\_2.Models

{

internal class Profile : IProfile

{

public ObservableObject<string> LastName { get; set; }

public ObservableObject<string> FirstName { get; set; }

public ObservableObject<string> AvatarUrl { get; set; }

public ObservableObject<string> PhoneNumber { get; set; }

public ObservableObject<DateTime> DateOfBirth { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.IO;

using System.Reflection;

namespace Laba\_2.Services

{

internal class ApplicationConfig

{

public static IConfig Configuration { get; }

private const string ConfigFilename = "Configs.json";

static ApplicationConfig()

{

try

{

var assembly = typeof(ApplicationConfig).GetTypeInfo().Assembly;

var pathToFile = GetResourcePath(assembly);

using (var stream = assembly.GetManifestResourceStream(pathToFile))

using (var reader = new StreamReader(stream))

{

Configuration = JsonConvert.DeserializeObject<Config>(reader.ReadToEnd());

}

}

catch

{

throw new ArgumentNullException(nameof(ApplicationConfig));

}

}

private static string GetResourcePath(Assembly assembly)

{

var assemblyPrefix = assembly.GetName().Name;

return $"{assemblyPrefix}.{ConfigFilename}";

}

}

}

namespace Laba\_2.Services

{

internal class Config : IConfig

{

public string OnlinerPhonesApiUrl { get; set; }

public string OnlinerTelescopeApiUrl { get; set; }

}

}

using Laba\_2.Services.Network.Api;

using Laba\_2.Services.Network.Request;

using Laba\_2.Services.Repositories;

using Laba\_2.ViewModels;

using Microsoft.Extensions.DependencyInjection;

using System;

namespace Laba\_2.Services

{

public class Container : IContainer

{

public ServiceProvider ServiceProvider { get; private set; }

public IConfig Config { get; private set; }

private readonly ServiceCollection \_services;

public Container(IConfig config)

{

\_services = new ServiceCollection();

Config = config ?? throw new ArgumentNullException(nameof(config));

var liteDbRepositoryFactory = new LiteDbRepositoryFactory("product.db");

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

\_services.AddSingleton<IRepository, MemoryRepository>();

\_services.AddSingleton<IRepository>(repository);

\_services.AddSingleton<IRequestService, RequestService>();

\_services.AddSingleton<IApiService, ApiService>();

\_services.AddSingleton<PhoneService>();

\_services.AddSingleton<TelescopeService>();

\_services.AddSingleton(Config);

\_services.AddTransient<HomeViewModel>();

\_services.AddTransient<ProfileViewModel>();

\_services.AddTransient<AboutViewModel>();

ServiceProvider = \_services.BuildServiceProvider();

}

}

}

namespace Laba\_2.Services

{

public interface IConfig

{

string OnlinerPhonesApiUrl { get; }

string OnlinerTelescopeApiUrl { get; }

}

}

using Microsoft.Extensions.DependencyInjection;

namespace Laba\_2.Services

{

public interface IContainer

{

IConfig Config { get; }

ServiceProvider ServiceProvider { get; }

}

}

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace Laba\_2.Services

{

public interface IDataStore<T>

{

Task<bool> AddItemAsync(T item);

Task<bool> UpdateItemAsync(T item);

Task<bool> DeleteItemAsync(string id);

Task<T> GetItemAsync(string id);

Task<IEnumerable<T>> GetItemsAsync(bool forceRefresh = false);

}

}

using Laba\_2.Services.Network;

using Laba\_2.Services.Network.Response;

using System.Threading;

using System.Threading.Tasks;

namespace Laba\_2.Services

{

internal interface IProductService

{

Task<ProductResponse> FetchAsync(ProgressHandler progressHandler, CancellationToken token);

}

}

namespace Laba\_2.Services

{

public interface IToast

{

void Show(string message);

}

}

using Laba\_2.Services.Network.Api;

using System;

namespace Laba\_2.Services

{

internal class PhoneService : ProductServiceBase, IProductService

{

private readonly IConfig \_config;

public PhoneService(IApiService apiService, IConfig config)

{

\_apiService = apiService ?? throw new ArgumentNullException(nameof(apiService));

\_config = config ?? throw new ArgumentNullException(nameof(config));

}

internal override Uri GenerateUri()

{

var uri = new Uri(\_config.OnlinerPhonesApiUrl, UriKind.Absolute);

return uri;

}

}

}

using Laba\_2.Services.Network;

using Laba\_2.Services.Network.Api;

using Laba\_2.Services.Network.Response;

using System;

using System.Threading;

using System.Threading.Tasks;

namespace Laba\_2.Services

{

internal abstract class ProductServiceBase

{

internal IApiService \_apiService;

public async Task<ProductResponse> FetchAsync(ProgressHandler progressHandler,

CancellationToken token)

{

Uri uri = GenerateUri();

return await \_apiService.GetAsync<ProductResponse>(uri.OriginalString, token, progressHandler);

}

internal abstract Uri GenerateUri();

}

}

using Laba\_2.Services.Network.Api;

using System;

namespace Laba\_2.Services

{

internal class TelescopeService : ProductServiceBase, IProductService

{

private readonly IConfig \_config;

public TelescopeService(IApiService apiService, IConfig config)

{

this.\_apiService = apiService ?? throw new ArgumentNullException(nameof(apiService));

\_config = config ?? throw new ArgumentNullException(nameof(config));

}

internal override Uri GenerateUri()

{

var uri = new Uri(\_config.OnlinerTelescopeApiUrl, UriKind.Absolute);

return uri;

}

}

}

namespace Laba\_2.ViewModels

{

internal class AboutViewModel : BaseViewModel

{

public AboutViewModel()

{

Title = "О Разработчике";

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Runtime.CompilerServices;

namespace Laba\_2.ViewModels

{

internal class BaseViewModel : INotifyPropertyChanged

{

bool isBusy = false;

public bool IsBusy

{

get { return isBusy; }

set { SetProperty(ref isBusy, value); }

}

string title = string.Empty;

public string Title

{

get { return 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;

}

#region INotifyPropertyChanged

public event PropertyChangedEventHandler PropertyChanged;

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

{

var changed = PropertyChanged;

if (changed == null)

return;

