CFGX CICLO

Proyecto Final de Ciclo

WLOG

Autor: Alberto F Morgado Linares

Tutor: Sergio Malo Molina

Fecha de entrega:

Convocatoria: Semestre – 2020

Enlace hosting:

https://github.com/elotrofausto/WLOG

Sumario

1 Introducción

A continuación exponemos una breve introducción al **proyecto WLOG**: La motivaión para llevarlo a cabo, sus puntos más importantes y los objetivos propuestos.

1.1 Motivación

La motivación que derivó en la creación de **WLOG** fue la necesidad de disponer de un **sistema de blogging** propio que nos permitiese adaptarlo a nuestras necesidades concretas, así como tener el completo control de la interfaz de usuario.

Existen en el mercado plataformas de blogging altamente conocidas como **Blogger** o **Wordpress**. Sin embargo, estos gestores de contenido están orientados a un público **sin conocimientos técnicos** y no ofrecen todo el abanico de customizaciones que podemos lograr con un sistema propio.

Para lograr, por ejemplo, adaptar la interfaz de usuario a nuestro gusto, usualmente debemos recurrir a la instalación de Temas prefabricados (gratuitos o de pago), que no nos ofrecen la misma **flexibilidad** que la posibilidad de construir nuestro sitio desde cero.

WLOG está orientado a aquellas personas que buscan crear un sitio web con un blog e información estática, y poseen los conocimientos necesarios para crear su propia plantilla con HTML y el lenguaje de plantillas de Django.

La idea surgió ante mi propia necesidad de crear un sitio web para mis libros, habiendo trabajado anteriormente con Blogger y Wordpress y encontrándome siempre con los problemas mencionados anteriormente.

1.2 Abstract

WLOG is a **blogging system** focused on **customization**. It is a system created **by developers and for developers**. It's main goal is to offer a basic configuration for people who want to create blogs and have the technical skils to work on the templating themselves.

WLOG will be a **Django project** that can be copied and modified to suit the user needs. The project will come with pre-configured models like Post, Page and Product, as well as with an admin panel to be able to perform **CRUD** operations using a friendly user interface.

So, anybody can just pull **WLOG** from **Github**, create or customize a basic template for the site, and start adding content. You'll have a system that can be deployed almost everywhere (**Python** works in a lot of environments nowadays). Also, it is **lightweight and intended for developers**, or companies who can hire them. It is the perfect solution if you are looking at creating your enterprise site in a few steps, with the ability of posting new content and modify the appearance yourself.

The **WLOG** project comes with an **included Node.js API** that makes it easy and secure to post content from anywhere. By calling this API, we can perform basic CRUD operations without having to use the Django Admin panel. This is intended to facilitate **future integrations** with the system.

To sum up, **WLOG is a free, open-sorce blogging system created for developers**.

1.3 Objetivos propuestos (generales y específicos)

Los objetivos propuestos son los siguientes:

- Crear un **sistema de blogging** gratuito y open-source para desarrolladores.
- Colgar el sistema en **Github** para que cualquiera lo pueda descargar.
- El sistema incluirá:
    - La definición de los **modelos** necesarios para su funcionamiento
    - Una **plantilla** básica
    - Un panel de **administración** para crear contenido
    - Una **API** externa para futuras integraciones
- Crear una **página web** de ejemplo utilizando el sistema (albertofausto.com)

2 Metodología utilizada

La metodología utilizada para el desarrollo del proyecto es una **metodología ágil**, y más en concreto la metodología **Scrum**.

Todas las estructuras ágiles se basan en el *Manifiesto Ágil*, realizado por varios autores que establecieron una serie de pautas o principios del software ágil.

A modo de resumen, algunas de las valoraciones del manifiesto son:

- **Individuos e interacciones** sobre procesos y herramientas
- **Software funcionando** sobre documentación extensiva
- **Colaboración con el cliente** sobre negociación contractual
- **Respuesta ante el cambio** sobre seguir un plan

El proyecto será desgranado en pequeñas tareas, divididas en las etapas de **análisis, desarrollo y testing**. Durante el desarrollo del proyecto, se llevarán a cabo diferentes interacciones del proceso o *Sprints*, que se corresponden con "entregas" parciales del producto final. Nótese que entrecomillamos entregas puesto que no estamos facilitando el producto a un cliente final en cada iteración, sino a nosotros mismos.

Esta metodología nos permite abordar proyectos con cierta **flexibilidad**, tratando de minimizar el acarreamiento de errores a lo largo del desarrollo, puesto que se llevan a cabo revisiones regulares del mismo para **asegurar la calidad** y los cumplimientos de objetivos.

Scrum, en definitiva, es una metodología orientada a la innovación, flexibilidad, competitividad y productividad, que pensamos será ideal para la realización de nuestro proyecto.

3 Tecnologías y herramientas utilizadas en el proyecto

A continuación detallaremos las tecnologías y herramientas utilizadas en el proyecto, así como los motivos para escogerlas.

- Tecnologías
  - **Python + Django**
    - **Pytho**n:

      A la hora de escoger un lenguaje de programación del lado del servidor, tuvimos en cuenta varias opciones, decantándonos finalmente por una combinación de **Python y el framework Django**.

      Python es un **lenguaje de programación que nos permite desarrollar rápidamente** (con menos líneas de código que otros lenguajes). Además, es muy eficiente, capaz de manejar grandes cantidades de datos (**big data**) y su uso está cada vez más extendido, por lo que podemos encontrar infinidad de recursos online como tutoriales o dudas resueltas.

    - **Django**:

      Django es un **Web framework para Python** de alto nivel que incentiva el desarrollo rápido y los diseños limpios y funcionales. Ha sido construido por desarrolladores expertos con el objetivo de facilitar el desarrollo Web: “Para que puedas **enfocarte en tu código** en lugar de en reinventar la rueda”, según su propia web. Es **gratuito y open source**.

      Algunas de las ventajas de Django son:

      - Rapidez: Diseñado para ayudar a los desarrolladores a ir del concepto a la finalización del proyecto lo más rápido posible.
      - Seguridad: Django se toma muy en serio la seguridad y ayuda a los desarrolladores a evitar errores comunes.
      - Escalabilidad: Algunos de los sitios con más tráfico de Internet confían en las capacidades de Django para escalar de forma rápida y flexible.

- **HTML5 + Django Templates**
  - HTML5:

    Sigue siendo la piedra angular de todo desarrollo web. Nos permite estructurar la información y es un estándar.
  - Django Templates

    En combinación con **HTML5**, las plantillas de Django nos permiten generar contenido dinámico que se cargará dependiendo de las peticiones concretas de los usuarios.
- **Node.JS**
  - Finalmente, Node.JS será utilizado para al realización de una API que permitirá operaciones CRUD en la base de datos de nuestra aplicación Web.

