SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

Design Patterns. MVC

SWIFTACADEMY

НИЕ ВЯРВАМЕ ВЪВ ВАШЕТО БЪДЕЩЕ

Front-End Development

Wikipedia:

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations.

Application Architecture
Architecture Patterns

- Под архитектура на дадено приложение (на деден софтуерен продукт, като уебсайт например) имаме предвид физическото разпределение на файловете, от които е изградено приложението, като те са групирани според своите отговорности и предназначение

- Например: това да слагаме css файловете в папка styles, javascript файловете - в папка js и картинките в папка img, е вид архитектурно решение

- С времето в уеб програмирането се утвърдиха няколко архитектурни модела, от които най-популярен е MVC


WHAT? MVC?..
EVERYBODY KNOWS MVC
imgflip.com
quickmeme.com

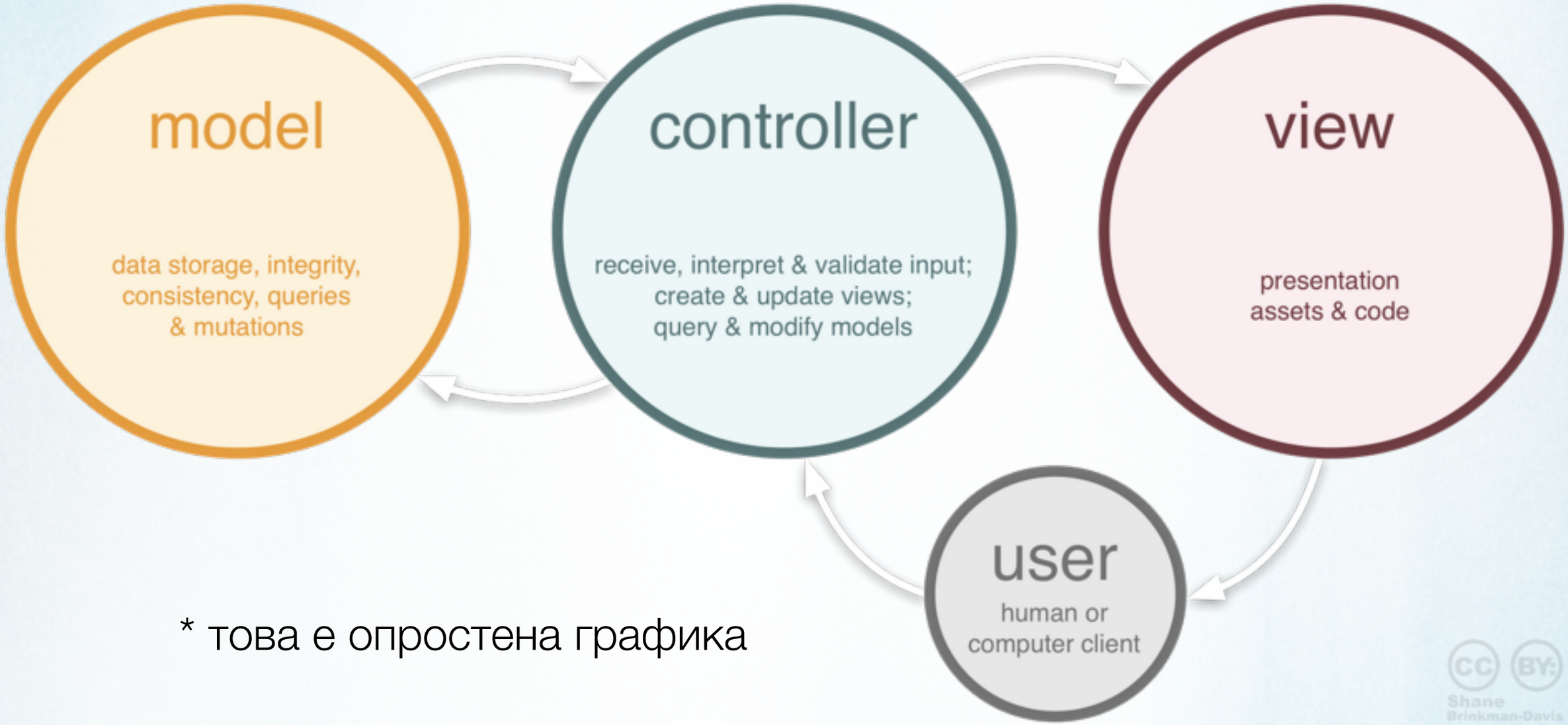
Model View Controller

- Това е архитектурно решение, което масово се наложи в сферата на уеб разработката, порди факта че беше заложен в от някои от най-успешните и широко-използвани през последните години web frameworks (Spring, Rails, Django)
- Основната идея в този pattern е, че нашият апликейшън (уебсайт) се дели основно на 3 части: Model, View и Controller

