eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Trabajo de Fin de Grado

Aplicación web para el diseño de Chatbots de Telegram

Iker Melero Mazagatos

dirigido por

Juanan Pereira Varela

Bilbao, 24 de julio de 2018

Resumen

En este proyecto se ha desarrollado una aplicación web, con el objetivo de proporcionar una herramienta para diseñar las pantallas o conversaciones de un *Chatbot* para *Telegram*. Para ello, se ha utilizado *Django*, el conocido *framework* web de código abierto escrito en *python*. La aplicación web tiene soporte multiusuario y es multiidioma. En lo que respecta al diseño de las conversaciones, se ofrece la utilidad *drag & drop* como método de añadir componentes a la pantalla, permitiendo también usar la misma utilidad para ordenar los componentes insertados. Permite además organizar las pantallas por proyectos, eliminarlas y guardarlas para su posterior edición. Además, es posible simular la pantalla de un dispositivo móvil para verificar que el diseño realizado hasta el momento se ajusta a sus límites, teniendo la posibilidad de generar y descargar una imagen del diseño de pantalla realizado.

Laburpena

Proiektu honetan, *Telegram*erako *Chatbot* pantaila edo elkarrizketak diseinatzeko tresna bat eskaintzeko helburuarekin, web aplikazio bat garatu da. Horretarako *Django* erabili da, *pythonen* idatzitako kode irekiko framework web ezaguna. Web aplikazioak erabiltzaile eta hizkuntza anitza da. Elkarrizketen diseinuari dagokionez, *drag & drop* aukera eskaintzen da osagaiak pantailara gehitzeko, era berean, aukera berdina erabili daiteke atxikitutako osagaiak antolatzeko. Gainera, pantailak proiektutan antolatzeko, ezabatzeko eta gordetzeko aukera ematen du, gerora editatu ahal izateko. Horrez gain, gailu mugikor baten pantaila simulatzea posible da, ordurarte egindako diseinua mugetara egokitzen dela egiaztatzeko, egindako pantaila diseinuaren irudi bat sortzeko eta deskargatzeko aukera izanez.

Abstract

In this project, a web application has been developed, with the goal to provide a tool to design screens or conversations of a *Chatbot* for *Telegram*. *Django* has been used for that, a known web framework of open code written in *python*. The web application has a multi-user support and it is multi-language. When it comes to the conversation designs, the *drag & drop* utility is provided as a method to add components to the screen, allowing to use the same utility to arrange the attached components. It allows also to organize the screens by projects, to eliminate them or to save them to edit them later on. In addition to that, it is possible to simulate a screen of a mobile device to verify that the created design until that time adjusts to its limits, having the possibility to create and download an image from the screen design.

Contenido

Lista de Figuras

Capítulo 1

Introducción

Es conocida la enorme expansión que están teniendo los `bots` en los últimos años, y más concretamente los `chatbots`. Con la llegada de los `smartphones` a nuestros bolsillos, se ha abierto un mundo de posibilidades para el desarrollo de estas herramientas. La posibilidad de comunicarse en cualquier momento y con la mayor rapidez de respuesta ha llevado a empresas, entidades y usuarios a interesarse por esta tecnología. Es por esto que muchas aplicaciones de comunicación han optado por incluir los chatbots como una funcionalidad más de sus plataformas; Telegram, Whatsapp, Skype y Facebook, entre otras.

Gracias a la inclusión de los *Chatbots* en las plataformas más famosas de comunicación, empresas y entidades han encontrado una forma directa y muy accesible de dar servicio a los usuarios. Estas empresas y entidades se ven en la necesidad de captar las necesidades, diseñar (lógica y gráficamente) y desarrollar este nuevo servicio que quieren ofrecer a medida.

Este proyecto se centra en cubrir la necesidad que en un momento dado pueden tener empresas, entidades e incluso usuarios de realizar un diseño gráfico del *Chatbot* que están desarrollando, para poder así mostrarle al cliente una aproximación bastante real de como será el *Chatbot* que han solicitado. Dada la variedad de plataformas que incluyen los *Chatbot* entre sus servicios y la diferencia en como representan graficamente esta herramienta, en los próximos puntos se detallará la herramienta desarrollada para diseñar las pantallas de los *Chatbots* de la aplicación de *Telegram*.

1.1. Origen del proyecto

Hace tiempo que el tutor de este proyecto, Juanan Pereira, profesor de la Escuela de Ingeniería Técnica de Bilbao, investiga en el campo de los ayudantes personales implementados con *Bots* de *Telegram*. Dada su experiencia en el desarrollo de *Chatbots*, tanto para su uso personal como para un uso público, ha ido viendo una carencia general, más notable cuando desarrolla un *Bot* para otra empresa o entidad, a la hora de presentar al cliente como va a ser la interacción de los usuarios con el *Chatbot*. No hay una herramienta de código libre y orientada a la aplicación de *Telegram*, con la que se pueda representar con detalle las pantallas finales del *Bot* y recibir el *feedback* del cliente para ir aproximándose al resultado deseado, salvo siguiendo una estrategia de prueba-error o esquemas realizados con alguna herramienta de dibujo.

En este proyecto se ha desarrollado una aplicación web que quiere cubrir la necesidad de una herramienta que sirva para diseñar las pantallas que representan las conversaciones de los *Chatbot*. En este caso, en la aplicación web se podrán diseñar pantallas para la *app* de mensajería instantánea *Telegram*.

1.2. Motivaciones para la elección del proyecto

El motivo principal por el cual acepté este proyecto, se debió a que vi la necesidad real que había de hacer una herramienta de estas características, que tenia una aplicación real y un potencial de crecimiento enorme. El gran auge que están teniendo los *Bots* no se puede ignorar y tener la oportunidad de formar parte de este cambio en las comunicaciones es muy motivante. Por otro lado, las tecnologías necesarias para su elaboración fue otro factor determinante a la hora de elegir este proyecto. Crear una aplicación web, moderna y con las últimas tecnologías en programación y diseño siempre ha sido un reto (y un deseo) para mi ,y con este proyecto lo he podido cumplir.

Desarrollo y diseño web avanzado

Como he mencionado más arriba, la idea de hacer una aplicación web moderna siempre me ha motivado. El hecho de poder publicarla en un ambiente real, y no solo verla en el entorno de desarrollo, ya es una gran satisfacción. Además, poder aprender a utilizar

los *framework* de desarrollo web más usados y aplicarlos en un proyecto real ha sido una gran motivación. Ya había oído hablar de algunos de ellos, como *Angular*, *Joomla* o *Django*, pero nunca había profundizado en ninguno de ellos. La elección de *Django* para este proyecto no ha hecho más que aumentar mi interés en estas herramientas.

Y no solo *framework*s de desarrollo web, sino también de diseño o maquetación web. He tenido la posibilidad de profundizar en las últimas funcionalidades de *HTML5*, *CSS3* y el *framework* de *Bootstrap 4* tan utilizadas en las aplicaciones web profesionales.

1.3. Caso de uso general

Imaginemos que el departamento post-venta de una empresa de automóviles está viendo incrementadas las llamadas a su centralita, está recibiendo más correos en sus buzones de atención al cliente y el *feedback* que recibe de los concesionarios sobre las consultas que les llegan van también en la misma línea. Supongamos que gran parte de estas consultas están relacionadas con preguntas comunes como: dónde se encuentra el taller oficial más cercano, consultas sobre garantías, como usar extras del vehículo, cuáles son sus emisiones, cuánto consume, etc.

La empresa de automóviles ve la posibilidad de automatizar la respuesta a todas estas preguntas y así poder liberar recursos humanos y derivarlos a otras tareas. Decide contratar a una empresa de programación y encargarle un *Chatbot* que se encargue de dar respuesta a estas consultas automáticamente. La empresa de programación recibe toda la información posible de la empresa de automóviles y le presenta un proyecto donde le recomienda empezar por la realización de un *Chatbot* para la aplicación *Telegram*. Ésta acepta el proyecto y la empresa de programación comienza a realizar el *Bot*.

Es la primera vez que esta empresa de programación se ve en la tesitura de realizar un proyecto de estas características. Los requerimientos y las expectativas de la empresa de automóviles son altas y les exigen un feedback continuo del proyecto. Entre esos requerimientos se encuentra el de saber la experiencia real que va a tener el usuario final del *Chatbot*. Les exigen conocer casi desde el primer momento como será la interacción entre *Bot* y cliente, para así poder ir cambiando el flujo de la conversación y pulir los

detalles. A la empresa de programación le supone un gran esfuerzo estar continuamente dibujando cómo van a ser las pantallas.

Finalmente la empresa de programación decide investigar un poco y encuentra nuestra aplicación. Se crea una cuenta en la plataforma y empieza a diseñar las pantallas del *Chatbot* para la empresa de automóviles según el gusto de esta. Pronto empiezan a tener los primeros prototipos listos y empiezan a exportar las pantallas en formato *.png* para usarlas en las reuniones que tendrán con la empresa de automóviles. Más adelante, tras recibir el *feedback* de la empresa de automóviles, vuelven a entrar a nuestra plataforma y modifican las pantallas para adaptarlas según lo acordado en las reuniones.

Esta empresa de programación valora muy positivamente la experiencia diseñando las pantallas en nuestra plataforma y decide utilizarla en futuros proyectos.

Capítulo 2

Captura de requisitos

En el siguiente capítulo se detallan los requisitos identificados durante el primer mes del proyecto. De está forma se quieren cubrir las necesidades que los desarrolladores de *Chatbots* se encuentran a la hora de diseñar y comunicar a sus clientes el contenido, el diseño y el flujo de las conversaciones.

Se incluirán también los diagramas de la jerarquía de actores, casos de uso y modelo de dominio, de manera que ayuden en la exposición de los requisitos recolectados.

2.1. Descripción general

Se requiere una aplicación web con la que se pueda diseñar conversaciones entre un *Chabot* de la plataforma *Telegram* y un usuario de esta. En la aplicación deberá haber gestión de usuarios y una forma lógica de organizar las pantallas diseñadas. Los usuarios tienen que tener la posibilidad de organizar las pantallas en proyectos y poder modificar y eliminarlos. Además, en lo que respecta a las pantallas, el usuario podrá guardar las pantallas para una futura modificación y exportarlas como imagen. En la parte de diseño de pantallas, existirá la posibilidad de añadir múltiples elementos a la pantalla, así como la posibilidad de personalizar colores, marcos o textos.

2.2. Requisitos funcionales

En este apartado se pretende explicar los requisitos que tiene que cumplir la aplicación web, de cara a satisfacer a los usuarios de esta. A continuación se enumeran dichos requisitos:

- **Alta en la plataforma**
- **Acceso a la plataforma**
- **Desconexión de la plataforma**
- **Creación de proyectos**
- **Eliminación de proyectos**
- **Creación de pantallas**
- **Edición/Diseño de pantallas**
- **Guardado de pantallas**
- **Exportación de pantallas**
- **Pre-visualización de pantallas**
- **Eliminación de pantallas**

2.3. Jerarquía de actores

La aplicación desarrollada en este proyecto está dirigida al uso público, tanto para personas individuales, como para empresas o entidades. De estos diferentes usuarios obtenemos un actor, el usuario.

Como en la aplicación web existen dos perfiles de usuarios, usuario sin registrar y usuario registrado, el actor anteriormente mencionado, *usuario*, se divide en dos nuevos actores: ***usuario no registrado***, ***usuario registrado***.

Además de los dos actores ya identificados, existe otro usuario que interactuará con la aplicación de manera diferente, lo denominaremos ***administrador***. Este actor se

encargará de administrar la aplicación pudiendo gestionar los usuarios ya registrados en la aplicación y pudiendo cambiar el contenido y la configuración del sitio web.

En la siguiente figura (figura 2.1) se muestra un esquema con los actores identificados.

FIGURA 2.1: Jerarquía de actores.

2.4. Casos de uso

A partir de los actores expuestos en el punto anterior, en la figura 2.2 se muestran los diferentes casos de uso de cada uno de ellos. Con ello se pretende mostrar las acciones que podrá realizar cada actor sobre la aplicación web.

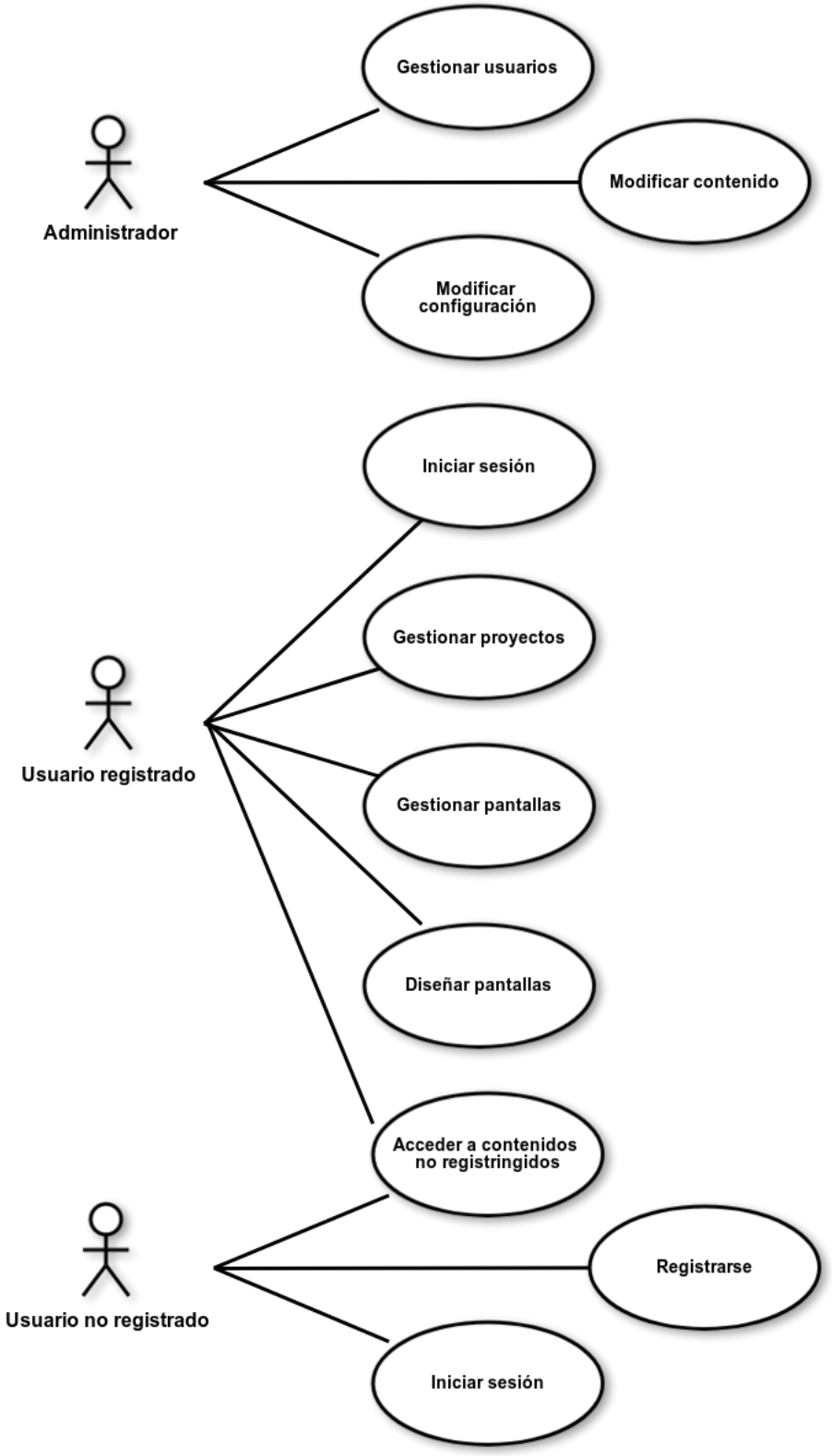

FIGURA 2.2: Casos de uso.

2.5. Modelo de dominio

En este punto se define el Modelo de Dominio, explicado gráficamente mediante la figura 2.3 y la explicación posterior de las Entidades y Relaciones que lo componen. Está definición será la base del diseño del modelo y su respectivo Diagrama de Entidad-Relación. En este Modelo de Dominio se han omitido todas las Entidades y Relaciones resultantes de la creación del proyecto con el *Framework web de Django*.

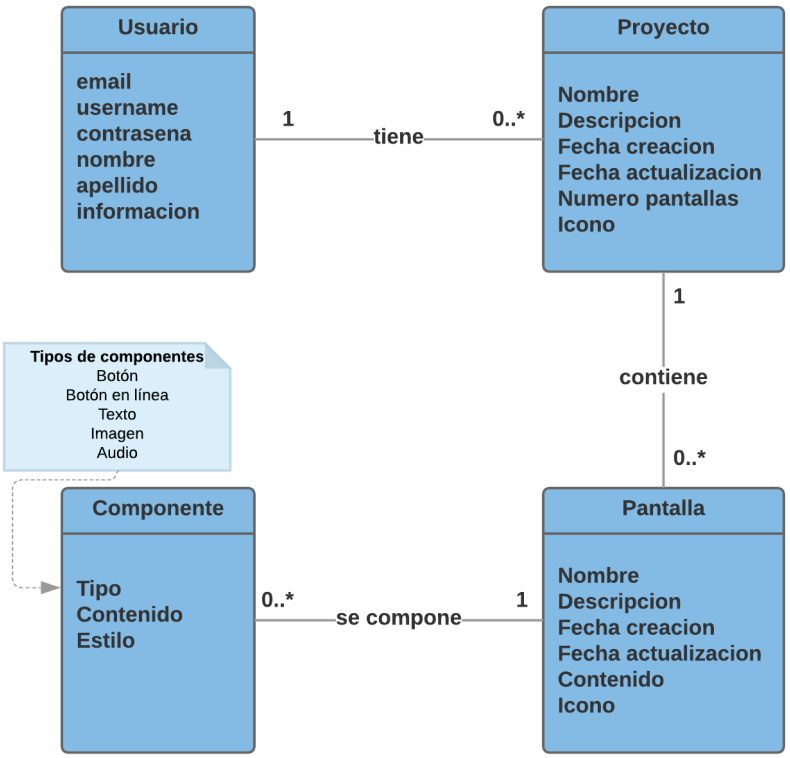

FIGURA 2.3: Modelo de dominio.

2.5.1. Entidades

Usuario

El usuario representará la persona, entidad o empresa que se registra en el sitio web. Se almacenará el email, nombre de usuario, nombre, apellido y descripción de cada usuario.

Proyecto

Representa la estructura en la que los usuarios podrán organizar las pantallas diseñadas. Almacena información de cada proyecto: Nombre, descripción, fechas de creación y actualización, icono y numero de pantallas.

Pantalla

Guarda cada diseño de pantalla y la información vinculada a esta: nombre, descripción, fechas de creación y actualización, icono y la pantalla en si.

Componente

Representa un tipo de componente presente en la pantalla. Almacena el contenido y el estilo de este.

2.5.2. Relaciones

Tiene

Relaciona un Usuario con ningún o múltiples proyectos. El Usuario es propietario de los Proyectos, que pueden ir de ningún a infinitos proyectos.

