Proyecto de fin de Grado

Aplicación: Peachstats
IES TETUAN DE LAS VICTORIA

Diego Cuenca Arriols

Contenido

Naturaleza del proyecto

Descripción del proyecto

Nuestra aplicación consiste en un gestor de estadísticas deportivas para equipos de baloncesto.

Extrae estadísticas básicas de otras aplicaciones, almacena esos datos y calcula estadísticas avanzadas de la temporada o partidos concretos. Presenta luego toda esta información de manera simple y sencilla de entender.

De esta manera, luego el equipo técnico (entrenadores, delegados...) de cada equipo puede emplear esta información para tomar decisiones estratégicas.

Justificación

Los orígenes de esta aplicación surgen de observar la manera de trabajar de mi padre, el cual como afición se dedica a la tarea de delegado para un equipo de baloncesto llevando estadísticas y otras gestiones. Toma estadísticas durante los partidos y luego mediante hojas Excel y revisando el partido en video, elabora informes con estadísticas avanzadas para el entrenador. También sube esta información a un blog Wordpress donde los jugadores y sus compañeros del equipo técnico pueden consultar el rendimiento del equipo.

Esta tarea es costosa y consume mucho tiempo por lo que la aplicación busca simplificar este proceso. Con tan solo mandar a la aplicación las estadísticas básicas que se toman en el partido, esta genera las estadísticas avanzadas, crea informes y presenta la información en un sitio web donde los jugadores y demás miembros del equipo pueden consultarlo.

Beneficiarios

Aparte de los beneficiarios de donde surgió la idea de la aplicación (mi padre y el equipo Nacional Femenino Corazonistas del que forma parte) La aplicación puede ser empleada por otros equipos de la categoría, una categoría de alto rendimiento pero con presupuestos modestos que no pueden permitirse pagar a una gran empresa para que gestione sus estadísticas (como hacen grandes equipos de la ACB).

Nuestra aplicación es por tanto una solución económica y simple para equipos más modestos y con una necesidad de conocer el rendimiento estadístico del equipo a un nivel medio-alto.

Definición de los Objetivos

Contamos con un único objetivo general:

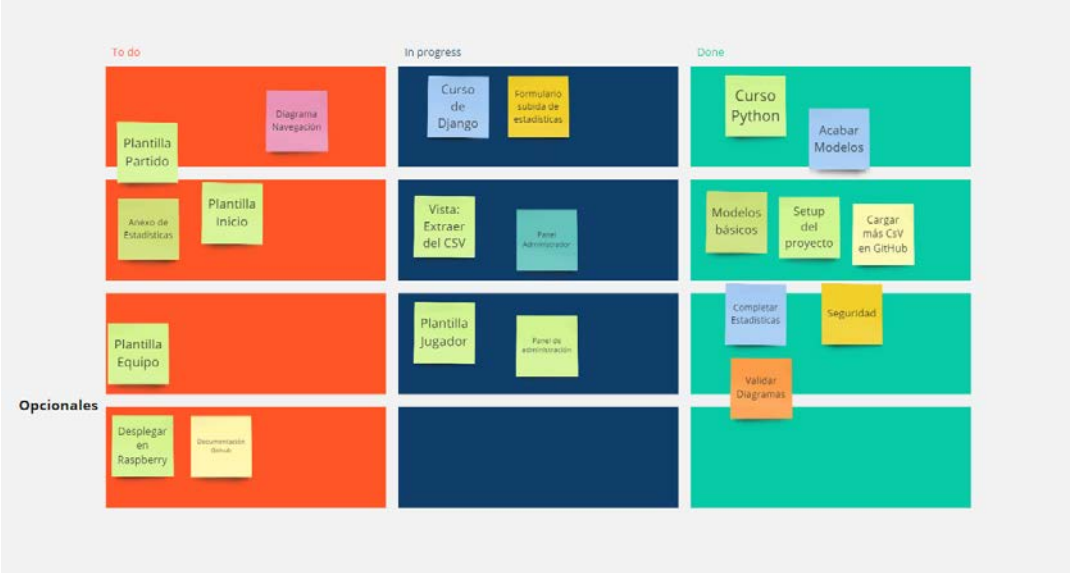
- Calcular estadísticas avanzadas y mostrarlas de manera atractiva: El cual podemos dividir en varios objetivos específicos
    - Extraer la información de nuestras fuentes y guardarla en bases de datos.
    - Calcular estadísticas avanzadas
    - Construir una interfaz para mostrar la información
    - Añadir un criterio de búsqueda para navegar
    - Presentar la información de manera atractiva y fácilmente entendible.
    - Exportar la información a otros formatos.

Tareas

Podemos dividir las tareas a realizar en los siguientes grandes grupos:

- Formación: Adquirir familiaridad con las tecnologías implementadas en el proyecto mediante cursos, documentación..
- Documentación: Elaboración del presente documento.
- Backend: Construir la parte Back de nuestra aplicación, concretamente nuestros Modelos y Controladores (aunque más adelante veremos que Django emplea una nomenclatura distinta para esta estructura).
- Extraer la información de los .CSV que recibimos.
- Frontend: Elaborar las Vistas con las que interactuarán nuestros usuarios

