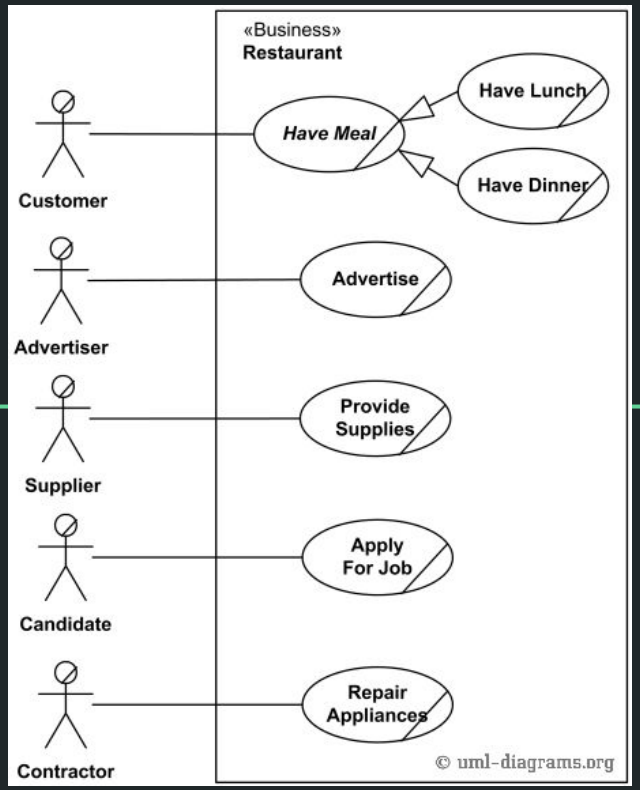
UML диаграммы

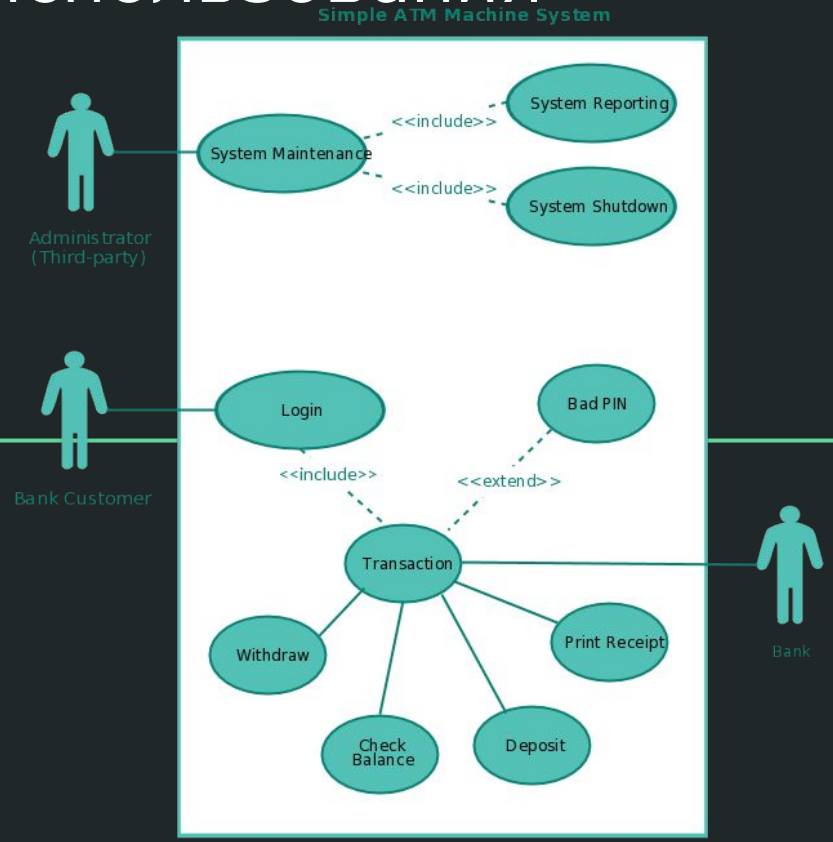
1. Их много (около 15 в версии 2.x)
2. В рамках курса нужны не все
3. На самом деле они все полезные
4. Диаграмма Классов (Class)
5. Диаграмма Компонентов (Component)
6. Диаграмма Развертывания (Deployment)
7. Диаграмма Сценариев использования (Use Case)

