Introducción a las preguntas teóricas:

Como sabemos, en la actualidad existen grandes modelos de inteligencia artificial con la capacidad de responder la mayoría de preguntas que podamos realizar como reclutadores. Si bien las respuestas van a ser evaluadas, la finalidad de la parte teórica es ser una guía para que el postulante pueda entender la orientación de la posición y afianzarse con los conocimientos necesarios para su desarrollo laboral.

Posteriormente a la evaluación del trabajo práctico, podrá existir una instancia de conversación donde validemos los conocimientos y el entendimiento de los conceptos.

Teoría:

Preguntas generales sobre HTTP/HTTPS:

¿Qué es HTTP y cuál es su función principal?

HTTP (Hypertext Transfer Protocol) es un protocolo que permite la comunicación entre clientes y servidores web. Permite que los clientes soliciten recursos y los servidores los entreguen.

¿Cuál es la diferencia entre HTTP y HTTPS?

HTTP transmite la información en texto plano sin cifrar, por lo cual los datos pueden ser interceptados o modificados. En contraste, HTTPS usa SSL/TLS para cifrar la comunicación entre cliente y servidor.

¿Cómo funciona el proceso de cifrado en HTTPS?

El cliente envía una solicitud al servidor para iniciar una conexión.
El servidor envía su certificado SSL/TLS, emitido por una Autoridad de Certificación (CA) que contiene su clave pública.
El cliente verifica el certificado.
El cliente envía una clave simétrica cifrada con la clave pública del servidor.
El servidor descifra el mensaje con su clave privada y obtiene la clave simétrica.
A partir de ese momento toda la comunicación se cifra con esta clave simétrica.

¿Qué es un certificado SSL/TLS y cuál es su importancia en HTTPS?

En un archivo digital que permite identificar a un servidor web. Es de vital importancia en HTTPS ya que garantiza que el cliente se conecta a un servidor legitimo y permite iniciar el handshake de TLS/SSL para establecer un canal de comunicación encriptado y privado.

¿Qué es un método HTTP? ¿Podrías enumerar algunos de los más utilizados?

Es una acción que especifica que operación quiere realizar un cliente sobre un recurso en un servidor web.
Métodos más usados:

GET: obtener un recurso
POST: enviar datos para que se procesen o crear un nuevo recurso.
PUT: crear un recurso, y si este ya existe, lo reemplaza completamente.
DELETE: eliminar un recurso.
PATCH: modificar parcialmente un recurso.

Explica las diferencias entre los métodos HTTP GET y POST.

La principal diferencia es que GET se utiliza para recuperar un recurso del servidor, mientras que POST se utiliza para enviar datos al servidor para que se procesen o creen un nuevo recurso.
Los datos en GET se envían a través de la URL, y en POST a través del cuerpo de la request.

¿Qué es un código de estado HTTP? ¿Podrías mencionar algunos de los más comunes y lo que significan?

Un código de estado HTTP es un numero (tres dígitos) que el servidor devuelve al cliente en la respuesta, para indicar el resultado de la request (si fue exitosa, si hubo un error). Están divididos en cinco categorías (identificados por el primer digito): Informativa (1xx), Exitosa (2xx), Redirección (3xx), Error del Cliente (4xx) y Error del servidor (5xx)

Ejemplos:

200: ok (Respuesta correcta)
201: created (Recurso creado exitosamente)
204: No content (Request exitosa, pero no hay nada que devolver)

301: Moved Permanently (El recurso solicitado ha sido movido a una nueva URL de forma permanente)
302: Found (El recurso se encuentra temporalmente en otra URL)
304: Not modified (El recurso no se modificó desde la última vez que se lo solicito)

400: Bad Request (La request está mal hecha)
401: Unauthorized (Se necesita autenticación)
403: Forbidden (Acceso denegado, aunque el cliente este autenticado)
404: Not Found (Recurso no encontrado)
405: Method Not Allowed (El método HTTP utilizado no está permitido para ese recurso)

500: Internal Server Error (Error genérico del servidor)
502: Bad Gateway (El servidor recibió una respuesta invalida del otro servidor)
503: Service Unavailable (El servicio no está disponible)
504: Gateway Timeout (El servidor no respondió a tiempo)

¿Qué es una cabecera HTTP? Da ejemplos de cabeceras comunes.

