INSTITUTO TÉCNICO RICALDONE
ESPECIALIDAD DESARROLLO DE SOFTWARE
TERCER AÑO DE BACHILLERATO

GUÍA INTRODUCTORIA:
EJEMPLO DE APLICACIÓN WEB

DOCENTE:

DANIEL STANLEY CARRANZA MIGUEL

daniel_carranza@ricaldone.edu.sv

https://github.com/dacasoft/coffeeshop

SAN SALVADOR, EL SALVADOR 2024

RESUMEN

El presente documento tiene por objetivo realizar un acercamiento al ejemplo proporcionado ***CoffeeShop***, para conocer y comprender su proceso de desarrollo, ayudando así al aprendizaje en los diferentes módulos, por lo que puede ser utilizado según convenga para trabajar los proyectos durante el año. Dicho ejemplo ha sido elaborado bajo las siguientes tecnologías y herramientas:

- FRONTEND
  - HTML5 y CSS3 con Bootstrap v5.3.0 y Bootstrap Icons v1.10.5
  - Lenguaje de programación JavaScript mediante Vanilla JS
  - Librerías SweetAlert v2.0 y Chart.js v4.3.0
- BACKEND
  - Servidor web Apache v2.4.54
  - Lenguaje de programación PHP v8.2.0
  - Librería FPDF v1.85
  - Gestor de bases de datos MariaDB v10.4.27

Actualmente es conocido que *JavaScript* es el lenguaje de programación más utilizado a nivel general, de acuerdo con algunas estadísticas de *stackoverflow* (https://insights.stackoverflow.com/survey/), o al menos es así para la mayoría de los desarrolladores profesionales (ver figura 1), es decir, *JavaScript* es hoy por hoy, la tecnología más difundida y preferida mundialmente, superando a lenguajes más nuevos o de moda.

La popularidad de *JavaScript* es debida a muchos factores, principalmente a su evolución y eficiencia en los últimos años. Si partimos de esta premisa, podríamos decir que es una buena opción para iniciar, ya que los proyectos a desarrollar son de carácter formativo, siendo la base para que cualquier desarrollador web principiante pueda dar el salto más adelante al uso de frameworks o librerías como React, Vue, Angular, etc.

En cuanto a *PHP*, es el lenguaje de programación más utilizado del lado del servidor de acuerdo con *W3Techs* (https://w3techs.com/), tal como se aprecia en la figura 2, por lo tanto, sigue estando vigente y en constante evolución. Algunas de sus ventajas son: aprendizaje intuitivo simplificado, código abierto, soporta una gran cantidad de peticiones y compatible con los principales gestores de bases de datos. La última versión lanzada hasta el momento la 8.3.2 (18 de enero de 2024). Para más información https://www.php.net/

Por otro lado, tenemos el gestor de bases de datos *MariaDB*, el cual es una réplica de *MySQL*, siendo este último el más popular según datos de *stackoverflow* (ver figura 3), aclarando que, la popularidad de *MySQL* se debe a que por varios años fue software de código abierto hasta que lo adquirió la empresa *Oracle*, lo que deja a *MariaDB* como una de las opciones gratuitas más conveniente para el almacenamiento en línea. Para más información https://mariadb.org/

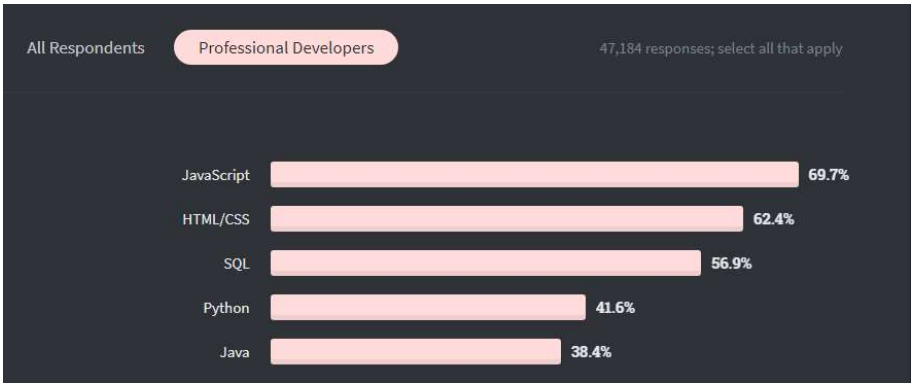


Figura 1. Top 5 de lenguajes más utilizados por desarrolladores profesionales

Server-side Programming Languages

Most popular server-side programming languages

© W3Techs.com	usage	change since 1 December 2023
1. PHP	76.6%	
2. ASP.NET	6.5%	-0.2%
3. Ruby	5.6%	
4. Java	4.7%	
5. JavaScript	3.2%	+0.1%

