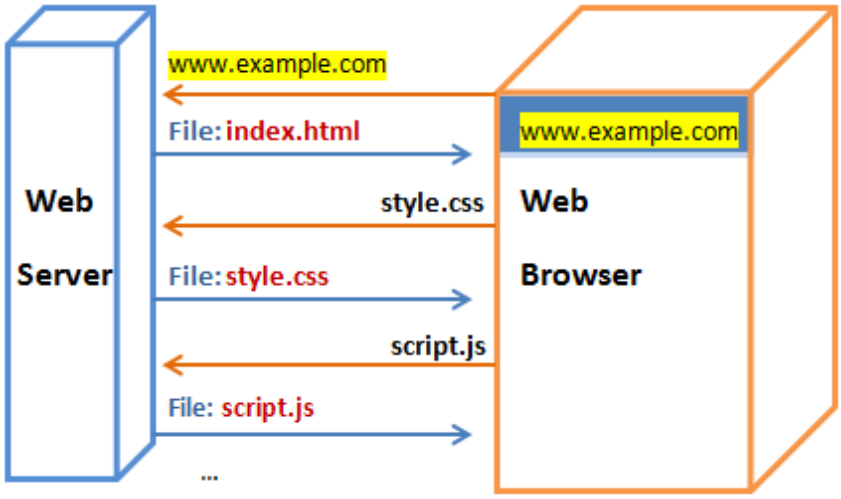
Introducción al Módulo Desarrollo Web en Entorno Cliente.

Tradicionalmente el modelo de Internet fue contar con servidores muy potentes que hacían todo el trabajo dinámico y generaban archivos html que eran enviados al navegador del usuario, y que lo único que hacia este era limitarse a mostrarlos.



Fue así durante mucho tiempo, porque los ordenadores de sobre mesa no tenían la memoria suficiente ni la capacidad de procesamiento para llevar a cabo las operaciones ni la generación de los datos que obligatoriamente tenían que hacerse en el servidor.

Esto esta cambiando continuamente ya que a medida que se mueven más datos en Internet también lo hacia la CPU y la RAM, esta situación es la que lleva a los programadores a que se modifique el modelo de trabajo aplicada a la Web.

Así aparecen Ember.js Angular.js Backbone.js y Meteor.js (2:37)

Todos tiene una característica común, su objetivo fundamental es mover todo o gran parte del trabajo desde el Servidor al Cliente.

Se trata de aplicaciones JavaScript lo que supone que la mayoría del código se ejecuta en el Navegador del Cliente, lo que incrementa las posibilidades de desarrollos interactivos. Esto se realiza utilizando las librerías: Ember, Angular, Backbone y Meteor por ejemplo. (Api, librería, framework).

Los Navegadores Web han dejado de ser simples marcos que servían para mostrar paginas mas o menos estáticas y se han convertido en verdaderas plataformas para ejecutar aplicaciones con todas las funcionalidades y en tiempos de ejecución cada vez más rápidos, gracias al uso de JavaScript y con el estándar HTML 5 que nos permiten crear aplicaciones ricas, que antes solo era posible hacer en plataformas nativas.

Lo fundamental en este nuevo tipo de desarrollo, consiste en reducir lo máximo posible la necesidad de estar interactuando entre el cliente y el servidor o incluso con otras capas añadidas como pueden ser las bases de datos, por eso una de las tendencias entre los desarrolladores hoy día es la de intentar construir aplicaciones enteras en el navegador, usando JavaScript, es lo que se denomina “Single Page Web Applications”, que son aplicaciones de una sola página que pueden responder inmediatamente a las

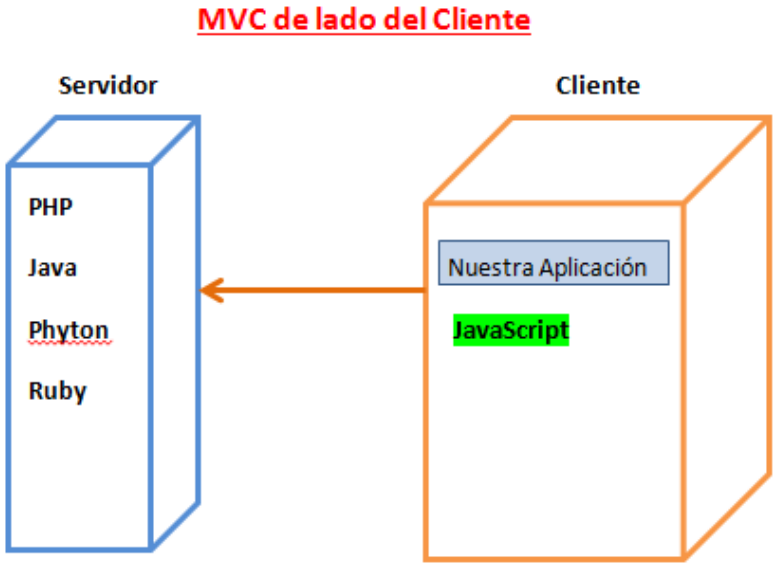
interacciones del usuario, ya que no tienen que estar haciendo un viaje de ida y vuelta al servidor cada vez que se cambie de página. Ej. Gmail

