# Test de Conocimientos Generales para Desarrollador Fullstack Junior

## Sección 1: Symfony

**1. \*\*Pregunta de Configuración: \*\***

- Describe los pasos básicos para levantar un proyecto en Symfony.

1. Instalación de Composer
2. Instalación de Symfony
3. Utilizar el servidor que tenemos con la instalación o bien un servidor como Xampp.

**2. \*\*Pregunta de Código:\*\***

- Crea un controlador en Symfony que maneje una ruta /hello/{name} y devuelva un saludo personalizado. Además, si el nombre no se proporciona, debe devolver un saludo predeterminado "Hello World". (opcional) Implementa también una prueba unitaria para verificar que la ruta devuelve el saludo correcto.

**3. \*\*Pregunta Teórica: \*\***

- Explica la arquitectura de Symfony y cómo se organiza un proyecto típico en términos de carpetas y archivos.

Symfony es un framework de desarrollo de aplicaciones web PHP que sigue una arquitectura MVC (Modelo-Vista-Controlador) y está diseñado siguiendo el paradigma de desarrollo basado en componentes. La arquitectura de Symfony está diseñada para ser modular, flexible y escalable, lo que permite a los desarrolladores construir aplicaciones web robustas y de alto rendimiento.

La estructura de directorios de un proyecto típico de Symfony está organizada de la siguiente manera:

1. **app/**: Este directorio contiene la configuración de la aplicación
2. **bin/**: Este directorio contiene los scripts ejecutables de la aplicación.
3. **src/**: Este es el directorio donde reside el código fuente de la aplicación
4. **vendor/**: Este directorio contiene las dependencias de la aplicación que son administradas por Composer, el administrador de paquetes de PHP. Aquí se encuentran las bibliotecas y componentes de terceros que la aplicación utiliza.
5. **var/**: Este directorio almacena archivos generados por la aplicación, como cachés, logs y archivos temporales.
6. **config/**: Aquí se encuentra la configuración de la aplicación. Los archivos YAML, XML o PHP en este directorio definen la configuración de enrutamiento, la configuración de la base de datos, la configuración de servicios y otros parámetros de la aplicación.
7. **public/**: Este es el directorio web raíz que contiene los archivos públicos accesibles desde el navegador web, como imágenes, hojas de estilo y archivos JavaScript. El punto de entrada de la aplicación, index.php, reside en este directorio.
8. **templates/**: Aquí se encuentran las plantillas de Twig, el motor de plantillas utilizado por Symfony. Las plantillas Twig se utilizan para generar la salida HTML de las vistas de la aplicación.
9. **tests/**: Este directorio contiene pruebas automatizadas para la aplicación, incluyendo pruebas unitarias y pruebas de integración.
10. **translations/**: Aquí se almacenan los archivos de traducción que se utilizan para localizar la aplicación en diferentes idiomas.
11. **var/**: Este directorio almacena archivos generados por la aplicación, como cachés, logs y archivos temporales.

**4. \*\*Pregunta de Código:\*\***

- Escribe un servicio en Symfony que se inyecta en un controlador y realiza una operación matemática básica (por ejemplo, sumar dos números). ¿Qué configuraciones son necesarias para poder usarlo? (opcional) Implementa también una prueba unitaria para verificar el correcto funcionamiento del servicio.

**5. \*\*Pregunta de Código:\*\***

- Muestra cómo crear un formulario en Symfony para una entidad User con campos username y email.

**6. \*\*Pregunta Teórica:\*\***

- Explica el concepto de Dependency Injection (DI) en Symfony y cómo se configura.

Dependency Injection es un patrón de diseño que se utiliza para gestionar las dependencias entre diferentes componentes de una aplicación. En lugar de que un componente cree directamente sus dependencias, estas se le pasan desde el exterior, lo que hace que el componente sea más flexible, reutilizable y fácil de probar.

En Symfony, el contenedor de servicios es el componente principal que se utiliza para implementar la Dependency Injection. El contenedor de servicios es una clase que almacena y administra todas las dependencias de una aplicación, y las inyecta automáticamente en los servicios que las necesitan.

Configuración de la Dependency Injection en Symfony:

1. **Definición de servicios**: En Symfony, cada componente de la aplicación es un servicio. Los servicios se definen en archivos de configuración, generalmente en el directorio config/services.yaml. En estos archivos, se especifica el nombre del servicio, su clase y cualquier dependencia que tenga.
2. **Inyección de dependencias**: Una vez que los servicios están definidos en el contenedor de servicios, Symfony se encarga de inyectar automáticamente las dependencias en los servicios cuando se crean. Esto se realiza mediante la configuración del contenedor de servicios.
3. **Uso de servicios**: Una vez que las dependencias están inyectadas en un servicio, este puede utilizarlas en su lógica interna.