Не UML диаграммы

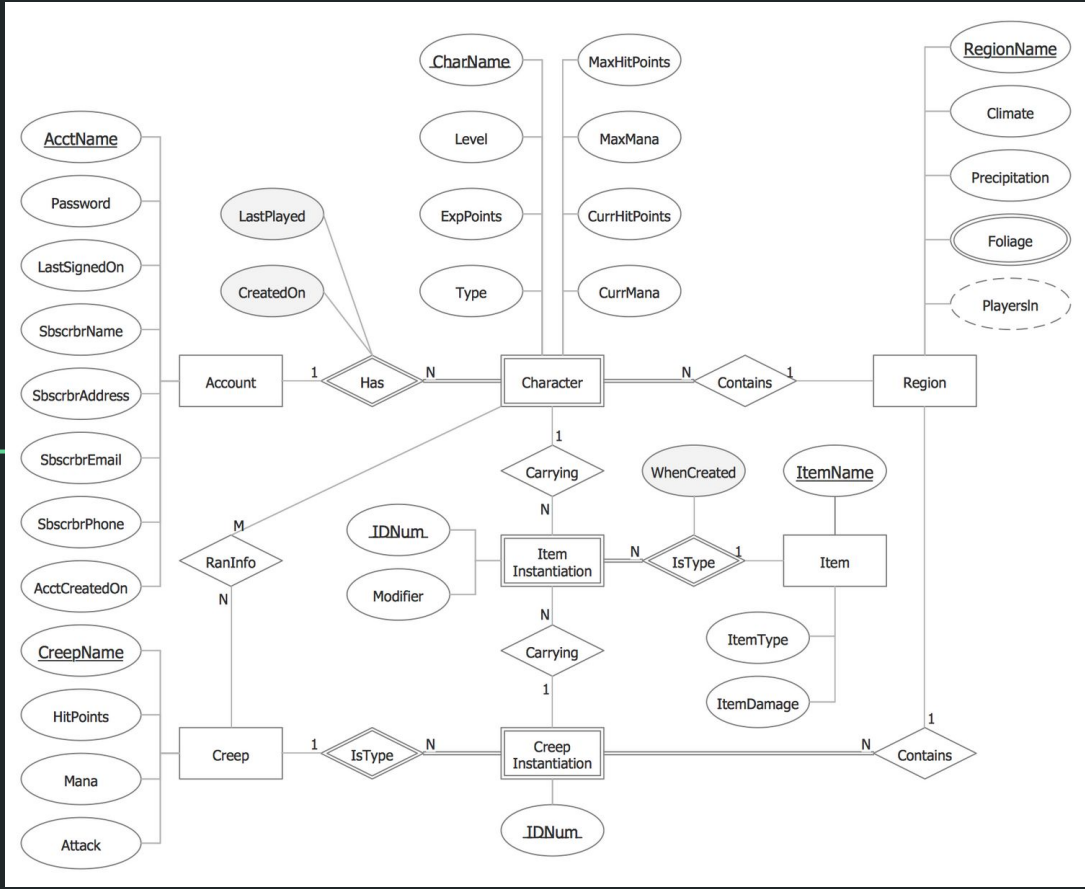
1. Диаграмма ER в нотации Чена (Entity Relationship)
2. Диаграмма БД (Database)
3. Нет, это не одно и то же

