CLEO: Aplicación Web para gestionar Centros Estéticos

Silvia Martín Gómez, Irene Ventura Farías

Grado en Ingeniería de Software
Facultad de Informática
Universidad Complutense de Madrid

Trabajo Fin Grado
Curso 2018-2019

Director: Luis Llana Díaz
Departamento de Sistemas Informáticos y Computación

Agradecimientos

En primer lugar, queremos agradecerle a nuestro director Luis Llana principalmente por habernos dado la oportunidad para trabajar este tema como trabajo de fin de grado y por motivarnos a trabajar con nuevas tecnologías, muchas de las cuales ya forman parte de nuestro trabajo día a día.

A nuestros padres, novios y amigos, que han sufrido nuestros momentos de estrés y ansiedad, os damos las gracias por los buenos consejos y porque siempre nos habéis apoyado y nos habéis animando a continuar hasta el final. Mención especialmente a Vanessa que, pese a las horas de diferencias, siempre estuvo dispuesta a ayudarnos y aportarnos nuevas ideas.

Índice

Resumen

CLEO es una aplicación web, diseñada principalmente para manejar y controlar los servicios básicos de centros médicos estéticos, la cual presenta una interfaz amigable e intuitiva para el usuario. Ofrece un entorno dónde los empleados tienen el control de los componentes principales de la organización, como, por ejemplo, registrar nuevos clientes, realizar pedidos a proveedores, tener el control de las máquinas y salas del centro, y asignar citas a los clientes que desean disfrutas de sus servicios, etc. Igualmente, los clientes pueden registrarse por cuenta propia, a fin de solicitar citas de los tratamientos y además podrán reservar productos que deseen comprar para uso diario, ya que la aplicación está desplegada en un servidor.

La aplicación permite notificar a los encargados cuando algún producto del *stock* está por debajo de la cantidad mínima, para que los encargados para que puedan realizar un pedido. Por otro lado, los clientes recibirán notificaciones sobre nuevas promociones, las cuales notifican la fecha de duración y el descuento que tiene dicha promoción, además de recibir otra notificación cuando soliciten una cita.

Todas las operaciones se realizan a través de la interfaz web y toda la información que es generada en la aplicación es almacenada en una base de datos relacional. Para cada componente de la aplicación se realizaron pruebas de caja blanca y caja negra para asegurar el buen funcionamiento de la aplicación. También se han realizado pruebas de seguimiento con el cliente y una vez puesta en producción se realizaron encuestas en relación con la interfaz, facilidad de uso y la utilidad de la misma.

Palabras Claves: *Aplicación Web, Gestión de Centros Médicos Estéticos, Base de Datos, Notificaciones, Tratamientos, Inventarios y Clientes.*

Summary

CLEO is a web application that was mainly designed for management and control the basics services of beauty clinics, with an intuitive system and a friendly interface for all users. It offers an environment where employees have control of the main components of the organization, such as, register new clients, make orders, have control of the machines, rooms and make appointments for all the clients who want to enjoy their services, etc. Equally, customers can also register themselves, they can make appointments of the treatments that they want to try out and they can reserve the products they want to buy for daily use because the app is deployed on a server.

The app let's notifies employees when it detects that some product has a short amount on the stock, so they can order. On the other hand, customers will receive notifications about new promotions that will be publicity campaigns made by employees, which they will have the period and the discount of this.

All operations will be done through a web interface and all data will be generated from the app, it will be stored in a relational database and for each component, they will be made tests of white box and black box to ensure the good performance of the app. Also, we made monitoring tests with our client y when we deploy the app, we made forms to see about what users think about the app.

Keywords: *Web Application, Aesthetic Treatment Controller, Databases, Email Notifications, Treatments, Stocktaking and Clients*

Capítulo 1: Introducción

En este capítulo describimos las características del proyecto, de donde nació la idea para desarrollar la aplicación, la motivación que nos impulsó a querer trabajarlo en nuestro TFG y los objetivos que queríamos alcanzar una vez finalizado el desarrollo de CLEO. Igualmente, antes de empezar la recolección de los requisitos, debíamos investigar y familiarizarnos con las nuevas tecnologías que tendríamos que implementar, así como también retomar conocimientos de aquellas otras que fueron requeridas en este trabajo y que hemos utilizado durante nuestra formación académica.