**7. \*\*Pregunta de Código: \*\***

- Escribe una consulta Doctrine en Symfony para obtener todos los registros de una entidad Product donde el precio sea mayor a 100.

**8. \*\*Pregunta Teórica:\*\***

- ¿Qué es el Event Dispatcher en Symfony y para qué se utiliza?

El Event Dispatcher en Symfony es un componente que proporciona un mecanismo para implementar el patrón de diseño Observer. Este componente permite a los objetos de tu aplicación notificar a otros objetos interesados cuando ocurren ciertos eventos dentro del sistema. En resumen, el Event Dispatcher se utiliza para gestionar y despachar eventos en una aplicación Symfony.

**9. \*\*Pregunta de Código:\*\***

- Crea un validador personalizado en Symfony para asegurar que el campo email de una entidad User no pertenece a un dominio específico (por ejemplo, "example.com"). Muestra cómo configurar este validador y cómo sería utilizado en la entidad User.

**10. \*\*Pregunta de Código: \*\***

- Implementa un Event Subscriber en Symfony que escuche el evento kernel.request y registre en un archivo de log cada visita a cualquier página de la aplicación. Asegúrate de configurar el servicio correctamente para que el suscriptor se registre con el evento.

**Sección 2 JavaScript**

**1. \*\*Pregunta Teórica: \*\***

- Explica la diferencia entre var, let y const en JavaScript.

var, let, y const son formas de declarar variables en JavaScript, y cada una tiene diferencias en cuanto a su alcance y capacidad de reasignación de valores.

***var:*** Las variables declaradas con var tienen un alcance de función, lo que significa que están disponibles dentro de la función en la que fueron declaradas, sin importar bloques {}. Pueden ser redeclaradas y reasignadas.

***let:*** Fue introducido en ECMAScript 6 para proporcionar un alcance de bloque más estricto. Las variables declaradas con let tienen alcance de bloque, lo que significa que están disponibles solo dentro del bloque en el que fueron declaradas. No pueden ser redeclaradas dentro del mismo ámbito. Sin embargo, sí pueden ser reasignadas.

***const:*** Fue introducido en ECMAScript 6 y se utiliza para declarar constantes. Al igual que let, las variables declaradas con const tienen alcance de bloque. Las variables declaradas con const deben recibir un valor al momento de la declaración y no pueden ser reasignadas.

**2. \*\*Pregunta de Código: \*\***

- Escribe una función en JavaScript que invierta una cadena de texto.

Correr script ***invertirCadena.js***

**3. \*\*Pregunta Teórica: \*\***

- ¿Qué es el Event Loop en JavaScript y cómo funciona?

El Event Loop es un concepto fundamental en JavaScript que controla cómo se manejan las tareas asíncronas y la concurrencia en el navegador y en entornos de ejecución como Node.js. En esencia, es el mecanismo que permite que JavaScript sea no bloqueante y asíncrono.

Cómo funciona:

El **Call Stack**: es una estructura de datos que mantiene un registro de las funciones que se están ejecutando en ese momento. Cuando una función es invocada, se coloca en la parte superior del Call Stack. Cuando esa función termina de ejecutarse, se elimina del Call Stack.

La **Cola de Eventos**: es una cola FIFO que contiene eventos y funciones asíncronas.

El **Event Loop**: es un ciclo que se ejecuta continuamente, revisando el Call Stack y la Cola de Eventos. Si el Call Stack está vacío y hay eventos en la Cola de Eventos, el Event Loop toma el primer evento de la cola y lo envía al Call Stack para su ejecución.

Esto significa que las tareas asíncronas no bloquean el hilo de ejecución principal de JavaScript. En lugar de eso, se manejan de manera asíncrona a través del Event Loop.

Las **Microtareas**: son tareas que se ejecutan después de que se vacía el Call Stack, pero antes de que el Event Loop tome un evento de la Cola de Eventos.

Esto significa que las promesas y otros procesos asíncronos de alta prioridad se ejecutan antes de que se manejen los eventos de la Cola de Eventos.

**4. \*\*Pregunta de Código: \*\***

- Escribe un script en JavaScript que filtre los números pares de un array de números y los muestre en la consola.

