

Reto 1º Eval, Grupo B

Desarrollo de Aplicaciones Web

Alejandro Rebollo, Markel Azpiazu, Markel Irastorza, Julen Salinas y Cristina Bayona

Grupo B

<b>Introducción del Reto.....</b>	<b>2</b>
<b>Empresa e Iniciativa Emprendedora.....</b>	<b>3</b>
Introducción.....	3
Financiación.....	3
<b>Desarrollo Web en Entorno Servidor.....</b>	<b>4</b>
Introducción.....	4
Trabajo realizado.....	4
Modelos.py.....	4
Base de datos.....	5
Tablas y relaciones - Diagrama y modelo lógico.....	5
Tablas y relaciones código Python.....	6
Vistas.py.....	8
Vistas para eventos pasados.....	8
Vistas para registrarse.....	9
Vistas para iniciar la sesión.....	10
Controlador.py.....	10
Funciones del programa.....	10
Funciones para mostrar datos.....	10
Funciones para registro e inicio de sesión:.....	11
PATH.....	14
<b>Despliegue.....</b>	<b>17</b>
Introducción.....	17
Trabajo realizado.....	17
<b>Interfaces Web.....</b>	<b>18</b>
Introducción.....	18
Trabajo realizado.....	18
<b>Problemas.....</b>	<b>24</b>
<b>Planes a futuro.....</b>	<b>24</b>
<b>Conclusiones.....</b>	<b>25</b>

## Introducción del Reto

Para el desarrollo del reto de Desarrollo de Aplicaciones Web (DAW2) de la 1ª evaluación se nos ha pedido realizar una página web donde se muestre resultados deportivos de una liga específica. Dichos resultados de los partidos deben obtenerse desde una API donde los datos estén en una base de datos donde se irá actualizando en un plazo de tiempo.

El documento está distribuido en asignaturas con el objetivo de facilitar al profesorado en lugar donde se sitúa su materia, cada asignatura está estructurada de la siguiente manera. Cada apartado contiene una introducción donde se explicará brevemente lo que nos pedía para este reto. Posteriormente, está el apartado “trabajo realizado” donde se muestra toda la información recopilada por el grupo.

Para finalizar, se encuentran los apartados donde indicamos los problemas que hemos tenido en estas dos semanas, las ideas a futuro que tenemos en caso de retomar los trabajos y las conclusiones sobre este reto por parte de cada integrante de este grupo.

# Empresa e Iniciativa Emprendedora

## Introducción

Para esta asignatura se nos pide que expliquemos en el informe del reto la rentabilidad de nuestra página web, es decir, de qué manera vamos a obtener un beneficio mediante la página.

## Financiación

La manera que se nos ha ocurrido de conseguir rentabilidad de la página web (Calciopoli) es crear un apartado de merchandising donde los usuarios podrán comprar ropa, bufandas y más cosas de tu equipo favorito de la liga italiana.

Por otra parte, se nos ha ocurrido poner a futuro anuncios para la página ya que muchas páginas hoy en día se basan en poner anuncios con el objetivo de financiar la página web. A su vez, otra forma de financiación sería permitir a los usuarios la posibilidad de realizar apuestas deportivas. Las apuestas deportivas generan hoy en día una gran cantidad de dinero.

# Desarrollo Web en Entorno Servidor

## Introducción

En esta parte del trabajo se corresponde a la parte de la conexión con el servidor donde se debía realizar mediante el lenguaje Python. En este apartado se mostrarán los pasos realizados, tanto la conexión de la API para la obtención de datos y posteriormente la inserción en la base de datos. Por otra parte, también se realizó mediante *Python* el registro e inicio de sesión de nuestra página web.

## Trabajo realizado

### Modelos.py

El archivo modelos.py es el archivo de *Python* que se conecta con una base de datos local de PostgreSQL. Por otro lado, en este archivo, creamos algunos métodos para abrir o cerrar sesión, leer consultas...

En cuanto a la inserción de datos, hemos obtenido los resultados de los partidos mediante una API :

<https://github.com/openfootball/football.json/blob/master/2024-25/it.1.json>

He aquí la parte de código de la inserción de datos:

```
def insertar_partidos_desde_json():
    url =
    "https://raw.githubusercontent.com/openfootball/football.json/refs/he
ads/master/2024-25/it.1.json"
    response = requests.get(url)

    if response.status_code == 200:
```

```

data = response.json()
partidos = data['matches']

# Crear una sesión
Sesion = sessionmaker(bind=engine)
sesion = Sesion()

for partido in partidos:

    if 'score' in partido and 'ft' in partido['score']:
        marcador_local = partido['score']['ft'][0]
        marcador_visitante = partido['score']['ft'][1]
    else:
        marcador_local = None
        marcador_visitante = None

    nuevo_partido = Partidos(
        equipo_local=partido['team1'],
        equipo_visitante=partido['team2'],
        fecha=partido['date'],
        marcador_local=marcador_local,
        marcador_visitante=marcador_visitante
    )
    sesion.add(nuevo_partido)

