Архитектура desktop-приложений

Часть II

Ранее...

- **Архитектурные решения** - это выбор принципов, которые позволят удовлетворить требования.
- Сопровождаемость
  - Скорость внесения изменений
  - Скорость включения нового человека в разработку
- MVVM
  - Model-мутант
  - ViewModel-монстр
  - View-чудовище

Далее...

- Команда отомстителей
- Чужой против Хищника
- Лабиринт Минотавра
- Щ.И.Т.

Команда отомстителей

- Внедрение зависимостей
- Логирование
- Настройки
- Исключения

Внедрение зависимостей

- стройте граф зависимостей в точке входа в ваше приложение
- передавайте зависимости через конструктор
- не используйте IoC-контейнеры
- если используете IoC-контейнеры, то используйте Unity
- если используете Autofac, то оставьте в README.md объяснение ваших мотивов, ваш адрес и контактную информацию

Логирование

- логирование бизнес логики
  - ошибки
    - сообщение об ошибки
    - контекст ошибки
  - расследование
    - входные параметры
    - точки принятия решения
    - выходные параметры
- трекинг действий пользователя
  - какие функции не используются
  - что использует чаще/реже

Логирование - это аспект Model

- ILogger - это зависимость
- не забывайте про настройку библиотек для логирования: ротация, архивирование
- отдавайте предпочтение логирующим обёрткам

```
public interface IAccount
{
    IReadOnlyList<IUser> Users();
}

public sealed class LoggedAccount: IAccount
{
    private readonly IAccount _origin;
    private readonly ILogger _logger;

    public LoggedAccount(IAccount origin, ILogger logger)
    {
        _origin = origin;
        _logger = logger;
    }

    public IReadOnlyList<IUser> Users()
    {
        _logger.Trace("User requested");
        var result = _origin.Users();
        _logger.Trace($"Return {result.Count} users");
        return result;
    }
}
```

Трекинг действий пользователя - это аспект View

- используйте механизм Behavior

```
<UserControl
    ...
    xmlns:behavior="clr-namespace:SlidesExample.Behavior"
    xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
    xmlns:local="clr-namespace:SlidesExample">
    ...
    <StackPanel>
        <TextBox Text="{Binding Path=AccountName}" />
        <ListBox ItemsSource="{Binding Path=UserPresentations}" />
        <Button
            Command="{Binding Path=Load}"
            Content="Load">
            <i:Interaction.Behaviors>
                <behavior:TrackBehavior />
            </i:Interaction.Behaviors>
        </Button>
    </StackPanel>
</UserControl>
```

```
public sealed class TrackBehavior : Behavior<Control>
{
    private readonly ILogger _logger;
    public TrackBehavior() {
        _logger = (ILogger) Application.Current.Resources["TrackWriter"];
    }
    protected override void OnAttached() {
        if (AssociatedObject is Button button) {
            button.Click += _track;
        }
    }
    protected override void OnDetaching() {
        if (AssociatedObject is Button button) {
            button.Click -= _track;
        }
    }
    private void _track(object sender, RoutedEventArgs e) {
        _logger?.Trace($"{e.RoutedEvent.Name} {(e.Source as Control)?.Name} {(e.Source as Control)?.Tag}");
    }
}
```

Настройки

Виды

- Настройки работы/запуска приложения
- Настройки пользовательского интерфейса
- Предыдущее состояние (последний выбранный путь, расположение окон, выбранные вкладки)

Где хранить

- конфигурационные файлы (ConfigurationManager),
- встроенные базы данных (SQLite, LiteDB)
- переменные окружения
- реестр
- удаленные источники (Database, key-value storage, web service)

Настройки работы/запуска приложения

- аспект Model и ViewModel
- зависимость для фабрик

Примеры:

- тип используемой БД
- параметры подключения
- лицензии

```
public sealed class AccountConfiguration {
    public Storage ActiveStorage { get; set; }
    public string Connection { get; set; }
    public void Load() {...}
    public void Save() {...}
    public enum Storage { SqlServer, Oracle }
}

public static class AccountBuilder
{
    public static IAccount Build(AccountConfiguration config)
    {
        switch (config.ActiveStorage)
        {
            case AccountConfiguration.Storage.SqlServer:
                return new SqlServerAccount(config.Connection);
            case AccountConfiguration.Storage.Oracle:
                return new OracleAccount(config.Connection);
            default:
                throw new ArgumentException(nameof(config.ActiveStorage));
        }
    }
}
```

```
public partial class App : Application
{
    protected override void OnStartup(StartupEventArgs e)
    {
        var config = new AccountConfiguration();
        config.Load();
        var account = AccountBuilder.Build(config);
        account = new ValidatedAccount(account);
        var vm = new AccountViewModel(account);
        var view = new AccountView {DataContext = vm};
        MainWindow = new MainWindow {Content = view};
        MainWindow.Show();
    }
}
```

Настройки пользовательского интерфейса

- Сложное или уникальное поведение - наследоваться
- Простое и переиспользуемое поведение - Behavior

```
public sealed class LoginViewSettings {
    public const string LoginViewSettingsKey = "LoginViewSettings";
    public bool AllowFacebookLogin { get; set; }
    public bool AllowTwitterLogin { get; set; }
}
```

```
<Application ...>
    <Application.Resources>
        <login:LoginViewSettings
            x:Key="{x:Static login:LoginViewSettings.LoginViewSettingsKey}"
            AllowFacebookLogin="False"
            AllowTwitterLogin="True" />
    </Application.Resources>
</Application>
```

```
public partial class LoginButton : Button {
    public LoginButton() {
        InitializeComponent();
        var settings = (LoginViewSettings)Application.Current.Resources[LoginViewSettings.LoginViewSettingsKey];
        FacebookButton.Visibility = settings.AllowFacebookLogin ? Visibility.Visible : Visibility.Collapsed;
        TwitterButton.Visibility = settings.AllowTwitterLogin ? Visibility.Visible : Visibility.Collapsed;
    }
}
```