Esto es lo que buscan librerías JavaScript MVC.
"The top 10 JavaScript MVC Frameworks Reviewed" (Clasificacion)

Estas librerías se conocen como "Librerías MVC (Modelo Vista Controlador) de lado del **Cliente**". Hay desarrolladores que las llaman MVVM (Modelo Vista Vista Modelo) por el modo de tratar los datos.

La clave está en la palabra **cliente**.
Vamos a intentar reproducir la arquitectura MVC del lado del Cliente (la arquitectura típica que siguen las APIs Ember.js, angular.js, backbone.js y meteor.js).

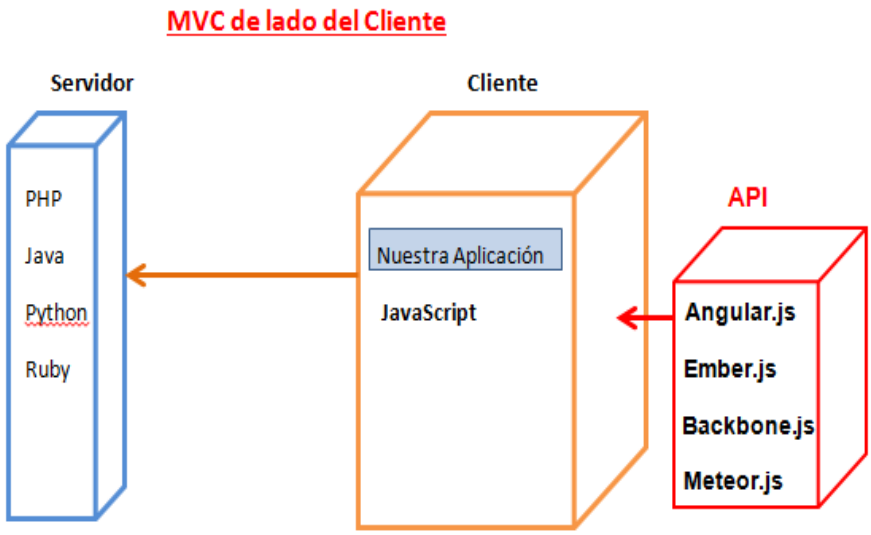
Evolucion del Marco de Trabajo.
El marco en el que se va a desarrollar la aplicación es un navegador y sobre este navegador es donde queremos que se ejecute nuestra aplicación Web. Esta aplicación Web necesita recibir la información desde fuera, desde el Servidor. Hasta aquí es lo mismo que lleva haciendo la Web desde que existe.



Hay un cliente que pide información y puede hacerlo con un lenguaje que no es de marcas, del lado del cliente llamado JavaScript. Esa petición se envía al servidor donde otro lenguaje de programación, conocido como lenguaje del lado del servidor (php, Java, Python, Ruby) se encarga de procesar la mayoría de la lógica y devolver una respuesta.

En los sistemas MVC del lado del servidor, solo la vista reside totalmente en el cliente, mientras que tanto el modelo como el controlador dependen del lenguaje del lado del servidor.

Con estas librerías (ember, angular,..) la mayor parte de la lógica de la aplicación (las vistas, las plantillas, los controles, los modelos, etc) van a pasar a residir en el cliente y la relación (peticiones), en lugar de hacerse con el servidor, se va a hacer con la API encargada de gestionar la lógica, por tanto, es algo que se hace en el cliente, en el Navegador.



La función del servidor con el lenguaje que sea, queda limitada a la de servir una página inicial (con el esqueleto html y los archivos necesarios), una vez que esos archivos son descargados al navegador, al cliente, son evaluados y se inicializa la aplicación del lado del cliente y que a partir de ese momento recuperara datos e interactuara desde la API, terminando de renderizar todo lo necesario para la página. En aplicaciones bien finalizadas, incluso puede trabajar offline, sin estar conectado al servidor (algo, cada vez más habitual).

