

Prompts para código. Parte 1.

Programación de un servidor
de una API HTTP REST automática
con sistema de autenticación
basado en PHP y MySQL
con la asistencia de ChatGPT v3.5

Por: Carlos Jimeno Hernández

ÍNDICE

- Capítulo 1. Prompts para el servidor: creación del modelo de datos.
- Capítulo 2. Prompts para el servidor: sistema de REST autenticado.
- Capítulo 3. Prompts para el servidor: sistema de autenticación y autorización completo.
- Capítulo 4. Consolidación del código.
- Capítulo 5. Minar información de campo mediante tablas SQL.
- Capítulo 6. Tests.
- Capítulo 7. Encapsulamiento de espacio de nombres con POO.
- Capítulo 8. Soporte para hooks.
- Capítulo 9. Soporte para plugins.
- Capítulo 10. Compartirlo en Github.

Capítulo 1. Prompts para la creación de la base de datos.

Explicación:

El primer paso para montar nuestro sistema, bastante definido en el título, es crear unas tablas necesarias para el sistema de autenticación y autorización. Sin ellas, la aplicación prácticamente carecería de sentido o abarcaría escasos escenarios reales de producción.

Por eso, lo primero que nos proponemos es definir las tablas MySQL para el sistema de autenticación y autorización. Luego nos ocuparemos del servidor.

Prompt 1. Hola.

Prompt 2. Puedes hacer un script para MySQL con el diseño de un sistema de autenticación y autorización. El script debe tener las tablas: usuarios, grupos, permisos, sesiones. Además:

- debe añadir mediante INSERTs a un usuario 'administrador' con la contraseña 'pordefecto'
- debe añadir a un grupo 'administración global' con una descripción al uso
- debe añadir a un permiso 'administrar globalmente' con una descripción al uso

Las tablas de relaciones deben seguir la nomenclatura {tabla 1}_y_{tabla 2}.

Las tablas y columnas deben tener una nomenclatura snake case.

Las tablas deben crearse con la sintaxis de MySQL "CREATE IF NOT EXISTS".

Antes de crearse las tablas, debería de intentar eliminarse en orden inverso al que son creadas, con la sentencia "DROP TABLE IF EXISTS".

Finalmente, el usuario 'administrador' debe pertenecer al grupo 'administración global' y este grupo debe estar relacionado con el permiso 'administrar globalmente'.

Prompt 3. Ahora eliminando la tabla de sesiones y permisos.

Prompt 4. Volvamos al script anterior. Intenta que no aparezca la tabla de sesiones_y_permisos solamente.

```
-- Eliminar tablas de relación en orden inverso
DROP TABLE IF EXISTS grupos_y_permisos;
DROP TABLE IF EXISTS usuarios_y_grupos;

-- Eliminar tablas en orden inverso
DROP TABLE IF EXISTS sesiones;
DROP TABLE IF EXISTS permisos;
DROP TABLE IF EXISTS grupos;
DROP TABLE IF EXISTS usuarios;

-- Crear tabla 'usuarios'
CREATE TABLE IF NOT EXISTS usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre_usuario VARCHAR(255) NOT NULL,
    contraseña VARCHAR(255) NOT NULL
);

-- Crear tabla 'grupos'
CREATE TABLE IF NOT EXISTS grupos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre_grupo VARCHAR(255) NOT NULL,
    descripcion VARCHAR(255)
);

-- Crear tabla 'permisos'
CREATE TABLE IF NOT EXISTS permisos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre_permiso VARCHAR(255) NOT NULL,
    descripcion VARCHAR(255)
);

-- Crear tabla de relación 'usuarios_y_grupos'
CREATE TABLE IF NOT EXISTS usuarios_y_grupos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    usuario_id INT,
    grupo_id INT,
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id),
    FOREIGN KEY (grupo_id) REFERENCES grupos(id)
);

-- Crear tabla de relación 'grupos_y_permisos'
```

```
CREATE TABLE IF NOT EXISTS grupos_y_permisos (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    grupo_id INT,  
    permiso_id INT,  
    FOREIGN KEY (grupo_id) REFERENCES grupos(id),  
    FOREIGN KEY (permiso_id) REFERENCES permisos(id)  
);  
  
-- Insertar usuario 'administrador'  
INSERT INTO usuarios (nombre_usuario, contraseña) VALUES ('administrador', 'pordefecto');  
  
-- Insertar grupo 'administración global'  
INSERT INTO grupos (nombre_grupo, descripcion) VALUES ('administración_global', 'Grupo de  
administración global');  
  
-- Insertar permiso 'administrar globalmente'  
INSERT INTO permisos (nombre_permiso, descripcion) VALUES ('administrar_globalmente', 'Permiso  
para administrar globalmente');  
  
-- Asociar usuario 'administrador' al grupo 'administración global'  
INSERT INTO usuarios_y_grupos (usuario_id, grupo_id) VALUES ((SELECT id FROM usuarios WHERE  
nombre_usuario = 'administrador'), (SELECT id FROM grupos WHERE nombre_grupo =  
'administración_global'));  
  
-- Asociar grupo 'administración global' al permiso 'administrar globalmente'  
INSERT INTO grupos_y_permisos (grupo_id, permiso_id) VALUES ((SELECT id FROM grupos WHERE  
nombre_grupo = 'administración_global'), (SELECT id FROM permisos WHERE nombre_permiso =  
'administrar_globalmente'));
```

Capítulo 2. Prompts para el servidor: sistema de autenticación y autorización.

