WEB PROGRAMMING 0601632 2

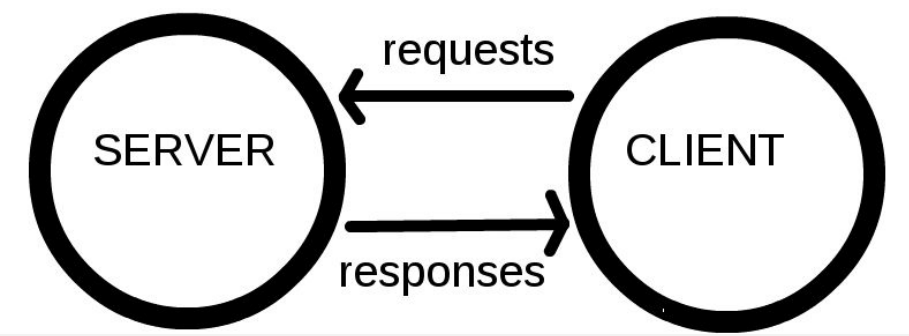
BY DR BUNDIT THANASOPON

HOW THE WEB WORKS

SRC: HTTPS://DEVELOPER.MOZILLA.ORG/EN-US/DOCS/LEARN/GETTING_STARTED_WITH_THE_WEB/HOW_THE_WEB_WORKS

CLIENTS AND SERVERS

- Computers connected to the web are called **clients** and **servers**. A simplified diagram of how they interact might look like this:



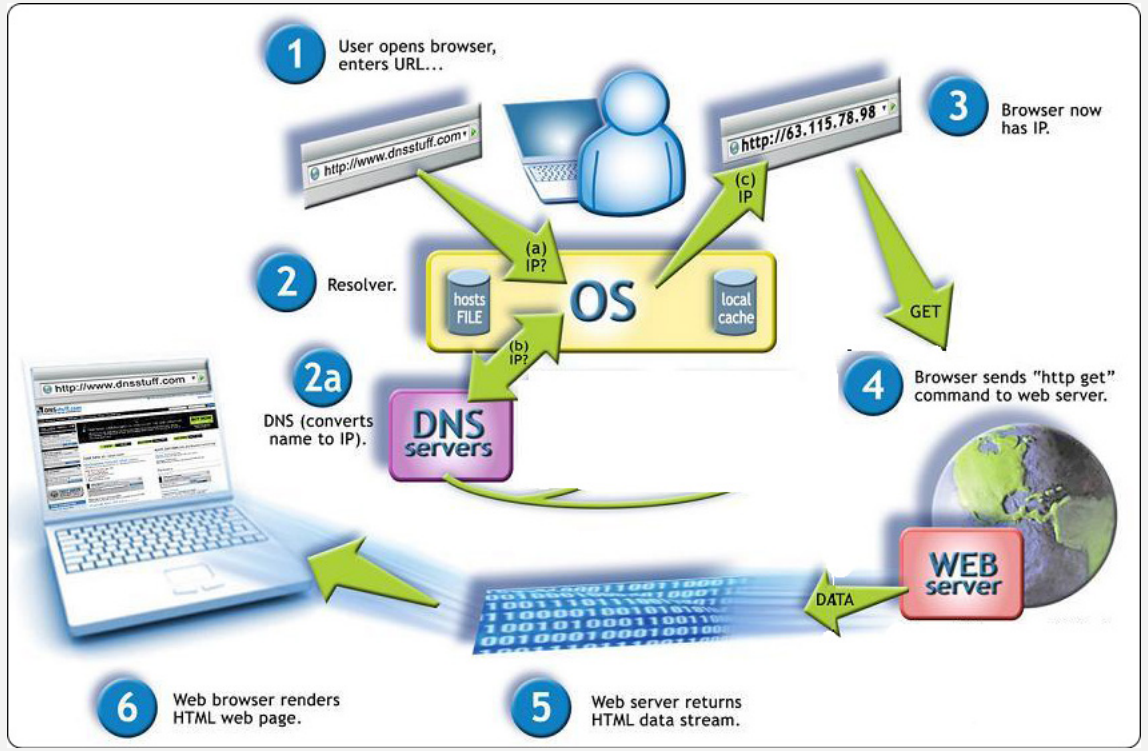
- **Clients** are the typical web user's internet-connected devices.

