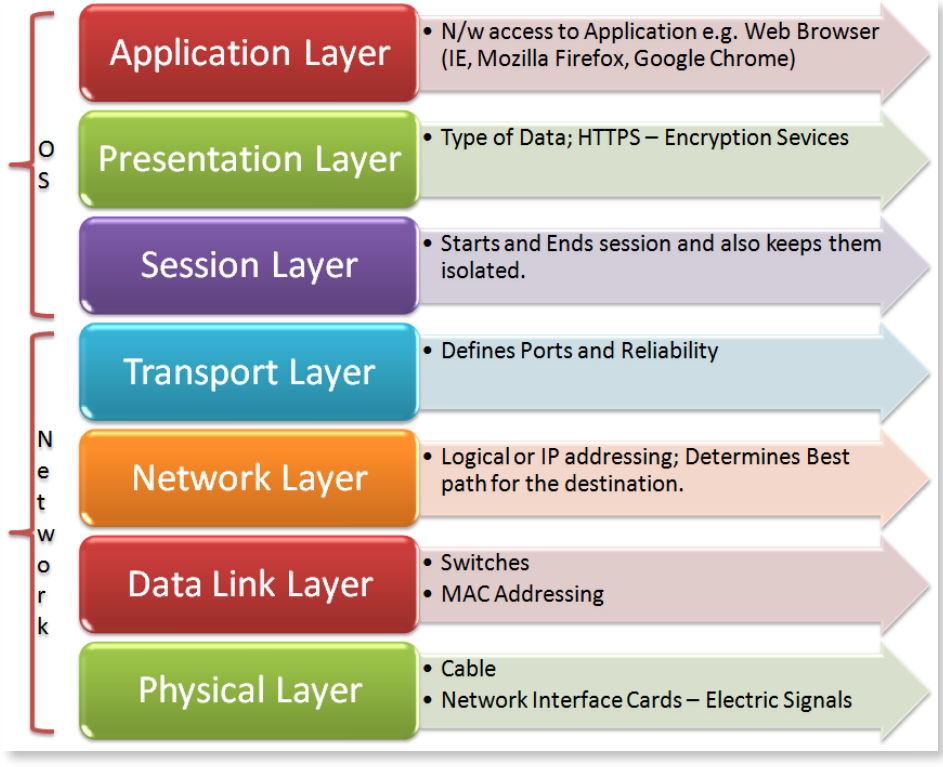
HTTP

REST

MVC

Модель OSI


O
S
N
e
t
w
o
r
k
Application Layer
• N/w access to Application e.g. Web Browser (IE, Mozilla Firefox, Google Chrome)
Presentation Layer
• Type of Data; HTTPS – Encryption Sevices
Session Layer
• Starts and Ends session and also keeps them isolated.
Transport Layer
• Defines Ports and Reliability
Network Layer
• Logical or IP addressing; Determines Best path for the destination.
Data Link Layer
• Switches
• MAC Addressing
Physical Layer
• Cable
• Network Interface Cards – Electric Signals

HTTP 1.0/1.1

— HyperText Transfer Protocol

— Протокол "Запрос/Ответ" (Request/Response)

— Текстовый (HTTP/2 - уже бинарный)

— Не хранит состояние (stateless)

Запрос HTTP

```
<Метод HTTP> <URI> HTTP/<версия>
<Имя заголовка>: <значение заголовка>
<Имя заголовка>: <значение заголовка>
. . .
(пустая строка)
<тело запроса>
```

```
GET /article?id=12 HTTP/1.1
Host: mysite.com
```

```
POST /contact_form.php HTTP/1.1
Host: developer.mozilla.org
Content-Length: 64
Content-Type: application/x-www-form-urlencoded

name=Joe%20User&request=Send%20me%20one%20of%20your%20catalogue
```

```
PUT /data HTTP/1.1
Host: mysite.com
Content-Length: 65
Content-Type: application/json

{"key1":"value1","key2":"value2","number1":1234,"arr":[1,2,"a"]}
```

Ответ HTTP

```
HTTP/<версия> <Код ответа> <Описание кода>
<Имя заголовка>: <значение заголовка>
<Имя заголовка>: <значение заголовка>
. . .
(пустая строка)
<тело ответа>
```

```
HTTP/1.1 200 OK
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (29769 bytes of the requested web page)
```

```
HTTP/1.1 404 Not Found
Content-Length: 10732
Content-Type: text/html

<!DOCTYPE html... (site-customized 404 page)
```

HTTP 1.0 vs 1.1

— HTTP 1.1 поддерживает virtual hosts (заголовок `Host`)

— HTTP 1.1 поддерживает keep-alive соединения (`Connection: keep-alive`)