Este nuevo paradigma, tiene ventajas:

- Para el usuario, porque una vez que la aplicación se carga por primera vez, va a hacer que la navegación sea mucho más rápida entre las paginas, sin necesidad de tener que estar contactando con el servidor
- Para los desarrolladores, porque la aplicación de esa única página que se envía desde el servidor (A Single Page App), hace que haya una clarísima separación entre las funcionalidades del servidor y las del cliente. Haciendo que el flujo de trabajo sea mucho más sencillo y mucho más claro, porque evita tener que compartir demasiada lógica entre el cliente y el servidor. Y además están escritos en diferentes lenguajes.

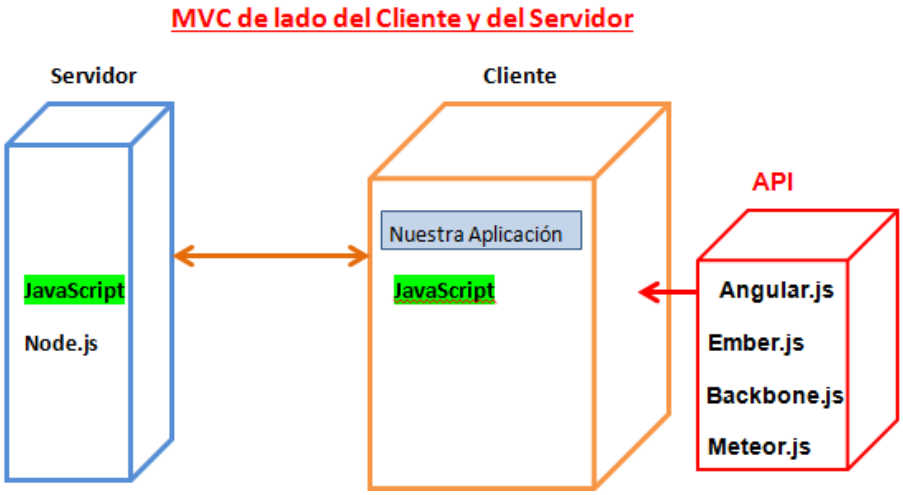
Por lo que necesitamos un código que pueda ser compatible con dos lenguajes, cliente y servidor. Entonces se empezó a trabajar con nuevas librerías que facilitaban esta comunicación (ember, angular,…). Pero aun así también aparecieron nuevos problemas, ya que la relación con el servidor se reduce muchísimo y la separación de funcionalidades cliente y servidor están claramente diferenciadas. Y se utilizan lenguajes diferentes para cada uno.

Siguiente paso en la Evolución.

Entonces, se llegó a la conclusión que para acabar con las inconsistencias, con las repeticiones de código y la repetición de lógica entre el cliente y servidor, lo ideal es crear aplicaciones a las que se denominan JavaScript Isomorfas.

Esto significa que se trata de aplicaciones que usan el mismo lenguaje de programación tanto del lado del cliente como del servidor, este lenguaje es JavaScript.

Una aplicación Isomorfa tendría la siguiente estructura:



Al tener el mismo lenguaje, parte de la lógica de la vista o de la lógica del servidor, pueden ser exactamente las mismas en el cliente y en el servidor. Ya que al estar escritas en el mismo lenguaje, no necesitan ningún tipo de traducción. Así, el mantenimiento de las aplicaciones será más fácil y también será mejor para el SEO (posicionamiento).

Esto se puede hacer gracias a Node.js en el lado del servidor.

Esta idea de JavaScript Isomorfo no es nueva ya que Google ya lo usaba hace tiempo. (2011)(Lo podemos ver en el artículo “Scaling Isomorphic JavaScript Code, nodejitsu”)

También tenemos que mencionar a “Mojito” el primer framework isomorfo de código abierto para el desarrollo de aplicaciones, su dependencia de “Yahoo” ha limitado su expansión entre los desarrolladores de JavaScript, para crear aplicaciones con Node.js en el cliente y servidor.

El proyecto Isomorfo que ha conseguido más resonancia es “meteor” para soportar aplicaciones en tiempo real. Mojito sigue el paradigma “Node.js” del lado del cliente y del servidor.

Esto es lo que se conoce como MVC JavaScript con meteor, también hay otros frameworks como Asana, Rendr (que utiliza Backbone).

Este en definitiva es el nuevo paradigma “JavaScript Isomorfo” utilizando el mismo lenguaje del lado del cliente y del lado del servidor.