Explicación:

Una vez hemos conseguido esas tablas, podemos proceder a la codificación del servidor en sí. Lo más deseable, en este caso y normalmente con Chat GPT, va a ser condensar a 1 script nuestro objetivo. Y un lenguaje como PHP si lo permite, mientras que uno como Java nos limitaría bastante. Así, si puedo conseguir reunir toda la lógica posible en 1 mismo fichero, será más fácil portarlo o seguirlo usando en otros prompts.

En este prompt se puede ver que más no siempre es más. Y que menos no siempre es menos. Es decir: no por usar muchas palabras, Chat GPT va a, necesariamente, llegar a tus objetivos. En cambio, el poder utilizar una palabra, una abreviatura, conceptos abstractos pivotantes aunque sean complicados, a veces marca la diferencia entre un prompt kilométrico y un prompt breve pero conciso y eficaz. Me refiero al prompt 7 fallido, y al prompt 8 bastante exitoso. Por poner un ejemplo, que jugar a los detalles no siempre sale bien. He obviado pegar la respuesta, pero tenía rarezas varias. Aunque, como de costumbre, se acercaba bastante, y te ahorraría mucho código.

Por tanto, esta comparación de prompts sirve para ver 2 cosas, entre otras quizá. La primera es que no por mucho escribir, va a ser mejor prompt necesariamente. Sin embargo, en el capítulo 1, sí ha salido bien ser muy conciso. Personalmente, lo achaco a que SQL y concretamente la creación de tablas, es un lenguaje mucho más simple sintácticamente que PHP. Además, considerablemente, dado que hemos reducido la salida a un SQL, y a un tipo de sentencia concreto. PHP en cambio es mucho más capaz y amplio, y por tanto, en su sistema de redes neuronales supongo, hay más nodos posibles que visitar, y eso lo hace más propenso al fallo. Otro aspecto es que esté más trillado, creo, un tema. Por ejemplo, en este capítulo, se ve claro que un prompt que usa una palabra ya dedicada al tema, a un tema ya muy hablado, como CRUD o REST, ya especificamos mucho qué queremos. Es decir, apoyarse en conceptos técnicos, mientras que con las personas suele ser un obstáculo, con Chat GPT es una ventaja. Así, gracias a tener un concepto que se acerca mucho a la idea final de mi objetivo, como es REST o CRUD en este caso, consigue que obtenga mejores salidas.

Prompt 5. Conforme a este diseño de base de datos, que llamaremos database.sql, procedemos a hacer un script en PHP.

Prompt 6. No, a ver. Ahora te voy a decir lo que lleva el script. Escúchame bien.

Prompt 7. El script de PHP debe ser capaz de hacer lo siguiente:

- Recibir obligatoriamente un parámetro POST llamado operacion, y que solo puede ser seleccionar, insertar, actualizar, eliminar.
- Recibir obligatoriamente un parámetro POST llamado tabla, y que solo puede ser uno de los nombres de las tablas del script sql anterior, llamado database.sql.
- Recibir opcionalmente un parámetro POST llamado autenticación.

Una vez hecho esto, el script procede a conectarse a una base de datos MySQL.

Aquí, el script va a construir una consulta SQL, dependiendo del valor del campo operación.

Si el campo operación es seleccionar, hará un SELECT simple a la tabla especificada en el parámetro tabla.

Si el campo operación es insertar, hará un INSERT simple a la tabla especificada en el parámetro tabla y pasándole los valores especificados en el parámetro POST con nombre valor.

Si el campo operación es actualizar, hará un UPDATE simple a la tabla especificada en el parámetro tabla, pasándole los valores especificados en el parámetro POST con nombre valor, y al registro identificado con el parámetro POST con nombre id.

Si el campo operación es eliminar, hará un DELETE simple a la tabla especificada en el parámetro tabla, y al registro identificado con el parámetro POST con nombre id.

El script debe seguir una nomenclatura snake case.

Las entradas de valores por el usuario deben ser sanitizadas debidamente.

El script debe gestionar todos los errores posibles mediante 1 misma función llamada gestionar_error.

Prompt 8. No me gusta. Volvamos a intentarlo con estas instrucciones. El script debe conectarse a una base de datos MySQL y ser capaz de autenticar al usuario mediante un token de sesión y de hacer las operaciones CRUD.

