

ДОДАТОК А

Посилання на Git: <https://github.com/kovass04/SimpleFirstApp/tree/master>

Папка Models(Представлення даних об'єктів)

Account.cs

```
[Table("Accounts")]
public class Account
{
    [PrimaryKey, AutoIncrement, Column("_id")]
    public int Id { get; set; }

    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public string UserName { get; set; }
    public string Password { get; set; }

    [OneToMany(CascadeOperations = CascadeOperation.All)]
    public List<RelatedObject> RelatedObjects { get; set; }
}
[Table("RelatedObject")]
public class RelatedObject
{
    [PrimaryKey, AutoIncrement, Column("Id")]
    public int Id { get; set; }

    public string SomeData { get; set; }

    [ForeignKey(typeof(Account))]
    public string UserId { get; set; }
}
```

DataAssets.cs

```
public class DataAssets
{
    public Assets[] data { get; set; }
    public long timestamp { get; set; }
}

public class Assets
{
    public string id { get; set; }
    public string rank { get; set; }
    public string symbol { get; set; }
    public string name { get; set; }
    public string supply { get; set; }
    public string maxSupply { get; set; }
    public string marketCapUsd { get; set; }
    public string volumeUsd24Hr { get; set; }
    public string priceUsd { get; set; }
    public string changePercent24Hr { get; set; }
    public string vwap24Hr { get; set; }
    public string explorer { get; set; }
}
```

DataAssetsMarkets.cs

```
public class DataAssetsMarkets
{

```

```

    public AssetsMarkets[] data { get; set; }
    public long timestamp { get; set; }
}

public class AssetsMarkets
{
    public string exchangeId { get; set; }
    public string baseId { get; set; }
    public string quoteId { get; set; }
    public string baseSymbol { get; set; }
    public string quoteSymbol { get; set; }
    public string volumeUsd24Hr { get; set; }
    public string priceUsd { get; set; }
    public string volumePercent { get; set; }
}

```

DataCharts.cs

```

public class DataCharts
{
    public Charts[] data { get; set; }
    public long timestamp { get; set; }
}

public class Charts
{
    public string priceUsd { get; set; }
    public long time { get; set; }
    public DateTime date { get; set; }
}

```

DataExchanges.cs

```

public class DataExchanges
{
    public Exchanges[] data { get; set; }
    public long timestamp { get; set; }
}

public class Exchanges
{
    public string exchangeId { get; set; }
    public string name { get; set; }
    public string rank { get; set; }
    public string percentTotalVolume { get; set; }
    public string volumeUsd { get; set; }
    public string tradingPairs { get; set; }
    public bool? socket { get; set; }
    public string exchangeUrl { get; set; }
    public long updated { get; set; }
}

```

DataMarkets.cs

```

public class DataMarkets
{
    public Markets[] data { get; set; }
    public long timestamp { get; set; }
}

public class Markets
{
    public string exchangeId { get; set; }
    public string rank { get; set; }
    public string baseSymbol { get; set; }
    public string baseId { get; set; }
}

```

```

public string quoteSymbol { get; set; }
public string quoteId { get; set; }
public string priceQuote { get; set; }
public string priceUsd { get; set; }
public string volumeUsd24Hr { get; set; }
public string percentExchangeVolume { get; set; }
public string tradesCount24Hr { get; set; }
public long updated { get; set; }
}

```

LoginRequestModel.cs

```

public class LoginRequestModel
{
    public string UserName { get; set; }
    public string Password { get; set; }
}

```

Папка Services(Обработка данных з REST API)

DataServices.cs

```

public class DataServices
{
    private readonly HttpClient _httpClient;

    public DataServices()
    {
        _httpClient = new HttpClient();
        _httpClient.BaseAddress = new Uri("https://api.coincap.io/v2/");
    }

    public async Task<DataAssets> GetAssetsAsync()
    {
        var response = await _httpClient.GetAsync("assets");
        response.EnsureSuccessStatusCode();

        var content = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<DataAssets>(content);
    }

    public async Task<DataAssetsId> GetAssetsIdAsync(string Id)
    {
        var response = await _httpClient.GetAsync($"assets/{Id}");
        response.EnsureSuccessStatusCode();

        var content = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<DataAssetsId>(content);
    }

    public async Task<DataMarkets> GetMarketsAsync()
    {
        var response = await _httpClient.GetAsync("markets");
        response.EnsureSuccessStatusCode();

        var content = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<DataMarkets>(content);
    }

    public async Task<DataExchanges> GetExchangesAsync()
    {
        var response = await _httpClient.GetAsync("exchanges");
        response.EnsureSuccessStatusCode();
    }
}