Предыдущее состояние

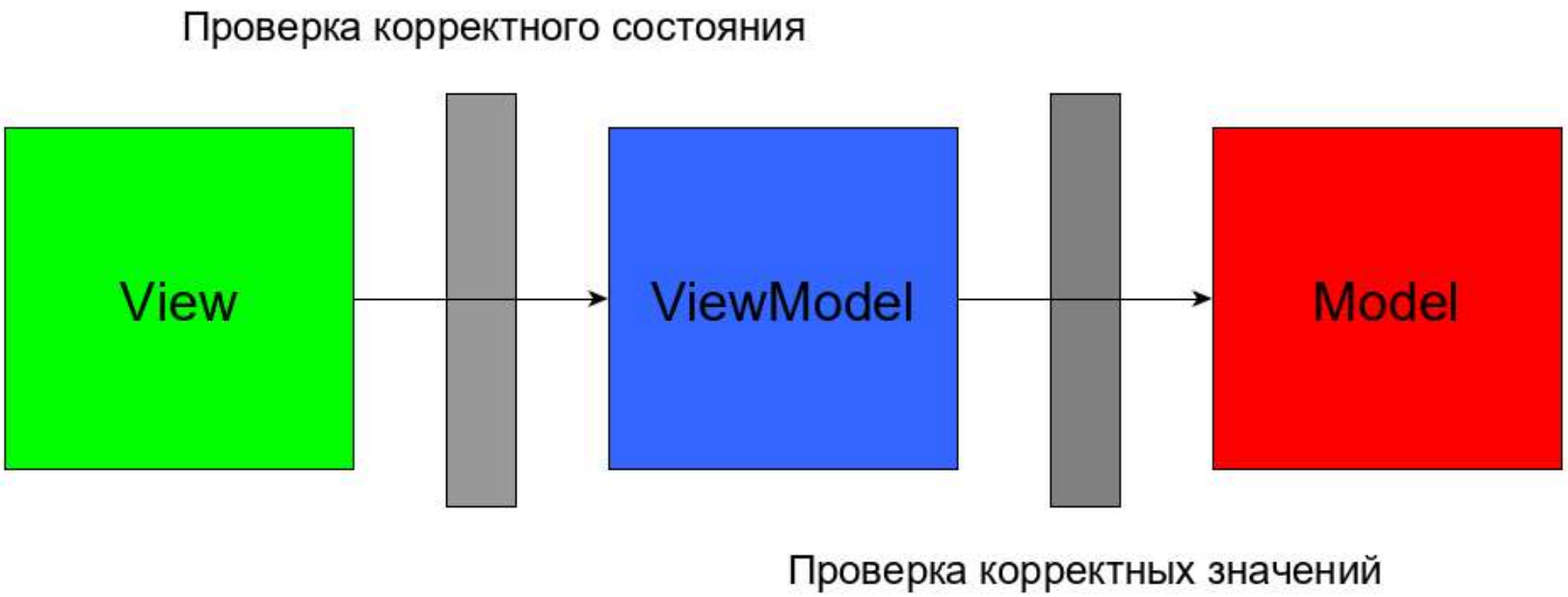
```
public class WindowStateBehavior: Behavior<Window> {
    protected override void OnAttached() {
        if (AssociatedObject is Window window) {
            window.Loaded += WindowOnLoaded;
            window.Closing += WindowOnClosing;
        }
    }
    private void WindowOnClosing(object sender, CancelEventArgs e) {
        var window = (Window) sender;
        Properties.Settings.Default.WindowHeight = window.Height;
        Properties.Settings.Default.WindowWidth = window.Width;
        Properties.Settings.Default.Save();
    }
    private void WindowOnLoaded(object sender, RoutedEventArgs e) {
        var window = (Window) sender;
        window.Height = Properties.Settings.Default.WindowHeight;
        window.Width = Properties.Settings.Default.WindowWidth;
    }
    protected override void OnDetaching() {
        if (AssociatedObject is Window window)
        {
            window.Loaded -= WindowOnLoaded;
            window.Closing -= WindowOnClosing;
        }
    }
}
```

```
<Window
    x:Class="MeetupManager.UI.Main.MainWindow"
    ...
    xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"    >
    <i:Interaction.Behaviors>
        <behavior:WindowStateBehavior />
    </i:Interaction.Behaviors>
    ...
</Window>
```

Исключения

- контуры безопасности
- защитное программирование
- собственные исключения
- документированные исключения

Контуры безопасности



```
public class InvalidUserException : ApplicationException     {
    public InvalidUserException(IUser user) {...}
}
public interface IAccount {
    /// <summary>
    /// Add user to account
    /// </summary>
    /// <param name="user"></param>
    /// <returns>Return true if user was added successfully.</returns>
    /// <exception cref="ValidatedAccount">Throw this exception if invalid or null user was passed.</exception>
    bool Add(IUser user);
}
public sealed class ValidatedAccount: IAccount
{
    private readonly IAccount _origin;
    public ValidatedAccount(IAccount origin)
    {
        _origin = origin;
    }
    public bool Add(IUser user)
    {
        if (user == null || !user.IsValid()) throw new InvalidUserException(user);
        return _origin.Add(user);
    }
}
```

Чужой против Хищника

UI Thread vs асинхронные операции

- **Task-based Asynchronous Pattern (TAP)** - *Task, async/await*
- **Event-based Asynchronous Pattern (EAP)** - *event-based legacy model*
- **Asynchronous Programming Model** - *legacy model that uses the IAsyncResult interface*

Давным давно...

- Dispatcher
- BackgroundWorker

Dispatcher

```
private void _doSomeThing()
{
    // do dome thing
    var result = 100;
    //return result in UI
    if (Dispatcher.CheckAccess())
    {
        Dispatcher.InvokeAsync(() => ProgressBar.Value = 100);
    }
}
...
System.Threading.ThreadPool.QueueUserWorkItem(state => _doSomeThing());
...
Task.Run(_doSomeThing);
```

BackgroundWorker

```
var worker = new BackgroundWorker
{
    WorkerReportsProgress = true,
    WorkerSupportsCancellation = true
};
worker.DoWork += (sender, args) =>
{
    worker.ReportProgress(0);
    // do some thing
    worker.ReportProgress(50);
    // do seme thing more
    worker.ReportProgress(100);
};
worker.ProgressChanged += (sender, args) => {
    ProgressBar.Value = args.ProgressPercentage;
};
worker.RunWorkerAsync();
```

Сейчас...

- заражение async/await вверх по стеку вызовов
- промежуточные коллекции перед обновлением ObservableCollection
- или AsyncObservableCollection, с передачей Dispatcher
- промежуточные значения перед блокировкам
- прогресс???

Синхронное API - асинхронный вызов

```
public sealed class SendNotificationCommand : ICommand {
    private readonly INotification _notification;
    private readonly IPersistedNotification _persisted;
    private readonly IErrorReport _report;
    private readonly ViewModel _vm;
    public SendNotificationCommand(ISmsNotification notification, IPersistedNotification persisted, IErrorReport report, ViewModel vm) {
        _notification = notification;
        _persisted = persisted;
        _report = report;
        _vm = vm;
    }
    public async void Execute(object parameter) {
        if (parameter is MessagePresenter presenter) {
            _vm.IsBusy = true;
            try {
                await Task.Run(() => {
                    try {
                        if (_notification.Send(presenter.Message)) {
                            _persisted.Save(presenter.Message);
                        }
                    }
                    catch (Exception e) {
                        _report.Report(e);
                    }
                });
            } finally {
                _vm.IsBusy = false;
            }
        }
    }
}
```