```

Tabla 1

## Base de datos

### Tablas y relaciones - Diagrama y modelo lógico

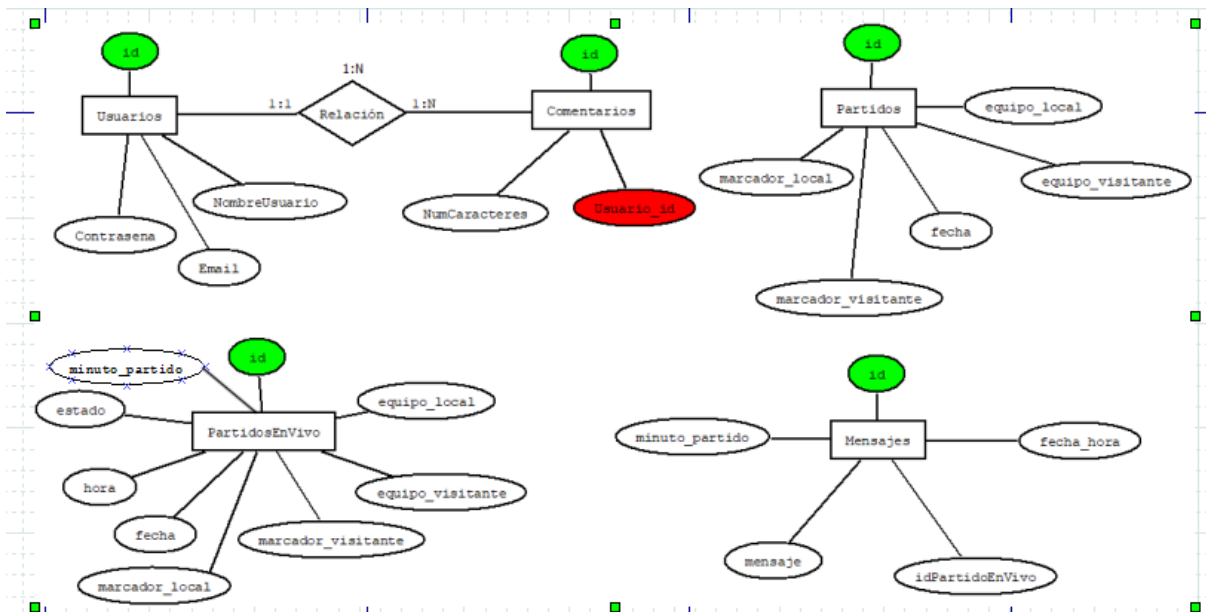


Imagen 1 : Diagrama Entidad Relación

## Tablas y relaciones código Python

En cuanto a la definición de nuestras tablas en Python, hemos seguido el diagrama entidad-relación antes mencionado. Aquí tenemos el código de la definición y relación de las tablas:

```
# Definir las tablas
class Usuarios(Base,miCRUD):
    __tablename__ = 'Usuarios'
    id = Column(Integer, primary_key=True)
    NombreUsuario = Column(String)
    Contraseña = Column(String)
    Email = Column(String)
    relacion_comentarios = relationship("Comentarios",
back_populates="relacion_usuarios")

class Comentarios(Base):
    __tablename__ = 'Comentarios'
    id = Column(Integer, primary_key=True)
    maxCaracteres = Column(String)
    Usuario_id = Column(Integer, ForeignKey('Usuarios.id'))
    relacion_usuarios = relationship("Usuarios",
back_populates="relacion_comentarios")
```

```

class Partidos(Base,miCRUD):
    __tablename__ = 'Partidos'
    id = Column(Integer, primary_key=True)
    equipo_local = Column(String)
    equipo_visitante = Column(String)
    fecha = Column(Date)
    marcador_local = Column(Integer)
    marcador_visitante = Column(Integer)


class PartidosEnVivo(Base,miCRUD):
    __tablename__ = 'PartidosEnVivo'
    id = Column(Integer, primary_key=True)
    fecha = Column(Date)
    hora = Column(String)
    equipo_local = Column(String)
    equipo_visitante = Column(String)
    marcador_local = Column(Integer)
    marcador_visitante = Column(Integer)
    estado = Column (String)
    minuto_partido = Column (Integer)
    relacion_Mensajes = relationship("Mensajes",
back_populates="relacion_PartidosEnVivo")


class Mensajes(Base,miCRUD):
    __tablename__ = 'Mensajes'
    id = Column(Integer, primary_key=True)
    idPartidoEnVivo = Column(Integer,
ForeignKey('PartidosEnVivo.id'))
    mensaje = Column (Text)
    fecha_hora = Column (DateTime)

    minuto_partido = Column (Integer)
    relacion_PartidosEnVivo = relationship("PartidosEnVivo",
back_populates="relacion_Mensajes")

```



--

Tabla 2

En base a la inserción de datos creamos una variable para cada campo del JSON para posteriormente crear las tablas en formato *Python* para su inserción en pgAdmin4. A la hora de visualizar los datos.

## Vistas.py

Un archivo de vista o una vista, es una función o programa en *Python* que hace una solicitud Web y devuelve una respuesta Web. Esta respuesta puede ser el contenido de una página, un error 404, una imagen etc.