Методы HTTP (verbs)

— `Safe` - не изменяет состояние сервера

— `Idempotent` - повторный запрос будет иметь точно такой же результат

— `Cacheable` - может ли браузер закешировать ответ

Метод	Описание	Safe	Idempotent	Cacheable
GET	Получить значение ресурса	✔	✔	✔
HEAD	Получить те же заголовки что выдал бы GET, но без тела ответа	✔	✔	✔
POST	Послать данные на сервер	-	-	-
PUT	Создать или полностью обновить данные ресурса	-	✔	-
PATCH	Обновить часть данных ресурса	-	-	-
DELETE	Удалить ресурс	-	✔	-

Cookies

— `Cookie` - небольшой кусок данных, посылаемый сервером при ответе на HTTP запрос

— Браузер посылает сохраненные cookies со следующими запросами к серверу

— Используются для управления сессиями, персонализации, трекинга поведения пользователя

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; HttpOnly
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry
```

```
GET /sample_page.html HTTP/1.1
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry; id=a3fWa
```

`localStorage` & `sessionStorage`

— `localStorage` - хранилище данных в формате "ключ-значение". Данные хранятся неограниченно долго

— `sessionStorage` - то же что и `localStorage`, но данные хранятся на протяжении сессии

```
localStorage.setItem("key", "value");
let data = localStorage.getItem("key");
localStorage.removeItem("key");
localStorage.clear();
```

Cookies vs `localStorage` vs `sessionStorage`

	Cookies	localStorage	sessionStorage
Кто устанавливает	Сервер (клиент)	Клиент	Клиент
Кто читает	Сервер (клиент)	Клиент	Клиент
Размер хранилища	~180 * 4Kb	~5-10Mb	~5-10Mb
Срок жизни	Настраивается	Вечно	Сессия
Что хранить?	Данные для сервера	Любые некритичные данные для клиента	Любые некритичные данные для клиента

RESTful API

— Стиль написания серверного API, использующий *семантику* HTTP

— Клиент-Сервер: REST подразумевает клиент-серверную архитектуру

— `Stateless`: сервер не сохраняет никакого состояния между двумя запросами. Каждый запрос содержит полную информацию о нужном действии.

— `Cacheable`: сервер сообщает клиенту информацию о возможности кеширования

```
POST /article-add.php

<article info>

GET /delete-article.php?id=17&action=DELETE

POST /deleteMyComment/13?userId=4
```

RESTful API

— URL представляет *ресурс* - объект системы (необязательно соответствующий данным в БД)

— Метод HTTP описывает действие над ресурсом

— Код ответа сервера описывает выполненное действие

— `Content-type` задает формат данных (обычно JSON/XML)

RESTful API

Запрос	Действие	Код ответа
`GET /articles`	Получить список статей	200 OK
`GET /articles?limit=10&offset=20`	Получить список статей	200 OK
`GET /articles/count`	Получить общее число статей	200 OK
`POST /articles`	Добавить новую статью	200 OK/201 Created
`GET /articles/15`	Получить статью	200 OK
`HEAD /articles/15`	Проверить существование	200 OK
`PUT /articles/15`	Обновить статью	200 OK/204 No Content
`PATCH /articles/15`	Обновить часть статьи	200 OK/204 No Content
`DELETE /articles/15`	Удалить статью	200 OK/202 Accepted/204 No Content

RESTful API

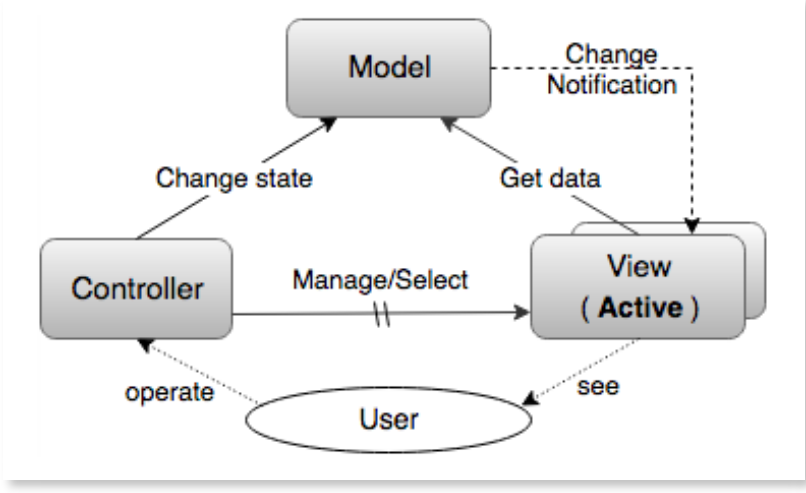
```
GET http://oursite.com/v1/users.json
```