Correr script **filtrarPares.js**

**5. \*\*Pregunta Teórica: \*\***

- ¿Qué es el DOM y cómo se manipula usando JavaScript?

El DOM (Document Object Model) es una interfaz de programación para documentos HTML y XML. Define la estructura lógica de los documentos y la forma en que se accede y manipula su contenido. En resumen, el DOM representa la estructura del documento como un árbol de objetos, donde cada elemento del documento es un nodo en ese árbol.

Para manipular el DOM en JavaScript, puedes seguir estos pasos básicos:

**Obtener referencias a elementos del DOM:**

Utiliza los métodos proporcionados por el objeto document para acceder a elementos específicos del DOM. Algunos de los métodos comunes son getElementById, getElementsByClassName, getElementsByTagName, querySelector, y querySelectorAll.

**Manipular contenido y atributos de elementos:**

Una vez que tengas una referencia al elemento deseado, puedes manipular su contenido y atributos utilizando propiedades como textContent, innerHTML, setAttribute, removeAttribute, entre otras.

**Agregar o eliminar elementos del DOM:**

Puedes agregar nuevos elementos al DOM utilizando métodos como createElement para crear un nuevo nodo y appendChild o insertBefore para agregarlo al árbol DOM. Para eliminar elementos, puedes utilizar métodos como removeChild.

**Manejar eventos:**

Puedes adjuntar controladores de eventos a los elementos del DOM utilizando el método addEventListener para responder a las interacciones del usuario, como clics de ratón, pulsaciones de teclas, cambios de valor de entrada, etc.

**6. \*\*Pregunta de Código: \*\***

- Escribe un código en JavaScript que añada un event listener a un botón con el id #myButton para mostrar una alerta con el mensaje "Hello World" al hacer clic.

Correr script **botonAlertHelloWord.html**

**7. \*\*Pregunta Teórica: \*\***

- Explica qué es una Promesa en JavaScript y proporciona un ejemplo básico.

Una promesa en JavaScript es un objeto que representa la eventual finalización o falla de una operación asincrónica, y su valor resultante. Las promesas son una forma de manejar operaciones asíncronas de manera más limpia y legible, evitando el anidamiento excesivo de callbacks. Una promesa puede estar en uno de los tres estados: pendiente (pending), resuelta (fulfilled), o rechazada (rejected).

Ejemplo básico de promesa en Javascript:

// Crear una función que devuelve una promesa

function miPromesa() {

return new Promise((resolve, reject) => {

// Simular una operación asincrónica (en este caso, un temporizador)

setTimeout(() => {

// Aquí puedes hacer cualquier tipo de operación asíncrona

// En este ejemplo, simplemente resolvemos la promesa después de 2 segundos

resolve('¡La promesa se resolvió exitosamente!');

}, 2000); // Simular una demora de 2 segundos

});

}

// Utilizar la promesa

console.log('Iniciando la ejecución de la promesa...');

miPromesa()

.then(resultado => {

// El caso de éxito: la promesa se resuelve

console.log('Resultado:', resultado);

})

.catch(error => {

// El caso de fallo: la promesa es rechazada

console.error('Error:', error);

});

console.log('Promesa en proceso...');

**8. \*\*Pregunta de Código: \*\***

- Escribe una función en JavaScript que haga una petición AJAX (usando fetch) para obtener datos de una API y los muestre en un elemento con el id #result.

Correr script **obtenerDatosAPI.html**

**9. \*\*Pregunta Teórica: \*\***

- ¿Cuál es la diferencia entre null, undefined y NaN en JavaScript?

En JavaScript, null, undefined y NaN son valores distintos con significados diferentes:

***null:*** es un valor especial que representa la ausencia intencional de cualquier valor o referencia a un objeto. Se utiliza cuando se quiere indicar que una variable no tiene un valor asignado o cuando se desea borrar el valor de una variable.

***undefined:*** es un valor que indica que una variable ha sido declarada pero no inicializada, o que una propiedad de un objeto no está definida.

***NaN (Not a Number):*** es un valor especial que indica que una operación numérica ha producido un resultado que no es un número válido.

**10. \*\*Pregunta de Código: \*\***

**- Implementa una función en JavaScript que use localStorage para guardar una clave-valor y luego recuperarla.**

Correr script **guardarEnLocalStorage.js** en el navegador

**## Sección 3: Git**

**1. \*\*Pregunta Teórica: \*\***

**- ¿Qué es Git y para qué se utiliza en el desarrollo de software?**

Git es una herramienta poderosa que facilita la colaboración, la gestión de versiones y el seguimiento del progreso en el desarrollo de software, lo que ayuda a los equipos a trabajar de manera más eficiente y a mantener la calidad del código a lo largo del tiempo.