Use Case / Сценарий использования

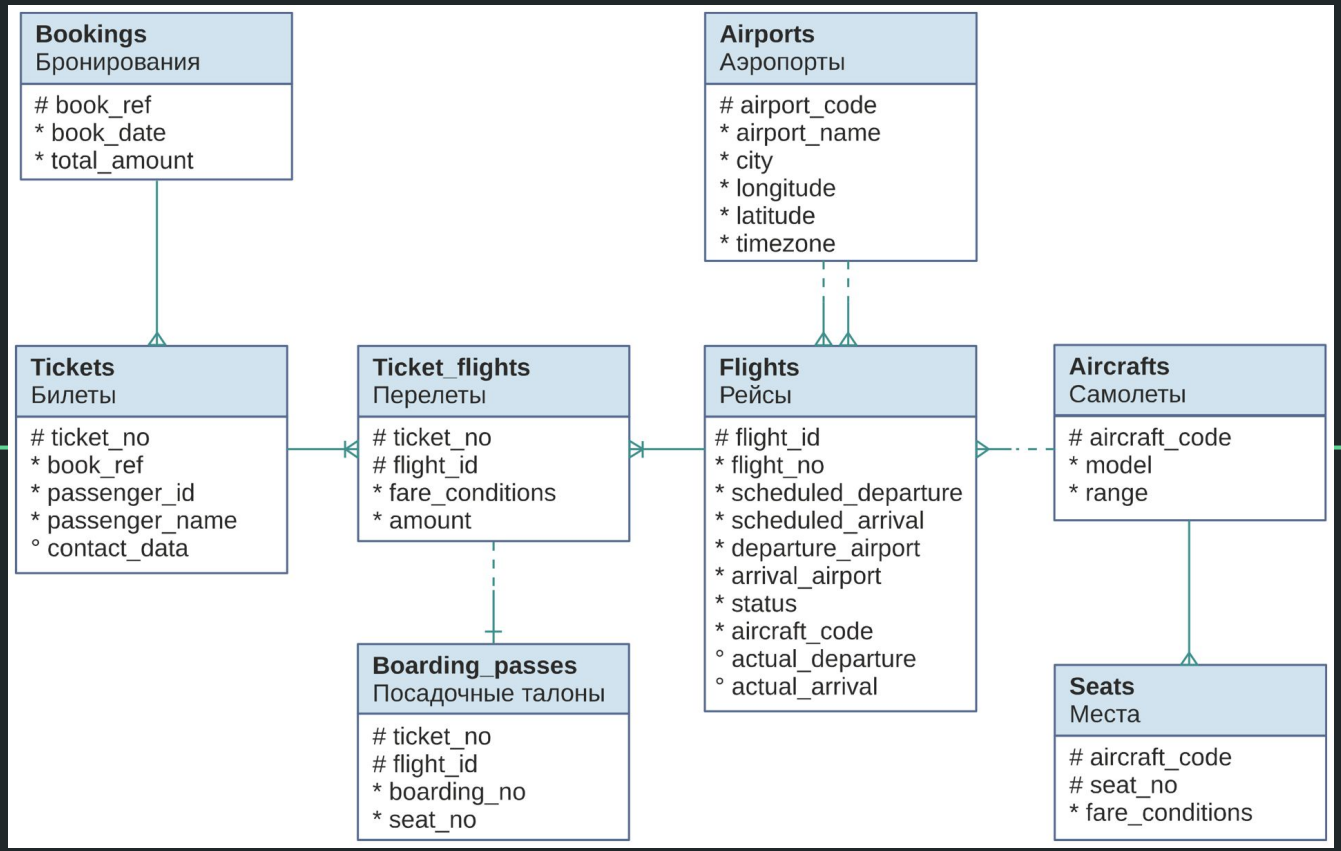




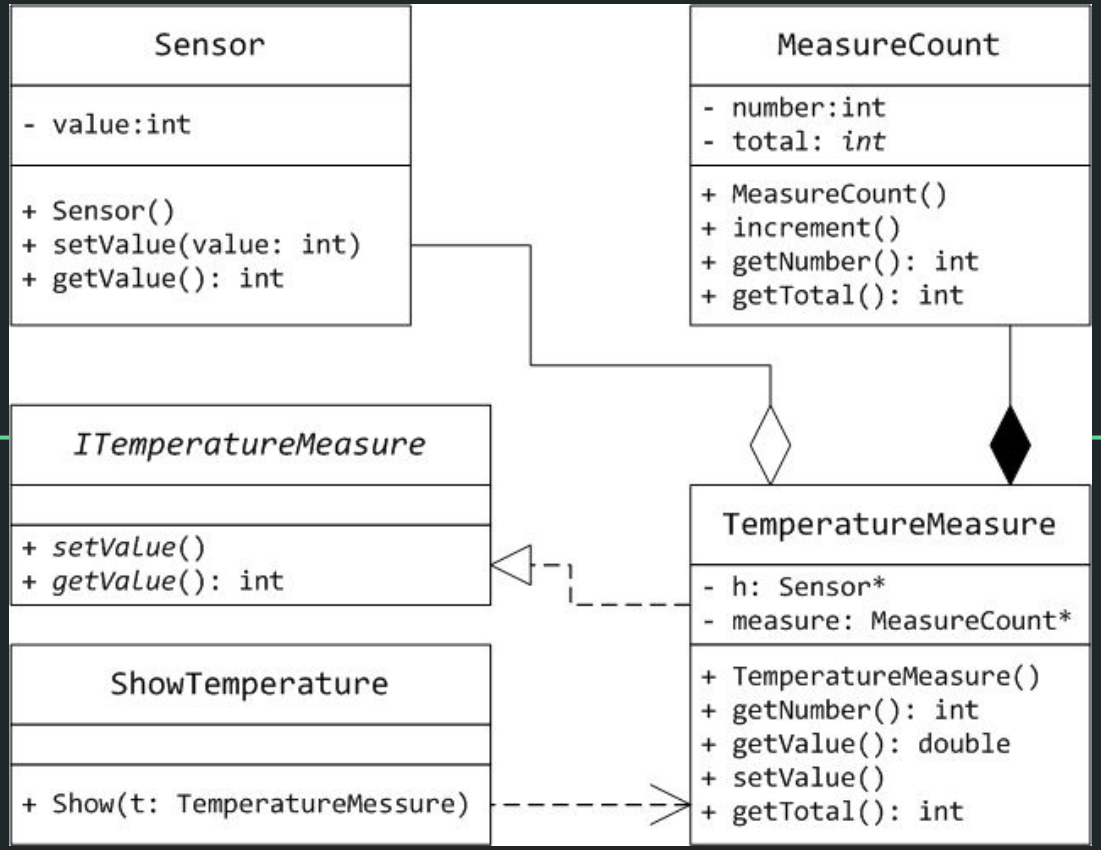
ER (Entity Relationship)

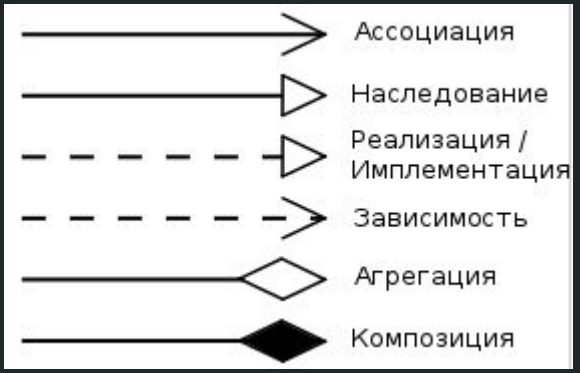


БД (Database)


Bookings
Бронирования
book_ref
* book_date
* total_amount
Airports
Аэропорты
airport_code
* airport_name
* city
* longitude
* latitude
* timezone
Tickets
Билеты
ticket_no
* book_ref
* passenger_id
* passenger_name
° contact_data
Ticket_flights
Перелеты
ticket_no
flight_id
* fare_conditions
* amount
Flights
Рейсы
flight_id
* flight_no
* scheduled_departure
* scheduled_arrival
* departure_airport
* arrival_airport
* status
* aircraft_code
° actual_departure
° actual_arrival
Aircrafts
Самолеты
aircraft_code
* model
* range
Boarding_passes
Посадочные талоны
ticket_no
flight_id
* boarding_no
* seat_no
Seats
Места
aircraft_code
seat_no
* fare_conditions