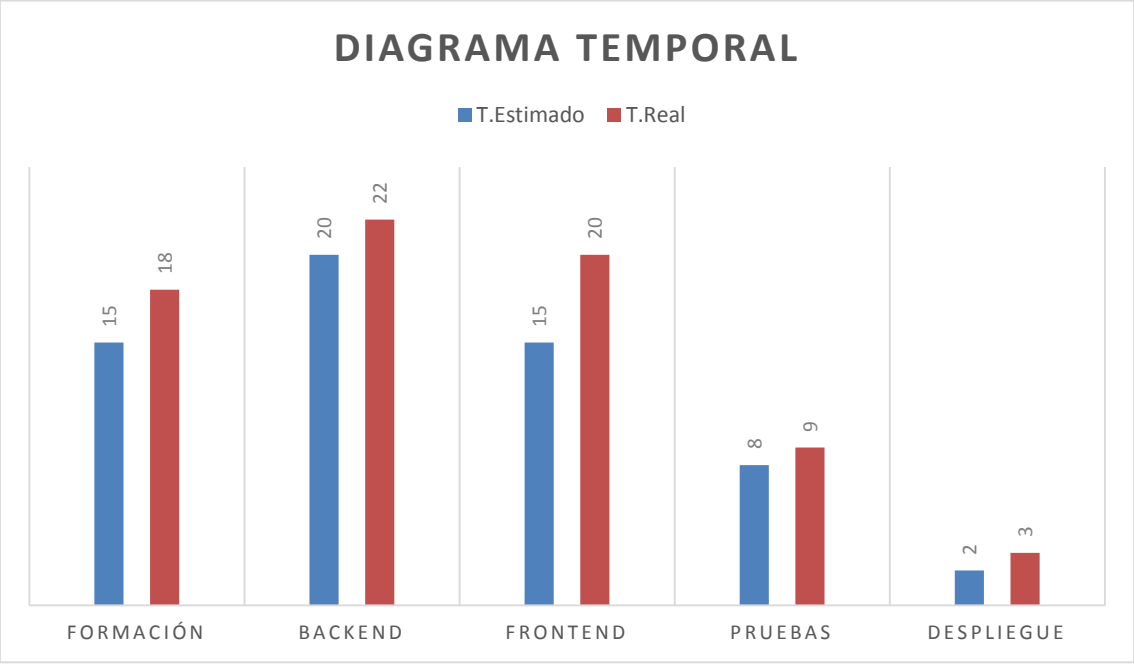
Como anotación, se ha empleado la aplicación “Miro” para organizar las tareas. Se trata de una aplicación que simula un tablero Kanban, pudiendo indicar tiempos y otras opciones más avanzadas que dada la envergadura del proyecto no se han usado.



Estado del tablero en un momento indeterminado del proyecto

Diagrama Temporal

Tarea	T. Estimado	T. Real
Formación y documentación	15h	18h
Codificación: Backend	20h	22h
Codificación: Frontend	15h	20h
Pruebas	8h	9h
Despliegue	2h	3h
TOTAL	60h	72h



Desarrollo de la Aplicación

Tipos de Usuario

Contamos con dos tipos de usuario:

- Jugadores y equipo-técnico: Equivalen al usuario estándar de nuestra aplicación, podrán usar todas las funcionalidades de la aplicación: consulta y exportación de datos.

- Delegado: Equiparable al administrador de la aplicación, aparte de contar con las mismas funcionalidades de un jugador podrá acceder al panel de administrador de la aplicación. Lugar donde se puede editar el contenido de la base de datos manualmente, aparte podrá agregar información de partidos subiendo un fichero .csv

Tecnologías empleadas

Se han empleado las siguientes tecnologías:

Django

Se trata de un potente framework de código abierto para Python. Su principal meta es facilitar la creación de sitios web complejos poniendo un énfasis en la reutilización de componentes.

Django sigue el patrón de diseño MVT (Modelo, vista, template). Donde los modelos son una abstracción de los datos de la aplicación web. La vista encapsula toda la lógica responsable de controlar las peticiones de los usuarios y las templates renderizan la información que se presenta al usuario. Es un patrón similar al MVC intercambiando las vistas por templates y el controlador por la vista.

Python

Python es un lenguaje de alto nivel y multiparadigma. Creado en los años 90 ha crecido exponencialmente su popularidad al ser bastante potente y fácil de aprender e interpretar.

MySQL

Es un sistema de base de datos relacional propiedad de Oracle. Django no cuenta con soporte nativo para esta base de datos por lo que hemos empleado un plugin llamado “mysqlclient” de Pypi.

Javascript

Para añadir funcionalidad a las templates (escritas en HTML5) se ha empleado Javascript, lenguaje de programación interpretado y enfocado al Frontend.

Bootstrap

Es un framework orientado al Frontend. Facilita el diseño de la interfaz al incorporar multiples plantillas, menús de navegación y estilos.

ChartJS

Se trata de una libreria de Javascript que permite la construcción de gráficas dinámicas y responsive junto con HTML5.

GitHub

Se ha empleado GitHub y su aplicación para escritorio “GitHub Desktop” para alojar el proyecto. GitHub es una página para alojar proyectos colaborativos de programación empleando el sistema de control de versiones Git. En la bibliografía se incluye un enlace a la página de GitHub donde se encuentra alojado el proyecto.

Análisis

Diagrama E/R

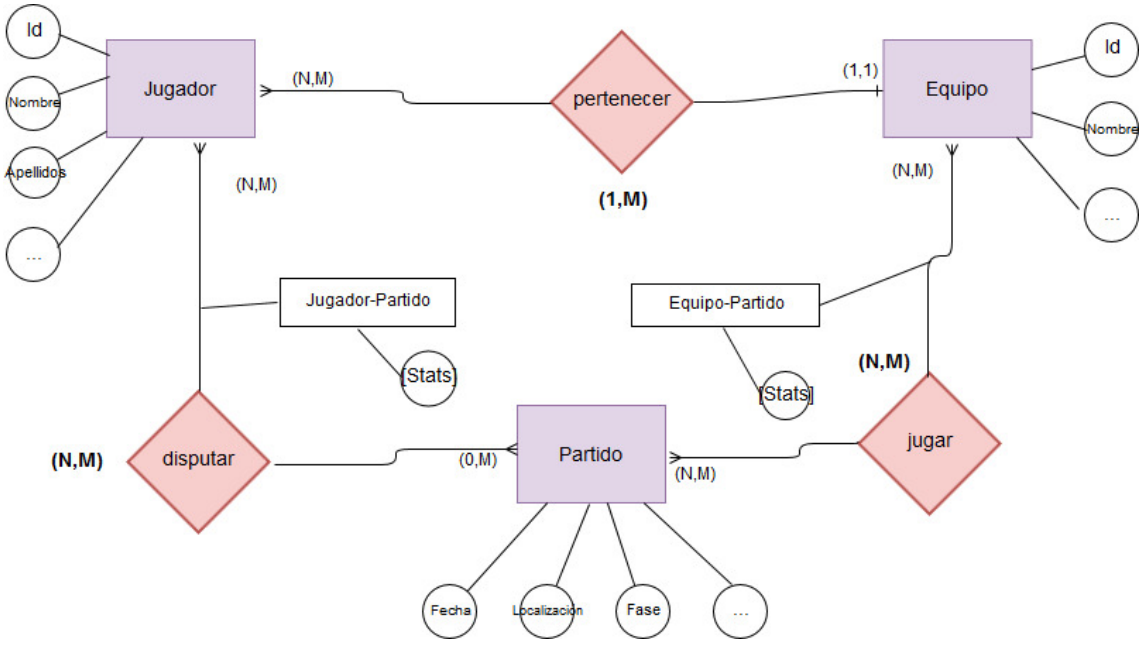
- Jugador: Los jugadores que participan en la competición, posee unos atributos básicos (nombre, apellidos, dorsal…), el equipo al que pertenecen (FK de Equipo) y todas las estadísticas acumuladas de la temporada (faltas que ha realizado, tiros libres que ha anotado...)