Contiene

Es el vinculo del Proyecto con sus Pantallas. El Proyecto puede contener ninguna o infinitas pantallas.

Se compone

Relaciona a la pantalla con sus componentes. La pantalla puede tener desde ninguno a infinitos componentes de diversos tipos.

2.6. Arquitectura

Esta sección, previa a la definición del alcance del proyecto, se divide en dos partes principales. En primer lugar se analiza la arquitectura física del servidor. En la segunda parte se pretende explicar la arquitectura lógica, dividiendo esta en dos partes: la arquitectura del servidor y la arquitectura de aplicación. A su vez esta última se desgrana en dos: el *framework Django* utilizado como base para el proyecto y la aplicación construida sobre este *framework*.

2.6.1. Arquitectura física

En lo referente a la arquitectura física del servidor, esta sigue la arquitectura clásica de cliente/servidor. Como vemos en la figura 2.4, existe un cliente que hace las peticiones (un navegador web) desde cualquier dispositivo con conexión a Internet, y al otro lado se encuentra el servidor que procesa las peticiones, realiza las operaciones necesarias y da una respuesta al cliente.

El servidor estará instalado en una máquina física, bajo un sistema operativo de la familia *Linux*. En esa misma máquina se albergará también la base de datos, con el software de *MySQL*.

2.6.2. Arquitectura lógica

Como se ha mencionado en la introducción de esta sección, la arquitectura lógica se divide en dos partes, con el objetivo de mantener una estructura más clara.

2.6.2.1. Arquitectura Servidor

En primer lugar, la aplicación web se ha alojado dentro de un entorno virtual. Esto va a permitir aislarla de cualquier otra aplicación que se encuentre en el servidor y tener las versiones de los paquetes *Python* necesarios para esta aplicación.

En segundo lugar, tal y como se presenta en la figura 2.4, se ha instalado el servidor de aplicaciones *uWSGI*. Este servidor se encargará de convertir las peticiones *http* provenientes del cliente en llamadas *Python* que la aplicación pueda entender.

En último lugar, sobre el servidor de aplicaciones, se encuentra el servidor *proxy Nginx*. Este servidor actuará como *proxy* inverso y distribuirá las peticiones que lleguen al servidor, pasándoselas al servidor de aplicaciones *uWSGI*. Gracias a *Nginx* se obtiene un alto rendimiento en el manejo de conexiones y unas medidas de seguridad extras de fácil configuración.

2.6.2.2. Arquitectura aplicación

Aplicación base *Django*

La arquitectura base de la aplicación es la arquitectura que sigue el *framework de Django*, se basa en un patrón propio llamado por sus creadores: **MTV (Model-Template-View)**. Este patrón es un interpretación que los creadores de *Django* hicieron del famoso patrón MVC (Model-View-Controller).

En esta interpretación, el primer término del nombre, la M, de **Modelo** (*Model* en inglés), es la parte que representa de que forma se van a guardar lo datos, como se van a validar y la relación que tienen entre ellos. Esta capa se traduce en uno o varios archivos dentro de la aplicación, donde se describen cada una de las entidades: su nombre, el tipo de datos, la longitud de estos, su relación con otras entidades, etc.

El segundo término del nombre, la T, de **Plantilla** (*Template* en inglés), hace referencia a como se van a mostrar los datos. Esta capa en *Django* esta representada en varios archivos *HTML*, donde el *HTML* se mezcla con un lenguaje propio del *framework* para las plantillas. Este lenguaje permite acceder a los datos, que junto a la llamada, la capa de Vista (se explicará en el siguiente párrafo) a transmitido a la plantilla. De esta manera, se podrá generar código *HTML* de manera dinámica.

El tercer y último término, la V, de **Vista** (*View* en inglés), se encarga de generar la información que se mostrará al usuario. Esta capa es el enlace entre el Modelo y la Plantilla. Esta capa esta representada por varias funciones que son llamadas desde otra parte del *framework*, la cual se explicará en el siguiente párrafo. Estas funciones leen la petición, interactúan con la capa Modelo, generan la información necesaria y llaman a la Plantilla correspondiente enviándole la información generada.

Otro término clave en la arquitectura de este *framework* es el módulo conocido como ***url resolver***. Se encarga de atender las peticiones que el servidor manda a la aplicación. Todas las peticiones pasan por él. Gracias a su configuración, elaborada mediante patrones, lee las *URLs* y las intenta encajar en sus patrones, si la URL encaja en algún patrón, se llama a la Vista definida para ese patrón, pasándole toda la información de la petición.

En la siguiente figura se pretende explicar gráficamente la arquitectura lógica del servidor y de la aplicación, dando una visión conjunta del funcionamiento de las dos partes interactuando entre si.

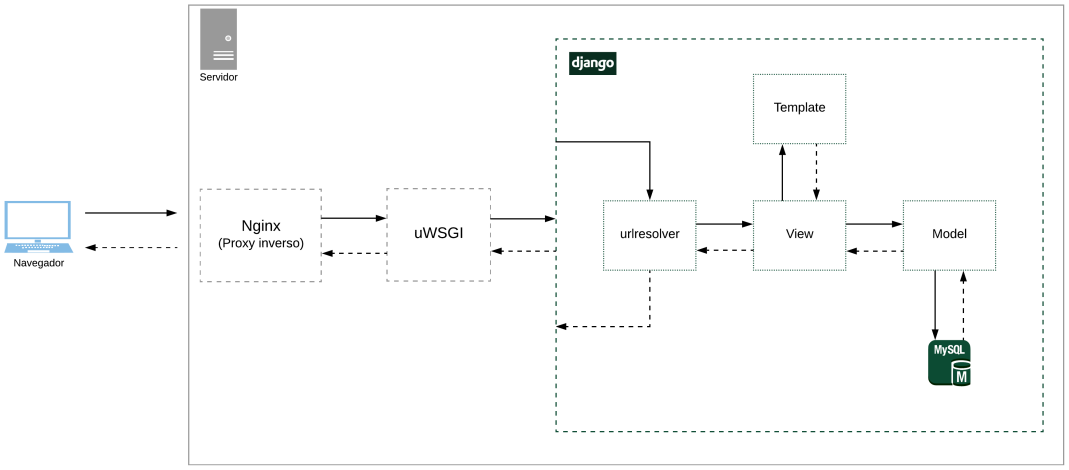

FIGURA 2.4: Esquema de solicitud, procesamiento y respuesta de peticiones.

Aplicación Web diseño pantallas

Además del *backend* en *Django* se ha desarrollado también el *frontend* en *HTML*, *CSS* y *JavaScript*. Se ha separado el contenido de cada uno de estos tres lenguajes, de manera que el contenido de la página web quede aislado del estilo.

Capítulo 3

Planteamiento inicial

Este capítulo tiene como fin exponer los objetivos, el alcance y hacer un análisis de la planificación temporal.

3.1. Objetivos

En esta sección se quieren presentar los objetivos principales del proyecto. Se dividen en tres, los dos primeros tienen que ver con la aplicación web y las funcionalidades que se quieren facilitar a los usuarios, el tercero está relacionado con la explicación del proyecto. Los tres puntos tienen unos objetivos bases comunes:

- **claridad y simpleza**, tanto en la interfaz de usuario, como en el código, como en la documentación, se ha querido dar una imagen de claridad, y simpleza en cierto modo.
- **organización**, tanto las funcionalidades y el código de la aplicación, como la documentación siguen una estructura lógica e interpretable.
- **facilidad de ampliación**, dada la estructura y las tecnologías utilizadas en los tres objetivos, se facilita la comprensión y la escalabilidad de cada uno de ellos.

Aplicación web

Con la aplicación web se proporciona a los usuarios una plataforma donde poder crear, pre-visualizar, modificar, exportar y eliminar las conversaciones que quieran diseñar para su *Chatbot*.

El diseño de las páginas web será lo más simple y auto-explicativo posible, de manera que no sea necesario formar a los usuarios de esta.

Se pretende presentar al usuario datos relevantes sobre los datos de los proyectos y pantallas creados como: las fechas de creación y actualización, el número de pantallas de un proyecto o la vista en miniatura de cada pantalla. Con estos detalles se quiere crear una interfaz de usuario amigable y ser transparentes con el usuario en todo momento.

Además, otro de los objetivos a tener en cuenta, es el de asegurar que la web cumpla requisitos no funcionales relacionados con la seguridad, eficiencia y accesibilidad. Protegiendo la página web frente a multitud de ataques posibles y garantizando la seguridad de la aplicación y del usuario en todo momento.

Diseño gráfico de conversaciones

El diseño gráfico de las conversaciones de un *Chatbot* es el objetivo principal de este proyecto. Con esta funcionalidad los usuarios dispondrán en una única página con:

- todos los elementos con los que crear las conversaciones.
- visualización en tiempo real de la conversación que van creando.
- posibilidad de navegar a través de todas las pantallas del proyecto.
- funciones para guardar, pre-visualizar, exportar o eliminar las conversaciones.

Documentación

En la redacción de esta documentación se pretende ofrecer, de una forma clara y organizada, una guía para la explicación de todas las partes que ha englobado el proyecto. Además, tiene como fin servir para futuros usuarios que quieran utilizar la aplicación o desarrolladores que quieran mejorar o aumentar su funcionalidad.

Organización personal

Uno de los objetivos fijados al comienzo del proyecto ha sido el de lograr una organización personal óptima. Para ello, se ha utilizado la herramienta web *Trello*. En la siguiente figura, se muestra una captura de pantalla donde se puede ver parte de las listas y las tareas fijadas durante el desarrollo.

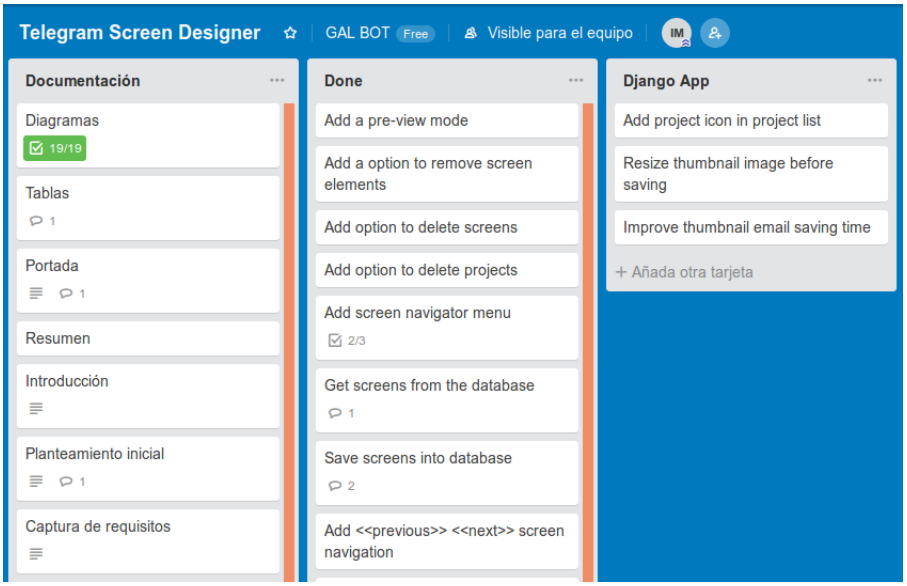

FIGURA 3.1: Captura ejemplo uso herramienta Trello.

3.2. Alcance

Definir el alcance de un proyecto es imprescindible para el éxito de este. En esta sección se van a desgranar todas las tareas que engloban el proyecto, dando información sobre la duración y los recursos necesarios para la consecución de cada una de ellas. Se utilizarán diagramas EDT y tablas con la intención de dar una imagen clara y ordenada del alcance del proyecto.

A continuación, en la figura 3.2, se muestra un esquema general de cómo se han clasificado las tareas del proyecto. Más adelante se irán mostrando esquemas para cada una de esas clasificaciones, a estos esquemas le acompañarán unas tablas, que ampliarán la información mostrada en las figuras.

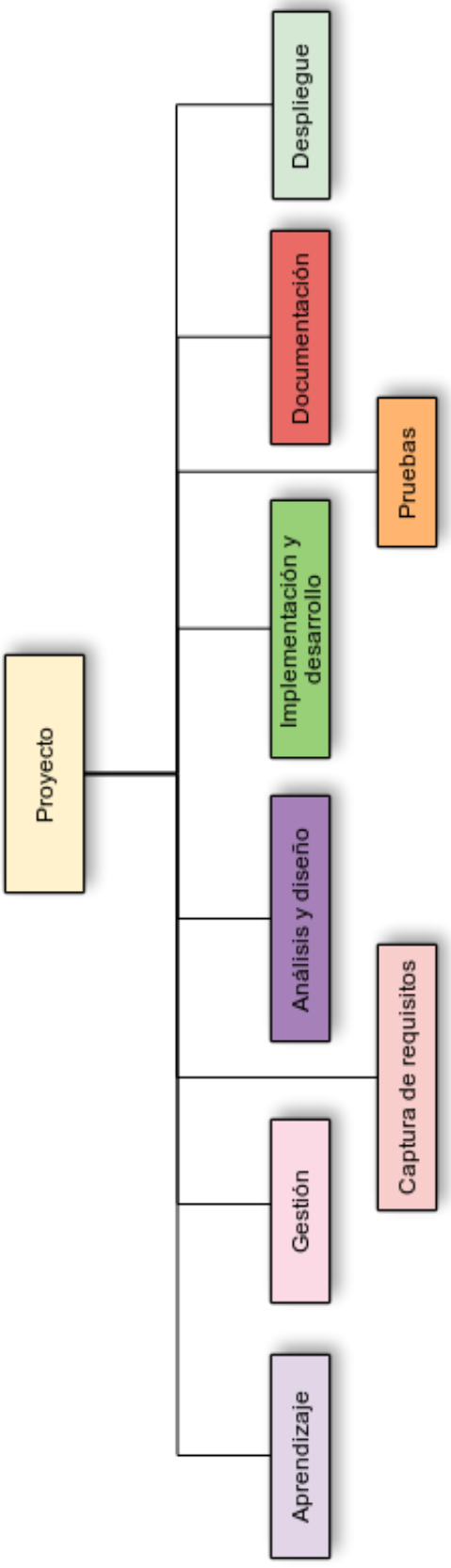

FIGURA 3.2: Diagrama EDT del proyecto.

3.2.1. Aprendizaje

La tarea del aprendizaje contempla los intervalos de estudio por los que se ha tenido que pasar para poder manejar de forma eficaz las herramientas utilizadas en el desarrollo del proyecto.

A continuación, se muestra un diagrama EDT con los puntos que componen la tarea del aprendizaje en este proyecto.

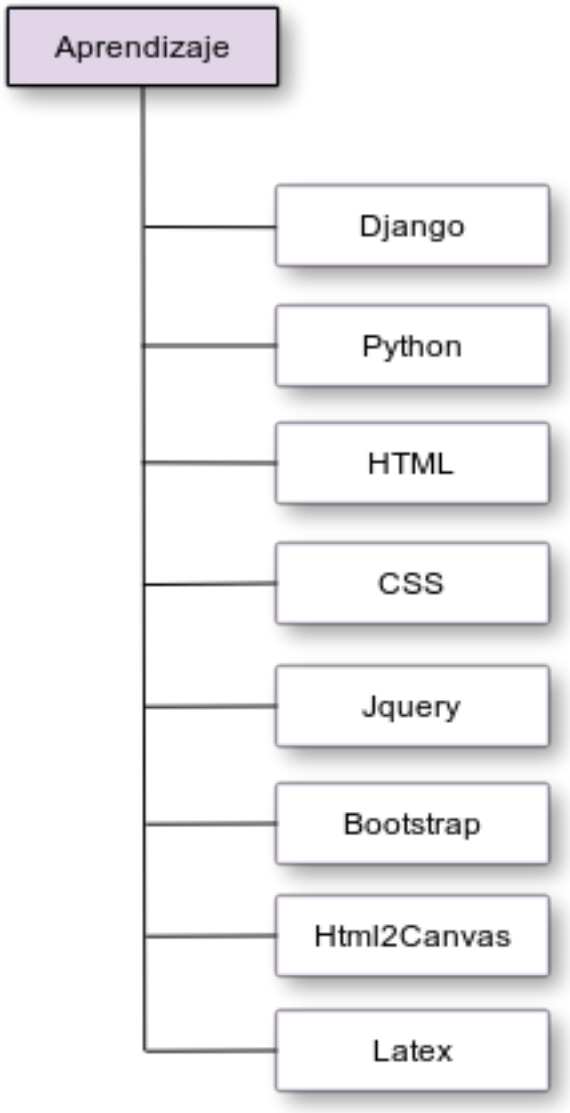

Figura 3.3: Diagrama EDT de la tarea de aprendizaje.

En la siguiente tabla se amplia la información mostrada en la figura 3.3.

Estudio y aprendizaje arquitectura y funcionalidades *framework Django*	
Duración estimada	60 horas
Descripción	Estudio de la arquitectura del *framework*, realización curso de iniciación y profundización en las funciones necesarias para el proyecto.
Recursos necesarios	*Python*, framework *Django*, *Brackets*, *Firefox*, documentación oficial *Django* y curso *Tango with Django*.
Aprender a programar en Python	
Duración estimada	10 horas
Descripción	Aprendizaje de la sintaxis de *Python* así como las diferencias con otros lenguajes ya interiorizados.
Recursos necesarios	*Firefox*, *Brackets*, *Python*.
Profundización en el lenguaje HTML	
Duración estimada	5 horas
Descripción	Estudio más avanzado del visto en clase, profundización en elementos del DOM.
Recursos necesarios	*Firefox*, Herramienta Desarrolladores *Firefox*, *Brackets*, HTML.
Aprendizaje de CSS avanzado	
Duración estimada	15 horas
Descripción	Aprendizaje completo del estilo web, profundizando en diferentes estilos de rejilla.
Recursos necesarios	*Firefox*, Herramienta Desarrolladores *Firefox*, *Brackets*, CSS.
Estudio JavaScript y la librería jQuery	
Duración estimada	10 horas
Descripción	Aprender a utilizar el lenguaje JS y las funciones de la librería jQuery necesarias para este proyecto. Hay que destacar que este lenguaje y liberia ya se estudiaron en el grado.
Recursos necesarios	Firefox, Herramienta Desarrolladores *Firefox*, JS, jQuery.
Aprendizaje del uso de *Bootstrap*	
Duración estimada	5 horas
Descripción	Interiorización de conceptos y aprendizaje de componentes de Bootstrap 4.
Recursos necesarios	*Firefox*, Herramienta Desarrolladores *Firefox*, JS, jQuery, HTML, CSS, Bootstrap.
Aprendizaje del uso de *Html2Canvas*	
Duración estimada	1 hora
Descripción	Aprender a generar una imagen a partir de un elemento HTML mediante esta libería.
Recursos necesarios	*Firefox*, Herramienta Desarrolladores *Firefox*, JS, jQuery, Html2Canvas.
Aprendizaje del lenguaje LaTeX	
Duración estimada	15 horas
Descripción	Aprender a crear documentos bien estructurados, separando el contenido del estilo usando el lenguaje LaTeX.
Recursos necesarios	Navegador, cuenta en *Overleaf*.