percentages of sites

Figura 2. Top 5 de lenguajes de programación más populares del lado del servidor

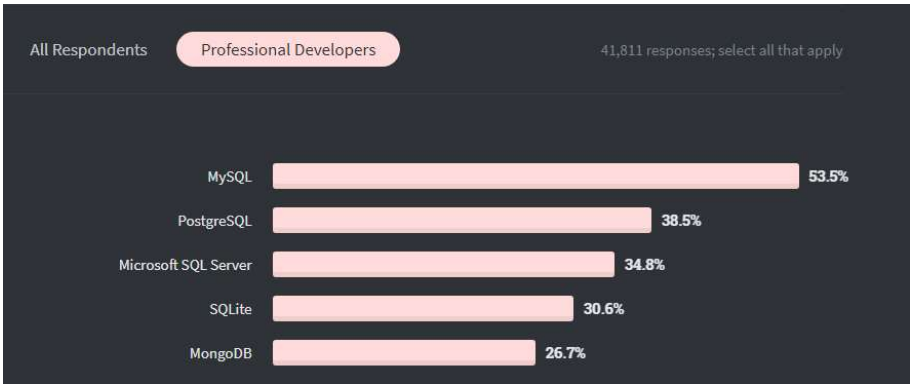


Figura 3. Top 5 de bases de datos más utilizadas por desarrolladores profesionales

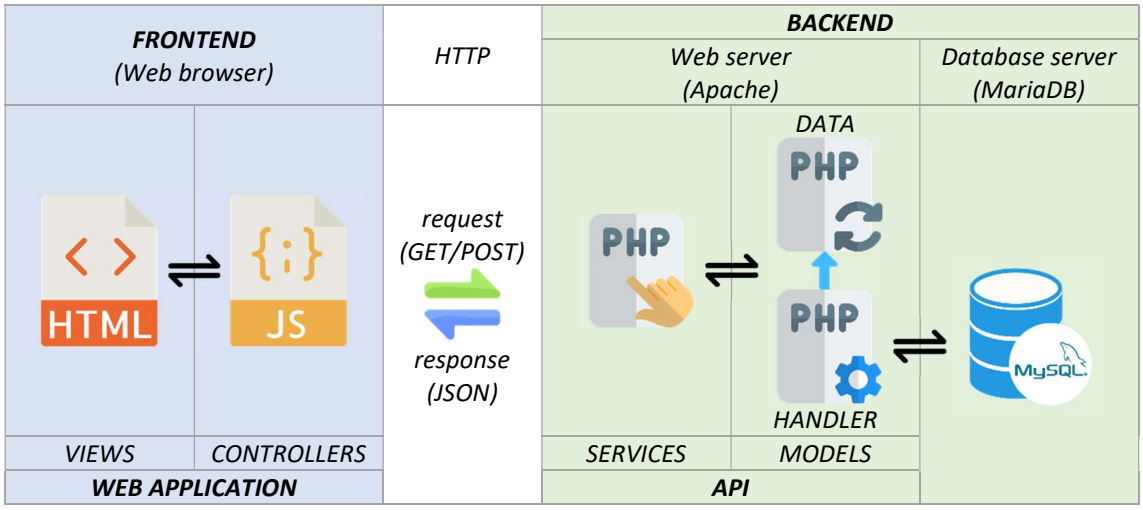
Como se puede intuir, el objetivo del ejemplo no es fomentar el uso de tecnologías de última generación o que sean tendencia, más bien es plantar los cimientos de la programación web mediante recursos ampliamente conocidos, para que posteriormente cada uno pueda tener un mejor conocimiento del camino a seguir.

Para el desarrollo del ejemplo se parte del hecho que ya se ha elaborado previamente el modelo relacional para entender la lógica de los datos a utilizar, así como también, la creación de *mockups* para tener una idea clara acerca del diseño de las páginas web que se planean construir.