- Equipo: Los equipos de la competición, al igual que con el jugador, almacenamos una información básica sobre los mismos (nombre, sede…) y las estadísticas acumuladas para el equipo (puntos en la temporada, porcentaje de tiros de 3 del equipo…)

- Partido: La información de cada uno de los partidos que se juegan, posee dos FK de los equipos participantes, un listado de los jugadores que han sido convocados y otra información como la fecha o el lugar donde ha tenido lugar.

- Jugador-Partido: Tabla relacional que almacena el desempeño del jugador para un partido en concreto.

- Equipo-Partido: Tabla relacional que almacena el rendimiento del equipo durante un partido concreto.



Casos de Uso

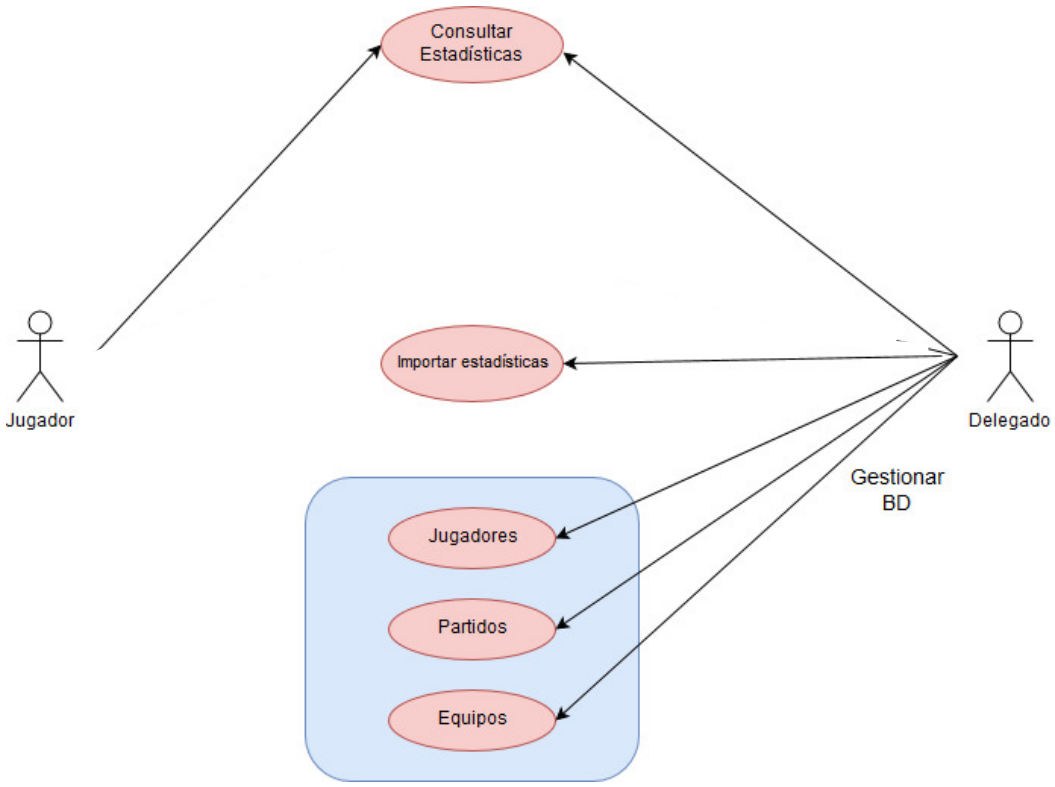
En el diagrama de casos de uso se indican las distintas funciones de la aplicación, se ha obviado el login para simplificar el diagrama. Existen los siguientes usos:

- Consultar estadísticas: Se engloban el consultar estadísticas de un partido concreto (un único jugador o el equipo en general) las estadísticas a lo largo de la temporada de un jugador o las del equipo.

- Exportar estadísticas: Se pueden exportar las estadísticas que estemos viendo a formato pdf.

- Importar estadísticas: Es la manera en la que se agrega información, el delegado agrega un nuevo partido (adjuntando un fichero .csv con la información de un partido). La aplicación extrae esta información y actualiza la base de datos.

- Gestionar BD: El delegado, como administrador, puede acceder y modificar directamente toda la información almacenada en la base de datos.