    Node es un entorno de ejecución para **Javascript** construido con el motor Javascript V8 de Chrome.

    Node ha demostrado ser verdaderamente eficiente para la construcción de **REST APIs**, manejando las peticiones de forma asíncrona y siendo capaz, por lo tanto, de atender una gran cantidad de estas al mismo tiempo.
  - A pesar de que **Django** dispone de su propio **REST framework**, decidimos utilizar **Node** para la construcción de la API por varias razones:
    - **Desacoplar la API de la aplicación Django**, ya que nos parecía mejor práctica que incluir la API en las misma aplicación que nuestra interfaz de usuario.
    - **Conocimiento del entorno**. Al haber trabajado previamente con REST API de Node, nos sería más sencillo desarrollar la API para este proyecto.

- Herramientas
  - **Visual Studio Code** (Microsoft)
    - **Visual Studio Code** es un editor de código definido para construir y hacer debug de modernas aplicaciones web y en la nube. Es gratuito y

open source, además de funcionar en sistemas **Windows, Linux y Mac**.

Escogimos VS Code para el proyecto porque nos permite trabajar con todas las herramientas y frameworks descritos anteriormente, en un solo editor de código. Incorpora una herramienta llamada *IntelliSense* que tiene soporte para **JavaScript, TrueScript, JSON, HTML, CSS**, etcétera. Además, dispone de infinidad de plugins que nos facilitarán el desarrollo. Algunos de los plugins utilizados en el proyecto son:

- Python
  - Ofrece precicción y autocompletado de código para el lenguaje de programación Python.
- JSLint
  - Detecta y marca partes del código JavaScripts mejorables o que no cumplen con las guías de la versión de JS que estamos utilizando.

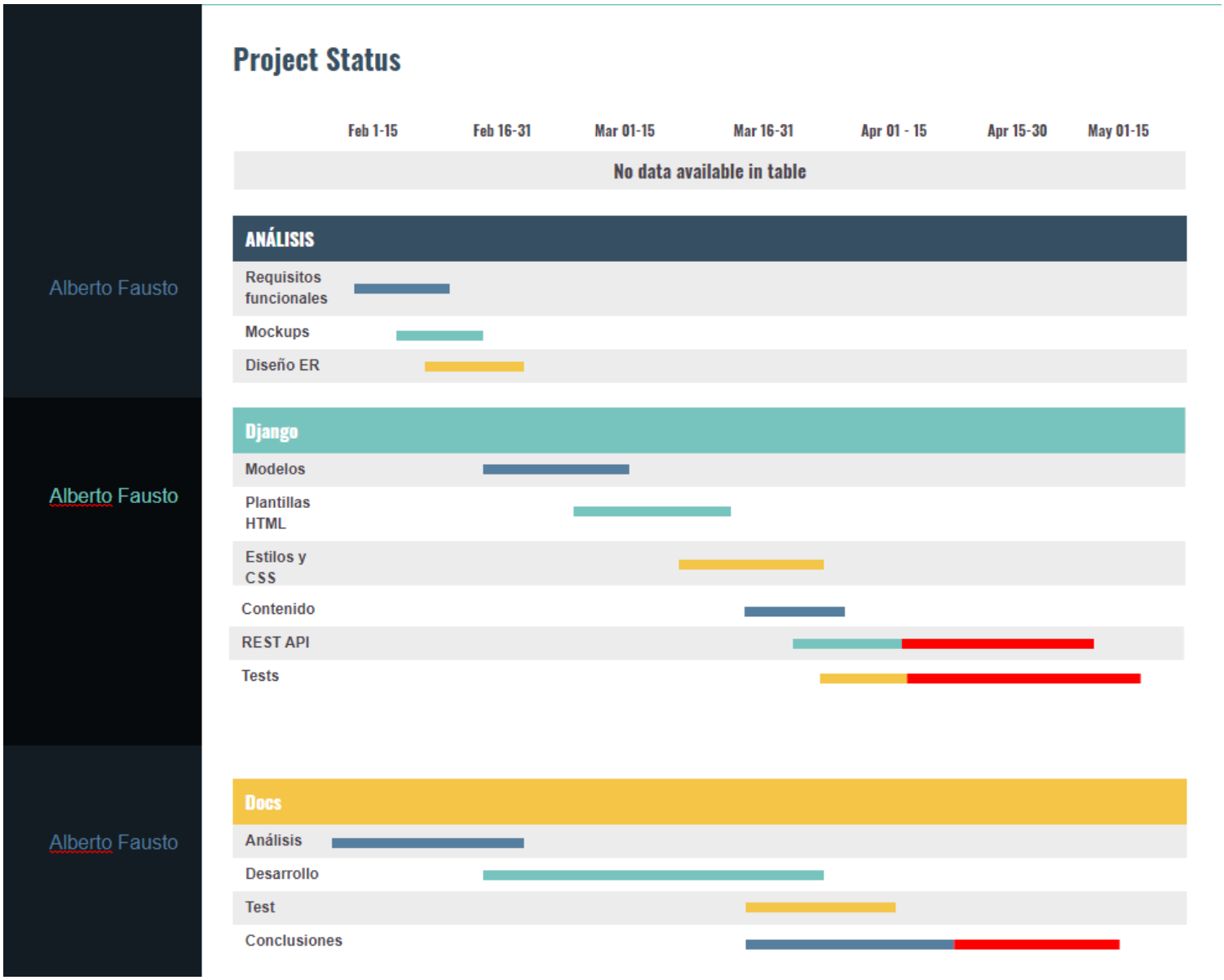
- **Github**
  - Para el control de versiones se opta por la utilización de Github, ya que el sistema ya nos era familiar y habíamos trabajado con él anteriormente. Además, desde la adquisición de Github por parte de **Microsoft** en 2018, se han ampliado algunas de las funciones gratuitas, como la capacidad de crear un número ilimitado de repositorios privados.

    WLOG está pensado para que cualquiera lo pueda descargar y utilizar, por lo que ponerlo disponible en la plataforma **Git** más conocida nos pareció una buena idea, por encima de otras opciones como **GitLab**.

4 Estimación de recursos y planificación

A priori, la **planificación** del proyecto ya suele enfrentarse a contratiempos propios del desarrollo o una mala planificación. En este caso, dada la situación especial vivida a nivel mundial en los primeros meses del año 2020 por el problema sanitario del Covid19, los plazos de entrega del proyecto fueron ampliados en varias ocasiones, dando a lugar a ajustes de nuestro **diagrama de Gantt**.

