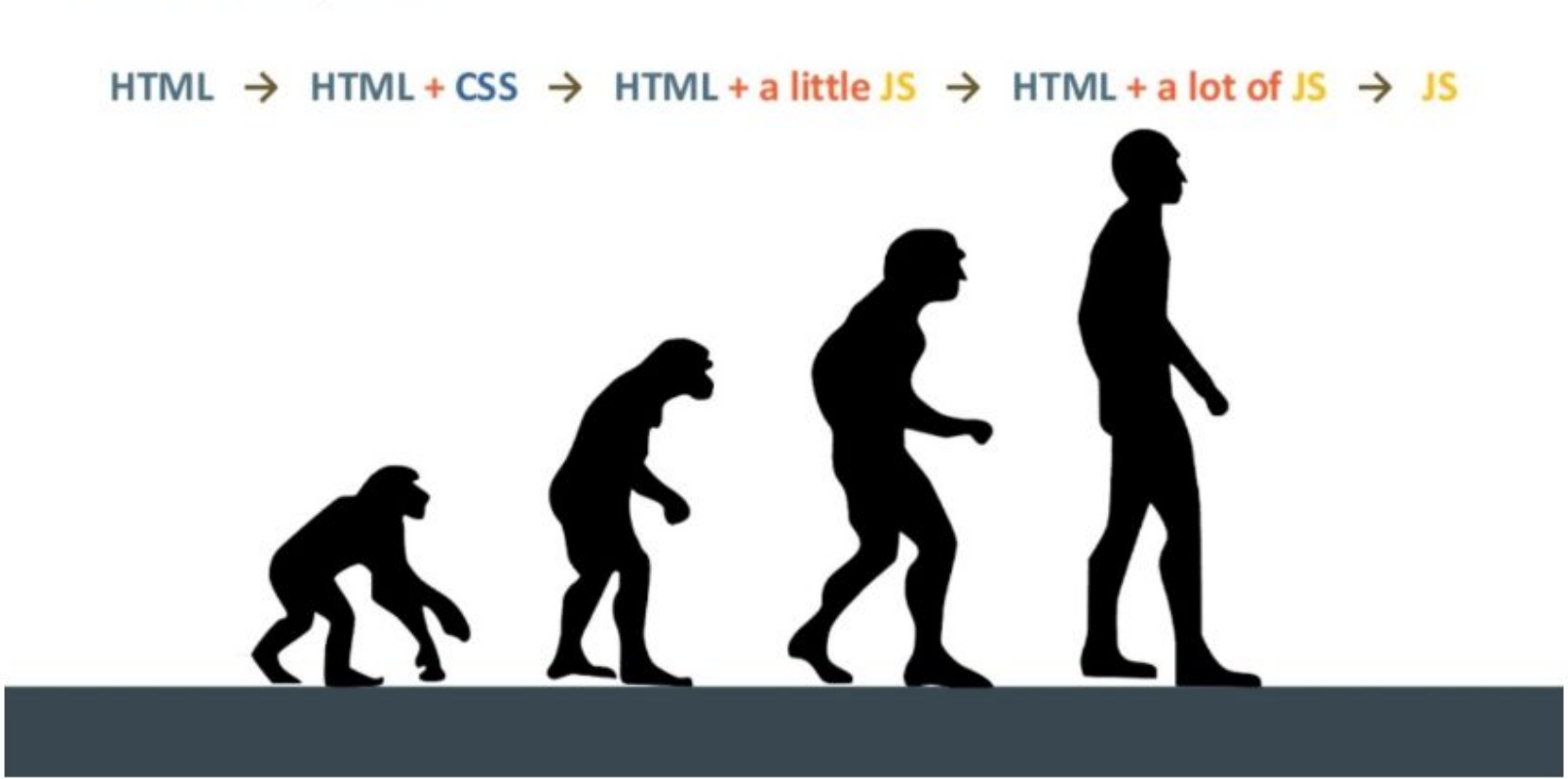
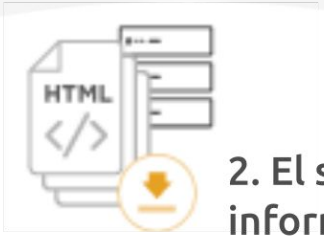
# SSR vs CSR