В промежутках

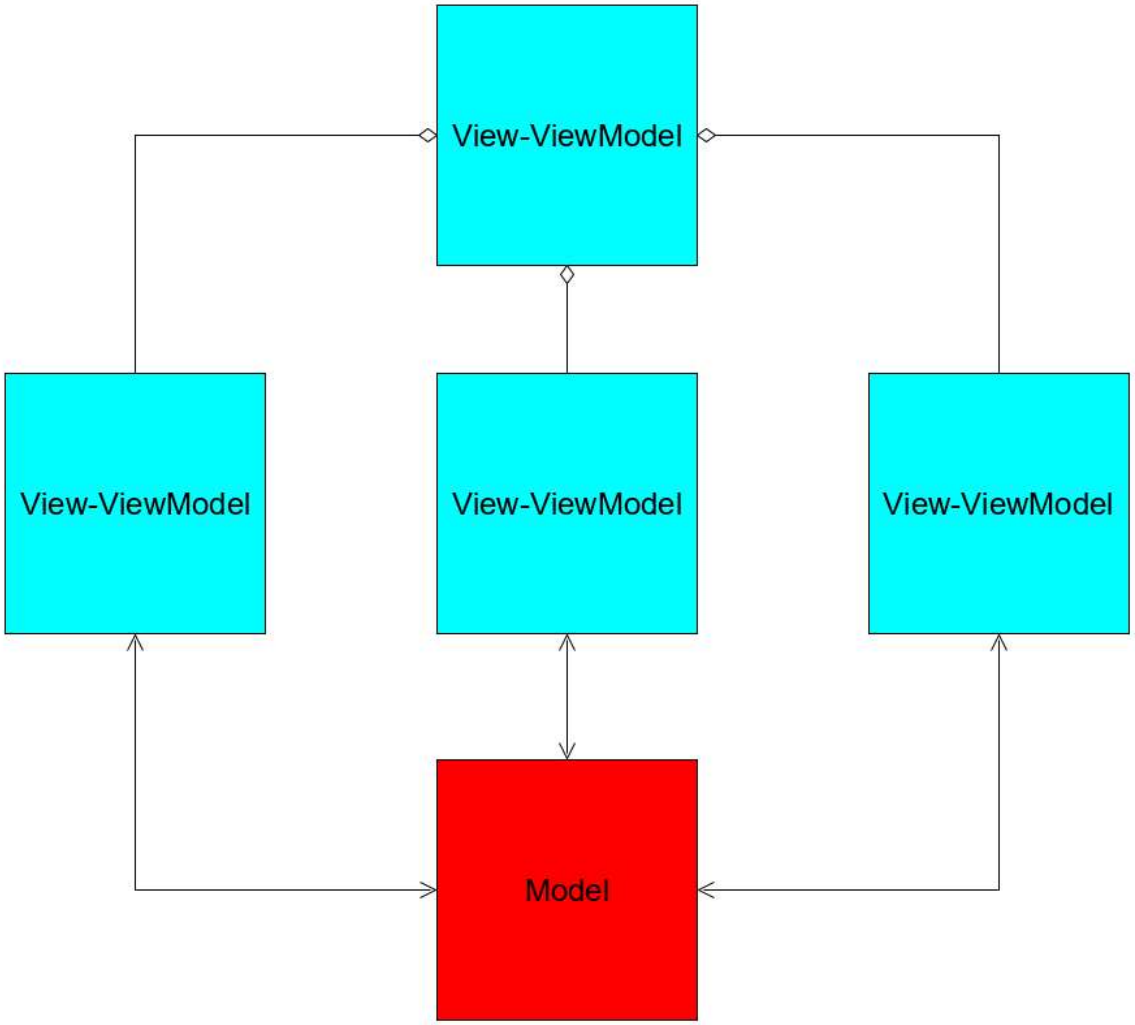
```
public sealed class UserPresentations : ObservableCollection<UserPresentation>
{
    private readonly IAccount _account;
    private readonly object _sync = new object();
    public UserPresentations(IAccount account)
    {
        _account = account;
    }

    public async Task LoadAsync()
    {
        var users = await Task.Run(() => _account.Users());

        lock (_sync)
        {
            foreach (var user in users)
            {
                Add(new UserPresentation(user));
            }
        }
    }
}
```

Прогресс

```
public interface IProgressView {
    int Value { get; set; }
}
public sealed class ProgressReport : System.IProgress<int> {
    private readonly Dispatcher _dispatcher;
    private readonly IProgressView _view;

    public ProgressReport(Dispatcher dispatcher, IProgressView view) {
        _dispatcher = dispatcher;
        _view = view;
    }

    public void Report(int value) {
        if (_dispatcher.CheckAccess())
        {
            _dispatcher.InvokeAsync(() => _view.Value = value);
        }
        else
        {
            _view.Value = value;
        }
    }
}
```