```

```

        var content = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<DataExchanges>(content);
    }

    public async Task<DataAssetsMarkets> GetMarketsIdAsync(string market)
    {
        var response = await _httpClient.GetAsync($"assets/{market}/markets");
        response.EnsureSuccessStatusCode();

        var content = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<DataAssetsMarkets>(content);
    }

    public async Task<DataCharts> GetChartsAsync(string assetId, string date)
    {
        var response = await
        _httpClient.GetAsync($"assets/{assetId}/history?interval={date}");
        response.EnsureSuccessStatusCode();

        var content = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<DataCharts>(content);
    }
}

```

Папка ViewModel(Обробка даних та зв'язок між моделлю та представленням)

BaseViewModel.cs

```

public class BaseViewModel : INotifyPropertyChanged
{
    bool isBusy = false;
    public bool IsBusy
    {
        get { return isBusy; }
        set { SetProperty(ref isBusy, value); }
    }
    public void OnAppearing()
    {
        IsBusy = true;
    }

    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
    public string firstItem; //fix
    private readonly DataServices _Services;
    private List<Assets> _assets;
    private List<Exchanges> _exchanges;
    private List<AssetsMarkets> _marks;
    private List<Charts> _charts;
    public List<Assets> Asset
    {
        get => _assets;
        set
        {
            SetProperty(ref _assets, value);
        }
    }
    public List<Exchanges> Exchange
    {
        get => _exchanges;
        set
        {
            SetProperty(ref _exchanges, value);
        }
    }
    public List<AssetsMarkets> Market
    {
        get => _marks;
        set
        {
            SetProperty(ref _marks, value);
        }
    }
    public List<Charts> Charts
    {
        get => _charts;
        set
        {
            SetProperty(ref _charts, value);
        }
    }
    protected async Task LoadAssetsAsync()
    {
        var assets = await _Services.GetAssetsAsync();
        Asset = new List<Assets>(assets.data);
    }
    protected async Task LoadAssetsAsync(string id)
    {
        var assets = await _Services.GetAssetsIdAsync(id);
        firstItem = assets.data.priceUsd;
    }
    protected async Task LoadExchangesAsync()
    {
        var exchanges = await _Services.GetExchangesAsync();
        Exchange = exchanges.data.Take(50).ToList();
    }
    //Market start f()

```

```

protected async Task LoadMarketsAsync(string id)
{
    var markets = await _Services.GetMarketsIdAsync(id);
    Market = new List<AssetsMarkets>(markets.data);
}
//Market sort f()
protected async Task LoadMarketsAsync(string id, bool sortDescending)
{
    var markets = await _Services.GetMarketsIdAsync(id);

    if (sortDescending)
    {
        Market = markets.data.OrderByDescending(m => m.priceUsd).ToList();
    }
    else
    {
        Market = markets.data.OrderBy(m => m.priceUsd).ToList();
    }
}
//Market end f()
protected async Task LoadChartsAsync(string i, string b)
{
    var charts = await _Services.GetChartsAsync(i, b);
    Charts = new List<Charts>(charts.data);
}
public BaseViewModel()
{
    _Services = new DataServices();
}
}

```

HomeViewModel.cs

```

public class HomeViewModel : BaseViewModel
{
    public HomeViewModel()
    {
        Title = "Homepage";
        _ = LoadAssetsAsync();
    }
}

```

WelcomePageViewModel.cs

```

public class WelcomePageViewModel : BaseViewModel
{
    public Task Init { get; }

    public WelcomePageViewModel()
    {
        Init = Initialize();
    }
    private async Task Initialize()
    {
        VersionTracking.Track();
        if (VersionTracking.IsFirstLaunchEver)
        {
            await Application.Current.MainPage.Navigation.PushAsync(new
LoginPage1());
        }
        else
        {
            await Application.Current.MainPage.Navigation.PushAsync(new
HomeView());
        }
    }
}