Aparte, hemos querido definir la gestión de nuestro proyecto, evaluando previamente los riesgos que podían suceder durante el desarrollo de la aplicación y saber que planes de contingencia había que tomar, en caso de presentarse alguna. Además, pudimos evaluar algunas herramientas que nos permitieron trabajar en un entorno fácil de usar y multiplataforma.

1.1.- Antecedentes

Mucho antes de plantearnos a desarrollar CLEO, Josefina Farías Vera, que es una esteticista con más de 20 años de experiencia y que ha trabajado con diferentes sistemas para administrar centros de estética, nos había comentado de las limitaciones de los diferentes desarrollos de software para centros médicos estéticos. Por ejemplo, muchos de ellos, como es el caso de: SADPE3000 y Flowww, entre otros, tenían un modelo de negocio en el que los costos se incrementaban con el número de empleados y de clientes. Por otra parte, en ninguno de ellos se daba la opción de que el propio cliente pueda gestionar sus citas. En otras palabras, la mayoría de los softwares disponibles se adaptan al tamaño y características del negocio, pero ninguno tiene un enfoque dirigido al cliente. Y es que los desarrolladores de este tipo de software están enfocados en su propio negocio y no en el del cliente o centro, pues están enfocadas en empresas grandes, con un número importante de empleados. La justificación para muchas compañías desarrolladoras estribe en el hecho de que el desarrollo de una gran cantidad de funcionalidades obliga a enfocarse en empresas grandes que puedan pagar por el uso mensual de la licencia. Permitiendo además integrar la administración de diferentes centros de una misma marca. Al ver los requerimientos de Josefina, se nos ocurrió que podía ser un buen tema para nuestro trabajo de fin de grado. Luego, nos reunimos con el profesor Luis Llana, para que fuese nuestro director de TFG, a quien le pareció interesante la propuesta, pero nos comentó desde el primer momento que el desarrollo del proyecto debía darnos conocimientos nuevos que pudiesen ser útiles para nosotras en un futuro, y es por eso

que nos recomendó trabajar con un lenguaje de programación que no hubiésemos utilizado anteriormente, como era el caso de Python.

Finalmente, al tener la idea de Josefina como proyecto y la sugerencia que nos dio nuestro tutor para obtener más conocimientos, nuestro objetivo claro era realizar una aplicación de código abierto que pudiera potenciar a las empresas del sector médico y estético, a la vez que nos sirviera como trabajo de fin de grado. Dicha aplicación debía cumplir algunos parámetros, a saber, debía ser fácil de usar, segura, escalable y que se adapte a las necesidades del negocio y sus clientes, no a la estructura de la empresa y a la cantidad de empleados. En otras palabras, teníamos que desarrollar una aplicación para centros médicos estéticos que crezca a la par del negocio; al tiempo que nos permitiera adquirir nuevos conocimientos en lenguajes de programación y en herramientas de desarrollo, para éste y futuros proyectos.

1.2.- Bases Tecnológicas

Para el desarrollo de esta aplicación hemos tenido que investigar y certificarnos de varias herramientas de software, puesto que no son de aprendizaje básico en la carrera, entre las que podemos destacar:

a) Lenguaje de Programación: Python

Aparte de ser el lenguaje que nuestro tutor nos sugirió para nuestro trabajo, consideramos que sería bueno para nuestro desarrollo como programadoras. Ya que es uno de los lenguajes de mayor uso en la actualidad y se caracteriza por ser un lenguaje de programación interpretado. Esto quiere decir, que no requiere compilación y enlaces requeridos por otras herramientas, lo que incrementa la velocidad y productividad para el desarrollo de aplicaciones.

Otras características importantes de este lenguaje, es que posee una sintaxis muy ligera, lo cual permite un código legible, que nos brinda una gran ayuda a la hora de mantenimiento y también se utiliza para diferentes áreas de trabajo, como lo son los videojuegos, animaciones gráficas, desarrollo web, inteligencia artificial, robótica y muchas otras áreas.

b) Framework: Django

Como hemos mencionado antes, Python es un lenguaje de programación bastante completo, el cual tiene un framework de alto nivel llamado Django que permite el desarrollo rápido de aplicaciones web.