- **Servers** are computers that store webpages, sites, or apps.

OTHER IMPORTANT COMPONENTS

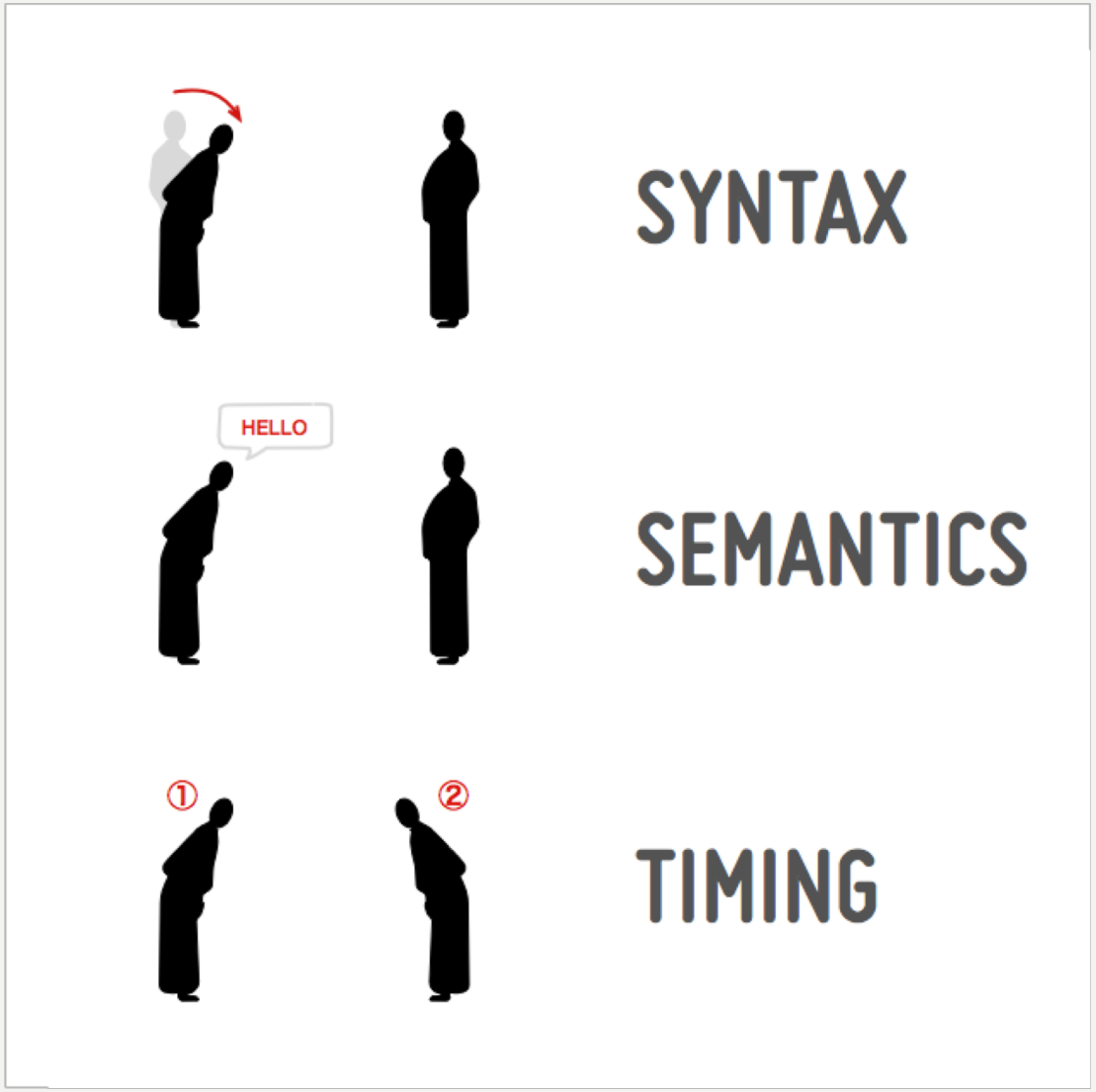
- **Your internet connection**: Allows you to send and receive data on the web.
- **TCP/IP**: Transmission Control Protocol and Internet Protocol are communication protocols that define how data should travel across the web.
- **DNS**: Domain Name Servers are like an address book for websites.
- **HTTP:** Hypertext Transfer Protocol is an application protocol that defines a language for clients and servers to speak to each other.
- **Component files**: A website is made up of many different files.
  - **Code files -** HTML, CSS, and JavaScript
  - **Assets -** images, music, video, Word documents, and PDFs