Tabla 3.1: Tabla ampliación EDT Aprendizaje.

3.2.2. Gestión

La organización es el proceso de ordenado y secuenciación de las partes del proyecto, con el objetivo de que se realicen de forma eficaz.

El diagrama EDT de la figura 3.4 muestra los puntos que componen la tarea de la organización.

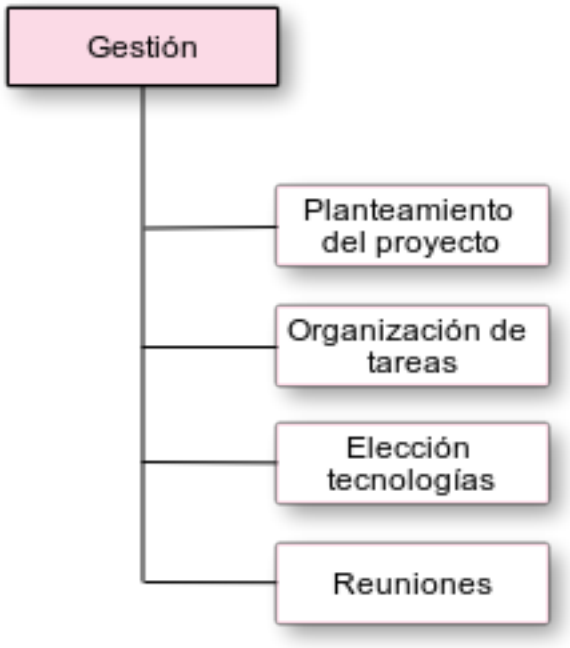

FIGURA 3.4: Diagrama EDT de la tarea de gestión.

En la siguiente tabla se amplia la información mostrada en la figura 3.4.

Planteamiento del proyecto	
Duración estimada	5 horas
Descripción	Plantear el proyecto a desarrollar, basándose en las necesidades y tecnologías disponibles. Estudiar la viabilidad de las opciones planteadas.
Recursos necesarios	Navegador, papel y lápiz.
Organización de las tareas	
Duración estimada	7 horas
Descripción	Desgranar las tareas principales y establecer los objetivos y primeros plazos de cada una de ellas.
Recursos necesarios	Papel, lápiz y *Trello*
Elección de las tecnologías	
Duración estimada	5 horas
Descripción	Elegir las tecnologías adecuadas para el proyecto. Definir las características del servidor necesarias para desplegar la aplicación.
Recursos necesarios	*Firefox* y *software* varios (los elegidos).
Reuniones	
Duración estimada	15 horas
Descripción	Reuniones con el tutor para sentar las bases del proyecto, fijar objetivos, resolver dudas y mejorar la aplicación.
Recursos necesarios	Papel, lápiz, *Skype* y *Trello*.

Tabla 3.2: Tabla ampliación EDT Gestión.

3.2.3. Captura de requisitos

Este apartado recoge la captura de los requisitos que se habrían que cumplir utilizando las herramientas elegidas durante el desarrollo del proyecto.

El diagrama EDT muestra los puntos que componen la tarea de la captura de requisitos.

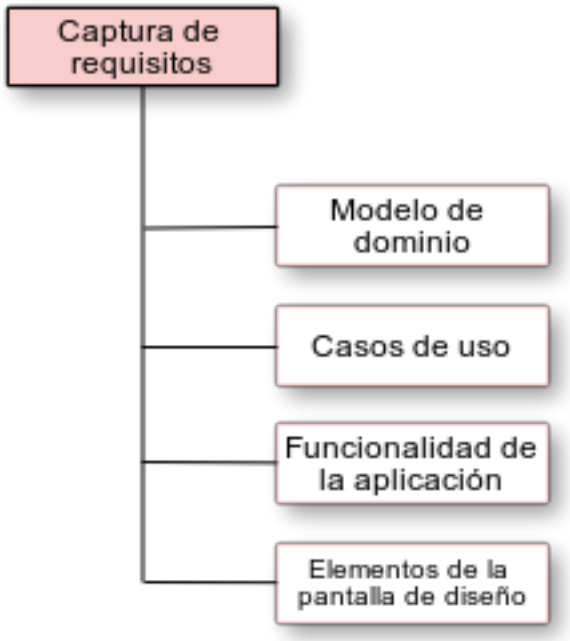

FIGURA 3.5: Diagrama EDT de la captura de requisitos.

En la siguiente tabla se amplia la información mostrada en la figura 3.5.

Casos de uso	
Duración estimada	6 horas
Descripción	Identificar los casos de uso resultantes de las diferente acciones que se lleven a cabo de la aplicación web.
Recursos necesarios	Papel y lápiz.
Funcionalidad de la aplicación	
Duración estimada	8 horas
Descripción	Analizar las funcionalidades que tiene que tener la aplicación y establecer las necesidades de diseño.
Recursos necesarios	*Firefox*, *Trello*, papel y lápiz.
Elementos de la pantalla de diseño	
Duración estimada	6 horas
Descripción	Identificar los elementos a incluir en el diseñador de pantallas para reproducir las conversaciones de *Telegram*.
Recursos necesarios	Aplicación de *Telegram*, *Trello*, papel y lápiz.
Modelo de dominio	
Duración estimada	2 horas
Descripción	Diseñar el modelo que mejor se adapte a la aplicación necesaria.
Recursos necesarios	Papel y lápiz.

Tabla 3.3: Tabla ampliación EDT Captura de requisitos.

3.2.4. Análisis y diseño

En este apartado, se detalla la planificación a la hora de analizar el proyecto y diseñar los distintos diagramas que se requieren.

El diagrama EDT muestra los puntos que componen la tarea del análisis y diseño.

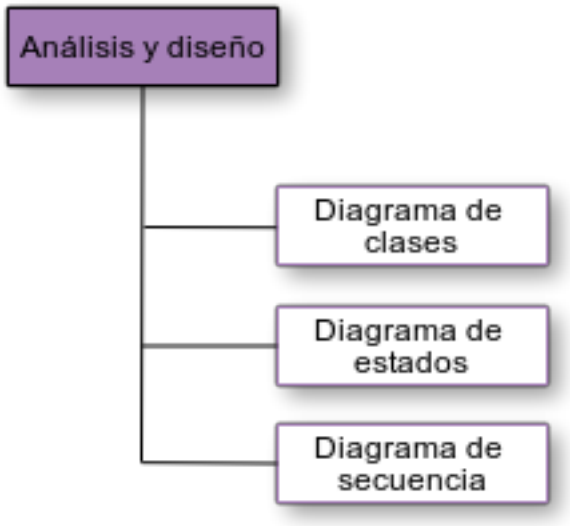

FIGURA 3.6: Diagrama EDT del análisis y diseño.

En la siguiente tabla se amplia la información mostrada en la figura 3.6.

Diagrama de clases	
Duración estimada	4 horas
Descripción	Elaboración del diagrama de clases del proyecto.
Recursos necesarios	*Cacoo*, papel y lápiz.
Diagrama de estados	
Duración estimada	4 horas
Descripción	Elaboración del diagrama de estados del proyecto.
Recursos necesarios	*Cacoo*, papel y lápiz.
Diagrama de secuencia	
Duración estimada	10 horas
Descripción	Realización de los diagramas de secuencia correspondientes a las diferentes funciones que se pueden realizar con la aplicación web.
Recursos necesarios	*Cacoo*, papel y lápiz.

TABLA 3.4: Tabla ampliación Análisis y diseño.

3.2.5. Implementación y desarrollo

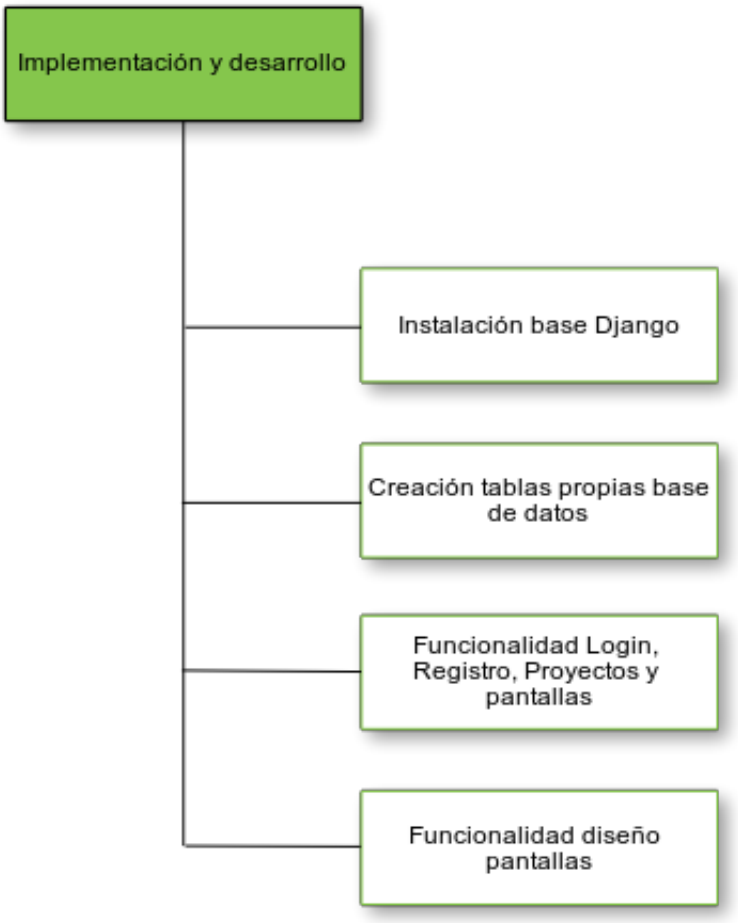

FIGURA 3.7: Diagrama EDT de la implementación y el desarrollo.

En la siguiente tabla se amplia la información mostrada en la figura 3.7.

Instalación base Django	
Duración estimada	20 horas
Descripción	Creación de un nuevo proyecto con el *framework Django*, asimilación del entorno, configuración básica y personalización de algunos apartados.
Recursos necesarios	*Brackets*, *Django* y *Consola de comandos*, y *Firefox*.
Creación tablas propias base de datos	
Duración estimada	4 horas
Descripción	Creación de las tablas necesarias, adicionales a las tablas base de *Django*, mediante el propio *framework*. Comprobación de las tablas obtenidas en *MySQL*.
Recursos necesarios	*Brackets*, *Django* y *MySQL*.
Funcionalidad *Login*, Registro, Proyectos y pantallas	
Duración estimada	40 horas
Descripción	Programación de las funcionalidades básicas de la aplicación web.
Recursos necesarios	*Brackets*, *Django* y *Consola de comandos*, y *Firefox*.
Funcionalidad diseño pantallas	
Duración estimada	30 horas
Descripción	Implementación de la funcionalidad de diseño de pantallas. ▪ Creación de los elementos que se añadirán al lienzo donde se podrán simular las conversaciones. ▪ Diseño del lienzo contenedor de los elementos.
Recursos necesarios	*Brackets*, *Django*, *Consola de comandos*, *Firefox* y Herramienta desarrollador de *Firefox*.

Tabla 3.5: Tabla ampliación EDT Implementación y desarrollo.

3.2.6. Pruebas

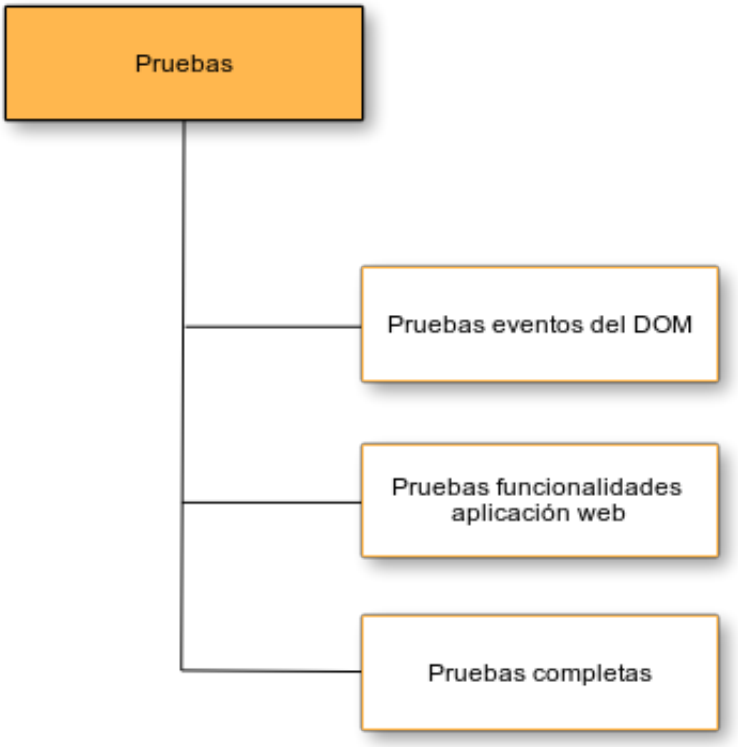

FIGURA 3.8: Diagrama EDT de las pruebas.

En la siguiente tabla se amplia la información mostrada en la figura 3.8.

Pruebas eventos del DOM	
Duración estimada	7 horas
Descripción	Pruebas realizadas durante la implementación para comprobar que las acciones del usuario lanzaban correctamente los eventos *jQuery* y las acciones programadas en estos.
Recursos necesarios	*Brackets*, *Firefox* y Herramienta desarrollador de *Firefox*.
Pruebas funcionalidades aplicación web	
Duración estimada	12 horas
Descripción	Comprobación de cada una de las funcionalidades básicas de la aplicación. Chequeo de los resultados en la base de datos.
Recursos necesarios	*Brackets*, *Firefox*, *Django* y *MySQL*.
Pruebas completas	
Duración estimada	7 horas
Descripción	Chequeo de las funciones descritas por los casos de uso y comprobación general de la aplicación web asegurando una buena experiencia de usuario.
Recursos necesarios	*Brackets*, *Django*, y *Firefox*.

Tabla 3.6: Tabla ampliación EDT Pruebas.

3.2.7. Documentación

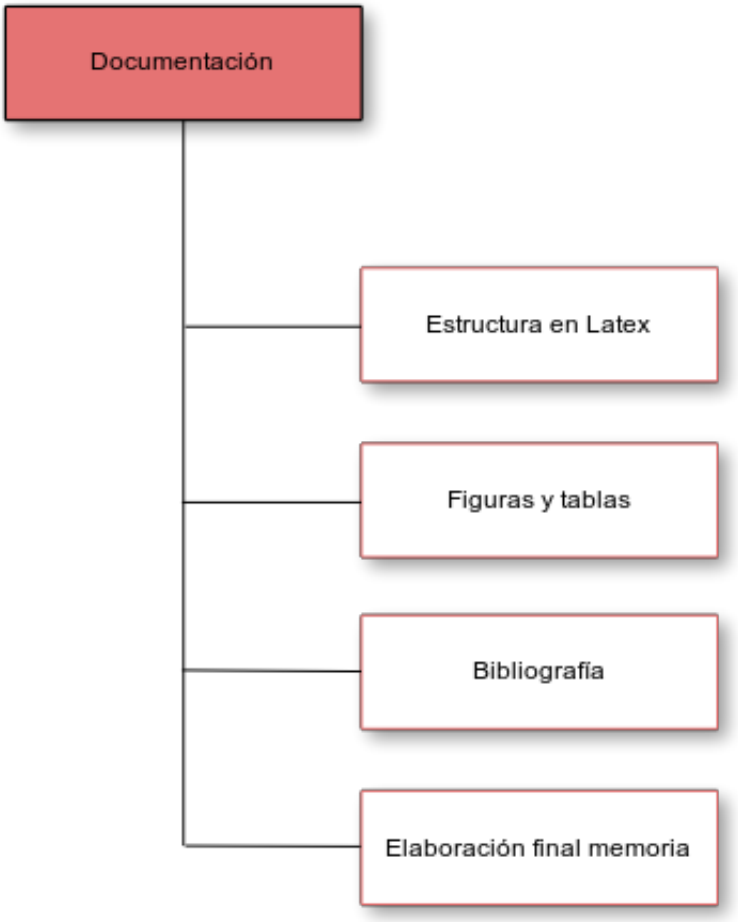

FIGURA 3.9: Diagrama EDT de la documentación.

En la siguiente tabla se amplia la información mostrada en la figura 3.9.

Estructura en Latex	
Duración estimada	4 horas
Descripción	Preparación de los puntos a desarrollar de la documentación, así como del código necesario en el lenguaje LaTeX
Recursos necesarios	*Firefox*, LaTeXy *Overleaf*.
Figuras y tablas	
Duración estimada	20 horas
Descripción	Creación de todas las figuras y tablas incluidas en la documentación del proyecto.
Recursos necesarios	*Firefox*, *Cacoo*, LaTeXy *Overleaf*.
Bibliografía	
Duración estimada	4 horas
Descripción	Recogida y organización de todas las referencias utilizadas en el desarrollo del proyecto.
Recursos necesarios	*Firefox*, LaTeXy *Overleaf*.
Elaboración final memoria	
Duración estimada	40 horas
Descripción	Elaboración completa de la documentación, tomando como base la estructura de LaTeXy utilizando las figuras, las tablas y la bibliografía anteriormente realizadas.
Recursos necesarios	*Firefox*, LaTeXy *Overleaf*.

Tabla 3.7: Tabla ampliación EDT Documentación.

3.2.8. Despliegue

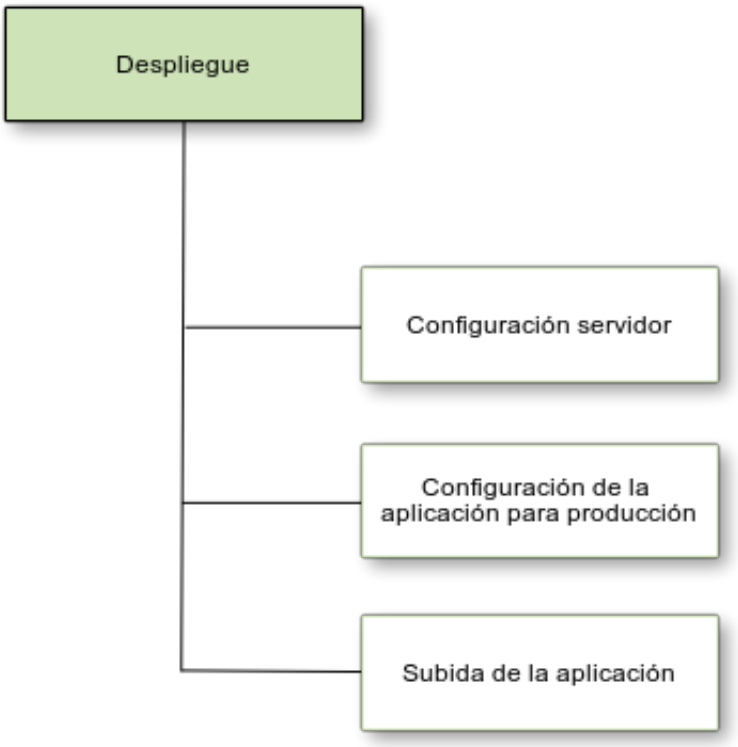

FIGURA 3.10: Diagrama EDT del despliegue.