Podemos observar el tiempo estimado por secciones. Destacan las barras en color rojo allí donde el proyecto se ha extendido más de lo previsto y no se han cumplido con los plazos estimados en un primer momento.



5 Desarrollo del proyecto

Apoyándonos en el diagrama de Gantt previo, podemos destacar diferentes etapas y/o secciones dentro del desarrollo del proyecto **WLOG**. Son las siguientes:

- Análisis

  - Requisitos funcionales:Los requisitos funcionales básicos pueden verse en el siguiente diagrama de casos de uso.

    El Administrador debe de ser capaz de realizar operaciones CRUD (Create Read Update Delete) de los siguientes elementos utilizando la aplicación:

    - **Posts**: Son las publicaciones periódicas que conforman típicamente un blog. El administrador debe poder crearlas, modificarlas y eliminarlas de forma sencilla, ya que son el núcleo de cualquier sistema de **blogging**.

    - **Páginas**: Son apartados estáticos de la web, a los cuales el Admin debe ser capaz de crear, actualizar y eliminar al igual que los posts, pero cuyo contenido se supone que cambia menos en el tiempo.

    - **Productos**: Hoy en día no es raro que cualquier empresa o individual venda un producto o servicio a través de su web. Es por eso que hemos querido reflejarlos en el sistema de **WLOG**.
      El **Administrador** debe de ser capaz de crear productos, de una forma igual de sencilla que las publicaciones. Estos productos pueden tener un código de barras asociado, fecha de publicación y una descripción, así como una imagen.
      Los posts tendrán la posibilidad de tener un producto asociado, para que se muestre de forma dinámica en algún lugar de la página cuando un lector esté leyendo dicho contenido.

El usuario, por su parte, será el "consumidor" de todo el contenido que el administrador ponga a su disposición, teniendo permiso de lectura sobre todos los elementos (Páginas, Posts y Productos) sin necesidad de estar identificado en el sistema.

- Requisitos no funcionales
    - Seguridad:
        - El usuario administrador debe identificarse mediante **usuario y contraseña** para poder realizar operaciones sobre la Base de Datos.
    - Eficiencia:
        - El sistema debe ser capaz de procesar **multitud de peticiones simultáneas**.
        - Las **páginas** se generarán de forma **dinámica** de acuerdo a la petición de los clientes. El tiempo de renderizado no deberá superar los 3 segundos, ya que algunos estudios indican que el usuario abandona tras este tiempo de inactividad.
    - Usabilidad:
        - El usuario final debe ser capaz de navegar por el sitio web de forma cómoda e intuitiva, encontrando la información que busca sin dificultad y siendo capaz de acceder al sitio desde dispositivos móviles o un ordenador.
        - La información se presentará de forma clara y ordenada. Los enlaces serán claros y descriptivos y llevarán al lugar de destino correcto.
- Diagramas ER

  El proyecto no era demasiado complejo en cuanto a entidades y sus relaciones, pero aún así nos pareció buena idea realizar el diagrama ER.