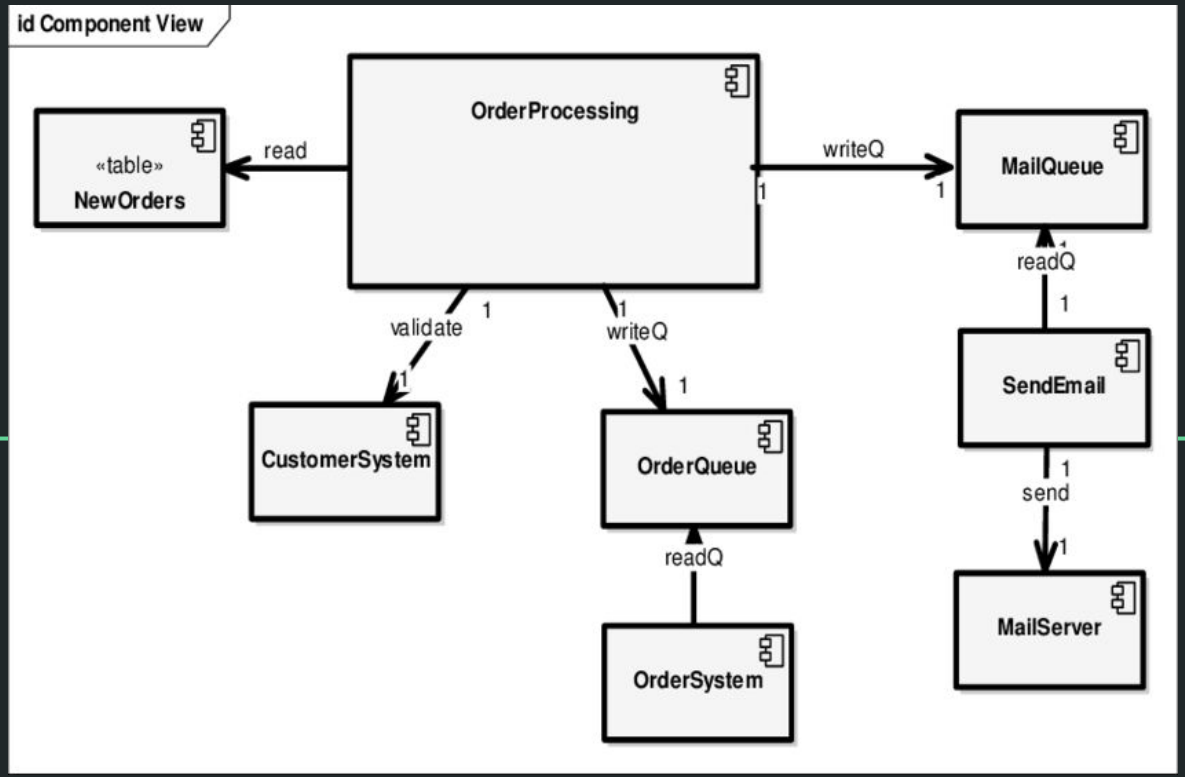
Классы (Class)



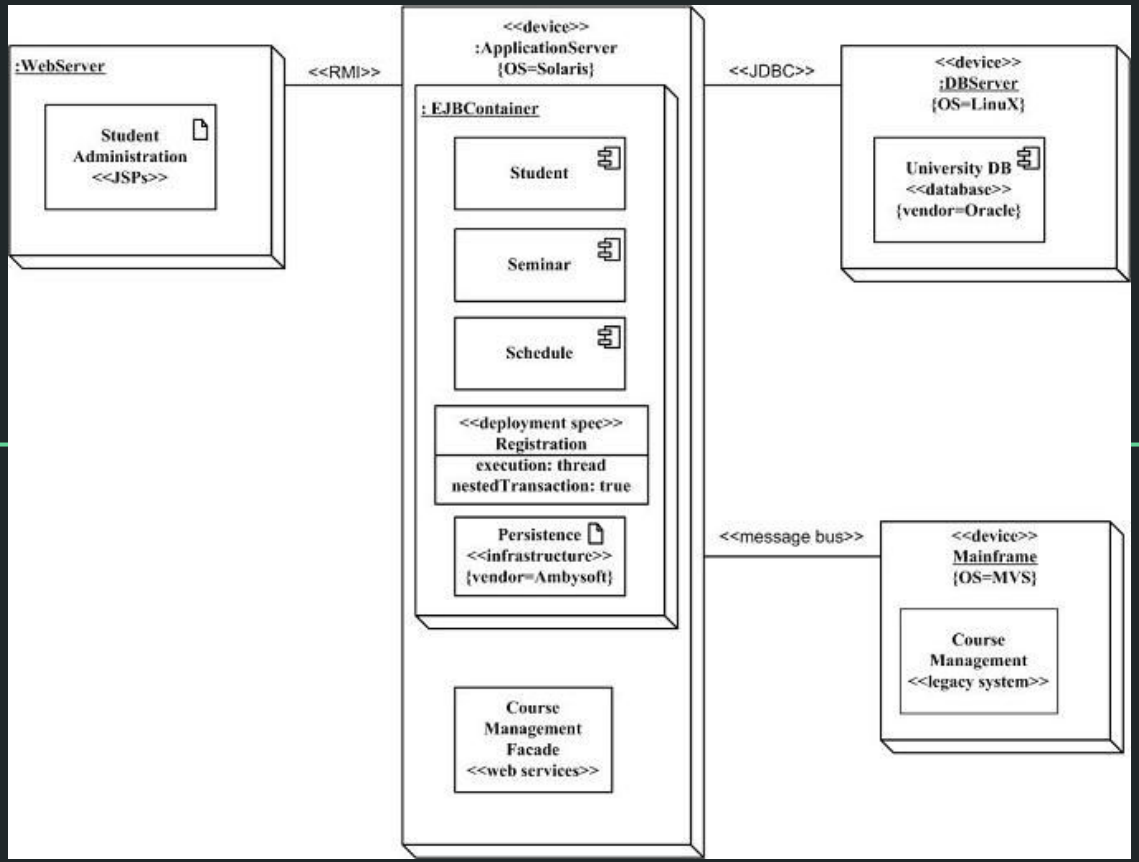


+	Публичный (Public)
-	Приватный (Private)
#	Защищённый (Protected)
/	Производный (Derived) (может быть совмещён с другими)
~	Пакет (Package)

Компонент (Component)



Развертывание (Deployment)



Многостраничное приложение (Multi page Application)

1. Легко разрабатывать
2. Легкая оптимизация в поисковых движках
3. Разные фреймворки на клиенте и сервере
4. Траффик
5. Нецелесообразная нагрузка на сервер

---

6. Front-end и back-end тесно связаны
7. Легко визуализировать действия пользователей в виде графа

Одностраничное приложение (Single page Application)