En la siguiente tabla se amplia la información mostrada en la figura 3.10

Configuración del servidor	
Duración estimada	5 horas
Descripción	Instalación de todo el software necesario para el funcionamiento de la aplicación web.
Recursos necesarios	Consola de comandos Linux.
Configuración de la aplicación para producción	
Duración estimada	2 horas
Descripción	Edición del archivo de configuración de la aplicación para la subida a producción.
Recursos necesarios	*Brackets*.
Subida de la aplicación	
Duración estimada	2 horas
Descripción	Subida de la aplicación al servidor y chequeo completo.
Recursos necesarios	*Brackets*, Consola de comandos Linux.

Tabla 3.8: Tabla ampliación EDT Despliegue.

3.3. Planificación temporal

En esta sección se profundiza en los tiempos requeridos para cada tarea, la ejecución temporal y la secuencia de cada una de ellas. En la tabla 3.9, podemos ver las tareas enumeradas en la sección anterior. En esta tabla, además, vemos la relación de procedencia y el total de horas invertidas en cada sección y en el total del proyecto.

ID	Tarea	Predecesor	Tiempo estimado (en horas)
	Aprendizaje		121
1	*Django*	-	60
2	*Python*	-	10
3	*HTML*	-	5
4	*CSS*	-	15
5	*jQuery*	-	10
6	*Bootstrap*	3,4,5	5
7	*Html2Canvas*	3,5	1
8	LaTeX	-	15
	Gestión		32
9	Planteamiento del proyecto	-	5
10	Organización de las tareas	9	7
11	Elección de las tecnologías	10	5
12	Reuniones	-	15
	Captura de requisitos		22
13	Casos de uso	11	6
14	Funcionalidad de la aplicación	13	8
15	Elementos de la pantalla de diseño	13	6
16	Modelo de dominio	14	2
	Análisis y diseño		18
17	Diagrama de clases	16	4
18	Diagrama de estados	13	4
19	Diagrama de secuencia	18	10
	Implementación y desarrollo		94
21	Instalación base *Django*	1,2	20
22	Creación tablas propias base de datos	17	4
23	Funcionalidad Login, Registro, Proyectos y Pantallas	1,2	40
24	Funcionalidad diseño de pantallas	3,4,5,6,7,22	30
	Pruebas		26
25	Pruebas eventos del DOM	24	7
26	Pruebas funcionalidades aplicación web	23	12
27	Pruebas completas	25,26	7
	Documentación		68
28	Estructura en LaTeX	8	4
29	Figuras y tablas	22,23,24	20
30	Bibliografía	-	4
31	Elaboración final memoria	28,29,30	40
	Despliegue		9
32	Configuración servidor	-	5
33	Configuración de la aplicación para producción	1	2
34	Subida de la aplicación	-	2
Total horas invertidas en el proyecto			**390**

Tabla 3.9: Detalle con tiempos del alcance temporal.

A continuación, y utilizando los datos presentados en la tabla 3.9, se muestra un diagrama de Gantt (figura 3.11). Mediante este diagrama, se pretende exponer el tiempo de dedicación previsto para cada una de las tareas dentro de los límites fijados de inicio y fin de proyecto. Además, también se puede observar la secuencia de ejecución de las tareas.

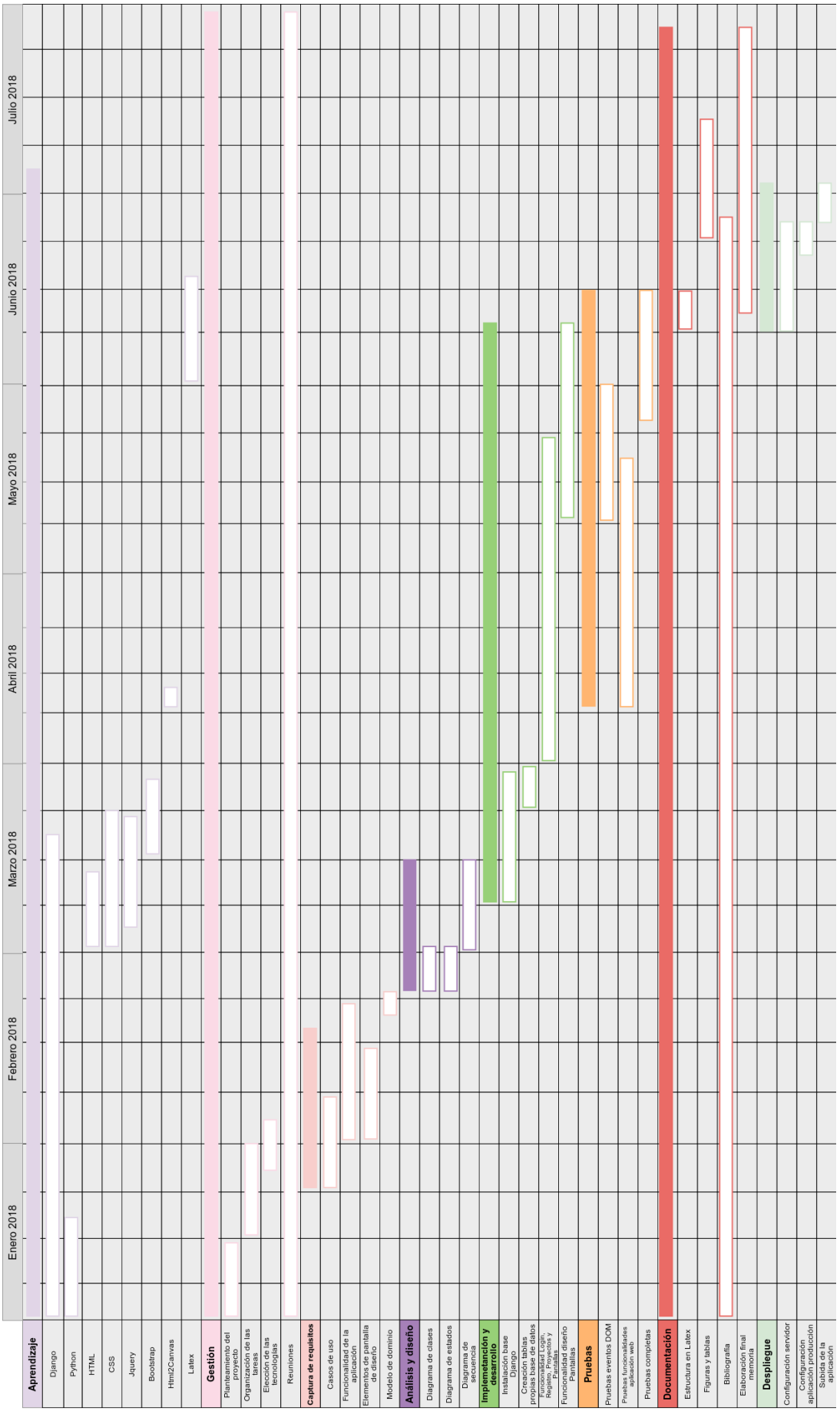

FIGURA 3.11: Diagrama Gantt del proyecto.

Como se ve en la tabla 3.9, el total de tiempo invertido en el proyecto es de 390 horas. Teniendo en cuenta las fechas de realización del proyecto, iniciándose el *2 de enero de 2018* y dándose totalmente por terminado el *24 de julio de 2018*, y una jornada laboral de 4 horas diarias y 4 en todo el fin de semana, el total de horas disponibles para la realización del proyecto es igual a 672. En la siguiente fórmula se muestra el cálculo:

$$Meses * Semanas * Horas\ semanales = 7 * 4 * 24 = 672\ horas \tag{3.1}$$

Si a este total de horas le restamos las horas correspondientes a posibles festivos, días de descanso y acontecimientos inesperados, que según la estimación supondrían 120 horas, obtenemos las siguientes horas disponibles totales:

$$672 - 120 = 552\ horas \tag{3.2}$$

Recuperando de nuevo el número total de horas invertidas en el proyecto, 390 horas, y comparándolo con el número total de horas disponibles resultado de la fórmula 3.2, 552 horas, la conclusión es que el periodo de tiempo fijado para este proyecto ha sido más que suficiente, e incluso se podrían haber acortado los plazos de su entrega.

Capítulo 4

Análisis de antecedentes

Antes de la realización de este proyecto, se investigó el estado del arte, posibles aplicaciones web con las que se pudiese diseñar las conversaciones de un *Chatbot*. Tras la investigación, se encontró una plataforma que se acercaba bastante a las necesidades que se tenían. La plataforma en cuestión se encuentra en la siguiente dirección: [**?**] https://www.fakewhats.com/generator .

Esta herramienta web permite crear conversaciones 'falsas' de *Whatsapp*, es bastante intuitiva y fácil de usar. Permite personalizar todos los elementos de la parte superior de la pantalla y añadir mensajes de texto y foto a la conversación. Una vez terminado el diseño permite exportarlo como imagen.

En cambio, podemos ver claramente que esta herramienta no está pensada para el diseño de conversaciones con un *Bot*, por ello, se echa en falta muchos elementos indispensables en una conversación de ese tipo, mucho más si nuestro objetivo es crear un *Chatbot* en *Telegram*. Por un lado, faltan tipos de mensaje, como mensajes de audio o teclados en línea. Tampoco encontramos la posibilidad de añadir teclados personalizados. Por otro lado, con esta plataforma no podemos guardar nuestros diseños y organizarlos de forma estructurada para posteriores actualizaciones.

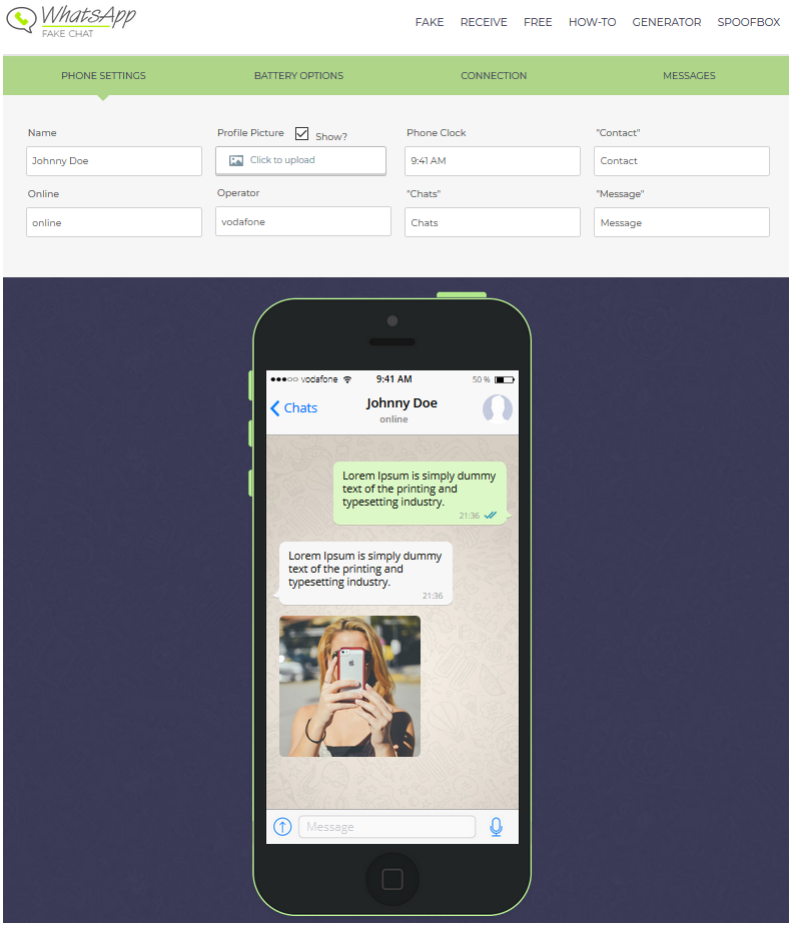

FIGURA 4.1: Captura pantalla pagina web Fake Whatsapp.

Además de la aplicación web expuesta anteriormente, se ha analizado otra plataforma, https://botsociety.io/. Esta herramienta si está pensada para el diseño de conversaciones con un *Chatbot*, pero de nuevo, se echan en falta elementos imprescindibles en una conversación con un *Bot*, nos limita a texto, imágenes y botones. Permite elegir diferentes plataformas objetivo para las que diseñar la conversación, aunque, a fecha de realización de este proyecto, entre esa lista no se encuentra *Telegram*. Por otro lado, tiene una gran funcionalidad, que es la de grabar un vídeo con el flujo de la conversación.

A pesar de tener grandes funcionalidades, al no ser de código abierto, su funcionalidad con la versión gratuita es muy limitada.

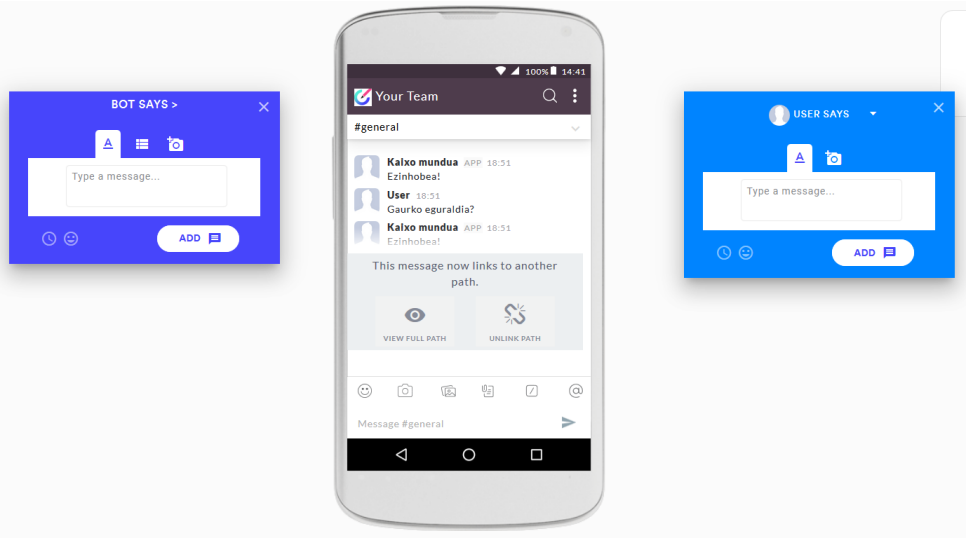

FIGURA 4.2: Captura pantalla pagina web Botsociety.

Tras el análisis realizado, se ha comprobado que no existe una herramienta capaz de cumplir con los requisitos que se exponen en el capítulo anterior.

Con este proyecto, se quiere dar una solución completa al diseño de conversaciones en *Chatbots* de *Telegram*. Incluyendo los elementos más importantes que ofrece *Telegram* para añadir a la conversación:

- Mensajes de texto
- Mensajes de audio
- Mensajes de imagen
- Botones en línea
- Teclados personalizados

Junto con los elementos proporcionados para el diseño de conversaciones, se ofrece la posibilidad de guardar la conversación que se está diseñando, exportarla rápidamente en formato *png* y organizar las pantallas por proyectos, para poder tener la posibilidad de trabajar en el diseño de más de un *Chatbot* bajo la misma cuenta.

Además de la funcionalidad y los elementos incluidos en este proyecto, gracias a la utilización del *framework Django*, es posible ampliar las características de una manera sencilla para alguien con conocimientos en este *framework*.

Capítulo 5

Análisis y diseño

Este capítulo se dedicará a la exposición de los diferentes diagramas imprescindibles en un desarrollo de estas características. Más tarde se utilizarán en la construcción de la página web y la base de datos, sirviendo como base sólida y guía durante la etapa de desarrollo, que se detallará en el siguiente capítulo.

En los siguientes puntos se detallan el Diagrama Entidad-Relación, el Diagrama de clases y un Diagrama de secuencia de la aplicación.

5.1. Diagrama Entidad-Relación

En la figura 5.1, se muestra el diagrama Entidad-Relación correspondiente al proyecto. En la figura no aparecen las entidades y relaciones correspondientes a las tablas que el *framework de Django* genera automáticamente al crear un nuevo proyecto. Salvo la de *core_customuser*, ya que esta entidad se ha personalizado y es clave en el funcionamiento de la aplicación web.

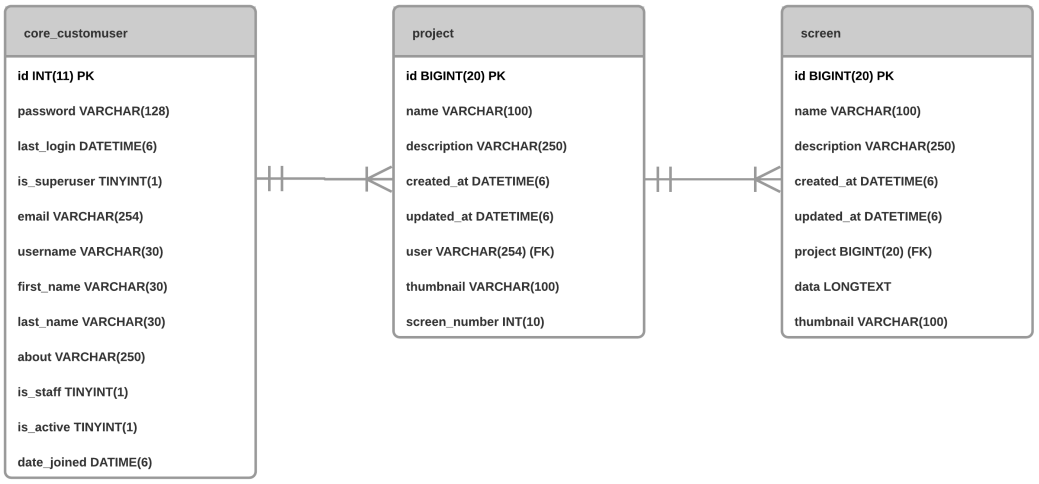

FIGURA 5.1: Diagrama de Entidad-Relación.

Como se aprecia en la figura 5.1, y obviando las tablas auto-generadas por *Django*, estas son las tablas resultantes:

- **core_customuser**
- **project**
- **screen**

Hay que destacar que la entidad Componente, presentada en el modelo de dominio de la figura 2.3, no está presente en este diagrama Entidad-Relación, y por lo tanto no será parte de la base de datos. Se ha decidido incorporar los componentes en la tabla *screen*, que representa la pantalla, almacenados en el atributo *data* como código *HTML*. A la hora de representar la información en pantalla, se hará el parseo correspondiente, leyendo todos los componentes almacenados en el atributo *data*.

5.2. Diagrama de clases

A continuación en la figura 5.2, se muestra el diagrama de clases del proyecto. Como en el diagrama anterior, se ha omitido todo lo relacionado con las clases creadas automáticamente por el *framework de Django*. En el diagrama siguiente únicamente aparecen las clases que han sido modificadas o creadas, entre las que se encuentran: la clase que

representa al Usuario, la que representa al Proyecto, la que representa a la Pantalla y las restantes clases que representan los cuatro formularios que se utilizan en el proyecto.

Toda la funcionalidad restante se ha implementado mediante *scripts* en *python*, ya que es la forma en la que se trabaja en una aplicación desarrollada con *Django*.