Лабиринт Минотавра

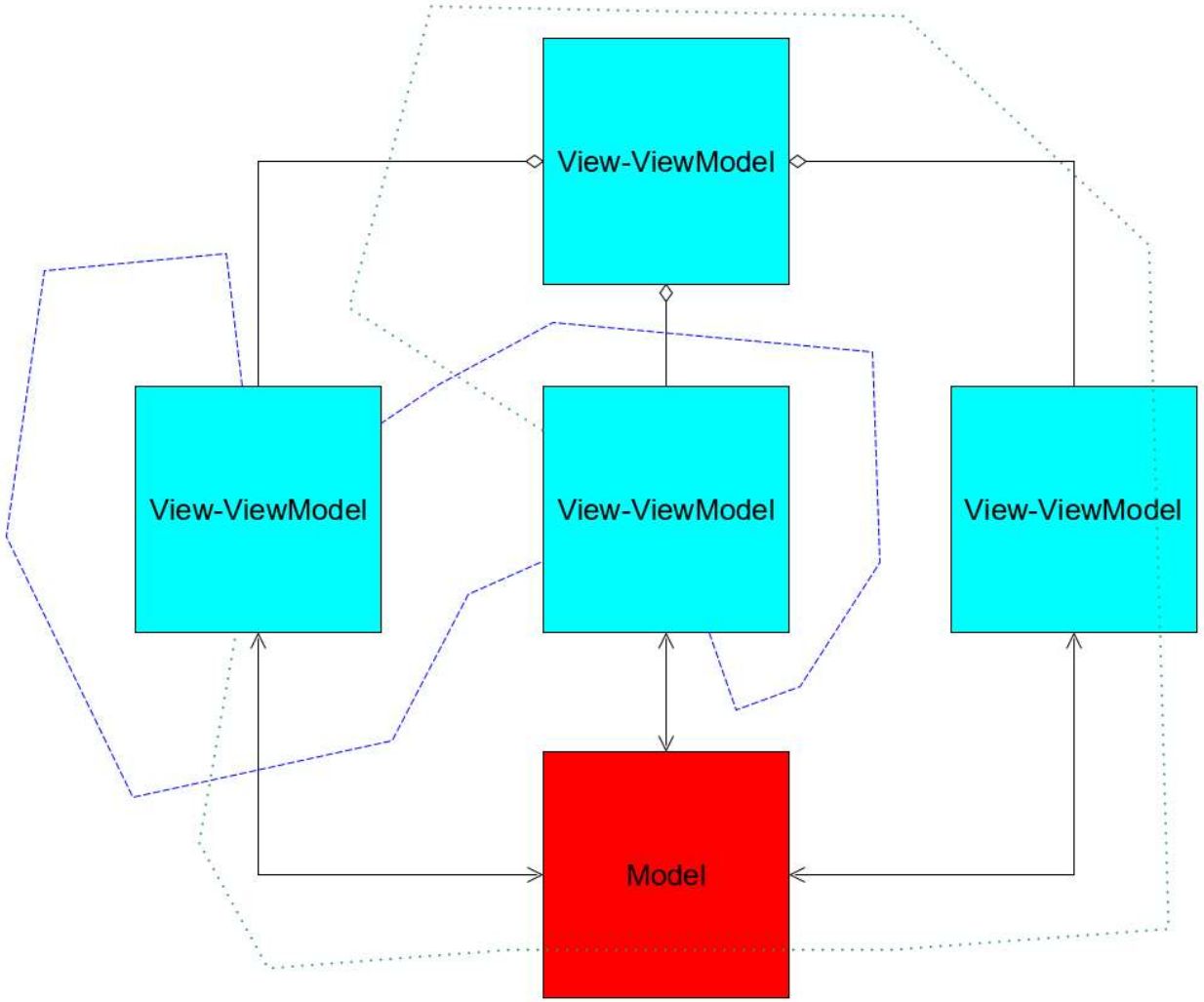
Композиция


View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
Model

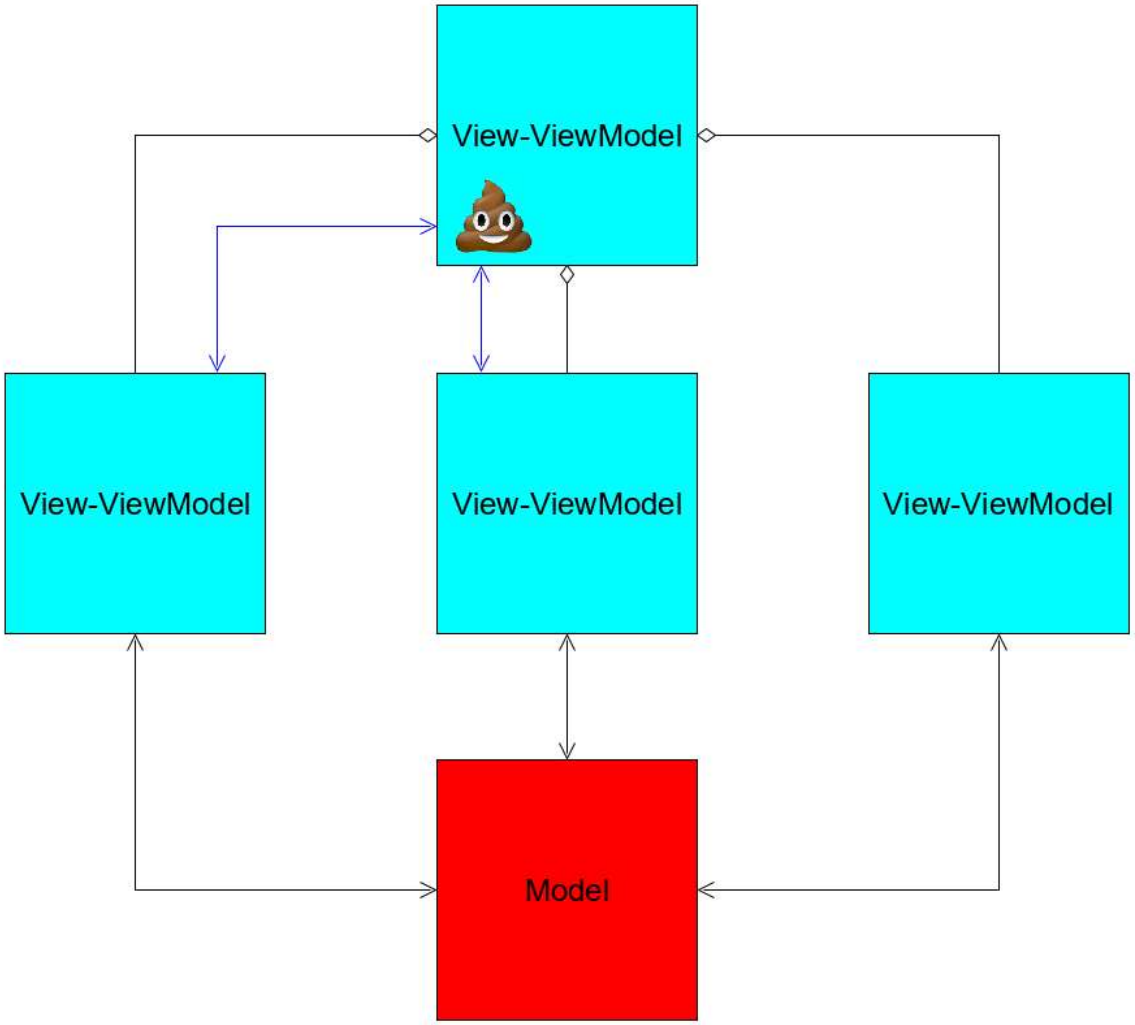
```
public sealed class ApplicationViewModel
{
    public ApplicationViewModel(IAccount account, INotification notification, IReport report)
    {
        AccountViewModel = new AccountViewModel(account);
        NotificationViewModel = new NotificationViewModel(notification);
        ReportViewModel = new ReportViewModel(report);
    }

    public AccountViewModel AccountViewModel { get; }
    public NotificationViewModel NotificationViewModel { get; }
    public ReportViewModel ReportViewModel { get; }
}
```

```
<Window ...>
    <TabControl>
        <TabItem>
            <local:AccountView DataContext="{Binding Path=AccountViewModel}" />
        </TabItem>
        <TabItem>
            <local:NotificationView DataContext="{Binding Path=NotificationViewModel}" />
        </TabItem>
        <TabItem>
            <local:ReportView DataContext="{Binding Path=ReportViewModel}" />
        </TabItem>
    </TabControl>
</Window>
```