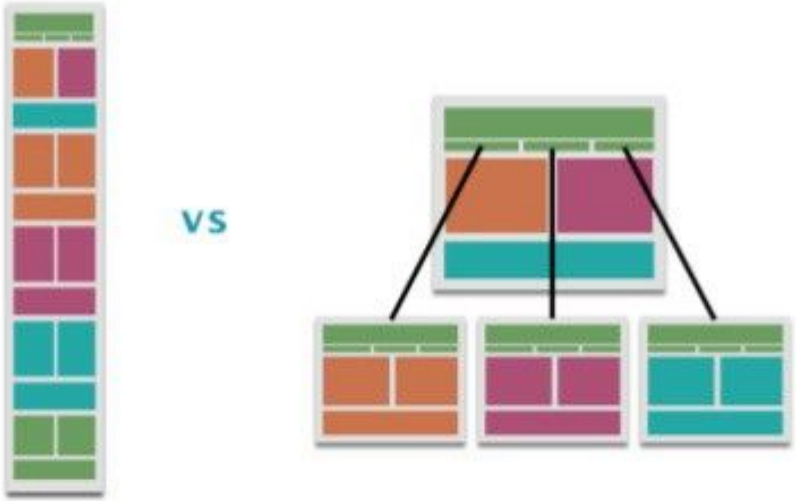
1. Придется разбираться с фреймворками типа AngularJS, Ember.js, Meteor.js, Knockout.js и всех появившихся за время семинара
2. Высокая нагрузка на клиент
3. Нужно выгружать много артефактов на клиенте (Java Script)
4. Дырки в безопасности
5. Утечки памяти
6. Меньше трафика и нагрузки на сервер
7. Back-end можно писать на любом фреймворке

MPA vs SPA

Which one prefered?