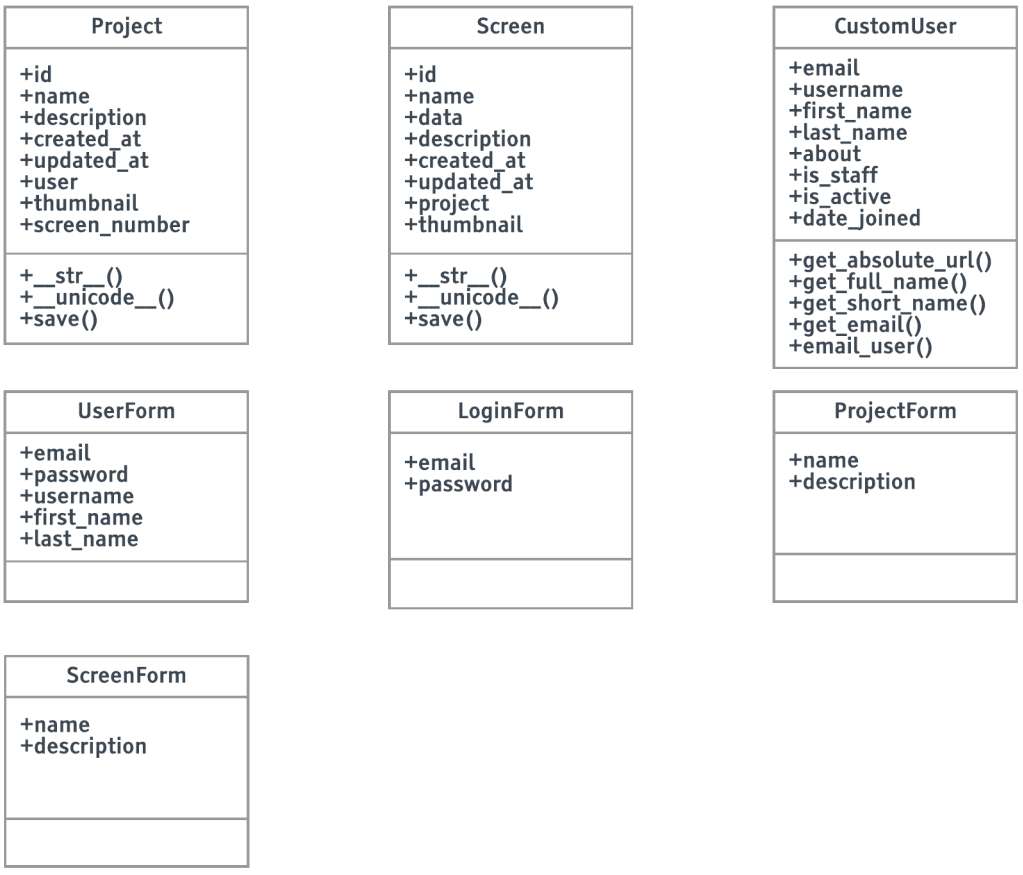

FIGURA 5.2: Diagrama de clases.

5.3. Diagrama de secuencia

Para terminar con este capítulo, en la figura 5.3, se muestra el diagrama de secuencia de una de las acciones más complejas y largas que un usuario puede realizar en la aplicación web. Además, se ha querido representar la arquitectura lógica que sigue el *framework de Django* a la hora de recibir una solicitud por parte del usuario (el navegador web). Como se puede observar en el apartado 2.6.2.2, y más concretamente en la figura 2.4, en el diagrama están presentes dos de las capas principales del *framework*, el conector

de estas con el exterior, el *urlresolver*, y el nivel más bajo de la aplicación, la base de datos.

A continuación, se muestra el diagrama de secuencia correspondiente a la acción de iniciar sesión y guardar una pantalla en la base de datos.

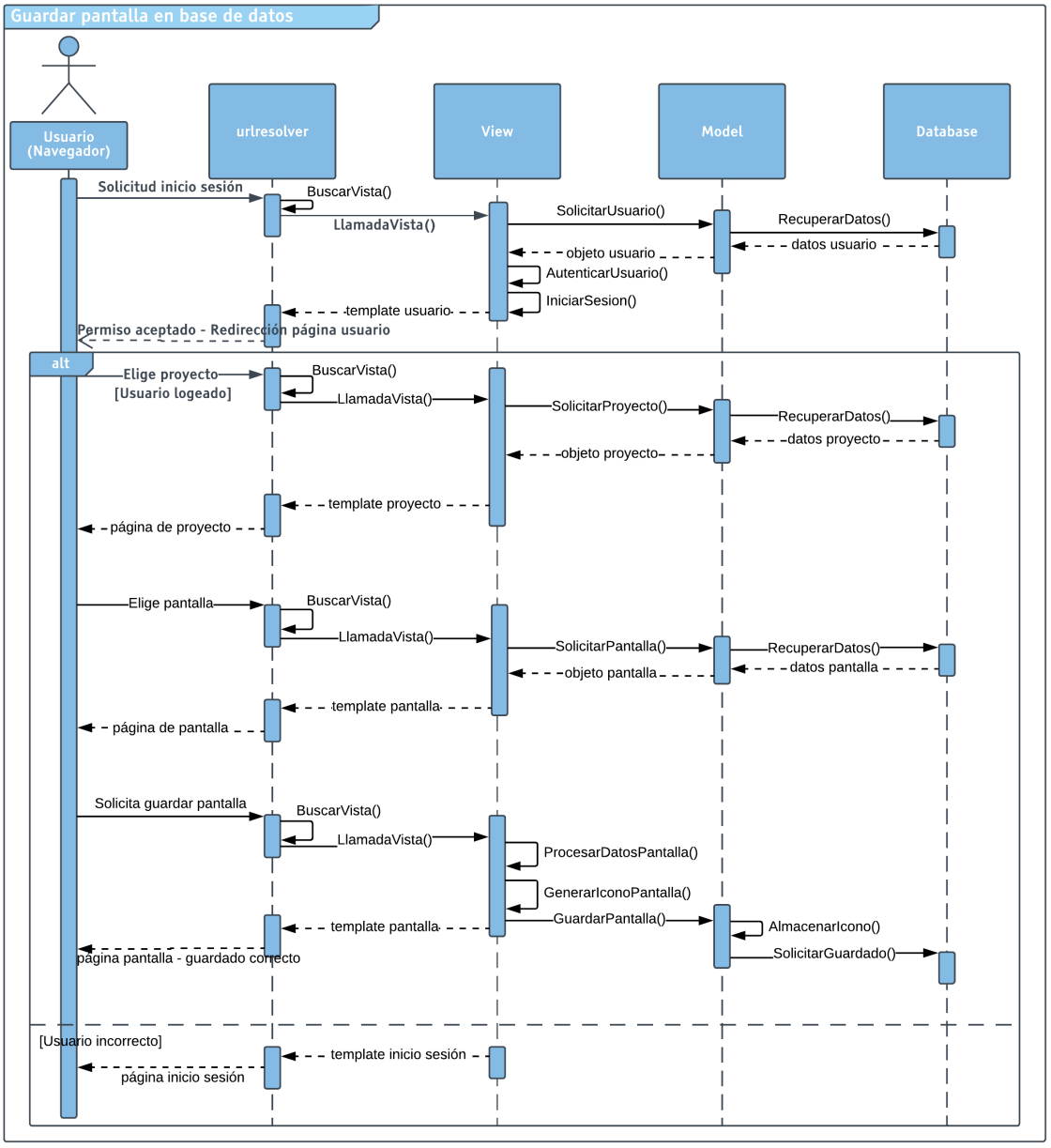

FIGURA 5.3: Diagrama de secuencia - Inicio sesión y guardado pantalla.

Como se ve en la figura anterior, en la secuencia de este diagrama, el usuario lleva a cabo cuatro llamadas a la aplicación web, la primera de ellas, es la de iniciar sesión. La respuesta a esta llamada decidirá si se podrán llevar a cabo las tres restantes o si, en

cambio, el usuario no tendrá acceso a esas llamadas. Toda petición que realiza el usuario pasa en primer lugar por el urlresolver, este mecanismo la redirige a la vista correspondiente y la vista es la encargada de realizar los cálculos y las consultas correspondientes al modelo. Una vez que la vista a realizado todas las operaciones oportunas, devuelve a través del urlsesolver el template correspondiente.

Capítulo 6

Desarrollo

En este capítulo se detallará la parte imprescindible del proyecto, y junto a la de aprendizaje, en la que más horas se han invertido. En las siguientes líneas, se explicará el progreso realizado durante la implementación de la aplicación, dividido en los puntos principales de está, se justificará el uso de algunas de las herramientas utilizadas y se expondrán las dificultades encontradas y como se han solucionado.

6.1. Framework Django

Como ya se ha repetido en capítulos anteriores, la aplicación se ha construido sobre la base del *Framework de Django*. En primer lugar hubo que estudiar esta herramienta y comprender su funcionamiento. Una vez entendido su funcionamiento principal se procedió a crear un proyecto con esta herramienta.

Este *framework* incluye la gestión de usuarios, pero el inicio de sesión se realiza mediante un nombre de usuario y contraseña. Se decidió sustituir el comportamiento original de *Django* respecto al inicio de sesión para que se pudiese hacer mediante email y contraseña. Para lograrlo, se creó un nuevo Modelo de Usuario que sustituyese al original que tiene *Django*, de manera que el inicio de sesión se hiciese mediante el email del usuario.

Un proyecto de *Django* se divide en aplicaciones, se pueden crear tantas aplicaciones como se desee, y más tarde incluirlas al proyecto utilizando el archivo de configuración

de este. Una aplicación puede referirse únicamente a una funcionalidad extra de una aplicación web, no tiene por que ser una aplicación web completa.

Para crear el nuevo Modelo de Usuario que incluía el inicio de sesión con el email se creó una nueva aplicación, a la que se llamó *core*. Además, se creo otra aplicación, *telegram_screen_designer*, que albergaría la funcionalidad base de este proyecto: la creación proyectos y pantallas y el diseño de estas últimas. El desarrollo de esta funcionalidad se explicará más adelante.

6.1.1. Función de *logging*

Al proyecto de *Django* se ha añadido la funcionalidad de *logging*. El objetivo de añadir esta funcionalidad no es otro que el de poder registrar toda actividad que ocurra en la aplicación en tiempo de ejecución. De esta manera se podrán detectar fallos futuros y optimizar la aplicación.

La inclusión de esta funcionalidad al proyecto se ha realizado mediante el módulo que el propio *framework de Django* trae integrado. Este módulo facilita enormemente el manejo de registros, ofrece diferentes niveles de registro, múltiples destinos de los registros, distintos formatos y filtros.

La configuración de este módulo ha sido bastante sencilla. En primer lugar se ha tenido que definir la configuración de la funcionalidad en el archivo *settings.py* del proyecto. Una vez realizada la configuración, con los distintos *loggers* y sus destinos definidos, se ha tenido que obtener el *logger* ya configurado en cada *script* y realizar una llamada a ese *logger* con el nivel de registro deseado por cada mensaje que se ha querido registrar.

Con los niveles de registro se ha conseguido filtrar los mensajes según la importancia o el objetivo de estos. En este proyecto se han utilizado tres niveles de registros:

- *DEBUG*: Este nivel de registro se ha utilizado a la hora de desarrollar el proyecto. Se han añadido mensajes con este nivel para hacer pruebas y depurar la aplicación.
- *INFO*: En este nivel se han registrado mensajes que entran dentro del flujo de correcto funcionamiento de la aplicación. El objetivo del registro de estos mensajes es el de poder ver que ocurre en la aplicación en todo momento.

- *ERROR*: Con este nivel se ha pretendido registrar todo error ocurrido en tiempo de ejecución y poder corregirlo, gracias a la información obtenida en el registro.

Además de los niveles de registro, se ha utilizado también, dos diferentes destinos para estos registros: la consola de comandos y el archivo de texto. La consola de comandos se ha utilizado a la hora de desarrollar el proyecto y en el momento de hacer las pruebas. El archivo de texto se ha utilizado como registro fijo, una vez que la aplicación ya se ha desplegado en el servidor.

6.1.2. Multidioma

Se ha querido que la aplicación sea multidioma, incluyendo como idiomas: Castellano, Euskera e Inglés. Para ello se ha utilizado el módulo de *internacionalización* que por defecto viene activado en el *framework*. Gracias a este módulo, es relativamente sencillo preparar la aplicación para que sea fácilmente traducida a los idiomas deseados.

En primer lugar se han definido los idiomas que se quieren ofrecer en el archivo *settings.py* del proyecto, en este caso: 'es', 'eu' y 'en'. Después, en el código de la aplicación, hay que marcar cada texto que se quiera traducir. Para ello, se ha escrito el texto en el idioma base, en este caso se ha elegido el Inglés, y se ha encapsulado en una función. Gracias a este marcado de los textos, mediante una utilidad propia de *Django*, se ha generado un archivo por cada idioma. Estos archivos recogen todos los textos marcados para ser traducidos y permiten escribir, junto a cada texto, la traducción al idioma del archivo. Por último, y con los archivos ya traducidos, el motor de *Django* leerá en tiempo de ejecución el archivo del idioma elegido por el usuario para ofrecerle los textos acordes con su elección. En la figura 6.1 se muestra una captura de pantalla de la aplicación con la funcionalidad mencionada.

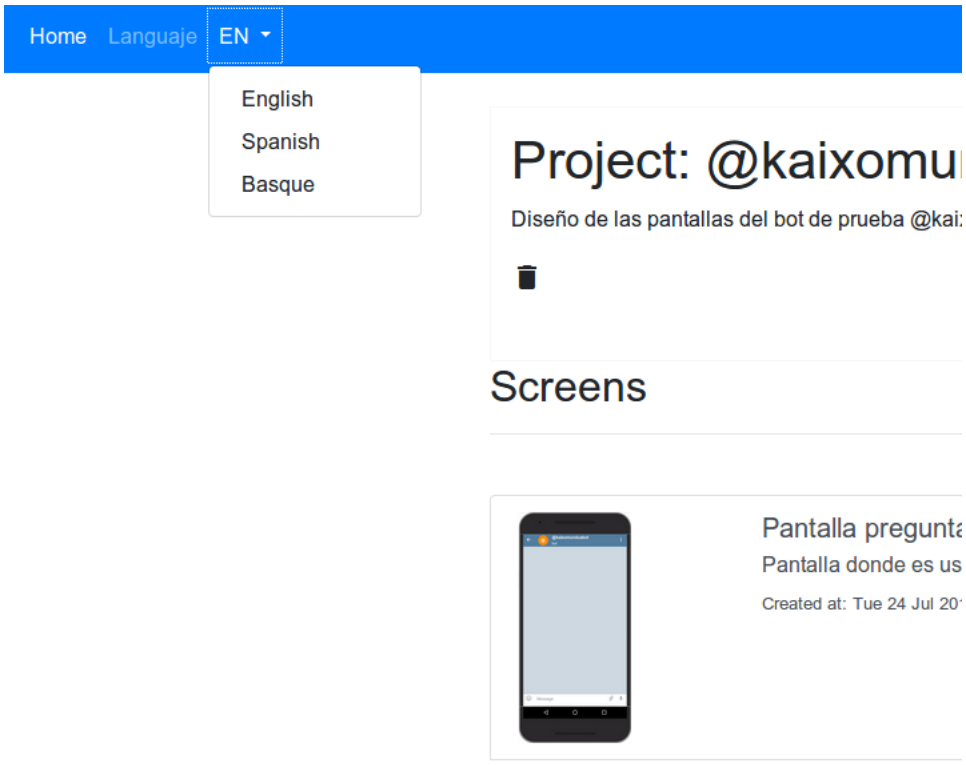

FIGURA 6.1: Elección de idioma en la aplicación web.

6.1.3. Estructura de archivos

La figura 6.2 muestra una visión general de los directorios y archivos incluidos en el proyecto.

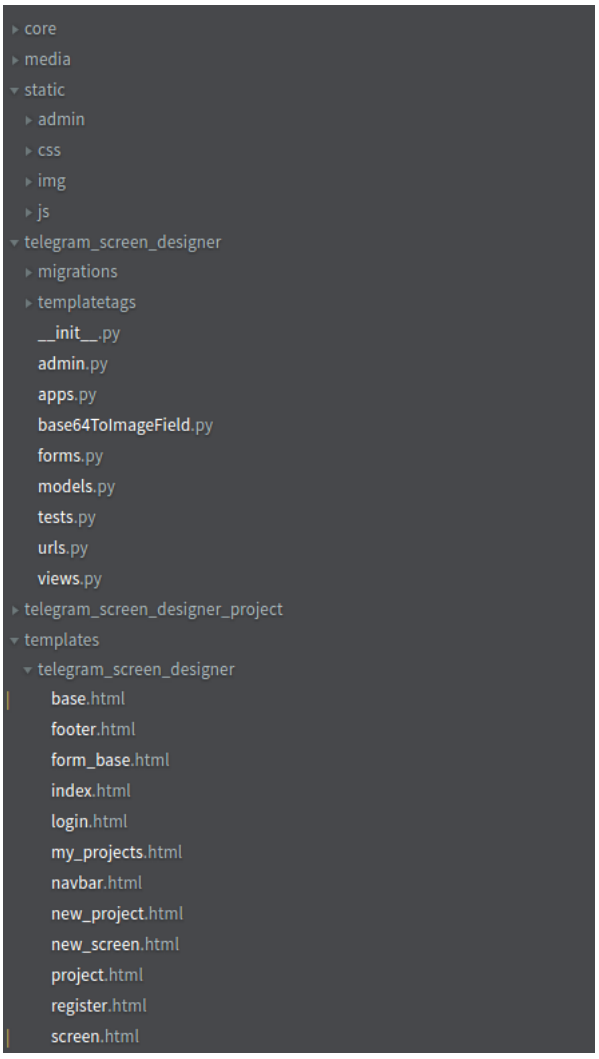

FIGURA 6.2: Estructura de archivos del proyecto.

6.2. Base de datos

El motor de base de datos utilizado para este proyecto ha sido *MySQL*. *Django* ofrece múltiples motores incluidos en su núcleo, pero la elección de *MySQL* ha venido motivada por ser de tipo relacional, por su gran extensión y facilidad de instalación en múltiples plataformas, por su completo soporte de la funcionalidad *SQL* y por incluir muchas características de seguridad

La configuración de la base de datos se define, al igual que otros funcionalidades comentadas más arriba, en el archivo *settings.py* del proyecto. Se ha tenido que definir el motor de base de datos, el nombre de la base de datos y los datos de acceso.

Gracias al *framework de Django* no ha sido necesario utilizar el lenguaje *SQL* para la creación de la base de datos. Este *framework* ofrece una potente herramienta para convertir los modelos definidos en el lenguaje *python*, dentro del archivo *models.py* de la aplicación, en tablas y relaciones.

Además, tampoco ha sido necesario usar *SQL* en la capa de Vista, a la hora de realizar consultas a la base de datos. Las consultas a la base de datos se han realizado siguiendo los métodos que el propio *framework* ofrece para ello, y que nos aíslan de interactuar directamente con la base de datos. El propio método se encarga de traducir la petición realizada a *SQL*, consultar a la base de datos, recibir la información y convertir la respuesta al objeto en el lenguaje *python* oportuno.