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

}

#endregion

}

}

using Laba\_2.Models;

using Laba\_2.Models.OnlinerProduct;

using Laba\_2.Services;

using Laba\_2.Services.Helpers;

using Laba\_2.Services.Network;

using Laba\_2.Services.Repositories;

using MvvmHelpers;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Net.Http;

using System.Threading;

using System.Windows.Input;

using Xamarin.Forms;

namespace Laba\_2.ViewModels

{

internal class HomeViewModel : BaseViewModel

{

public ObservableRangeCollection<IProductItem> Items { get; set; }

public ObservableRangeCollection<IProductItem> AllItems { get; set; }

public ObservableRangeCollection<string> FilterOptions { get; }

public ICommand LoadCommand { get; set; }

private readonly IRepository \_repository;

private readonly PhoneService \_phonesService;

private readonly TelescopeService \_telescopeService;

private readonly CancellationTokenSource \_cancellationToken;

string selectedFilter = "Все";

public string SelectedFilter

{

get => selectedFilter;

set

{

if (SetProperty(ref selectedFilter, value))

FilterItems();

}

}

public ObservableObject<string> SearchFilter { get; set; }

public HomeViewModel(PhoneService phonesService, TelescopeService telescopeService, IRepository repository)

{

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

\_phonesService = phonesService ?? throw new ArgumentNullException(nameof(phonesService));

\_telescopeService = telescopeService ?? throw new ArgumentNullException(nameof(telescopeService));

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

SearchFilter = new ObservableObject<string> { Property = string.Empty };

Items = new ObservableRangeCollection<IProductItem>();

AllItems = new ObservableRangeCollection<IProductItem>();

\_cancellationToken = new CancellationTokenSource();

FilterOptions = new ObservableRangeCollection<string>

{

"Все",

"Телескоп",

"Смартфон"

};

\_repository.InsertCompleted += \_repository\_InsertCompleted;

SearchFilter.PropertyChanged += SearchFilter\_PropertyChanged;

LoadPhonesItems();

LoadTelescopesItems();

\_repository.Add(\_repository.Load<ProductItem>().ToList());

}

private void SearchFilter\_PropertyChanged(object sender, System.ComponentModel.PropertyChangedEventArgs e)

{

FilterItems(SearchFilter.Property);

}

private void FilterItems()

{

Items.ReplaceRange(AllItems.Where(a => a.NamePrefix == SelectedFilter || SelectedFilter == "Все"));

}

private void FilterItems(string searchPattern)

{

Items.ReplaceRange(AllItems.Where(a =>a.NamePrefix == SelectedFilter || SelectedFilter == "Все" && (

a.ExtendedName.ToLower().Contains(searchPattern.ToLower()) ||

a.Name.ToLower().Contains(searchPattern.ToLower())

)));

}

private void OnExecuteLoadCommand()

{

if (IsBusy) return;

IsBusy = true;

try

{

var items = \_repository.GetAll<IProductItem>();

AllItems.ReplaceRange(items);

FilterItems();

}

catch (Exception ex)

{

Debug.WriteLine(ex);

}

finally

{

IsBusy = false;

}

}

private async void LoadPhonesItems()

{

var progressHandler = CreateProgressHandler();

try

{

var phones = await \_phonesService.FetchAsync(progressHandler, \_cancellationToken.Token);

BuildAndSaveProductItems(phones.Products);

}

catch (HttpRequestException ex)

{

Debug.WriteLine(ex);

\_cancellationToken.Cancel();

DependencyService.Get<IToast>().Show("Ошибка подключения!");

}

}

private async void LoadTelescopesItems()

{

var progressHandler = CreateProgressHandler();

try

{

var telescopes = await \_telescopeService.FetchAsync(progressHandler, \_cancellationToken.Token);

BuildAndSaveProductItems(telescopes.Products);

}

catch (HttpRequestException ex)

{

Debug.WriteLine(ex);

\_cancellationToken.Cancel();

DependencyService.Get<IToast>().Show("Ошибка подключения!");

}

}

private void \_repository\_InsertCompleted(object sender)

{

OnExecuteLoadCommand();

}

private void BuildAndSaveProductItems(List<Product> products)

{

List<IProductItem> productItems = new List<IProductItem>();

try

{

products.ForEach(item =>

{

productItems.Add(new ProductItem

{

ExtendedName = item.ExtendedName,

Id = item.Id,

ImageUrl = new UriBuilder("https", item.Images.Header.Remove(0, 2)).Uri.OriginalString.TrimEnd('/'),

Name = item.Name,

Price = Convert.ToDouble(item.Prices.PriceMin.Amount),

NamePrefix = item.NamePrefix.ToString(),

Currency = item.Prices.PriceMin.Currency.ToString()

});

});

}

catch (Exception ex)

{

Debug.WriteLine(ex);

}

finally

{

\_repository.Add(productItems);

}

}

protected ProgressHandler CreateProgressHandler()

{

var progressHandler = new ProgressHandler((size, downloaded) =>

{

if (size.HasValue)

{

var percent = Convert.ToDouble(downloaded) / size.Value \* 100;

}

});

return progressHandler;

}

}

}

using Laba\_2.Models;

using Laba\_2.Services.Helpers;

using Newtonsoft.Json;

using System;

using System.IO;

using System.Threading.Tasks;

using System.Windows.Input;

using Xamarin.Essentials;

using Xamarin.Forms;

namespace Laba\_2.ViewModels

{

internal class ProfileViewModel : BaseViewModel

{

private readonly string PROFILE\_FOLDER = Path.Combine(FileSystem.CacheDirectory, "Profile");

private const string PROFILE\_FILE\_NAME = "profile.json";

public ObservableObject<Profile> Profile { get; set; }

public ICommand PickAvatarCommand { get; set; }

public ICommand SaveProfileCommand { get; set; }

public ProfileViewModel()

{

PickAvatarCommand = new Command(() => OnPickAvatarCommand());

SaveProfileCommand = new Command(() => OnSaveProfileCommand());

Directory.CreateDirectory(PROFILE\_FOLDER);

if (!File.Exists(Path.Combine(PROFILE\_FOLDER, PROFILE\_FILE\_NAME)))

{

File.Create(Path.Combine(PROFILE\_FOLDER, PROFILE\_FILE\_NAME)).Close();

}

Profile = LoadProfile();

}

protected void OnSaveProfileCommand()

{

var jsonString = JsonConvert.SerializeObject(Profile);