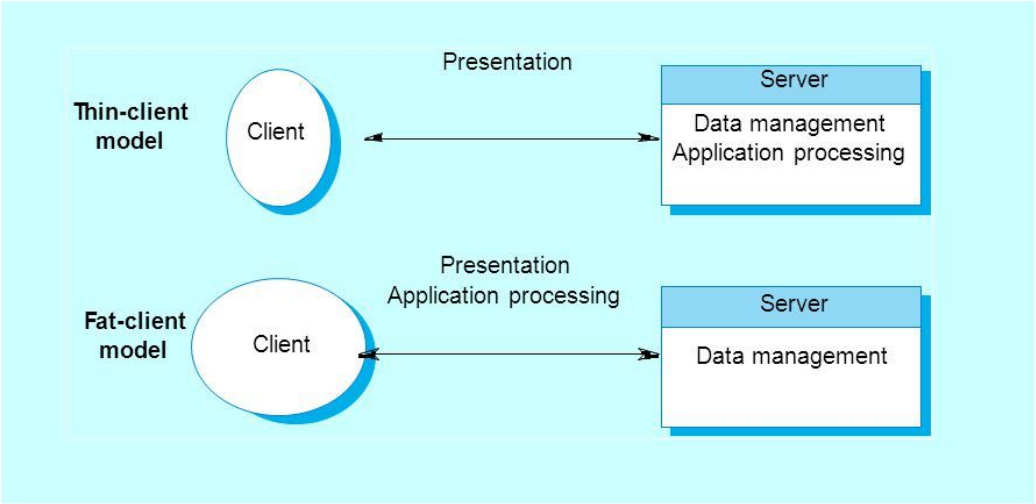
Single Page App

Multi Page App

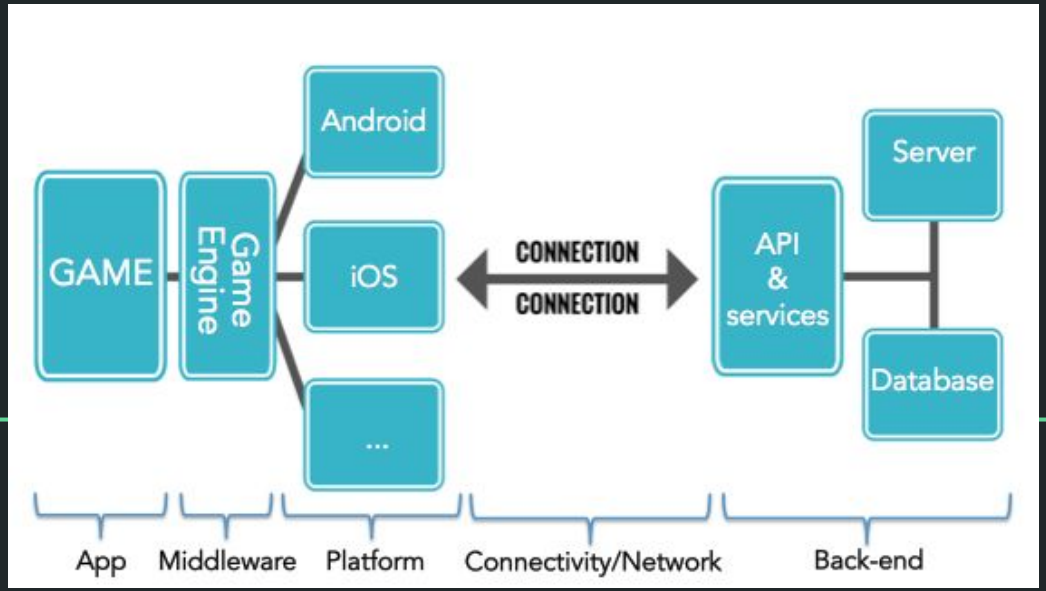


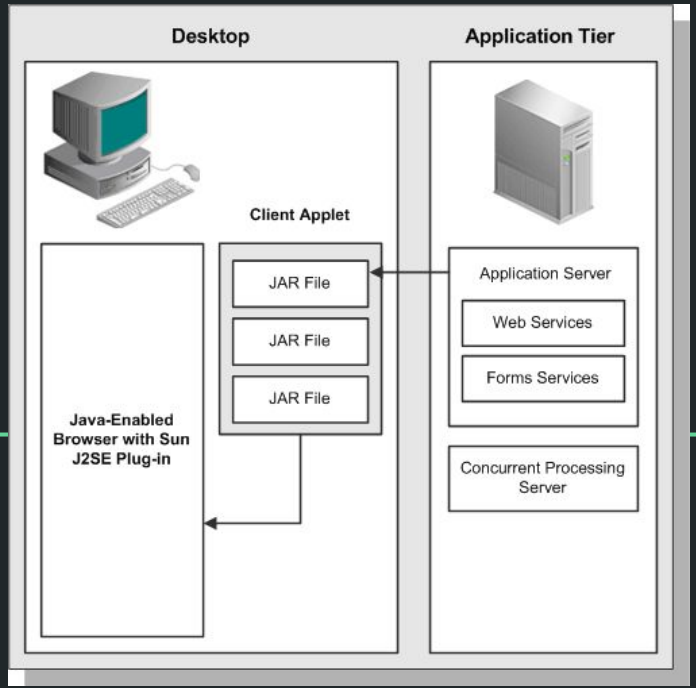
Тонкие и толстые клиенты

Thin and fat clients



Десктопные и мобильные приложения





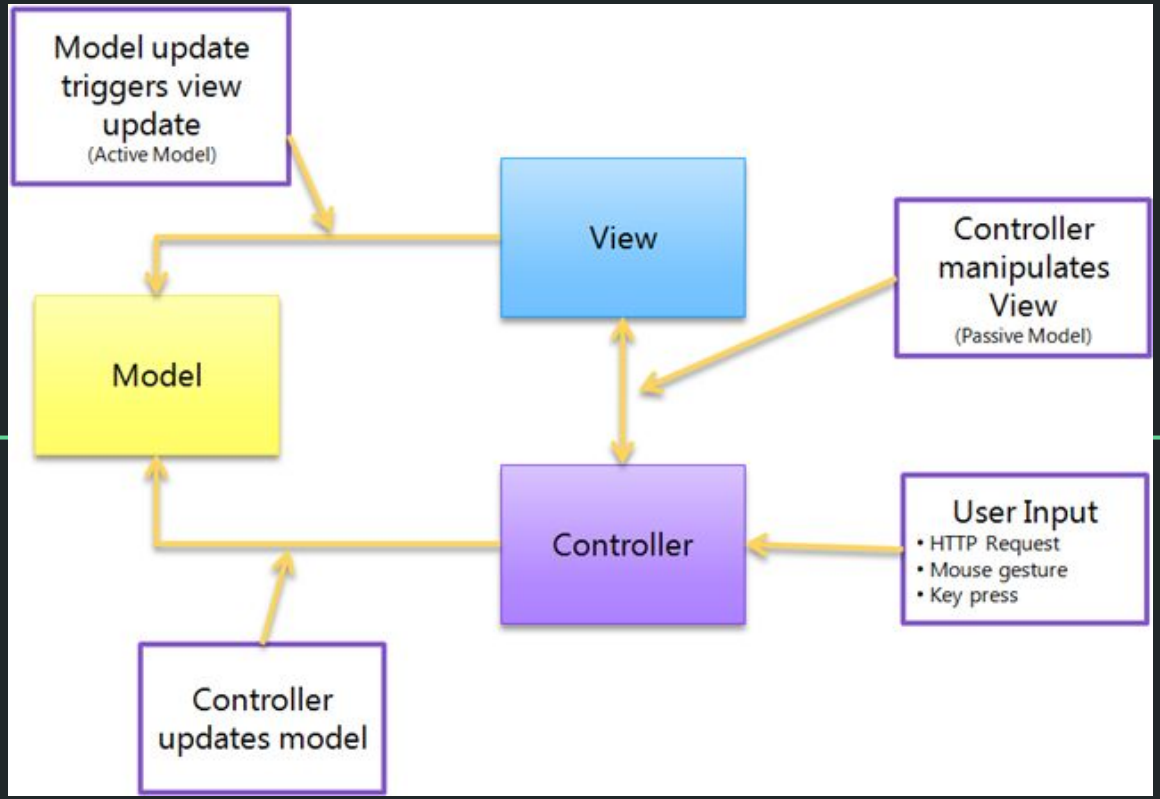
MVC (Model View Controller)

Модель содержит в себе всю логику приложения, она хранит и обрабатывает данные, при этом не взаимодействуя с пользователем напрямую (обратиться к Модели можно только из кода, вызывая ее функции). Например, сохранение информации в БД, проверка правильности введенных в форму данных — это задача Модели, но получение этих данных от пользователя или вывод информации на экран или обработка нажатия на кнопку — нет.