Hemos usado un programa de vistas para cada página de mostrar partidos, ya que no hemos visto otra manera de poder computar todo en un solo archivo de vistas, ya que cada uno de estos pide una página html, he aquí un ejemplo:

A continuación vamos a mostrar 3 ejemplos de los archivos vistas que tenemos y estos serán los más diferentes entre sí : A continuación explicamos el archivo de vistas para los partidos ya jugados, el archivo vistas para el registro y el archivo vistas para iniciar la sesión:

## Vistas para eventos pasados

Empezamos asignando un *template* para este archivo:

```
env = Environment(loader=FileSystemLoader('templates'))
template = env.get_template('eventos_pasados.html')
```

Tabla 3

Al igual que en otros archivos de vistas, este programa incluye funciones como por ejemplo, en casos de error 404, en caso de archivos estáticos... A pesar de compartir estas funcionalidades con otros programas de vistas, tiene una función específica para mostrar los partidos que ya han sido disputados:

```
def eventos_anteriores(environ, start_response, eventos):
    response = template.render(eventos=eventos).encode('utf-8')
    status = '200 OK'
    response_headers = [('Content-type', 'text/html')]
    start_response(status, response_headers)
    return [response]
```

Tabla 4

*eventos\_anteriores()* tiene relación con la función de mostrar partidos pasados en el programa controlador.py llamado *mostar\_eventos()* y este a su vez con su PATH específico.

## Vistas para registrarse

Empezamos asignando un *template* para este archivo:

```
env = Environment(loader=FileSystemLoader('templates'))
template = env.get_template('registro_sesion.html')
```

Tabla 5

Este archivo de vistas, solo tenemos una función que se encarga de renderizar el template de la página de registro:

```
def registro_sesion(environ, start_response):
    response = template.render().encode('utf-8')
    status = '200 OK'
    response_headers = [('Content-type', 'text/html')]
    start_response(status, response_headers)
    return [response]
```

Tabla 6

*registro\_sesion()* tiene relación con la función para realizar esta tarea el programa controlador.py llamado *add\_user()* y este a su vez con su PATH específico.

## Vistas para iniciar la sesión

Empezamos asignando un *template* para este archivo:

```
env = Environment(loader=FileSystemLoader('templates'))
template = env.get_template('iniciar_sesion.html')
```

Tabla 7

Al igual que en otros archivos de vistas, este programa incluye funciones como por ejemplo, el de los archivos estáticos. A pesar de compartir estas funcionalidades con otros programas de vistas, tiene otras funciones : *sesion\_init()*, *logica\_sesion()*, *sesion\_finish()*, *handle\_index()* y *no\_user\_handle()*

## Controlador.py

Este programa es el que controla en términos generales las páginas web y el resto de programas de *Python*. El controlador entre otras cosas contiene el código fuente necesario para responder a las acciones que el usuario solicita y sirve como puente de interacción entre las vistas y el modelo de datos.

Podemos dividir el programa *controlador.py* en varias partes:

## Funciones del programa

En este apartado del programa, definimos las funciones necesarias para realizar las tareas que se nos piden. A continuación vamos a definir las distintas funciones por partes, ya que tienen funcionalidades diferentes:

### Funciones para mostrar datos

En *controlador.py* definimos 3 funciones (una para cada página de mostrar datos) para visualizar datos en la web que provienen de nuestra base de datos. A continuación tenemos la definición de éstas (*mostrar\_eventos()* para los partidos que ya se han jugado, *mostrar\_eventos\_en\_vivo()* para los partidos en vivo y *mostrar\_eventos\_futuros()* para los partidos que se jugarán próximamente):

```

def mostrar_eventos():
    sesion = modelos.abrir_sesion()
    hoy = datetime.now().strftime("%Y-%m-%d")
    eventos =
sesion.query(modelos.Partidos).filter(modelos.Partidos.fecha <=
hoy).filter(modelos.Partidos.marcador_local !=
None).order_by(modelos.Partidos.fecha.desc()).all()
    modelos.cerrar_sesion(sesion)
    return eventos

def mostrar_eventos_en_vivo():
    sesion = modelos.abrir_sesion()
    en_vivo =
sesion.query(modelos.PartidosEnVivo).filter(modelos.PartidosEnVivo.id
!= None).all()

    modelos.cerrar_sesion(sesion)
    return en_vivo

def mostrar_eventos_futuros():
    sesion = modelos.abrir_sesion()
    hoy = datetime.now().strftime("%Y-%m-%d")

    eventos_futuros =
sesion.query(modelos.Partidos).filter(modelos.Partidos.fecha >
hoy).order_by(modelos.Partidos.fecha.asc()).all()

    modelos.cerrar_sesion(sesion)
    return eventos_futuros

```

Tabla 8

Funciones para registro e inicio de sesión:

Por otra parte, tenemos otras funciones para controlar el registro e inicio de sesión. Empecemos con el registro:

Tenemos una función definida llamada `add_user()` que obtiene los datos que el usuario introduce en un formulario y obtiene cada campo del mismo en una variable de *Python*. Tras eso, hacemos un `add()`, un `commit()` y cerramos la sesión. Veamos el código:

```
def add_user(envIRON, start_response):
    if envIRON['REQUEST_METHOD'] == 'POST':
        try:

            size = int(envIRON.get('CONTENT_LENGTH', 0))
            data = envIRON['wsgi.input'].read(size).decode()
            params = parse_qs(data)

            producto = modelos.Usuarios(
                NombreUsuario=params['NombreUsuario'][0],
                Contraseña=params['Contraseña'][0],
                Email=params['Email'][0]
            )
            sesion = modelos.abrir_sesion()
            sesion.add(producto)
            sesion.commit()
            modelos.cerrar_sesion(sesion)
            sesion = None

            # en vez de llamar al método create() hago el
            sesion.add() directamente como en sqlalchemy, además tengo que hacer
            el sesion.comit() ya que importo modelos.py importo a su vez
            sqlalchemy

            start_response('303 See Other', [('Location', '/es')])
            return [b'']
        except Exception as e:
            print(f"Error al agregar usuario: {e}")
            start_response('500 Internal Server Error',
                [('Content-type', 'text/plain')])
            return [b'Error interno del servidor.']
    else:
        return vistas.handle_404(envIRON, start_response)
```