File.WriteAllText(Path.Combine(PROFILE\_FOLDER, PROFILE\_FILE\_NAME), jsonString);

}

protected ObservableObject<Profile> LoadProfile()

{

var fileData = File.ReadAllText(Path.Combine(PROFILE\_FOLDER, PROFILE\_FILE\_NAME));

if (string.IsNullOrWhiteSpace(fileData))

{

return new ObservableObject<Profile>

{

Property = new Profile

{

AvatarUrl = new ObservableObject<string> { Property = string.Empty },

FirstName = new ObservableObject<string> { Property = string.Empty },

LastName = new ObservableObject<string> { Property = string.Empty },

PhoneNumber = new ObservableObject<string> { Property = string.Empty },

DateOfBirth = new ObservableObject<DateTime> { Property = DateTime.Now.Date }

}

};

}

else

{

return JsonConvert.DeserializeObject<ObservableObject<Profile>>(fileData);

}

}

protected async void OnPickAvatarCommand()

{

await PickAndShow(PickOptions.Images);

}

protected async Task<FileResult> PickAndShow(PickOptions options)

{

try

{

var result = await FilePicker.PickAsync(options);

if (result != null)

{

if (result.FileName.EndsWith("jpg", StringComparison.OrdinalIgnoreCase) ||

result.FileName.EndsWith("png", StringComparison.OrdinalIgnoreCase))

{

var stream = await result.OpenReadAsync();

var filePath = SaveFileFromStream(result.FileName, stream);

Profile.Property.AvatarUrl.Property = filePath;

}

}

return result;

}

catch (Exception ex)

{

// The user canceled or something went wrong

}

return null;

}

protected string SaveFileFromStream(string fileName, Stream stream)

{

var filePath = Path.Combine(PROFILE\_FOLDER, fileName);

using (var fileStream = new FileStream(filePath, FileMode.Create, FileAccess.Write))

{

stream.CopyTo(fileStream);

}

return filePath;

}

}

}

namespace Laba\_2.Models.OnlinerProduct

{

public enum Currency { Byn, Byr };

}

using System;

using Newtonsoft.Json;

namespace Laba\_2.Models.OnlinerProduct

{

internal class CurrencyConverter : JsonConverter

{

public override bool CanConvert(Type t) => t == typeof(Currency) || t == typeof(Currency?);

public override object ReadJson(JsonReader reader, Type t, object existingValue, JsonSerializer serializer)

{

if (reader.TokenType == JsonToken.Null) return null;

var value = serializer.Deserialize<string>(reader);

switch (value)

{

case "BYN":

return Currency.Byn;

case "BYR":

return Currency.Byr;

}

throw new Exception("Cannot unmarshal type Currency");

}

public override void WriteJson(JsonWriter writer, object untypedValue, JsonSerializer serializer)

{

if (untypedValue == null)

{

serializer.Serialize(writer, null);

return;

}

var value = (Currency)untypedValue;

switch (value)

{

case Currency.Byn:

serializer.Serialize(writer, "BYN");

return;

case Currency.Byr:

serializer.Serialize(writer, "BYR");

return;

}

throw new Exception("Cannot marshal type Currency");

}

public static readonly CurrencyConverter Singleton = new CurrencyConverter();

}

}

using Newtonsoft.Json;

namespace Laba\_2.Models.OnlinerProduct

{

public partial class Images

{

[JsonProperty("header")]

public string Header { get; set; }

[JsonProperty("icon")]

public object Icon { get; set; }

}

}

using System;

namespace Laba\_2.Models.OnlinerProduct

{

public interface IProduct

{

string Description { get; set; }

string ExtendedName { get; set; }

string FullName { get; set; }

Uri HtmlUrl { get; set; }

long Id { get; set; }

Images Images { get; set; }

string Key { get; set; }

string MicroDescription { get; set; }

string Name { get; set; }

NamePrefix NamePrefix { get; set; }

Prices Prices { get; set; }

Sale Sale { get; set; }

Uri Url { get; set; }

}

}

using Newtonsoft.Json;

namespace Laba\_2.Models.OnlinerProduct

{

public partial class MinPricesMedian

{

[JsonProperty("amount")]

public string Amount { get; set; }

[JsonProperty("currency")]

public Currency Currency { get; set; }

}

}

namespace Laba\_2.Models.OnlinerProduct

{

public enum NamePrefix { МобильныйТелефон, Смартфон, Телескоп };

}

using System;

using Newtonsoft.Json;

namespace Laba\_2.Models.OnlinerProduct

{

internal class NamePrefixConverter : JsonConverter

{

public override bool CanConvert(Type t) => t == typeof(NamePrefix) || t == typeof(NamePrefix?);

public override object ReadJson(JsonReader reader, Type t, object existingValue, JsonSerializer serializer)

{

if (reader.TokenType == JsonToken.Null) return null;

var value = serializer.Deserialize<string>(reader);

switch (value)

{

case "Мобильный телефон":

return NamePrefix.МобильныйТелефон;

case "Смартфон":

return NamePrefix.Смартфон;

case "Телескоп":

return NamePrefix.Телескоп;

}

throw new Exception("Cannot unmarshal type NamePrefix");

}

public override void WriteJson(JsonWriter writer, object untypedValue, JsonSerializer serializer)

{

if (untypedValue == null)

{

serializer.Serialize(writer, null);

return;

}

var value = (NamePrefix)untypedValue;

switch (value)

{

case NamePrefix.МобильныйТелефон:

serializer.Serialize(writer, "Мобильный телефон");

return;

case NamePrefix.Смартфон:

serializer.Serialize(writer, "Смартфон");

return;

case NamePrefix.Телескоп:

serializer.Serialize(writer, "Телескоп");

return;

}

throw new Exception("Cannot marshal type NamePrefix");

}

public static readonly NamePrefixConverter Singleton = new NamePrefixConverter();

