Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Licenciatura en Sistemas - 2023

Diseño de aplicaciones 2

Obligatorio I (Evidencia del diseño y especificación de la API)

Agustín Fioritto	**Nro est. 238927**
Joaquín Calvo	**Nro est. 203832**
Chiara Sosa	**Nro est. 241699**

ÍNDICE

Introducción

La presente documentación describe el diseño de la API desarrollada para la gestión de nuestro sistema. Esta API ha sido diseñada siguiendo los principios de arquitectura REST (Representational State Transfer), lo que garantiza una estructura lógica, escalable y coherente para el acceso a los recursos del sistema.

Este enfoque REST asegura que los recursos sean el núcleo de nuestra API y que los verbos HTTP estándar sean los utilizados para operar sobre estos recursos.

Diseño de la API

Para el diseño de la APi nos basamos en los siguientes criterios y principios REST. Se adjuntan recortes de nuestro código para demostrar las decisiones implementadas.

- ## Buenas Prácticas

El diseño de la API sigue las mejores prácticas de diseño de API REST de la industria. Esto garantiza que la API sea coherente, escalable y fácil de mantener a medida que evoluciona con las necesidades del sistema. Se han evitado redirecciones (códigos de estado 3xx) ya que no son necesarias en el contexto de la API, lo que contribuye a la eficiencia en el tráfico de datos.

- ## Recursos Bien Definidos

Cada recurso se representa como una URL única y está asociado con un conjunto de operaciones permitidas (GET, POST, PUT, DELETE).

- ## Comunicación Sin Estado

Cada solicitud HTTP contiene toda la información necesaria para comprender y procesar la solicitud, lo que hace que la API sea independiente del estado del servidor.

- ## Verbos HTTP Apropiados

Se utilizan los verbos HTTP estándar para indicar la acción deseada en cada solicitud (GET para lecturas, POST para creaciones, PUT para actualizaciones, DELETE para eliminaciones.

Ejemplo de una solicitud POST, para crear un usuario.

```
33      [HttpPost]
        2 referencias
34      public IActionResult RegisterUser([FromBody] User user)
35      {
```

Ejemplo de una solicitud GET, para obtener el producto con cierto ID.

```
57      [HttpGet("{id}")]
        0 referencias
58      public IActionResult GetProductByID([FromRoute] int id)
59      {
```

Ejemplo de una solicitud PUT, para actualizar los datos de un usuario.

```
399     [HttpPut("UpdateUserProfile")]
        2 referencias
400     public IActionResult UpdateUserProfile([FromBody] User user)
401     {
```

Ejemplo de una solicitud DELETE, para eliminar un producto del carrito.

```
74      [HttpDelete("deleteProduct")]
        1 referencia
75      public IActionResult DeleteProductFromCart([FromBody] Product product)
76      {
```

- Respuestas Coherentes

Se utilizan códigos de estado HTTP adecuados en las respuestas para indicar el resultado de cada solicitud.

Algunos ejemplos:

```
return BadRequest($"Error al obtener el historial de compras: {ex.Message}");
```

```
return NotFound(new { Message = "There are no purchases" });
```

```
return Ok(purchases);
```

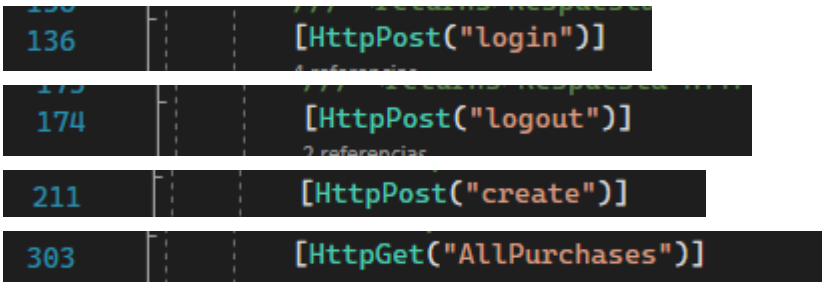
```
return CreatedAtAction(nameof(GetUserByID), new { id = createdUser.UserID }, createdUser);
```

- Convención de Rutas

Se ha seguido una convención de nomenclatura clara y uniforme para las rutas de acceso a los recursos. Por ejemplo, los usuarios se acceden a través de la ruta /api/users, lo que proporciona una organización lógica y predecible.

```
14      [Route("api/users")]
```

Las rutas de acceso a las distintas operaciones también respetan esta convención:



Mecanismos de autenticación

El mecanismo de autenticación de requests de la API se basa en la combinación de correo electrónico y contraseña. Los usuarios deben proporcionar sus credenciales válidas para acceder a las funcionalidades protegidas. Además, se ha implementado la verificación de roles para garantizar que solo los usuarios autorizados puedan realizar ciertas operaciones. Por ejemplo, las acciones propias de administración como ser la creación, eliminación y modificación de productos o la visualización de todas las compras realizadas están restringidas a usuarios con el rol "Administrador".