Un header HTTP es una parte de la request o respuesta que contiene metadatos de la comunicación. Estos permiten al cliente y al servidor enviar información adicional junto a una request o respuesta. Ejemplos:

Host: indica el dominio al cual se le realiza la request.

User-Agent: identifica el navegador o la aplicación del cliente que realiza la solicitud.

Content-Type: para especificar el tipo de medio de los datos en el cuerpo de la request o respuesta.

Content-Lenght: especifica el tamaña del cuerpo de la respuesta en bytes.

Date: La fecha y hora en que se originó el mensaje

Server: contiene información sobre el software del servidor web que respondió a la request.

Accept: indica los tipos de datos que el cliente puede procesar.

Accept-Language: informa al servidor sobre el lenguaje que el cliente prefiere.

Cookie: contiene los datos de cookies que el cliente envía al servidor para mantener el estado de la sesión.

Set-Cookie: utilizada por el servidor para enviar una cookie al cliente y que el browser la almacene para futuras request.

Authorization: contiene las credenciales para autenticar a un usuario con un servidor.

Location: indica la nueva URL a la que el cliente debe redirigirse.

Cache-Control: reglas de cache.

¿En qué consiste el concepto de "idempotencia" en los métodos HTTP? ¿Qué métodos cumplen con esta característica?

Que un método sea idempotente significa que al utilizar el mismo método varias veces tendrá el mismo efecto que al haberlo realizado una sola vez. Los métodos que cumplen con esta característica son: GET, PUT, DELETE, HEAD y OPTIONS.

¿Qué es un redirect (redirección) HTTP y cuándo es utilizado?

Una redireccion HTTP es una respuesta del servidor que le indica al cliente que el recurso solicitado no está disponible en la URL original y que, debe solicitarlo en otra URL. El servidor le devuelve un código de estado 3xx junto con el header LOCATION la cual contiene la nueva URL.

Se utiliza cuando:
Se cambia el dominio (de http://ejemplo.es a http://ejemplo.com).
Para obligar a los usuarios a utilizar siempre HTTPS (de http://ejemplo.com a https://ejemplo.com).
Evitar contenido duplicado, redirigiendo todo a una URL principal (de https://ejemplo.com a https://www.ejemplo.com).
Recursos en mantenimiento o movidos temporalmente a otra URL.
Cambia la estructura de la URL (de /ropa/campera-negra a /ropa/campera/negro) para que los enlaces antiguos sigan funcionando.

Preguntas técnicas y de seguridad en HTTP/HTTPS:

¿Cómo se asegura la integridad de los datos en una conexión HTTPS?

Antes de iniciar la comunicación con el handshake el cliente se asegura que el servidor sea legitimo verificando el CA.
Toda la comunicación entre el cliente y el servidor esta cifrada. Ademas, cada bloque de datos transmitido tiene un código de autenticación (MAC/HMAC), que SSL/TLS usa para verificar que los datos no se han modificado.

¿Qué diferencia hay entre un ataque de "man-in-the-middle" y un ataque de "replay" en un contexto HTTPs?

En un ataque “man in the middle” el atacante se interpone en el medio de la comunicación entre el cliente y el servidor, actuando de intermediario. El atacante puede espiar o modificar la información que se transmite. En contraste, en un ataque de “replay” el atacante no modifica el mensaje, sino que lo captura y lo retransmite para tratar de obtener un efecto no autorizado.

Explica el concepto de "handshake" en HTTPS.

El handshake es el proceso que ocurre al inicio de una conexión segura entre un cliente y un servidor para obtener un canal de comunicación seguro y cifrado. El cliente y el servidor acuerdan las reglas para comunicarse antes de empezar a enviar información sensible.