}

}

using Newtonsoft.Json;

namespace Laba\_2.Models.OnlinerProduct

{

public partial class PriceM

{

[JsonProperty("amount")]

public string Amount { get; set; }

[JsonProperty("currency")]

public Currency Currency { get; set; }

}

}

using Newtonsoft.Json;

namespace Laba\_2.Models.OnlinerProduct

{

public partial class Prices

{

[JsonProperty("price\_min")]

public PriceM PriceMin { get; set; }

[JsonProperty("price\_max")]

public PriceM PriceMax { get; set; }

}

}

using Newtonsoft.Json;

using System;

namespace Laba\_2.Models.OnlinerProduct

{

public partial class Product : IProduct

{

[JsonProperty("id")]

public long Id { get; set; }

[JsonProperty("key")]

public string Key { get; set; }

[JsonProperty("name")]

public string Name { get; set; }

[JsonProperty("full\_name")]

public string FullName { get; set; }

[JsonProperty("name\_prefix")]

public NamePrefix NamePrefix { get; set; }

[JsonProperty("extended\_name")]

public string ExtendedName { get; set; }

[JsonProperty("images")]

public Images Images { get; set; }

[JsonProperty("description")]

public string Description { get; set; }

[JsonProperty("micro\_description")]

public string MicroDescription { get; set; }

[JsonProperty("html\_url")]

public Uri HtmlUrl { get; set; }

[JsonProperty("prices")]

public Prices Prices { get; set; }

[JsonProperty("sale")]

public Sale Sale { get; set; }

[JsonProperty("url")]

public Uri Url { get; set; }

}

}

using Newtonsoft.Json;

namespace Laba\_2.Models.OnlinerProduct

{

public partial class Sale

{

[JsonProperty("is\_on\_sale")]

public bool IsOnSale { get; set; }

[JsonProperty("discount")]

public long Discount { get; set; }

[JsonProperty("min\_prices\_median")]

public MinPricesMedian MinPricesMedian { get; set; }

}

}

using System.ComponentModel;

namespace Laba\_2.Services.Helpers

{

internal 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;

namespace Laba\_2.Services.Network

{

public class ProgressHandler

{

private readonly ProgressChangedDelegate progressChangedDelegate;

public delegate void ProgressChangedDelegate(long? totalStepsCount, long passedStepsCount);

public ProgressHandler(ProgressChangedDelegate progressChangedDelegate)

{

this.progressChangedDelegate = progressChangedDelegate ?? throw new ArgumentNullException(nameof(progressChangedDelegate));

}

public void ProgressChanged(long? totalStepsCount, long passedStepsCount)

{

progressChangedDelegate.Invoke(totalStepsCount, passedStepsCount);

}

}

}

using System;

using System.IO;

using System.Threading;

namespace Laba\_2.Services.Network

{

internal sealed class ProgressStream : Stream

{

private readonly Stream stream;

private readonly long? totalFileSize;

private CancellationToken token;

private long currentPosition = 0;

public delegate void ProgressChangedHandler(long? totalSteps, long passedSteps);

public event ProgressChangedHandler ProgressChanged;

internal ProgressStream(Stream stream, long? totalFileSize, CancellationToken token)

{

this.totalFileSize = totalFileSize;

this.token = token;

this.stream = stream ?? throw new ArgumentNullException(nameof(stream));

}

public override bool CanRead => stream.CanRead;

public override bool CanSeek => stream.CanSeek;

public override bool CanWrite => stream.CanWrite;

public override long Length => stream.Length;

public override long Position

{

get => stream.Position;

set => stream.Position = value;

}

public override void Flush()

{

stream.Flush();

}

public override int Read(byte[] buffer, int offset, int count)

{

var downloadedBytesCount = stream.Read(buffer, offset, count);

token.ThrowIfCancellationRequested();

currentPosition += downloadedBytesCount;

OnProgressChanged(totalFileSize, currentPosition);

return downloadedBytesCount;

}

public override long Seek(long offset, SeekOrigin origin)

{

return stream.Seek(offset, origin);

}

public override void SetLength(long value)

{

stream.SetLength(value);

}

public override void Write(byte[] buffer, int offset, int count)

{

stream.Write(buffer, offset, count);

}

private void OnProgressChanged(long? totalContentSize, long totalBytesLoaded)

{

ProgressChanged?.Invoke(totalContentSize, totalBytesLoaded);

}

}

}

using Laba\_2.Models;

using System.Collections.Generic;

namespace Laba\_2.Services.Repositories

{

public interface IRepository

{

event MemoryRepository.RepositoryMethodContainer InsertCompleted;

T GetById<T>(long id) where T : class, IBaseEntity;

IEnumerable<T> GetAll<T>() where T : class, IBaseEntity;

void Add<T>(IEnumerable<T> entities) where T : class, IBaseEntity;

void Add<T>(T entity) where T : class, IBaseEntity;

void Save();

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

}

}

using Laba\_2.Models;

using Laba\_2.Services.Repositories;

using LiteDB;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Laba\_2.Services.Repositories

{

public class LiteDbRepository : IRepository

{

private readonly LiteRepository \_liteRepository;

public LiteDbRepository(LiteRepository liteRepository)

{

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

}

public T GetById<T>(long id) where T : class, IBaseEntity

{

return \_liteRepository.First<T>(item => item.Id == id);

}

public IEnumerable<T> GetAll<T>() where T : class, IBaseEntity

{

return \_liteRepository.Query<T>().ToEnumerable();

}

public void Add<T>(IEnumerable<T> entities) where T : class, IBaseEntity

{

\_liteRepository.Insert<T>(entities);

}

public void Add<T>(T entity) where T : class, IBaseEntity

{

\_liteRepository.Insert<T>(entity);

}

public void Save()

{

}

public IList<T> Load<T>() where T : class, IBaseEntity

{

return GetAll<T>().ToList();

}

public event MemoryRepository.RepositoryMethodContainer InsertCompleted;

}

}

using LiteDB;

using System;

using System.IO;

namespace Laba\_2.Services.Repositories

{

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(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), DbDir);

Directory.CreateDirectory(dirPath);

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

}

public LiteRepository GetRepository()

{

return new LiteRepository(\_dbFilePath);

}

}

}

using Laba\_2.Models;

using Laba\_2.Services.Network.Api;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace Laba\_2.Services.Repositories

{

public class MemoryRepository : IRepository

{

private readonly List<object> \_dbList;

private readonly string FILE\_PATH = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "db\_sdmp\_laba2.json");

#region Events

public delegate void RepositoryMethodContainer(object sender);

public event RepositoryMethodContainer InsertCompleted;

#endregion

public MemoryRepository()

{

\_dbList = new List<object>();

}

public T GetById<T>(long id) where T : class, IBaseEntity

{

return \_dbList.OfType<T>().ToList().Find(item => item.Id == id);