Tabla 9

Ahora comentaremos la parte del código para iniciar la sesión:

Al igual que en el registro, tenemos en este caso un par de funciones para manejar el inicio de sesión. Las funciones utilizadas son las siguientes: *iniciar\_sesion()* y *finalizar\_sesion()*

```
def iniciar_sesion(environ, start_response):
    hayUser = False # Declaramos hayUser como false

    if environ['REQUEST_METHOD'] == 'POST':
        size = int(environ.get('CONTENT_LENGTH', 0))
        data = environ['wsgi.input'].read(size).decode()
        params = parse_qs(data)
        NombreUsuario = params.get('NombreUsuario', [None])[0]
        Contraseña = params.get('Contraseña', [None])[0]
        print('contraseña: ' + Contraseña)
        sesion = modelos.abrir_sesion()

        consulta = {"NombreUsuario": NombreUsuario, "Contraseña":
Contraseña}
        usuarios = modelos.Usuarios.metodo_inicio_sesion(sesion,
**consulta)

        # usuarios =
sesion.query(modelos.Usuarios).filter(modelos.Usuarios.NombreUsuario
== NombreUsuario).filter(modelos.Usuarios.Contraseña ==
Contraseña).all()
        #print('usuario de BD: ' + usuarios.user)
        if usuarios:
            hayUser = True
            return NombreUsuario, sesion, hayUser
        else:
            sesion.close()
            sesion = None
            return None, None, None
```

```

    return None, None, hayUser # tenemos que asegurar que devuelva
    algo aunque no se haga la solicitud el POST

def finalizar_sesion(session):
    # Cerrar la sesión al terminar
    modelos.cerrar_sesion(session)

```

Tabla 10

## PATH

En este apartado final del programa definimos una función que tendrá como condicionales las rutas (PATH) que el usuario visita en la página, y según cada una de las direcciones, se le devolverá algo (un método, un error...). He aquí dicha función llamada *app()*:

```

def app(environ, start_response):
    global sesion, usuario, hayUser
    path = environ.get('PATH_INFO')
    if path == '/':
        return vistas.spanish_handle_index(environ, start_response) #
    aquí no pongo los productos ya que no es un campo obligatorio
    if path == '/index.html':
        return vistas.spanish_handle_index(environ, start_response) #
    aquí no pongo los productos ya que no es un campo obligatorio
    elif path == '/es':
        return vistas.spanish_handle_index(environ, start_response) #

```

```

aquí no pongo los productos ya que no es un campo obligatorio
    elif path == '/add_user':
        return add_user(environ, start_response)
    elif path.startswith('/static/'):
        return vistas.serve_static(environ, start_response)
    elif path == '/favicon.ico':
        start_response('404 Not Found', [('Content-type',
'text/plain')])
        return [b'Favicon no encontrado.']
    elif path == '/logoEmpresa.png':
        start_response('404 Not Found', [('Content-type',
'text/plain')])
        return [b'Foto no encontrada.']
    elif path == '/kebab.png':
        start_response('404 Not Found', [('Content-type',
'text/plain')])
        return [b'Foto no encontrada.']
    elif path == '/venezia.jpg':
        start_response('404 Not Found', [('Content-type',
'text/plain')])
        return [b'Foto no encontrada.']
    elif path == '/roma.jpg':
        start_response('404 Not Found', [('Content-type',
'text/plain')])
        return [b'Foto no encontrada.']

    elif path == '/registro_sesion.html': # Nueva ruta para el
registro
        return vistas_registro.registro_sesion(environ,
start_response) # Llamar a la nueva vista

    elif path == '/merch.html':
        return vistas_merch.merchandising(environ, start_response)

    elif path == '/eventos_pasados.html':
        eventos = mostrar_eventos()
        return vistas_eventos_anteriores.eventos_anteriores(environ,
start_response, eventos)

```



```

elif path == '/eventos_futuros.html':
    eventos_futuros = mostrar_eventos_futuros()
    return
vistas_eventos_posteriores.eventos_posteriores(environ,
start_response, eventos_futuros)

elif path == '/en_vivo.html':
    en_vivo = mostrar_eventos_en_vivo()
    mensajes_en_vivo = mostrar_mensajes_en_vivo()
    return vistas_en_vivo.eventos_en_vivo(environ,
start_response, en_vivo)

elif path == '/login':
    usuario, sesion, hayUser = iniciar_sesion(environ,
start_response)

    return vistas_sesion.logica_sesion(start_response, hayUser)
elif path == '/noUser':
    return vistas_sesion.no_user_handle(environ, start_response)
elif path == '/logout':
    finalizar_sesion(sesion)
    sesion = None
    return vistas_sesion.sesion_finish(start_response, hayUser)

elif path == '/iniciar_sesion.html':
    # Mostrar el formulario de inicio de sesión
    return vistas_sesion.sesion_init(environ, start_response)
else:
    return vistas.handle_404(environ, start_response)

```

Tabla 11

Hay que tener en cuenta que hay casos en los que el PATH no tiene una funcionalidad completa, por ejemplo con respecto a las imágenes, ya que finalmente decidimos obtener imágenes directamente mediante su enlace en Internet.