Server Side Rendering vs Client Side Rendering

# EVOLUCIÓN DE LA WEB






2. El servidor busca toda la información que necesita y genera el HTML completo


1. Sitio requerido por el usuario

3. El browser descarga la página y ya está lista para su interacción

**SERVER SIDE RENDERING (ENFOQUE TRADICIONAL)**


1. Sitio requerido por el usuario


2. El servidor responde muy rápido con un HTML básico vacio con referencias a JS files

3. El browser descarga el HTML + JS y la página puede comenzar a interactuar por ejemplo con una API REST


4. El servidor mediante su API REST busca y responde solo los datos solicitados (JSON)

5. El browser termina de renderizar la página y ya está lista para su interacción

**CLIENT SIDE RENDERING**

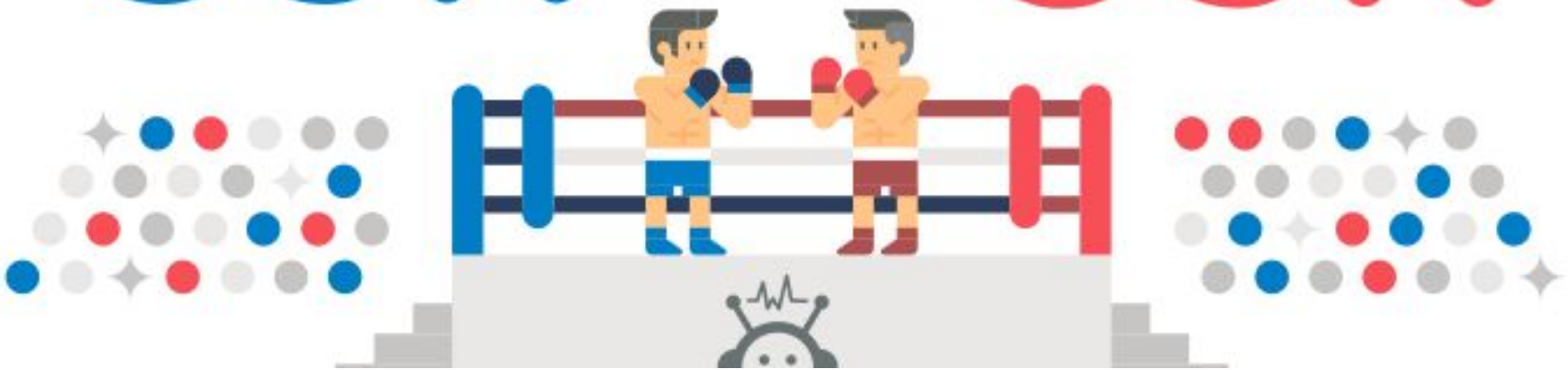
## Server Side Rendering (SSR)

Todos los recursos del sitio o app están alojados o son creados **dentro del servidor**. Cuando un usuario realiza una solicitud, el servidor envía el documento HTML final completo para que el browser lo muestre.

## Client Side Rendering (CSR)

En lugar de obtener todo el contenido del documento HTML desde el servidor, se recibe un documento HTML básico vacío con un **archivo JS que completa el render** del sitio del lado del cliente.

# SSR VS CSR



# SSR

- Optimizado para SEO
- Página inicial carga más rápido
- Buena para sitios semi-estáticos con baja interacción

**VENTAJAS**

- La página completa se recarga
- Mayor transferencia por solicitud
- Pobre UX

**DESVENTAJAS**

# CSR

- Buena UX
- Renderización rápida después de la carga inicial
- Excelente para aplicaciones web

**VENTAJAS**

- Dificulta SEO si no se implementa correctamente
- Página inicial carga más lento
- Requiere librería externa para utilizarlo fácil

**DESVENTAJAS**

# Client Side Rendering

JS + API REST

# ¿Como podemos hacer CSR usando nuestra API REST?

- Devolvemos un HTML básico vacío.
- Una vez cargado, interactuamos desde JS con la API REST.
- Generamos el HTML desde JS.