}

public IEnumerable<T> GetAll<T>() where T : class, IBaseEntity

{

return \_dbList.OfType<T>().ToList();

}

public void Add<T>(IEnumerable<T> entities) where T : class, IBaseEntity

{

entities = AddIndexToEntities(entities);

\_dbList.AddRange(entities);

Save();

OnInsertCompleted();

}

public void Add<T>(T entity) where T : class, IBaseEntity

{

entity = AddIndexToEntity(entity);

\_dbList.Add(entity);

Save();

OnInsertCompleted();

}

public void Save()

{

var jsonString = JsonConvert.SerializeObject(\_dbList, Converter.Settings);

File.WriteAllText(FILE\_PATH, jsonString);

}

public IList<T> Load<T>() where T : class, IBaseEntity

{

var fileData = string.Empty;

if (File.Exists(FILE\_PATH))

{

fileData = File.ReadAllText(FILE\_PATH);

return JsonConvert.DeserializeObject<List<T>>(fileData, Converter.Settings);

} else

{

return new List<T>();

}

}

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

{

var items = entities.ToList();

var currentIndex = \_dbList.Count;

foreach (var item in items)

{

item.Id = currentIndex++;

}

return items;

}

private T AddIndexToEntity<T>(T entity) where T : class, IBaseEntity

{

var currentIndex = \_dbList.Count;

entity.Id = currentIndex++;

return entity;

}

#region protected

protected virtual void OnInsertCompleted()

{

InsertCompleted?.Invoke(this);

}

#endregion

}

}

using Laba\_2.ViewModels;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace Laba\_2.Views.About

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class AboutPage : ContentPage

{

public AboutPage()

{

InitializeComponent();

this.BindingContext = App.Current.Container.ServiceProvider.GetService<AboutViewModel>();

}

}

}

using Laba\_2.ViewModels;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace Laba\_2.Views.Home

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class HomePage : ContentPage

{

public HomePage()

{

InitializeComponent();

this.BindingContext = App.Current.Container.ServiceProvider.GetService<HomeViewModel>();

}

}

}

using Laba\_2.Services.Helpers;

using Laba\_2.ViewModels;

using System.IO;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace Laba\_2.Views.Profile

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class ProfilePage : ContentPage

{

private readonly ProfileViewModel \_viewModel;

public ProfilePage()

{

InitializeComponent();

\_viewModel = App.Current.Container.ServiceProvider.GetService<ProfileViewModel>();

this.BindingContext = \_viewModel;

\_viewModel.Profile.Property.AvatarUrl.PropertyChanged += AvatarUrl\_PropertyChanged;

if (File.Exists(\_viewModel.Profile.Property.AvatarUrl.Property))

{

var stream = File.OpenRead(\_viewModel.Profile.Property.AvatarUrl.Property);

AvatarImage.Source = ImageSource.FromStream(() => stream);

}

}

private void AvatarUrl\_PropertyChanged(object sender, System.ComponentModel.PropertyChangedEventArgs e)

{

var profile = (ObservableObject<string>)sender;

var stream = File.OpenRead(profile.Property);

AvatarImage.Source = ImageSource.FromStream(() => stream);

}

}

}

// <autogenerated />

using System;

using System.Reflection;

[assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETStandard,Version=v2.0", FrameworkDisplayName = "")]

//------------------------------------------------------------------------------

// <auto-generated>

// Этот код создан программой.

// Исполняемая версия:4.0.30319.42000

//

// Изменения в этом файле могут привести к неправильной работе и будут потеряны в случае

// повторной генерации кода.

// </auto-generated>

//------------------------------------------------------------------------------

[assembly: global::Xamarin.Forms.Xaml.XamlResourceIdAttribute("Laba\_2.App.xaml", "App.xaml", typeof(global::Laba\_2.App))]

namespace Laba\_2 {

[global::Xamarin.Forms.Xaml.XamlFilePathAttribute("App.xaml")]

public partial class App : global::Xamarin.Forms.Application {

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]

private void InitializeComponent() {

global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(App));

}

}

}

//------------------------------------------------------------------------------

// <auto-generated>

// Этот код создан программой.

// Исполняемая версия:4.0.30319.42000

//

// Изменения в этом файле могут привести к неправильной работе и будут потеряны в случае

// повторной генерации кода.

// </auto-generated>

//------------------------------------------------------------------------------

[assembly: global::Xamarin.Forms.Xaml.XamlResourceIdAttribute("Laba\_2.AppShell.xaml", "AppShell.xaml", typeof(global::Laba\_2.AppShell))]

namespace Laba\_2 {

[global::Xamarin.Forms.Xaml.XamlFilePathAttribute("AppShell.xaml")]

public partial class AppShell : global::Xamarin.Forms.Shell {

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Xamarin.Forms.Build.Tasks.XamlG", "2.0.0.0")]

private void InitializeComponent() {

global::Xamarin.Forms.Xaml.Extensions.LoadFromXaml(this, typeof(AppShell));

}

}

}

//------------------------------------------------------------------------------

// <auto-generated>

// Этот код создан программой.

// Исполняемая версия:4.0.30319.42000

//

// Изменения в этом файле могут привести к неправильной работе и будут потеряны в случае

// повторной генерации кода.

// </auto-generated>

//------------------------------------------------------------------------------

using System;

using System.Reflection;

[assembly: System.Reflection.AssemblyCompanyAttribute("Laba\_2")]

[assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]

[assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]

[assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]

[assembly: System.Reflection.AssemblyProductAttribute("Laba\_2")]

[assembly: System.Reflection.AssemblyTitleAttribute("Laba\_2")]

[assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]

// Создано классом WriteCodeFragment MSBuild.

using Laba\_2.Services.Network.Response;

using System;

namespace Laba\_2.Services.Network.Api

{

internal class ApiRequestException : Exception

{

private string message;

public ErrorResponse ErrorDto { get; private set; }

public ApiRequestException()

{

}

public ApiRequestException(string message) : base(message)

{

}

public ApiRequestException(string message, ErrorResponse errorDto)

{

this.message = message;

ErrorDto = errorDto;

}

public ApiRequestException(string message, Exception innerException) : base(message, innerException)

{

}

public override string ToString()

{

if (ErrorDto == null) return base.ToString();

return "Request error: \n " +

$"Code: {ErrorDto?.Code} \n";