DOMAIN NAME SERVERS - DNS



Src: http://www.gargasz.info/how-internet-works-dns/

OVERVIEW OF HTTP

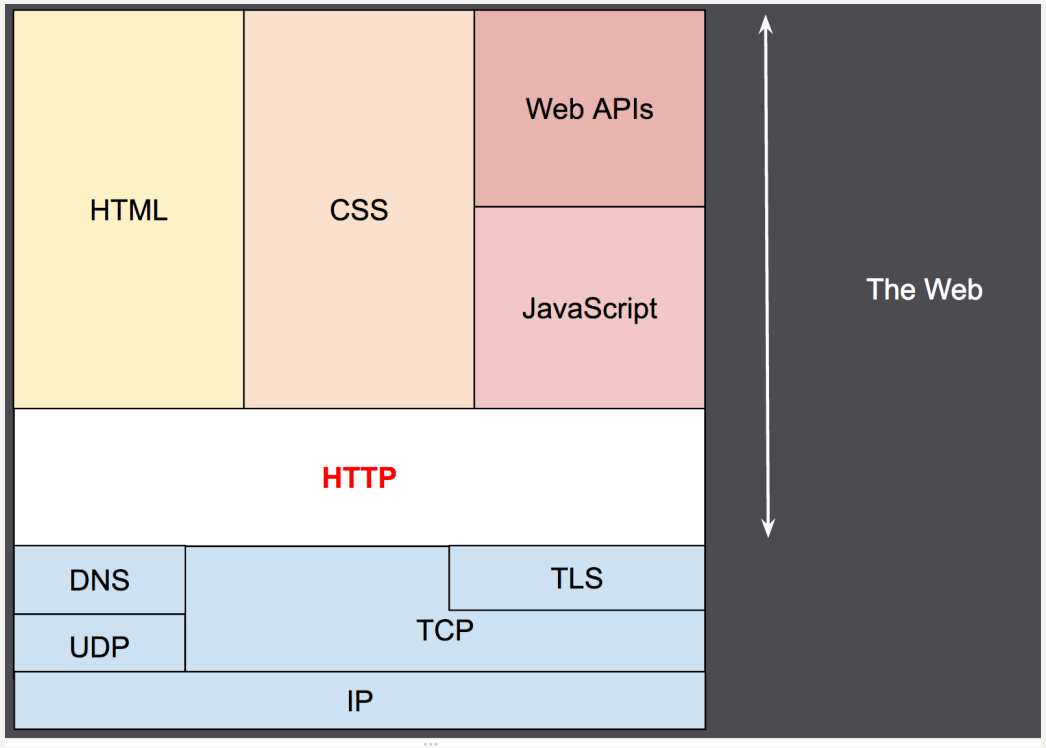


WHAT IS A COMMUNICATION PROTOCOL?

To be able to communicate, two parties (be they software, devices, people, etc.) need:

HTTP

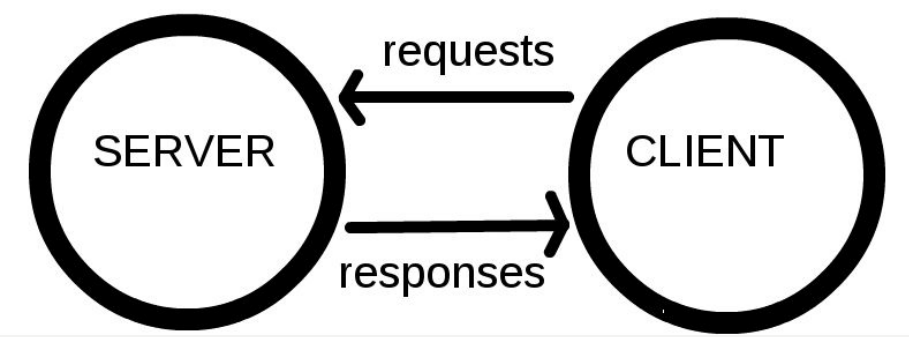
- **HTTP** is a protocol which allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and a client-server protocol.
- It is an application layer protocol that is sent over TCP, or over a TLS-encrypted TCP connection



COMPONENTS OF HTTP-BASED SYSTEMS

- HTTP is a client-server protocol: **requests** are sent by one entity, the **user-agent** -> a Web browser

- Each individual request is sent to a **server**, which will handle it and provide an answer, called the ***response***.