Además, se han tomado en cuenta las buenas prácticas de desarrollo sugeridas en los diferentes módulos, así como otras recomendaciones que se irán tratando durante el año, teniendo la gran ventaja de que existe documentación abundante en Internet ante cualquier necesidad.

Para finalizar, el ejemplo trata de recopilar una diversidad de conceptos de programación y aplicarlos al desarrollo de software para solucionar problemas o mejorar necesidades bajo un enfoque formativo, que catapulte a los estudiantes a un mundo productivo y profesional en un futuro cercano.

ARQUITECTURA DEL PROYECTO



Las vistas (*views*) son prácticamente páginas web elaboradas con *HTML5* empleando *Bootstrap*, las cuales interactúan directamente con los controladores (*controllers*) programados en *JavaScript* mediante *Vanilla JS*. En términos generales, la programación asíncrona con JavaScript permite realizar peticiones y recibir respuestas en tiempo real para manipular el *DOM* (*Document Object Model*) según las necesidades de la aplicación, es decir, es lo que hace funcionar el contenido dinámico en las páginas web.

La programación asíncrona facilita el manejo de la interacción del usuario con las vistas sin recargar la página web, o sea, sin refrescar el navegador. Por medio de los controladores se hacen peticiones (*request*) y se reciben respuestas (*response*) en formato *JSON* (*JavaScript Object Notation*). Cuando una petición implica enviar datos como por ejemplo actualizar un registro, se remiten por medio del método *POST*, en cambio, si lo que se requiere únicamente es solicitar datos como por ejemplo listar registros, se piden a través del método *GET*.

Las vistas y los controladores son los únicos elementos que funcionan del lado del cliente (*frontend*) interpretados por el navegador web, donde la implementación de programación asíncrona puede realizarse de diversas formas, no solo con *Vanilla JS*, por ejemplo, se podría optar por utilizar librerías como *jQuery* (no recomendable), *React*, entre otras; o dar un salto más allá mediante la utilización de un framework basado en JavaScript como *Vue, Angular* u otro similar.

Para cada página web existe un archivo *JavaScript* que actúa como controlador principal, capaz de gestionar todas las peticiones y respuestas, ya sea de forma automática o por intervención del usuario. Dicha gestión se realiza a través de diversos métodos, eventos y funciones; que dotan a la página web de toda la funcionalidad necesaria para realizar una o varias tareas específicas. Por conveniencia, se recomienda que el nombre del controlador principal sea el de la página web.

Por otro lado, tenemos los elementos del lado del servidor (*backend*) que llamaremos *API* (*Application Programming Interface – Interfaz de Programación de Aplicaciones* –), la cual interactúa directamente con los controladores por medio de la capa de servicios (*SERVICES)*, recibiendo peticiones *GET* o *POST* y retornando respuestas en formato *JSON*. La *API* reacciona según las acciones requeridas por los controladores, donde cada acción está definida en la *URL* del archivo que la contiene, lo que recibe el nombre de *endpoint*.

La *API* está programada con *PHP*, instanciando en *SERVICES* una o más clases de los modelos (*MODELS*), es decir, se implementa *OOP* (*Object Oriented Programming – Programación Orientada a Objetos* –). Además, en los modelos se validan todos los datos de entrada del lado del servidor al momento de establecer (*SET*) los valores a los atributos, específicamente en la capa *DATA*, como parte del encapsulamiento por si se escapa algún dato erróneo o malicioso desde el *frontend*.

La capa *SERVICES* está formada por varios archivos o *scripts*, cada archivo contiene una o varias acciones (*endpoints*). Algunas acciones pueden requerir autenticación del usuario, por lo que se debe validar para evitar exponer datos sensibles o realizar operaciones sin control, previniendo de esta forma posibles ataques o robos de información. También, aquí se obtienen las excepciones de la base de datos y se realiza el manejo de errores en los valores de entrada.

Los modelos son una representación fidedigna de la base de datos, modelando generalmente cada tabla por medio de dos clases: *DATA* y *HANDLER*. Cada clase debe elaborarse en un archivo, o por lo menos eso sugieren las *PSR* (*PHP Standards Recommendations – Recomendaciones de Estándares PHP* –). En dichos modelos se realizan las operaciones con la base de datos por medio de la capa *HANDLER*, donde se implementa la clase *PDO* (*PHP Data Object – Objetos de Datos PHP* –).

