

DAW 2

Desarrollo de Aplicaciones Web

Infinity XP

Reto 2a Evaluación

Integrantes

**Alejandro Rebollo
Markel Azpiazu
Bryan Murillos
Jon Zorzano
Julen Salinas**

2024-2025

Índice

| | |
|---|-----------|
| Introducción al reto | 2 |
| Planificación | 2 |
| DWES (Desarrollo Web en Entorno Servidor) | 2 |
| Modelos | 2 |
| Tablas normales de Flask y relación 1:N | 3 |
| Creación de tabla mediante relación N:M | 4 |
| Tabla de Users | 5 |
| Vistas | 6 |
| Inserción a la Base de Datos mediante JSON | 6 |
| Ruta para poder visualizar los videojuegos | 7 |
| Insertar datos al carrito | 8 |
| Vaciar Favoritos | 9 |
| Eliminar un artículo de favoritos | 10 |
| Visualizar el historial | 10 |
| Registro | 12 |
| Login | 13 |
| Proceso de compra | 15 |
| Interfaces web | 17 |
| Diseño de la página | 17 |
| Index | 17 |
| Artículos | 21 |
| Carrito | 23 |
| Favoritos | 24 |
| Login/Registro | 24 |
| Historial de Compras | 25 |
| Código fuente | 26 |
| JavaScript | 26 |
| Vue | 27 |
| Uso de Python en la página | 27 |
| Visualizar Cards | 29 |
| Visualizar datos de un sidebar | 30 |
| Visualizar el historial | 31 |
| Despliegue de aplicaciones web | 32 |
| Base de datos | 32 |
| Diagrama E/R | 32 |
| EIE (Empresa e Iniciativa Emprendedora) | 34 |
| ¿Cuánto cobraríamos nosotros por hacer este trabajo para una empresa? | 34 |
| Herramientas Utilizadas | 34 |
| Planes a Futuro | 35 |
| Bibliografía | 36 |

Introducción al reto

¿Quiénes somos en Infinity XP?

En Infinity XP somos una tienda online especializada en la venta de juegos digitales para todas las plataformas. Ofrecemos una experiencia de compra rápida, segura y conveniente, permitiéndote acceder a los mejores títulos de videojuegos al instante.

Nos enfocamos en tener una amplia selección de juegos para PC, Xbox, PlayStation, Nintendo y más. En Infinity XP, nuestra pasión es el mundo de los videojuegos y nos esforzamos para que tu experiencia de compra sea tan emocionante como jugar. ¡Explora, compra y empieza a jugar en minutos!

Planificación

Para el tema de planificación creamos un grupo de WhatsApp donde indicamos las normas básicas que tendríamos en el grupo para estas dos semanas de trabajo. A su vez, el primer día de reto en vez de empezar a trabajar ya con el proyecto decidimos reunirnos y realizar un plan donde analizamos las posibles herramientas que debíamos utilizar y por donde podíamos empezar el proyecto.

A su vez, nos reunimos todas las mañanas para analizar qué cosas estaban terminadas y qué cosas no estaban hechas y podíamos hacer ese día. Por último, guardamos todos los archivos en una carpeta en Drive.

DWES (Desarrollo Web en Entorno Servidor)

Modelos

En este archivo nos encontramos con las tablas diseñadas para la base de datos, sus respectivos atributos, y relaciones. Al estar utilizando *Flask*, tenemos que emplear el parámetro *db.Model* a cada clase/tabla para poder realizar la base de datos.

Tenemos 3 tipos de tablas en este archivo, empezando por la tabla normal antes mencionada de *Flask*, por otro lado, las tablas que se crean mediante las relaciones *N:M*, y la tabla de *Users* que tiene más añadidos. Vamos a empezar analizando 1 ejemplo de cada una de estas tablas:

Tablas normales de *Flask* y relación 1:N

En este ejemplo vemos nuestras tablas de Videojuegos y Categoría, con sus respectivos parámetros (teniendo una clave foránea debido a una relación), por otro lado, tenemos una variable de Python llamada categoría que nos sirve como puente entre esta y la otra tabla de Categoría, esta relación sería una 1:N

```
class Videojuegos(db.Model):

    id = db.Column(db.Integer, primary_key=True)
    titulo = db.Column(db.String(255))
    precio = db.Column(db.Integer)
    imagen = db.Column(db.String(255))

    id_categoria = db.Column(db.Integer,
db.ForeignKey('categoria.id'))
    categoria = db.relationship('Categoria',
backref=db.backref('videojuegos', lazy='dynamic'))

    def __init__(self, titulo, precio, imagen, categoria):
        self.titulo = titulo
        self.precio = precio
        self.imagen = imagen
        self.categoria = categoria

    def __repr__(self):
        return f'<Videojuego {self.id}>'

class Categoria(db.Model):
```

```

id = db.Column(db.Integer, primary_key=True)
nombre_categoria = db.Column(db.String(120))

def __init__(self, nombre_categoria):
    self.nombre_categoria = nombre_categoria

def __repr__():
    return f'<Categoria {self.id}>'

```

Creación de tabla mediante relación N:M

Teniendo en cuenta la relación entre la tabla de Videojuegos y la tabla de Compras, tenemos que crear una nueva tabla e introducir las claves primarias de ambas tablas en esta como claves foráneas, debido a que la tabla Videojuegos ya la hemos ejemplificado, a continuación observamos el código de la tabla Compras que nos falta y la nueva tabla que se forma:

```

class Compras(db.Model):

    id = db.Column(db.Integer, primary_key=True)

    id_usuario = db.Column(db.Integer, db.ForeignKey('user.id'))
    usuario = db.relationship('User', backref=db.backref('compras',
    lazy='dynamic'))

    videojuegos = db.relationship('Videojuegos',
    secondary=compras_videojuegos, backref='compra')

    def __init__(self, usuario):
        self.usuario = usuario

    def __repr__(self):

```

```

    return f'<Compras {self.id}>'

compras_videojuegos = db.Table(
    'compras_videojuegos',
    db.Column('id_compra', db.Integer, db.ForeignKey('compras.id')),
    db.Column('id_videojuego', db.Integer,
              db.ForeignKey('videojuegos.id')),
    db.Column('precio_compra', db.Integer)
)

```

También hay que mencionar que en la tabla de Compras tenemos un parámetro de *Python* llamado videojuegos que es indispensable para crear esta nueva tabla.