```
{
    "data": [
        { id: 1, name: "...", ... },
        { id: 4, name: "...", ... },
        ...
    ],
    "metadata": {
        "code": 200,
        "codeDescr": "OK",
        "count": 3076,
        "user_url": "v1/users/{id}",
        "user_avatar": "v1/users/{id}/avatar"
    }
}
```

MVC

— `MVC` - архитектурный паттерн для приложений с UI

— Разделяет компонент на три взаимосвязанных части



Проблемы в понимании MVC

— Огромное число вариантов реализации (стрелочки на схеме можно нарисовать почти как угодно!)

— Отсюда появляются дополнительные названия `MVP/MVVM/MVA/Model2/MV*`

— Искаженные или упрощенные описания MVC в литературе

— Отсюда - огромное число заблуждений и неверных трактовок MVC

— Следствие: описывать MVC как *паттерн* с жесткой схемой нет смысла

MVC

— `MVC` - набор архитектурных идей/принципов/подходов

— Система должна разбиваться на модули, *слабо связанные друг с другом*

— Шаг 1: разбиение системы на бизнес-логику (Domain Model) и UI (M-VC)

— Шаг 2: Model реализует паттерн `Observer` (но не всегда)

— Шаг 3: UI разбивается на Controller и View (но это не обязательно)

Схема приложения без архитектуры

M-VC

— Шаг 1: разбиение системы на бизнес-логику (Domain Model) и UI (M-VC)

— Model и UI - разные области и могут разрабатываться отдельно

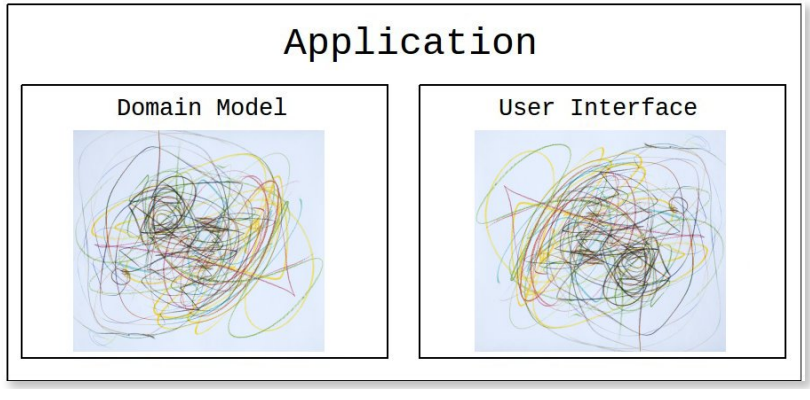
— Можно использовать несколько разных View с одной и той же Model

— Model можно покрыть Unit-тестами (а не e2e тестами)

— Model - **это не просто данные!!!** это данные и бизнес-логика

— Model не должна содержать логику и данные UI

— UI не должен содержать ни капли бизнес-логики



M-VC

— Шаг 2: для слабой связанности Model реализует паттерн `Observer`

— Модель рассылает извещения об изменениях

— UI подписывается на изменения и узнаёт когда нужно обновиться

— Модель ничего не знает о UI, и даже о его существовании

```
// Без Observer
class ToDoModel {
    constructor(view) {
        this.view = view;
        this.list = [];
    }
    getList() {
        return list;
    }
    add(text) {
        this.list.push({ text, complete: false });
        this.view.notify();
        // А что если нам нужно несколько View?
        // А что если у нас пока что нет ни одного View?
    }
}
```

Observer

```
// Простейший observer
class Subject {
  constructor() {
    this.observers = [];
  }
  add(item) {
    this.observers.push(item);
  }
  removeAll() {
    this.observers = [];
  }
  notifyObservers() {
    for (elem of this.observers) {
      elem.notify();
    }
  }
}
```

```
class ToDoModel {
  constructor() {
    this.subject = new Subject();
    this.list = [];
  }
  getList() { return this.list; }
  add(text) {
    this.list.push({
      text, complete: false
    });
    this.subject.notifyObservers();
  }
  complete: function(index, isComplete) {
    this.list[index].complete = !!isComplete;
    this.subject.notifyObservers();
  }
  // observer
  register(...args) {
    for (const elem of args) {
      this.subject.add(elem);
    }
  }
}
```

M-V-C

— Шаг 3: UI разбивается на Controller и View

— Вид отображает модель

— Контроллер реагирует на действия пользователя и вызывает методы модели

— Второй уровень декомпозиции, почти всегда пропускается в FE-фреймворках