CLIENT: THE USER-AGENT

- The ***user-agent*** is any tool that acts on the behalf of the user.

- The browser is **always** the entity initiating the **HTTP request**. It is never the server (though some mechanisms have been added over the years to simulate server-initiated messages).

- To present a Web page, the browser sends an original request to fetch the HTML document from the page.
- It then parses this file, fetching additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (usually images and videos).
- The Web browser then mixes these resources to present to the user a complete document, the Web page.

THE WEB SERVER

- On the opposite side of the communication channel, is the server which *serves* the document as requested by the client.

- A server presents only as a single machine virtually: this is because it may actually be a collection of servers:
  - Load balancers
  - Database servers
  - E-commerce servers

BASIC ASPECTS OF HTTP

- **HTTP is simple** - HTTP messages can be read and understood by humans, providing easier developer testing, and reduced complexity for new-comers.

- **HTTP is stateless, but not sessionless** - HTTP is stateless: there is no link between two requests being successively carried out on the same connection. If so:
  - How shopping basket on e-commerce websites work?
  - How websites maintain logged-in states?

- **HTTP Cookies** allow session creation on each HTTP request to share the same context, or the same state.

HTTP FLOW

- When the client wants to communicate with a server, either being the final server or an intermediate proxy, it performs the following steps:
  - Open a TCP connection
  - Send an HTTP message