Tabla de Users

Esta tabla tiene algo diferencial en cuanto al resto de las clases que tenemos definidas en el código ; ya que es la encargada de gestionar los usuarios de la base de datos. Al igual que antes hemos mencionado, esta tabla tiene un parámetro de *Python* para poder crear una nueva tabla mediante una relación *N:M*.

Por otro lado, a la hora de crear la tabla en vez de únicamente utilizar el parámetro *db.Model* habrá que emplear el parámetro *UserMixin*; gracias a él podremos utilizar métodos como *check_password* para verificar una contraseña y otro tipo de funcionalidades útiles para el tema de control de usuarios.

```

class User(db.Model, UserMixin):

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100))
    correo_electronico = db.Column(db.String(150))

```

```

pwdhash = db.Column(db.String())
videojuegos = db.relationship('Videojuegos',
secondary=videojuegos_usuarios, backref='usuarios')

def __init__(self, username, correo_electronico, password):
    self.username = username
    self.correo_electronico = correo_electronico
    self.pwdhash = generate_password_hash(password)

def check_password(self, password):
    return check_password_hash(self.pwdhash, password)

```

Vistas

En este archivo nos encontramos una serie de rutas, métodos y renderizado para poder utilizar variables, datos de nuestra base de datos en nuestra página web haciendo más fácil trabajar y combinar tanto el *Back end* como el *Front end*.

Teniendo en cuenta de que tenemos muchas *vistas* en nuestro proyecto y algunas de estas tienen la misma funcionalidad, vamos a explicar en este documento únicamente las esenciales y diferentes.

Inserción a la Base de Datos mediante JSON

A continuación vamos a ver un ejemplo de cómo hemos insertado unos valores a partir de un *JSON* en nuestra base de datos, en este caso, introducimos los datos en la tabla de categorías.

```
@catalog.route('/insertarcategorias')
def insertar_categorias():
    with open('./mi_app/static/data/categorias. , 'r') as datos_cat:
        datos_categorias = json.load(datos_cat)

    for dato_cat in datos_categorias['categorias']:
        nueva_categoria = Categoria(
            nombre_categoria = dato_cat['nombre_categoria']
        )
        db.session.add(nueva_categoria)
        db.session.commit()

    return redirect(url_for('catalog.home'))
```

Gracias al `with open` podemos acceder al archivo y obtener los datos para insertarlos en dicha tabla.

Ruta para poder visualizar los videojuegos

Esta es una de las rutas principales de nuestro proyecto, en esta obtenemos datos de diversas tablas para poder utilizarlos en la página web; En esta ruta vemos que usamos una paginación de 30 *items* por página (todos los artículos en 1 sola).

```
@catalog.route('/articulos')
@catalog.route('/articulos/<int:page>')
@login_required
def videojuegos_paginados(page=1):
    videojuegos_python = Videojuegos.query.paginate(page=page,
per_page=30)
    detalle_db = Detalles.query.all()
    usuario_actual = current_user
    favoritos = usuario_actual.videojuegos

    precio_total=0
    for i in detalle_db:
```

```

    precio_total += i.videojuego.precio

    print(precio_total)

    return render_template('articulos.html',
videojuegos_html=videojuegos_python, detalle_html=detalle_db,
precio_total_html= precio_total, favoritos_html=favoritos)

```

Tras obtener los datos que nos interesan, a través del `render_template` pasamos las variables al *HTML*

Insertar datos al carrito

En esta ruta introducimos una serie de datos a la tabla de detalle, para ello, necesitamos dos parámetros que obtenemos en el *HTML*. En la ruta de *Python* los parámetros se introducen de la siguiente manera `<id_usuario>/<id_videojuego>`

```

@catalog.route('/insertar_datos_detalles/<id_usuario>/<id_videojuego>
')
@login_required
def metodo_insertar_carrito(id_usuario, id_videojuego):
    detalle_usuario = User.query.filter_by(id=id_usuario).first()
    detalle_videojuego =
Videojuegos.query.filter_by(id=id_videojuego).first()

    existe_detalle = Detalles.query.filter_by(id_usuario=id_usuario,
id_videojuego=id_videojuego).first()

    if existe_detalle:
        return redirect(url_for('catalog.home'))

    dato_detalle_final = Detalles(detalle_usuario,
detalle_videojuego)

```

```

db.session.add(dato_detalle_final)
db.session.commit()

return redirect(url_for('catalog.home'))

```

Gracias a las tablas Videojuegos y *User* obtenemos datos para poder hacer la consulta con la tabla de Detalles y gracias a eso podemos introducir los datos en dicha tabla.

Vaciar Favoritos

A continuación vamos a ver un ejemplo de como poder vaciar una tabla, en este caso la de favoritos o cómo nosotros le llamamos *videojuegos_usuarios*

```

@catalog.route('/vaciar_favoritos')
@login_required
def vaciar_fav():
    usuario_actual = current_user
    favoritos = usuario_actual.videojuegos

    for favorito in favoritos:
        usuario_actual.videojuegos.clear() # al ser un db.table y no
        un db.model tengo que borrarlo con el método clear

    db.session.commit()
    return redirect(url_for('catalog.home'))

```

Vemos aquí que hacemos una serie de consultas mediante los atributos *backref* para acceder a las distintas tablas, posteriormente y gracias a ser una clase *db.Table* en vez de *db.Model* podemos utilizar el método *.clear()* para vaciar los datos de dicho usuario.