6.3. Funcionalidad *Proyectos y Pantallas*

En esta sección se explicará en detalle los pasos seguidos en la funcionalidad implementada para la gestión de proyectos y pantallas de la aplicación web.

Para introducir y clarificar en qué ha consistido el desarrollo de esta funcionalidad, a continuación, se enumerarán las cinco páginas principales que el usuario se va a encontrar y se hará una breve explicación de cada una de ellas.

- Página de usuario: esta es la página a la que se le redirige al usuario una vez que ha iniciado sesión en la plataforma. En esta página, el usuario puede ver los proyectos creados y crear uno nuevo. Los proyectos se listan ordenados por fecha de modificación y junto a cada nombre de proyecto se muestra un icono, la fecha de creación y de modificación y el número de pantallas que contiene. Además, haciendo clic en cada proyecto accede a la página de proyecto correspondiente.
- Página de crear proyectos: esta página contiene un formulario con los datos necesarios para crear un proyecto.
- Página de proyecto: en esta página, el usuario puede ver las pantallas creadas dentro del proyecto, borrar el proyecto y las pantallas asociadas y crear una nueva pantalla. Al igual que en los proyectos, las pantallas se muestran en una lista, ordenadas por fecha de modificación. Junto al nombre de la pantalla se muestra

un icono en miniatura del diseño de la pantalla, la descripción y la fecha de creación y actualización. Haciendo clic en cada una de las pantallas se accede a la pantalla de diseño.

- Página de crear pantalla: con esta página, el usuario puede crear una pantalla asociada al proyecto en el que se encuentra.
- Página de diseñar pantalla: en esta página se centra la funcionalidad principal de la aplicación, que no es otra que el diseño de conversaciones con un *Chatbot*. El usuario puede diseñar la conversación, función que se explica más en profundidad en la sección 6.4, pero también puede realizar algunas acciones más, a continuación se enumeran todas ellas:
  - Guardar el diseño realizado.
  - Exportar el diseño de la pantalla en formato imagen *png*.
  - Obtener una vista preliminar de la pantalla diseñada.
  - Borrar la pantalla.
  - Navegar a las pantallas anterior y siguiente del proyecto.
  - Crear una nueva pantalla.

Cabe destacar que el acceso a estas cinco páginas es restringido y solo es posible acceder una vez que el usuario se ha registrado e iniciado sesión en la web. Además de estas cinco páginas, existen tres más, no se entrará en detalle, ya que son páginas comunes presentes en toda web que implementa la gestión de usuarios. Estas páginas son:

- Página de inicio, o *home* de la web.
- Página de registro de usuarios.
- Página de inicio de sesión.

Una vez que se tiene una idea más clara de las páginas creadas para lograr toda la funcionalidad, se va a detallar el proceso seguido para lograrlo.

La primera funcionalidad en la que se empezó a trabajar fue la de la gestión de usuarios. Con el Modelo de Usuario personalizado ya implementado, se procedió a crear las páginas de registro e inicio de sesión. Se crearon dos formularios, uno para cada página, en el

archivo *forms.py* del proyecto. En este archivo se definen, de forma similar a en el archivo *models.py* mencionado anteriormente, clases que representaran cada formulario. En cada clase se indica a que Modelo pertenece el formulario que se esta definiendo y los campos que se van a solicitar. La ventaja de crear los formularios de esta manera radica en la facilidad de guardado en la base de datos y los controles que trae implícitos a la hora de comprobar los datos introducidos por el usuario.

Tras la definición de los formularios, el siguiente paso fue crear una vista, una plantilla y el nexo de unión entre todos los elementos, una *url*, para cada una de las páginas. Esta secuencia de acciones se repitió por cada página nueva que se fue creando, ya que es la forma de trabajar con el *framework de Django*. En ese momento, ya se tenia una aplicación web con página de inicio, página de registro y página de inicio de sesión.

En este punto, el código necesario para convertir los formularios a código *HTML* estaba repetido en las dos plantillas que representaban las páginas de registro e inicio de sesión. Más tarde se vera como se unificó ese código para todos los formularios de la aplicación.

Desde el primer momento se creó una plantillas base, donde se alojó el código *HTML* base para toda página de la aplicación web, incluyendo la cabecera y el pie. A su vez, la cabecera y el pie, se alojaron en plantillas diferentes. De esta forma, a cada página nueva que se iba creando, se le indicaba que heredara de la plantilla base, y solo hacia falta escribir el código necesario para esa funcionalidad.

Las siguientes páginas que se implementaron fueron la página de usuario y la página crear proyecto. La página de usuario es la primera página que se encuentra el usuario al iniciar sesión, y en ella puede ver la lista de proyectos creados. En un principio no se incluía nada más que el nombre del proyecto, pero más adelante, se incluyeron datos extra por cada proyecto, ya mencionados más arriba. Además, desde un principio, también se incluyo un botón para crear un nuevo proyecto.

La página de crear proyecto consistió, desde un principio, en un simple formulario con los campos necesarios para crear un proyecto. En la figura 6.3 se puede ver el botón para crear un nuevo proyecto, la opción de crear un proyecto nuevo o la lista de todos los proyectos creados.

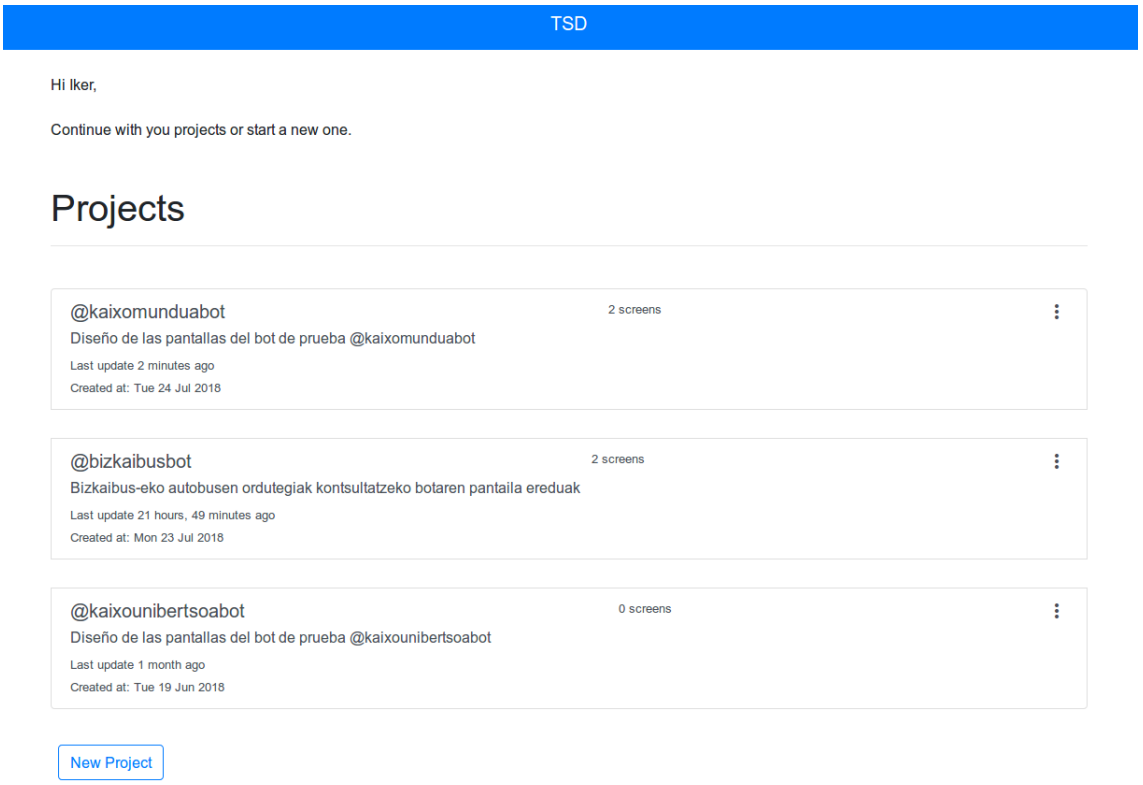

FIGURA 6.3: Página de usuario en la aplicación web.

Después, se creo la página de proyecto y la página de crear una nueva pantalla. Ninguna de las dos tuvo ninguna dificultad añadida, ya que su funcionamiento es similar a la página de usuario y de crear proyectos. Salvo mostrar una imagen junto al nombre de la pantalla del progreso de diseño de la pantalla en miniatura, que se agregó más tarde. Para conseguirlo, se buscaron diferentes soluciones, y al final, se decidió guardar una imagen *png* por cada pantalla en el servidor, sobrescribiendo esta cada vez que el usuario guardaba la pantalla. Otra funcionalidad nueva que se añadió, fue la de borrar el proyecto. Con este botón se permitió borrar el proyecto y las pantallas asociadas a este.

En la siguiente figura, se muestra la página de proyecto, donde se pueden ver todas las pantallas, acceder a ellas, crear una nueva o borrar el proyecto y todas las pantallas asociadas.

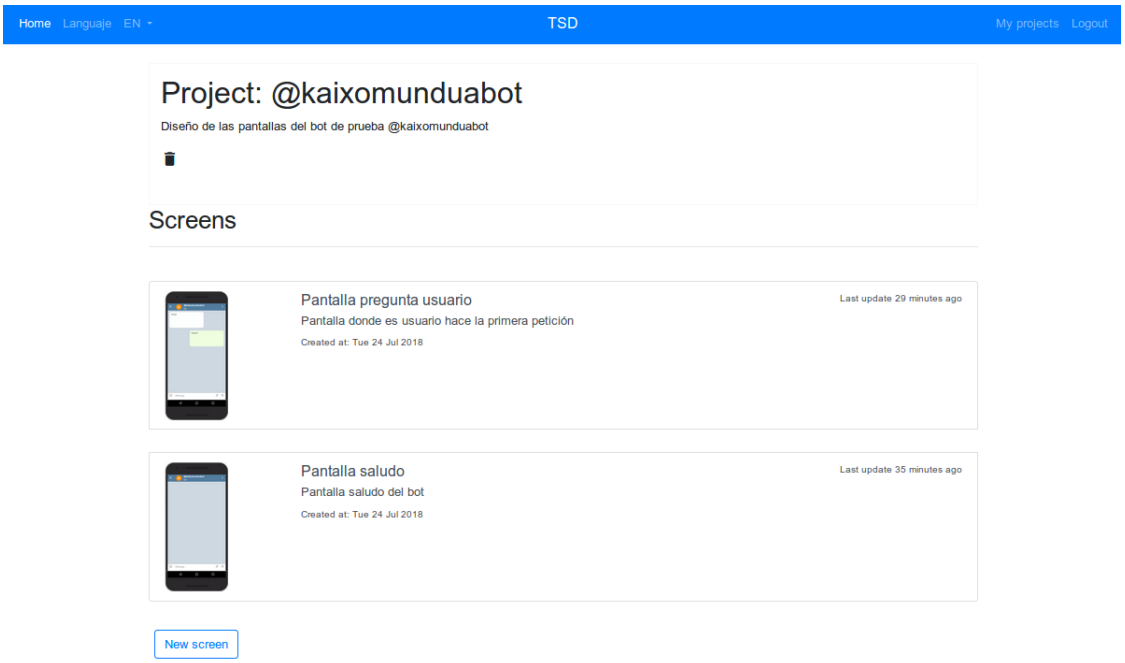

FIGURA 6.4: Página de proyecto en la aplicación web.

En este momento, y con cuatro formularios ya creados, se pensó la manera de reducir el código necesario para la representación *HTML* de cada uno de ellos, ya que era bastante similar. La solución fue crear una plantilla base, *form_base.html*, de manera que todas las plantillas con formularios la incluyeran. En la siguiente figura, se muestra como ejemplo de los formularios descritos, el resultado final del formulario de crear una cuenta.

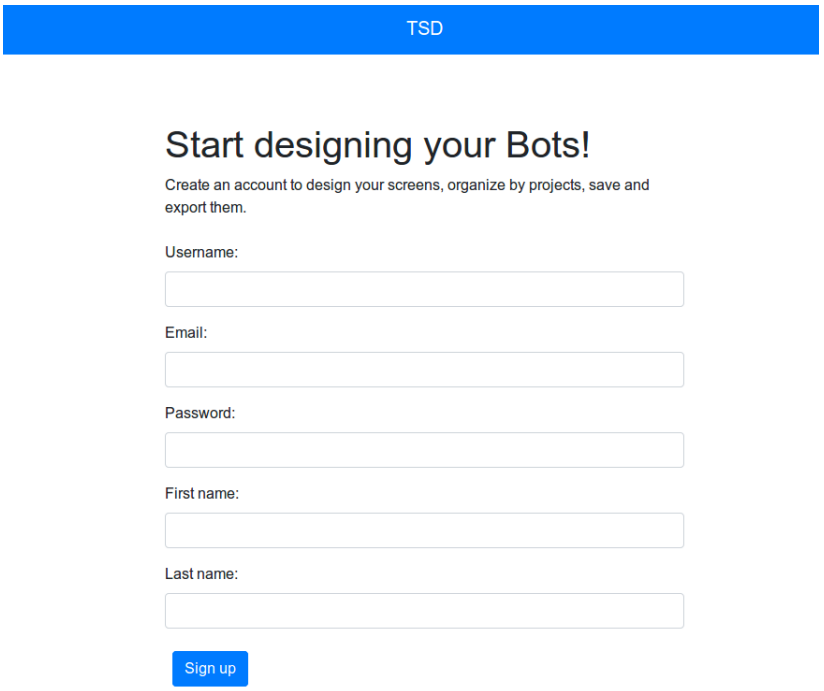

FIGURA 6.5: Creación de una cuenta de usuario en la aplicación web.

Por último, se creó la página de diseñar la pantalla. En esta página, como se vera en la sección 6.4, se incluyo el prototipo que se había desarrollado paralelamente con los elementos del *Chatbot* y el contenedor donde se agregaban estos.

A parte de esta funcionalidad, se incluyó la opción de guardar la pantalla. Esta funcionalidad supuso alguna dificultad, en un primer momento se empezó guardando el *HTML* del contenedor directamente en un campo de la tabla *screen* de la base de datos. Más tarde, hubo que modificar la forma de guardar los datos. Al guardar solamente el *HTML*, no se estaban guardando los valores de de los elementos incluidos en el contenedor: mensajes de texto, teclados en línea y teclados personalizados. Estos están representados por etiquetas *input* de *HTML*, y si no se accede directa e individualmente a cada valor, no se guardan por sí solos. La solución fue crear una nueva estructura de guardado, representada en formato *JSON*, donde, por un lado, se almacena el *HTML* de todo el contenedor, y por otro lado, se almacenan los valores de cada elemento, representados por un identificador único.

Otra funcionalidad incluida fue la de exportar la pantalla en formato imagen *png*. Para ello se utilizo, la librería *html2canvas*. Gracias a esta, se pudo convertir el *HTML* en imagen con unas pocas líneas de *JavaScript*.

Además de estas dos funcionalidades, se incluyeron en la parte final del proyecto, otras tres más sencillas: la pre-visualización de la pantalla, el borrado de esta y la navegación entre las pantallas del proyecto.

En la figura 6.6 se puede ver la página de diseño de pantallas. En esta, se pueden observar las funcionalidades descritas anteriormente como una barra de navegación con iconos en la parte superior derecha de la página. Además también está presente la barra de herramientas y el contenedor que simula el dispositivo móvil que se explicará en la siguiente sección.

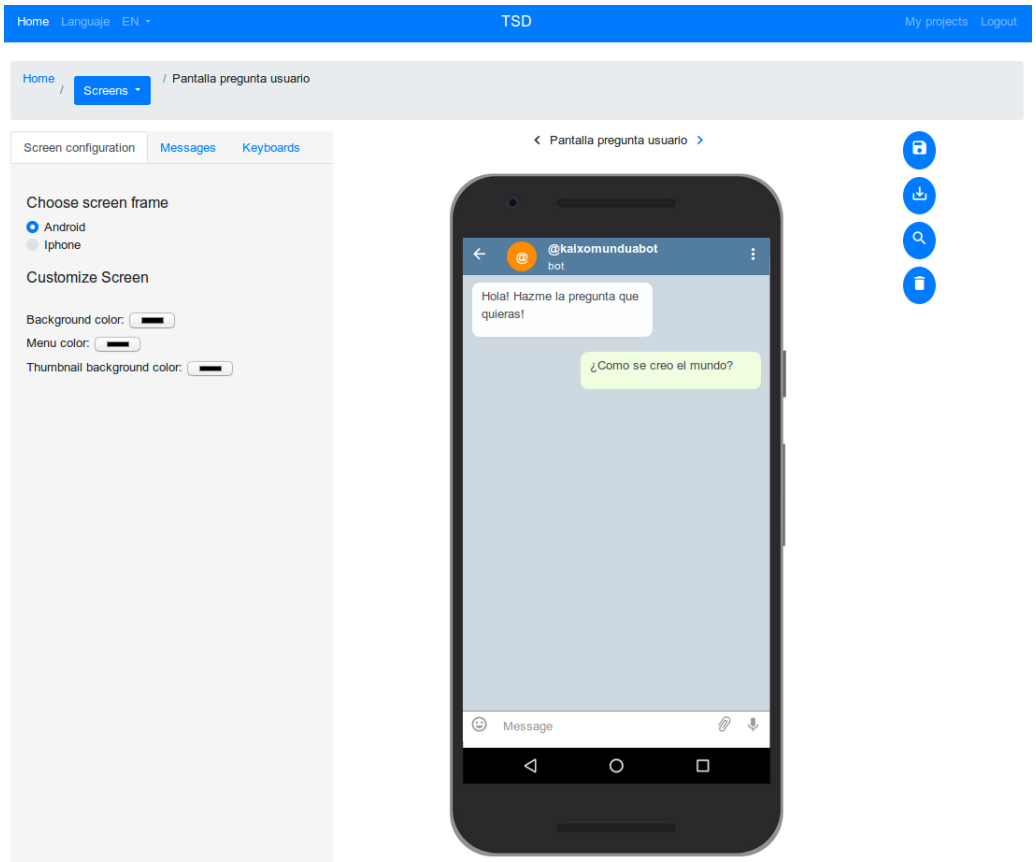

FIGURA 6.6: Diseño de pantallas en la aplicación web.

6.4. Pantalla diseño conversaciones

Esta parte del desarrollo ha sido una de las más laboriosas. En un principio se empezó a desarrollar un prototipo para diseñar las pantallas fuera del la instalación base de *Django*, de hecho, fue antes de hacer la instalación base del *framework*. Para este prototipo no se utilizó tampoco el *Framework de Bootstrap*, únicamente se trabajo con el *HTML* y los estilos *CSS*.

Este prototipo consistía en una contenedor, representado en *HTML* por la etiqueta *div*, donde se añadían cajas de texto desde una barra de navegación. Estas cajas de texto pretendían simular los mensajes dentro de una conversación. Para añadir las cajas de texto al contenedor, se utilizo *JavaScript*, utilizando un evento asociado a un botón creado para añadir la caja de texto al contenedor.