Proceso:

El cliente envía una solicitud al servidor para iniciar una conexión.
El servidor envía su certificado SSL/TLS, emitido por una Autoridad de Certificación (CA) que contiene su clave pública.
El cliente verifica el certificado.
El cliente envía una clave simétrica cifrada con la clave pública del servidor.
El servidor descifra el mensaje con su clave privada y obtiene la clave simétrica.
A partir de ese momento toda la comunicación se cifra con esta clave simétrica.

¿Qué es HSTS (HTTP Strict Transport Security) y cómo mejora la seguridad de una aplicación web?

HSTS es un mecanismo de seguridad que obliga a los browsers a comunicarse únicamente a traves HTTPS, incluso si el cliente intenta acceder usando HTTP.
Esto previene ataques “man-in-the-middle” y “downgrade”
Previene el robo de cookies
Evita que los usuarios accedan a sitios con certificados SSL/TLS no validos o caducados.

¿Qué es un ataque "downgrade" y cómo HTTPS lo previene?

Es un tipo de ataque en la que se fuerza a dos partes que se comunican (cliente y servidor) a utilizar un protocolo o algoritmos de seguridad antiguos o obsoletos para aprovechar vulnerabilidades conocidas.
HTTPS lo previene con las versiones actuales de TLS, ya que durante el handshake el servidor envía una flag si responde con una versión inferior a la que manda el cliente. Además, no permite utilizar algoritmos de cifrado obsoletos. Y por último con HSTS, evita que un atacante redirija un sitio de HTTPS a HTTP.

¿Qué es el CORS (Cross-Origin Resource Sharing) y cómo se implementa en una aplicación web?

CORS es un mecanismo de seguridad basados en headers HTTP que permite a un navegador permitir o denegar el acceso a un recurso que reside en un dominio diferente al de la app web que lo solicita.
Se implementa de la siguiente forma:
El navegador hace la request con el header Origin, que indica el dominio de origen
El servidor al recibir este header responde con headers CORS (como Access-Control-Allow-Origin o Access-Control-Allow-Methods).
El navegador comprueba el valor del header Access-Control-Allow-Origin con el que hizo la request. Si hay coincidencia se permite la solicitud, y si no hay coincidencia se bloquea.

¿Qué diferencia hay entre una cabecera Authorization y una cabecera Cookie?

Aunque los dos headers se utilizan para la autenticación y gestión de estado de sesión tienen diferencias importantes en cómo se usan:

Generalmente el header Authorization lo gestiona el cliente mientras que Cookie lo gestiona el navegador.
Authotization utiliza tokens o credenciales. Cookie usa un formato Clave: valor.
Authotization es stateless, es decir, el servidor no recuerda nada y hay que enviar manualmente el token en cada request. En cambio, Cookie es stateful, es decir, que el servidor guarda la sesión y el navegador se encarga de enviar automáticamente las cookies.

¿Qué son las cabeceras de seguridad como Content-Security-Policy o X-Frame-Options? ¿Cómo ayudan a mitigar ataques comunes?

Los headers de seguridad son aquellas que el servidor envía junto a una respuesta para indicarle al browser como cargar o ejecutar recursos.
Por ejemplo: Content-Security-Policy define que orígenes de contenido están disponibles para tu pagina y ayuda a mitigar XSS al evitar que scripts maliciosos se ejecuten.
Otro ejemplo sería el header HSTS que obliga a usar HTTPS y protege contra ataques downgrade.

¿Cuáles son las diferencias entre HTTP/1.1, HTTP/2 y HTTP/3?

HTTP/1.1 y HTTP/2 utilizan TCP mientras que HTTP/3 utiliza QUIC que se ejecuta sobre UDP.
HTTP/1.1 no tiene multiplexación, por lo tanto, permite únicamente una sola solicitud-respuesta en una conexión TCP. HTTP/2 y HTTP/3 si permiten varios solicitudes-respuestas en una única conexión.
Al no tener multiplexación en HTTP/1.1 una solicitud lenta bloquea las siguientes. En HTTP/2 esto no pasa pero se puede bloquear por TCP (al perderse un paquete y tener que retransmitirlo se frena los otros paquetes). Y en HTTP/3 al ser en UDP los flujos de datos son independientes y no hay bloqueos.
HTTP/1 no comprime los datos de los headers. En cambio, HTTP/2 y HTTP/3 sí.
HTTP/1 no tiene “server push” (el servidor envia recursos que el cliente va a necesitar antes de que este los solicite), HTTP/2 y HTTP/2 si lo poseen.
HTTP/3 es el único que tiene integrado TLS por QUIC.