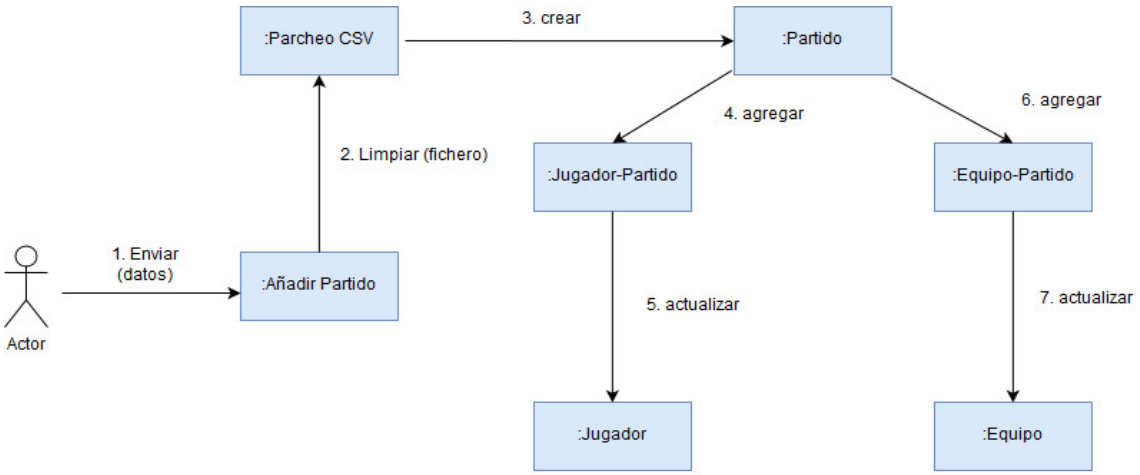


Consultar Estadísticas	
Descripción	Un usuario accede a las estadísticas de un jugador.
Actores	Jugadores, Delegados
Precondiciones	Estar logueado
Curso Normal	1. El usuario accede a un equipo 2. En el listado de jugadores selecciona uno 3. Se le muestra una página con las estadísticas del jugador

Importar Estadísticas	
Descripción	El delegado agrega un nuevo partido
Actores	Delegados
Precondiciones	Estar logueado
Curso Normal	1. El delegado accede al formulario para agregar partidos. 2. Indica la fecha, equipos y adjunta el CSV 3. La aplicación extrae información del CSV 4. Crea un partido nuevo. 5. Actualiza las estadísticas acumuladas de Jugador y Equipo
Alternativas	2. El fichero adjuntado no es un CSV 3. La aplicación devuelve un error y redirige al formulario.

Diagrama de colaboración

Se añade el diagrama de colaboración del importar estadísticas, al tratarse de la función principal de la aplicación.



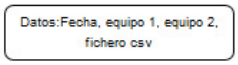
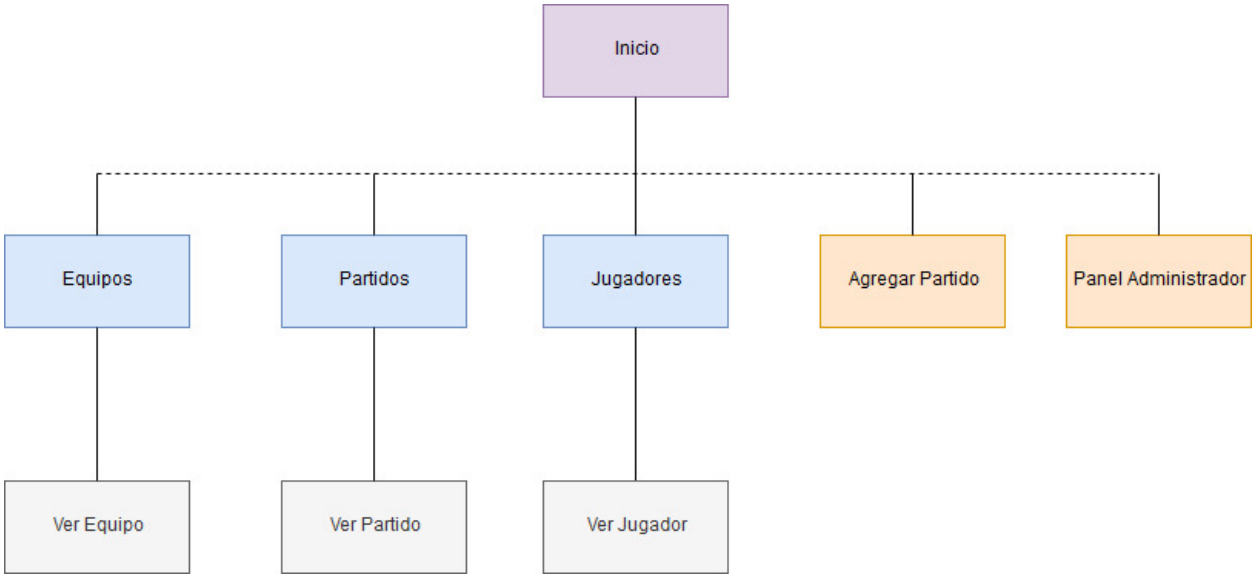


Diagrama navegación



Contamos con una página de Inicio desde la cual podemos acceder a todas las páginas. Equipos, Partidos y Jugadores muestra un listado con todos estos elementos, podemos acceder a una página con uno concreto. En caso de ser administrador podemos acceder a otras dos páginas: Agregar Partido y Panel de Administrador.

Seguridad

El estado de la aplicación actual en lo relativo a seguridad es el siguiente

Nombre del ataque	¿Lo evita?
Cross site Scripting (XSS)	Si
Cross site forgery (CSRF)	Si
SQL Injection	Si
Clickjacking protection	No

Como evitamos estos ataques:

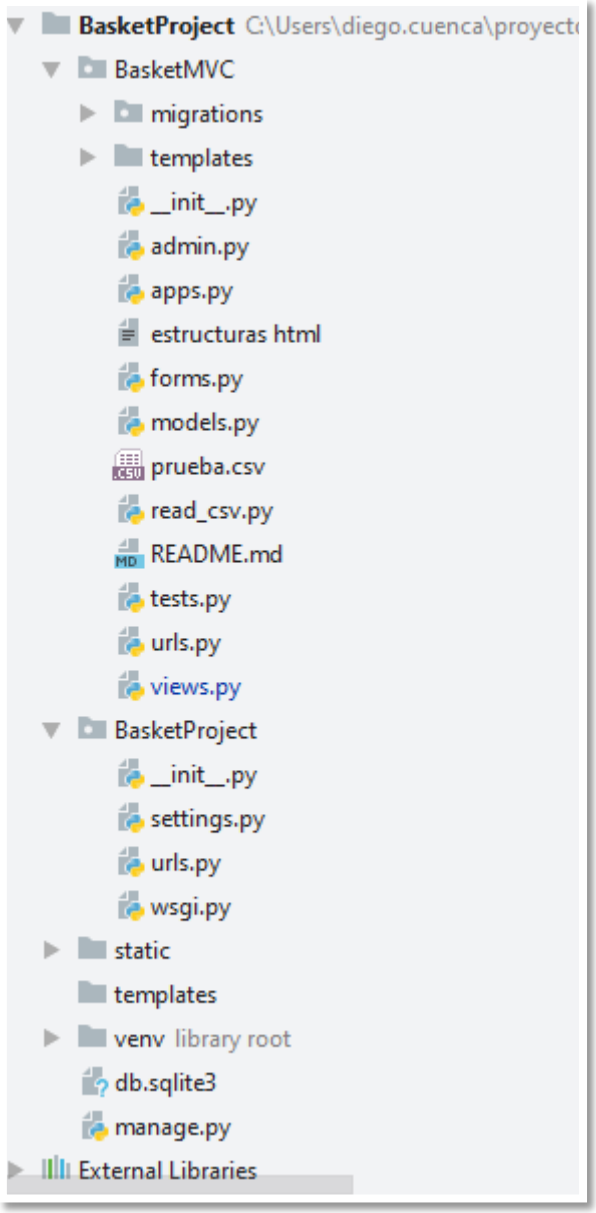
- ➔ Cross Site Scripting: Las templates de Django evitan por defecto la mayoría de ataques de XSS. Las templates tratan los caracteres más peligrosos de HTML lo cual protege contra casi todos estos ataques.

- ➔ Cross Site Forgery: Django posee funciones para evitar este tipo de ataques, en nuestros formularios basta con añadir un tag específico para evitar los ataques de esta índole.

- ➔ SQL Injection: Las querysets de Django protegen contra estos ataques usando "query parameterization", esto significa que la query es construida por separado de sus parámetros ya que estos pueden ser suministrados por el usuario y por tanto potencialmente maliciosos.

Diseño

Estructura de archivos



Estructura del proyecto

Nuestro proyecto se estructura de la siguiente manera. “BasketProject” contiene a todo nuestro proyecto Django.

De ella cuelgan:

- Static: en esta carpeta colocamos todos los archivos estáticos: css, imágenes, música… que usaremos en las templates (equivalente a las vistas en Django)
- Settings.py: indica toda la configuración del proyecto: donde buscar archivos estáticos, apps, plugins, seguridad, configuración de la base de datos.

```
# Database
# https://docs.djangoproject.com/en/2.2/ref/settings/#databases
#Aqui están las bases de datos
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': "pruebabasket",
        'USER': "root",
        'PASSWORD': "",
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

settings.py: configuración de la base de datos

- Urls.py: El enrutamiento de la aplicación. Indicamos las urls a las que dirigirá las peticiones.

En Django una aplicación se compone de distintas apps, en nuestro caso tenemos una única app llamada "BasketMVC". En su interior encontramos los siguientes archivos destacables:

- Migrations: en Django, al modificar los modelos deberemos actualizar la base de datos de manera manual. Primero introducimos el siguiente comando:

  Python manage.py makemigrations

  El cual generará archivos en la carpeta migrations con los cambios a realizar. Acto seguido realizamos la migración con:

  Python manage.py migrate

  Lo cual actualiza la estructura de las bases de datos (creando tablas, alterando estructura de las mismas…)
- Templates: En esta carpeta se incluyen todos los archivos .html de la aplicación (el equivalente a las vistas en Django)
- Admin.py: Hemos incluido un modulo propio de Django que es el panel de administrador, el cual nos permite editar de manera cómoda la base de datos y otras funciones que incluyamos.
- Models.py: indica los modelos de la aplicación.
- Forms.py: Django nos permite crear nuestros propios formularios de manera "similar" a los modelos, es en este archivo donde los declaramos y luego incluimos en las templates.
- Views.py: Las vistas en Django equivalen a los Controladores tradicionales, es aquí donde indicamos que realizar para cada petición recibida.

Implementación

Back-End

Modelos

Vamos a ir comentando como hemos construido nuestra aplicación. La parte más elemental fue la construcción de modelos. Un modelo de Django tiene el siguiente aspecto:

```
6   class equipo(models.Model):
7       id = models.AutoField(primary_key=True)
8       nombre = models.CharField('Nombre', max_length=30)
9       sede = models.CharField('Dirección', max_length=100)
10
11      # Estadísticas acumuladas
12
13      TC_P = models.IntegerField('TC%',default=0)
14      TC2 = models.IntegerField('TC2',default=0)
15      I_TC2 = models.IntegerField('2PI',default=0)
16      TC2_P = models.IntegerField('TC2%',default=0)
17
18      TC3 = models.IntegerField('TC3',default=0)
19      I_TC3 = models.IntegerField('3PI',default=0)
20      TC3_P = models.IntegerField('TC3%',default=0)
```

Ejemplo de modelo: equipo

Cada “tabla” corresponderá a una clase del tipo “modelo” (tipo interno de Django). A continuación definimos las columnas como atributos de la clase indicando el tipo (integer, char, date…) Indicando en el tipo “primary key” la convertimos en nuestra clave primaria, también indicamos entre comillas el nombre que recibirá la columna y otros atributos como un valor por defecto, si el campo no se puede dejar vacío.