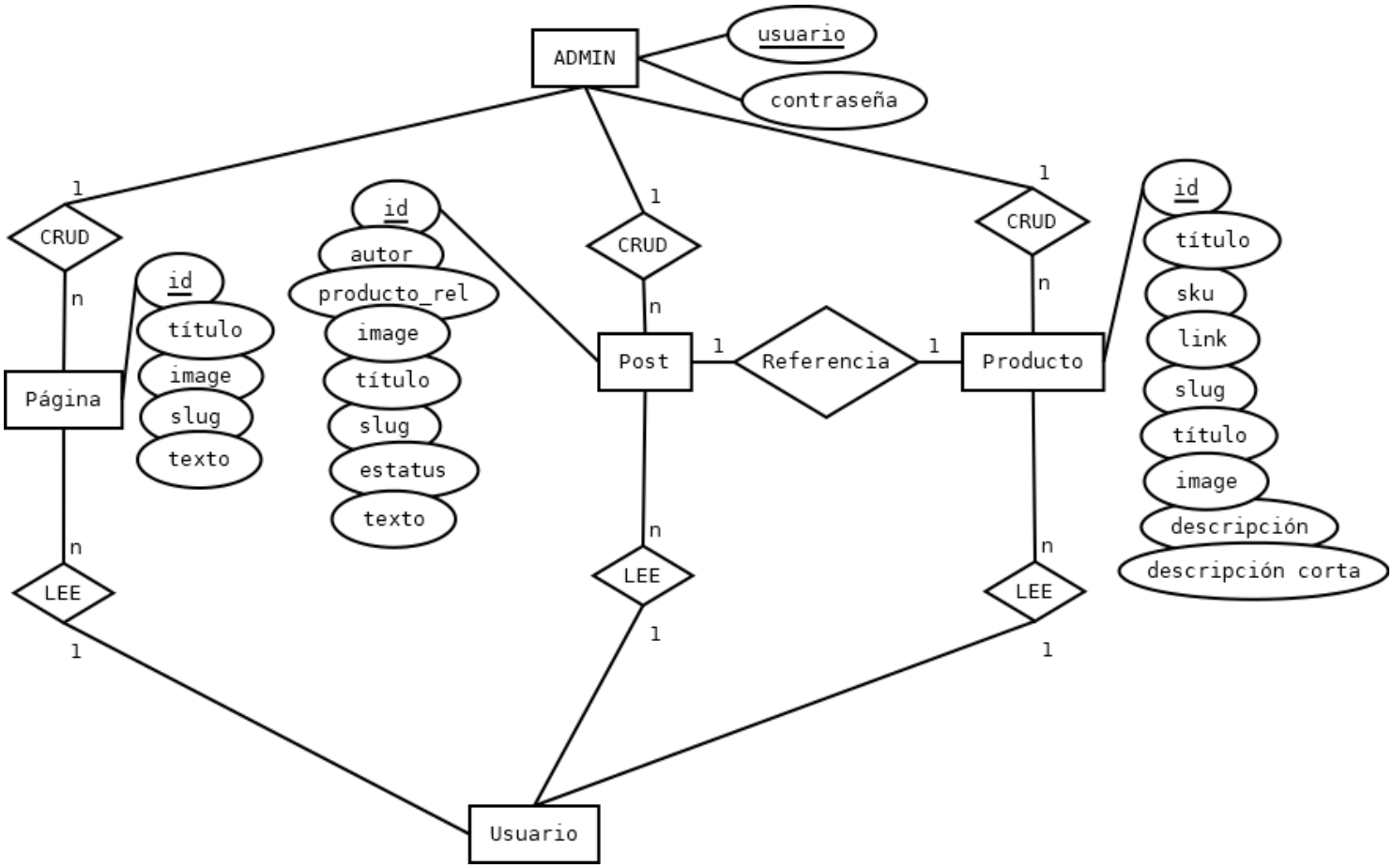


Imagen 1. Diagrama Entidad Relación

- Diseño
  - Diagramas de clases

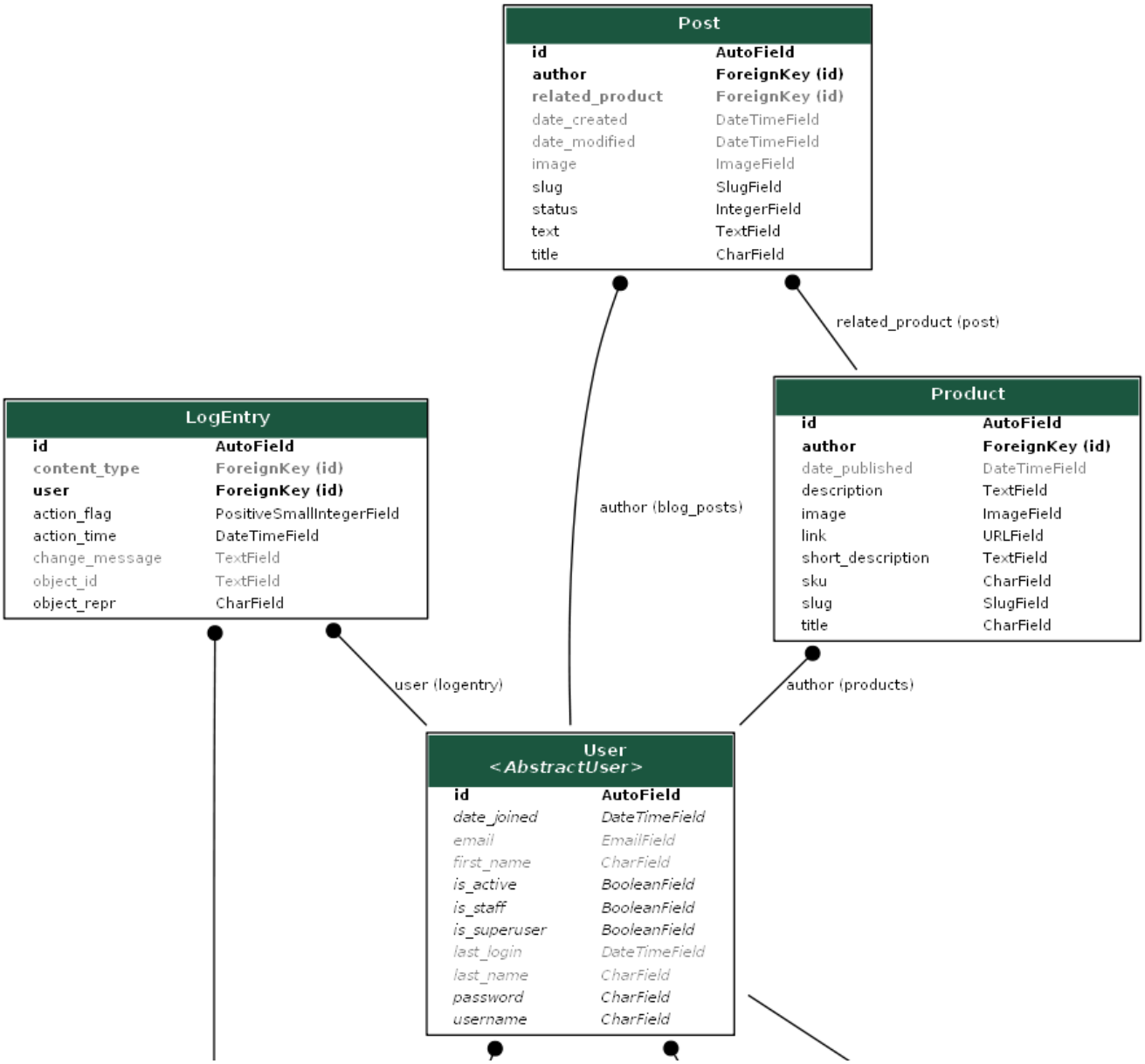


Imagen 2. Diagrama de Clases. Detalle

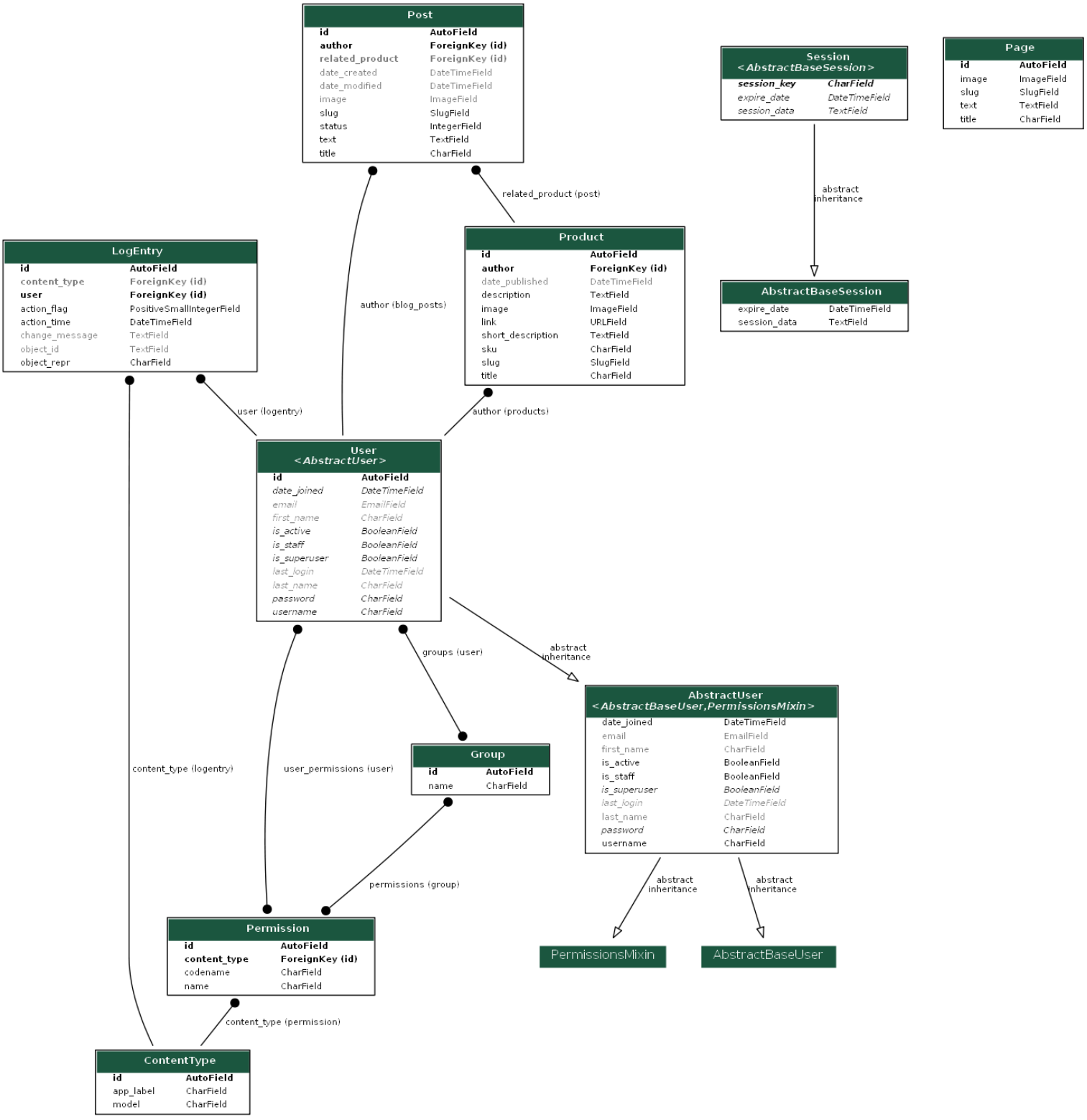


Imagen 3. Diagrama de clases completo.

- Casos de uso

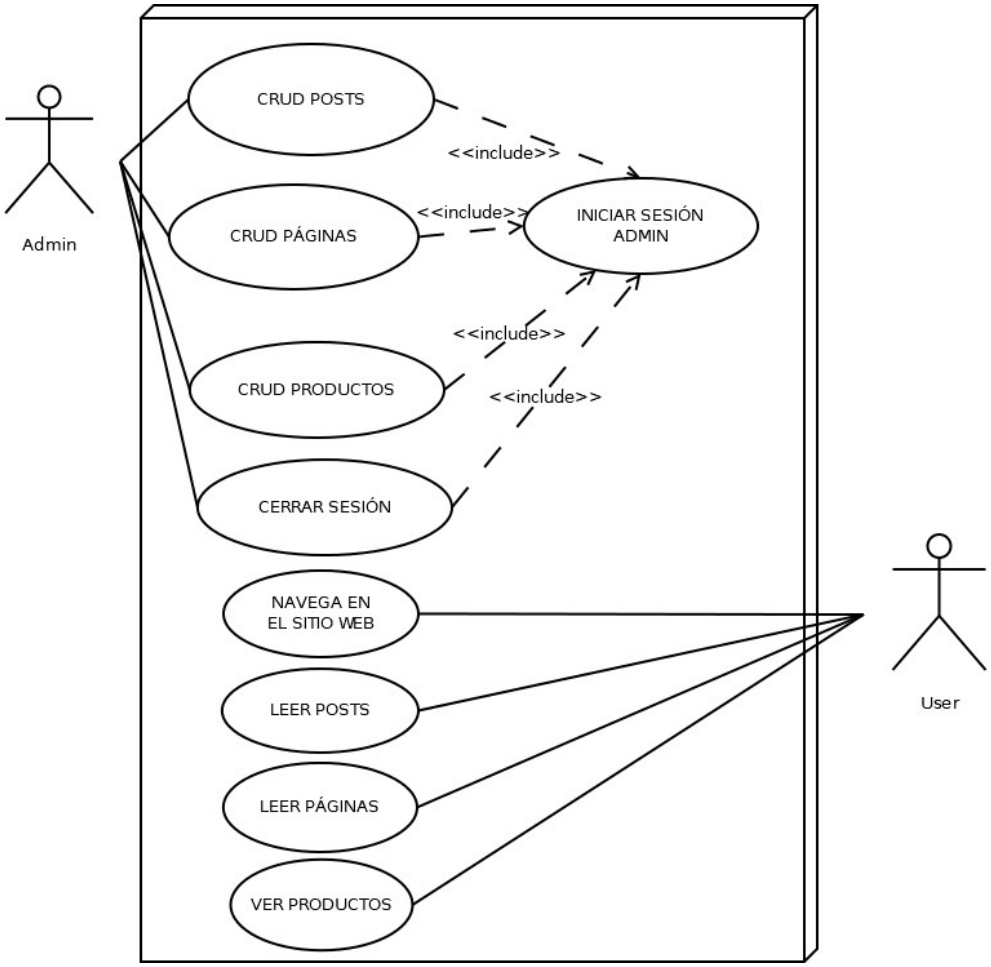


Imagen 4. Diagrama de Casos de Uso

- Desarrollo

  - **Postgres**

    - Cuando creamos una aplicación **Django**, podemos elegir entre multitud de sistemas de Base de Datos. Escogimos **Postgres** para el proyecto por su robustez y flexibilidad, además de que dispone de librerías para **Node.js** que nos facilitarán la creación de la **API externa** para realizar operaciones CRUD de las publicaciones de **WLOG**.

La instalación para el sistema **WLOG** es muy sencilla, ya que Django es dispone de **ORM** (Object Relational Mapping) y generará autmaticamente las tablas y los campos a partir de nuestras definiciones de los modelos. Por tanto, solo debemos tener instalado **Postgres** en la máquina que queramos usar para alojar la base de datos, y obtener la cadena de conexión de la misma.

- **Django**

  Cuando creamos nuestro **proyecto de Django**, este presentaba la siguiente estructura:

```
wlog/
    manage.py
    wlog/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.p
```

  A esto hay que añadir la carpeta blog/ que albergará nuestra aplicación de **blogging**, junto con otras para los archivos estáticos y las plantillas de Django. El resultado final, sin entrar en el detalle de las carpetas, sería el siguiente:

```
wlog/
    manage.py
    wlog/
    blog/
    templates/
    media/
```

Podríamos decir que “wlog/” es la carpeta de proyecto. En ella encontramos el fichero manage.py que, como su nombre indica, nos servirá para realizar operaciones para gestionar el mismo. Por ejemplo, para iniciar el servidor deberemos escribir “py manage.py runserver”.

Seguidamente encontramos una carpeta con el mismo nombre que el proyecto “wlog”. Esta es la **carpeta de la aplicación principal** del proyecto. Podemos crear nuevas aplicaciones para nuestro proyecto, que estarán situadas en el mismo nivel que la aplicación principal. Así, por ejemplo, en nuestro proyecto tenemos la aplicación principal “wlog” y la específica “blog”, que es donde realmente suceden la mayoría de las cosas.