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

  - Read the response sent by the server

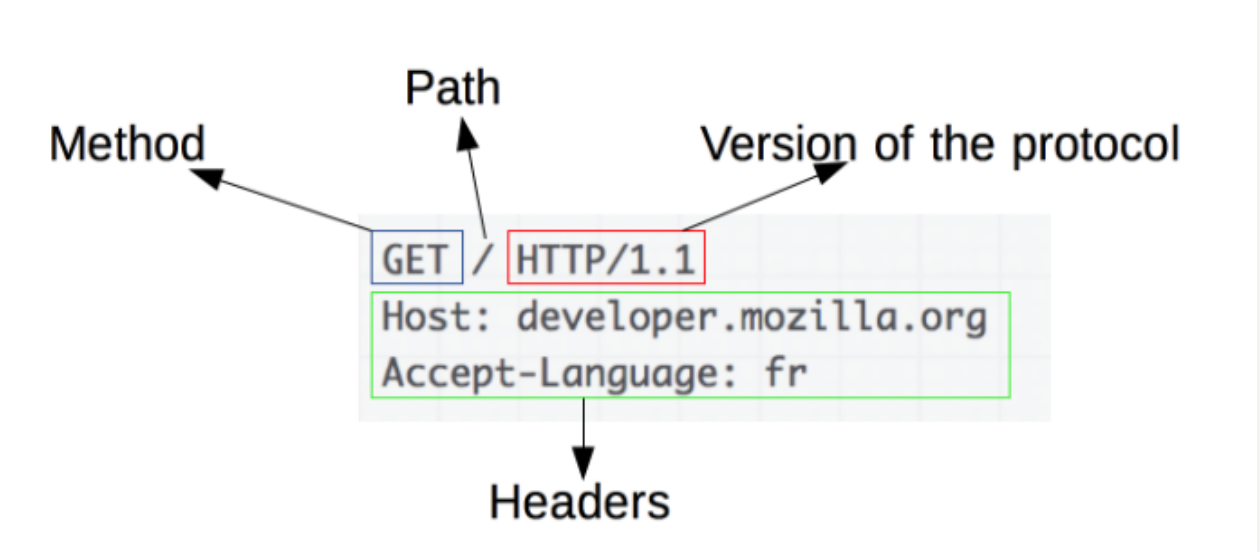
```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

  - Close or reuse the connection for further requests.

HTTP MESSAGES

- HTTP/1.1 and earlier HTTP messages are human-readable. In HTTP/2, these messages are embedded into a new binary structure, a frame, allowing optimizations like compression of headers and multiplexing.

- There are two types of HTTP messages, **requests and responses**, each with its own format.
- An example HTTP request:

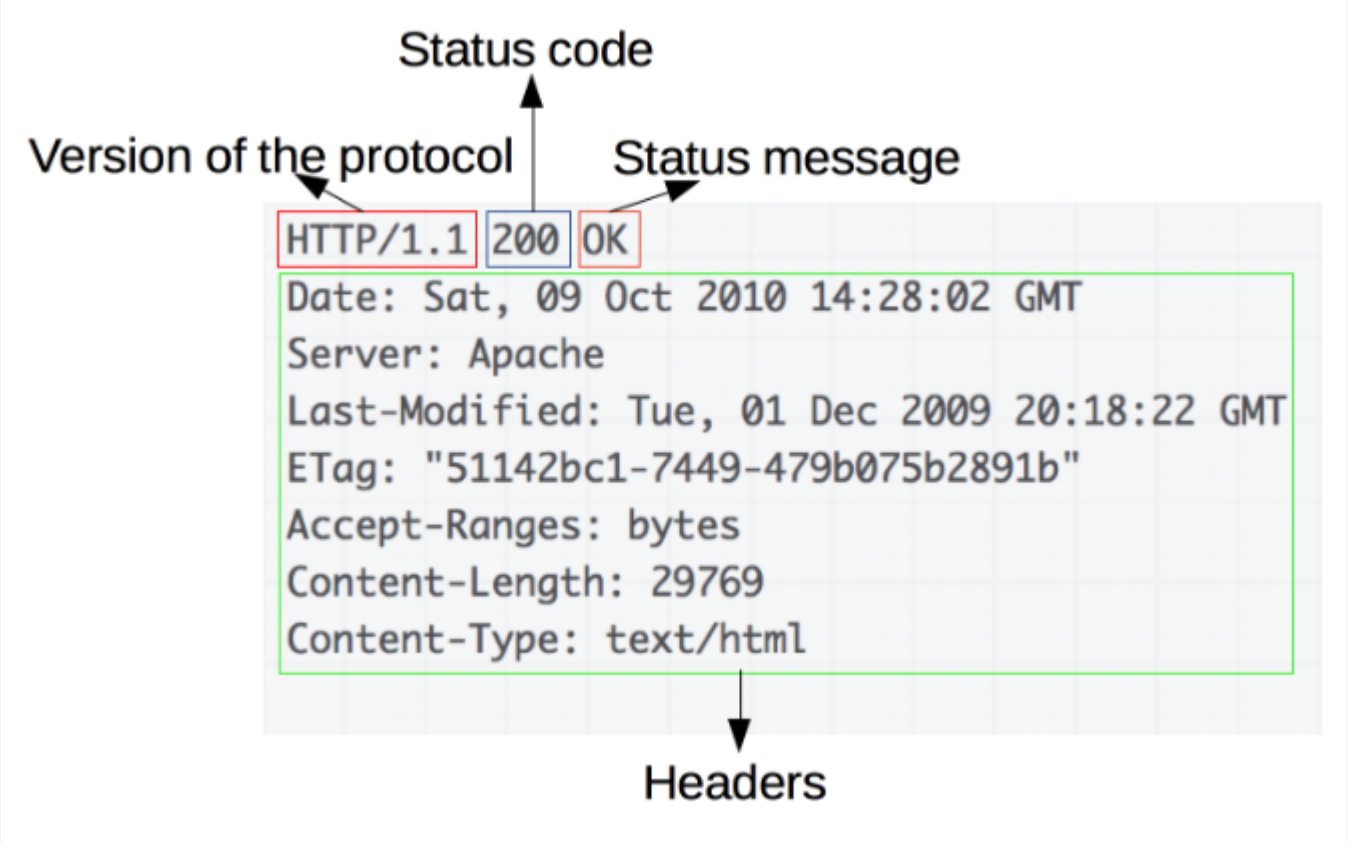


HTTP REQUESTS

- Requests consists of the following elements:
  - An HTTP method, usually a verb like GET, POST or a noun like OPTIONS or HEAD that defines the operation the client wants to perform.
    - Typically, a client wants to fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in other cases.
  - The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context.
  - The version of the HTTP protocol.
  - Optional headers that convey additional information for the servers.
  - Or a body, for some methods like POST, similar to those in responses, which contain the resource sent.

HTTP RESPONSES

- An example response:



HTTP RESPONSES

- Responses consist of the following elements:
  - The version of the HTTP protocol they follow.
  - A status code, indicating if the request has been successful, or not, and why.
  - A status message, a non-authoritative short description of the status code.
  - HTTP headers, like those for requests.
  - Optionally, a body containing the fetched resource.

MVC CONCEPTS

SRC: HTTPS://DEVELOPER.MOZILLA.ORG/EN-US/DOCS/WEB/APPS/FUNDAMENTALS/MODERN_WEB_APP_ARCHITECTURE/MVC_ARCHITECTURE

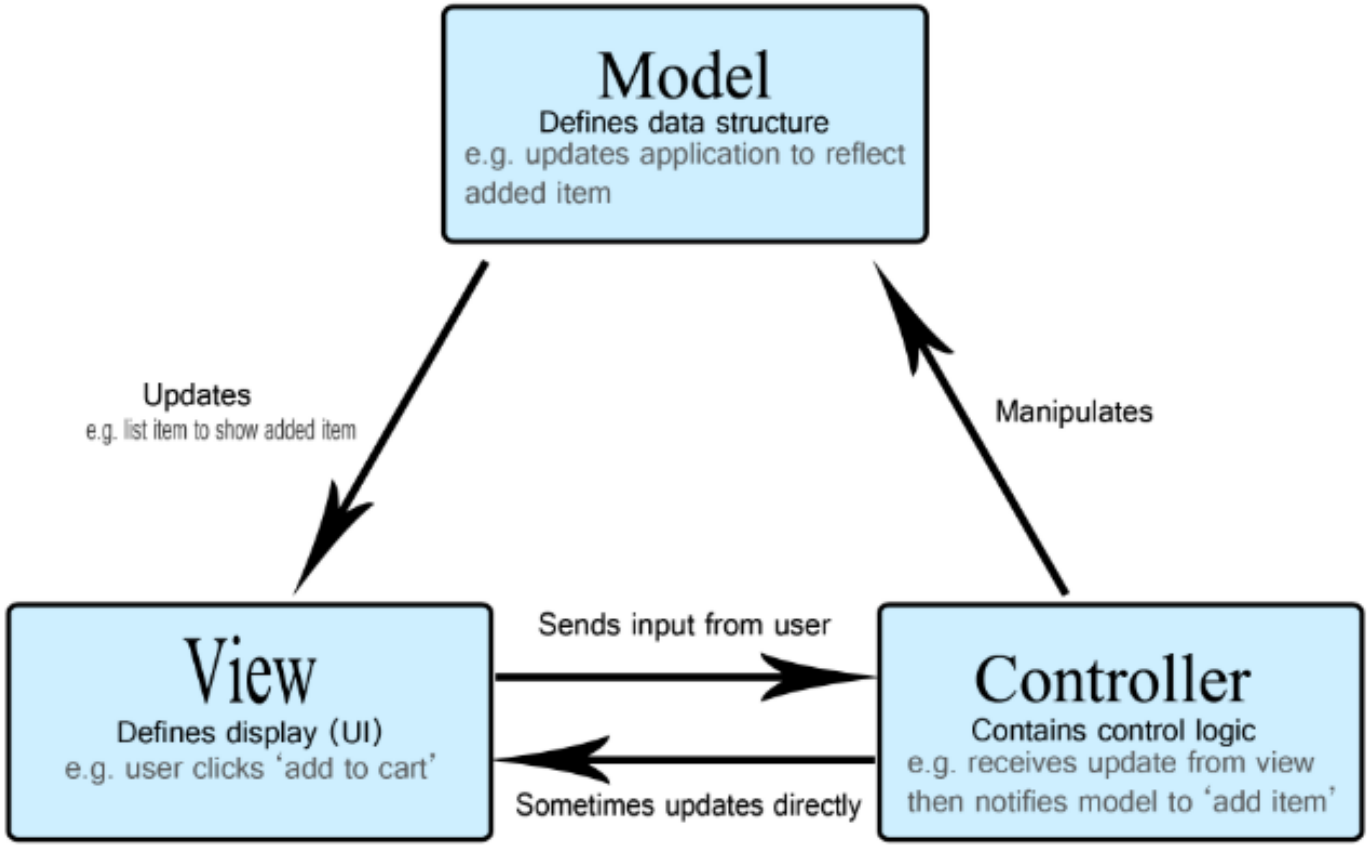
THE THEORY BEHIND MODEL VIEW CONTROLLER

- Model View Controller (MVC) is a software architecture pattern, commonly used to implement user interfaces: it is therefore a popular choice for architecting web apps.

