**Архитектурные паттерны**

*Архитектурный паттерн или шаблон* – это обобщенные часто используемые решения распространённой задачи в архитектуре ПО.

Обычное приложение состоит из следующих уровней:

1. Уровень представления
2. Уровень обработки данных (бизнес-логика)
3. Уровень доступа к данным (data access)

Для взаимодействия между уровнями может использоваться **API** (Application Programming Interface).

Сейчас часто используется архитектура **MV(x)**

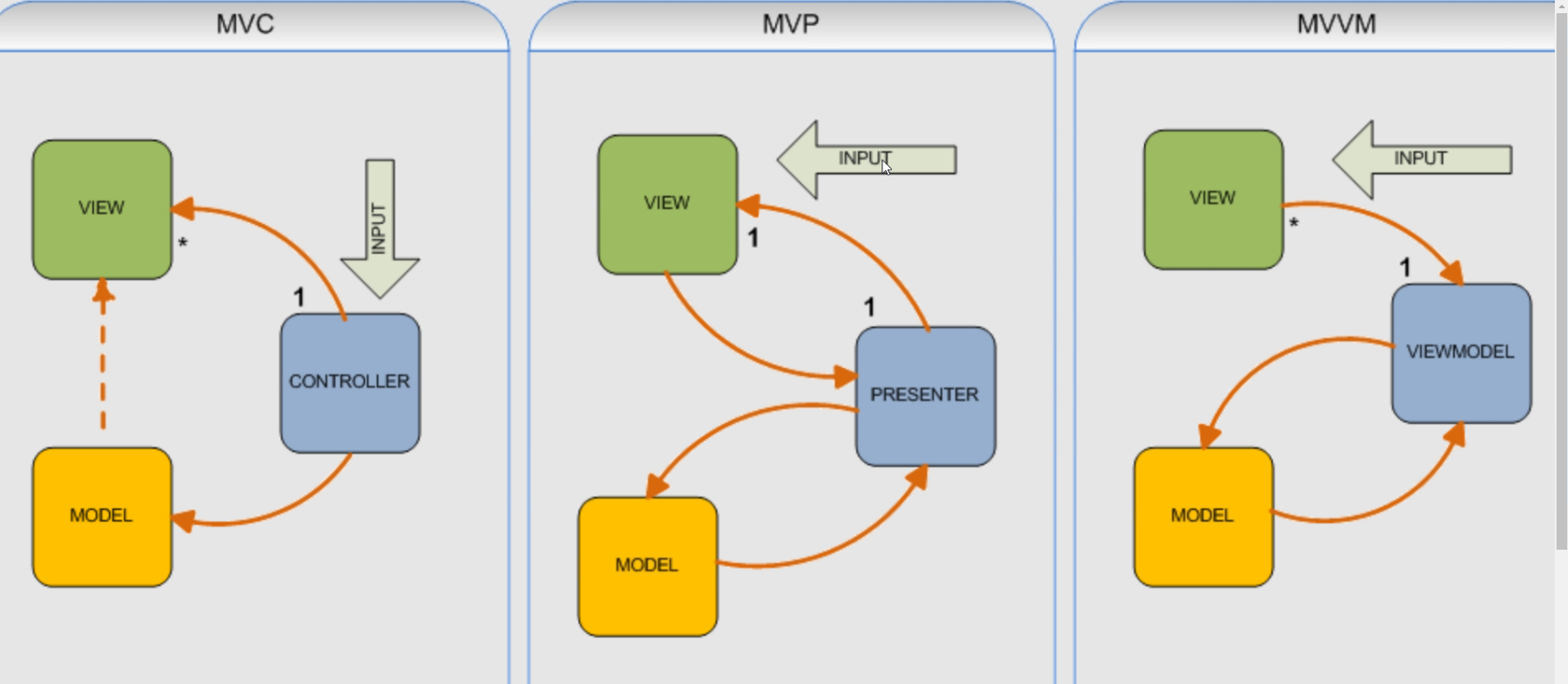
1994– описан паттерный GoF. В них упоминается архитектура MVC.

*MVC* – это набор паттерных наблюдатель, стратегия и компоновщик.

80е – появление архитектуры **MVC**.

90е – появление архитектуры **MVP**.

2005 – появился паттерн **MVVM** (Microsoft) (в мобильных приложениях).