Debemos, por tanto, hacer una distinción entre las dos aplicaciones que componen nuestro proyecto:

- wlog (Aplicación principal del proyecto)
    - Urls → wlog/urls.py

        En este fichero definiremos las **rutas del proyecto** a nivel raíz, esto es, redireccionaremos a las diferentes aplicaciones del proyecto Django, asignándoles una o varias rutas a cada una de ellas.

        Es también muy importante definir la **ruta para la aplicación admin**, que viene incluida en el sistema Django y nos proporciona un panel de administración para nuestra aplicación web.

    - Settings → wlog/settings.py

        En este fichero estableceremos todas las **configuraciones más importantes** del mismo. Cuando creamos el proyecto Django, ya viene con algunas opciones por defecto, que podremos modificar y ampliar a nuestro gusto. Algunos de los cambios que realizamos en este fichero son:

- Establecer los **directorios** base de la aplicación, así como los directorios para los archivos estáticos y las plantillas de Django.
- Indicaremos las **aplicaciones instaladas** en nuestro proyecto Django (ya sean propias o del Framework)
- Estableceremos si el sistema está en modo **Debug** o no. Para esta presentacón lo mantendremos activado, ya qye Django no sirve archivos estáticos en modo producción. Para subsanar esto, deberíamos servir los archivos con nuestro propio servidor Nginx o IIS, y configurar la ruta en nuestro settings.py
- Establecemos el **middleware** utilizado.
- Configurar las **bases de datos**. En este caso, añadiendo la conexión a nuestra BD Postgres previamente instalada, y especificando el ENGINE que utilizará Django para manejarla (psycopg2).
- La MEDIA_ROOT para los **archivos multimedia**. Esta se diferencia de la carpeta para los archivos estáticos en que puede ser utilizada por Django para que los usuarios de la aplicación suban sus propios archivos a través de la misma.

- blog (Aplicación que implementa la funcionalidad de nuestro blog)
  - Panel de administración de Django → admin.py

    En este fichero podemos modificar a nuestro gusto las **vistas del panel de administración** de Django. En este caso, hemos seleccionado los campos que se mostrarán en las vistas de lista para los Productos, las Páginas y los Post.

En resumen, el panel de Administración de Django es una herramienta que nos facilitará sobremanera la creación de contenido para nuestras aplicaciones web. Podemos escoger qué modelos estarán disponibles en el panel, así como personalizar sus vistas.

- Modelos → blog/models.py
  - Suele ser uno de los ficheros más importantes de cualquier aplicación Django. En él, **definiremos los modelos (clases)** de la aplicación.

    Aquí es donde se aprecia la importancia de **Django ORM**, ya que con un par de sencillos comandos y el fichero manage.py, Django creará automáticamente la base de datos definida, haciendo uso de la conexión definida en el fichero settings.py

    A lo largo del desarrollo, podremos hacer **cambios en los modelos** y estos se verán reflejados en la base de datos. Solamente deberemos tener en cuenta las posibles inconsistencias con claves únicas, ajenas… y ejecutar los siguientes comandos:

    py manage.py makemigrations → Prepara los cambios

    py manage.py migrate → Para aplicar los cambios

- Vistas → blog/views.py

  En este fichero **definiremos todas las vistas de la aplicación**. Cada aplicación Django incluida en el proyecto tendrá su propio fichero.

  En particular, hemos definido la lógica de todas las vistas de nuestro blog en este fichero. En Django se pueden definir las vistas mediante funciones o clases. Se optó por las clases, ya que nos ofrecen una forma minimalista de trabajar con los objetos de nuestra base de datos.

Las vistas hacen uso de plantillas dinámicas para renderizar el contenido solicitado en cada petición. Ahondaremos en estas plantillas un poco más adelante en esta misma sección, cuando expliquemos las otras carpetas del proyecto.

- Urls → blog/urls.py

  En este fichero configuraremos las **rutas específicas de la aplicación blog**. Como hemos explicado anteriormente, tenemos otro fichero urls.py en la aplicación principal, en el cual estableceremos rutas al resto de aplicaciones.

  Básicamente, la aplicación principal configura rutas a las sub-aplicaciones (en este claso blog) y estas dirigen a las **diferentes vistas** de las mismas.

  En este fichero, por tanto, hemos configurado las **rutas a todas las vistas disponibles en nuestra aplicación de blogging**, tanto las genéricas (mostrar todos los post, todos los productos…) como las concretas (detalle de post, detalle de producto).

  Además, el proyecto se apoya en otras carpetas que contienen ficheros estáticos y otros elementos como las plantillas Django.

- CSS, Bootstrap e imágenes → blog/static

  En esta carpeta **alojaremos todos los archivos estáticos** de la aplicación, ya se trate de imágenes, CSS o Javascript. Es importante que configuremos correctamente la ruta a esta carpeta en el fichero de configuración. Así mismo, hay que tener en cuenta que en producción debemos servir los ficheros de esta carpeta con un servidor de nuestra elección (como Nginx en Linux o IIS en Windows), ya que Django no lo hará por motivos de seguridad.

  Hemos aplicado los estilos de nuestro sitio web en un solo fichero styles.css que es incluido en la plantilla HTML, de la que todas las demás heredan (explicamos esto en más profundidad en el siguiente apartado).

Se ha utilizado **Bootstrap** para facilitar la creación de los estilos de la plantilla base. Es una librería muy popular que nos permitirá diseñar de forma rápida sitios web *responsive* con la filosofía **mobile-first** (dispositivos móviles primero).

- Plantillas Django (HTML + Template tags) → templates/

  Las **plantillas** son otro de los elementos más importantes de **Django Framework**, y es que mediante ellas podemos hacer que el contenido de nuestras aplicaciones se cree de forma dinámica atendiendo a las peticiones de los clientes.

  Cuando utilizamos Django, podemos escoger nuestro propio lenguaje de plantillas (templating language) como *Jinja2*. No obstante, y a no ser que estemos realmente familiarizados con uno de estos lenguajes, es recomendable utilizar el **lenguaje de plantillas de Django**.

  Django template language ofrece una sintaxis intuitiva para crear contenido basado en texto (HTM, XML, CSV, etc.). Las plantillas que creamos pueden contener **variables**, que serán reemplazadas cuando la plantilla sea evaluada, y **etiquetas** que controlen la lógica de la plantilla. Esto nos permite utilizar tomas de decisiones dentro de las propias plantillas, así como iterar sobre los datos, por ejemplo. Además, podemos ofrecer información precisa a cada usuario cargando los valores adecuados en las variables.