Finalmente, concluimos el programa asignando el puerto:

```
if __name__ == "__main__":  
    host = 'localhost'  
    port = 888  
  
    httpd = make_server(host, port, app)  
    print(f"Servidor en http://{host}:{port}")  
    httpd.serve_forever()
```

Tabla 12

Por último en cuanto a este apartado respecta, ésta funcionalidad no está del todo terminada y esto se explica a fondo en el apartado de planes a futuro.

## Despliegue

### Introducción

En esta asignatura se nos pide realizar la subida de los archivos a un servidor web, el servidor es el que nosotros queramos.

### Trabajo realizado

Un problema con el que nos topamos fue, que se nos dió libertad a la hora de elegir donde subir la página. Nuestra idea principal fue subirla en Cloudflare pero se nos comentó que este servidor no puede ejecutar lenguaje Python, a su vez, Dinahosting, Awardspace tampoco podían, dejándonos como única posibilidad el servidor de Nazaret pero no dejaba acceder con SSH. Además, por falta de tiempo tenemos todo el trabajo en LocalHost. Por último la falta de tiempo se debe a que no sabíamos instalar Python en el servidor.

Para finalizar, el problema de no subirlo al servidor se debe principalmente a la falta de tiempo que hemos tenido en el reto, consideramos que con una semana más de tiempo seríamos capaces de subir todo el trabajo al servidor de Nazaret.

# Interfaces Web

## Introducción

En esta sección del documento se explica cómo realizamos la estructura de la página mediante HTML (HyperText Markup Language) y su CSS (Cascading Style Sheets). Uno de los requisitos que se nos pedía para el HTML era el uso de Bootstrap. Además, debemos crear una página que sea "responsive".

## Trabajo realizado

La estructura de nuestra página está distribuida de la siguiente manera : Hemos realizado una portada (Index.html) dónde se mostrará un menú dónde podrás navegar a través de nuestra página web. En total, son 4 páginas de HTML sin contar la portada (Partidos Resueltos, En vivo, Partidos a futuro y Merchandising). A su vez, tenemos los botones de Registro y de Iniciar Sesión.

Hemos dividido nuestra web en diferentes páginas HTML. La primera de ellas es la de inicio, "index.html", también tenemos la página "eventos\_pasados.html", "eventos\_futuros.html", "en\_vivo.html", "iniciar\_sesion.html", "registro\_sesion.html" y por último la página destinada al merchandising; "merch.html".

La página principal, la de inicio está dividida por varios "div" con el header de la página, el logo y los botones de iniciar sesión y registrarse en la web. Después hemos colocado un "nav" donde se sitúa el menú de la página. También tenemos una tabla dónde figuran las imágenes de los diferentes equipos de la serie A.

Al final de la página observamos la marquesina con la etiqueta "MARQUEE" dónde colocamos los diferentes logotipos de las empresas patrocinadoras. Y, por último, el "footer", el pie de página.



Imagen 3 :Página de Inicio

La página de eventos pasados nos muestra la información sobre los partidos que se han jugado con anterioridad, mencionando el nombre del equipo local, el visitante, la fecha en la que se jugó el partido y el resultado final del mismo. Para poder obtener los datos tenemos que conectarnos a la base de datos a través del lenguaje de programación de Python.

Esta información se muestra en forma de tabla, con la etiqueta “table” y sus correspondientes filas y columnas, “tr” como table row, “td” como table data y “th” como table header.

Eventos Anteriores			
<div> <div>Inicio</div> <div>En vivo</div> <div>Próximos Partidos</div> <div>Merchandising</div> </div>			
Equipo Local	Equipo Visitante	Fecha	Marcador
Parma Calcio 1913	Genoa CFC	2024-11-04	0 - 1
Empoli FC	Como 1907	2024-11-04	1 - 0
SSC Napoli	Atalanta BC	2024-11-03	0 - 3
FC Internazionale Milano	Venezia FC	2024-11-03	1 - 0
Hellas Verona FC	AS Roma	2024-11-03	3 - 2
Torino FC	ACF Fiorentina	2024-11-03	0 - 1
Udinese Calcio	Juventus FC	2024-11-02	0 - 2
AC Monza	AC Milan	2024-11-02	0 - 1
Bologna FC 1909	US Lecce	2024-11-02	1 - 0
Como 1907	SS Lazio	2024-10-31	1 - 5
AS Roma	Torino FC	2024-10-31	1 - 0
Genoa CFC	ACF Fiorentina	2024-10-31	0 - 1
Venezia FC	Udinese Calcio	2024-10-30	3 - 2
Juventus FC	Parma Calcio 1913	2024-10-30	2 - 2
Atalanta BC	AC Monza	2024-10-30	2 - 0
Empoli FC	FC Internazionale Milano	2024-10-30	0 - 3
US Lecce	Hellas Verona FC	2024-10-29	1 - 0
AC Milan	SSC Napoli	2024-10-29	0 - 2
Cagliari Calcio	Bologna FC 1909	2024-10-29	0 - 2
Parma Calcio 1913	Empoli FC	2024-10-27	1 - 1

Imagen 4 : Partidos ya disputados

La página de eventos futuros está compuesta por una estructura muy similar a la de eventos pasados, únicamente que se incluyen los partidos que se jugarán próximamente, de los cuales aún no se ha obtenido ningún resultado. También hemos generado una tabla para mostrar dichos partidos.