A continuación se muestra un claro ejemplo de una solicitud para eliminar un usuario, en la cual se corrobora la identidad del usuario para permitir eliminar un usuario únicamente si el rol es “Administrador” (línea 268). Si no se cumple con esta condición se retorna un mensaje de error indicando que el usuario no tiene permiso para eliminar usuarios.

```
259     [HttpDelete("{id}")]
        3 referencias
260     public IActionResult DeleteUser([FromRoute] int id)
261     {
262         try
263         {
264             Log.Information("Intentando eliminar el usuario con ID: {UserID}", id);
265
266             // Verifica si el usuario actual tiene permisos de administrador
267             var loggedInUser = _userService.GetLoggedInUser();
268             if (loggedInUser == null || loggedInUser.Role != "Administrador")
269             {
270                 return BadRequest("No tiene permiso para eliminar usuarios.");
271             }
272
273             // Llama al m◆todo DeleteUser del servicio para eliminar el usuario
274             _userService.DeleteUser(id);
275
276             Log.Information("Usuario eliminado exitosamente con ID: {UserID}", id);
277
278             return NoContent(); // Devuelve una respuesta HTTP 204 No Content
279         }
280         catch (UserException ex)
```

Códigos de Estado

La API utiliza códigos de estado HTTP estándar en sus respuestas para comunicar el resultado de las solicitudes. Estos códigos de estado han sido seleccionados

cuidadosamente para proporcionar información significativa sobre los errores o éxitos en las operaciones. Esto facilita la comprensión de errores en el lado del cliente y promueve una comunicación efectiva entre el servidor y los clientes.

Se decidió restringir la variedad el uso de los códigos de estado a 4 o 5 a modo de comprender entre los integrantes del equipo el uso concreto que le estamos dando a cada uno y evitar situaciones de confusión. Apuntamos a una comunicación clara y concisa. Estos son los códigos utilizados y sus significados:

- 200 OK: Operación exitosa.
- 201 Created: Recurso creado exitosamente.
- 400 Bad Request: Solicitud incorrecta o error en el procesamiento.
- 404 Not Found: Recurso no encontrado.

Recursos de la API

User

URL Base: /api/users

Resource: User

Description: Representa a un usuario registrado en el sistema.

Endpoints:

- POST /api/users

Registra un nuevo usuario en el sistema.
Parameters: User (Cuerpo de la solicitud): Datos del usuario a registrar.
Responses:
200 OK: Registro exitoso.
400 Bad Request: Error al registrar el usuario.

- GET /api/users

Obtiene la lista de todos los usuarios registrados en el sistema.
Responses:
200 OK: Lista de usuarios.
400 Bad Request: Error al obtener usuarios.
401 Unauthorized: No tiene permiso para obtener la lista de usuarios.

- GET /api/users/{id}

Obtiene un usuario por su ID.
Parameters: id (en la URL): ID del usuario.
Responses:
200 OK: Usuario encontrado.

400 Bad Request: Error al obtener el usuario.
404 Not Found: Usuario no encontrado.

- DELETE /api/users/{id}

Elimina un usuario por su ID.
Parameters: id (en la URL): ID del usuario.
Responses:
204 OK: Usuario eliminado.
400 Bad Request: Error al eliminar el usuario.
404 Not Found: Usuario no encontrado.

- POST /api/users/login

Inicia sesión de un usuario registrado.
Parameters: email (en el cuerpo de la solicitud): Correo electrónico del usuario, password (en el cuerpo de la solicitud): Contraseña del usuario.
Responses:
200 OK: Inicio de sesión exitoso.
400 Bad Request: Error al iniciar sesión.
401 Unauthorized: Autenticación fallida.

- POST /api/users/logout

Cierra sesión de un usuario registrado.
Parameters: user (en el cuerpo de la solicitud): Datos del usuario.
Responses:
204 Ok: Sesión cerrada exitosamente.
400 Bad Request: Error al cerrar sesión.

- GET /api/users/AllPurchases

Obtiene todas las compras realizadas en el sistema (solo para administradores).
Responses:
200 OK: Compras obtenidas exitosamente.
400 Bad Request: Error al obtener compras.
403 Forbidden: No tiene permiso para acceder a todas las compras (solo para usuarios no administradores).