1. Principio del formulario
2. Final del formulario

**2. \*\*Pregunta de Comandos: \*\***

- ¿Cuál es el comando para clonar un repositorio de Git?

git clone <URL\_del\_repositorio>

**3. \*\*Pregunta Teórica: \*\***

- Explica qué es un "branch" (rama) en Git y para qué se utiliza.

En Git, una rama es una línea independiente de desarrollo que se bifurca del flujo principal del proyecto (generalmente denominado "rama principal" o "rama maestra", por defecto denominada "master" en Git). Cada rama en Git representa una serie de cambios en el código fuente y su historial de versiones.

Las ramas en Git se utilizan por varias razones:

* **Desarrollo de nuevas características**:
* **Solución de problemas (bug fixing)**
* **Versiones estables y de lanzamiento**
* **Experimentación y pruebas**

**4. \*\*Pregunta de Comandos: \*\***

- Proporciona los comandos necesarios para crear una nueva rama llamada feature-xyz, cambiar a esa rama, y luego fusionarla con la rama main.

**Crear una nueva rama llamada feature-xyz**:

**git checkout -b feature-xyz**

Este comando crea una nueva rama llamada feature-xyz y te cambia automáticamente a ella

**Fusionar la rama feature-xyz con la rama main**:

Primero, hay que asegurarse de estar en la rama main:

**git checkout main**

Luego, hay que fusionar la rama feature-xyz con la rama main:

**git merge feature-xyz**

**5. \*\*Pregunta Teórica: \*\***

- ¿Qué es un "merge conflict" y cómo se resuelve?

Un conflicto de fusión (merge conflict) ocurre cuando Git no puede realizar una fusión (merge) automáticamente entre dos ramas debido a cambios contradictorios en el mismo archivo, en las mismas líneas, o cuando un archivo fue eliminado en una rama pero modificado en otra.

Cuando Git detecta un conflicto de fusión, marcará el archivo con conflictos y no realizará la fusión automáticamente. Es entonces responsabilidad del usuario resolver estos conflictos manualmente.

Para resolver un conflicto hay que hacer lo siguiente:

1. **Identificar los archivos con conflictos**
2. **Abrir el archivo en conflicto**
3. **Resolver los conflictos manualmente**
4. **Eliminar las marcas de conflictos**
5. **Agregar los archivos modificados al área de preparación** (utilizando el comando git add <nombre\_del\_archivo>).
6. **Completar la fusión** ( git merge –continue).

**6. \*\*Pregunta de Comandos: \*\***

- ¿Cuál es el comando para visualizar el estado actual del repositorio en Git?

***git status***

**7. \*\*Pregunta Teórica: \*\***

- Explica la diferencia entre git pull y git fetch.

La principal diferencia entre git pull y git fetch radica en que git fetch solo descarga los cambios del repositorio remoto sin fusionarlos automáticamente, mientras que git pull descarga los cambios y los fusiona automáticamente con tu rama local.

**8. \*\*Pregunta de Comandos: \*\***

- ¿Cuál es el comando para revertir el último commit en Git?

**git revert HEAD**

**9. \*\*Pregunta Teórica: \*\***

- ¿Qué es un "remote repository" y cómo se configura en Git?

Un repositorio remoto (remote repository en inglés) en Git es una versión de tu proyecto almacenada en un servidor remoto, como GitHub, GitLab o Bitbucket. Estos repositorios remotos sirven como ubicaciones centrales donde varios colaboradores pueden colaborar en un proyecto, sincronizando sus cambios entre sí.

Configurar un repositorio remoto en Git generalmente implica dos pasos:

**Agregar un repositorio remoto:** Esto implica indicar a Git la ubicación del repositorio remoto, es decir, su URL. Para hacer esto, usamos el comando git remote add <nombre\_remoto> <URL\_del\_repositorio>. Aquí, <nombre\_remoto> es el nombre que deseas darle al repositorio remoto (por lo general, origin es el nombre comúnmente utilizado), y <URL\_del\_repositorio> es la URL del repositorio remoto.

Por ejemplo:

git remote add origin https://github.com/ejemplo/ejemplo.git

**Sincronización con el repositorio remoto:** Una vez que el repositorio remoto está configurado, puedes enviar (push) tus cambios al repositorio remoto o traer (pull) los cambios del repositorio remoto a tu repositorio local. Los comandos comunes para esto son git push y git pull.