- In general, it separates out the application logic into three separate parts
  - Models
  - Views
  - Controllers

MVC EXAMPLE

- To make this a little more clear, let's imagine a simple shopping list app. All we want is a list of the name, quantity and price of each item we need to buy this week.



MVC COMPONENTS

- **Model:** Model code typically reflects real-world things. This code can hold raw data, or it will define the essential components of your app.
  - For instance, if you were building a To-do app, the model code would define what a "task" is and what a "list" is

- **View:** View code is made up of all the functions that directly interact with the user. This is the code that makes your app look nice, and otherwise defines how your user sees and interacts with it.

- **Controller:** Controller code acts as a liaison between the Model and the View, receiving user input and deciding what to do with it. It's the brains of the application, and ties together the model and the view.

AN ANALOGY - COOKING

- MVC is a way to think about how an web application works.
- It's kind of like how you make Thanksgiving dinner. You have a fridge full of food, which is like the **Model**. The fridge (Model) contains the raw materials we will use to make dinner.
- You also probably have a recipe or two. A recipe (assuming you follow it exactly) is like the **Controller** of Thanksgiving dinner. Recipes dictate which stuff in the fridge you'll take out, how you'll put it together, and how long you need to cook it.
- Then, you have table-settings, silverware, etc., which are what your hungry friends and family use to eat dinner. Table-top items are like the **View**. They let your guests interact with your Model and Controller's creation.

FRONT-END VS. BACK-END

FRONT-END VS. BACK-END

- When we discuss the "**frontend**" of the web, what we're really talking about is the part of the web that you can see and interact with.