```

```
}  
}
```

ProfileViewModel.cs

```
public class ProfileViewModel  
{  
    #region Commands  
  
    public ICommand BackCommand { get; set; }  
    public ICommand DarkModeToggleCommand { get; set; }  
  
    #endregion  
  
    #region Properties  
  
    public Task Init { get; }  
    public bool IsDarkMode { get; set; }  
  
    #endregion  
  
    #region Constructors  
  
    public ProfileViewModel()  
    {  
        BackCommand = new Command(BackCommandHandler);  
        DarkModeToggleCommand = new Command(DarkModeToggleCommandHandler);  
  
        Init = Initialize();  
    }  
  
    public Task Initialize()  
    {  
        IsDarkMode = Application.Current.UserAppTheme.Equals(OSAppTheme.Dark);  
        return Task.CompletedTask;  
    }  
  
    #endregion  
  
    #region Command Handlers  
  
    private async void BackCommandHandler()  
    {  
        await Shell.Current.Navigation.PushAsync(new HomeView());  
    }  
  
    private void DarkModeToggleCommandHandler()  
    {  
        if (IsDarkMode)  
        {  
            Application.Current.UserAppTheme = OSAppTheme.Dark;  
            Preferences.Set("theme", "dark");  
        }  
        else  
        {  
            Application.Current.UserAppTheme = OSAppTheme.Light;  
            Preferences.Set("theme", "light");  
        }  
    }  
  
    #endregion  
}
```

LoginViewModel.cs

```

public class LoginViewModel : INotifyPropertyChanged
{
    private LoginRequestModel myloginRequestModel = new LoginRequestModel();
    public LoginRequestModel MyloginRequestModel
    {
        get { return myloginRequestModel; }
        set
        {
            myloginRequestModel = value;

            OnPropertyChanged(nameof(MyLoginRequestModel));
        }
    }

    public ICommand LoginCommand { get; }

    public LoginViewModel()
    {
        LoginCommand = new Command(PerformLoginOperation);
    }

    private async void PerformLoginOperation(object obj)
    {
        //Perform API Operation/ DB Operation

        var data = MyloginRequestModel;
        await App.Database.CreateTable();
        var data2 = await App.Database.GetAccountItemsAsync();

        foreach (var item in data2)
        {
            if (data.UserName == item.UserName && data.Password ==
item.Password)
            {
                Preferences.Set("UserAlreadyloggedIn", true);

                Preferences.Set("UserLogin", item.UserName);

                await Application.Current.MainPage.Navigation.PushAsync(new
FavoritePage());
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}

```

FavoritePageViewModel.cs

```

internal class FavoritePageViewModel
{
    public ICommand LogoutCommand { get; }

    public FavoritePageViewModel()
    {
        LogoutCommand = new Command(PerformLogoutOperation);
    }
}

```



```

private async void PerformLogoutOperation(object obj)
{
    Preferences.Clear();

    await Application.Current.MainPage.Navigation.PushAsync(new
LoginPage1());
}
}

```

Папка Views(Відповідає за графічний інтерфейс)

AccountAsyncRepository.cs

```

public class AccountAsyncRepository
{
    SQLiteAsyncConnection database;

    public AccountAsyncRepository(string databasePath)
    {
        database = new SQLiteAsyncConnection(databasePath);
    }

    public async Task CreateTable()
    {
        await database.CreateTableAsync<Account>();
        await database.CreateTableAsync<RelatedObject>();
    }

    public async Task<List<RelatedObject>> GetItemsAsync()функція
    {
        return await database.Table<RelatedObject>().ToListAsync();
    }

    public async Task<int> SaveRelatedObjectItemAsync(RelatedObject item)
    {
        if (item.Id != 0)
        {
            await database.UpdateAsync(item);
            return item.Id;
        }
        else
        {
            return await database.InsertAsync(item);
        }
    }

    public async Task<List<Account>> GetAccountItemsAsync()
    {
        return await database.Table<Account>().ToListAsync();
    }

    public async Task<Account> GetItemAsync(int id)
    {
        return await database.GetAsync<Account>(id);
    }

    public async Task<int> DeleteItemAsync(Account item)
    {
        return await database.DeleteAsync(item);
    }

    public async Task<int> SaveItemAsync(Account item)
    {
        if (item.Id != 0)
        {
            await database.UpdateAsync(item);
            return item.Id;
        }
        else
        {
            return await database.InsertAsync(item);
        }
    }
}

