Основы архитектуры

MVC

**MVC (Model, View, Controller) –** это архитектурный паттерн предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо. Controller общается с клиентом посредством запросов. Он получат запрос и передает view, если пользователь передал данные, которые надо обработать, Controller передает их model, получает то что вернула model передает это view и отдает view клиенту.

* **Model**– это данные вашего приложения, логика их получения и сохранения. Зачастую это модель предметной области (domain model), основанная на базе данных или на результатах от веб-сервисов. В некоторых случаях domain model хорошо проецируется на то, что вы видите на экране. Но иногда перед использованием ее необходимо адаптировать, изменить или расширить.
* **View** отвечала за отображение UI на экране. Без библиотек виджетов, это означало самостоятельную отрисовку блоков, кнопок, полей ввода и т. п. View также может наблюдать за моделью и отображать данные из неё.
* **Controller** обрабатывает действия пользователя и затем обновляет Model или View. Если пользователь взаимодействует с приложением (нажимает кнопки на клавиатуре, передвигает курсор мыши), контроллер получает уведомление об этих действиях и решает, что с ними делать.

**HMVC (Hierarchical model–view–controller)** - в приложении используются только жестко фиксированные триады модель-вид-контроллер, которые взаимодействуют с остальными подсистемы исключительно через контроллер. То есть Model, View, Controller имеют только единичные экземпляры, а общаются они с помощью запросов между собой через контроллер.

**MVP(****Model, View, Presenter) -** Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Presenter, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель. То есть View принимает изменения, и через абстракцию передает данные Presenter он в свою очереди общается с Model и обновляет View.

* **Model** это данные вашего приложения, логика их получения и сохранения. Зачастую она основана на базе данных или на результатах от веб-сервисов. В некоторых случаях потребуется ее адаптировать, изменить или расширить перед использованием во View.
* **View** обычно представляет собой форму с виджетами. Пользователь может взаимодействовать с ее элементами, но когда какое-нибудь событие виджета будет затрагивать логику интерфейса, View будет направлять его презентеру.
* **Presenter** содержит всю логику пользовательского интерфейса и отвечает за синхронизацию модели и представления. Когда представление уведомляет презентер, что пользователь что-то сделал (например, нажал кнопку), презентер принимает решение об обновлении модели и синхронизирует все изменения между моделью и представлением.

**MVVM (Model-View-ViewModel) -** Данный подход позволяет связывать элементы представления со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя.

ViewModel не может общаться со View напрямую. Вместо этого она представляет легко связываемые свойства и методы в виде команд. View может привязываться к этим свойствам, чтобы получать информацию из ViewModel и вызывать на ней команды (методы). Это не требует того, чтобы View знала о ViewModel. XAML Databinding использует рефлексию, чтобы связать View и ViewModel. Таким образом, вы можете использовать любую ViewModel для View, которая предоставляет нужные свойства.