- **Модела** (model от MVC) е този компонент, който отговаря за физическите данни, които съхраняваме в нашата база от данни (потребителски профили, продукти, имена, адреси, кредитни карти и изобщо цялата информация, с която оперира нашият уебсайт)

- **View** е компонента, който е отговорен за графичното представяне и визуализиране пред потребителя на онова, което е в базата

- **Контролер**-а е свързващият компонент между view-то и модела. Той получава заявки от потребителя (GET), взима данните от модела и ги подава на view-то, за да може да се визуализират. Когато потребителя въвежда информация (POST), тя се изпраща към контролера под формата на параметри и съответно контролера ъпдейт-ва модела (т.е. базата), след което връща резултата в ново view



* това е опростена графика

Други архитектурни модели

- MVP (model view presenter) - тук по-голямата част от логиката е във фронт-енда, докато при MVC модела, цялата плогика е в модела и контролера, които пък са част от бекенд-а

- Presentation model (Model-View-ViewModel) - концепция развита от Windows Presentation Foundation, за разработка на windows графични и уеб приложения на .Net

- http://stackoverflow.com/questions/2056/what-are-mvp-and-mvc-and-what-is-the-difference

API

- Application Program Interface

- За какво служи ?

    - представете си база от данни, която съдържа чувствителна информация (банкови данни, потребителски профили, локации и т.н.), но която да е широко използвана (банки, платежни оператори, социални мрежи, карти)

    - Обикновенно собственикът на тази база от данни изгражда свое собствено софтуерно приложение, което работи с базата (чете и пише в нея)

    - Ако този собственик реши, че иска да даде възможност на външни приложения да оперират с неговата база от данни, той изгражда API - server-side приложение, което играе роля на посредник между базата данни и околния свят

    - Т.е. чрез API-а се дава възможност на всеки (който е ауторизиран) да оперира с данните в базата, но по определен начин, така че да не може да направи поразии

API

- Как се ползва ?
    - Когато някой прави API към свойта база от данни, той трябва да създаде функционалността, която да борави директно с данните от базата и след това да опише как тази функционалност може да се ползва от външните приложения
    - Външните приложения могат единствено да правят http заявки към API-а, чрез които да получават данни (GET) от базата или да я ъпдейтват (POST). След всяка заявка към API-а получават http response като отговор, който трябва да съдържа резултата от изпълнението на заявката
    - В най-честия случай външните приложения ползват ключ или парола, които да ги автентикират пред API-а (нещо като face control). Ако някой направи заявка без да се автентикира или се автентикира грешно, API-а ще му върне статус 403 и няма да изпълни заявката

Популярни APIs

- Google Maps API: https://developers.google.com/maps/
- YouTube APIs: https://developers.google.com/youtube/
- Facebook API: https://developers.facebook.com/
- http://www.programmableweb.com/news/what-exactly-api/analysis/2015/12/03