Prompt 9. ¿No deberías cerrar la sesión?

Prompt 10. No, perdón. Quería decir la conexión de la base de datos, me equivoqué.

Prompt 11. Me sirve, pero vamos a quitar lo de cerrar sesión, lo dejaremos para más adelante.

Prompt 12. Por favor, reescribemelo sin esa funcionalidad.

```
<?php
```

```

function gestionar_error($mensaje) {
    echo json_encode(array("error" => $mensaje));
    exit();
}

function conectar_bd() {
    $servername = "tu_servidor";
    $username = "tu_usuario";
    $password = "tu_contraseña";
    $dbname = "tu_base_de_datos";

    $conn = new mysqli($servername, $username, $password, $dbname);

    if ($conn->connect_error) {
        gestionar_error("Error de conexión a la base de datos: " . $conn->connect_error);
    }

    return $conn;
}

function autenticar_usuario($token) {
    $conn = conectar_bd();

    $token_sanitized = $conn->real_escape_string($token);
    $query = "SELECT * FROM sesiones WHERE token = '$token_sanitized'";
    $result = $conn->query($query);

    if ($result->num_rows > 0) {
        $usuario = $result->fetch_assoc();
        return $usuario['usuario_id'];
    } else {
        gestionar_error("Autenticación fallida. Token de sesión no válido.");
    }
}

function seleccionar($tabla) {
    $conn = conectar_bd();
    $query = "SELECT * FROM $tabla";
    $result = $conn->query($query);
    $data = $result->fetch_all(MYSQLI_ASSOC);
    echo json_encode($data);
}

function insertar($tabla, $valores) {
    $conn = conectar_bd();
    $valores_sanitized = array_map(array($conn, 'real_escape_string'), $valores);
    $columnas = implode(',', array_keys($valores_sanitized));
    $valores_str = "'" . implode("', '", $valores_sanitized) . "'";
    $query = "INSERT INTO $tabla ($columnas) VALUES ($valores_str)";
    $conn->query($query);
    echo json_encode(array("mensaje" => "Registro insertado con éxito."));
}

function actualizar($tabla, $valores, $id) {
    $conn = conectar_bd();
    $valores_sanitized = array_map(array($conn, 'real_escape_string'), $valores);
    $set_clause = '';
    foreach ($valores_sanitized as $columna => $valor) {
        $set_clause .= "$columna = '$valor',";
    }
    $set_clause = rtrim($set_clause, ',');
    $query = "UPDATE $tabla SET $set_clause WHERE id = $id";
    $conn->query($query);
    echo json_encode(array("mensaje" => "Registro actualizado con éxito."));
}

function eliminar($tabla, $id) {
    $conn = conectar_bd();

```

```

$query = "DELETE FROM $tabla WHERE id = $id";
$conn->query($query);
echo json_encode(array("mensaje" => "Registro eliminado con éxito.));
}

// Recibir parámetros POST
$token = $_POST['token'] ?? '';
$operacion = $_POST['operacion'] ?? '';
$tabla = $_POST['tabla'] ?? '';
$id = $_POST['id'] ?? '';
$valores = $_POST['valores'] ?? '';

// Autenticar usuario
$usuario_id = autenticar_usuario($token);

// Validar operación
$operaciones_validas = array('seleccionar', 'insertar', 'actualizar', 'eliminar');
if (!in_array($operacion, $operaciones_validas)) {
    gestionar_error("Operación no válida.");
}

// Validar tabla
$tablas_validas = array('usuarios', 'grupos', 'permisos', 'usuarios_y_grupos',
'grupos_y_permisos');
if (!in_array($tabla, $tablas_validas)) {
    gestionar_error("Tabla no válida.");
}

// Ejecutar operación correspondiente
switch ($operacion) {
    case 'seleccionar':
        seleccionar($tabla);
        break;

    case 'insertar':
        $valores_array = json_decode($valores, true);
        insertar($tabla, $valores_array);
        break;

    case 'actualizar':
        $valores_array = json_decode($valores, true);
        actualizar($tabla, $valores_array, $id);
        break;

    case 'eliminar':
        eliminar($tabla, $id);
        break;

    default:
        gestionar_error("Operación no válida.");
}

?>

```

Capítulo 3. Prompts para el servidor: sistema de autenticación y autorización completo

Explicación:

Ahora que ya tenemos la base de datos, y tenemos el script base para una aplicación REST, nos hemos quitado 2 cosas grandes. Toca pensar la 3ª cosa grande que nos podemos sacar. Y una cuestión delicada son las queries del sistema de autenticación y autorización. Luego nos encargaremos de integrarlo con el resto, y de hacer que todo funcione correctamente, pero la gran mayoría si no todo el código que hemos generado, es código bastante válido, y con una lógica bastante indiscutible, porque además es conciso, va sin rodeos a la faena, es código muy limpio, yo me sentía orgulloso de haber hecho una API REST con 300 líneas, y Chat GPT me ha barrido. Debo profundizar sobre este tema, y por eso escribo esto.