¿Qué es un "keep-alive" en HTTP y cómo mejora el rendimiento de las aplicaciones?

“Keep-alive” es un mecanismo que permite que una única conexión TCP permanezca abierta para múltiples requests y respuestas HTTP entre un cliente y un servidor, en lugar de cerrarla y reabrirla después de cada intercambio de datos.
Esto hace que se reduzca la latencia ya que cada conexión TCP requiere de un three-way handshake, y al permanecer abierta la conexión se evita este paso.
Reduce el uso de socket para la conexión, lo cual hace que se utilice menos recursos del servidor.
Mejora la velocidad de carga de paginas con muchos recursos, ya que, todos los recursos pueden ser enviados por la misma conexión TCP, en lugar de abrir varias conexiones separadas.

Preguntas de implementación práctica:

¿Cómo manejarías la autenticación en una API basada en HTTP/HTTPS? ¿Qué métodos conoces (Basic, OAuth, JWT, etc.)?

Depende de la política de seguridad que haya, la arquitectura que va a tener la API, si es compatible con el entorno que van a utilizar los clientes.

HTTP Basic Autentication: El cliente envia las credenciales cifradas en Base64 en cada solicitud en el header Authorization.

JWT: basado en tokens que se envían en cada request en el header Authorization

¿Qué es un proxy inverso (reverse proxy) y cómo se utiliza en entornos HTTP/HTTPS?

Un proxy inverso es un servidor que hace de intermediario entre el cliente y uno o mas servidores web. El cliente envía la petición a un servidor, este en lugar de conectarse directamente con el servidor de origen se conecta al proxy. Este último recibe la petición y la reenvía al servidor de origen correspondiente. El servidor de origen procesa la petición y la envía la respuesta al proxy, y por último, el proxy recibe la respuesta del servidor y la envía al cliente.
Esto permite hacer un balanceo de carga, ya que el trafico se distribuye a través de múltiples servidores web.
Mejora la seguridad ya que la dirección del servidor de origen no se expone a los clientes.
Los proxys pueden almacenar en cache las respuestas a peticiones comunes, lo que reduce la carga de los servidores y el tiempo de respuesta de la entrega de los recursos al cliente.

¿Cómo implementarías una redirección automática de HTTP a HTTPS en un servidor?

Que cualquier request que llegue por el puerto 80 (HTTP) en un servidor para un dominio en específico se redirija con un 301 a HTTPS.

¿Cómo mitigarías un ataque de denegación de servicio (DDoS) en un servidor HTTP?

Configurando el servidor para limitar el numero de request para una misma IP en un intervalo de tiempo.
Haciendo un balanceo de carga para distribuir el tráfico de un servidor si este saturado.
Bloqueando las direcciones IP maliciosas.

¿Qué problemas podrías enfrentar al trabajar con APIs que dependen de HTTP, y cómo los resolverías?

Que no utilice HTTPS: esto lo resolvería haciendo una redirección automática de HTTP a HTTPS

Que no tengan poca o nula autenticación: implementando algún método de autenticación.

Que los request sean lentos: implementar algún método de cache de respuestas (en un proxy inverso o del lado del cliente).

Problemas con CORS: utilizando los headers CORS (como Access-Control-Allow-Origin).

¿Qué es un cliente HTTP? ¿Mencionar la diferencia entre los clientes POSTMAN y CURL?

Un cliente HTTP es cualquier dispositivo, software o aplicación que se comunica con un servidor web a través de HTTP.
CURL es una herramienta de la línea de comandos para transferir datos usando protocolos. No posee una interfaz de usuario. Se utiliza en scripts, automatizaciones y pruebas rápidas desde la terminal.
POSTMAN es una aplicación de escritorio (o web) con una interfaz de usuario que facilita la creación de peticiones HTTP. Se utiliza para desarrollar y depurar APIs.

