

**Examen: TodoCamisetas Api**

**Asignatura**: T1-25-(51) - DESARROLLO BACKEND - IF201IINF

**Profesor**: RAUL BILBAO BARRENECHEA

**Estudiante**: Cristopher Andrés Rivera Concha

**Fecha**: 13-06-2025

**2. Arquitectura del Proyecto**

**Estructura de carpetas y archivos**: Explica brevemente la función de cada carpeta clave (controllers/, models/, routes/, config/, etc.).

* **controllers/**

Contiene los controladores de la API. Cada archivo gestiona la lógica de negocio y las respuestas HTTP para una entidad (por ejemplo, CamisetaController.php, ClienteController.php). Los controladores reciben las solicitudes, validan los datos y llaman a los modelos.

* **models/**

Incluye los modelos de datos (Camiseta.php, Cliente.php, Talla.php). Cada modelo representa una tabla de la base de datos y contiene métodos para operaciones CRUD (crear, leer, actualizar, eliminar) usando consultas SQL preparadas y PDO.

* **routes/**

Define las rutas de la API y mapea las URLs y métodos HTTP a los controladores y métodos correspondientes. Aquí se usan expresiones regulares y $\_SERVER['REQUEST\_METHOD'] para el enrutamiento en PHP puro.

* **config/**

Archivos de configuración, como la conexión a la base de datos (db.php). Por ejemplo, config/db.php contiene los parámetros de conexión y crea la instancia PDO para toda la aplicación.

* **utils/**

Funciones utilitarias y clases de apoyo, como validadores de datos (Validator.php) y manejo de respuestas HTTP (Response.php).

* **index.php**

Punto de entrada principal de la API. Recibe todas las solicitudes y delega al enrutador.

* **swagger/**

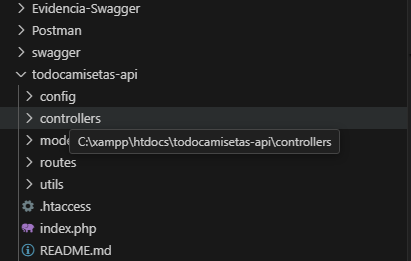
Contiene la documentación interactiva de la API generada con Swagger (OpenAPI).Incluye los archivos necesarios para visualizar y probar todos los endpoints de la API desde el navegador, permitiendo ver la estructura de los requests, responses, parámetros y ejemplos.

* **postman/**

Incluye la colección de pruebas automatizadas para Postman.Aquí se encuentra el archivo .postman\_collection.json que contiene todos los requests organizados para probar cada endpoint de la API, incluyendo casos de éxito, error, relaciones y limpieza de datos.

* **evidencia\_swagger/**

Contiene la evidencia de las pruebas realizadas con Swagger.Aquí se almacenan capturas de pantalla, archivos PDF o exportaciones que muestran los resultados de las pruebas ejecutadas desde la interfaz Swagger, organizados por endpoint.



La conexión se realiza en config/db.php usando PDO, lo que permite consultas seguras y preparadas, evitando inyección SQL.