  Otro de los aspectos fundamentales de las plantillas de Django es que nos permiten **extender otras plantillas**. Por ejemplo, podemos crear una plantilla base.html con la estructura común de nuestro sitio web.. En dicha plantilla definiremos bloques de contenido que serán rellenados desde otras plantillas que extiendan a esta.

También podemos incrustar otras plantillas. Esto es muy útil, por ejemplo, para crear una barra de navegación lateral e incluirla en aquellas plantillas que deseemos con solamente una línea de código.

Las plantillas son **utilizadas por las vistas** en el archivo views.py, que renderizará una u otra pasándole los datos oportunos.

Para este proyecto, hemos creado las siguientes plantillas:

- base.html → Estructura del sitio. La gran mayoría de las otras plantillas la extenderán.
- index.html → **Listado de los Posts** de nuestro blog que hará las funciones de página principal de nuestro sitio.
- page_detail.html → **Detalle de página**. Mostrará el contenido de la página que el cliente solicite.
- post_detail.html → **Detalle de post**. Mostrará el contenido del post (publicación) que el cliente solicite.
- product_detail.html → **Detalle de producto**. Mostrará el contenido del producto que el cliente solicite.
- product_list.html → **Listado de los productos** dados de alta en el sistema.
- sidebar.html → **Panel lateral** que incluiremos en todas las plantillas. Su información se generará de forma dinámica para complementar el contenido principal. La lógica está definida de forma que:

- En las plantillas de listado mostrará **información estática sobre el autor**.
- Si estamos en la página de autor, mostrará i**nformación sobre los productos** para no repetir el contenido.
- Cuando estemos en una página de detalle de publicación, mostrará el **producto relacionado** con su enlace de compra. Si el post no tiene producto relacionado, se mostrará información sobre el autor.

- Archivos multimedia → media/

  Esta carpeta albergará **archivos multimedia subidos al sistema** mediante el panel de administración de Django o por los propios usuarios (si la aplicación lo permitiera). En nuestro caso, se tratará de las imágenes de los Post, Páginas y Productos que subiremos desde Django Admin a medida que creemos contenido.

- Node.JS API

  Como aplicación complementaria, se ha diseñado e implementado una pequeña **REST API** en el **lenguaje Node.js** que se conectará directamente a nuestra **base de datos Postgres** para realizar operaciones CRUD sobre los Post (Publicaciones) del sistema de blogging de la aplicación principal.

  La motivación para realizar esta API no es otra que la **flexibilidad** que ofrece, ya que nos permitirá gestionar las publicaciones de nuestro blog desde cualquier dispositivo con una conexión a internet.

  Desde que se presentaron los objetivos, se tuvo en cuenta que solamente se desarrollaría esta API, sin crear ninguna interfaz de usuario que haga uso de ella. Podemos decir que fue una decisión acertada, ya que por motivos de tiempo ya ha sido difícil incorporar la propia API en el desarrollo del proyecto.

  Sin entrar en demasiado detalle, puesto que se trata de un **complemento al proyecto principal**, pasaremos a detallar la funcionalidad y los archivos que componen esta **REST API**.

Estructura de carpetas:

WLOGAPI/

certs/ → Certificados auto-firmados para soporte https

config/ → Archivos de propiedades con variables globales

logs/ → Logs generados por la API conforme funciona

node_modules/ → Dependencias de Node

users/ → Registro de usuarios para la autenticación básica

Ficheros más importantes (Se encuentran en el directorio raíz):

- index.js

  Este es el **fichero principal de la aplicación**. Se encargará de importar y lanzar el resto de ficheros en el orden apropiado, así como de poner a la escucha el servidor en el puerto especificado.

- package.json

  Fichero **ligado a Node.js** en el que se van acumulando todos los cambios en las **dependencias de la aplicacón**. Es muy util para automatizar el despliegue de la misma.

- postgres.js

  En este fichero configuramos todas las **operaciones sobre la base de datos**, exponiéndolas para que puedan ser llamadas desde el fichero de rutas cuando la API reciba una petición.

- routes.js

  En este fichero definiremos **todas las rutas de la API**, espeficando si se trata de peticiones GET o POST. Se ha optado por utilizar POST para todas aquellas peticiones que requieren envío de información por parte del cliente, ya que nos pareció una buena práctica de seguridad.

- winston.js

  En este fichero está la **configuración de nuestros logs rotatorios**. La carpeta en la que se almacenarán, la frecuencia de rotación, el nombre que tendrán los ficheros y los diferentes niveles de logging (Info, errores…)

  Es realmente importante tener información de lo que ocurre en cada momento en la aplicación para poder subsanar errores y mantenerla actualizada.

6 Despliegue y pruebas

Durante el desarrollo del proyecto llevamos a cabo **pruebas unitarias** para asegurar la consistencia de la base de datos cada vez que realizábamos un cambio en nuestros modelos en Django.

Pruebas unitarias:

> Las pruebas están implementadas en el fichero blog/tests.py y nos permiten comprobar la correcta creación de cada uno de los diferentes objetos disponibles en nuestra base de datos.
>
> Estas pruebas se han ido repitiendo cada vez que durante el desarrollo nos hemos visto obligados a modificar algún detalle de los modelos, reflejando los cambios en la base de datos.

Pruebas de caja negra:

Nº	ESPECIFICACIÓN DE PRUEBAS
1	Objetivo probado: **Navegar por la aplicación web** Requisitos probados: Todos los enlaces funcionan correctamente Pruebas a realizar: Acceder a la aplicación web con un navegador y comprobar el correcto funcionamiento de todos los enlaces de la misma: • Se identifican claramente • Llevan al lugar indicado • Si son enlaces externos, se abren en una nueva pestaña.
2	Objetivo probado:**CRUD desde el panel de administración** de Django Requisitos probados: CRUD sobre Post, Páginas y Productos Pruebas a realizar: • Acceder al panel de administración de Django mediante usuario y contraseña. Realizar todas las operaciones CRUD sobre cada uno de los objetos Post, Página y Producto. Comprobar que los cambios son aplicados correctamente.
3	Objetivo probado: **Crear Posts desde la API externa** Requisitos probados: Crear una publicación utilizando la API Pruebas a realizar: • Hacer una llamada a nuestra REST API utilizando Postman y comprobar que la nueva publicación aparece en nuestra aplicación WLOG.