Eliminar un artículo de favoritos

Siguiendo una estructura prácticamente similar a la anterior, gracias a esta vista podemos borrar únicamente un solo artículo de dicha tabla, cambiando solamente el método `.clear()` por el `.remove()`

```
@catalog.route('/vaciar_individual_fav')
@login_required
def vaciar_individual_favoritos():
    usuario_actual = current_user
    favoritos = usuario_actual.videojuegos

    for favorito in favoritos:
        usuario_actual.videojuegos.remove(favorito) # al ser un
db.table y no un db.model tengo que borrarlo con el método remove
para borrar solo ese

    db.session.commit()
    return redirect(url_for('catalog.home'))
```

Visualizar el historial

Para visualizar el historial hemos tenido que realizar una serie de procesos algo más complejos; por ejemplo, un diccionario para obtener de una forma adecuada los datos, ya que necesitamos datos de muchas tablas distintas para introducirlas en la tabla `compras_videojuegos` que es una N:M, en este caso vamos a necesitar la tabla de `User`, Compras y acceder a su vez a la tabla Videojuegos mediante el atributo de `Backref`

```
@catalog.route('/leer_historial')
@login_required
def metodo_mostrar_historial():
    usuario_actual = current_user

    compras = usuario_actual.compras.all()

    print(f"Compras: {compras}")

    detalles_historial_python = {}

    for compra in compras:
        print(f"Compra: {compra}")

        for videojuego in compra.videojuegos:
            obtener_dato =
db.session.query(compras_videojuegos).filter_by(id_compra=compra.id,
id_videojuego=videojuego.id).first()

            if obtener_dato:
                detalles_historial_python[videojuego.id] = {
                    'titulo': videojuego.titulo,
                    'precio': obtener_dato.precio_compra,
                    'imagen': videojuego.imagen
                }

    return render_template('historialcompras.html',
detalles_historial_html=detalles_historial_python)
```

Tras introducir esos datos al diccionario, podemos pasar esa variable al *HTML* para poder visualizar de manera correcta un diccionario con los datos de la base de datos y así poder iterar con este.

Registro

Gracias al *UserMixin* y al *flask_login* podemos tener un control de usuarios de una manera más sencilla y simple de configurar, con las rutas de registro, *login*, *logout*, formularios para las dos primeras de estas y una tabla para los usuarios, podemos realizar esto. Ahora vamos a explicar la ruta de registro y el formulario para poder insertar un nuevo usuario en la base de datos.

```
@catalog.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('catalog.home'))

    username = request.args.get('username')
    correo_electronico = request.args.get('correo_electronico')
    password = request.args.get('password')

    if (username and correo_electronico and password):
        existing_user =
User.query.filter_by(username=username).first()
        if existing_user:
            return render_template('login.html')
        user = User(username,correo_electronico, password)
        db.session.add(user)
        db.session.commit()
        return redirect(url_for('catalog.login'))
    return render_template('register.html')
```

Vemos que obtiene los datos del input de un formulario con los mismos nombres de los campos de la tabla usuarios, posteriormente si el usuario no existe en la base de datos, se insertará en dicha tabla con los valores introducidos en el formulario.

He aquí el archivo *HTML* del formulario de registro:

```
{% extends 'base.html' %}

{% block container %}
<div>
    <h1>Registro</h1>
    <form method="GET" action="{{ url_for('catalog.register') }}"
        role="form" class="formularioUsuario">
        <div>
            <label for="username">Nombre Usuario:</label>
            <input type="text" id="username" name="username" required>
        </div>

        <div>
            <label for="correo_electronico">Correo Electrónico:</label>
            <input type="email" id="correo_electronico"
                name="correo_electronico" required>
        </div>

        <div>
            <label for="password">Contraseña:</label>
            <input type="password" id="password" name="password" required>
        </div><br>
        <button type="submit" class="btn">Registrarme</button>
    </form>
</div>
{% endblock %}
```

Login

La estructura del login es muy similar a la del registro, con la diferencia de que no tenemos que insertar datos, únicamente comparar campos y abrir la sesión en caso de 'éxito'. Podemos observar que obtiene los datos del formulario como antes hemos mencionado y luego hace la comparación y mantiene la sesión abierta al cerrar la ventana.

```
@catalog.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('catalog.home'))

    username = request.args.get('username')
    correo_electronico = request.args.get('correo_electronico')
    password = request.args.get('password')
    existing_user = User.query.filter_by(username=username).first()

    if not (existing_user and
existing_user.check_password(password)):
        return render_template('login.html')

    login_user(existing_user, remember=True) # recuerda usuario al
cerrar la ventana
    return redirect(url_for('catalog.home'))
```

He aquí el formulario del *login*:

```
{% extends 'base.html' %}

{% block container %}
<div>
    <h1>Inicio de Sesión</h1>
    <form method="GET" action="{{ url_for('catalog.login') }}"
role="form" class="formularioUsuario">
        <div>
            <label for="username">Nombre Usuario:</label>
            <input type="text" id="username" name="username" required>
        </div>

        <div>
            <label for="correo_electronico">Correo Electrónico:</label>
            <input type="email" id="correo_electronico"
name="correo_electronico" required>
        </div>

        <div>
            <label for="password">Contraseña:</label>
```

```

<input type="password" id="password" name="password"
required>
</div><br>
<button type="submit" class="btn">Iniciar Sesión</button>
</form>
</div>
{ % endblock %}

```

Proceso de compra

Para el proceso de compra, hemos decidido utilizar *Stripe*, esta ruta realiza un proceso de compra ficticio dónde los videojuegos que están en el carrito se ‘compran’ y su correspondiente pago llega a nuestra compra de *Stripe*, obtenemos los datos de una variable que le pasamos como parámetro del método, aplicamos la moneda correspondiente y trás ese proceso insertamos los datos necesarios tanto a la tabla de Compras como a la tabla *compras_videojuegos*.

```

@catalog.route('/proceso_compra/<int:total>', methods=['GET'])
@login_required

def proceso_compra(total):

    try:

        intent = stripe.PaymentIntent.create(
            amount=total*100,
            currency="eur",
            payment_method_types=["card"],
        )