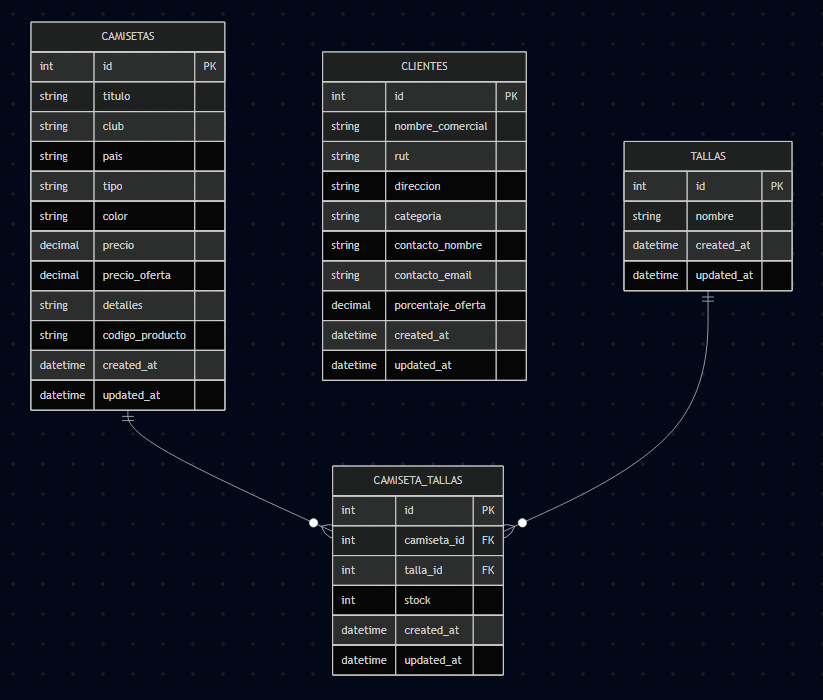
Separación lógica y modularidad

Los controladores gestionan la lógica de negocio y las respuestas HTTP.

Los modelos encapsulan el acceso a la base de datos y la lógica de datos.

El código es modular, facilitando el mantenimiento y la escalabilidad.

**3. Modelo de Datos**



**Descripción de las tablas y relaciones:**

* **camisetas:** Representa camisetas deportivas.
* **tallas:** Representa tallas disponibles.
* **clientes:** Representa clientes B2B.
* **camiseta\_tallas:** Tabla intermedia para la relación muchos a muchos entre camisetas y tallas, con campo de stock.

**Relaciones:**

* Una camiseta puede tener muchas tallas y viceversa (relación muchos a muchos a través de camiseta\_tallas).
* Los clientes no tienen relación directa en el modelo físico, pero se usan para lógica de negocio (por ejemplo, cálculo de precio final).

**4. Rutas HTTP y Enrutamiento**

**Enrutador en PHP puro**

El enrutamiento se implementa en index.php y/o en archivos de la carpeta routes/, usando expresiones regulares y $\_SERVER['REQUEST\_METHOD'] para determinar la ruta y el método HTTP, y así llamar al controlador y método correspondiente.

**Ejemplo de fragmento de código:**

|  |
| --- |
| **if ($\_SERVER['REQUEST\_METHOD'] === 'GET' && preg\_match('#^/api/camisetas$#', $uri)) {**  **CamisetaController::index();**  **}** |

**5. Operaciones CRUD**

* **Métodos implementados:**

Cada entidad (camisetas, clientes, tallas) tiene métodos all(), find(id), create(data), update(id, data), delete(id) en su modelo y métodos estáticos equivalentes en su controlador.

* **Validación de existencia y datos obligatorios:**

Antes de crear o actualizar, se validan los campos obligatorios (por ejemplo, título, club, precio para camisetas; nombre, RUT para clientes).Antes de eliminar, se valida que no existan dependencias (por ejemplo, no se puede eliminar un cliente con camisetas asociadas).

* **Relación camisetas-tallas:**

Se gestiona mediante la tabla intermedia camiseta\_tallas, permitiendo asignar múltiples tallas a una camiseta y viceversa.

**6. Lógica de negocio del precio final**

* **Cálculo de precio\_final:**
* Si el cliente es "Preferencial" y la camiseta tiene precio\_oferta, se usa ese precio.
* Si el cliente es "Regular" o no hay precio\_oferta, se usa el precio base.
* Además, se aplica el descuento correspondiente al cliente (porcentaje\_oferta).

**7. Control de errores y respuestas**

* **Errores y validaciones:**
* Si falta un campo obligatorio, se retorna un error 400 (Bad Request) con un mensaje JSON detallado.
* Si se intenta acceder a un recurso inexistente, se retorna 404 (Not Found).
* Si hay conflicto de integridad (por ejemplo, RUT duplicado), se retorna 409 (Conflict).
* Todas las respuestas usan el header Content-Type: application/json.
* **Validación de formato de entrada:**
* Se valida el formato de los datos (por ejemplo, email válido, RUT con formato correcto, valores numéricos positivos).