Django es una tecnología tan potente que proporciona herramientas bastante avanzadas; como, por ejemplo: la administración de cuentas de usuarios de forma segura; en donde las contraseñas se almacenan en un hash de contraseñas. También permite la protección de datos contra algunas vulnerabilidades como por ejemplo la inyección SQL. Igualmente, simplifica la creación, validación y procesamiento de formularios. Y finalmente permite la escalabilidad, ya que cada parte de la arquitectura es independiente del resto.

c) Entorno Ligeros de Desarrollo y Despliegue: Docker

Docker es una tecnología que consiste en empaquetar software en unidades estandarizadas llamadas contenedores y se utiliza mucho para el desarrollo y despliegue de aplicaciones.

A pesar de que era una tecnología desconocida para nosotras, nuestro tutor también nos sugirió utilizarla ya que permite ejecutar un proceso de manera totalmente aislada del resto, puesto que, con él, podemos definir todas las dependencias que necesita nuestra aplicación para ser ejecutado. Esto, ayuda a que el programa sea totalmente portable y permite trabajar desde diferentes sistemas operativos para el desarrollo de la aplicación.

Finalmente, también necesitamos de herramientas con las que hemos trabajado durante la carrera y que su uso era preciso en nuestro proyecto, a la hora de obtener mejores resultados con el producto final, como, por ejemplo:

a) Base de Datos Relacional: MySQL

Es la tecnología con la que estamos más acostumbradas a trabajar, debido a que la hemos utilizado en múltiples ocasiones (administración de bases de datos, modelado de software, aplicaciones web, etc.), ya que es un requisito obligatorio conocer el manejo y diseño de bases de datos en nuestra formación académica.

MySQL es un sistema de administración de bases de datos relacionales, la cual se basa en la organización de la información para mantener la persistencia de los datos. Es una herramienta multiplataforma, fácil de aprender y con buena documentación, es escalable y la gestión de transacciones son atómicas, consistentes, aisladas y durables (ACID).

b) Diseño de maquetación de las Interfaces: Bootstrap

Nosotras vimos una introducción al uso de esta herramienta cuando vimos la asignatura de aplicaciones web, pero utilizarla era algo opcional y finalmente no la usamos. Sin embargo, una de nosotras se certificó para utilizar esta tecnología, ya que fue un requisito en un proyecto en el que estuvo antes de este.

Bootstrap es un framework de CSS y JavaScript que nos permite incluirlo en nuestra web de forma rápida y sencilla, se utiliza para crear interfaces web que se adaptan en función del tamaño de la pantalla del dispositivo con el que estamos navegando, es decir, alcanzar un diseño responsivo (Responsive Design), permitiendo tener unas plantillas bien estructuradas visualmente.

1.3.- Metodología

Al ser un proyecto con pocas restricciones, consideramos que había que desarrollar la aplicación desde un enfoque adaptativo, realizando continuas entregas de software para que el cliente pudiese testearlo y existiera una constante comunicación con él, ya que, al ser un proceso flexible, podían existir cambios o modificaciones durante el desarrollo. Es por ello, por lo que decidimos trabajar con una metodología de desarrollo ágil, en donde los procesos son iterativos con entregas incrementales y que pueden estar sujetas a cambios.

Durante el desarrollo del proyecto podemos clasificar las iteraciones en dos fases:

a) **Fase I: Introducción a las Herramientas de Desarrollo y recolección de los primeros requisitos con el cliente.**

   i. **Desarrollo de la primera iteración**

   La primera iteración fue una entrevista con el cliente, explicándonos un poco lo que quería y mencionándonos los sistemas con los que había trabajado anteriormente, explicándonos un poco lo que no le gustaba de ellas y sugiriéndonos probar para conocer cómo funcionaba una de ellas (Flowww), la cual estaba en funcionamiento en el centro donde trabaja.

   ii. Desarrollo de la segunda iteración

   Para la segunda iteración habíamos probado el sistema con el que ella trabajaba, investigamos sobre las herramientas que íbamos a utilizar para desarrollar la aplicación, para la que realizamos prototipos que iban a servirnos al momento de desarrollar la aplicación. Y finalmente haciendo un repaso de las características que el cliente quería, además de las que le sugerimos, pudimos obtener lista de requisitos funcionales para la aplicación.

b) Fase II: Desarrollo de los requisitos, entregas al cliente y modificaciones.

i. Desarrollo de la tercera iteración

Nuestra tercera iteración de desarrollo consistía en hablar del proceso completo de nuestra aplicación. En esta iteración realizamos funcionalidades básicas: diseño de la base de datos y los usuarios que pueden acceder a ella (administrador de la BBDD y un usuario general que será el que utilice la aplicación, el cual tendrá los permisos restringidos) y boceto de las interfaces que presentara la aplicación realizadas con HTML5, Bootstrap y CSS.

ii. Desarrollo del resto de iteraciones

El resto de las iteraciones mantuvieron un protocolo similar, solo variaba en la relación que tenía un módulo con el resto de los elementos en la aplicación y el rol que podía utilizarlo; desarrollando primero los módulos más básicos y de ultimo los más complejos.