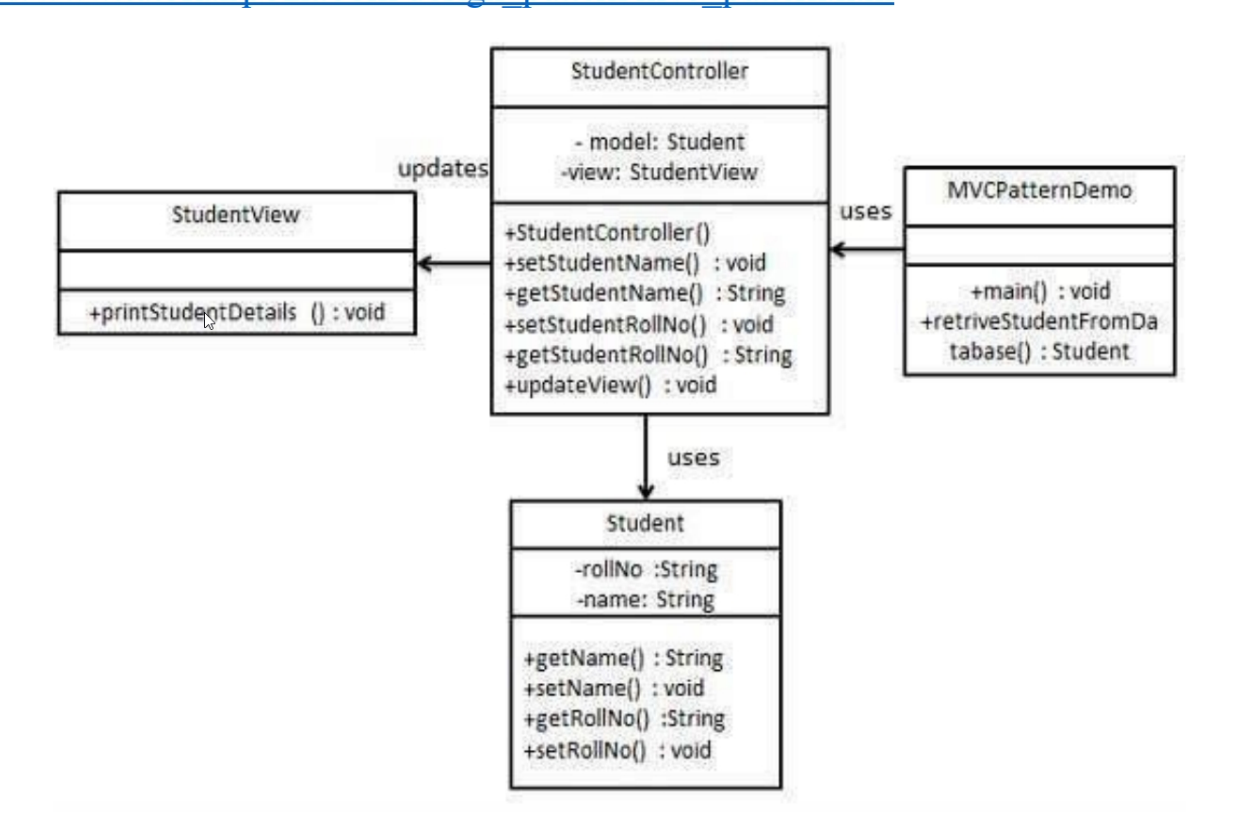
**MVC**

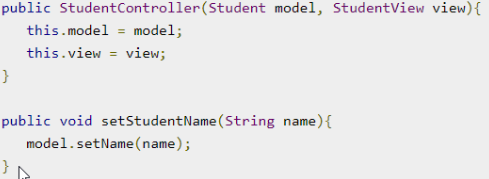
*MVC* – это фундаментальный шаблон проектирования. Отделяет логику интерфейс пользователя от логики программирования. MVC имеет множество реализаций. Сейчас подходит для описания объектов.

*Model* содержит модель данных и бизнес-логику.

*View* или представление/вид – это интерфейс взаимодействия с пользователем. Может содержать свою бизнес логику.

*Controller* связывает модель и представление между собой, обрабатывает события пользователя и управляет состоянием моделью





MVP.

Для реализации создается интерфейс IView, содержащий набор методов, необходимых Presenter.

MVVM.

Представление не требует реализации IView. Для получения данных требуется предать ссылку на источник данных (DataContext).

ViewModel или модель построения – это посредник между передачей данных от модели DataContext. Элементы View (представления) ссылаются на источник данных через Binding (привязка данных). Может потребоваться реализация паттерна Command.