Para enviar tus cambios al repositorio remoto, puedes usar:

git push <nombre\_remoto> <rama\_local>

Para traer los cambios del repositorio remoto a tu repositorio local, puedes usar:

git pull <nombre\_remoto> <rama\_remota>

También se puede usar git clone para clonar un repositorio remoto, lo que automáticamente configura el repositorio remoto y lo añade como origin.

**10. \*\*Pregunta de Comandos: \*\***

- Proporciona los comandos para añadir todos los cambios en los archivos al staging area y luego realizar un commit con el mensaje "Initial commit".

**Añadir todos los cambios al área de preparación**:

git add .

**Realizar el commit con el mensaje "initial commit"**:

git commit -m "initial commit"

**## Sección 4: MySQL**

**1. \*\*Pregunta de Código:\*\***

**-** Escribe una consulta SQL para crear una base de datos llamada company y una tabla llamada employees con las siguientes columnas: id (INT, auto-increment, primary key), name (VARCHAR(100)), position (VARCHAR(50)), salary (DECIMAL(10, 2)), y hire\_date (DATE).

cargar script **1bdCompany.sql**

**2. \*\*Pregunta Teórica:\*\***

- Explica la diferencia entre una clave primaria (Primary Key) y una clave foránea (Foreign Key) en MySQL. ¿Cuándo y por qué se utilizan?

La clave primaria identifica de forma única cada fila en una tabla, mientras que la clave foránea establece y mantiene las relaciones entre las tablas al referenciar la clave primaria de otra tabla. Juntas, estas claves permiten establecer relaciones coherentes y seguras entre los datos en una base de datos relacional.

**3. \*\*Pregunta de Código: \*\***

- Escribe una consulta SQL para insertar tres registros en la tabla employees creada en la pregunta 2.

cargar script **3addDataEmployess.sql**

**4. \*\*Pregunta de Código: \*\***

- Muestra cómo actualizar el salario de un empleado específico en la tabla employees. Por ejemplo, actualiza el salario del empleado con id = 1 a 60000.00.

cargar script **4updateSalaryEmployer.sql**

**5. \*\*Pregunta de Código: \*\***

- Escribe una consulta SQL para seleccionar todos los empleados cuyo salario sea mayor a 50000.00 y ordenarlos por el campo hire\_date en orden descendente.

cargar script **5seletcAndOrderByDesc.sql**

**6. \*\*Pregunta Teórica: \*\***

**- ¿Qué es una transacción en MySQL y cómo se utiliza? Proporciona un ejemplo de uso.**

Una transacción en MySQL es una secuencia de operaciones de base de datos que se realizan como una sola unidad lógica y atómica de trabajo. Esto significa que o bien todas las operaciones de la transacción se completan con éxito y se guardan permanentemente en la base de datos, o bien ninguna de ellas se realiza y la base de datos se mantiene en su estado original. Esto asegura la integridad y la consistencia de los datos.

Para utilizar transacciones en MySQL, necesitas ejecutar las operaciones dentro de una secuencia de comandos SQL, donde agrupas las operaciones que deseas que se realicen como una única unidad lógica de trabajo.

**Iniciar una Transacción**: Utiliza el comando START TRANSACTION para iniciar una transacción.

**Ejecutar Operaciones**: Esto puede incluir actualizaciones, inserciones, eliminaciones o cualquier otro comando SQL válido.

**Confirmar la Transacción**: Si todas las operaciones se ejecutan con éxito y deseas guardar los cambios permanentemente en la base de datos, utiliza el comando COMMIT para confirmar la transacción.

**Cancelar la Transacción (Rollback)**: Si ocurre algún error durante la transacción o si deseas revertir todas las operaciones realizadas dentro de la transacción, puedes utilizar el comando ROLLBACK.

**Ejemplo de uso**

START TRANSACTION;

//Insertar un nuevo empleado

INSERT INTO employees (name, position, salary, hire\_date)

VALUES ('Wendy', 'Analista', 12000.00, '2024-06-06');

//Actualizar el salario de un empleado existente

UPDATE employees

SET salary = 70000.00

WHERE id = 2;

//Eliminar a un empleado

DELETE FROM employees WHERE id = 3;

//Confirmar la transacción

COMMIT;

**7. \*\*Pregunta de Código: \*\***

- Crea una vista en MySQL llamada high\_earning\_employees que seleccione todas las columnas de los empleados cuyo salario sea mayor a 70000.00.

cargar script **7viewSalaryHigher7000.qsl**