```

```
dato_compra = Compras(current_user)
db.session.add(dato_compra)
db.session.commit()

detalles_carrito =
Detalles.query.filter_by(id_usuario=current_user.id).all()

for detalle in detalles_carrito:

    videojuego_python = detalle.videojuego
    db.session.execute(
        compras_videojuegos.insert().values(
            id_compra=dato_compra.id,
            id_videojuego=videojuego_python.id,
            precio_compra=videojuego_python.precio
        )
    )
    db.session.commit()

return redirect(url_for('catalog.vaciar_carro'))

except Exception as e:

    print(f'Error al procesar el pago: {str(e)}')

return redirect(url_for('catalog.home'))
```

Interfaces web

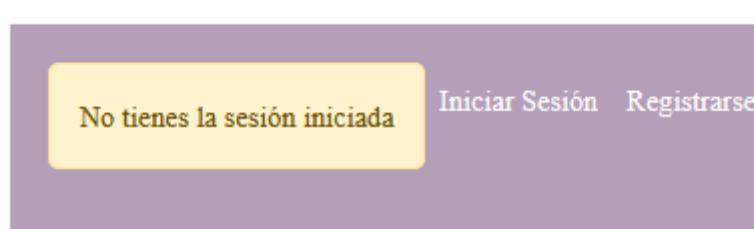
Diseño de la página

Index

Para la portada de la página de Infinity XP hemos creado una cabecera donde en la parte izquierda de la cabecera tenemos el logo de nuestra empresa seguido de un menú donde podrás navegar a través de nuestra página. El menú está compuesto con los enlaces de “Inicio”, “Artículos” e “Historial de compra”.

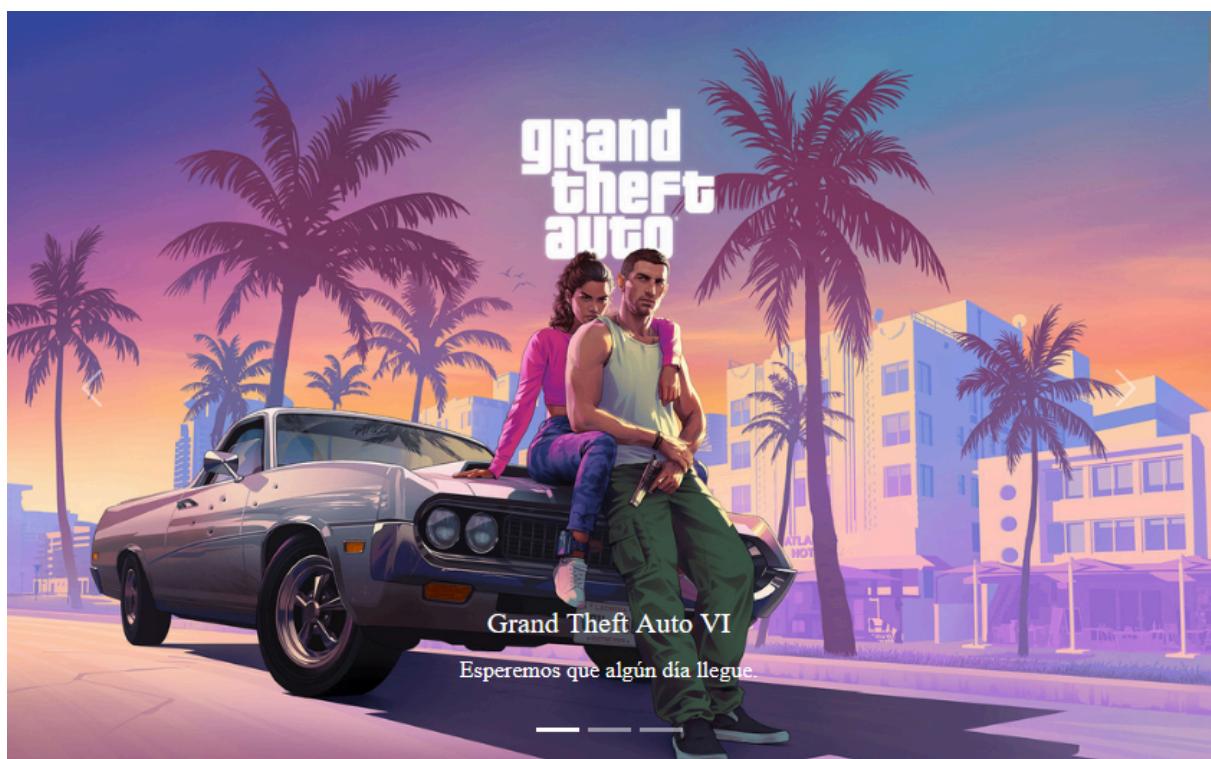


Por último, tenemos el inicio de sesión en la parte derecha de la cabecera, una vez iniciada la sesión se mostrará el nombre del usuario y su correo electrónico. En las siguientes imágenes se aprecia un ejemplo con el inicio de sesión.





Posteriormente, tenemos mediante bootstrap tres imágenes de videojuegos más conocidos de nuestra empresa donde podrás deslizar para ver la imagen. En la siguiente imagen se puede ver un ejemplo.





Además de lo antes mencionado, tenemos varios planes de suscripción para otorgar ciertas personas respecto a otros usuarios, este diseño está realizado con *Bootstrap 5* y su código fuente está en la bibliografía, he aquí una imagen de dichos planes de suscripción:

Planes de Pricing

Adquiere ventajas adicionales sobre otros usuarios con cualquiera de los siguientes planes.

| Plan Básico | Plan Avanzado | Plan Pro |
|--|---|--|
| € 10.90 /Al mes | \$ 24.90 /Al mes | \$ 49.90 /Al mes |
| Más caracteres por comentario Mayor cantidad comentarios Soporte 24/7 Información adicional | Más cantidad de comentarios que el plan básico Backups Crear múltiples usuarios Más caracteres por comentario Mayor cantidad comentarios Soporte 24/7 Información adicional | Mayor velocidad de la página Opción de personalizar la página Cambiar entre tema claro/oscuro Más cantidad de comentarios que el plan avanzado Backups Crear múltiples usuarios Más caracteres por comentario Mayor cantidad comentarios Soporte 24/7 Información adicional |
| ADQUIRIR AHORA | ADQUIRIR AHORA | ADQUIRIR AHORA |