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Net.Http;

using System.Threading;

using System.Threading.Tasks;

using Laba\_2.Services.Network.Request;

using Laba\_2.Services.Network.Response;

using Newtonsoft.Json;

using Xamarin.Essentials;

namespace Laba\_2.Services.Network.Api

{

internal class ApiService : IApiService

{

private readonly IRequestService \_requestService;

public ApiService(IRequestService requestService)

{

this.\_requestService = requestService ?? throw new ArgumentNullException(nameof(requestService));

}

public async Task<TResponse> GetAsync<TResponse>(

string apiUrl,

CancellationToken token,

ProgressHandler progressHandler = null)

where TResponse : BaseResponse

{

if (apiUrl == null) throw new ArgumentNullException(nameof(apiUrl));

var response = await \_requestService.RequestAsync(() => CreateRequestMessage(apiUrl), 10000, token);

var deserializedResponse = await HandleResponse<TResponse>(response, progressHandler, token);

return deserializedResponse;

}

private HttpRequestMessage CreateRequestMessage(string apiUrl)

{

return RequestBuilder.CreateRequest()

.SetUrl(apiUrl)

.SetHeaders(new Dictionary<string, string>

{

//{AppIdHeader, config.InstallationId},

//{VoucherIdHeader, config.VoucherId},

})

.SetHttpMethod(HttpMethod.Get)

.Build();

}

private async Task<TResponse> HandleResponse<TResponse>(

HttpResponseMessage response,

ProgressHandler progressHandler,

CancellationToken token)

where TResponse : BaseResponse

{

Stream stream = await GetStreamFromResponse(response, progressHandler, token);

TResponse responseObject;

try

{

responseObject = await Task.Run(() => Deserialize<TResponse>(stream), token);

}

catch (OperationCanceledException)

{

throw;

}

catch (Exception e)

{

throw new InvalidOperationException($"Can not deserialize object with type {typeof(TResponse)}", e);

}

if (responseObject?.Error != null)

{

throw new ApiRequestException("Error was encounted while request processing", responseObject.Error);

}

return responseObject;

}

private async Task<Stream> GetStreamFromResponse(HttpResponseMessage response, ProgressHandler progressHandler, CancellationToken token)

{

var stream = await response.Content.ReadAsStreamAsync();

if (progressHandler == null) return stream;

long? contentLength = response.Content.Headers.ContentLength;

var progressStream = new ProgressStream(stream, contentLength, token);

progressStream.ProgressChanged += progressHandler.ProgressChanged;

return progressStream;

}

private TResponse Deserialize<TResponse>(Stream stream)

{

StreamReader reader = new StreamReader(stream);

string json = reader.ReadToEnd();

return JsonConvert.DeserializeObject<TResponse>(json, Converter.Settings);

}

}

}

using Laba\_2.Models.OnlinerProduct;

using Newtonsoft.Json;

using Newtonsoft.Json.Converters;

using System.Globalization;

namespace Laba\_2.Services.Network.Api

{

internal static class Converter

{

public static readonly JsonSerializerSettings Settings = new JsonSerializerSettings

{

MetadataPropertyHandling = MetadataPropertyHandling.Ignore,

DateParseHandling = DateParseHandling.None,

Formatting= Formatting.Indented,

Converters =

{

NamePrefixConverter.Singleton,

CurrencyConverter.Singleton,

new IsoDateTimeConverter { DateTimeStyles = DateTimeStyles.AssumeUniversal }

},

};

}

}

using Laba\_2.Services.Network.Response;

using System.Threading;

using System.Threading.Tasks;

namespace Laba\_2.Services.Network.Api

{

public interface IApiService

{

Task<TResponse> GetAsync<TResponse>(string apiUrl, CancellationToken token, ProgressHandler progressHandler = null)

where TResponse : BaseResponse;

}

}

using System;

using System.Net.Http;

using System.Threading;

using System.Threading.Tasks;

namespace Laba\_2.Services.Network.Request

{

public interface IRequestService

{

Task<HttpResponseMessage> RequestAsync(Func<HttpRequestMessage> requestMessageFactory, int timeoutMillis, CancellationToken token);

}

}

using System;

using System.Collections.Generic;

using System.Net.Http;

namespace Laba\_2.Services.Network.Request

{

internal class RequestBuilder

{

private HttpMethod method;

private string url;

private IDictionary<string, string> headers;

private RequestBuilder() { }

public static RequestBuilder CreateRequest()

{

return new RequestBuilder();

}

public RequestBuilder SetHttpMethod(HttpMethod method)

{

this.method = method ?? throw new ArgumentNullException(nameof(method));

return this;

}

public RequestBuilder SetUrl(string url)

{

this.url = url ?? throw new ArgumentNullException(nameof(url));

return this;

}

public RequestBuilder SetHeaders(IDictionary<string, string> headers)

{

this.headers = headers ?? throw new ArgumentNullException(nameof(headers));

return this;

}

public HttpRequestMessage Build()

{

if (string.IsNullOrEmpty(url)) throw new InvalidOperationException("Url can not be empty");

var requestMessage = new HttpRequestMessage(method ?? HttpMethod.Get, url);

if (headers != null)

{

foreach (var header in headers)

{

var added = requestMessage.Headers.TryAddWithoutValidation(header.Key, header.Value);

if (!added)

{

throw new InvalidOperationException($"Header {header.Key} with value {header.Value} can not be added");

}

}

}

return requestMessage;

}

}

}

using System;

using System.Diagnostics;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Threading;

using System.Threading.Tasks;

namespace Laba\_2.Services.Network.Request

{

internal class RequestService : IRequestService, IDisposable

{

private readonly HttpClient \_client;

public RequestService()

{

\_client = new HttpClient();

\_client.DefaultRequestHeaders.CacheControl = new CacheControlHeaderValue { NoCache = true };

}

public async Task<HttpResponseMessage> RequestAsync(

Func<HttpRequestMessage> requestMessageFactory,

int timeoutMillis,

CancellationToken token)

{

if (requestMessageFactory == null) throw new ArgumentNullException(nameof(requestMessageFactory));

if (timeoutMillis <= 0) throw new ArgumentOutOfRangeException(nameof(timeoutMillis));

var requestMessage = GetRequestMessage(requestMessageFactory);

var url = BuildFullUrl(requestMessage);

requestMessage.RequestUri = url;

var chainedCancellationToken = CreateCancellationToken(timeoutMillis, token);

var sw = Stopwatch.StartNew();

