EXAMEN TRANSVERSAL FINAL

INFORME FINAL DEL PROYECTO BACKEND: TODOCAMISETAS

Asignatura:
DESARROLLO BACKEND
Alumnos:
Efren Tovar Silva Rut 25698445-8
Eduardo Ahumada Catalan Rut 17304258-2
Profesor:
Ivan Bilbao

INFORME FINAL DEL PROYECTO BACKEND: TodoCamisetas

```
todocamisetas-backend/
│
├── config/
│   └── database.php
│
├── controllers/
│   ├── CamisetaController.php
│   ├── ClienteController.php
│   └── TallaController.php
│
├── models/
│   ├── Camiseta.php
│   ├── Cliente.php
│   └── Talla.php
│
├── routes/
│   └── index.php
│
├── public/
│   └── index.php
│
├── swagger/
│   └── swagger.json
│
└── postman/
    └── TodoCamisetas.postman_collection.json
```

Información General

- **Nombre del Proyecto:** TodoCamisetas
- **Tecnología:** PHP Puro (sin frameworks)
- **Base de Datos:** MySQL (phpMyAdmin)
- **Entorno de desarrollo:** XAMPP, VS Code
- **Herramientas de apoyo:** Postman, Swagger UI
- **Objetivo:** Construir una API RESTful para la gestión de camisetas de fútbol, clientes y tallas, incluyendo lógica de descuentos según el tipo de cliente.

Arquitectura del Proyecto (MVC)

Se implementó una arquitectura basada en MVC:

- /config/database.php: Configuración de conexión PDO a MySQL.

- /models/: Contiene los modelos Camiseta.php, Cliente.php y Talla.php con métodos CRUD.
- /controllers/: Controladores que gestionan las peticiones HTTP (métodos: index, show, store, update, destroy).
- /routes/index.php: Sistema de enrutamiento mediante expresiones regulares que mapea las rutas a métodos de los controladores.
- /public/index.php: Punto de entrada principal. Define cabeceras HTTP, método y ruta solicitada.
- /swagger/swagger.json: Documentación de la API en formato OpenAPI.
- /postman/: Colección de pruebas para Postman.

Esta estructura permite cumplir con el principio de separación de responsabilidades y evidencia comprensión del rol de cada componente

Base de Datos y Relaciones

Se crearon las siguientes tablas:

- clientes: incluye porcentaje_oferta, categoria, etc.
- camisetas: incluye precio_oferta, codigo_producto, etc.
- tallas: tallas disponibles.
- camiseta_tallas: tabla intermedia (relación muchos a muchos).

Incluye claves foráneas, restricciones y validaciones. Representa un modelo de datos claro y funcional para soportar relaciones

Rutas de la API (RESTful)

Rutas implementadas con expresiones regulares en routes/index.php:

- GET /camisetas: Listar camisetas
- GET /camisetas/{id}?cliente_id={id}: Obtener una camiseta con precio final
- POST /camisetas: Crear camiseta
- PUT /camisetas/{id} / PATCH /camisetas/{id}: Actualizar camiseta
- DELETE /camisetas/{id}: Eliminar camiseta

Mismo patrón para clientes y tallas.

Cada ruta fue documentada en Swagger con:

- Método HTTP
- Ruta exacta y parámetros
- JSON de entrada/salida
- Códigos de respuesta y errores

Lógica de Negocio: Precio Final

La API calcula el precio_final según:

- Si existe precio_oferta, se toma como base.
- Se aplica porcentaje_oferta del cliente.
- Se retorna el campo precio_final.

Ejemplo:

GET /camisetas/1?cliente_id=3 → "precio_final": 47.99

Esto evidencia el diseño de reglas de negocio claro y reutilizable (Tarea 6).

Documentación Swagger

Se incluyó el archivo swagger.json y se desplegó usando Swagger UI localmente.
Incluye:

- Todas las rutas documentadas
- Parámetros
- Ejemplos de request y response
- Errores posibles

Pruebas con Postman

Se generó y exportó una colección Postman completa:

- Pruebas de cada endpoint: GET, POST, PUT, PATCH, DELETE
- Datos realistas
- Verificación de lógica de negocio
- Respuestas en formato JSON UTF-8

Cada petición contiene ejemplos en cuerpo y headers adecuados (Content-Type: application/json).

Operaciones CRUD y Validaciones

Se implementó CRUD completo para camisetas, clientes y tallas:

- Validaciones: campos obligatorios, unicidad de RUT o código de producto.
- Restricción lógica: no se puede eliminar cliente con camisetas asociadas.
- Operación GET por cliente permite filtrar camisetas asociadas.

Cumplimiento de la Rúbrica

Requisito técnico-académico	Estado
Arquitectura MVC y separación de responsabilidades	Cumplido
CRUD completo y validaciones	Cumplido
Lógica de precio final y descuentos por cliente	Cumplido
Enrutamiento eficiente con PHP puro	Cumplido
Documentación Swagger clara y completa	Cumplido
Colección Postman lista para pruebas	Cumplido
Codificación UTF-8, JSON y manejo de errores	Cumplido
Rol de cada componente y uso del lenguaje backend	Cumplido

Conclusión

El proyecto TodoCamisetas cumple con los requerimientos académicos y técnicos establecidos en la evaluación. Se aplicaron principios del backend, se utilizó PHP de forma eficiente para construir una API RESTful segura, modular, validada y documentada. La solución está lista para integrarse con un frontend moderno y escalable.