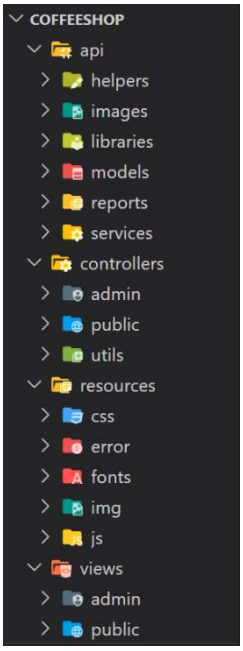
La extensión *PDO* es una clase nativa en *PHP* a partir de la versión 5.1, la cual define una interfaz ligera para trabajar con diferentes bases de datos a través de controladores específicos que proveen características propias de cada base de datos, siendo *MariaDB* una de las opciones soportadas. *PDO* proporciona una capa de abstracción de acceso a datos, lo que significa que, puede utilizarse con diversos gestores de bases de datos de forma automática o realizando pequeñas configuraciones.

Ahora viene la pregunta ¿por dónde se empieza? No existe una respuesta única, sin embargo, se puede seguir cierta lógica para llevar a cabo el desarrollo, tal como se sugiere a continuación:

1. Construir la base datos.
2. Elaborar los mockups definiendo todos los detalles de diseño y tomando en cuenta la base de datos.
3. Crear las vistas (*views*) con un framework de acuerdo con los mockups.
4. Programar las clases de los modelos (*models*) con PHP según la base de datos, así:
    a. Clase Handler: definir las propiedades a partir de las columnas de la tabla y luego un método por cada sentencia SQL.
    b. Clase Data: crear un método ***set*** por cada propiedad e incluir la validación respectiva antes de asignar el valor.
5. Implementar los modelos para las acciones de los servicios (*services*) y probar con Postman o una herramienta similar.
6. Programar los controladores (*controllers*) con JavaScript.

ESTRUCTURA DE LA APLICACIÓN

Veamos de forma general cómo está organizado el proyecto, partiendo del hecho que existe un directorio raíz que en el ejemplo se llama ***coffeeshop***. Como se puede observar en la imagen, el directorio raíz cuenta con cuatro carpetas principales (pueden ser más o menos dependiendo de las necesidades): ***api, controllers, resources*** *y* ***views***; las cuales se describen a continuación.



- ***api:*** esta carpeta es la interfaz de programación de la aplicación del lado del servidor, donde ***services*** contiene toda la lógica del negocio para recibir peticiones (*request*) y enviar resultados (*response*), ***models*** cuenta con todo lo relacionado al manejo de los datos, ***helpers*** sirve para albergar funcionalidad de uso general como por ejemplo: validaciones, conexión a la base de datos, plantillas, etc.; ***images*** guarda todos los archivos de imágenes relacionadas con los datos, ***libraries*** aloja librerías de terceros como por ejemplo: *PHPMailer* y *fpdf*, las cuales sirven para ampliar la funcionalidad de la aplicación; y ***reports*** contiene toda la programación necesaria para generar reportes a partir de la base de datos.
- ***controllers:*** representa la lógica de programación del lado del cliente que interactúa con la *API*, es decir, realiza peticiones y recibe resultados. Las carpetas ***admin*** y ***public*** cuentan con los controladores necesarios para cada sitio web y ***utils*** es para alojar scripts de uso general.
- ***resources:*** este directorio contiene esencialmente todos aquellos archivos estáticos de propósito general, ya sean propios o de terceros, es decir: hojas de estilo en cascada (***css***), fuentes tipográficas (***fonts***), imágenes (***img***), scripts (***js***) y páginas web personalizadas para el manejo de errores (***error***).
- ***views:*** aquí es donde se encuentran todas las páginas web de la aplicación, es decir, las vistas. Dentro de este directorio se ha creado una carpeta para el sitio público (*public*) y otra para el sitio privado (*admin*).

Volviendo a la carpeta ***api***, se puede considerar como una aplicación autónoma, construida exclusivamente con *PHP* para funcionar del lado del servidor (*backend*), independiente de todo lo demás, de esta forma es posible proporcionar servicios a diferentes plataformas como web y móvil, siempre y cuando existan las acciones necesarias en el directorio *services* para suplir dichos requerimientos.