A su vez, tenemos una sección de comentarios donde cualquier persona con la sesión iniciada puede ver los comentarios del resto de usuarios e incluso poner los suyos propios:

Comentarios

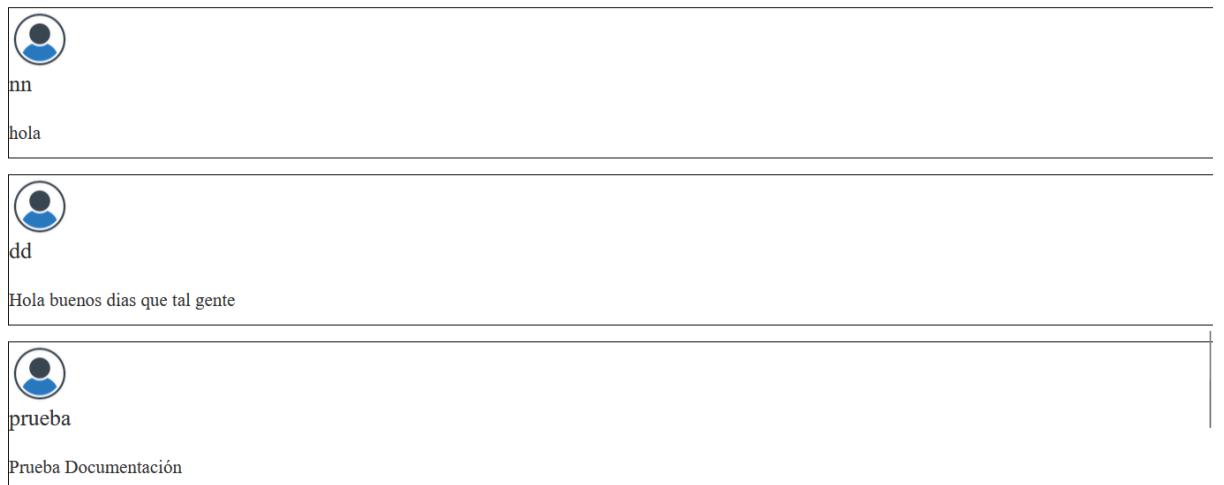
Nuevo Comentario

Compartir el comentario

Comentarios

Nuevo Comentario

Compartir el comentario



Para finalizar, tenemos un pequeño texto explicando nuestra idea de negocio para el usuario y en la parte inferior tenemos un *Footer* donde se muestra una pequeña información extra sobre nuestra empresa.

Artículos

El apartado de artículos lo hemos hecho de la siguiente manera.

Hemos utilizado bootstrap para tener cada artículo en diferentes card, hemos conseguido que lea de la base de datos el título, la imagen, el precio y la categoría de cada videojuego, en la parte baja de cada card tenemos 2 botones, el botón de añadir al carrito y el botón de favoritos, como podemos ver en la imagen de abajo.

Hemos añadido un buscador de videojuegos para facilitar al usuario su búsqueda.

The screenshot displays a user interface for a video game catalog. At the top left is the XP INFINITY logo. Navigation links include 'Inicio', 'Artículos', and 'Historial de compras'. On the right, a user profile box shows a placeholder profile picture, the greeting 'Hola a!', the email 'Tu correo es: a@gmail.com!!', and a 'Cerrar Sesión' button. Below this is a large image of a character from 'Watch Dogs 2' standing on a rooftop overlooking a city. To the left of the main content area is a sidebar titled 'Catálogo de Videojuegos' with buttons for 'Ver Carrito', 'Ver Favoritos', and a search bar labeled 'Buscar Videojuegos'. Below these are five game thumbnails: 'Grand Theft Auto V', 'Grand Theft Auto V', 'WATCH_DOGS 2' (highlighted in blue), 'ASSASSIN'S CREED SHADOWS', and 'PALWORLD'. To the right of the game thumbnails is a detailed product card for 'WATCH_DOGS 2'. It features the game's cover art, the title 'Watch Dogs 2', the price '20 €', and the genre 'Mundo Abierto'. At the bottom right of the card are two buttons: 'Añadir al carrito' and 'Favoritos'.



Carrito

Al darle a añadir a un artículo al carrito, se meterá en un sidebar que se despliega a la izquierda, y ahí aparecerán los videojuegos, el título del juego, el precio y un botón para poder eliminarlos, abajo del todo sale cuento es el precio total y un botón para finalizar la compra y debajo de este otro botón para vaciar todo el carrito. Todo lo que se compra en el carrito llega a nuestra cuenta de *Stripe*

Transacciones

+ Crear pago Análizar

| Todos 13 | Exitosos 0 | Reembolsados 0 | Disputados 0 | Error 0 | No capturados 0 |
|---|----------------|-----------------------------|-----------------|------------|--------------------|
| <input type="button" value="Fecha y hora"/> <input type="button" value="Importe"/> <input type="button" value="Divisa"/> <input type="button" value="Estado"/> <input type="button" value="Método de pago"/> <input type="button" value="Más filtros"/> <input type="button" value="Exportar"/> <input type="button" value="Editar columnas"/> | | | | | |
| <input type="checkbox"/> Importe | Método de pago | Descripción | Cliente | Fecha | Fecha de reembolso |
| 25,00 € EUR Incomplete | — | pi_3QpoKwIwJEPIdHzX1eRaaLQq | 7 feb. 10:59 | — | — |
| 133,00 € EUR Incomplete | — | pi_3QpoJRIwJEPIdHzX2UJegRLW | 7 feb. 10:57 | — | — |
| 118,00 € EUR Incomplete | — | pi_3QpnfbIwJEPIdHzX0TgKl0ow | 7 feb. 10:16 | — | — |
| 60,00 € EUR Incomplete | — | pi_3QpnUIIwJEPIdHzX21SvtRrR | 7 feb. 10:04 | — | — |
| 90,00 € EUR Incomplete | — | pi_3QpU5qIwJEPIdHzX1Ea2pbKq | 6 feb. 13:22 | — | — |
| 90,00 € EUR Incomplete | — | pi_3QpSeVIwJEPIdHzX0EzGdXRO | 6 feb. 11:49 | — | — |
| 206,00 € EUR Incomplete | — | pi_3QnRFnTw1FPtDHzX154oh17k | 6 feb. 10:17 | — | — |

Favoritos

En favoritos al añadir un artículo se meterá en el sidebar que se despliega a la izquierda, ahí aparecerán los videojuegos con un botón para eliminarlos y abajo del todo está otro botón para vaciar todo todos los artículos metidos en favoritos.