La codificación de cada módulo requería de varios pasos, y aunque es cierto que no todos los módulos cumplen estrictamente con esta serie de pasos, pero, podemos destacar los más generales, a saber:

1. Crear estructura básica del módulo y añadirlo a la aplicación.
2. Configurar el modelo de ese modulo.
3. Crear los formularios que el módulo.
4. Crear las rutas y funcionalidades básicas del módulo (CRUD).
5. Asignarle los permisos y comprobaciones de seguridad según el tipo de usuario que este visualizando la aplicación.
6. Hacer las pruebas de visualización de interfaz, verificación y validación de datos, comprobación de restricción de usuarios e independencia de módulos para mantener la escalabilidad.

1.4.- Gestión de Riesgos

Al momento de desarrollar un proyecto, era importante evaluar los posibles eventos inesperados que pueden suceder durante el desarrollo de la aplicación. Según el tipo de riesgo, es importante saber tomar las medidas necesarias, para evitar que sucedan o minimizar

su efecto sobre el proyecto. Para gestionar los riegos debemos realizar varios procesos, entre los que cabe destacar:

1. **Identificación y Análisis de Riesgos**

En este proceso se identificaron los posibles riesgos que podían ocurrir durante el ciclo de software. Una vez identificados, procedimos a evaluar la probabilidad con la que podían suceder y el efecto que podían tener cada uno de ellos, según la técnica de SQAS-SEI:

Riesgo	*Probabilidad*	*Efecto*
a. El personal necesita un tiempo extra para aprender un lenguaje de programación nuevo.	**Ocasional**	**Crítico**
b. Las personas clave sólo están disponibles una parte del tiempo.	**Ocasional**	**Serio**
c. El personal trabaja más lento de lo esperado.	**Remota**	**Serio**
d. El tiempo de comunicación del cliente (por ejemplo, tiempo para responder a las preguntas para aclarar los requisitos) es más lento del esperado.	**Remota**	**Critico**
e. Cambio excesivo de requerimientos debidos a que el cliente no tiene claro lo que quiere originan retrasos y aumento de costes.	**Ocasional**	**Critico**
f. Los requisitos no se han definido correctamente. y su redefinición aumenta el ámbito del proyecto.	**Remota**	**Critico**
g. El desarrollo de una interfaz de usuario inadecuada requiere volver a diseñarla y a implementarla.	**Probable**	**Tolerable**
h. El trabajo con un entorno software desconocido causa problemas no previstos.	**Probable**	**Critico**
i. El desarrollo de funciones software erróneas requiere volver a diseñarlas y a implementarlas	**Ocasional**	**Critico**
j. Los componentes desarrollados por separado no se pueden integrar de forma sencilla, teniendo que volver a diseñar y repetir algunos trabajos	**Remota**	**Catastrófico**
k. En el último momento, a los usuarios finales no les gusta el producto, por lo que hay que volver a diseñarlo y a construirlo.	**Improbable**	**Catastrófico**

2. **Planificación de Riesgos**

Para este proceso, asignamos los planes de contingencia a cada uno de los posibles riesgos. Es importante destacar que, para cada reunión, en la que realizábamos entregas,

comprobábamos y actualizamos la tabla de riegos y los planes de contingencia que debíamos seguir, para cada uno de los nuevos riesgos.

