

# **DESARROLLO DE APLICACIONES WEB**

## **Proyecto Final**

**NIDER-ZAPAS**

**Daniel Andrés Pérez**

## ÍNDICE

- [Introducción](#)
- [Objetivos](#)
- [Base de datos](#)
- [Back-end](#)
- [Front-end](#)
- [Mejoras](#)
- [Conclusiones](#)

## INTRODUCCIÓN

Se pretende crear una web e-commerce centrada en zapatillas y ropa deportiva usando Java en el back-end y NET Core con Razor en el front-end.

Cómo sistema de persistencia se usará Hibernate sobre MSSQL Server.

## OBJETIVOS

- Crear una base de datos que soporte la lógica del negocio.
- Levantar un servicio RESTful para gestionar la conexión con la BD.
- Levantar una web con NET Core capaz de consumir el servicio, se usará JQuery para hacerla más llamativa.

En su composición, la web debe permitir:

- Registro en nuestra BD
- Login, generando token para la autenticación.
- Muestra y paginado de productos.
- Añadir y quitar productos del carro.
- Registrar una dirección de entrega
- Finalizar la compra y mandar correo al cliente.



En cuanto a protección:

- La página debe negar las llamadas a finalizar la compra si el usuario no está logueado o no tiene un token válido.



# BASE DE DATOS


MSSQL SERVER

Nuper-zapas

Categorias	
	CAT_COD_1
	CAT_COD_2
	NOMBRE










Valoraciones	
	ID_VALORACION
	PRODUCTO_SKU
	TEXTO
	PUNTUACION



Pedidos	
	ID_PEDIDO
	PRODUCTO_SKU
	UNIDADES
	PRECIO_UNITARIO
	PRECIO_TOTAL

Productos	
	SKU
	NOMBRE
	DESCRIPCION
	MARCA
	CATEGORIA
	CATEGORIA_PADRE
	TALLAS
	PRECIO
	STOCK
	ES_OFERTA
	ES_REBAJAS
	IMG_SMALL
	IMG_BIG

Carros	
	ID_CARRO
	CLIENTE_EMAIL
	ID_PEDIDO
	PRODUCTO_SKU

Clientes	
	NOMBRE
	APELLIDOS
	USUARIO
	CLAVE
	EMAIL

Direcciones	
	CLIENTE_EMAIL
	CP
	PAIS
	LOCALIDAD
	CIUDAD
	CALLE
	NUMERO
	BLOQUE
	PISO
	PUERTA
	OBSERVACIONES

Pedidos_Total	
	ID_PEDIDO
	CLIENTE_EMAIL
	TOTAL_PEDIDO

# Java 8

## API RESTFUL

Nuper-zapas

## RESUMEN

- Usa Hibernate como ORM contra MSSQL Server.
- Usa Jersey library sobre JAX-RS (API para RESTful).
- Gson para serializar/deserializar JSON.
- JWT para generar/verificar los tokens.
- Javax Mail para mandar los correos.
- Todas las respuestas devuelven Type="application/json"
- Usa threads para mandar los correos de manera asíncrona.
- Se utiliza JWT para verificar al usuario.

Cuenta con una etiqueta personalizada de JAX-RS para tratar los tokens. Esta etiqueta podemos añadirla a cualquier método y así se obliga a verificar el token antes de entrar al método.

El token se genera en `/login`.

El método `/comprarCarro` es el único que precisa de verificación del token y es obligatorio estar logueado antes de finalizar la compra.

- Se aplican POJOs para mover la información de la manera más conveniente.
- La clase Operaciones soporta la lógica de las llamadas.