Login/Registro

El apartado de “Login” y “Registrarse” se divide en dos partes diferentes.

El apartado de “Login” o “Iniciar sesión” tiene una estructura rectangular, en la que se compone de tres campos diferentes y un botón al final para ejecutar los campos.

Inicio de Sesión

Nombre Usuario:

Correo Electrónico:

Contraseña:

Iniciar Sesión

Los tres campos se dividen en: “Username”, “Correo Electrónico” y “Password”. Estos datos son importantes para poder buscar en la base de datos los parámetros que

coincidan con el usuario, y de esa manera poder ejecutar correctamente el botón de “Iniciar sesión” y poder navegar por la página con la sesión iniciada.

Por otro lado, está el apartado de “Registrarse”, que se utiliza para poder crear una cuenta en la base de datos. Esta parte de la página es parecida a la anterior, lo único que cambia es el botón de abajo, que pone, “Registrarse” y en el título pone “Registrarse”. Gracias a esta función, en la base de datos se guardan todos los datos de las personas nuevas que se registran en la web, y de esa manera luego con iniciar sesión ya puedes navegar en la página con un usuario propio.

Registro

Nombre Usuario:

Correo Electrónico:

Contraseña:

Registrarme

Historial de Compras

Tus Compras

Historial de Compras

| | |
|---|---|
|  | Grand Theft Auto Vice City Precio: 15 € |
|  | Skyrim Precio: 40 € |
|  | No Mans Sky Precio: 58 € |

Código fuente

En nuestro proyecto, tenemos mucho código fuente en cuanto al diseño y su funcionalidad, ya que utilizamos diversas herramientas y/o tecnologías como *Bootstrap 5*, *JavaScript*, *Vue*, bucles de *Python*... Es por esto que nos vamos a centrar por así decirlo en lo complejo o diferente como *JavaScript*, *Vue* y la parte de *Python*.

JavaScript

El uso de JavaScript puro en nuestro proyecto se ha enfocado en el tema de la barra de búsqueda, cuyo código hemos encontrado en un vídeo de Youtube (enlace en la bibliografía); y lo hemos modificado para que cuadre con nuestras clases de etiqueta. A continuación vamos a ir mostrando los bloques de código para darle funcionalidad a dicha barra de navegación.

El código HTML de la barra es el siguiente:

```
<div class="search-wrapper">
    <label for="search">Buscar Videojuego</label>
    <input type="search" id="search" data-search>
</div>
```

Además de esto, las clases que utilizamos en el *script* se encuentran en un card dentro de un bucle de *Python* (cuyo código explicaremos en su apartado correspondiente). Por último tenemos aquí el código que le da la funcionalidad a la barra de búsqueda.

```
const userCardTemplate =
document.querySelector("[data-user-template]");
const userCardContainer = document.querySelector(".containerbucle");
const searchInput = document.querySelector("[data-search]");

let users = [];

searchInput.addEventListener("input", e => {
    const value = e.target.value.toLowerCase();
    users.forEach(user => {
```

```

        const isVisible = user.Nombre.toLowerCase().includes(value);
        user.element.classList.toggle("hide", !isVisible);
    }) ;
}) ;

userCardContainer.querySelectorAll(".card").forEach(card => {
    const header = card.querySelector(".card-title").innerText;

    const body = card.querySelector(".card-text").innerText;

    users.push({
        Nombre: header,
        Precio: body,
        element: card
    }) ;
}) ;
}

```

La primera parte del código como antes hemos mencionado la hemos sacado de un vídeo (únicamente hemos modificado las clases). En cambio, la segunda la hemos programado nosotros mismos, quitando un *fetch* que venía por defecto. En cuanto a este bloque, utilizamos un *foreach()* para iterar por cada elemento de clase *Card* y obtener mediante *.innertext* la información que hay en el *body* y *header* del *Card* para finalmente introducirlo en un array llamado *users* (es el nombre del array que se utiliza en el vídeo)

Vue

El uso de *Vue* en *Infinity XP* se centra únicamente en los diseños de los *sidebar* (ya que su funcionalidad se hace con *Python*). Este tema no lo diseñamos ni lo programamos nosotros ya que los ejemplos de estos los cogimos de Internet para un trabajo en *Desarrollo de Entorno Cliente*. Lo único que vemos conveniente mencionar es que utiliza funciones de *JavaScript* para abrir y cerrar el *sidebar*.

Uso de *Python* en la página

El uso de la sintaxis y funcionalidades de *Python* en los archivos *HTML* ha sido indispensable para visualizar los datos, y aplicar condicionales sobre todo para el tema del *login*. Por ejemplo, el *navbar* cambia según si el usuario está con la sesión iniciada o no. La estructura de este ejemplo es algo así :

```
{% if current_user.is_authenticated %}

<div id="sesioniniciada">
    <h2>Tu perfil de usuario</h2>
    <img src ="../static/img/usuario.png" width="100px">
    <h3>Hola {{ current_user.username }}!!</h3> <h3>Tu correo
es: {{ current_user.correo_electronico }}!!!</h3>
    <a id="enlacecerrar" href="{{ url_for('catalog.logout') }}>Cerrar Sesión</a>
</div>
</div>

{% else %}

<ul class="navbar-nav me-auto mb-2 mb-lg-0">
    <li class="nav-item">
        <div class="alert alert-warning" role="alert">
            No tienes la sesión iniciada
        </div>
    </li>

    <li class="nav-item">
        <a class="nav-link active" aria-current="page" href="{{ url_for('catalog.login') }}>Iniciar Sesión</a>
    </li>
</ul>
```

```

<li class="nav-item">
    <a class="nav-link" href="{{ url_for('catalog.register') }}>Registrarse</a>
</li>
</ul>

```

Vamos ahora a ver varios ejemplos de visualización de datos en las páginas.