- GET /api/users/GetPurchaseHistory/{id}

Obtiene el historial de compras de un usuario registrado por su ID.
Parameters: id (en la URL): ID del usuario.
Responses:
200 OK: Historial de compras obtenido exitosamente.
400 Bad Request: Error al obtener el historial de compras.
404 Not Found: Usuario no encontrado.

- PUT /api/users/UpdateUserProfile

Actualiza el perfil de un usuario registrado.
Parameters: user (en el cuerpo de la solicitud): Datos del usuario a actualizar.
Responses:
200 OK: Perfil del usuario actualizado exitosamente.

400 Bad Request: Error al actualizar el perfil del usuario.

- PUT /api/users/UpdateUserInformation

Actualiza la información de un usuario registrado (solo para administradores).
Parameters: user (en el cuerpo de la solicitud): Datos del usuario a actualizar.
Responses:
200 OK: Información del usuario actualizada exitosamente.
400 Bad Request: Error al actualizar la información del usuario.
401 Unauthorized: No tiene permiso para actualizar la información del usuario.

Cart

URL Base: /api/carts

Resource: Cart

Description: Representa el carrito del usuario registrado en el sistema.

Endpoints:

- POST /api/carts/addProduct

Agrega un producto al carrito del usuario logueado.
Parameters: product (en el cuerpo de la solicitud): Datos del producto a agregar.
Responses:
200 OK: Producto agregado exitosamente
400 Bad REquest: Error al agregar el producto

- DELETE /api/carts/deleteProduct

Elimina un producto del carrito del usuario logueado.
Parameters: product (en el cuerpo de la solicitud): Datos del producto a eliminar.
Responses:
200 OK: Producto eliminado exitosamente
400 Bad REquest: Error al eliminar el producto

- GET /api/carts

Obtiene el carrito del usuario logueado.
Parameters: -
Responses:
200 OK: Carrito obtenido
400 Bad REquest: Error al obtener el carrito

- GET /api/carts/AllProducts

Obtiene todos los productos del carrito del usuario logueado.
Parameters: -
Responses:
200 OK: productos obtenidos

400 Bad REquest: Error al obtener los productos del carrito

- GET /api/carts/PromotionApplied

Obtiene la promoción aplicada al carrito del usuario logueado.
Parameters: -
Responses:
200 OK: Promoción obtenida
400 Bad REquest: Error al obtener la promoción del carrito

- GET /api/carts/TotalPrice

Obtiene el precio total del carrito del usuario logueado.
Parameters: -
Responses:
200 OK: Precio obtenido
400 Bad REquest: Error al obtener el precio del carrito

Purchase

URL Base: /api/purchases

Resource: Purchase

Description: Representa un purchase realizado.

Endpoints:

- POST /api/purchases

Obtiene Crea un purchase
Parameters: cart (en el cuerpo de la solicitud): Datos del carrito para el purchase.
Responses:
200 OK: Compra satisfactoria
400 Bad REquest: Error de compra

- GET /api/purchases/specificPurchase/{id}

Obtiene Obtiene un purchase por id
Parameters: id (en el cuerpo de la solicitud): id del purchase
Responses:
200 OK:
404 Not Found: compra no encontrada

- GET /api/purchases/

Obtiene Obtiene todos los purchases
Parameters: -
Responses:
200 OK: retorna los purchases
404 Not Found: no hay purchases

- GET /api/purchases/usersPurchases/{id}

Obtiene Obtiene todos los purchases de un usuario
Parameters: id (en el cuerpo de la solicitud): id del usuario
Responses:
200 OK: retorna los purchases
404 Not Found: no hay purchases

Product

URL Base: /api/products

Resource: Product

Description: Representa un producto del sistema.

Endpoints:

- POST /api/products

Obtiene Crea un producto
Parameters: product (en el cuerpo de la solicitud)
Responses:
200 OK: Producto registrado correctamente
400 Bad Request: Error al registrar el usuario

- GET /api/products

Obtiene Obtiene la lista de los productos
Parameters: product (en el cuerpo de la solicitud)
Responses:
200 OK
400 Bad Request: Error al obtener productos

- GET /api/products{id}

Obtiene Obtiene un producto por id
Parameters: id (en el cuerpo de la solicitud)
Responses:
200 OK
400 Bad Request: Error al obtener producto

- DELETE /api/products

Obtiene Obtiene la lista de los productos
Parameters: id (en el cuerpo de la solicitud)
Responses:
200 OK: Producto eliminado exitosamente con ID
400 Bad Request: Error al eliminar producto

Configuración e Inyección de Dependencias

La clase Program en la aplicación de API web es fundamental para la configuración y la inyección de dependencias. En esta sección, se describen los principales componentes y servicios que se configuran y se inyectan en la aplicación.