# Ajax en ES7

REPASO

## ¿Como interactuamos con una API Rest desde JavaScript?

ES7 incorpora la interfaz **fetch()**

```
fetch(url, opciones)
.then(response => //do something)
.catch(error => //do something);
```

El uso más simple de fetch() toma un argumento (la ruta del recurso que se quiera traer) y **el resultado es una promesa** que contiene la respuesta (un objeto Response)

# Let’s Work

# Let’s Work

¿Qué vamos a hacer?

El ABM de tareas usando nuestro SERVICIO WEB

# Que tenemos que hacer?

TO-DO (usando el servicio web)

- **Listar** las tareas
- **Agregar** una tarea
- **Borrar** una tarea
- Marcar una tarea como **realizada**

# Listar tareas

1. Generamos una nueva **acción** para traer la nueva pantalla.
2. Creamos un .**tpl** que no renderice la lista, solo el espacio (la vamos a crear con AJAX).
3. Incluimos un script **javascript** que interactue con la API.
4. Definimos el **endpoint** de la API del que vamos a utilizar /api/tareas => GET
5. Modificamos el DOM y renderizamos la lista de tareas.

# Listar Tareas - Template

- El **template** debe generar solo el esqueleto vacío para que se termine de renderizar en el cliente:

```
<section id="tareas">
   <ul class="lista-tareas">
   </ul>
</section>

 <form id="form-tarea" action="insertar" method="post">
     <input type="text" name="titulo" placeholder="Titulo">
     <input type="text" name="descripcion" placeholder="Descripcion">
     <input type="number" name="prioridad"  max="10">
     <input type="submit" value="Insertar">
 </form>

 <script src="js/tareas.js"></script>
```

# Listar Tareas - Funcionalidad JS

- Creamos e incluimos un archivo javascript **tareas.js**
- Creamos una función que traiga e imprima las **tareas**

```
function getTasks() {
    fetch('api/tareas/')
    .then(response => response.json())
    .then(tasks => {
        let content = document.querySelector(".lista-tareas");
        content.innerHTML = "";
        for(let task of tasks) {
            content.innerHTML += createTaskHTML(task);
        }
    })
    .catch(error => console.log(error));
}
```

# Creamos la función crearTareaHTML

```
function createTaskHTML(task) {
    let element = `${task.titulo}: ${task.descripcion}`;

    if (task.finalizada == 1)
        element = `<strike>${element}</strike>`;
    else {
        element += `<a href="tarea/${task.id}">Ver</a> `;
        element += `<a href="finalizar/${task.id}">Finalizar</a> `;
        element += `<a href="borrar/${task.id}">Eliminar</a>`;
    }

    element = `<li>${element}</li>`;
    return element;
}
```

# Agregar tareas

```
document.querySelector("#form-tarea").addEventListener('submit', addTask);
function addTask(e) {
    e.preventDefault();

    let data = {
        titulo:  document.querySelector("input[name=titulo]").value,
        descripcion:  document.querySelector("input[name=descripcion]").value,
        prioridad:  document.querySelector("input[name=prioridad]").value
    }

    fetch('api/tareas', {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(data)
     })
     .then(response => {
         getTasks();
     })
     .catch(error => console.log(error));
}
```

# Resultado


BOLIVAR
https://gitlab.com/unicen/Web2/LiveCoding2018/Bolivar/EjemploTareas/commit/13a7244c6a66de8e070154a0ddad2f001f3b6726

TANDIL
https://gitlab.com/unicen/Web2/livecoding2019/tandil/todo-list/commit/9493de6cad704104ee968c5abe31eec2e1e01119

¿Seguimos mezclando JS + HTML?

¿Cómo resolvimos la mezcla de HTML y PHP?