ViewModel используется в:

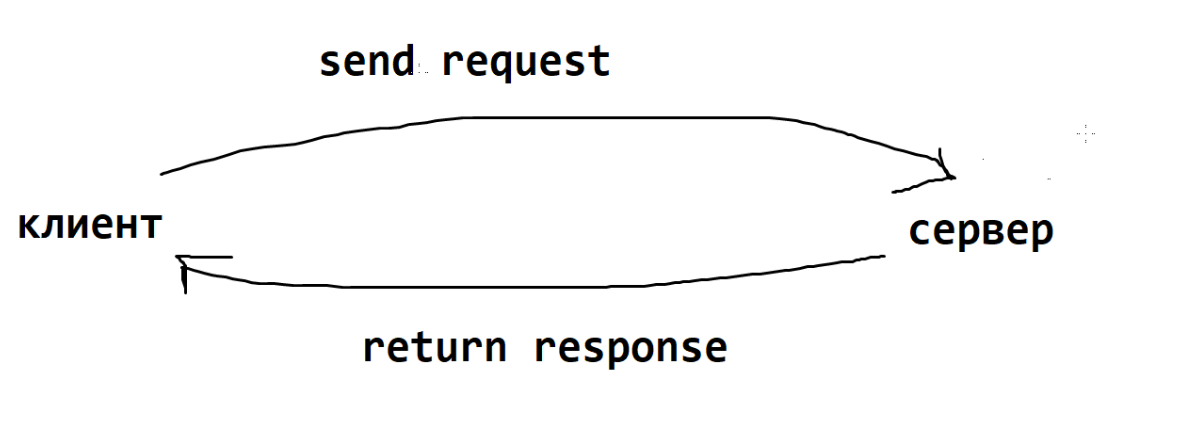
WPF, MAUI (C#)

Android

REST(RESTful) – это архитектурный подход для создания API.

REST : REpresentational State Transfer

Не требует привязки http, не зависит от базовых протоколов, не хранит состояние.



Ресурс – это то, что доступно пользователю. Он существительный. Например: example.com.

endpoint или конечная точка – это путь к ресурсу. Указывается в конце.

example.com/api/v1/users

v1 – версия

users – какие данные мы хотим обрабатывать

Методы для обращения к ресурсам или конечным точкам:

1. GET (для получения данных) (JSON/XML)

GET: /urers

GET: /urers/идентификатор

GET:

/родители/иденфикаторРодителя/дети/иденфикаторРебенка

1:М

/users/идПользователя/photos

/users/идПользователя/photos/идФото

GET:

/ресурс?параметр1=значение1&параметр2=значение2&…

/users?age=18&city=Архангельск

или

/users/18/Архангельск

Для запросов, возвращающих набор строк, рекомендуется предусмотреть постраничный вывод.

/users?page=2&size=10

Набор возвращается в **[]**

Атрибуты в **“”** перед **:**

Значение после двоеточия. У значения может быть другой объект.



*POST* – это создание нового ресурса. Требуется передача объекта в параметрах метода.

POST:

/users – путь такой же, как и для получения всего списка

*PUT* – это изменение ресурса с указанным идентификатором

PUT:

/users/3

PATCH (закладка) меняет часть данных ресурса

PUTCH:

/users/3

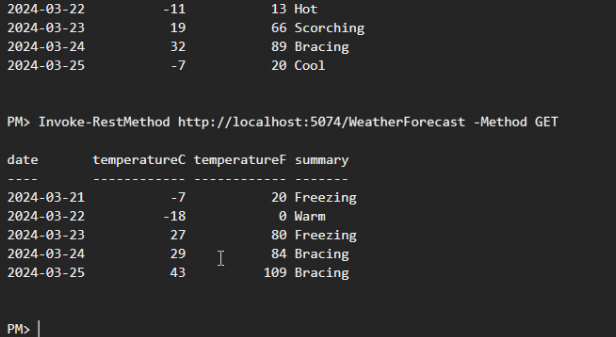
*DELETE* – удаляет ресурс по указанным идентификаторам.

DELETE:

/users/3

Для тестирования методов изменения данных обычно используются Postman, Swagger доступен в WS (WEBAPI (включить поддержку OpenAPI, убрать галочку “Настроить для https”)), PowerShall

Вызов метода GET:



Структура запроса:

1. ORL-адрес
2. Метод(Method) (GET, POST, …)
3. Заголовок ( в формате JSON/XML)
4. body – параметр метода
5. Response или ответ сервера. Приходит код состояния

Код состояние

2xx – всё стабильно

4xx – ошибка доступа

5xx – ошибка сервера

Примеры ответа:

200 – всё окей

201 – удачный для запроса POST(удалось создать ресурсы)

404 – не найдено GET

403 – доступ запрещен

401 – не авторизован

409 – не удалось обновить PUT

Может вернуть заголовок и тело