```
37  class jugador(models.Model):
38      id = models.AutoField(primary_key=True)
39      nombre = models.CharField('Nombre', max_length=30, blank=False)
40      apellido1 = models.CharField('Primer Apellido', max_length=30, blank=False)
41      apellido2 = models.CharField('Segundo Apellido', max_length=30, blank=False)
42      fecha_nac = models.DateField(blank=False)
43      dorsal = models.CharField('Dorsal', max_length=3, blank=False, default="0")
44      equipo = models.ForeignKey(equipo, on_delete=models.PROTECT, related_name="Equipo", default=1)
```

modelo: jugador

Construimos la relación de esta manera (equipo y jugador poseen una Many to One) indicando la foreign key “equipo” en el modelo de jugador. Indicando de que modelo lo coge (equipo), que realizar cuando intentamos borrar y otros casos.

```
143     class Meta:
144         unique_together = ('id_jugador', 'id_partido')
```

Podemos también incorporar una clase dentro de la clase de un modelo llamado “Meta” que incluye algunas reglas, en esta captura vemos como en la tabla stats_jugador (stats del jugador para un partido) hemos definido que id_jugador e id_partido deben ser únicos (el par de ambos no puede repetirse en la tabla).

Vistas

```
10  ## Devuelve la página de inicio ##
11  def index(request):
12      return render(request, 'inicio.html')
13
14  ## Muestra varios jugadores ##
15  def Mostrar(request):
16      return render(request, 'jugador.html', {'jugadores': jugador.objects.all()})
17
18  ## Muestra un único jugador ##
19  def mostrar_jugador(request):
20      return render(request, 'jugadores.html', {'jugador': jugador.objects.filter(name='nombre')})
21
22  ## Muestra un equipo ##
23  def mostrar_equipo(request):
24      return render(request, 'equipo.html', {'equipos': equipo.objects.filter(name='nombre')})
25
26  ## Muestra listado equipos ##
27  def mostrar_Equipos(request):
28      return render(request, 'list_equipos.html', {'equipos': equipo.objects.all()})
29
30  ## Muestra un listado de partidos ##
31  def mostrar_Partidos(request):
32      return render(request, 'list_partidos.html', {'partidos': partido.objects.all()})
```

Vistas de nuestra aplicación

En nuestra vista indicamos las funciones que se ejecutarán con cada petición. Estas son unas de nuestras vistas más sencillas en las que recibimos la “request” todos los datos de la petición, incluidos campos de formularios o ficheros si es el caso, y devolver un renderizado donde indicamos información que añadir a la request (un listado de jugadores por ejemplo) además de indicar la plantilla a renderizar (jugadores.html por ejemplo).

Sin embargo, nuestra vista más interesante es la que gestiona el agregar partidos.

```
38  ## Agrega un partido ##
39  def formulario_partido(request):
40
41      ## Obtenemos la petición y mostramos el formulario ##
42      if request.method == 'POST':
43          form = PartidoForm(request.POST, request.FILES)
44          resultado = upload_partido(request.FILES['file'], request.POST['equipo1'], request.POST['equipo2'], request.POST['fase'])
45          return render(request, 'prueba.html', {'resultado': resultado})
46      else:
47          form = PartidoForm()
48          return render(request, 'formulario_partido.html', {'form': form})
49
```

Vista agregar partido

Si el método que hemos recibido es de tipo POST indica que el formulario ya ha sido completado. Llamamos a una función auxiliar llamada "upload_partido" con los datos del partido (nombres de los equipos, fase y el fichero de estadísticas).

En caso de que la petición fuese de otro tipo, cargamos nuestra template del formulario para que el usuario lo rellene.

En nuestra función upload_partido realizamos lo siguiente:

```
90          rows = []
91          file_data = file.read().decode("utf-8")
92          lines = file_data.split("\n")
93          for line in lines:
94              fields = line.split(",")
95              rows.append(fields)
```

Abrimos el fichero csv y lo guardamos en una lista separando por filas.

A continuación limpiamos estos datos (eliminando caracteres extraños, cambiando campos vacios en las estadísticas por ceros). Dividimos la lista en otras más pequeñas y manejables (una con la información de los jugadores de un equipo, otra con la información general del partido…)

```
200     ## Creamos y guardamos el objeto partido ##
201
202     p = partido.objects.create(
203         equipo1 = id1,
204         equipo2 = id2,
205         fecha = info_partido[1][5],
206         localizacion = id1.sede,
207         fase = fase1,
208         cuarto1 = info_puntos[2][1] + "-" + info_puntos[3][1],
209         cuarto2 = info_puntos[2][2] + "-" + info_puntos[3][2],
210         cuarto3 = info_puntos[2][3] + "-" + info_puntos[3][3],
211         cuarto4 = info_puntos[2][4] + "-" + info_puntos[3][4],
212         tanteo_final = info_puntos[2][5] + "-" + info_puntos[3][5],
213     )
214     p.save()
```

Creación de un partido

A continuación creamos un objeto de tipo partido y lo rellenamos con la información necesaria para luego guardarlo en la base de datos.

Realizamos lo mismo para la tabla de stats_jugador y stats_partido (información de un jugador para ese partido y lo mismo para un equipo).

Por último llamamos a los siguientes métodos auxiliares:

```
348     id_partido = partido.objects.get(equipo1=id1, equipo2=id2, fase=fase1)
349     ## Llamada a función que actualiza las estadísticas globales de jugador y equipo ##
350     actualizar_jugadores(local, visitante, id_partido)
351     actualizar_equipos(local, visitante, id_partido)
```

Estos dos métodos: actualizar_jugadores y actualizar_equipos lo que realizan es actualizar las tablas jugadores y equipos con la información nueva.

A continuación definimos las urls:

```python
urlpatterns = [
    ## Inicio ##
    path('index', views.index, name = 'index'),