Pues bueno, las queries del sistema de autorización las averiguaremos a medida que sigamos el ciclo natural de vida de un usuario:

- Registro al sistema.
- Confirmación al sistema.
- Iniciar sesión.
- Refrescar sesión.
- Cerrar sesión.
- Olvido de credenciales.
- Recuperación de credenciales.
- Baja del sistema.

Estos son los pasos que un sistema de autenticación y autorización, de base, debe cubrir. Por eso, voy a poner un trazo de mis intentos para que Chat GPT me genere una salida lo más cercana posible a mi objetivo.

Prompt 13. Podrías ahora hacer un ejemplo de sistema de autenticación en PHP que cubra los siguientes aspectos de una aplicación:

- Registro al sistema. Con un envío de correo electrónico usando el servicio de la API de Gmail.
- Confirmación al sistema.
- Iniciar sesión.
- Refrescar sesión.
- Cerrar sesión.
- Olvido de credenciales. Con un envío de correo electrónico usando el servicio de la API de Gmail.
- Recuperación de credenciales.
- Baja del sistema.

Prompt 14. Puedes hacer que los datos persistan en la base de datos que hemos creado anteriormente?

Prompt 15. Puedes hacer un ejemplo completo de autorización en PHP que cubra el registro, la confirmación, el login, el logout, el olvido de credenciales, la recuperación de credenciales y la baja al sistema, todo basado en una supuesta base de datos MySQL?

```
<?php
session_start();
require_once('db.php');

function generar_token($longitud = 20) {
    return bin2hex(random_bytes($longitud));
}

function enviar_correo($destinatario, $asunto, $mensaje) {
    // Aquí puedes implementar el código para enviar correos electrónicos utilizando Gmail u
    otro servicio.
}

function registrar_usuario($email, $password) {
    $conn = conectar_bd();

    // Verificar si el usuario ya está registrado
    $stmt = $conn->prepare("SELECT * FROM usuarios WHERE email = ?");
    $stmt->bind_param("s", $email);
```



```

$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    echo json_encode(array("error" => "El usuario ya está registrado."));
    return;
}

// Insertar usuario en la base de datos
$hash_password = password_hash($password, PASSWORD_DEFAULT);
$token = generar_token();

$stmt = $conn->prepare("INSERT INTO usuarios (email, password, confirmado, token) VALUES (?, ?, 0, ?)");
$stmt->bind_param("sss", $email, $hash_password, $token);
$stmt->execute();

// Enviar correo de confirmación
$mensaje = "¡Gracias por registrarte! Para confirmar tu cuenta, haz clic en el siguiente enlace:\n\n";
$mensaje .= "http://tudominio.com/confirmar.php?email=$email&token=$token";
enviar_correo($email, "Confirmación de Registro", $mensaje);

echo json_encode(array("mensaje" => "Te has registrado con éxito. Confirma tu cuenta a través del enlace enviado a tu correo."));
}

function confirmar_cuenta($email, $token) {
    $conn = conectar_bd();

    // Verificar si el token coincide con el almacenado en la base de datos
    $stmt = $conn->prepare("SELECT * FROM usuarios WHERE email = ? AND token = ? AND confirmado = 0");
    $stmt->bind_param("ss", $email, $token);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        // Confirmar la cuenta
        $stmt = $conn->prepare("UPDATE usuarios SET confirmado = 1 WHERE email = ?");
        $stmt->bind_param("s", $email);
        $stmt->execute();

        echo json_encode(array("mensaje" => "Tu cuenta ha sido confirmada con éxito."));
    } else {
        echo json_encode(array("error" => "Token de confirmación no válido o cuenta ya confirmada."));
    }
}

function iniciar_sesion($email, $password) {
    $conn = conectar_bd();

    // Verificar credenciales
    $stmt = $conn->prepare("SELECT * FROM usuarios WHERE email = ?");
    $stmt->bind_param("s", $email);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        $usuario = $result->fetch_assoc();

        // Verificar contraseña y cuenta confirmada
        if (password_verify($password, $usuario['password']) && $usuario['confirmado'] == 1) {
            $_SESSION['usuario'] = $usuario['email'];
            echo json_encode(array("mensaje" => "Inicio de sesión exitoso."));
        } else {
            echo json_encode(array("error" => "Credenciales no válidas o cuenta no confirmada."));
        }
    }
}