Eventos Futuros			
<div> <div>Inicio</div> <div>Partidos Anteriores</div> <div>En vivo</div> <div>Merchandising</div> </div>			
Equipo Local	Equipo Visitante	Fecha	Marcador
Atalanta BC	Udinese Calcio	2024-11-10	Marcador no disponible, el partido aún no se ha jugado
ACF Fiorentina	Hellas Verona FC	2024-11-10	Marcador no disponible, el partido aún no se ha jugado
AS Roma	Bologna FC 1909	2024-11-10	Marcador no disponible, el partido aún no se ha jugado
AC Monza	SS Lazio	2024-11-10	Marcador no disponible, el partido aún no se ha jugado
FC Internazionale Milano	SSC Napoli	2024-11-10	Marcador no disponible, el partido aún no se ha jugado
Hellas Verona FC	FC Internazionale Milano	2024-11-23	Marcador no disponible, el partido aún no se ha jugado
AC Milan	Juventus FC	2024-11-23	Marcador no disponible, el partido aún no se ha jugado
Parma Calcio 1913	Atalanta BC	2024-11-23	Marcador no disponible, el partido aún no se ha jugado
Genoa CFC	Cagliari Calcio	2024-11-24	Marcador no disponible, el partido aún no se ha jugado
Como 1907	ACF Fiorentina	2024-11-24	Marcador no disponible, el partido aún no se ha jugado
Torino FC	AC Monza	2024-11-24	Marcador no disponible, el partido aún no se ha jugado
SSC Napoli	AS Roma	2024-11-24	Marcador no disponible, el partido aún no se ha jugado
SS Lazio	Bologna FC 1909	2024-11-24	Marcador no disponible, el partido aún no se ha jugado
Empoli FC	Udinese Calcio	2024-11-25	Marcador no disponible, el partido aún no se ha jugado
Venezia FC	US Lecce	2024-11-25	Marcador no disponible, el partido aún no se ha jugado
Cagliari Calcio	Hellas Verona FC	2024-11-29	Marcador no disponible, el partido aún no se ha jugado
Como 1907	AC Monza	2024-11-30	Marcador no disponible, el partido aún no se ha jugado
AC Milan	Empoli FC	2024-11-30	Marcador no disponible, el partido aún no se ha jugado
Bologna FC 1909	Venezia FC	2024-11-30	Marcador no disponible, el partido aún no se ha jugado
Udinese Calcio	Genoa CFC	2024-12-01	Marcador no disponible, el partido aún no se ha jugado

Imagen 5 : Partidos que se jugarán en un futuro

Otra de las páginas de nuestra web es la de “iniciar\_sesion.html”. En esta página los usuarios que ya estén registrados podrán iniciar sesión en su cuenta.

Los campos `<input type="hidden">` son campos que no queremos que aparezcan visibles al usuario que rellena el formulario. Cuando se carga la página esas etiquetas son invisibles. En este caso corresponden al nombre de la base de datos y de la tabla en las cuales va a comparar la información introducida por el usuario.

Tenemos dos campos “required” ya que si no se rellenan, el formulario es inválido y no se puede llevar el proceso adelante. Son el nombre del usuario y su contraseña, datos que él debe de insertar.

Después tenemos un botón para comprobar los datos y finalmente un enlace que el usuario puede clicar en caso de no tener cuenta, para realizar el registro.



El formulario para iniciar sesión está centrado y contiene los siguientes elementos:

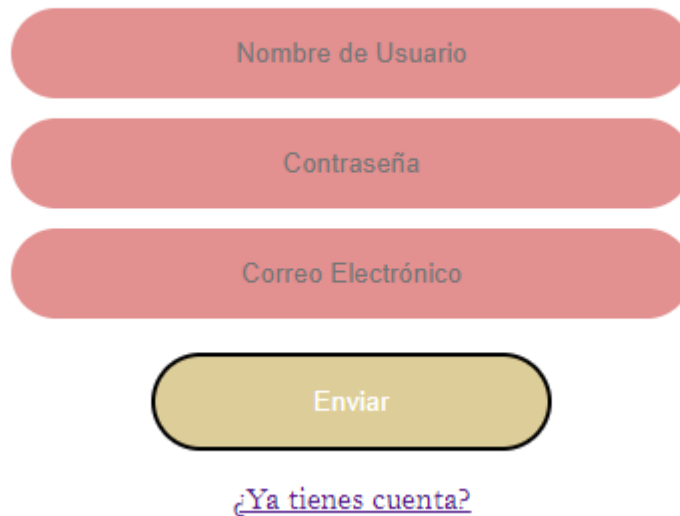
- Un título "Inicia Sesión" en un color azul oscuro.
- Un campo de entrada para el "Nombre de Usuario" con un fondo rosa pálido.
- Un campo de entrada para la "Contraseña" con un fondo rosa pálido.
- Un botón "Inicia Sesión" con un fondo amarillo claro y un contorno negro.
- Un enlace hipertexto "¿No tienes cuenta?" en color morado.