    ## Ver jugador ##
    path('jugador', views.mostrar_jugador, name = 'Jugador'),

    ## Ver jugadores ##
    path('mostrar_jugadores', views.Mostrar, name = 'Mostrar'),

    ## Ver listado de equipos ##
    path('equipos', views.mostrar_Equipos, name='Equipos'),

    ## Ver un equipo ##
    path('equipo', views.mostrar_equipo, name='Equipo'),

    ## Ver un partido ##
    path('partido', views.mostrar_partido, name='Partido'),

    ## Ver un listado de partidos ##
    path('partidos', views.mostrar_Partidos, name='Partidos'),

    ## Agregar un partido ##
    path('upload', views.formulario_partido, name='upload_partido'),
```

Patrones de url

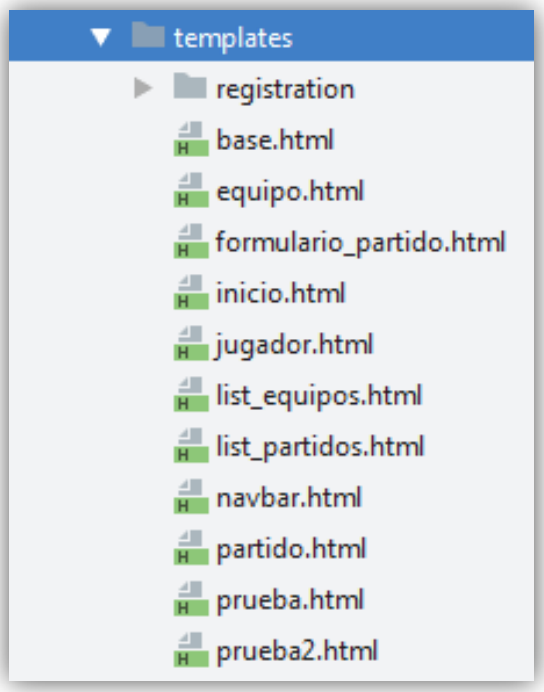
En nuestro fichero urls.py lo que indicamos son los patrones url que recibimos. La función path incorpora los siguientes argumentos:

Path('nombre',views.funcion,name="")

Donde el nombre es el nombre de la url a continuación la función a ejecutar de la vista para esa petición y luego el nombre que recibe.

Front-End

Nuestro front está enteramente definido en las templates de la aplicación. En ellas mezclamos lo llamado “template tags” con código HTML y Javascript. Las template tags son código python insertado en las templates. Existen unas por defecto pero podemos definir las nuestras propias en caso de ser necesario (para este proyecto no se ha creado tags nuevos)



Estas son algunas de nuestras templates, son destacables dos: base.html y navbar.html.

1. Base.html: En este archivo encontramos el bloque html sobre el que construimos nuestras demás plantillas.

2. Navbar.html: hemos incorporado una navbar de bootstrap al proyecto.

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="icon" href="">
    <title>Corazonistas</title>

    <link href="{% static 'node_modules/bootstrap/dist/css/bootstrap.min.css' %}" rel="stylesheet"/>
    <link href="{% static 'css/main.css' %}" rel="stylesheet"/>
</head>
<body>
  {% include "navbar.html" %}

    <div class="container">

      <div class="starter-template">
        {% block content %}{% endblock %}
      </div>

    </div>>
    <!-- Bootstrap core JavaScript -->
    <script src="{% static 'node_modules/jquery/dist/jquery.slim.min.js' %}" integrity="sha384-KJ3o2D
    <script>window.jQuery || document.write('<script src="{% static 'node_modules/jquery/dist/jquery.
```

base.html

En base.html empezamos con el template tag para cargar los archivos estáticos del proyecto y poder usarlos (imágenes o css).

Dentro del body incluimos nuestro fichero html con la barra de navegación. A continuación definimos un bloque de contenido, es en este bloque donde cargará el resto de las templates para cada caso.

```
{% extends 'base.html' %}
{% load static %}
{% block content %}
    <form method = "post" enctype="multipart/form-data" action = "{% url 'upload_partido' %} ">
        {% csrf_token %}
        {{ form }}
        <button type="submit"> Entrar</button>
    </form>
{% endblock %}
```

formulario_partido.html

A pesar de tener nuestro formulario como un modelo en el back, debemos encerrar el tag donde lo cargamos {{ form }} en unos tags <form> indicando el tipo de petición, el action al que

llamar. Incluimos también un tag csrf_token para incluir seguridad cross site request forgery.