```

```

    }
}
}

```

LoginPage1.cs

```

[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class LoginPage1 : ContentPage
{
    public LoginPage1()
    {
        InitializeComponent();
        BindingContext = new LoginViewModel();
    }
    protected override async void OnAppearing()
    {
        await App.Database.CreateTable();
        base.OnAppearing();
    }
    private async void TapGestureRecognizer_Tapped(object sender, EventArgs e)
    {
        Account account = new Account();
        RegistrationPage accountPage = new RegistrationPage();
        accountPage.BindingContext = account;
        await Navigation.PushAsync(accountPage);
    }

    private void btnSkip_Clicked(object sender, EventArgs e)
    {
        Navigation.PushAsync(new HomeView());
    }
}

```

RegistrationPage.cs

```

[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class RegistrationPage : ContentPage
{
    public RegistrationPage()
    {
        InitializeComponent();
    }

    private async void btnRegister_Clicked(object sender, EventArgs e)
    {
        var account = (Account)BindingContext;
        if (!String.IsNullOrEmpty(account.FirstName))
        {
            await App.Database.SaveItemAsync(account);
        }
        await this.Navigation.PopAsync();
        DisplayAlert("", "registration Successful!", "OK");
    }
}

```

HomeView.cs

```

[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class HomeView : ContentPage
{
    HomeViewModel _viewModel;
    public HomeView()
    {
        InitializeComponent();
    }
}

```

```

        BindingContext = _viewModel = new HomeViewModel();
    }
    private async void ProfileButton_Click(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new ProfileView());
    }

    protected override void OnAppearing()
    {
        base.OnAppearing();
        _viewModel.OnAppearing();
    }
    private async void OnButtonClicked(object sender, EventArgs e)
    {
        var getUserSavedkey =
Xamarin.Essentials.Preferences.Get("UserAlreadyLoggedIn", false);

        if (getUserSavedkey)//must fix
        {
            Account account = new Account();
            accountPage.BindingContext = account;
            await Navigation.PushAsync(accountPage);
        }
        else
        {
            await Navigation.PushAsync(new LoginPage1());
        }
    }

    private List<Assets> _assets; //fix
    private async void OnItemSelected(object sender,
SelectedItemChangedEventArgs e)
    {
        Assets selectedAssetData = (Assets)e.SelectedItem;

        await Navigation.PushAsync(new ChartsView(selectedAssetData.id));
    }

```

MarketsView

```

[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class MarketsView : ContentPage
{
    MarketsViewModel _viewModel;
    public MarketsView()
    {
        InitializeComponent();

        BindingContext = _viewModel = new MarketsViewModel();
    }

    protected override void OnAppearing()
    {
        base.OnAppearing();
        _viewModel.OnAppearing();
    }

    void picker_SelectedIndexChanged(object sender, EventArgs e)
    {
        header.Text = "You choose: " + picker.Items[picker.SelectedIndex];
        BindingContext = _viewModel = new
MarketsViewModel(picker.Items[picker.SelectedIndex]);
    }

```

```
}
void picker_SelectedIndexChanged2(object sender, EventArgs e)
{
    bool a;
    if (picker2.Items[picker2.SelectedIndex] == "Low") { a = false; }
    else { a = true; }
    //need to fix because I have -1 and not null
    if (picker.Items[picker.SelectedIndex] != null)
    {
        BindingContext = _viewModel = new
MarketsViewModel(picker.Items[picker.SelectedIndex], a);
    }
}

private async void OnButtonClicked(object sender, EventArgs e)
{
    string marketId = (string)((Button)sender).CommandParameter;
    await Navigation.PushAsync(new MarketIdView(marketId));
}
}
```