**IDE:** Eclipse 2018-12

## MÉTODOS

DESCRIPCIÓN: Listado de TODOS los productos

TIPO: GET

PATH: `/getProductos`

OUTPUT: `List<Productos>`

DESCRIPCIÓN: Listado de TODAS las categorías

TIPO: GET

PATH: `/getCategorias`

OUTPUT: `List<Categorias>`

DESCRIPCIÓN: Devuelve un producto buscando por su sku

TIPO: GET

PATH: `/getProducto/{sku}`

INPUT: `@PathParam("sku") String sku`

OUTPUT: `Productos`

DESCRIPCIÓN: Lista de productos buscando por categoría

TIPO: GET

PATH: `/getProductosCat/{cat_cod_1}/{cat_cod_2}`

INPUT: `@PathParam("cat_cod_1") String cat_cod_1,`

`@PathParam("cat_cod_2") String cat_cod_2`

OUTPUT: `Productos`



DESCRIPCIÓN: Lista de productos buscando por filtro

TIPO: GET

PATH: `/getProductosCat/{filtro}`

INPUT: `@PathParam("filtro") String filtro`

OUTPUT: `Productos`

DESCRIPCIÓN: Registro de un nuevo usuario

TIPO: POST

PATH: `/registro`

INPUT: `Clientes`

OUTPUT: `Response.status(OK)` si no existe usuario con ese email y si el registro va bien.

`Response.status(BAD_REQUEST)` si ya hay registro con ese usuario o falla hibernate al insertar.

DESCRIPCIÓN: Segundo registro de usuario para obtener dirección

TIPO: POST

PATH: `/segundoRegistro`

INPUT: `DireccionesPOJO`

OUTPUT: `Response.status(OK)` si es nueva dirección

`Response.status(BAD_REQUEST)` si ya existe

DESCRIPCIÓN: Login de usuario. Genera JWT.

TIPO: POST

PATH: `/login`

INPUT: `CredencialesPOJO`

OUTPUT: `Response.ok(gson.toJson(jwt))`

`Response.status(BAD_REQUEST)` si ya existe

DESCRIPCIÓN: Comprueba el token, almacena el carro y manda correo de compra.

TIPO: POST

PATH: `/comprarCarro`

INPUT: CarroTokenPOJO

OUTPUT: `Response.ok()`

`Response.status(BAD_REQUEST)` si no hay token o es inválido

# ASP.NET CORE FRONT END

Razo Pages C#

Nuper-zapas

## **RESUMEN**

- ASP Net Core 2.1
- Utiliza Razor Pages para tratar los modelos en las vistas.
- Bootstrap + CSS3.
- JQuery y JQueryUI para las animaciones.
- Ajax puntualmente para llamar a una acción sin renderizar su vista.
- LINQ para tratar listas y otros propósitos generales (no hacemos queries a la BD).

### **Nuggets:**

- NewtonsoftJSON
- X.PagedList

### **IDE:**

Microsoft Visual Studio Community

## ESTRUCTURA

Puesto que tenemos que atacar un RESTful externo conectado a la BD, no podemos usar Entity Framework.

Utilizamos la clase `APIRESTClient` para establecer conexión y procesar las llamadas.

### Modelos

Modelos básicos para intercambiar llamadas con el servicio.

`Productos`

`Pedidos`

`Clientes`

`Carros`

`Categorias & CategoriasId`

### Modelos Vistas

Estos modelos tratan los datos de los formularios.

`VistaLoginModelo`

`VistaRegistroModelo`

`VistaSegundoRegistroModelo`

Estos modelos los utilizamos para pasar múltiples modelos a una vista con vistas parciales en su interior.