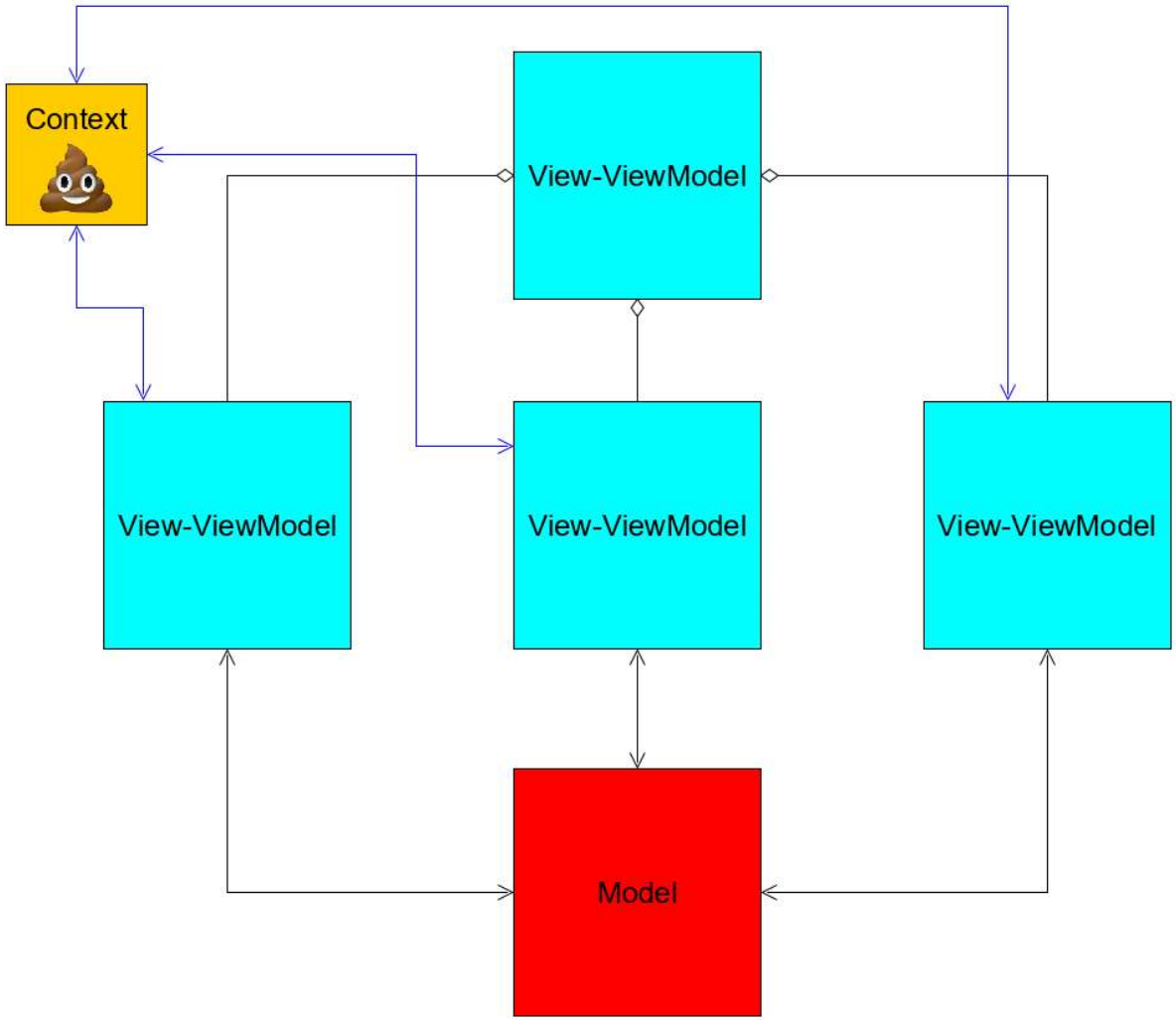
Внезапные повороты сюжета



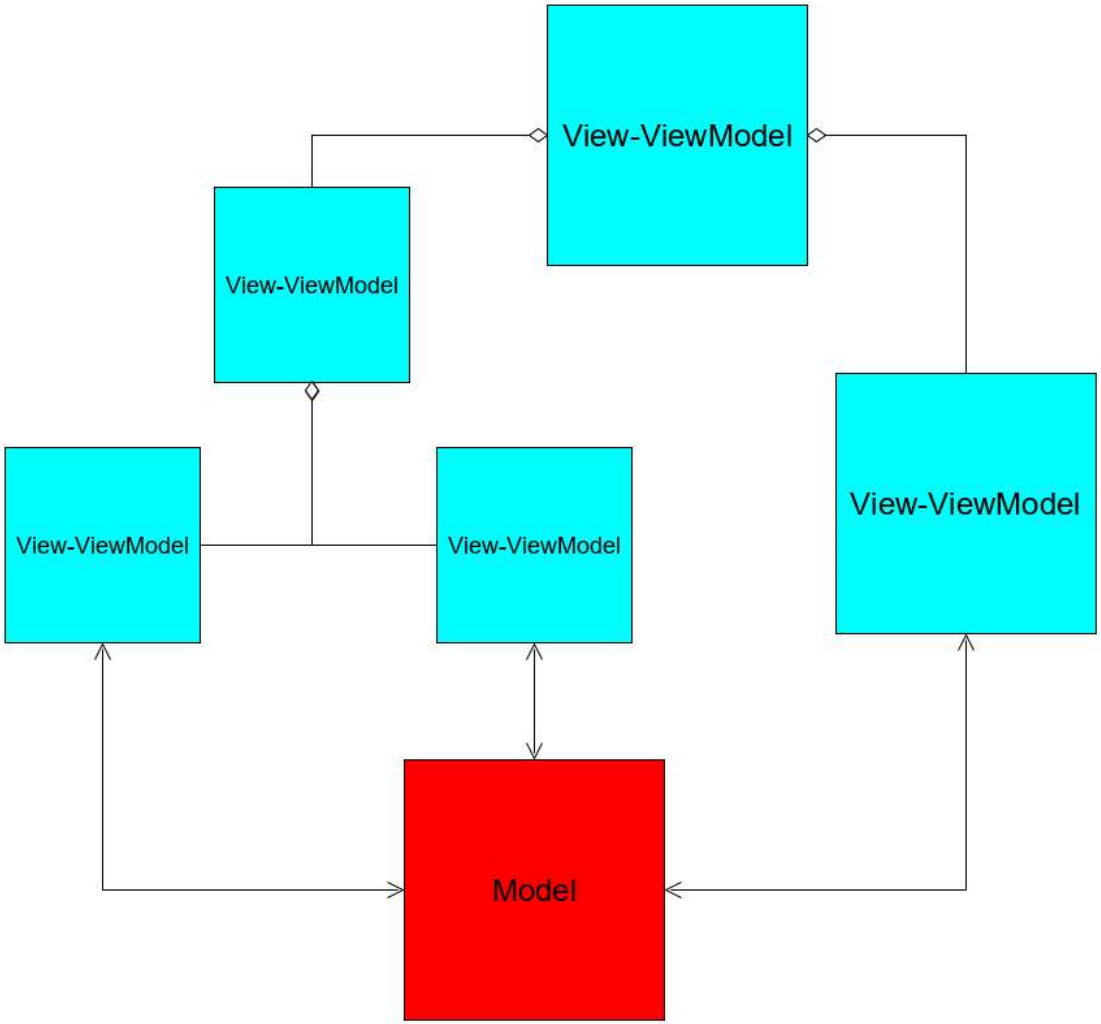
Соберём в одном месте



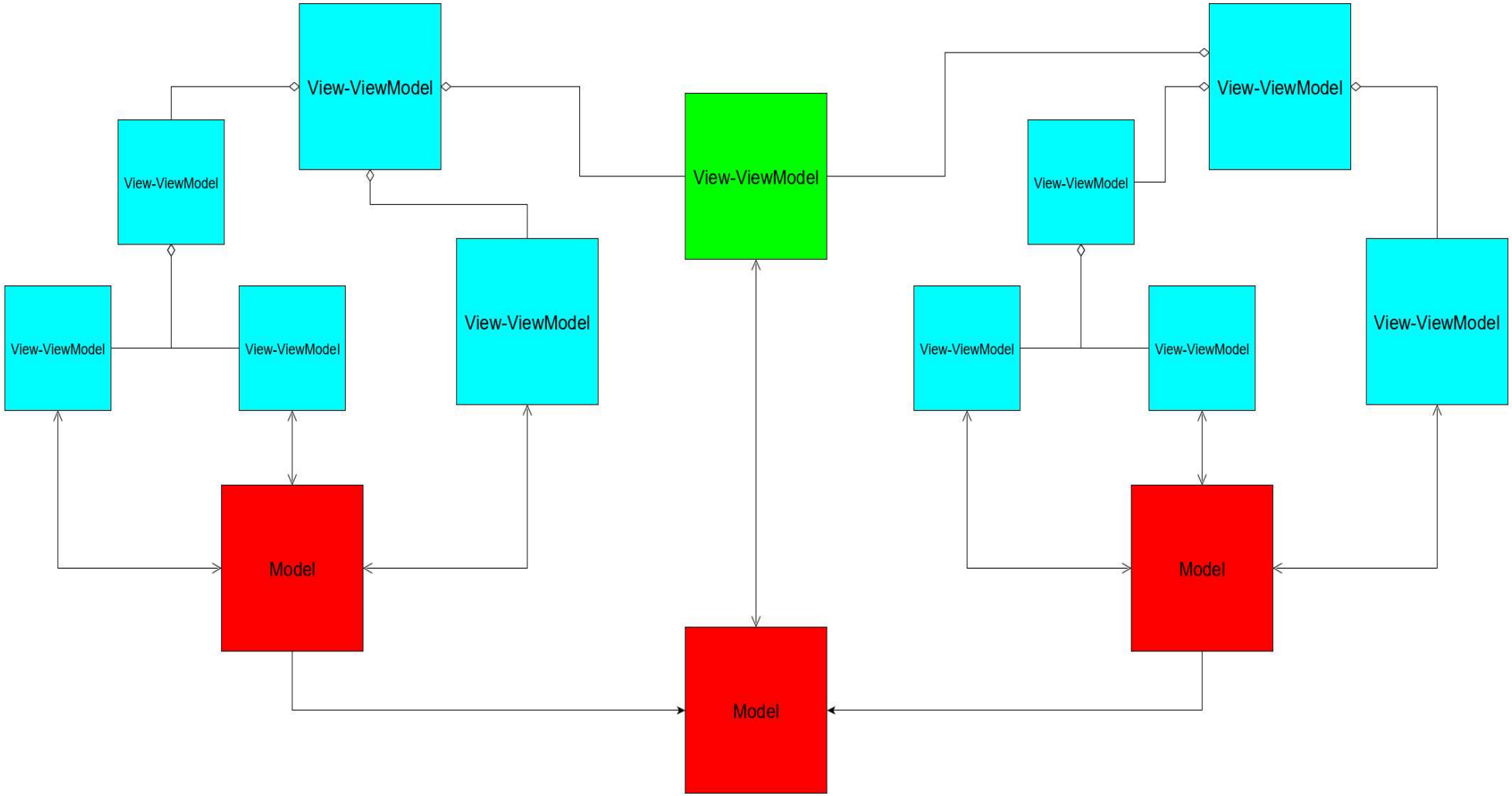
Соберем в одном отдельном месте



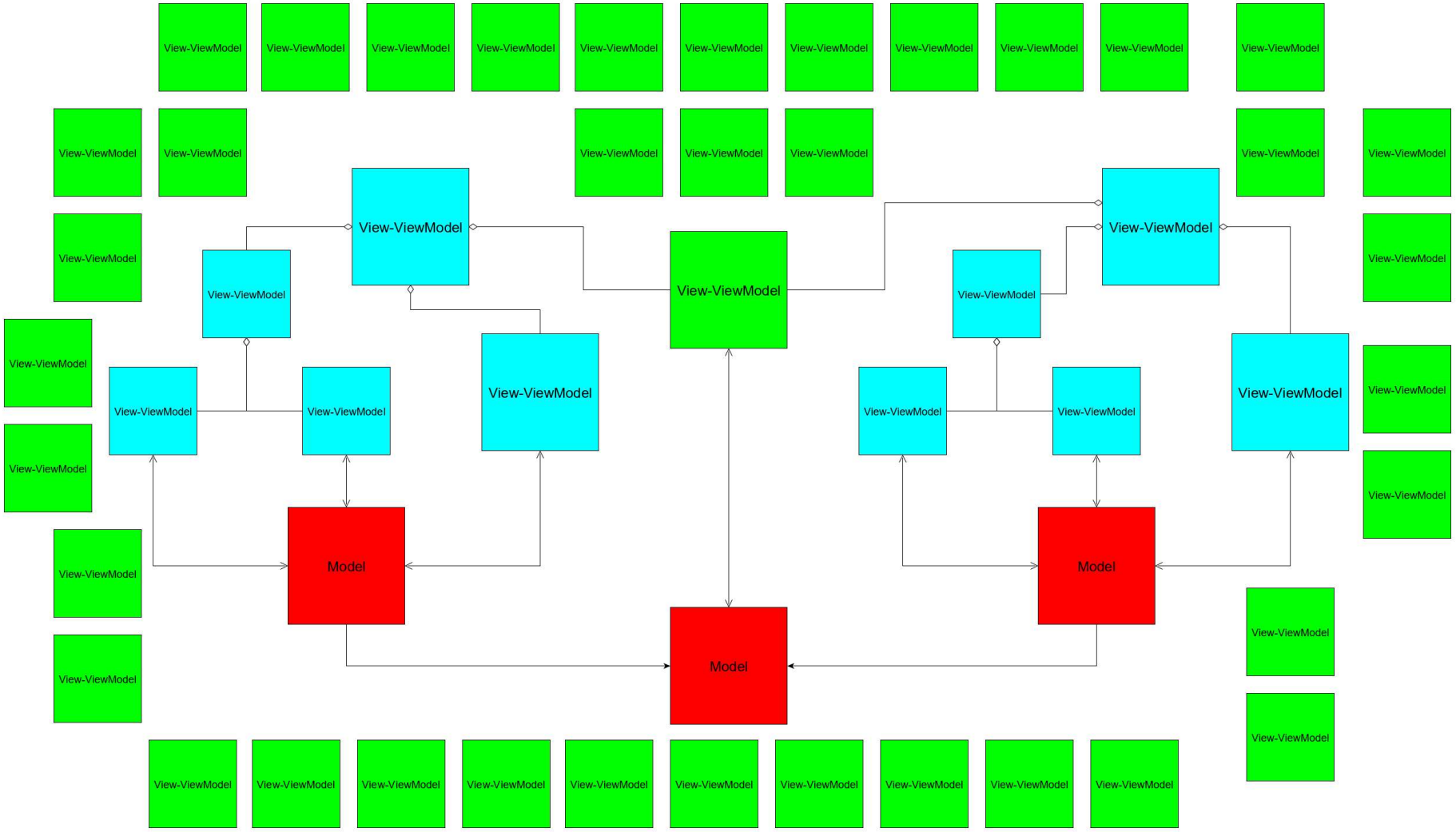
Подумаем ещё раз



У нас есть "Платформа"



Всем по модулю за мой счет



Модуль это -

- независимый элемент (логическая структура)
- отвечающий за одну функциональность (функциональная структура)
- повторно используемый

У модуля есть:

- версионность
- CI
- документация
- покрытие тестами
- свой issue-track

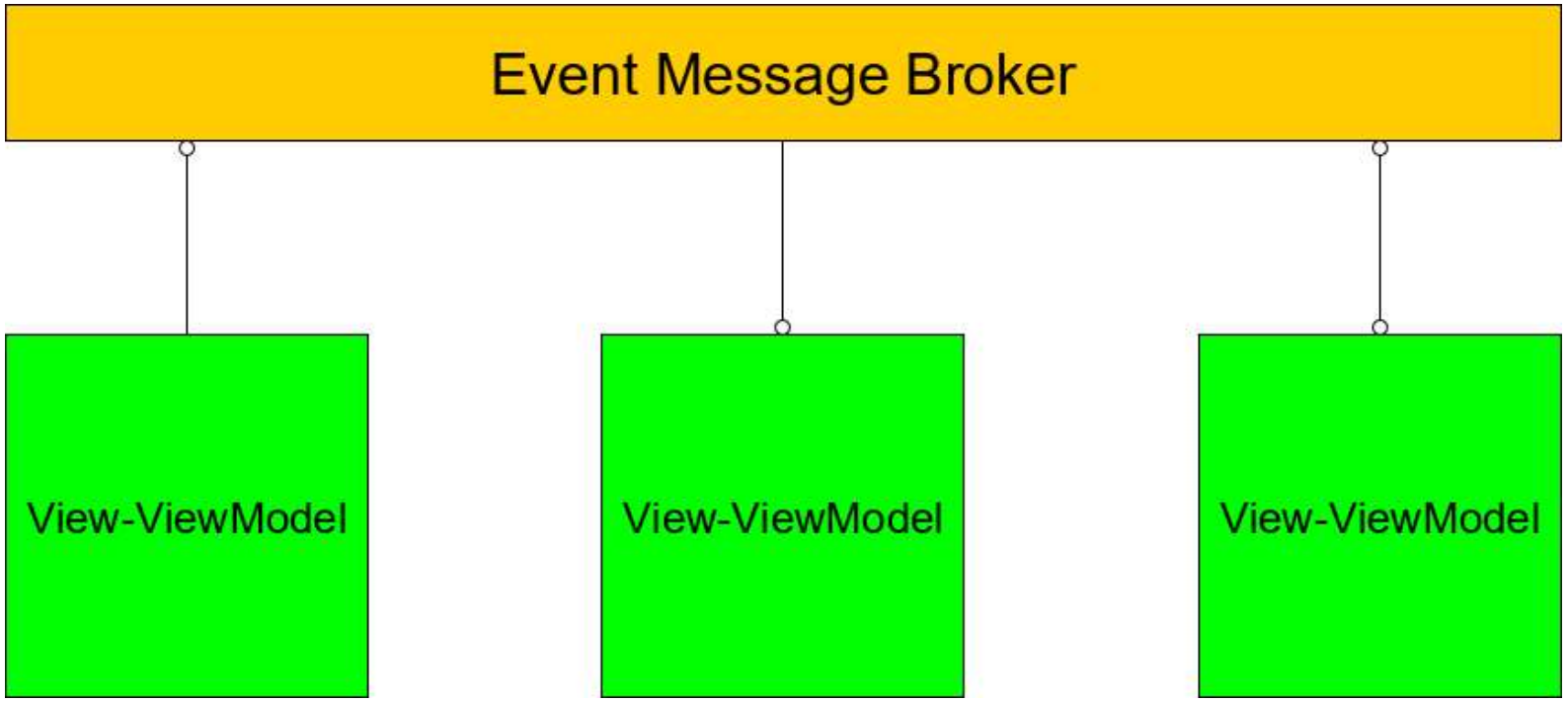
Коммуникации с модулем

- Events/Methods
- Pub/Sub

Events/Methods

```
public sealed class LoginViewModel: ViewModelBase
{
    public LoginViewModel(ILogin login, IMessenger messenger) : base(messenger){}
    public event EventHandler<SuccessEventArgs> SuccessEvent;
    public void InvokeSuccessEvent() {
        SuccessEvent?.Invoke(this, new SuccessEventArgs());
    }
    public sealed class SuccessEventArgs: EventArgs {}
}
public sealed class ApplicationViewModel: ViewModelBase
{
    public ApplicationViewModel() {
        LoginViewModel = new LoginViewModel();
        LoginViewModel.SuccessLoginEvent += (sender, args) => IsActive = true;
    }
    public LoginViewModel LoginViewModel { get; }
    public bool IsActive { get; set; }
}
```