7 Conclusiones

- Objetivos alcanzados

  Considero alcanzados todos los objetivos marcados al inicio del proyecto, si bien es verdad que me habría gustado poder dedicar más tiempo a alguno de los apartados. Veamos, punto por punto, cada uno de ellos:

  - ✅Crear un sistema de blogging gratuito y open-source para desarrolladores.
  - ✅Colgar el sistema en Github para que cualquiera lo pueda descargar.
  - ✅El sistema incluirá:
    - ✅La definición de los modelos necesarios para su funcionamiento
    - ✅Una plantilla básica
    - ✅Un panel de administración para crear contenido
    - ✅Una API externa para futuras integraciones
  - ✅Crear una página web de ejemplo utilizando el sistema (albertofausto.com)

- Conclusiones del trabajo

  El proyecto **WLOG** me ha servido sobre todo para conocer y familiarizarme con el framework de Django y el lenguaje de programación Python. Al inicio del proyecto, me encontraba **aprendiendo Python por mi cuenta** para el puesto de trabajo que actualmente ocupo. Añadir la capa del framework y trabajar con él me ha servido para **afianzar mis conocimientos** del lenguaje Python, así como para aprender a trabajar con uno de los web frameworks más potentes del momento, que es utilizado por sitios tan populares y con un tráfico tan pesado como **Youtube** o **Twitter**. Creo que añadir estos conocimientos a mi currículum le **aportan valor** y hacen que mi perfil sea **más competitivo** en los tiempos que corren.

  También me ha parecido especialmente constructivo trabajar con un lenguaje de plantillas como **Jinja2** o el propio **Django Template Language**. Estuve comparando ambos, aprendiendo acerca de sus diferencias y

decantándome finalmente por el incluido en el propio sistema Django (en realidad son muy similares).

A la fecha de publicación de este proyecto, hace ya un par de años que trabajo como *Software Developer* y año y medio que lo hago en el extranjero. Si bien es verdad que estoy acostumbrado a **trabajar a diario como desarrollador**, hay ciertas partes del análisis e implementación de una aplicación que hacía tiempo que pasaba por alto. Bien es sabido que en el mundo laboral a veces **las prisas no nos dejan trabajar como nos gustaría**, y volver a tomarme tiempo para analizar de forma adecuada los requisitos de la aplicación, hacer los diagramas de clase, de uso… ha servido para refrescarme la memoria sobre muchos conceptos y me ha ayudado a **arrojar perspectiva** sobre la forma en que desarrollamos software en el mundo real.

También me ha sorprendido la rapidez y agilidad con la que Django permite desarrollar **aplicaciones web**. Creo que se ha avanzado mucho en los últimos 20 años en lo que al desarrollo web se refiere. **Frameworks** como **React.js** o el propio **Django** nos permiten crear sitios dinámicos con una agilidad que hace unos años no hubiésemos creído. Realmente nos facilitan las cosas y están ahí para que nos podamos enfocar en **desarrollar lo que verdaderamente importa**, que es la estructura y lógica de nuestras aplicaciones.

He podido, así mismo, ser más consciente sobre la gran cantidad de **software de código abierto** y de calidad del que disponemos. Esto es una tendencia al alza e incluso compañías como Microsoft están haciendo movimientos al respecto con frameworks como **.NET Core**, que corre en plataformas como Linux y es gratuito y open-source. A día de hoy, es raro que no encontremos una librería gratuita y open-source para cualquiera de nuestras necesidades, y las comunidades que se generan en torno a repositorios como **Github** o páginas como **StackOverflow** son simplemente fascinantes.

A modo de reflexión personal, si se me permite, creo que nos encontramos en un punto de **madurez en lo que se refiere al desarrollo de software**, donde priman las buenas ideas por encima de los desarrollos complejos y enrevesados de otros tiempos. Cualquiera con la suficiente motivación

puede aprender a programar haciendo uso de **recursos gratuitos en internet**, y eso es algo a valorar muy positivamente.

En definitiva, trabajar en el proyecto **WLOG** me ha hecho renovar mi pasión por el mundo de la programación, dándome cuenta de que ningún lenguaje dista tanto de otro y de que no es difícil hacer el cambio si el proyecto lo requiere. Me gustaría enfocar mis próximos pasos a **aprender buenas prácticas del desarrollo de software** que pueda aplicar a cualquier idioma y entorno de programación.

- Vías futuras

  Creo que el proyecto podría tener vías a futuro si se implementan ciertas **mejoras que agilicen el despliegue** y lo hagan **más amigable**. Algunas ideas de mejora serían:

  - Utilizar herramientas como **Fabric y Ansible** para automatizar el despliegue de la aplicación Django, haciendo mucho más fácil la adopción por parte de los usuarios.
  - Añadir una pequeña **selección de plantillas básicas** para que el usuario pueda escoger la que más convenga a su proyecto.
  - Integraciones para la API.
    - Móvil
      - Aplicación **Android / iOs** para interactuar con la API:
    - Email
      - Integración de la API con sistema de email. Publicación de posts mediante el **envío de correos electrónicos**.

8 Glosario

- **REST** → La transferencia de estado representacional o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.
- **API** → Del inglés Application Program Interface (Interfaz de Programación de Aplicaciones). Conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- **Framework** → Estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.
- **CRUD** → Del inglés Create, Read, Update, Delete (Crear, Leer, Actualizar, Borrar)
- **ORM** → Del inglés Object Relational Mapping (Mapeo Relacional de Objetos)
- **Jinja2** → Lenguaje de plantillas
- **Midleware** → Software que se sitúa como una capa intermedia entre otras dos.
- **Python** → Lenguaje de programación
- **Django** → Web Framework para Python
- **Javascript** → Lenguaje de programación
- **HTML** → Lenguaje de marcas
- **CSS** → Del inglés Cascade Style Sheet (Hojas de Estilo en Cascada)

9 Bibliografía

- **Sitio Oficial Python - https://www.python.org/**
- **Sitio Oficial Django - https://www.djangoproject.com/**
- **Sitio Oficial Node.js - https://nodejs.org/es/**
- **Manifiesto Ágil - http://agilemanifesto.org/iso/es/manifesto.html**
- **Sitio Oficial JInja2 - https://jinja.palletsprojects.com/en/2.11.x/**
- **Sitio Oficial Bootstrap - https://getbootstrap.com/**
- **Tutorial. Cómo crear un blog con Django - https://realpython.com/get-started-with-django-1/**
- **Tutorial 2. Cómo crear un blog con Django - https://www.skysilk.com/blog/2017/how-to-make-a-blog-with-django/**

10 Anexos

- Manual de instalación: Ver archivo readme.md en la raíz del proyecto en Github
- Manual del usuario: Ver archivo readme.md en la raíz del proyecto en Github
- Documentación de la API: Ver archivo Documentación WLOGAPI.pdf en /WLOGAPI en Github