Debug.WriteLine($"{url}..");

var response = await SendRequest(requestMessage, chainedCancellationToken);

Debug.WriteLine($"{url} - completed in {sw.ElapsedMilliseconds} ms");

ThrowIfUnsuccessfull(response);

return response;

}

private CancellationToken CreateCancellationToken(int timeoutMillis, CancellationToken token)

{

var cancellationTokenSource = new CancellationTokenSource(timeoutMillis);

var result = cancellationTokenSource.Token;

if (token != CancellationToken.None)

{

result = CancellationTokenSource.CreateLinkedTokenSource(result, token).Token;

}

return result;

}

private static HttpRequestMessage GetRequestMessage(Func<HttpRequestMessage> requestMessageFactory)

{

var requestMessage = requestMessageFactory.Invoke();

if (requestMessage == null) throw new InvalidOperationException("Request message can not be null");

return requestMessage;

}

private Uri BuildFullUrl(HttpRequestMessage requestMessage)

{

var originalUrl = requestMessage.RequestUri.OriginalString;

if (requestMessage.RequestUri.IsAbsoluteUri)

{

return new Uri(originalUrl);

}

return new Uri(string.Empty);

}

private async Task<HttpResponseMessage> SendRequest(HttpRequestMessage requestMessage, CancellationToken token)

{

try

{

return await \_client.SendAsync(requestMessage, HttpCompletionOption.ResponseHeadersRead, token);

}

catch (HttpRequestException ex)

{

Debug.WriteLine(ex, "Network is unawailable");

throw;

}

catch (TaskCanceledException ex)

{

Debug.WriteLine(ex, "Network connection timeout");

throw;

}

catch (Exception ex)

{

Debug.WriteLine(ex, "Can not perform request");

throw;

}

}

private void ThrowIfUnsuccessfull(HttpResponseMessage response)

{

if (!response.IsSuccessStatusCode)

{

Debug.WriteLine($"Unsuccessfull status code returned: {response.StatusCode}; {response.Content}");

throw new UnsuccessfullRequestException(response);

}

}

public void Dispose()

{

\_client?.Dispose();

}

}

}

using System;

using System.Net.Http;

using System.Runtime.Serialization;

namespace Laba\_2.Services.Network.Request

{

[Serializable]

internal class UnsuccessfullRequestException : Exception

{

private readonly HttpResponseMessage response;

public UnsuccessfullRequestException()

{

}

public UnsuccessfullRequestException(HttpResponseMessage response)

{

this.response = response;

}

public UnsuccessfullRequestException(string message) : base(message)

{

}

public UnsuccessfullRequestException(string message, Exception innerException) : base(message, innerException)

{

}

protected UnsuccessfullRequestException(SerializationInfo info, StreamingContext context) : base(info, context)

{

}

}

}

namespace Laba\_2.Services.Network.Response

{

public class BaseResponse

{

public ErrorResponse Error { get; set; }

}

}

namespace Laba\_2.Services.Network.Response

{

public class ErrorResponse

{

public string Code { get; set; }

}

}

using Laba\_2.Models.OnlinerProduct;

using Newtonsoft.Json;

using System.Collections.Generic;

namespace Laba\_2.Services.Network.Response

{

internal class ProductResponse : BaseResponse

{

[JsonProperty("products")]

public List<Product> Products { get; set; }

[JsonProperty("total")]

public long Total { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace Laba\_2.Services.Network.Api.Extensions

{

public static class UriExtensions

{

public static Uri AddQueryParameter(this Uri uri, string name, string value)

{

if (uri == null) throw new ArgumentNullException(nameof(uri));

if (name == null) throw new ArgumentNullException(nameof(name));

if (value == null) throw new ArgumentNullException(nameof(value));

var originalUrl = uri.OriginalString;

var questionMarkIndex = originalUrl.IndexOf('?');

var query = questionMarkIndex != -1 ? originalUrl.Substring(questionMarkIndex + 1) : string.Empty;

var rawUrl = questionMarkIndex != -1 ? originalUrl.Substring(0, questionMarkIndex) : originalUrl;

var paramsDictionary = GetParamsFromQuery(query);

paramsDictionary.Add(name, value);

var pairedParams = paramsDictionary.Select(keyValuePair =>

{

return $"{Uri.EscapeUriString(keyValuePair.Key)}={Uri.EscapeUriString(keyValuePair.Value)}";

});

string joined = string.Join("&", pairedParams.ToArray());

var isAbsolute = uri.IsAbsoluteUri;

var fullUrl = $"{rawUrl}?{joined}";

return new Uri(fullUrl, isAbsolute ? UriKind.Absolute : UriKind.Relative);

}

private static Dictionary<string, string> GetParamsFromQuery(string query)

{

var pairs = query.Split('&');

return pairs

.Select(o => o.Split('='))

.Where(items => items.Length == 2)

.ToDictionary(pair => Uri.UnescapeDataString(pair[0]),

pair => Uri.UnescapeDataString(pair[1]));

}

}

Результат выполнения программы представлен на рисунке 1,2,3.