Más tarde, se añadió un marco al contenedor, de esta manera, el contenedor se convirtió en la pantalla de un *smartphone*. En la barra de navegación, se añadió la opción del tipo de marco que se quería utilizar, dan dos opciones: un marco para simular un dispositivo *Apple*, y otro para un dispositivo *Android*.

Con la visión del contenedor mucho más parecida a la de un *smartphone*, se quiso ofrecer una forma más interactiva de añadir las cajas al contenedor y se decidió utilizar la función de *Drag & Drop*. En este momento se añadió la librería *jQuery* al proyecto. Esta librería ofrece la función de *Drag & Drop* de una manera fácil de usar. Con está opción añadida, el usuario ya tenía la posibilidad de arrastrar la caja de texto al contenedor con el ratón. Además, se añadió otro elemento de *jQuery* al contenedor, la función *sortable*. Con esta función se ofrecía la posibilidad de re-colocar las cajas de texto dentro del contenedor.

El siguiente paso fue implementar la función de crear teclados personalizados y añadirlos al contenedor. Este paso fue uno de los más complicados, ya que requería generar código *HTML* dinamicamente, dando respuesta así a las peticiones que el usuario hacia de añadir más botones al teclado. Este comportamiento se solucionó mediante *JavaScript*. Una vez que el usuario había creado la estructura del teclado y añadido el texto deseado a cada botón, podía añadirlo al contenedor mediante un botón. La complejidad de este punto no radicó únicamente en la generación del contenido, la aplicación de estilos *css* fue un punto complicado, ya que se tuvo que lograr mantener la uniformidad del teclado para las filas con distinto número de botones.

Con la funcionalidad del teclado creada, se procedió a dar todavía más realismo al contenedor que representaba la pantalla del *smartphone*. Para lograrlo se incluyó la cabecera y la barra inferior que aparecen en todas las conversaciones de la aplicación de *Telegram*.

En ese momento, y con las tres partes principales del contenedor añadidas (cabecera, zona de mensajes y barra inferior), surgió un problema al añadir el teclado a la pantalla. Al añadir el teclado, que se debe situar entre la barra inferior y el marco, se descolocaban las partes y no ocupaban todo el espacio disponible en el contenedor. Para dar solución a este problema, se investigaron las posibles soluciones y se descubrió la herramienta idónea: *CSS Flexbox*. Esta función viene incluida en las últimas versiones de *CSS* y consiste en una nueva forma de organizar los elementos. Mediante esta función se pudo tener un control total del comportamiento, tanto horizontal, como vertical de los elementos dentro de un contenedor, y solucionar así el problema comentado.

Tras una reunión con el tutor, y al ver que el prototipo se acercaba al resultado esperado, se decidió incluirlo en la aplicación base que paralelamente se estaba desarrollando. En este punto el prototipo pasó a ser parte de la funcionalidad de la aplicación y se siguió mejorándolo.

El siguiente paso fue el de ampliar los tipos de mensajes que se podían añadir al contenedor. Se implementaron los mensajes de tipo imagen y de tipo audio. Además se añadió la opción del lado al que añadir los mensajes, dependiendo si el que hablaba era el usuario o el *Bot*.

Se decidió incluir también la opción de teclados en línea, que ofrece *Telegram* para los *Bots*. Para esta implementación se reutilizó la programación ya creada para los teclados normales y únicamente hubo que añadir estilos nuevos para dar el aspecto deseado.

Por último, se incluyeron más opciones de personalización de la pantalla, tales como: el cambio del color de la cabecera, la posibilidad de cambiar el nombre de usuario, el color de la imagen de perfil y la opción de cambiar el color de fondo. Además de ampliar las opciones de personalización, se fue mejorando el aspecto de los elementos ya implementados para acercarlos al diseño de la aplicación de *Telegram*.

6.5. Maquetación de la web

Para la maquetación de la web se usó el *Framework de Bootstrap*. La utilización de esta herramienta, ha facilitado en gran medida esta tarea, gracias a esta, se han podido añadir elementos vistosos y amigables para el usuario como los formularios, los botones, las ventanas emergentes, las barras de navegación, las listas, etc.

Todos los elementos mencionados cuentan con la característica de ser *responsivos*, es decir, se adaptan automaticamente al tamaño de la pantalla desde la que el usuario está visualizando la aplicación.

La manera de incluir las funcionalidades de *Bootstrap* a las páginas web, ha sido mediante la inclusión de clases ya predefinidas en este *framework* al *HTML* de las páginas del proyecto. Estas clases tienen unos estilos *CSS* asociados y en algunas ocasiones sirven también para añadir una funcionalidad determinada que el *framework* implementa utilizando la librería *jQuery*. Además de las clases, en muchas ocasiones, ha sido necesario incluir una estructura *HTML* fijada por *Bootstrap*, de otro modo el comportamiento no era el esperado.

Además, se han utilizado los iconos de *Material Design de Google*, para dar un mejor aspecto a la página web y con el objetivo de que sea más intuitiva para el usuario.

A parte de la utilización de las funcionalidades de *Bootstrap*, ha sido necesaria la creación de una hoja de estilos *CSS* propia. Durante el desarrollo del proyecto, se ha utilizado la Herramienta de Desarrolladores de *Firefox* para modificar los estilos de cada página y obtener el resultado deseado, una vez se había obtenido el estilo deseado, se copiaban los estilos creados y se trasladaban a la hoja de estilos del proyecto.

Capítulo 7

Pruebas

Durante todo el proceso de desarrollo, continuamente se han ido testeando y se ha asegurado el correcto funcionamiento de todos los elementos de la aplicación web. Para ello, se ha utilizado la funcionalidad de registro o ***logging*** que proporcionan tanto la ***Herramienta de Desarrolladores de Firefox***, como el propio ***framework de Django***.

Además de estas dos herramientas, *Django* ofrece una utilidad de depuración. Con esta utilidad, en el momento que una llamada realizada mientras se navega por la web produce un error, se muestra una página de error con infinidad de detalles sobre la petición y el error que ha generado esta. Para activar esta opción es suficiente con incluir la siguiente línea en el archivo *settings.py* del proyecto.

```
DEBUG = True
```

Este depurador ha sido activado durante la etapa de desarrollo y desactivado en el despliegue al servidor. Modificando la siguiente línea en el archivo *settings.py*.

```
DEBUG = False
```

Al ser una aplicación web, muchas de las pruebas han tenido que ver con la interfaz gráfica. Estas pruebas no han sido documentadas ya que se han ido corrigiendo según se iban desarrollando. Se ha contado con la ayuda de la ***Herramienta de Desarrolladores de Firefox*** para comprobar los estilos *CSS* aplicados y comprobar su visión en diferentes tamaños de pantallas, pero el ojo humano ha sido la herramienta principal que ha permitido detectar y corregir los fallos.

En las siguientes secciones se van a detallar algunas de las pruebas realizadas con las dos herramientas de testeo mencionadas. Hay que destacar que solo se van exponer algunas de las pruebas, ya que su metodología ha sido similar para todas ellas.

7.1. Pruebas consola Herramienta de Desarrolladores de Firefox

Relacionadas con el uso de la *Herramienta de Desarrolladores de Firefox*, se van a exponer las pruebas realizadas para una de las herramientas más complicadas de desarrollar en relación con la interfaz gráfica: el **generador de teclados personalizados**.

La forma de registrar los mensajes que nos ha permitido comprobar que está ocurriendo en la ejecución del código *JavaScript* se ha realizado mediante la siguiente función:

```
console.log("Cualquier mensaje");
```

Para demostrar el funcionamiento de esta prueba, se va a construir un teclado personalizado con tres filas, en la dos primeras tres botones y en la última, dos botones. Para la correcta construcción de teclado se requiere que cada fila y cada botón tengan un identificador único dentro de cada pantalla. De esta manera se asegurará el correcto guardado y recuperación del teclado.

A continuación, se muestran las diferentes sentencias introducidas en las funciones *JavaScript* para lograr su visualización en tiempo de ejecución.

```
$('#add-row button').click(function(){
...
    console.log("Row added");
    console.log("Row number: " + next_row);
...
}

$('#common-keyboard-generator').on('click','.add-button',function(){
...
```

```
    console.log("New button");
    console.log('Button number: ' + input_id_counter);
...
}
```

Tras haber realizado las acciones necesarias en la interfaz de usuario para lograr el teclado deseado, en la figura 7.1 se muestran los registros obtenidos en la terminal de la *Herramienta de Desarrolladores de Firefox*.

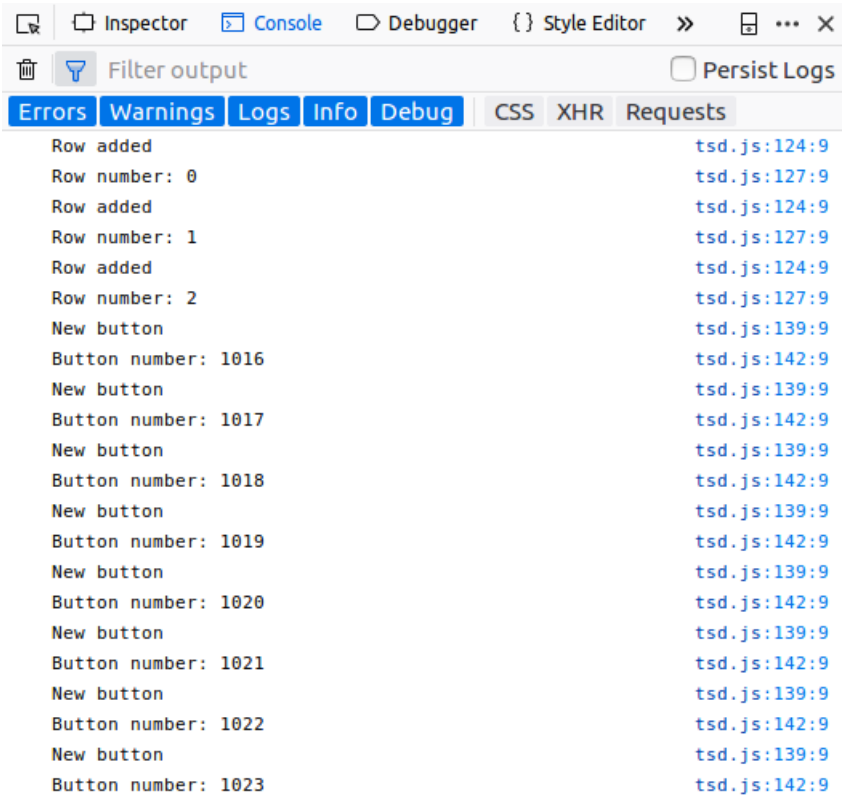

FIGURA 7.1: Ejemplo registro consola navegador.

Como se puede ver en la secuencia de registros de la imagen anterior, las acciones que se han llevado a cabo la construcción del teclado han producido los registros deseados, asignando un identificador único a cada uno de los elementos, por lo que se puede determinar el correcto funcionamiento de la herramienta.

7.2. Pruebas logging Django

En esta sección, se van a exponer las pruebas realizadas para una de las funcionalidades principales de la aplicación, el **guardado de la pantalla en la base de datos**, acompañado del **inicio de sesión** obligatorio.

Para poder utilizar la herramienta de *logging* de *Django*, ha sido necesario incluir las siguientes líneas de código en el archivo de configuración del proyecto. Con estas lineas se define un registro (*loggers*), que va a tener dos destinos (*handlers*) y que va a tener dos posibles formatos (*formatters*). Además, se indica que no se quiere desactivar los registros ya existentes, mediante la opción *'disable_existing_loggers': False*. Con esto mantendremos los registros que el propio *framework de Django* realiza en tiempo de ejecución.

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '%(levelname)s %(asctime)s %(module)s %(process)d
            %(thread)d %(message)s'
        },
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'simple'
        },
        'general-file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
```

```
            'filename': BASE_DIR + "/telegram-screen-designer.log",
            'formatter': 'verbose'
        },
    },
    'loggers': {
        'telegram-screen-designer': {
            'handlers': ['general-file', 'console'],
            'level': 'DEBUG',
            'propagate': False,
        },
    },
}
```

A parte de incluir el código anterior, se a de incluir la siguiente función al inicio del *script views.py* donde se encuentra la implementación de todas las vistas.

```
import logging
logger = logging.getLogger('telegram-screen-designer')
```

Con esta línea obtendremos el objeto *logger* sobre el que haremos las siguientes llamadas en las vistas correspondientes:

```
def user_login(request):
    logger.debug("user_login view reached!")
    ...
    logger.debug("Login request. User: " + login_form.cleaned_data['email'])
    ...
    logger.debug("User " + user.email + " logged succesfully")
    ...
    logger.debug("Incorrect login data for user: " + login_form.cleaned_data['email']
    ...


@login_required
```

```
def save_screen(request, username, project_id, screen_id):
    logger.debug("save_screen view reached!")
    ...
    logger.debug("Image succesfully decoded!")
    ...
    logger.debug("Screen " + screen.id + " correctly obtained from database.")
    ...
    logger.debug("Screen " + str(screen.id) + " correctly saved into database.")
    ...
```

Con la configuración y las llamadas a el método de registrar con los mensajes fijados, se ha procedido a realizar la acción de inicio de sesión y guardado de una pantalla. Tras la simulación de acciones completada, los registros se muestran en la terminal. Además, también se almacenan en un archivo de texto, proporcionando más información por cada línea de registro (hora y fecha, módulo, proceso e hilo). A continuación se muestran los registros obtenidos en la terminal.

```
DEBUG user_login view reached!
DEBUG Login request. User: iker93@gmail.com
DEBUG Incorrect login data for user: iker93@gmail.com
DEBUG user_login view reached!
DEBUG Login request. User: iker93@gmail.com
DEBUG User iker93@gmail.com logged succesfully
DEBUG save_screen view reached!
DEBUG Image succesfully decoded!
DEBUG Screen 23 correctly obtained from database.
DEBUG Screen 23 correctly saved into database.
```

Como vemos en la líneas anteriores, se ha producido un inicio de sesión fallido de forma intencionada, y, a continuación se ha realizado un inicio de sesión correcto. Después, se ha guardado la pantalla con identificador 23.

Con estos registros, se concluye que la secuencia de acciones ha producido el resultado deseado y se puede confirmar el correcto funcionamiento de esta funcionalidad.

Capítulo 8

Conclusiones

En este capítulo se van a exponer las conclusiones del proyecto. Comenzando por la comparación entre objetivos y resultado final, se explicarán también las líneas futuras, y para terminar el capítulo se presentará la conclusión personal.

8.1. Comparación entre los objetivos fijados y el resultado final

En líneas generales, los objetivos fijados al comienzo del proyecto se han llevado a cabo con éxito. Aunque, como en todo proyecto, han existido problemas que han derivado en el cambio del objetivo inicial o en la supresión de este.

8.1.1. Planificación temporal

El plazo de entrega del proyecto al inicio de este se fijó para finales de junio, pero debido a cuestiones personales, que nada tienen que ver con las exigencias de aprendizaje o desarrollo del proyecto, la entrega final se pospuso para finales del mes de julio. En cambio, y dejando a un lado los problemas mencionados, los tiempos fijados al inicio del proyecto han sido suficientes para la realización de las tareas.

8.1.2. Objetivos

Los objetivos del proyecto han sido cumplidos en su gran mayoría, a excepción de algunos tipos de mensajes que no se han llegado a desarrollar: mensajes de archivo, de localización y de vídeo. Además de los mensajes, no se ha llegado a desarrollar la realización de peticiones de forma asíncrona mediante *AJAX*. Esta forma de realizar algunas peticiones se planteó en el inicio del proyecto y se empezó a implementar, pero debido a la complejidad de desarrollo con el *framework de Django* se descartó. Estos dos objetivos no realizados, se presentan en la siguiente sección como lineas futuras de la aplicación web.

8.2. Líneas futuras

Como en todo proyecto, durante su desarrollo o una vez terminado este, van surgiendo nuevas funcionalidades que son interesantes de incorporar o elementos desarrollados que se pueden mejorar, pero que, debido a la falta de tiempo o recursos, resulta imposible realizar en los plazos disponibles. En los siguientes puntos se presentan algunas líneas de mejora que pueden servir de inspiración en el futuro.

8.2.1. Mejorar la interfaz de usuario

Debido a que la interfaz de usuario en general no ha sido uno de los ejes principales del proyecto, tomando como base la maquetación realizada con el *framework de Bootstrap*, resulta interesante mejorar el diseño responsivo, elegir una buena combinación de colores y plantear una nuevo diseño de menús y otros elementos como listas, formularios, botones, etc.

Además de los elementos visuales, otro punto que mejoraría en gran medida la experiencia del usuario, sería la introducción de peticiones asíncronas en algunas de las funciones que realiza el usuario: borrado de proyectos y pantallas, guardado de la pantalla o exportación de la pantalla, entre otras.

8.2.2. Añadir más elementos al diseño de las conversaciones

Se considera interesante, y necesario a futuro, ofrecer más elementos para añadir a la pantalla del *Chatbot*. La aplicación de *Telegram* está en continuo crecimiento y para que la aplicación desarrollada no quede obsoleta, es necesario incorporar los elementos que se vayan añadiendo. Dejando a un lado los posibles cambios que pueda haber en la aplicación de *Telegram*, existen elementos que ya incluye actualmente la aplicación y que sería interesante implementar:

- Emoticonos
- Mensaje de vídeo
- Mensaje de archivo
- Mensaje de localización
- Pagos mediante el *Bot*
- Juegos

8.2.3. Funcionalidad de colaborar entre usuarios

La aplicación desarrollada únicamente permite que cada usuario edite sus propias pantallas. En un futuro sería muy interesante implementar la compartición de pantallas entre usuarios de la plataforma. De esta manera, dos o más usuarios podrían trabajar en un mismo proyecto y editar simultáneamente las pantallas de los *Chatbots*.

8.2.4. Automatizar las pruebas

Además de las pruebas realizadas para este proyecto, el módulo del *framework de Django* para realizar pruebas unitarias (*unittest*), proporcionaría mayores garantías a la hora de asegurar el correcto funcionamiento de la web y, sobre todo, cuando se produzcan actualizaciones de código.

8.3. Conclusión personal

La realización de este proyecto ha supuesto una gran inversión de tiempo en el aprendizaje de nuevas tecnologías y herramientas, y gracias a ello, he adquirido muchos conocimientos en el ámbito del desarrollo web. Sobre todo, me ha encantado conocer en profundidad el *framework de Django*. Además de la facilidad que ofrece este *framework* para crear rápidamente una aplicación web, me ha gustado la arquitectura que sigue y la lógica que aplica a cada una de las capas. Sin duda, es una herramienta que voy a utilizar mucho en el futuro. Otro punto que destaco del proyecto es todo lo que he aprendido sobre los estilos *CSS*. Este es un lenguaje que voy a utilizar probablemente toda mi vida y las horas invertidas en este proyecto me han dado mucha experiencia.

Haber utilizado todas estas tecnologías para desarrollar una aplicación relacionada con los *Bots* no ha hecho más que aumentar mi interés, sin duda es un campo apasionante y con una gran capacidad de crecimiento. Ha sido muy motivador formar parte de ello.

Sin embargo, soy consciente de que mi aportación ha sido mínima, y por ello, me gustaría ver crecer la aplicación siguiendo las líneas indicadas en la sección anterior, ya sea participando en los futuros desarrollos o como usuario.

Por último, me gustaría agradecer a Juanan, director de este proyecto, su paciencia y su predisposición a ayudarme en todo momento.

Apéndice A

Despliegue de la aplicación en el servidor

La siguiente configuración se ha aplicado a un servidor con una instalación de Ubuntu Server, versión 17.10. Se asume que el servidor está correctamente configurado, tiene python instalado y que el usuario con el que se va a operar tiene permisos *sudo*.

El acceso a este servidor se ha realizado de forma remota utilizando la terminal de un equipo con instalación Ubuntu 16.04.4 LTS, mediante el comando ***ssh***.