Preguntas de GIT

¿Qué es GIT y para qué se utiliza en desarrollo de software?

GIT es un sistema de control de versiones distribuido. Permite registrar cada cambio que se realiza en el código o los archivos de un proyecto (con commits). Que sea distribuido significa que cada desarrollador que trabaje en el proyecto pueda tener una copia completa del repositorio, es decir, no depende de un único servidor.
En el desarrollo de software esto permite:
Tener el historial de modificaciones hechas del código a lo largo del tiempo y facilita volver a una versión anterior ante una falla.
Colaboración en equipo, ya que cada desarrollador puede tener un repositorio local y trabajar con el mismo. Además, GIT permite fusionar los cambios y ayudar a resolver los conflictos.
Trabajar con ramas para realizar nuevas funcionalidades o corregir errores sin afectar la rama principal.

¿Cuál es la diferencia entre un repositorio local y un repositorio remoto en GIT?

El repositorio local se encuentra en tu propio pc y contiene todo el historial de commits, ramas y versiones de un proyecto. También posibilita a los desarrolladores trabajar sin conexión a internet.
En cambio, un repositorio remoto está alojado en un servidor o servicio externo (github, gitlab, etc.), y facilita la colaboración entre varios desarrolladores. Permite centralizar los cambios de un proyecto para sincronizar varios repositorios locales.
Los desarrolladores suben sus cambios de su repositorio local hacia el repositorio remoto.

¿Cómo se crea un nuevo repositorio en GIT y cuál es el comando para inicializarlo? Explica la diferencia entre los comandos git commit y git push.

Para crear un nuevo repositorio en git debes abrir la línea de comandos y navegar hasta la raíz de tu proyecto, y en esa ubicación con el comando git init se inicializa el repositorio. Esto creara una carpeta oculta “.git” la cual guardara todo el historial.

Git commit: guarda los cambios realizados en tu repositorio local.
Git push: sube los commits de tu repositorio local a un repositorio remoto.

¿Qué es un "branch" en GIT y para qué se utilizan las ramas en el desarrollo de software?

Un “Branch” es un puntero hacia un commit especifico. Cuando se crea una rama, se bifurca el estado actual del código generando una línea de trabajo independiente de la rama desde la que se creó (puede ser la main u otra, y pueden coexistir varias ramas al mismo tiempo).
Esto beneficia al desarrollo de software en:
Permite que múltiples desarrolladores puedan tener su propia rama del mismo proyecto al mismo tiempo sin pisarse los cambios.
Facilita crear nuevas features, corregir bugs o experimentar con el código sin afectar la estabilidad del código principal. Si hay un error la rama simplemente se puede borrar.
Cuando un desarrollador sube sus cambios al repositorio remoto puede pedir que otros colaboradores lo revisen y prueben antes de ser mergeado.
También se puede tener varias ramas para mantener diferentes versiones del software, como una rama de desarrollo o la rama main para la versión en producción.

¿Qué significa hacer un "merge" en GIT y cuáles son los posibles conflictos que pueden surgir durante un merge?

Un “merge” en GIT es el proceso de fusionar los cambios de una rama en otra.
Los posibles conflictos son:
Edición en la misma línea de un archivo.
Una rama elimina un archivo y la otra lo modifica.
Mover/Renombrar archivos/carpetas en una rama y modificarlos en otra.

Siempre que aparezca un conflicto el developer deberá resolverlo manualmente.

Describe el concepto de "branching model" en GIT y menciona algunos modelos comunes (por ejemplo, Git Flow, GitHub Flow).

Un branching model es una estrategia de como se crean, usan y mergean las ramas dentro de un proyecto.

Git Flow:
Ramas principales: main (código en produccion) y develop (código en desarrollo).
Ramas de soporte: feature (nuevas funcionalidades) , realese (preparación de una nueva versión) y hotfix (correcciones urgentes en producción).