- The frontend usually consists of two parts: the **web design** and **front end web development**.
  - Web designers – those how can work strictly Photoshop and Fireworks
  - Front-end web developers – those who code using **HTML, CSS, JavaScript, jQuery, Vue, React**, etc.

- The **backend** usually consists of three parts: a server, an application, and a database.
- We call a person that builds all of this technology to work together a *backend developer*.
  - Backend technologies usually consist of languages like **PHP, Ruby, Python**, etc.

SHOULD YOU BE A BACK-END, FRONT-END OR FULL-STACK DEVELOPER?

- **Key front-end development skills**
  - have some artistic vision to present the data, UX and UI
  - mastering HTML, CSS, some CSS pre-processor like SAS, and some (mainstream) JavaScript frameworks such as Angular, React or Vue
  - have an understanding of event-based interaction, security, and performance.

- **Key back-end development skills**
  - Backend developers work implementing the business logic
  - have knowledge of frameworks, software architecture, design patterns, databases, APIs
  - be able to manage abstract concepts and complex logic
  - have a deep understanding of servers and databases (SQL or no SQL), API layer
  - Mastering program languages such as Java, python, PHP, C#, go and scala

COURSE SYLLABUS

- HTML & handing out assignment
- CSS & Bootstrap
- Javascript & DOM
- Vue.js basics (2 weeks)
- Assignment presentation
- !!! Mid-term exam !!!

COURSE SYLLABUS

- Python
- Django framework
- Models & Django admin
- Views & templates
- Forms & form validation
- Model forms
- Ajax & Django REST framework
- Assignment presentation
- !!! Final exam !!!

ASSIGNMENTS

- **Part 1 – front-end development & design (10 points)**
  - Requirements
  - UX & UI designs
  - Workable prototype using:
    - HTML
    - CSS
    - Javascript
    - Vue.js
- **Part 2 – back-end development (20 points)**
  - ER diagram and descriptions
  - Models
  - The final product