```

```

    }
    } else {
        echo json_encode(array("error" => "Usuario no encontrado."));
    }
}

function cerrar_sesion() {
    // Cerrar sesión eliminando la información del usuario actual
    unset($_SESSION['usuario']);
    echo json_encode(array("mensaje" => "Sesión cerrada."));
}

function olvido_credenciales($email) {
    $conn = conectar_bd();

    // Verificar si el usuario existe y está confirmado
    $stmt = $conn->prepare("SELECT * FROM usuarios WHERE email = ?");
    $stmt->bind_param("s", $email);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        // Generar token de recuperación
        $token = generar_token();

        // Almacenar el token en la base de datos
        $stmt = $conn->prepare("UPDATE usuarios SET token_recuperacion = ? WHERE email = ?");
        $stmt->bind_param("ss", $token, $email);
        $stmt->execute();

        // Enviar correo de recuperación
        $mensaje = "Has solicitado restablecer tu contraseña. Para continuar, haz clic en el siguiente enlace:\n\n";
        $mensaje .= "http://tudominio.com/recuperar.php?email=$email&token=$token";
        enviar_correo($email, "Recuperación de Contraseña", $mensaje);

        echo json_encode(array("mensaje" => "Se ha enviado un correo con instrucciones para recuperar tu contraseña."));
    } else {
        echo json_encode(array("error" => "Usuario no encontrado o cuenta no confirmada."));
    }
}

function recuperar_credenciales($email, $token, $nueva_password) {
    $conn = conectar_bd();

    // Verificar si el token de recuperación coincide con el almacenado en la base de datos
    $stmt = $conn->prepare("SELECT * FROM usuarios WHERE email = ? AND token_recuperacion = ?");
    $stmt->bind_param("ss", $email, $token);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        // Cambiar la contraseña del usuario
        $hash_password = password_hash($nueva_password, PASSWORD_DEFAULT);
        $stmt = $conn->prepare("UPDATE usuarios SET password = ?, token_recuperacion = NULL WHERE email = ?");
        $stmt->bind_param("ss", $hash_password, $email);
        $stmt->execute();

        echo json_encode(array("mensaje" => "Contraseña recuperada con éxito."));
    } else {
        echo json_encode(array("error" => "Token de recuperación no válido."));
    }
}

function baja_del_sistema($email) {
    $conn = conectar_bd();

```

```
// Verificar si el usuario existe y está confirmado
$stmt = $conn->prepare("SELECT * FROM usuarios WHERE email = ?");
$stmt->bind_param("s", $email);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    // Eliminar al usuario de la base de datos
    $stmt = $conn->prepare("DELETE FROM usuarios WHERE email = ?");
    $stmt->bind_param("s", $email);
    $stmt->execute();

    // Cerrar sesión si el usuario eliminado es el actual
    if (isset($_SESSION['usuario']) && $_SESSION['usuario'] == $email) {
        cerrar_sesion();
    }

    echo json_encode(array("mensaje" => "Usuario eliminado con éxito."));
} else {
    echo json_encode(array("error" => "Usuario no encontrado o cuenta no confirmada."));
}

?>
```

Capítulo 4. Consolidación del código

Explicación:

Dado que Chat GPT nos ha generado un código base muy ventajoso, ahora solo tenemos que unificar las lógicas que hemos conseguido programar por separado. Podríamos intentar consolidar todo el código mediante prompts, pero es dudoso que se pueda conseguir.

Sin embargo, hemos venido a jugar, así que vamos a intentarlo.

Prompt 16. Es perfecto. Ahora, podrías consolidar este último snippet, con el de la base de datos que te he dicho que llamaríamos, a partir de ahí, database.sql?

Efectivamente, nos salió un poco por peteneras. Pero era de esperar, y los fallos nos sirven para ver hasta dónde puede llegar este hijo de perra, y hasta dónde no. De momento, claro.

Definitivamente, vamos a dejarlo aquí, y a dar por positivo este paso. En este punto, paro de escribir para consolidar el código, a mano. **He comenzado a las 23:00pm. Y ahora son las 01:30am. Es decir, he invertido 2:30 horas.** Y creo que he conseguido llegar lo más lejos posible, con el menor código posible, con el mejor código posible, en el mejor tiempo posible desde que llevo programando. Es la segunda-tercera vez que uso ChatGPT. Yo... no sé, estoy asombrado, ¿no? Por lo menos. Puedo hablar todo el tiempo que quiera de código, solo tengo que saber qué me interesa hablar.

Bueno. Solo queda trabajar un poco más sobre esto, y creo que tendremos algo construido en muy poco tiempo.

Mientras tanto, seguimos trabajando con ChatGPT para ir sacándonos códigos.