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

{% block content %}
    <table class="table table-striped">
        <thead>
            <tr>
                <th scope="col">Nombre</th>
                <th scope="col">Sede</th>
                <th scope="col">Puntos a favor</th>
                <th scope="col">Puntos en contra</th></tr>
        </thead>
      <tbody>
        {% for equipo in equipos %}
        <tr>
          <td scope="row"> {{ equipo.nombre}}</td>
          <td >{{ equipo.sede}}</td>
            <td class = "verde">{{ equipo.puntos_favor }}</td>
          <td class = "rojo">{{ equipo.puntos_contra }}</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
{% endblock %}
```

list_equipos.html

Aquí podemos ver una template para mostrar el listado de equipos (se ha reducido el tamaño de la tabla en la captura para poder ver mejor el código). Empezamos extendiendo nuestra template base y cargando los archivos estáticos.

Con el tag “for” recorremos el resultado recibido (un listado de equipos en este caso) rellenando con esta información las tablas.

Base de Datos

Por último vamos a indicar como está estructura la base de datos, para observar que información guardamos.

Contamos con un total de cinco tablas propias más un conjunto de tablas de configuración y gestión que crea Dango. Estas últimas son:

- Auth_users: en esta tabla se almacenan los usuarios de la aplicación, entre otros datos guarda su username, la última vez que se loguearon, si es superuser (permiso de administrador) y la contraseña cifrada.
- Admin_log: guarda un log sobre las acciones realizadas en el panel de administrador como la fecha, el tipo de acción y a que tabla afectó.
- Django-migrations: almacena información sobre todas las migraciones realizadas incluida la fecha y el nombre de la migración.

Nuestras tablas propias son las siguientes:

Equipo	
Id	Id autogenerado
Nombre	Nombre del club
sede	Sede de sus partidos locales

TC_P		
TC2	I_TC2	TC2_P
TC3	I_TC3	TC3_P
TL	I_TL	TL_P
AS	TAP	
REBO	REBD	REBT
REC	DES	PER
F	FTO	

Jugador	
Id	Id autogenerado
Nombre	Nombre del jugador
Apellido1	Primer Apellido
Apellido2	Segundo Apellido
Fecha_nac	
Dorsal	Número del jugador
equipo	Id de su equipo

Partidos		
pts		
TC_P		
TC2	I_TC2	TC2_P
TC3	I_TC3	TC3_P
TL	I_TL	TL_P
AS	AS_p	
TAP	TAP_p	
REBO	REBD	REBT
	REB_p	
REC	REC_p	

DES	PER	
F	FTO	

Partido	
Id	Id autogenerado
Equipo1	Equipo local
Equipo2	Equipo Visitante
Fecha	Fecha del partido
Localización	Sede del equipo local
Fase	Fase de la competición
Cuarto1	Tanteo del primer cuarto
Cuarto2	Tanteo del segundo cuarto
Cuarto3	Tanteo del tercer cuarto
Cuarto4	Tanteo del cuarto final
Tiempos_extra	Tanteo de tiempos extra
Tanteo_final	Resultado final del partido

Stats_Jugador	
Id_jugador	Id de jugador
Id_partido	Id de partido
Pts	
TC2	**I_TC2**
TC3	**I_TC3**
TL	**I_TL**
AS	**TAP**
REBO	**REBD**
REBT	**REC**
DES	**F**
PER	**FTO**
TIEMPO	**EFI**

Stats_Equipo	
Id_equipo	Id de equipo
Id_partido	Id de partido
Pts	
TC2	**I_TC2**
TC3	**I_TC3**
TL	**I_TL**
AS	**TAP**
REBO	**REBD**
REBT	**REC**
DES	**F**
PER	**FTO**
EFI	
POS	**P_POS**
ZONA	**I_ZONA**
VENT	

Conclusiones y Propuestas

Propuestas de mejora

Las principales mejoras a realizar en la aplicación son las siguientes:

- Mejorar la seguridad para evitar todo tipo de ataques.
- Convertir las views de funciones a clases permitiendo reutilizar mucho código.
- Crear un programa separado que realice el parseo del .csv que recibimos de manera que la aplicación sea más ligera.
- Incorporar nuevas fuentes de datos como actas de partido o .csv de otras aplicaciones para tener mayor volumen de información.
- Convertir el back en una API REST, separando de manera efectiva el Back y el Front. Pudiendo construir nuestro Front con un framework como Angular.
- Poder exportar la información a formatos distintos, como hojas de cálculo propias o informes en pdf.
- Crear un panel de administrador propio y personalizado y no el propio de Django.

Conclusiones

La realización de este proyecto ha sido una gran experiencia personal y profesional. No solo me ha permitido realizar una aplicación que encuentro interesante (con utilidad real).

He aprendido bastante al realizar el proyecto, no solo las tecnologías que he empleado: Django y Python en especial. He adquirido mucha experiencia real trabajando con ellas. Al no usar tecnologías vistas en el ciclo o en las prácticas de empresa, he debido buscar documentación y formación por mi cuenta lo cual aunque ha sido un inconveniente y ha lastrado el desarrollo al no tener alguien de referencia a quien acudir, sí me ha enseñado a investigar por mi cuenta sin guía alguna.

De la misma manera, el hacerlo en solitario me ha ayudado a mejorar mi organización personal (usando por ejemplo el método Kanban). No ha resultado fácil realizar la fase de análisis dada la complejidad de analizar una aplicación tan sencilla como esta, pero entiendo la importancia que representa de cara a proyectos de mayor envergadura.

Anexo

Estadísticas: Referencias

A lo largo de este proyecto se emplean varios acrónimos propios de la estadística de baloncesto. Para no tener que ir explicándolos cada vez que aparecen, se ha relegado a esta sección la explicación sobre su significado.

Acrónimo	Significado
Pts	Puntos anotados
TC2	Tiros de 2 anotados
I_TC2	Intentos de tiros de 2
TC2_P	% de tiros de 2
TC3	Tiros de 3 anotados
I_TC3	Intentos de tiros de 3
TC3_P	% de tiros de 3
TL	Tiros libres anotados
I_TL	Intentos de tiros libres
TL_P	% de tiros libres
AS	Asistencias realizadas
AS_p	Asistencias por partido
TAP	Tapones realizados
TAP_p	Tapones por partido
REBO	Rebotes ofensivos
REBD	Rebotes defensivos
REBT	Rebotes totales
REB_p	Rebotes por partido
REC	Recuperaciones
REC_p	Recuperaciones por partido
DES	Desvios realizados
TIEMPO	Tiempo en pista
EFI	Formula de eficiencia: estadísticas positivas (tiros anotados + faltas recibidas…) – estadísticas negativas (pérdidas, faltas realizadas…)
POS	Posesiones
P_poS	Puntos por posesiones
ZONA	Tiros anotados en zona
I_ZONA	Intentos de tiros en zona
VENT	Máxima ventaja

Bibliografía

Estadísticas Baloncesto

- Delegado Equipo Corazonistas Nacional Femenino: Tomás Cuenca Carrasco

Análisis

- Libro Entornos de Desarrollo (Editorial Anaya)

Backend

- Documentación Django
  - https://docs.djangoproject.com/en/2.2/
- Curso Python 3 desde 0
  - https://openwebinars.net/
- Curso Django
  - https://openwebinars.net/
- Python Data Science Handbook (Editorial O'Reilly) "Jake VanderPlas"
- Blog sobre Django
  - https://simpleisbetterthancomplex.com/
- StackOverFlow: Resolución de dudas y problemas concretos
  - https://stackoverflow.com/

Frontend

- Documentación Bootstrap
  - https://getbootstrap.com/docs/4.3/getting-started/introduction/