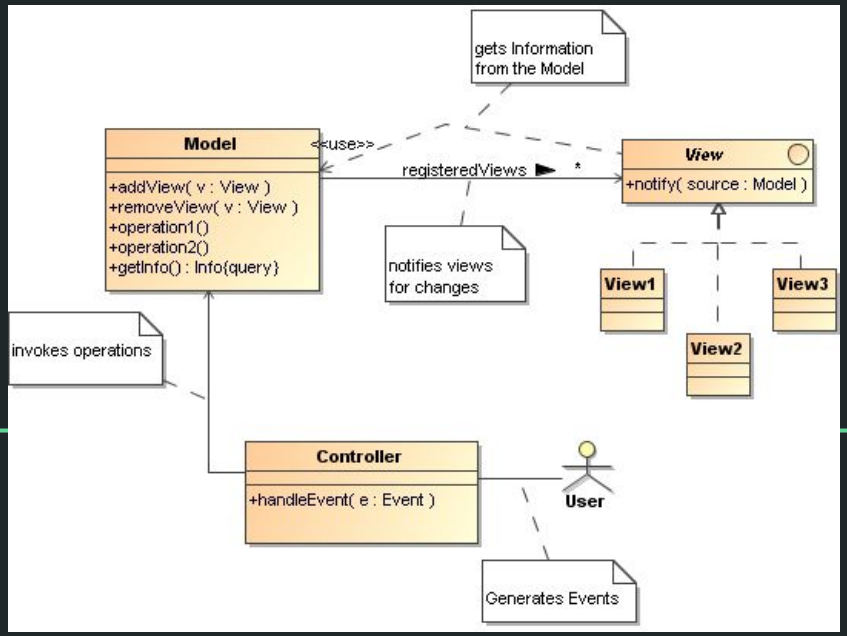
Представление отображает данные, которые ему передали. В веб-приложении оно обычно состоит из HTML-шаблонов страниц, в десктопных или мобильных приложениях Преставление - это код, который отвечает за отображение информации на экране, отрисовку кнопочек и других элементов интерфейса.

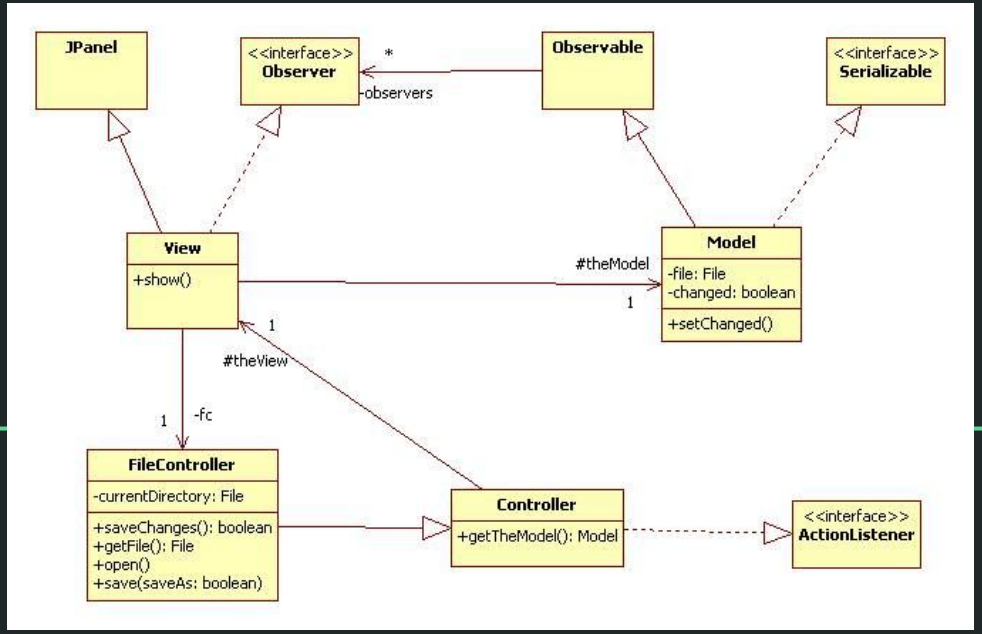
Контроллер отвечает за выполнение запросов, пришедших от пользователя. В веб-приложении обычно контроллер разбирает параметры HTTP-запроса из $_POST/$_GET, обращается к модели, чтобы получить или изменить какие-то данные, и в конце вызывает Представление, чтобы отобразить результат выполнения запроса. Число контроллеров определяется числом разделов или страниц сайта. В десктопных приложениях Контроллер отвечает за обработку нажатий на кнопки и других воздействий от пользователя.

MVC (Model View Controller)



Анатомия MVC





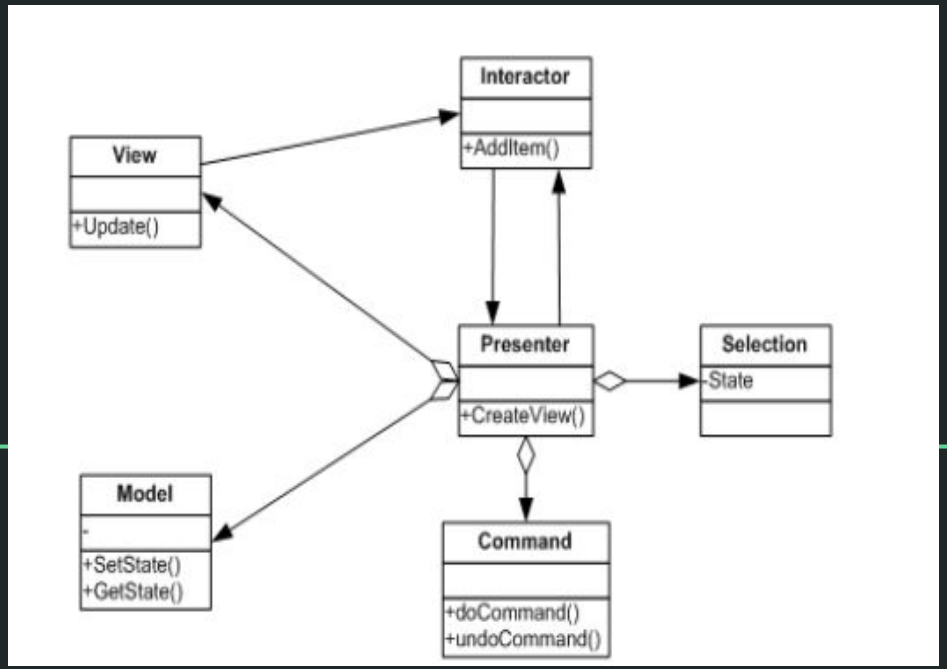
MVP (Model View Presenter)