Prompt 17. Dado este diseño de base de datos: { aquí le inserto el sql final que he retocado } ¿podrías construir una consulta que, dando un token de sesión, devuelva el usuario, los grupos del usuario, los permisos del usuario y los permisos de los grupos del usuario?

```
-- Obtener información del usuario y sus grupos
SELECT
    u.nombre AS usuario,
    g.nombre AS grupo,
    p.nombre AS permiso,
    gp.nombre AS permiso_grupo
FROM
    sesiones s
JOIN
    usuarios u ON s.id_usuario = u.id
LEFT JOIN
    usuarios_y_grupos ug ON u.id = ug.id_usuario
LEFT JOIN
    grupos g ON ug.id_grupo = g.id
LEFT JOIN
    grupos_y_permisos gp ON g.id = gp.id_grupo
LEFT JOIN
    permisos p ON u.id = p.id
WHERE
    s.token = 'tu_token_de_sesion';
```

Mientras seguía con el siguiente capítulo, que he usado para minar ciertas tablas SQL pero eso lo explico en el siguiente capítulo, he hecho algunos prompts también para ir consolidando el código. Concretamente, le he estado (1) pidiendo snippets de esto, de aquello, para ahorrarme usar librerías externas. Uno para tener hooks, otro para peticiones, otro para los tests con mocha, luego los adaptaba, si hacía falta, con varios prompts más especificando los detalles de la implementación, y luego los he ido juntando todos en el fichero index.php. Esta es la idea de que sea minimal-oriented: no ocupar muchos ficheros.

Los prompts que he usado han sido los siguientes, por orden, y mezclados con los del siguiente capítulo:

Prompt 23. Puedes darme un ejemplo de cómo se usa mocha para nodejs [Se da por bueno]

Prompt 24. Puedes darme un ejemplo del uso de fetch en nodejs

Prompt 25. Puedes darme un ejemplo de cómo hacer una petición POST HTTP con node.js plano?

Prompt 26. Puedes wrappearlo en una función [Bien]

Prompt 27. Puedes hacer que funcione con promesas [Bien]

Prompt 28. Puedes hacer que la url y la ruta se envíen unidas, y que dentro de la función se separen parseando la url [Bien]

Prompt 29. Puedes hacer que se llame petición la función? [Bien]

Prompt 30. Puedes hacer que se exporte a módulo para node.js? [Bien, y cerramos el módulo de peticiones aquí]

Prompt 31. Puedes crear un sistema de hooks en PHP [Bien]

Prompt 32. Puedes hacer que tengan parámetros para ordenarlos por prioridad? [Bien]

Prompt 33. Puedes hacer una función de logging en PHP que apéndice un objeto en un fichero y que este fichero vaya rotando en función de si su peso excede cierto tamaño? [Bien, y aquí ya se podía cerrar, pero como no soy un experto, le pregunté luego...]

Prompt 34. Puedes actualizarlo para PHP 8.2? [Dice que está actualizado en la 8.1 pero que no debería cambiar nada]

Aquí, ya llevamos unas horas. Pero hemos conseguido más que nunca, a una velocidad nunca antes alcanzada. Bueno, más que nunca no, pero todo bastante idílico. Hemos invertido en hacer algunos cambios, y todavía no conocemos todo el código. Pero me parece que se puede confiar bastante, se ve un código muy conciso, sin rodeos, un código del que aprender.

Así que aunque llevemos horas, apenas hemos picado código, y tenemos un framework artesanal con las siguientes características:

- Orientado a funciones pero con orientación a objetos también. Muchas de las funciones se han definido en un prompt, como funciones globales. Luego, en otro prompt, nos ha devuelto una clase para gestionar los hooks, para los logs ha servido una función, pero la REST y el AUTH, que son el core fuerte de la aplicación, los ha plantado con funciones. Pues con funciones, de momento, no es una buena práctica andar mezclando quizá, pero sí bastante inevitable, y nos evita de ser puritanistas. Ahora somos chatgepeteistas. Le podemos pedir un snippet de cualquier lenguaje y en un momento nos lo explica si hace falta. Qué maravilla.

- Código conciso, útil, sin rodeos. Me encanta.

Estas son algunas de las características que nos hemos ahorrado prácticamente de picar, y sobre todo, sobre todo, de pensar, de imaginar, de abstraer, de crear:

- Un logger. Es una función, y nos hace muy eficientemente el trabajo. Y la hemos sacado en menos de un minuto.

- Un gestor de hooks. Predisponiendo ya el tema de plugins, que ya veremos cómo lo gestionamos, pero que está claro que entra dentro, porque a este ritmo, hay tiempo para todo.

- Un controlador que nos da el esquema de la base de datos. No una foto que se tomó, sino una consulta real del esquema de la base de datos.

- Un sistema de CRUD tipo REST que permite parámetros GET, POST y POST+JSON.

- Un sistema de autenticación y de autorización. Tenemos las tablas, tenemos los controladores. Tenemos una AUTH API, lo que nos queda es testearla, pero parece que gran parte ya nos lo hemos sacado.

Pues todo esto, conseguido en unas cuantas horas, sí, pero con menos de 500 líneas de código súper-al-grano, y que nos ha costado nada apenas.