# Template engine (Motor de templates)

REPASO

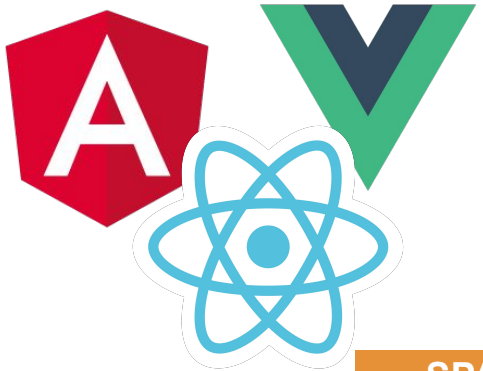
Los “Template Engine” son herramientas que se utilizan para separar la **lógica del programa** y la **presentación del contenido** en dos partes independientes.

### VENTAJAS

- Facilita el desarrollo tanto de la lógica como de la presentación.
- Mejora la flexibilidad.
- Facilita la modificación y el mantenimiento.

Las técnicas de CSR ganaron popularidad gracias a la aparición de **librerías/frameworks JS** que hicieron mucho más fácil esta tarea.


TEMPLATE ENGINES



SPA LIB/FW

# Vue.js

Opción 1: Framework JS

**Vue.js** provee un sistema que nos permite manipular el DOM utilizando una sintaxis de plantilla sencilla (template)

HTML

```
<div id="app">
  {{ message }}
</div>
```

JS

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

# Instalar Vue.js

## ¿Cómo lo incluyo en mi proyecto?

- Utilizamos el CDN

```
<!-- development version, includes helpful console warnings -->

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

- O se descarga desde el sitio oficial https://vuejs.org/v2/guide/installation.html y se incluye como cualquier dependencia JS.

# Vue.js - Documentación

**Render declarativo**

https://vuejs.org/v2/guide/#Declarative-Rendering

**Condicionales y bucles**

https://vuejs.org/v2/guide/#Conditionals-and-Loops

**Event handlers**

https://vuejs.org/v2/guide/#Handling-User-Input

# Vue.js | Render de tareas

## 1. Creamos el template de **vue** dentro de `templates/vue/`

**vue/task_list.tpl**

```
{literal}
<section id="template-vue-tasks">
    <h3> {{ subtitle }} </h3>

    <ul>
       <li v-for="task in tasks">
           <span v-if="task.finalizada == 1"> <strike>{{ task.titulo }} - {{task.descripcion}} </strike></span>
           <span v-else> {{ task.titulo }} - {{task.descripcion}} </span>

           <span v-if="task.finalizada == 0">
                <a :data-id="task.id" class="btn-eliminar" href="#">eliminar</a>
                <a :data-id="task.id" class="btn-completar" href="#">completar</a>
           </span>
       </li>
    </ul>
</section>
{/literal}
```

Se encierra el template entre **{literal}** para que Smarty no intente compilarlo.

# Vue.js | Render de tareas

2. Incluimos el template **vue** dentro del template **smarty** para que sea enviado al cliente:

```
{include file="header.tpl"}

        {include file="vue/task_list.tpl"}

        <form id="form-tarea" action="insertar" method="post">
           <input type="text" name="titulo" placeholder="Titulo">
             ...
           <input type="submit" value="Insertar">
        </form>

        <script src="js/tareas.js"></script>
{include file="footer.tpl"}
```

# Vue.js | Render de tareas

3. Generamos el archivo **tareas.js** e iniciamos **Vue.js**

```
let app = new Vue({
    el: "#template-vue-tasks",
    data: {
        subtitle: "Estas tareas se renderizan desde el cliente usando Vue.js",
        tasks: []
    }
});
```

4. Renderizamos las tareas obteniendo la información desde la API REST