Configuración de la Aplicación

- Se utiliza WebApplication. CreateBuilder(args) para crear la instancia de la aplicación web.
- Se configuran los servicios esenciales, como controladores y la configuración de sesiones.

```
namespace Obligatorio1.WebApi
{
    0 referencias | fiioro25, Hace 2 horas | 3 autores, 21 cambios
    public class Program
    {
        0 referencias | fiioro25, Hace 2 horas | 3 autores, 21 cambios
        public static void Main(string[] args)
        {
            var builder = WebApplication.CreateBuilder(args);

            // Add services to the container.
            builder.Services.AddControllers();

            // Configure sessions
            builder.Services.AddDistributedMemoryCache(); // Opcional, para almacenar en caché la sesión en memoria
            builder.Services.AddHttpContextAccessor();
            builder.Services.AddSession(options =>
            {
                options.IdleTimeout = TimeSpan.FromMinutes(30); // Establece el tiempo de inactividad de la sesión
                options.Cookie.HttpOnly = true;
                options.Cookie.IsEssential = true; // Si deseas que la sesión sea esencial para la aplicación
            });
```

Configuración de Sesiones

- Se configura la administración de sesiones para mantener la información del usuario durante el tiempo de inactividad.
- Se almacenan las sesiones en memoria y se establece un tiempo de inactividad de 30 minutos.

Swagger y Documentación

- Se configura Swagger para generar documentación interactiva de la API.
- Se incluyen comentarios XML en el código fuente para proporcionar descripciones detalladas en Swagger.

```
            builder.Services.AddEndpointsApiExplorer();
            builder.Services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "Obligatorio 1", Version = "v1" });
                var xmlFile = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
                var xmlPath = Path.Combine(AppContext.BaseDirectory, xmlFile);
                c.IncludeXmlComments(xmlPath); // Esto incluirá el archivo XML de documentación en Swagger
            });
```

Inyección de Dependencias

Se realizan inyecciones de dependencias para los siguientes servicios:

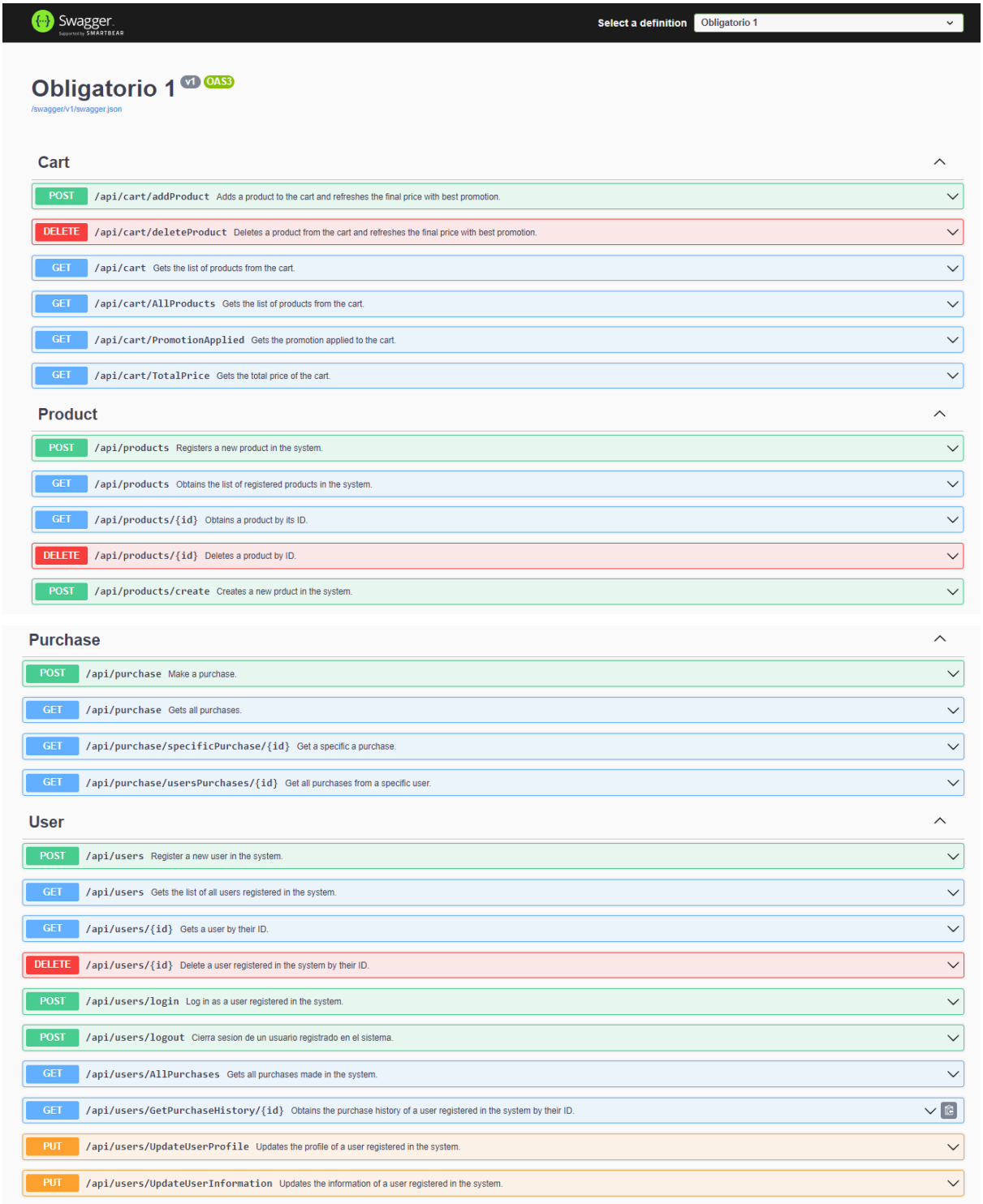
- IUserService para la gestión de usuarios.
- IProductService para la gestión de productos.
- IUserManagment para la administración de usuarios.
- IPromoManagerManagment para la administración de promociones.
- ICartService para la gestión de carritos de compra.
- IProductManagment para la administración de productos.
- DbContext para el acceso a la base de datos.
- Repositorios genéricos para las entidades User, Product, Purchase, y Cart.

```
builder.Services.AddScoped<Obligatorio1.IBusinessLogic.IUserService, Obligatorio1.BusinessLogic.UserService>();

builder.Services.AddScoped<Obligatorio1.IBusinessLogic.IProductService, Obligatorio1.BusinessLogic.ProductService>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.IUserManagment, UserManagment>();

builder.Services.AddScoped<Obligatorio1.IDataAccess.IPromoManagerManagment, PromoManagerManagment>();
builder.Services.AddScoped<Obligatorio1.IBusinessLogic.ICartService, Obligatorio1.BusinessLogic.CartService>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.IProductManagment,ProductManagment>();
builder.Services.AddDbContext<Context>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.IGenericRepository<User>, Obligatorio1.DataAccess.Repositories.GenericRepository<User>>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.IGenericRepository<Product>, Obligatorio1.DataAccess.Repositories.GenericRepository<Product>>();


builder.Services.AddScoped<Obligatorio1.IDataAccess.IUserManagment, Obligatorio1.DataAccess.Repositories.UserManagment>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.IPromoManagerManagment, Obligatorio1.DataAccess.Repositories.PromoManagerManagment>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.IPurchaseManagment, Obligatorio1.DataAccess.Repositories.PurchaseManagment>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.ICartManagment, Obligatorio1.DataAccess.CartManagment>();
builder.Services.AddScoped<Obligatorio1.Domain.User>();
builder.Services.AddScoped<Obligatorio1.IBusinessLogic.IPurchaseService, Obligatorio1.BusinessLogic.PurchaseService>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.IGenericRepository<Purchase>, Obligatorio1.DataAccess.Repositories.GenericRepository<Purchase>>();
builder.Services.AddScoped<Obligatorio1.IDataAccess.IGenericRepository<Cart>, Obligatorio1.DataAccess.Repositories.GenericRepository<Cart>>();
```

En conclusión, la clase Program se encarga de configurar y gestionar servicios esenciales para la API web, como sesiones, Swagger y la inyección de dependencias de servicios relacionados con la gestión de usuarios, productos y compras. Esta configuración sólida y las inyecciones de dependencias permiten que la API funcione de manera eficiente y escalable.

Especificación de la API con Swagger

Se utilizó la herramienta Swagger para la visualización electrónica de los endpoints generados:



Conclusión

La API ha sido diseñada con atención a los detalles, siguiendo las mejores prácticas de diseño de APIs RESTful. Este diseño proporciona una base sólida para el desarrollo y la expansión futura del sistema, garantizando la seguridad y eficiencia en la comunicación

entre clientes y servidores. Seguiremos mejorando la API en la iteración siguiente cuando se implemente toda la parte del FrontEnd.