Visualizar Cards

Para visualizar los cards hemos utilizando un bucle para poder obtener los datos de una variable renderizada y visualizar los campos de la tabla que nos interesa mediante su atributo de la tabla.

```

<div class="containerbucle" data-user-cards-container>
    {% for videojuego in videojuegos_html.items %}

        <div class="card" style="width: 18rem;" :key="item.id">
            
            <div class="card-body">
                <h5 class="card-title" data-header>{{ videojuego.titulo }}</h5>
                <p class="card-text" data-body>{{ videojuego.precio }} €</p>
            </div>

            <ul class="list-group list-group-flush">
                <li class="list-group-item">{{ videojuego.categoria.nombre_categoria }}</li>
            </ul>

            <div class="card-body">
                <a class="btn btn-primary" href="{{ url_for('catalog.metodo_insertar_carrito', id_usuario=current_user.id) }}>Comprar</a>
            </div>
        </div>
    {% endfor %}

```

```

, id_videojuego=videojuego.id) }}">Añadir al carrito</a>
    <a class="btn btn-primary" href="{{
url_for('catalog.metodo_insertar_favoritos',
id_usuario=current_user.id , id_videojuego=videojuego.id)
}}">Favoritos</a>
    </div>
</div>
{ % endfor %}
</div>

```

Vemos a su vez que tenemos algunos enlaces que llaman a un método de vistas con varios parámetros para el tema de dar funcionalidad a los dos *sidebars*. A su vez, tenemos varias clases en la mayoría de elementos de los cards ya que como antes hemos mencionado, nuestro *script* necesita ciertas clases para poder darle funcionalidad a la barra de navegación, que es dependiente de los *cards* que se muestran en el *DOM*.

Visualizar datos de un sidebar

En cuanto a la visualización de los datos en el *sidebar*, hemos usado un bucle for en el *template* de *Vue*. En el archivo de vistas hemos obtenido los datos en una variable que hemos utilizado para el renderizado y es gracias esto que podemos visualizar los datos de la tabla de Videojuegos en un *card* de *Bootstrap*.

```

<div id="mySidebar" class="sidebar">
    <a href="javascript:void(0)" class="closebtn"
onclick="closeNav()">x</a>

    <h2>Carrito</h2>

    <div class="contenedorsidebar">
        { % for itemdetalle in detalle_html %}
            <h3>{{ itemdetalle.videojuego.titulo }}</h3>
            <p>{{ itemdetalle.videojuego.precio }} €</p>
            <a class="a_vaciar_individual" href="{{
url_for('catalog.vaciar_individual_detalles',
detalle_id=itemdetalle.id) }}">Quitar del carrito</a> <br> <br>
        { % endfor %
    
```

```

</div>

<h4>El precio total es: {{ precio_total_html }} €</h4>
<a id="a_comprar" href="{{ url_for('catalog.proceso_compra',
total=precio_total_html) }}">Compra Ya!</a>

<br><br>
<a id="a_vaciar" href="{{ url_for('catalog.vaciar_carro')
}}">Vaciar el carrito</a>
</div>

```

Gracias al *for* y a la variable renderizada, podemos acceder a los campos de la tabla Videojuegos, por otro lado, vemos en el código que los botones llaman a un método de vistas con un par de parámetros (explicado en la parte de vistas) para poder realizar el proceso de compra, borrar elementos o vaciar todo el *sidebar*.

Visualizar el historial

Para visualizar el historial, hemos hecho un bucle iterando sobre un diccionario de *Python* y obteniendo los datos desde ahí.

```

<div id="historial_de_compras">

    <h2 id="titulo_historial">Historial de Compras</h2>
    {% if detalles_historial_html %}

        <ul id="lista_compras">
            {% for videojuego_id, detalles in
detalles_historial_html.items() %}
                <li>
                    
                    <div>
                        <h2 id="titulo_videojuego">{{
detalles.titulo }}</h2>
                        <p id="precio_videojuego">Precio: {{
detalles.precio }} €</p>

```

```

        </div>
    </li>
    {%
      endfor %
    </ul>
    {%
      else %
    <p>No tienes compras registradas.</p>
    {%
      endif %
    </div>
  
```

Podemos observar que para iterar sobre un diccionario necesitamos clave y valor iterando sobre `.items()` y es gracias a este último que podemos acceder a estos datos.

Despliegue de aplicaciones web

Base de datos

Diagrama E/R

Antes de empezar con el código decidimos realizar un diagrama de entidad-relación para evitar problemas con las relaciones a la hora de insertar los datos en pgAdmin4. Comenzamos identificando las entidades de nuestra empresa.

Entidades:

| Entidades | Atributos |
|----------------------------|--|
| Usuarios | Id, Nombre, Contraseña, Correo_electronico |
| Videojuegos | Id, Título, Precio, Id_categoria, Imagen |
| Videojuegos_usuarios (n:m) | Id, Favoritos, Id_videojuego |
| Compras | Id, Id_usuario |
| Compras_videojuegos (n:m) | Id, Id_compra, Id_videojuego |
| Categoría | Id, Nombre |
| Detalles | Id, Id_usuario, Id_videojuego |

| | |
|-------------|----------------------------|
| Comentarios | Id, Id_usuario, Caja_texto |
|-------------|----------------------------|

Relaciones:

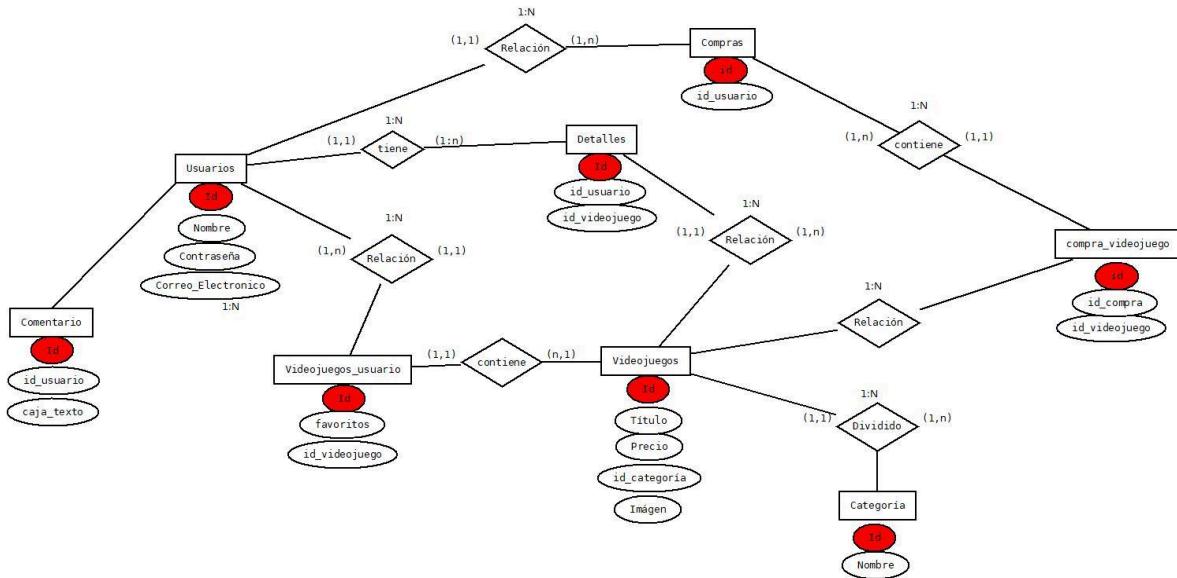
Usuarios-Videojuegos(N:M): En esta relación vemos que un usuario puede seleccionar sus juegos favoritos donde no puede tener videojuegos favoritos o puede haber varios. A su vez, los videojuegos pueden no ser favoritos o ser favoritos de varios usuarios, es decir, tenemos una relación N:M llamado Videojuegos-usuarios donde tendremos como id foráneos el id_usuario y el id_videojuego.

Usuarios-Videojuegos(N:M): Ya que un usuario puede comprar varios videojuegos y varios videojuegos pueden pertenecer a varios usuarios, por este motivo, creamos una tabla extra llamada detalles donde sus campos tendrá el id primario y sus campos serán los id de usuarios y videojuegos en forma de id foráneos. A su vez, hay otra tabla que contiene los mismos campos, sin embargo, es otra tabla donde se hace referencia a los juegos seleccionados como favoritos.

Usuarios-Compras (1:N): La entidad de compras es la referencia al historial de compras que está relacionada con usuarios, ya que cada compra pertenece a un cliente y un cliente puede realizar varias compras.

Usuarios-Comentarios (1:N): En esta relación los usuarios pueden realizar varios comentarios donde cada comentario pertenece a un solo usuario.

Videojuegos-Categoría(1:N): Es una relación simple donde cada juego pertenece a una categoría y cada categoría tiene varios videojuegos. En la siguiente se muestra nuestro diagrama de entidad-relación.



EIE (Empresa e Iniciativa Emprendedora)

¿Cuánto cobraríamos nosotros por hacer este trabajo para una empresa?

Nosotros por hacer este trabajo para una empresa cobraríamos unos 5200 € haciendo los siguientes cálculos, si esos 5200 € los dividimos entre 5 personas nos saldría a 1040 € por persona y hemos hecho ese cálculo viendo que una persona con un poco de experiencia cobra 30 € la hora y al nosotros no tener experiencia hemos pedido 20 €.

Los 1040 € los hemos conseguido de la siguiente manera, como el reto dura 2 semanas y son 26 horas de clase a la semana lo duplicamos por 2 porque son las semanas de las que disponemos para terminar el reto, y al tener esas 52 horas hemos decidido cobrar eso.

Herramientas Utilizadas

Gimp

Herramienta OpenSource similar a PhotoShop utilizada para la edición, compresión y escalado de las imágenes.

VisualStudio

Un editor de textos o entorno de desarrollo necesario para poder trabajar con los lenguajes que hemos utilizado y poder ejecutar dicho código para llevar a cabo el proyecto

YouTube

La herramienta de Google conocida por tener una cantidad gigante de videos de todo tipo, en nuestro caso la hemos empleado para ver algún tutorial sobre algún código requerido

PgAdmin 4

Un cliente de base de datos con lenguaje *SQL* y *PostgreSQL* compatible con varios lenguajes de programación.

Drive

La herramienta de Google para poder almacenar documentos, archivos... en la nube de manera gratuita, a su vez podemos ir subiendo el trabajo en una carpeta alcanzable para todos los miembros del grupo

Dia

Creación del diagrama E/R

Pixlr

Creación del Logo de la web.

Planes a Futuro

Para los planes a futuro, tenemos varios apartados ya pensados, que sería los siguientes:

- Más categorías
- Ampliar el catálogo de juegos
- Ofertas en los juegos
- Reservar juegos
- Aplicar diferentes divisas
- Poner la fecha de la compra al ser realizada
- Visualización de la ventana de Stripe

Con estas implementaciones para futuro, creemos que se podría mejorar el nivel de la página y el servicio que le podríamos ofrecer a los usuarios mejoraría considerablemente con estos puntos, ya que nos acercaremos bastante a los servicios mínimos que otorgan hoy en día este tipo de páginas.

Bibliografía

Vídeo de relación N:M en flask

<https://www.youtube.com/watch?v=47i-jzrrIGQ>

Vídeo de barra de búsqueda

https://www.youtube.com/watch?v=TIP5WIxVirU&ab_channel=WebDevSimplified

Footer

<https://bootstrapexamples.com/@isaac/clean-modern-footer-design-with-social-media>

Planes de Pago

<https://frontend.com/blog/bootstrap-pricing-table-snippets/>