`navbarPartialModel`

`navbarPartialModelCat`

Contiene los pedidos y el token

`CarroToken`

## Vistas

### Cliente

<code>RegistroUsuario</code>	formulario de registro principal
<code>SegundoRegistroUsuario</code>	formulario de registro de dirección; Dispara la compra

### Home

<code>VistaProducto</code>	muestra imagen grande y permite comprar
<code>VistaProductosCat</code>	paginación de productos por categoría
<code>VistaProductosFiltro</code>	paginación de productos por filtro

### Shared

<code>Index</code>	vista principal de la página, mucho JQuery
<code>Unauthorized</code>	en caso de que no se pueda validar el token
<code>Error</code>	en caso de error

### Partial

<code>_navbarPartial</code>	contenida en Index, VistaProductosCat y VistaProductoFiltro
<code>_miniCategorias</code>	contenida en navbarPartial
<code>_miniLogin</code>	contenida en navbarPartial
<code>_miniProducto</code>	contenida en VistaProductosCat y VistaProductoFiltro

## Controladores

### HomeController

Maneja el contenido básico de la página.

```
public async Task<IActionResult> Index(int? page)

public async Task<IActionResult> VistaProducto(string sku)

public async Task<IActionResult> VistaProductosCat(int catCod1, int catCod2,
int? page)

public async Task<IActionResult> VistaProductosFiltro(string filtro, int?
page)
```

### CienteController

Maneja la lógica de los clientes.

```
public IActionResult RegistroUsuario()

public async Task<IActionResult> RegistroUsuario(VistaRegistroModelo model)

public async Task<IActionResult> Login(VistaLoginModelo model)

public IActionResult Logout()

public IActionResult SegundoRegistroUsuario()

public async Task<IActionResult>
SegundoRegistroUsuario(VistaSegundoRegistroModelo model)
```

### PedidoController

Maneja la lógica de los pedidos del carro.

```
public async Task<IActionResult> AddProducto(string sku, int cantidad, string
returnUrl)

public void RemoveProducto(string sku)

public async Task<IActionResult> ComprarCarro()
```

## JQUERY & JQUERY UI

### **PaginationSlider:**

Vista: Index

Muestra TODOS los productos en un slider.

Utiliza JQUERYUI para las animaciones (explode en .hide y clip en .show).

### **Efecto burbujas:**

Vista: Index

Genera círculos a diferentes alturas, detrás del carousel, éstas ascienden a una velocidad determinada y pasado un tiempo vuelven a generarse.

### **Cambio color navbar**

Vista: \_navbarPartial

Cambia el color de fondo del navbar cada cierto tiempo, los colores se almacenan en un array y se va recorriendo.

### **Navbar “plegable”**

Vista: \_navbarPartial

Al hacer scroll hacia abajo, esconde el navbar superior, el inferior queda con poca opacidad, su posición es fixed y se muestran sus botones.

Inicia **Cambio color navbar**.

Al volver arriba, todo vuelve a estar cómo al inicio.



## JAVASCRIPT

### Lupa

Vista: VistaProducto

Crea un efecto de “lupa” sobre la imagen del producto.

## MEJORAS

- Usar Spring para generar una ruta estática con las imágenes.
- Correo de activación de la cuenta.
- Encriptar passwords.
- Manejar stock, tallas, oferta..
- Llamada para ver historial de pedidos.
- Cargar direcciones previas, países, cp...
- Mantener formulario relleno del segundo login en caso de tener que loguear o registrarse.
- No recargar al quitar productos del carro.
- Asegurar carga del script de la lupa.
- Cargar un fondo con la lupa.
- Validación formularios.
- Traducción

## CONCLUSIONES

Hacer este proyecto ha supuesto un gran reto. Se inició desde cero y se ha utilizado una combinación peculiar entre Java y NetCore.

Esto limita ciertos aspectos del desarrollo, por ejemplo, no podemos usar EntityFramework de NetCore dado que no hay conexión directa a la base de datos.

Por otro lado, nos obliga a buscar alternativas lo que conlleva una mejor comprensión de ciertos aspectos que de otra manera “damos por hecho”.

También me ha ayudado a tener una mejor visión general de éstos lenguajes y a descubrir nuevas herramientas como JWT, etiquetas personalizadas de JAX-RS..

Considero que éste proyecto abarca muchos de los conocimientos aprendidos durante el curso tanto en Java cómo en C# y otros totalmente nuevos.