Pub/Sub


Event Message Broker
View-ViewModel
View-ViewModel
View-ViewModel

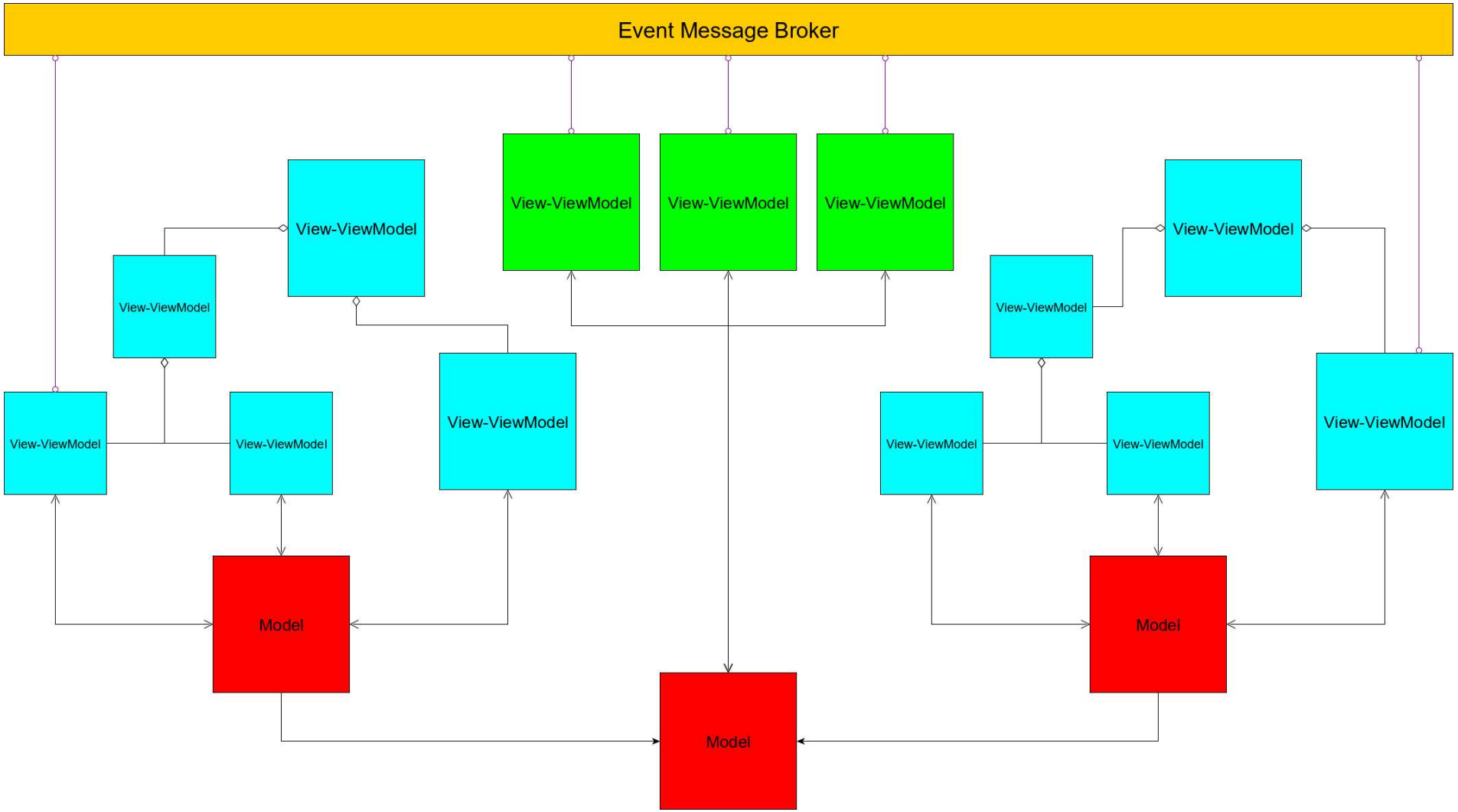
Pub/Sub здорового человека

- Односторонние
- Сообщения - неизменяемые структуры данных
- 1 Pub - n Sub

Pub/Sub курильщика

- Двусторонние (с большой осторожностью)
- n Pub - n Sub
- Сообщения - изменяемые объекты (привязки, события)
- Получение одного сообщения приводит к отправке другого сообщения

Pub/Sub


Event Message Broker
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
View-ViewModel
Model
Model
Model

```
public sealed class LoginViewModel: ViewModelBase
{
    public LoginViewModel(ILogin login, IMessenger messenger) : base(messenger) {}
    public void SendSuccess()
    {
        MessengerInstance.Send<SuccessLoginMessage>(new SuccessLoginMessage());
    }
    public sealed class SuccessLoginMessage: MessageBase
    {
        internal SuccessLoginMessage() {}
    }
}
public sealed class NotificationViewModel: ViewModelBase
{
    public NotificationViewModel(INotification notification, IMessenger messenger): base(messenger)
    {
        this.MessengerInstance.Register(this, (LoginViewModel.SuccessLoginMessage msg) => {});
    }
}
```

Pub/Sub

Плюсы:

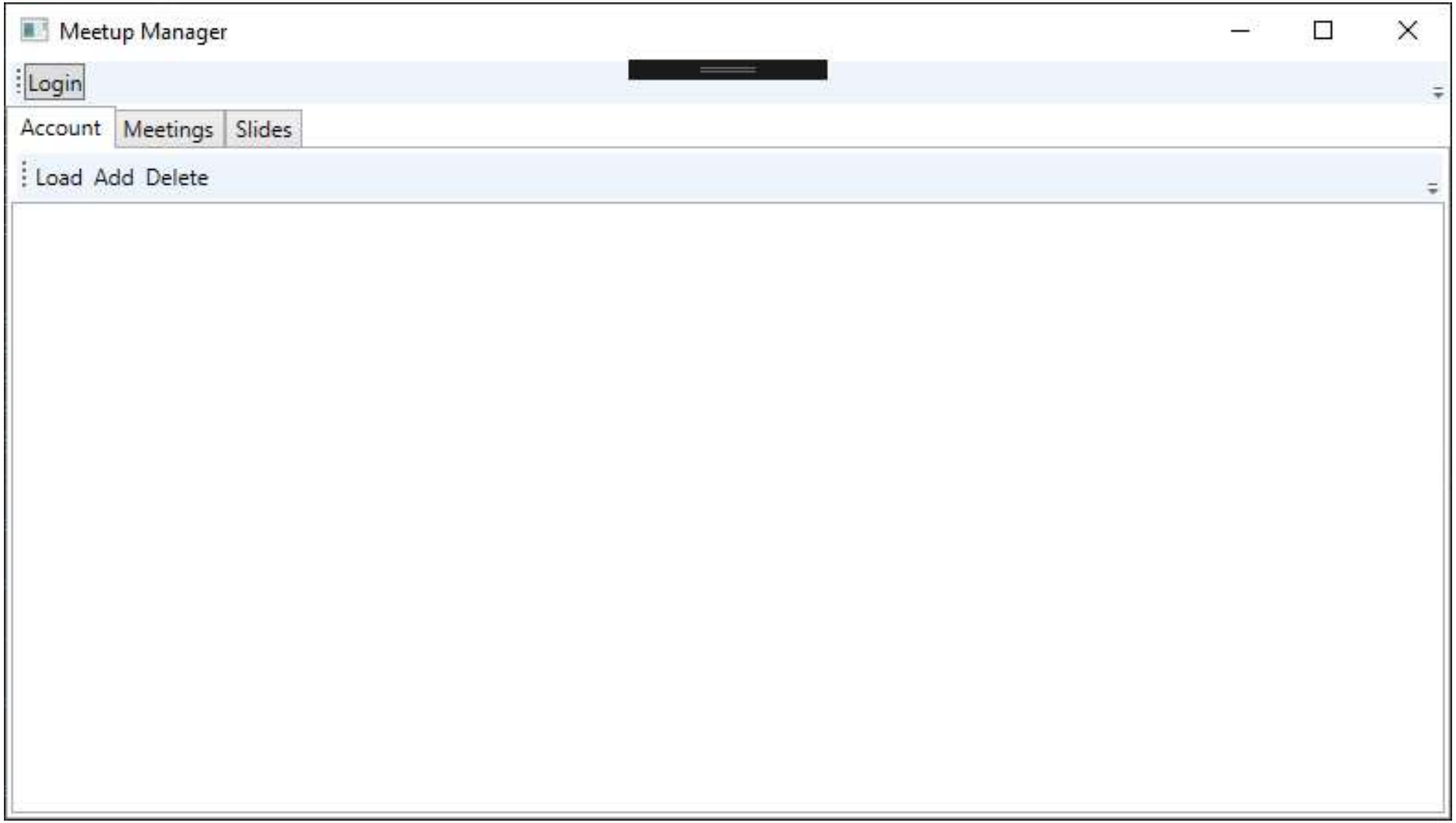
- упрощает интеграцию модулей, зависимые элементы могут быть не связаны с модулями логически