AJAX
HISTORY.COM
imgflip.com

AJAX

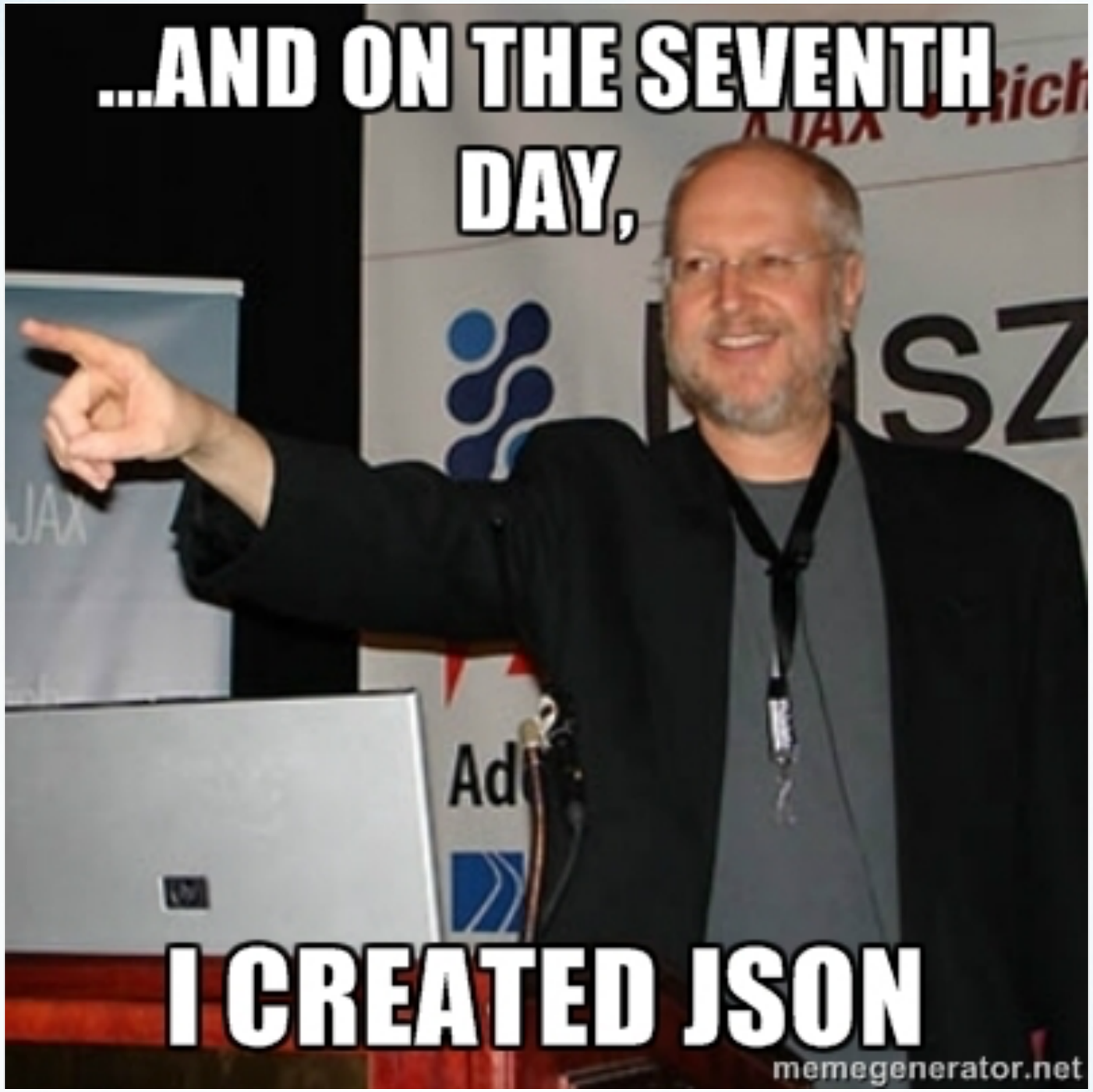
- Asynchronous JavaScript and XML
- Това е JavaScript функционалност, която позволява асинхронни заявки (тези, които се случват незабелязано от потребителя) от уеб приложението към сървъра
- Въпреки че е базирана на JavaScript, Ajax функционалността е придобила самостоятелно значение и идентичност и често се говори за нея като за отделна технология
- AJAX използва JSON нотацията или XML форматът за да предаде данните към сървъра

```
// This is the client-side script.

// Initialize the Http request.
var xhr = new XMLHttpRequest();
xhr.open('get', 'send-ajax-data.php');

// Track the state changes of the request.
xhr.onreadystatechange = function () {
    var DONE = 4; // readyState 4 means the request is done.
    var OK = 200; // status 200 is a successful return.
    if (xhr.readyState === DONE) {
        if (xhr.status === OK) {
            alert(xhr.responseText); // 'This is the returned text.'
        } else {
            alert('Error: ' + xhr.status); // An error occurred.
        }
    }
};

// Send the request to send-ajax-data.php
xhr.send(null);
```

AJAX Frameworks

- С чист Javascript е трудно, затова препоръчвам да се ползва фреймуърк за Ajax
- http://api.jquery.com/jquery.ajax/

```
1 | $.ajax({
2 |   method: "POST",
3 |   url: "some.php",
4 |   data: { name: "John", location: "Boston" }
5 | })
6 |   .done(function( msg ) {
7 |     alert( "Data Saved: " + msg );
8 |   });
```


...AND ON THE SEVENTH DAY,
I CREATED JSON
memegenerator.net

JSON

- Javascript Object Notation
- Това е един много опростен начин за представяне на данни
- Принципа е асоциативен списък:

```
{ param_name1: param_value1,
  param_name2: param_value2,
  ...
}
```

Въпроси?

Направете html регистрационна форма, която да има следните полета:
username, password, password_confirmation, email, avatar_url
и която да събмитва към http://zenifytheweb.com/api/examples/register

Използвайте Ajax за да направите асинхронен рикуест при blur на username полето към check_username ендпойнт-а на API-то

Използвайте отговора на API-то за да визуализирате резултата

Ето я и спецификацията на API-то:

API url: http://zenifytheweb.com/api/examples

Endpoint: /check_username, **method**: Post, **Params**: username (String)

Endpoint: /register, **method**: Post, **Params**: username, password,
password_confirmation, email, avatar_url

Returns on success: 200 OK, "The success message here"

Returns on error: 510, {error: "The error message here."}

За домашно

- Довършете задачата от предния слайд
- Направете така, че вместо да се събмитва формата, когато потребителя натисне бутона "register", да се правит Ajax request
- Използвайте Javascript (или Jquery) , за да покажете резултата на потребителя (ако резултата е успешен, регистрационната форма трябва да се скрива)
- Подгответе си 1 въпрос върху Ajax (или нещо друго от материала досега) за следващият път, тъй като ще правим подготовка за контролното

Примери

http://zenifytheweb.com/courses/lessons/lesson17/example/index.html

Домашно

https://github.com/zzeni/swift-academy-homeworks/tree/master/tasks/L17