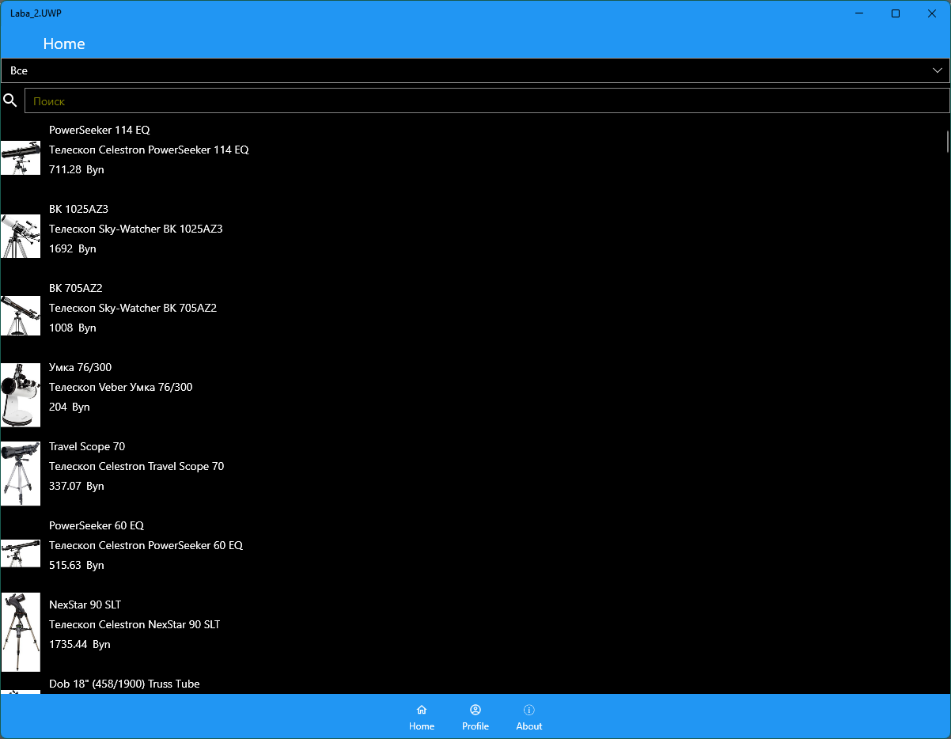


Рисунок 1 – Главное меню

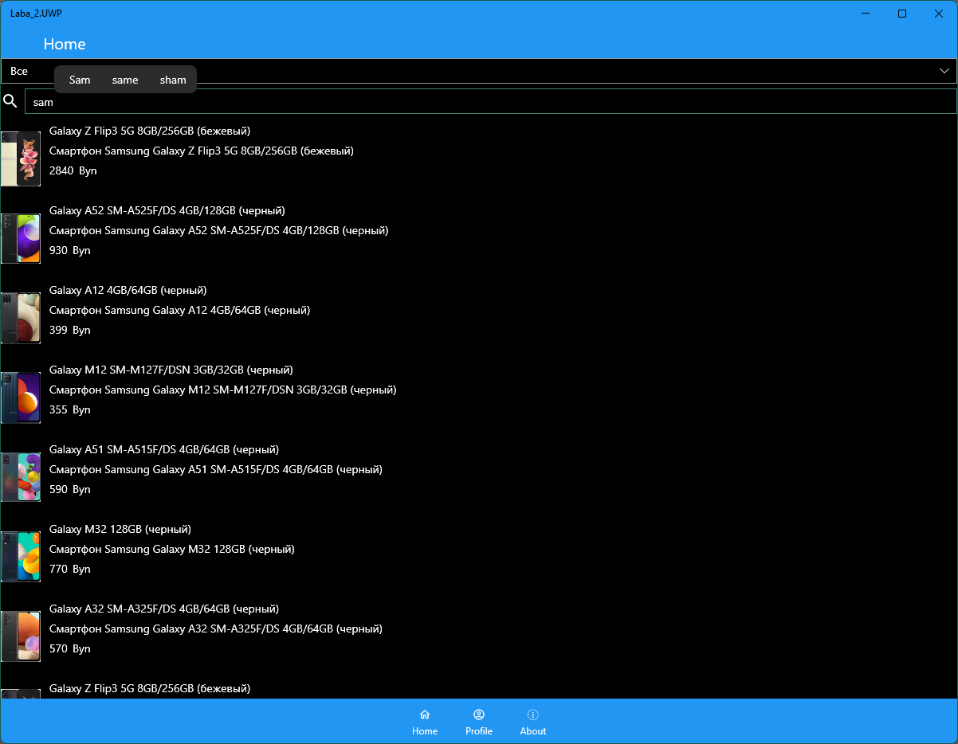


Рисунок 2 – Поиск

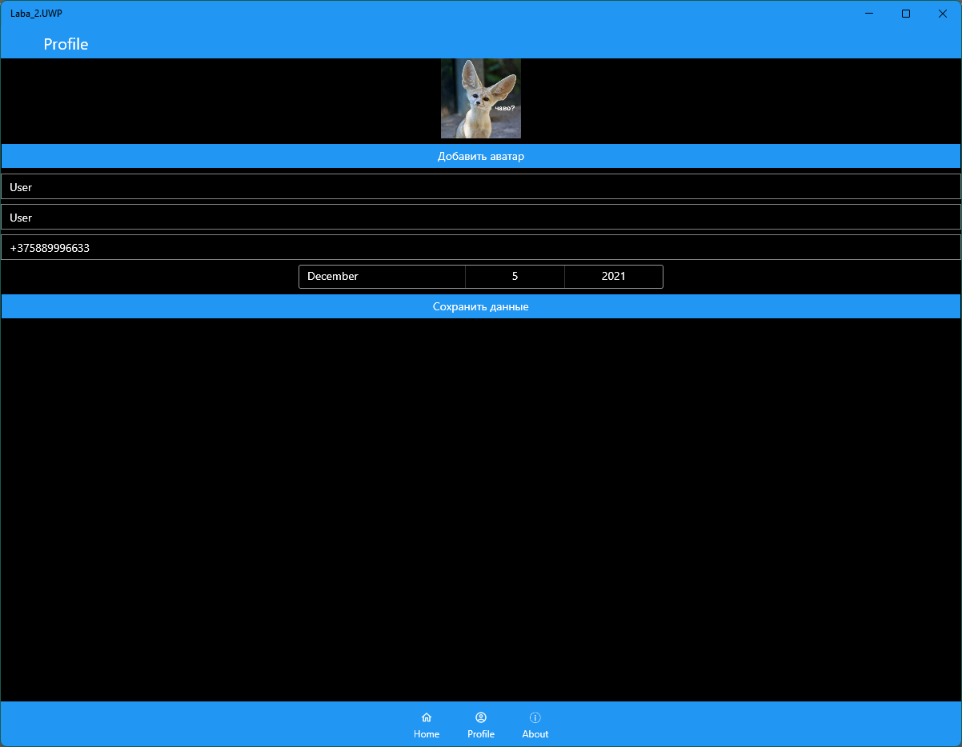


Рисунок 3 – Профиль

**Вывод:** в ходе выполнения лабораторной работы было изучено мобильное приложения для операционной системы Android, написанное в среде разработки «Xamarin», на языке программирования C# в ходе разработки была изучена встроенная СУБД SQLite для хранения данных при отсутствии Интернет-соединения и механизмы для описания одной операции, вывод информации с сайта в мобильные приложения используя парсер.