GitHub Flow:
Solo una rama principal (main) y todas las features o corrección de bugs van en una rama desde main y cuando se terminan se hace PR para revisar el código y cuando se aprueba se hace merge en main.

Trunk-based development:
Todos los developers trabajan en una sola rama y estos mergean sus cambios al menos una vez al día. Se utilizan feature flags para activar o desactivar features en producción para no romperlo

¿Cómo se deshace un cambio en GIT después de hacer un commit pero antes de hacer push?

Si los cambios aun no aun sido pusheados se puede utilizar el comando git reset.
Habiendo usado previamente el comando git log y copiado el hash id del commit a deshacer, usamos el comando git reset <hash id commit a deshacer>.

¿Qué es un "pull request" y cómo contribuye a la revisión de código en un equipo?

Un PR es una petición que hace un desarrollador en plataformas (como github) para que sus cambios en una rama sean revisados y fusionados en otra rama (si se aprueban).
Esto permite que otros miembros del equipo puedan leer los cambios, dejar comentarios, sugerencias, mejoras u otra implementación.

¿Cómo puedes clonar un repositorio de GIT y cuál es la diferencia entre git clone y git pull?

Se puede clonar un repositorio remoto con el comando git clone <url del repositorio>.
Git clone hace una copia de un repositorio remoto en tu PC (se hace una sola vez al inicio).
En cambio, git pull actualiza el repositorio local con los últimos cambios del remoto.

Preguntas de PHP con Laravel

¿Qué es Laravel?

Es un framework open source para desarrollar aplicaciones web con PHP, el cual ofrece una gran cantidad de herramientas y funcionalidades para facilitar el desarrollo.

¿Cómo maneja Laravel el modelo de ejecución de peticiones HTTP y en qué se diferencia del manejo tradicional en PHP puro?

Ciclo de vida de una request HTTP en Laravel:
Todas las requests tienen un único punto de entrada, el archivo “public/index.php” .
Este archivo carga el autoload del Composer, arranca la aplicación y crea una instancia de esta (service container, que es responsable de gestionar las dependencias y componentes).
Luego, el request se pasa al HTTP Kernel, el cual ejecuta un stack de middlewares de la aplicación y definir una matriz de bootstrappers (como service providers).
Luego de pasar el stack de middlewares, el request es pasado al router el cual examina la URL para asociarla al controller correcto y además ejecuta middlewares específicos de esta.
Después, el controller recibe la request y ejecuta la lógica del negocio generando una respuesta que se envía devuelta por el stack de middlewares.
Y por último el kernel devuelve la respuesta al servidor web que la devuelve al cliente.

En PHP puro en contraste, el desarrollador es el encargado de gestionar cada aspecto mencionado arriba.

Diferencias:

Cada archivo PHP es un endpoint mientras que en Laravel index.php recibe todas las request.
En PHP puro cada archivo contiene toda la lógica mezclada, Laravel usa el patron MVC.
En PHP puro el desarrollador tiene que implementar manualmente medidas de seguridad (como validaciones, CSRF), y en Laravel están integradas.
Laravel utiliza un sistema de enrutamiento flexible, además de usar middlewares, controllers, view, models.

¿Qué es el ciclo de vida de una petición en Laravel y cuál es el rol del Kernel (HTTP Kernel) en dicho proceso?

Es la serie de pasos que laravel hace para manejar una request y devolver la respuesta. El kernel es como un orquestador que toma la request la pasa por los filtros necesarios (middlewares), la envia al router y luego devuelve la respuesta procesada al cliente.
Tambien, se encarga de hacer el boteo de los services providers (componente, bases de datos, colas, validaciones, etc.)

¿Cuál es la diferencia entre require/include de PHP y el sistema de autoloading de Composer en Laravel?

Require/include son instrucciones para incluir manualmente archivos.
Esta tiene ciertos problemas como:
El programador tiene que incluir manualmente todos los archivos que necesita en cada archivo.
Si se actualiza una ruta, hay que actualizar todas las sentencias require que lo referencian.
Si quieres usar librerías externas tenes que descargarlas e incluirlas manualmente.