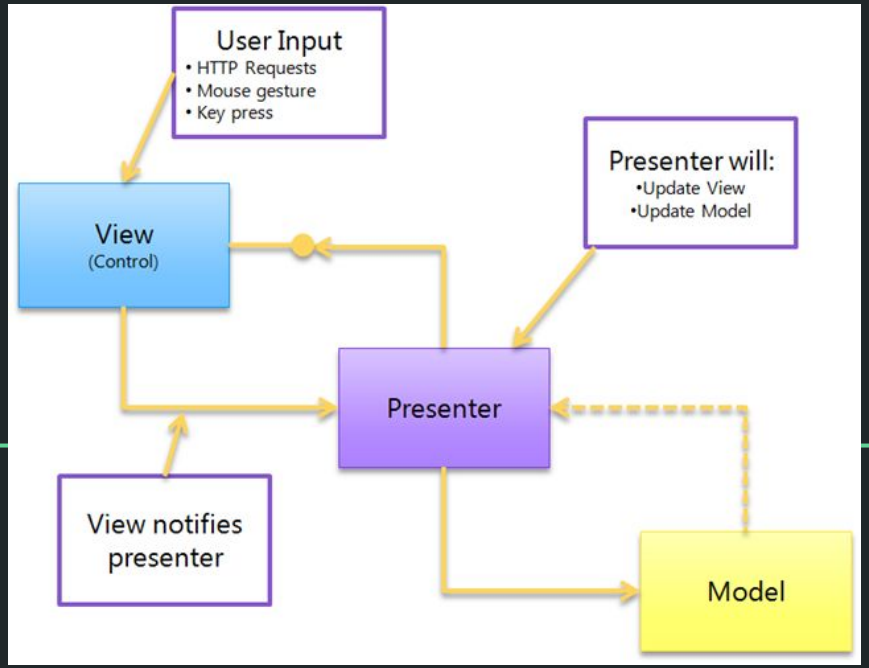
Модель - это данные вашего приложения, логика их получения и сохранения. Зачастую она основана на базе данных или на результатах от веб-сервисов. В некоторых случаях потребуется ее адаптировать, изменить или расширить перед использованием во View.

Представление - 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.

Презентер 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.

MVP (Model View Presenter)





Другие варианты развития этого семейства паттернов

MTV (Model Template View) -- вариант MVC в Python+Django
MVPVM (The Model-View-Presenter-ViewModel)

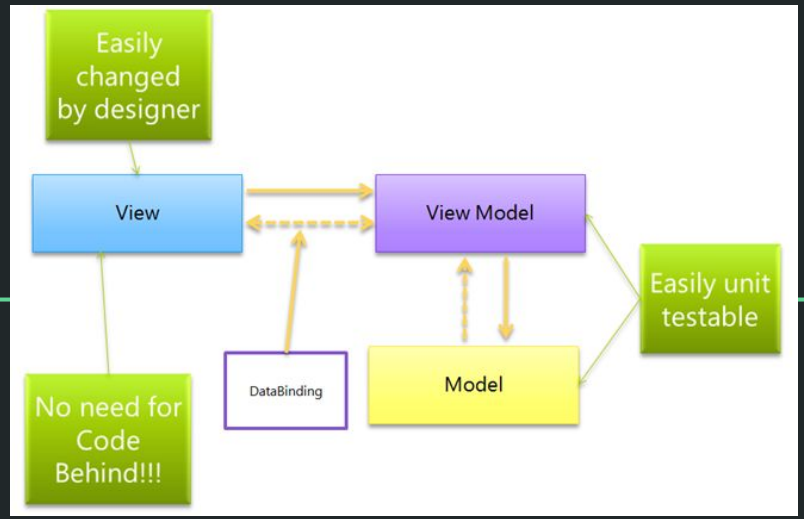
Model-View-ViewModel

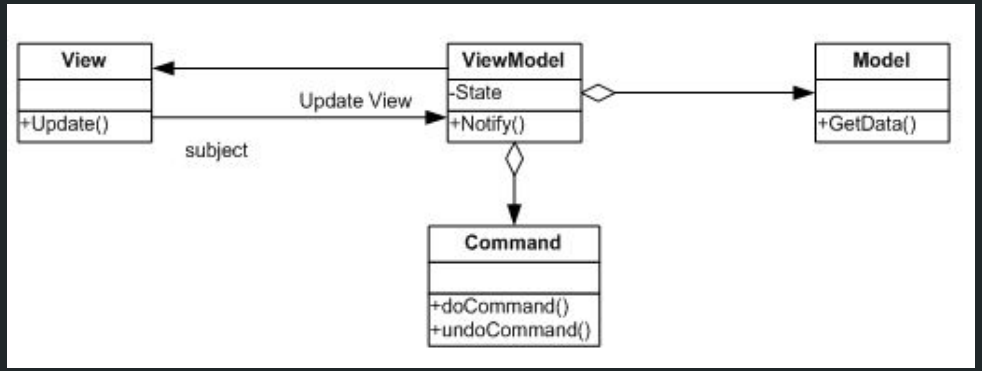
ViewModel не обращается напрямую к View

ViewModel содержит методы и свойства в виде команд

Само View получается простым в реализации

Требуется реализованный механизм DataBinding





Другие варианты развития этого семейства паттернов

Presentation Model

Представляет собой логическое представление пользовательского интерфейса, не опираясь на какие-либо визуальные элементы.

Предоставляет данные из модели для отображения на экране

Хранит состояние пользовательского интерфейса