Capítulo 5. Minar información de campo mediante tablas SQL

En este mundo caótico, no tenemos nada claro. Pero hay una cosa, que son los moldes de los datos parecidos. Esto es con lo que juega la informática, en general. Lo que ocurre con las bases de datos es que nos permiten manejar muchos registros parecidos de una forma óptima, y con algoritmos que no podríamos recrear tan fácilmente. Bueno, igual, con ChatGPT, pero otro día. De momento solo quería apuntar la mucha importancia que tienen los moldes de los datos. Y esto es lo mismo que decir lo mucho que importan las tablas SQL.

Pues una de las cosas más interesantes de las bases de datos es que... te interese lo que te interese, te interesará una a la larga. Porque puedes hacer una base de datos de cualquier cosa. Este hecho es un arma de doble filo: igual que te sirve para lo que son tus intereses, también te sirve para entrar en temas ajenos de primeras de otra forma. Pues poder extraer esta información, moldes abstractos de las cosas en ámbitos concretos de la vida, es un muy interesante uso de ChatGPT, y te permite tener una comprensión de las cosas más general normalmente.

Perfecto. Pues lo que he hecho es usarlo para extraer bases de datos de distintos temas que me interesaría soportar manejar.

Le he pedido varios prompts de la forma de: Escribeme un script en SQL que cree las tablas necesarias para { x }. Y en { x }, he aplicado: un blog, una escuela, una agencia de noticias, una organización asamblearia, etc. Y perfecto, me ha dado muchas tablas, luego las unificaré y pegaré, o quizá las deje ahí simplemente.

Los prompts concretamente son:

Prompt 18. Un script en SQL para un blog típico. Que soporte comentarios recursivos. [Se da por bueno]

Prompt 19. Puedes hacer un script SQL para crear una escuela en línea. [Se da por bueno]

Prompt 20. Puedes hacer un script SQL para crear una agencia de noticias en línea. [Se da por bueno]

Prompt 21. Puedes hacer un script para una organización asamblearia en línea. [Se da por bueno. Sin mentar SQL, lo hace bien]

Prompt 22. Puedes hacer un script SQL para la gestión de un hospital. [Se da por bueno]

Capítulo 6. Tests

Los tests son necesarios para irse tranquilos, y saber que si falla algo algún día, tenemos las migas de pan que nos dirán dónde ha fallado. Eso sería si hiciéramos bien los tests. Nosotros los vamos a hacer rápidos, que estamos on fire.

Para hacerlos más rápido, haremos una demostración del ciclo de vida de la autenticación de un usuario, desde el registro hasta la baja del sistema. Para todo eso, necesitaríamos un método que nos abrevie la petición, y otro método que nos abrevie la comprobación, y así reduciremos al mínimo nuestro código del test.

Como esto es una tarea relativamente pequeña, y además hay que andar por seguro, los tests los voy a picar a mano, no espero que ChatGPT me ayude a averiguar la interfaz HTTP que he preparado: yo la sé, yo la usaré directamente. Y así iremos más rápido, no se me ocurre un buen prompt para ello tampoco.

Como mi mejor herramienta es JavaScript, y Node.js es un aliado, haré los tests en Node.js + Mocha.

Sí que le hemos pedido a ChatGPT que nos haga un método para hacer las peticiones, que era un módulo a parte que parecía bastante claro de picar. En JavaScript. Es decir, para módulos aislados, ChatGPT puede ahorrarte mucha faena.

También le hemos pedido tareas más pequeñas todavía, como funciones polyfill, o utilitarias. Pero bueno, se nos intercalan las secciones, así que vamos a dejarlo como indeterminado en este documento, solo digamos que nos ha seguido siendo útil en el proceso. En siguientes ocasiones, sí que quizá nos centremos más directamente en definir los prompts. Seguramente podríamos haberlo utilizado para aproximarnos al script final, pero hemos ido bastante directos codificándolo sin asistencia.

Con esto, nos empieza a quedar una aplicación bastante respetable. Y sobre todo, sencilla, pero potente.

Ahora que ya tenemos los tests, y pasando, nos han salido 14, vamos a continuar con la siguiente tarea.

Capítulo 7. Encapsulamiento de espacio de nombres con POO

ChatGPT nos ha ayudado mucho a programar cosas que hubiéramos tardado más. Realmente, he tardado 2 días para hacer una tarea que de normal me llevaría más, no sé especificar cuánto, porque hay algoritmos que pueden ser un stopper de días. Con ChatGPT ha sido indoloro.

Sin embargo, el resultado son funciones sueltas por ahí, contaminando el espacio de nombres global. No es que esté mal, porque ya es un framework tan ambicioso, que se entiende que puede permitirse ocupar espacios de nombres globales, porque se convierte en el centro del servidor. Pero tampoco es necesario que se haga así. Y nosotros queremos hacerlo un poco mejor.