El autoloading de Composer soluciona estos problemas:
El autoloader de Composer se encarga de cargar automáticamente las clases PHP solo cuando son necesarias.
En vez de incluir el archivo manualmente, se basa en namespaces lo cual tambien permite que se abstraiga de la ruta.
Cuando instalas una librería externa, Composer la descarga y automáticamente la añade al autoloading.

¿Qué es Composer y cuál es su función dentro del ecosistema de PHP y Laravel?

Es una herramienta de gestión de dependencias para PHP.

En PHP se encarga de:
Gestionar dependencias
Autoloading automático
Versionado y control de dependencias
Scripts y automatizaciones antes o después de eventos (como composer install o composer update)

Composer es una pieza fundamente dentro de Laravel ya que:
Para crear un proyecto se hace con Composer

Laravel en si mismo es un conjunto de paquetes de Composer. Composer se encarga de la correcta instalación y actualización de Laravel.
Autoloading para todo el proyecto.
Integracion automática con librerías de terceros

¿Cómo se inicializa un nuevo proyecto de Laravel usando Composer y cuál es el propósito del archivo composer.json?

Para crear un nuevo proyecto en Laravel usando Composer se usa el comando:
composer create-project laravel/laravel “nombre del proyecto”

Composer.json es el archivo de configuración principal de Composer en el proyecto. Su propósito principal es:

Declarar las dependencias que el proyecto necesita
Restricciones de compatibilidad
Configurar el autoloading
Definir scripts personalizados para eventos
Especificar metadatos del proyecto

¿Qué son las dependencias en Composer y cómo se instalan? Explica la diferencia entre dependencias normales y dependencias de desarrollo.

Una dependencia es una librería o paquete externo que un proyecto necesita para funcionar. Estas dependencias se declaran en el archivo composer.json dentro de la sección require o require-dev.

Para instalar una nueva dependencia se usa el comando:
composer require “nombre del paquete”

Las dependencias normales son esenciales para el funcionamiento de la aplicación en producción, mientras que, las dependencias de desarrollo solo se usan durante el desarrollo o testing.

¿Cómo puedes gestionar versiones específicas de paquetes en Composer y cuál es el propósito del archivo composer.lock?

Se puede especificar la versión que quieres usar en “composer.json”.
Por ejemplo:

Para una versión exacta: “guzzlehttp/guzzle”: “7.2”
Para un rango de versiones: “guzzlehttp/guzzle”: “>=7.0 <8.0”
Operador ~: “guzzlehttp/guzzle”: “~7.2” equivale a >=7.2.0 pero <8.0.0 version menor
Operador ~: “guzzlehttp/guzzle”: “~7.2.3” equivale a >=7.2.3 pero <7.3.0 version parche
Operador ^: “guzzlehttp/guzzle”: “^7.2” equivale a >=7.2.0 pero <8.0.0 manteniendo la compatibilidad con versiones anteriores

Composer.lock es un archivo generado automáticamente cuando se instalan o actualizan dependencias. Se encarga de registrar las versiones exactas y especificas de todos los paquetes instalados y sus dependencias en un momento dado. Es importante ya que permite reproducibilidad, es decir, que cualquier persona que clone el proyecto puede instalar exactamente las mismas versiones de dependencias.

¿Qué es Eloquent ORM y cómo facilita la interacción con bases de datos en Laravel?

Es el Object-Relational Mapper que viene integrado con Laravel. Permite interactuar con bases de datos utilizando objetos y métodos en PHP, en lugar de escribir directamente SQL. Cada tabla se representa como un modelo de PHP y cada fila como una instancia de ese modelo
Facilita la interacción de la siguiente manera:
No se necesita escribir SQL manualmente sino que la reemplaza por una sintaxis mas sencilla. Por esto tambien es compatible con múltiples motores de base de datos, ya que la sintaxis será siempre la misma. Laravel y Eloquent se encargan de traducir los metodos HTTP a las sentencias SQL correctas.
Al no escribir SQL y usar bindings internos previene de ataques SQL injection.
Posee metodos que simplifican las tareas comunes como: find($id), all(), first(), save(), delete(), etc.
Eloquent gestiona relaciones como métodos en el modelo, lo cual permite acceder a los datos relacionados como si fueran propiedades de tu modelo.