```
ssh iker@40.89.128.211
```

En las siguientes secciones se expondrán los **comandos utilizados para la instalación de todas las dependencias necesarias en servidor**, la **configuración a cambiar en la aplicación** para su funcionamiento en el entorno de producción y los **ajustes necesarios en el servidor**.

A.1. Pip

Lo primero que hay que realizar es la actualización de los paquetes del sistema y la instalación de pip, el gestor de paquetes de python.

```
$ sudo apt-get update
$ sudo apt-get install python-pip
```

A.2. Virtualenv, Django y depenencias

La instalación de Django y todas las dependencias necesarias para la ejecución de la aplicación web, se van a realizar en un entorno virtual. De este modo, los paquetes instalados estarán aislados de las otras aplicaciones que pueda haber en el servidor. En definitiva, nuestra aplicación podrá tener versiones propias de los paquetes.

Con los siguientes comandos, en primer lugar actualizamos pip y en segundo lugar instalamos los paquetes que nos permitirán utilizar los entorno virtuales.

```
$ sudo -H pip install --upgrade pip
$ sudo -H pip install virtualenv virtualenvwrapper
```

Tras la instalación de virtualenv, es necesario ejecutar los siguientes comandos para tener disponible el paquete en la terminal del sistema.

```
$ echo "export WORKON_HOME=~/Env" >> ~/.bashrc
$ echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bashrc
$ source ~/.bashrc
```

A continuación, se creará un entorno virtual para la aplicación web y se instalarán todas las dependencias necesarias en su interior utilizando el gestor de paquetes pip.

```
$ mkvirtualenv tsdapp
```

Al crear el entorno virtual, nos introduce directamente en el. Lo podemos ver como prefijo en la terminal. Para poder salir del entorno virtual y volver a entrar se tendrían que utilizar los siguientes comandos respectivamente.

```
(tsdapp) $ deactivate
$ workon tsdapp
```

A continuación, con el entorno virtual activado, instalaremos Django y todas las dependencias necesarias.

```
(tsdapp) $ pip install django
(tsdapp) $ pip install django-widget-tweaks
(tsdapp) $ pip install mysql-python
(tsdapp) $ pip install pillow
(tsdapp) $ pip install pytz
(tsdapp) $ pip install setuptools
(tsdapp) $ pip install six
(tsdapp) $ pip install wheel
(tsdapp) $ pip install uwsgi
```

Con Django y las dependencias necesarias instaladas, se procederá a la instalacion MySQL, siempre que no esté ya instalado, y la creación base de datos.

```
$ sudo apt-get install mysql-server
$ mysql_secure_installation mysql_secure_installation
```

A continuación accedemos a MySQL e introducimos las siguientes sentencias SQL para crear la base de datos, crear el usuario y dar permiso al usuario sobre la base de datos.

```
CREATE DATABASE ikerdb;
CREATE USER 'iker'@'localhost' IDENTIFIED BY 'contraseña';
GRANT ALL PRIVILEGES ON ikerdb.* TO 'iker'@'localhost' WITH GRANT OPTION;
```

El siguiente paso será instalar GIT y configurarlo para poder descargar el proyecto desde el repositorio donde se encuentra almacenado.

```
$ sudo apt-get install git
$ git config --global user.name "Iker Melero"
$ git config --global user.email "iker93@gmail.com"
```

Procederemos a generar una clave ssh mediante el siguiente comando y siguiendo las instrucciones que se muestran por consola.

```
$ ssh-keygen
```

Con la clave ssh generada, tendremos que copiarla en nuestro repositorio remoto para poder clonar el repositorio. Una vez copiada, ejecutamos el siguiente comando para clonar el proyecto.

```
$ git clone git@bitbucket.org:ikerztipot/telegram-screen-designer-app.git
telegram_screen_designer_project
```

En este momento ya tenemos el proyecto en el servidor, es hora de ajustar la configuración. En primer lugar cambiaremos el nombre a el archivo *settings_proc.py*, lo dejaremos así, *settings.py*. A continuación, modificaremos algunos parámetros.

```
#Generaremos una clave aleatoria única para la aplicación
SECRET_KEY = ''
#Añadimos la IP de nuestro servidor a la lista
ALLOWED_HOSTS = ['localhost','127.0.0.1','0.0.0.0','40.89.128.211']
#Completamos la configuración de la base de datos
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': '',
        'USER': '',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

Con el archivo de configuración correctamente ajustado, ejecutamos los siguientes comandos para crear las tablas correspondientes en la base de datos.

```
$ python manage.py makemigrations core
$ python manage.py makemigrations telegram_screen_designer
$ python manage.py migrate core
$ python manage.py migrate telegram_screen_designer
$ python manage.py migrate
```

Por último, con el siguiente comando crearemos el usuario administrador de la aplicación.

```
$ python manage.py createsuperuser
```

```
$ python manage.py collectstatic
```

Una vez que el entorno, la base de datos y la aplicación se han configurado correctamente, se procederá a la instalación y configuración del servidor de aplicaciones (uWSGI) y el proxy inverso (Nginx).

A.3. Instalación y configuración de uWSGI

En primer lugar se instalará el paquete python-dev, necesario para la posterior instalación de uWSGI.

```
$ sudo apt-get install python-dev
```

```
$ sudo -H pip install uwsgi
```

A continuación, hay que crear el directorio para almacenar el archivo de configuración para la aplicación web y el archivo en sí.

```
$ sudo mkdir -p /etc/uwsgi/sites
$ sudo nano /etc/uwsgi/sites/tsdapp.ini
```

Copiamos las siguientes líneas de configuración en el archivo tsdapp.ini recién creado. Hay modificar los valores de las variables project y uid dependiendo del nombre de la aplicación y el usuario del servidor respectivamente.

```
[uwsgi]
env = tsdapp
project = telegram_screen_designer_project
uid = iker
base = /home/%(uid)
```

```
chdir = %(base)/%(project)
home = %(base)/Env/%(env)
module = %(project).wsgi:application

master = true
processes = 5

socket = /run/uwsgi/%(project).sock
chown-socket = %(uid):www-data
chmod-socket = 660
vacuum = true
```

Una vez guardada la configuración, se creará el siguiente archivo para poder ejecutar cada vez que se inicia el sistema la configuración de nuestra aplicación automáticamente.

```
sudo nano /etc/systemd/system/uwsgi.service
```

Una vez creado el archivo, copiamos las siguientes líneas. Es necesario modificar el usuario que va después del parámetro chown con el nombre de usuario del servidor.

```
[Unit]
Description=uWSGI Emperor service

[Service]
ExecStartPre=/bin/bash -c 'mkdir -p /run/uwsgi; chown iker:www-data /run/uwsgi'
ExecStart=/usr/local/bin/uwsgi --emperor /etc/uwsgi/sites
Restart=always
KillSignal=SIGQUIT
Type=notify
NotifyAccess=all

[Install]
WantedBy=multi-user.target
```

A.4. Instalación y configuración de Nginx

Como se ha explicado en el capítulo 2.6.2.1 sobre la arquitectura, se va utilizar Nginx como proxy inverso. Lo primero que hay que hacer es instalar Nginx.

```
$ sudo apt-get install nginx
```

A continuación hay que crear un nuevo archivo de configuración.

```
$ sudo nano /etc/nginx/sites-available/tsdapp
```

Introducimos la siguiente configuración en el archivo creado. Es necesario cambiar algunos parámetros de esta configuración: server_name, root y uwsgi_pass.

```
server {
    listen 80;
    server_name 40.89.128.211;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/iker/telegram_screen_designer_project;
    }

    location / {
        include         uwsgi_params;
        uwsgi_pass      unix:/run/uwsgi/telegram_screen_designer_project.sock;
    }
}
```

Con el siguiente comando añadiremos el archivo de configuración creado a las configuraciones activas de Nginx mediante un acceso directo.

```
$ sudo ln -s /etc/nginx/sites-available/tsdapp /etc/nginx/sites-enabled
```

Para poder empezar a utilizar la configuración realizada se tienen que reiniciar tanto Nginx como uWSGI.

```
$ sudo systemctl restart nginx
$ sudo systemctl start uwsgi
```

También es necesario ejecutar el siguiente comando para permitir el acceso total a Nginx.

```
$ sudo ufw allow 'Nginx Full'
```

Por último, ejecutamos los siguientes comandos para arrancar ambos servicios en el inicio del sistema.

```
$ sudo systemctl enable nginx
$ sudo systemctl enable uwsgi
```

Con esta configuración correctamente fijada, debería de ser posible acceder a la página web a través de los nombres de dominio fijados, en este caso mediante esta URL:

`http://40.89.128.211/`

Apéndice B

Herramientas utilizadas

En esta sección se muestra un listado, junto con una breve descripción, de las herramientas, lenguajes y tecnologías utilizadas para el desarrollo del proyecto.

Python

Es el lenguaje de programación principal del proyecto. Python es la base del *framework* de Django y es también el lenguaje que se usa para programar la lógica de la aplicación. Este lenguaje se destaca por su rapidez y potencia, por su sintaxis que favorece un código extremadamente legible, por su facilidad de aprendizaje y su código abierto.

Django

Es la base de la aplicación web del proyecto. Es un Framework Web de alto nivel desarrollado en Python y de código libre. Ofrece una gran base de funciones que nos permiten centrarnos en escribir nuestra aplicación, ocupándose por nosotros de aspectos relacionados con las bases de datos, la seguridad, la eficiencia... Sigue el patrón de diseño conocido como Modelo–Vista–Template. Gracias a este diseño, y a las muchas aplicaciones gratuitas que se pueden añadir a nuestro proyecto, ofrece una gran y rápida escalabilidad.

Pip

Es el gestor de paquetes de Python. Ha sido muy usado en el proyecto, ya que es imprescindible para la instalación de todos los paquetes de software: Django, Virtualenv, Six, django-widget-tweaks, etc.

Virtualenv

Esta herramienta permite crear entornos aislados de Python. Nos permite simular uno o varios entornos de trabajo, ya sean de desarrollo o producción, sin tener que utilizar más de una maquina. En este proyecto se ha utilizado esta herramienta para simular el mismo entorno tanto en desarrollo como en producción, gracias a ello se ha podido utilizar las mismas versiones de todos los paquetes de Python en los dos entornos. Al tener dos entornos idénticos el paso de desarrollo a producción es rápido y sin ningún problema de compatibilidad.

uWSGI

Es un contenedor de servidor de aplicaciones que tiene como objetivo proporcionar una pila completa para desarrollar y desplegar aplicaciones y servicios web. Permite manejar aplicaciones de diferentes idiomas y se comunica con la aplicación utilizando los métodos definidos por la especificación WSGI y con otros servidores web a través de una variedad de protocolos. Es la pieza que traduce las solicitudes de un servidor web convencional a un formato que la aplicación puede procesar.

Se ha utilizado uWSGI como un servidor de contenedor de aplicaciones. Se encargará de recibir las peticiones que le lleguen desde el servidor proxy (Nginx) y las traducirá a un formato que la aplicación desarrollada con Django pueda entender.

Nginx

Nginx es un servidor web/proxy ligero y de muy alto rendimiento. Es multiplataforma, software libre y de código abierto. Está muy extendido y es usado en sitios web muy conocidos como: Netflix, Wordpress, Github, etc.

Se ha configutado Nginx como proxy inverso a uWSGI, dándonos acceso a sus características de seguridad y rendimiento para servir nuestra aplicaciones. Nginx redireccionará todas las peticiones que lleguen al servidor relacionadas con la aplicación web y las enviará al servidor uWSGI para que se encargue de traducirlas.

Html2canvas

Es un script escrito en Javascript que permite crear capturas de pantalla de cualquier parte de la página web. Esta herramienta va recorriendo el DOM desde el punto que nosotros fijemos y crea una imagen leyendo las propiedades del DOM y los estilos CSS.

En este proyecto se ha utilizado para crear las imágenes de las pantallas que los usuarios van diseñando.

Bootstrap

Es una librería HTML, CSS y JS de componentes front-end. Proporciona un conjunto de herramientas que facilita la maquetación web siempre con un diseño *responsive* y priorizando el diseño para móviles.

Javascript

Es un lenguaje de programación interpretado. Junto con HTML y CSS es una de las tres tecnologías principales de la Web. En este proyecto se usa, junto con jQuery, para modificar el DOM y responder a los eventos de los diferentes botones y otros elementos.

jQuery

Es una librería escrita en Javascript que simplifica muchas de sus funciones. En resumen, facilita el uso de Javascript, permitiendo hacer más escribiendo menos.

Json

Considerado hoy en día un formato de lenguaje independiente, es muy usado para la transmisión de datos, siendo muy sencillo su análisis y conversación a objetos de casi cualquier lenguaje.

Se ha usado en el proyecto como formato para almacenar las pantallas diseñadas por el usuario y su transmisión y recuperación hacia y desde la base de datos.

HTML

Junto con CSS y Javascript uno de los tres pilares de Web. Es un lenguaje de marcado, utilizado casi en la totalidad de las páginas web del mundo como forma de organizar, dar sentido semántico y en cierta medida dar aspecto a cualquier página web.

CSS

Es un lenguaje de hojas de estilo, usado para dar el aspecto deseado a cualquier documento HTML. Su creación tenia como objetivo separar la presentación del contenido, es decir, extraer de los documentos HTML todo lo relacionado con el aspecto de la página web y aislarlo en un documento independiente.

MySQL

El sistema de gestión de bases de datos utilizado en este proyecto. Se ha establecido en la configuración de Django MySQL como el sistema de base de datos ha utilizar. Además de las tablas propias que utiliza Django, ha hecho falta la creación de tablas nuevas para gestionar los proyectos y las pantallas.

Git

Software de control de versiones que aporta una mayor comodidad a la hora de actualizar y mantener el código de cualquier proyecto. La ayuda que ofrece Git marca la diferencia en lo que a mantenimiento y supervisión de los archivos fuente de un proyecto se refiere y hace el desarrollo del proyecto mucho más profesional.

Brackets

Es el editor de texto que se ha usado en este proyecto. Desarrollado en Javascript, fue fundado por Adobe y es de código abierto. Gracias a la gran cantidad de plugins disponibles, es muy sencillo ampliar su funcionalidad, como el plugin de Git, usado en este proyecto.

Firefox

Navegador web de software libre desarrollado por la fundación Mozilla. Rápido, multidioma, personalizable, seguro y con multitud de herramientas para desarrolladores. Esto último lo hace idóneo para este proyecto.

Herramientas desarrollador *Firefox*

Herramientas para desarrolladores web incluidas en el navegador web *Firefox*. Incluye funcionalidades como consola de comandos, inspector HTML y CSS, análisis de la red, simulación de diferentes dispositivos, etc.

Bitbucket

Servicio de almacenamiento y gestión de proyectos de Git y Mercurial. Bitbucket ofrece crear y guardar un ilimitado numero de proyectos, es rápido y se puede navegar por su interfaz de forma intuitiva. Se ha elegido Bitbucket en lugar del más popular Github para poder hacer uso de repositorios privados de forma gratuita.

Overleaf

Herramienta online para la elaboración de documentos con LaTeX. Permite escribir, compartir y publicar documentos profesionales fácilmente. Se ha usado para la elaboración de esta documentación.

Cacoo

Herramienta para la creación de diagramas utilizada para generar muchas de las figuras insertadas en la memoria del proyecto. Permite la creación de todo tipo de esquemas y diagramas, guardándolos en la nube y pudiendo acceder a ella desde cualquier dispositivo conectado a la red. También permite la exportación de los esquemas en muchos formatos, incluyendo el formato *.png* usado en este proyecto.

Lucidchart

Al igual que *Cacoo*, es una herramienta para crear diagramas de todo tipo. Permite también, almacenar diagramas y exportarlos. Se ha utilizado para crear algunos de los diagramas incluidos en el proyecto.

Trello

Aplicación web (con versión para móvil) para organizar proyectos. Permite estructurar las tareas por listas y tarjetas, pudiendo establecer a estas ultimas: descripciones, comentarios, vencimientos, participantes, etc. La interacción con la aplicación es muy intuitiva y amigable, permite desplazar las tarjetas entre las listas arrastrándolas con el ratón.

Se ha utilizado constantemente en este proyecto para tener una vista de las tareas pendientes y realizadas, poner comentarios necesarios durante el desarrollo y anotar las mejoras a realizar surgidas sobre la marcha.

Apéndice C

Repositorio y dirección web

C.1. Repositorio

EL código del proyecto se podrá descargar mediante el siguiente repositorio público en Bitbucket:

`https://ikerztipot@bitbucket.org/ikerztipot/telegram-screen-designer-app.git`

C.2. Dirección web

En esta URL se tendrá acceso a la aplicación web a través de cualquier navegador:

`http://40.89.128.211/`

Bibliografía

[1] Fabrizio Romano. *Learning Python*. Packt Publishing, 2015.

[2] Leiz Azzopardi & David Maxwell. *Tango with Django*. Packt Publishing, 2015.

[3] Chris Coyier. A complete guide to flexbox. Disponible en: `https://css-tricks.com/snippets/css/a-guide-to-flexbox/`, 2018.

[4] Justin Ellingwood. How to serve django applications with uwsgi and nginx on ubuntu. Disponible en: `https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-uwsgi-and-nginx-on-ubuntu-16-04`, 2016.

[5] uWSGI. Setting up django and your web server with uwsgi and nginx. Disponible en: `https://uwsgi.readthedocs.io/en/latest/tutorials/Django_and_nginx.html`, 2016.

[6] Django Software Foundation. Django official documentation. Disponible en: `https://docs.djangoproject.com/en/2.0/`, 2018.

[7] Bootstrap Team. Bootstrap official documentation. Disponible en: `https://getbootstrap.com/docs/4.1/getting-started/introduction/`, 2018.

[8] Sharelatex. Sharelatex documentation. Disponible en: `https://www.sharelatex.com/learn`, 2018.

[9] Google. Material design icons. Disponible en: `https://material.io/tools/icons/`, 2018.

[10] Niklas von Hertzen. Html2canvas documentation. Disponible en: `https://html2canvas.hertzen.com/documentation`, 2018.

[11] jQuery. jquery documentation. Disponible en: `https://api.jquery.com/`, 2018.