Imagen 6 : Formulario para iniciar sesión

La página de registro es otro formulario que mantiene una estructura parecida a la de inicio de sesión, varía en cuanto que aquí introducen sus datos por primera vez los usuarios que quieren registrarse. Todos los campos (Usuario, contraseña y correo) son campos requeridos para poder realizar el registro correctamente.

Igual que en el caso anterior, también hay un enlace al final de la página por si acaso ya tiene el usuario una cuenta y quiere ir directamente a la página de inicio de sesión.

## Regístrate



A registration form titled "Regístrate". It consists of three stacked, rounded rectangular input fields with a light red background and a thin black border. The first field is labeled "Nombre de Usuario", the second "Contraseña", and the third "Correo Electrónico". Below these fields is a single, wider, rounded rectangular button with a light yellow background and a thin black border, labeled "Enviar". At the bottom of the form is a link that reads "¿Ya tienes cuenta?" in a purple, underlined font.

Imagen 7 : Formulario para registrarse

Otra de las páginas de gran relevancia de nuestra web es la “merch.html”, aquí se encuentra el merchandising de la liga italiana.

Tenemos un total de 20 productos, las camisetas con sus diferentes precios pueden añadirse al carrito de la compra.

El elemento básico que hemos utilizado es el bloque de construcción “card”, con la etiqueta “card-body”, nos permite generar una tarjeta con un padding alrededor. Dentro de cada una de las tarjetas se muestra la imagen del logo que representa a la camiseta que estamos vendiendo.

Cada tarjeta incluye un “card header”, “card tittle”, “card-text” (dónde colocamos la imagen), “card-footer” (donde mostramos el nombre y el precio de la camiseta).

La anchura de las tarjetas la hemos establecido cogiendo como base el elemento raíz, por eso hemos usado la etiqueta “rem”.

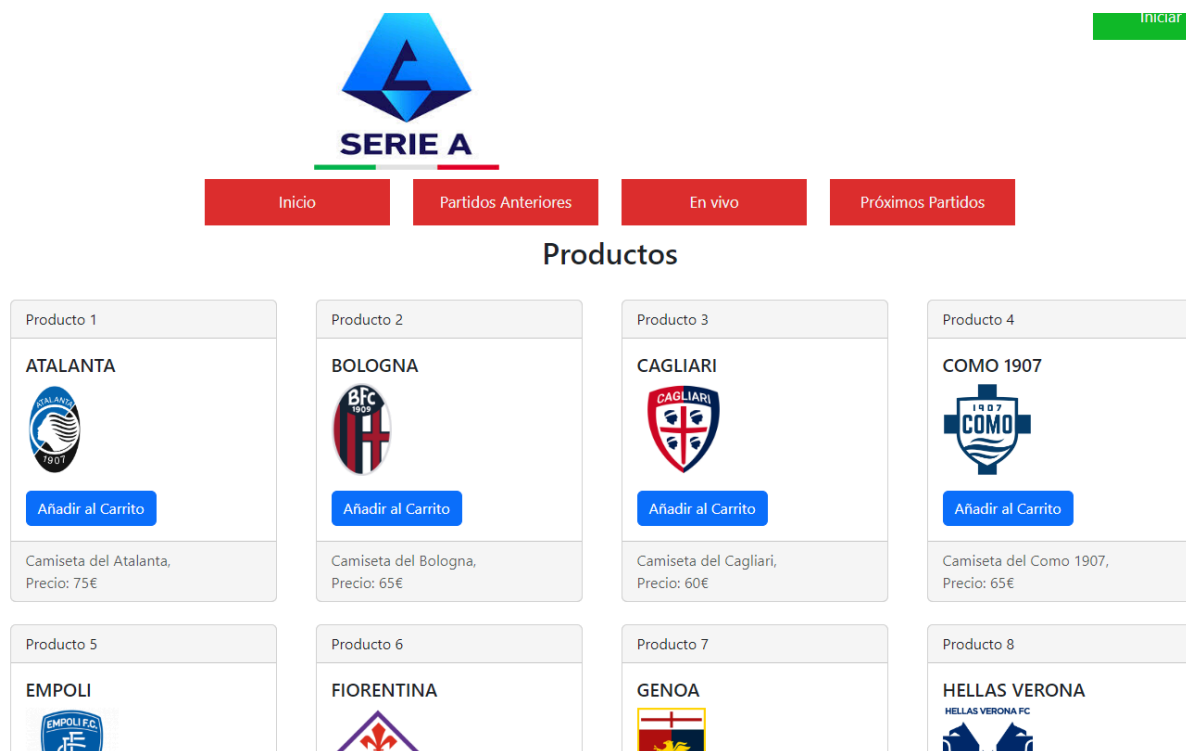


Imagen 8 : Página de merchandising

Y por último se encuentra la página “en\_vivo.html”. Está página la hemos generado basándonos en una tabla que muestra el resultado en directo de un único partido. Es una página sencilla dónde la complejidad la encontramos en la consecución de los datos que se van a mostrar.

Todos los eventos que sucedan durante el partido se mostrarán en los mensajes de la página. Hemos insertado los valores en el HTML dado que no era posible obtener los datos de dos tablas de la base de datos de manera simultánea, pero la información que se muestra en los mensajes está en la base de datos.