Es por esto que se decide encapsular la lógica de las funciones y variables que nos han ayudado a completar el script, en clases que limiten el alcance del espacio de nombres, así como permitimos variables locales. Es decir, no es por reusar la lógica realmente, es por no contaminar el espacio de nombres. Ya lo tenemos en funciones, y son funciones que se van a llamar 1 vez muchas de ellas. Sin embargo, tener un método "seleccionar" ocupando tal nombre en el espacio global, me incomoda un poco.

Nos ha llevado poco, y no hemos usado ChatGPT para ello, principalmente porque ya nos está quedando un script de 600 líneas. Podría probarlo, probar de decirle que me ayude él. Pero quiero empezar a quitarme esta tarea ya pronto, y con ChatGPT quizá damos demasiadas vueltas.

Capítulo 8. Soporte para hooks

ChatGPT nos ha ayudado con los hooks creando unas funciones que nos guardan y ejecutan hooks. De hecho, nos lo ha dado ya encapsulado, porque con la clase ya tiene un espacio de nombres para variables locales, y los hooks se deben guardar en una. Perfecto.

En tanto que hooks se refiere, nos queda esparcirlos por la aplicación, acordar la nomenclatura de los eventos de hook, y reservar un momento para la carga de plugins, que será parte de la siguiente tarea.

Los hooks deben estar bien hechos para que los plugins tengan un buen funcionamiento.

Los plugins son el gran target de estas dos tareas: hooks y plugins. Con los plugins podremos escalar la lógica de negocio de la aplicación sin interferir en el código del core, dejándolo intacto para otros escenarios.

Capítulo 9. Soporte para plugins

Una vez tenemos un sistema de hooks apto, podemos proceder al sistema de plugins. Que va a ser muy sencillo, nada de complicaciones de versiones y compatibilidades e instaladores y desinstaladores, nono. Simple. Simple al máximo.

La idea es la siguiente. La aplicación ya tiene un ciclo de vida de las peticiones, ya tiene un funcionamiento concreto. Los plugins se comportarían, en este sentido, como satélites pequeños (o grandes) que giran entorno al cuerpo grande, que es el ciclo de vida normal de las peticiones. Pero los plugins tienen el poder de alterar el ciclo de vida. ¿Cómo, si no se mezclan en el código? Pues mediante los hooks.

Así, un plugin puede sobrecargar un hook dado para interferir en el ciclo de vida concreto de ciertas peticiones, y alterar sus resultados u operaciones intermedias.

Los plugins son importantes para que los siguientes desarrollos no consistan en mirar al código fuente que ya nos hemos sacado de encima. Para que podamos empezar, desde 0, justo encima de donde lo dejamos, y empezar proyectos que no tengan nada que ver, partiendo de la misma base, y sin tener que alterarla.

Los plugins permiten que la aplicación pueda escalar de forma limpia. Y es la última parte de nuestra aplicación, porque los siguientes desarrollos, si lo hemos hecho bien, consistirán simplemente en bajarse el proyecto, y escribir una base de datos y un plugin con la lógica de negocio asociada.

Esto es, hoy por hoy, y dada mi experticia en el ámbito, es lo más avanzado que puedo dejar hecha una API REST de la que no sé nada. Lo siguiente es echar muchas horas haciendo fronts para acelerar este proceso o procesos que se derivan de él, pero el proceso de fabricación de APIs REST, con este método, queda relegado casi a tareas de usuario: solo tenemos que congrega el SQL para la base de datos (o usar una ya activa), por un lado, y por el otro picar el plugin en PHP para que se ocupe estrictamente de la lógica estrictamente de negocio.

10. Compartirlo en Github

Finalmente, si todo es correcto, y los tests parece que siguen pasándose, podemos subirlo a Github y comenzar las iteraciones para arreglar bugs o mejorar features de forma más controlada. Como hemos ido documentando el proceso, también nos hemos preocupado de dejar la documentación necesaria en el README del proyecto. Así que ahora no nos queda mucho más por explicar.

El proyecto es este:

<https://github.com/allnulled/tiki>

Esto es todo. El resto de la documentación la encontraréis en el proyecto. El código que ChatGPT ofreció sirvió para la base de este proyecto, y apenas tuve que picar código comparado con lo que tendría que haber picado. Tuve que picar, claro que sí, pero mucho menos, y con mucho menos riesgo. Y el proceso ha podido acelerarse mucho. Y el resultado, si todo marcha bien, creo que será satisfactorio, porque yo tenía RESTs en Node.js. Pero en la realidad, no hay apenas servidores Node.js gratuitos. En cambio, de PHP estático sí los hay. Y muchos frameworks web están basados en PHP, no solo Wordpress.

Ya está. A partir de aquí, podemos picar el cliente rápido rápido, sin tener que estar pensando en el backend. Y es una solución que puede funcionar en escenarios de producción reales, usa tecnologías estándar capaces de muchas cosas: PHP y MySQL, el stack más sobado de la web. Pero al servicio de las interfaces HTTP REST API, sea lo que sea que signifique, ¿no?

Feliz codificación, pues.