En cuanto a la carpeta *services*, se recomienda crear un directorio por cada sitio web del proyecto, dentro de los cuales es conveniente agregar un archivo por cada tabla que se necesite de la base de datos, el cual debe contener todas aquellas acciones (*endpoints*) requeridas para dicha tabla dentro del sitio web. En algunos casos es indispensable validar la autenticación del usuario para realizar ciertas acciones.

Si se dice que la carpeta *api* es un programa independiente elaborado únicamente con *PHP* (sin interfaz gráfica), puede surgir la pregunta ¿cómo probar o testear las acciones? Existen diversas alternativas, sobre todo si necesitamos enviar datos, una sería ***Postman*** (https://www.postman.com/), en cambio sí solo se quiere solicitar datos, estos pueden visualizarse directamente en el navegador web.

Con respecto al código, se emplea el siguiente estándar para tener una programación más ordenada y con menos riesgos de errores, esto es: en *HTML* se usan comillas dobles para asignar el valor a los atributos de las etiquetas, por ejemplo: ***<input type="password" id="userPassword" name="userPassword">***; por otro lado, se utilizan comillas simples en *JavaScript* y *PHP* (salvo algunas excepciones).

De acuerdo con lo anterior, se pueden utilizar estilos propios o de terceros para establecer los valores personalizados a ciertos atributos en las etiquetas *HTML*, en este caso se sigue la propuesta de *Bootstrap* que es *camelCase*, como por ejemplo para el valor del atributo ***id*** de la etiqueta ***input***. Adicionalmente, es útil pero no obligatorio asignar al atributo ***name*** de la etiqueta *input* el mismo valor del atributo *id*, siempre y cuando no se repita dicho valor en una misma página web.

CONSIDERACIONES FINALES

- Se recomienda analizar la programación de las validaciones disponibles del lado del servidor (***api/helpers/validator.php***), el manejo de la base de datos (***api/helpers/database.php***) y los modelos; donde el código se encuentra suficientemente documentado para comprender, de lo contrario existe la opción de preguntar o investigar.
- Para elaborar todo el diseño del ejemplo, se han utilizado únicamente los componentes del framework *Bootstrap*, en ningún momento se ha recurrido a crear CSS personalizado, con el objetivo de aprovechar todos los recursos que ofrece la herramienta, teniendo así más tiempo para enfocarse en otras tareas. De igual forma, casi toda la programación del lado del cliente funcionaría con otros frameworks similares, realizando únicamente pequeñas modificaciones.
- Si se toma de base el ejemplo para desarrollar otros proyectos, se recomienda encarecidamente cambiar al menos algunos elementos del diseño y ponerle un toque original a la vista de la aplicación web, para que no sea tan evidente la copia y exista un valor agregado por parte de los estudiantes. Además, es necesario que se vaya documentando el código de acuerdo con cada tarea.
- En cuanto a las validaciones, el ejemplo solo dispone del lado del servidor con *PHP* (*backend*), por lo que es responsabilidad de los estudiantes investigar cómo realizar las validaciones del lado del cliente (*frontend*), ya sea con *HTML* o *JavaScript*, ya que sería la primera línea de defensa para evitar el ingreso de datos no validos o maliciosos.
- Cuando se tenga alguna duda sobre una palabra reservada, función o algo especifico de la programación (*JavaScript* o *PHP*), lo mejor es buscar en la documentación oficial, ya que el ejemplo contiene comentarios bien básicos, sin profundizar mucho en la sintaxis para fomentar la investigación y el aprendizaje autónomo.
- El uso de estándares de programación es una buena práctica y algo conveniente, sobre todo cuando vamos a trabajar con otras personas. En este sentido podemos ocupar reglas ya existentes de PHP y JavaScript, tal como las sugeridas en el módulo 1. También se pueden definir las propias, pero es muy importante seguirlas al pie de la letra para que tengan sentido.
- Al seleccionar una o más tecnologías cuando se inicia un proyecto de desarrollo de software, no solo hay que dejarse llevar por la moda del momento o impulsos, es necesario pensar y tomar la decisión con mente fría, teniendo en cuenta al menos los siguientes puntos: tiempo de aprendizaje, personal capacitado y amplia documentación en línea.