```
function getTasks() {
    fetch("api/tareas")
    .then(response => response.json())
    .then(tasks => {
        app.tasks = tasks; // similar a $this->smarty->assign("tasks", $tasks)
    })
    .catch(error => console.log(error));
}
```

# Resultado


TANDIL
https://gitlab.com/unicen/Web2/livecoding2019/tandil/todo-list/commit/7b0addafc14bb790a7dd9744cdc1d1769996aa5b

BOLIVAR:
https://gitlab.com/unicen/Web2/livecoding2019/bolivar/todo-list/commit/155a5d037dbe0094b878a99c244fde86d1b1e306

# {{handlebars}}

Opción 2: Template Engine JS

# Handlebars.js

- Motor de templates Javascript.
- Al igual que Smarty, usa una combinación de etiquetas HTML y **etiquetas de plantilla** para formatear la presentación del contenido.

# Instalar handlebars.js

## ¿Cómo lo incluyo en mi proyecto?

- Se descarga desde el sitio oficial https://handlebarsjs.com/ y se incluye como cualquier dependencia JS.
- Ó directamente utilizamos el CDN: https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.12/handlebars.js

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.12/handlebars.js"></script>
```

# Handlebars - Template

```
<ul class="list-group list-group-flush">
    {{#each tasks}}
        <li class="list-group-item">
            <div class="tarea">
                <span {{#if finalizada}}class="finalizada"{{/if}} >{{titulo}}</span> |
                <a href="tareas/{{id_tarea}}/">Ver mas</a>
                <a href="end/{{id_tarea}}/">Hecha</a>
                <a href="delete/{{id_tarea}}/">Borrar</a>
            </div>
        </li>
    {{/each}}
</ul>
```

UTILIZA **{{ algo }}** PARA DELIMITAR EXPRESIONES

indirecciones implícitas

# Handlebars - Compilando un template

1. Descargar y compilar template

```
let templateTareas;

fetch('js/templates/tareas.handlebars')
 .then( response => response.text())
 .then(text => {
      let templateTareas = Handlebars.compile(text);
});
```

SE DESCARGA UNA VEZ AL PRINCIPIO

2. Instanciar Template con un contexto

```
let context = {
      tasks: [{"id_tarea":"21","titulo":"Hacer la API",...},
                  {"id_tarea":"23","titulo":"zxc",...],
      otraVariable: “valor”
}
let html = templateTareas(context); //acá tenemos el HTML final!!
```

ES COMO EL **assing()** DE SMARTY

3. Reemplazar el html generado por handlebars

```
document.querySelector("#tareas-container").innerHTML = html
```

# Handlebars y las evaluaciones

- Handlebars el 0 lo evalúa como verdadero
- Debemos transformar el 0 a booleana en código antes de pasarlo al template
- Lo ideal sería que lo haga la API (en PHP)
  - Que se guarde en la BBDD en un entero es tema del modelo
  - La API debe ocultar detalles de implementación
- En otro caso podría estar bien que la API lo devuelva así y sería responsabilidad de Javascript transformarlo

# Recorriendo el arreglo en Handlebars

- A Handlebar le pasamos un objeto
- Ese objeto se desencapsula automaticamente
- Como la API nos devuelve un arreglo, lo tenemos que meter en un objeto (usemos JSON) para que al desarmarlo lo pueda recorrer Mustache

```
template({arreglo:tareas});
```

# Recorriendo la lista de Tareas en Mustache

```
{{#each arreglo}}
  <li>
    {{#if completado}}<s>{{/if}}
      {{nombre}}
    {{#if}}</s>{{/if}}
  </li>
{{/each}}
```

# Modificando la API

- Es buena práctica en el diseño de APIs que devuelvan un objeto, no un arreglo suelto
- En Web 1 Heroku devuelve el objeto con un “information” donde estaba la respuesta que queríamos
- Esto no solo arregla el problema de Mustache, da facilidades para cambios a futuro si tenemos que agregar info adicional a la respuesta
  - Paginado
  - Fecha del pedido
  - Tiempo de cálculo

# Resultado


https://gitlab.com/unicen/Web2/LiveCoding2018/Bolivar/EjemploTareas/commit/2dfcf1d69d31601cb3220318f78d48133680d1a8