Inicio

Partidos Anteriores

Próximos Partidos

Merchandising

Equipo Local	Equipo Visitante	Marcador	Estado	Minuto de Juego
<div>AS Roma</div> 	<div>Juventus FC</div> 	1 - 1	Segundo Tiempo	82

Mensaje	miunto_partido
Tarjeta amarilla para Koné	10
Gol de Vlahovic	31
Gol de Dybala	55

@2024 calciopli

Imagen 9 : Partidos en vivo



Hemos incluido una “Meta Description” en cada una de las páginas HTML del proyecto, además de textos alternativos en cada una de las imágenes que hemos incluido. Ambas acciones nos ayudan en el posicionamiento web.

## Problemas

Uno de los principales inconvenientes del reto fue la falta de información con respecto al código, es decir, ya que debíamos realizar gran parte del código en lenguaje *Python* y tuvimos la necesidad de pedirle al profesor ayuda. Un ejemplo fue el hecho de realizar un formulario de inicio de sesión mediante *Python* (algo que nunca habíamos hecho). Finalmente, se pudo crear el formulario de Inicio Sesión, sin embargo, perdimos un valioso tiempo.

Otro problema con el que nos topamos, fue que se nos dió libertad a la hora de elegir donde subir la página. Nuestra idea principal fue subirla en Cloudflare pero se nos comentó que este servidor no puede ejecutar lenguaje Python, a su vez, Dinahosting, Awardspace tampoco podían, dejándonos como única posibilidad el servidor de Nazaret, pero no dejaba acceder con SSH. Además, por falta de tiempo tenemos todo el trabajo en LocalHost. Por último, la falta de tiempo se debe a que no sabíamos instalar *Python* en el servidor.

Otro de los problemas ha sido el tema de las imágenes, una vez ejecutamos el servidor local no cargan las imágenes en la página y como alternativa decidimos emplear enlaces de Internet.

## Planes a futuro

1. Añadir apuestas deportivas a la aplicación: Cómo mucha gente sabe en estos tiempos las casas de apuestas dan mucho dinero a las empresas para que ellas las patrocinen.
2. Añadir el botón de ‘LIKE’: Tenemos pensado que el usuario pueda reaccionar a los comentarios de otros usuarios, por ejemplo, añadiendo el mítico botón de me gusta.
3. Contactar con marcas: Lo que queremos hacer es que las marcas contacten con nosotros para que se publiquen en nuestra página.
4. Noticias: Queremos implementar noticias sobre la Serie A y sobre otras ligas y sobre fútbol.

5. Terminar de desarrollar la funcionalidad de iniciar sesión, ya que por el momento se puede iniciar sesión si el usuario con esa contraseña existe en la base de datos (si no, no te redirige a *index.html*) pero no nos ha dado tiempo a realizar la muestra del *username* en la página ni tampoco el cierre de la misma, es decir, un botón para poder cerrar la sesión del usuario.
6. Poder desplegar todo nuestro proyecto a un servidor que sea capaz de ejecutar el lenguaje de programación *Python*.
7. Poder emplear las imágenes descargadas en vez de las de Internet. Una vez hecho esto, podríamos reducir el tamaño de las mismas cambiando su extensión a *.webp* y reduciendo su escala.

## Conclusiones

### **Cristina Bayona Marcos**

Este reto me ha permitido aprender a introducir datos en una base de datos a través del lenguaje de programación de python, además de aplicar nuevos selectores de css. También a negociar con mis compañeros y trabajar en equipo con ellos.

### **Alejandro Rebollo Salegui**

En este reto he sentido que hemos tenido más problemas que en otros ya que considero que nos ha faltado bastante información al inicio del reto, a su vez, tampoco ha ayudado la reducción del reto de 3 a 2 semanas. Por último, considero que el grupo ha trabajado bastante bien y hemos podido entregar un trabajo bastante decente.

### **Markel Azpiazu Centeno**

Mi conclusión personal sobre el reto es la siguiente. Ha sido más complicado que los del año pasado, ya que los problemas han sido más frecuentes y de un calibre superior en el sentido de buscar la solución. La reducción del tiempo del proyecto también nos ha afectado, ya que nos hemos sentido más presionados a la hora de poder terminar el trabajo cómodamente. Lo positivo del proyecto se podría decir que ha sido muy completo y ha tocado todos los temarios de esta evaluación.

### **Julen Salinas**

En cuanto a mi respecta, el planteamiento del reto me pareció muy confuso desde el principio al igual que pobre en cuanto a contenido dado en clase, ya que la grán mayoría de este tuvimos que improvisar sobre la marcha con píldoras. Por otro lado, en cuanto a mis

compañeros, considero que alguno ha trabajado bastante más que otro. Por último y en base a mi punto de vista, ha sido un reto desesperante en cuanto a tiempo y contenido, pero como algo positivo puedo decir que he aprendido cosas en *Python* que solo sabía hacer en otros lenguajes, siendo impensable para mi poder conseguirlo en *Python* .

### **Markel Irastorza**

Este reto, para mí ha sido más complicado porque hemos tenido que hacer más cosas en menos tiempo y hemos tenido muchas complicaciones, pero gracias a mis compañero lo hemos sacado adelante