¿Cómo se manejan errores en Laravel? Explica el rol del Handler, el uso de excepciones personalizadas y cómo se combinan con middlewares y validaciones.

El handler es un componente que actúa como único punto de entrada para todos los errores de la aplicación.
Posee dos metodos principales:
Report: se define como reportar las excepciones
Render: determina la respuesta http que se le enviara al cliente dependiendo del tipo de excepción.

Las excepciones personalizadas son clases de PHP que extienden la clase base Exception. Su uso permite a los programadores crear errores que son específicos de la lógica del negocio.

Los middlewares se ejecutan antes de que la request llegue al controller. Estos son un mecanismo para inspecciones y filtrar las requests que entran a la aplicación. Por ejemplo, si un middleware detecta un error (un usuario no está autenticado) puede lanzar una excepción que será capturada por el handler.

Laravel tiene un sistema de validación que, cuando falla, lanza automáticamente una excepción de tipo ValidationException que el handler captura.
Si falla en requests web redirige automáticamente con mensajes de error.
Si falla en APIs devuelve un JSON.

Práctica

Para realizar los ejercicios prácticos deberás contar en tu ambiente de trabajo con las siguientes herramientas:

- Postman
- GIT
- Node.js instalado

La entrega deberá realizarse en un repositorio de código público que se deberá compartir al equipo de reclutamiento. El repositorio deberá contener tanto la parte teórica como la práctica

Actividad práctica número 1
Pasos:

1) Realizar una petición GET a la siguiente URL a través de Postman: https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json

Ejemplo con CURL:

```
curl --location --request GET 'https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json' \
--header 'Content-Type: application/json'
```

2) Realizar una petición POST a la siguiente URL a través de Postman: https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json y con el siguiente body:

```
{
   "name":"TuNombre",
   "suraname":"TuApellido",
   "birthday":"1995/11/16",
   "age":29,
   "documentType":"CUIT",
   "documentNumber":20123456781
}
```

Reemplazar los campos por los valores personales tuyos.

Enlace de la resolución: https://agustin-gonzalez-s-team.postman.co/workspace/Agustin-Gonzalez's-Workspace~984b3863-54d2-42c2-8fc2-b41f5bd0d513/collection/47850998-39b16ace-44fa-4e91-b612-a79a260abc27?action=share&creator=47850998

Actividad práctica número 2

Construir una mini-API en Laravel que:

A. Consuma por **GET** el recurso `https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json` y lo presente de forma legible para humanos.

B. Exponga un **POST /recluta** en PHP con Laravel:

```
{
  "name":"TuNombre",
  "suraname":"TuApellido",
  "birthday":"1995/11/16",
  "documentType":"CUIT",
  "documentNumber":20123456781
}
```

y lo **mapee** a:

```
{
  "name":"TuNombre",
  "suraname":"TuApellido",
  "birthday":"1995/11/16/",
  "age":29,
  "documentType":"CUIT",
  "documentNumber":20123456781
}
```

para luego **POST**earlo a `https://reclutamiento-dev-procontacto-default-rtdb.firebaseio.com/reclutier.json`

Reglas de negocio y validaciones

- `name` y `suraname`: convertir a “Title Case” (primera letra de cada palabra en mayúscula). Si llegan en otro formato, **normalizar**.

- `birthday`: formato **YYYY/MM/DD**. No puede ser posterior a hoy ni anterior a **1900/01/01**. Si no cumple, **rechazar** (400 con detalle).

- `documentType`: sólo **“CUIT”** o **“DNI”**; otros valores → **rechazar**.

- `age`: calcular a partir de `birthday` (años cumplidos al día de la solicitud).

- La salida enviada a Firebase deberá incluir `birthday` **con una barra final** (`.../`) para coincidir con el ejemplo provisto.