Минусы:

- +1 элемент инфраструктуры
  - логирование
  - настройки
  - асинхронность
- менее явная связь между элементами приложения и модулями

Щ.И.Т.

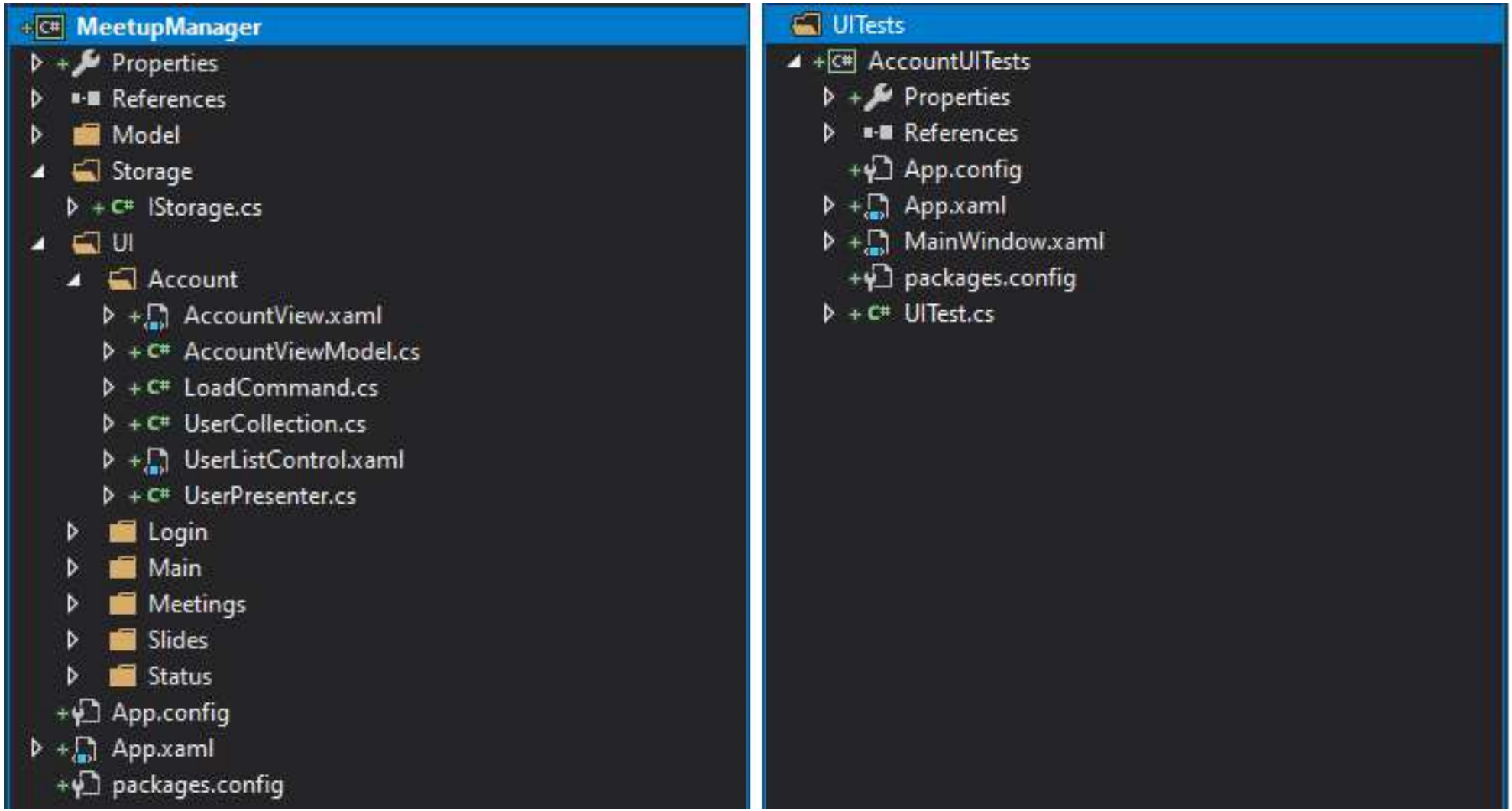
- Unit tests
- UI tests
- Вспомогательные тесты


Meetup Manager
Login
Account
Meetings
Slides
Load
Add
Delete

Unit tests

```
[TestMethod]
public void LoadCommand_OneTimeExecute()
{
    var command = new LoadCommand(new UserCollection(new FakeAccount()));

    Assert.IsTrue(command.CanExecute(null));

    command.Execute(null);

    Assert.IsFalse(command.CanExecute(null));
}

[TestMethod]
public void UserCollection_Load()
{
    var userCollection = new UserCollection(new Account(new FakeStorage()));

    userCollection.Load();

    Assert.IsTrue(userCollection.Any());
}
```

UI tests


MeetupManager
Properties
References
Model
Storage
IStorage.cs
UI
Account
AccountView.xaml
AccountViewModel.cs
LoadCommand.cs
UserCollection.cs
UserListControl.xaml
UserPresenter.cs
Login
Main
Meetings
Slides
Status
App.config
App.xaml
packages.config
UITests
AccountUITests
Properties
References
App.config
App.xaml
MainWindow.xaml
packages.config
UITest.cs

```
public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            MainWindow = new MainWindow
            {
                DataContext = new AccountViewModel(new FakeAccount())
            };
            ShutdownMode = ShutdownMode.OnMainWindowClose;
            MainWindow.Show();
        }
    }
```

```
<Window ...>
    <Grid>
        <account:AccountView />
    </Grid>
</Window>
```

```
using FlaUI.UIA3;
...
[TestClass]
public class UITest {
    [TestMethod]
    public void Application_Launch()
    {
        var app = FlaUI.Core.Application.Launch("AccountUITests.exe");
        try
        {
            using (var automation = new UIA3Automation())
            {
                var window = app.GetMainWindow(automation);
                Assert.IsNotNull(window.Title);
            }
        }
        finally
        {
            app.Close();
        }
    }
}
```

Вспомогательные тесты

```
[TestClass]
public class UnitTest1
{
    [TestMethod]
    public void TestInitialization()
    {
        _testInitialization(new MainViewModel());
    }

    private void _testInitialization<T>(T vm)
    {
        var type = vm.GetType();
        foreach (var propertyInfo in type.GetProperties(BindingFlags.Public))
        {
            Assert.IsNotNull(propertyInfo.GetValue(vm));
        }
    }
}
```