Id. Riesgo	*Plan de Contingencia*
a.	Tomar en cuenta el plazo de investigación de las herramientas en la planificación de la elaboración del producto.
b.	Asignarle las tareas que en la planificación sean por debajo del tiempo medio de desarrollo, para mantener el nivel de producción constante.
c.	Al tener entregas iterativas permite evaluar la productividad del empleado y ver que tareas se le dan mejor y así mantener un buen nivel de producción.
d.	Se realizan reuniones semanales con el cliente para consultarle sobre los requisitos de la aplicación y mantener una buena comunicación en cuanto a que es lo que se espera del producto.
e.	Definir bien los requisitos básicos de la aplicación y priorizarlos. En caso de que el requisito se genere después de los primeramente acordados, evaluamos su prioridad y consideramos si debe añadirse ahora o en versiones futuras.
f.	El cliente debe definir los requisitos en colaboración con los desarrolladores, para poder tomar en cuenta los diferentes tipos de escenarios que puede tener un requisito.
g.	Consultar diferentes prototipos de interfaz con el cliente antes de desarrollarlos.
h.	Se busca trabajar con un entorno multiplataforma que no requiera ser adaptarlo a los Sistemas Operativos, sino que permita que sea lo más genérico posible.
i.	Antes de desarrollar, consultar los diferentes casos que puede tener la función, evaluando siempre que sea optima y si es reutilizable.
j.	Siempre se toma en cuenta todas las dependencias que requiere el componente, para evitar que, al momento de unir, cause algún tipo error o los errores sean menores.
k.	Al realizar entregas funcionales iterativas, el cliente siempre está evaluando lo que hay y lo que considera que se debe cambiar. El resto de los usuarios que utilicen la aplicación depende de las decisiones que tome el cliente en el momento de desarrollo y mantenimiento de la aplicación.

El haber trabajado con metodologías ágiles, nos facilitó evaluar continuamente el proceso de gestión de riesgos, minimizando los fallos.

1.5.- Herramientas de Desarrollo

a) Trello

Es una herramienta de gestión de proyectos que hace que la colaboración en proyectos sea más simple. En nuestro caso, utilizamos esta herramienta, implementando un sistema de Kanban indicando los requisitos y las tareas que estaban "por hacer", "en proceso" y "hechas". Gracias a su etiquetado de colores, sabíamos a qué rol pertenecía dichas tareas y su sistema de notificaciones permitía saber cuándo existían cambios en el tablero de requisitos.

b) Git/GitHub

Git es un sistema de control de versiones distribuido, mientras que GitHub es una plataforma web que aloja repositorios de códigos usando Git, lo cual permite gestionar versiones del código y gestionar errores para las siguientes releases. En nuestro caso, creamos un repositorio principal y un repositorio copia (fork), para poder trabajar desde repositorios independientes. Una vez realizada las tareas que nos habíamos asignado cada una, sincronizábamos ambos repositorios. Esto nos permitió la integridad del código durante su desarrollo

c) PyCharm IDE

Toda codificación Python, plantillas y CSS fue realizada desde este IDE puesto que es uno de los más completos para desarrollar en Django. A parte de tener un depurador avanzado y fácil de utilizar, permite configurar proyectos con sistema de control de versiones (Git). Además de eso, soporta entornos virtuales e intérpretes de Python que fueron de gran ayuda para la configuración de nuestro contenedor en Docker.

d) **MySQL Workbench**

Es una herramienta desarrollada por MySQL que ayuda al diseño y documentación de bases de datos. Es una aplicación multiplataforma que fue de mucha ayuda para nosotras, ya que ambas desarrollamos la aplicación en diferentes Sistemas Operativos y esta herramienta nos permitió trabajar y tener una misma visualización de la BBDD. También es importante mencionar que es una aplicación que permite desarrollar Diagramas de E-R para aportar una mejor visión de la BBDD e importar archivos SQL que sirvieron al momento de hacer cambios y pruebas durante el desarrollo de CLEO.

Capítulo 2: Desarrollo

En el capítulo 2 del proyecto, nos centramos en explicar las funcionalidades que están implementadas en la aplicación. Para ello, explicamos un poco para qué son cada una de las entidades que la componen y detallamos qué actividades puede realizar el usuario que esta autenticado en ese momento (anónimo, administrador, empleado encargado, empleado básico o cliente) mediante casos de uso. Además, explicamos cómo gestionamos todos los requisitos de la aplicación con el uso de Trello.

2.1.- Componentes de la Aplicación

Para comprender la organización y como está compuesta la aplicación, se explicarán conceptos generales que permitan al lector tener una visión más clara de CLEO:

e) Cliente

Un cliente es un tipo de usuario en la aplicación., que podrá identificarse y hacer uso de ella. De tal forma que pueda pedir una cita, cancelarla o recibir notificaciones. Estas notificaciones servirán para; confirmar una cita, reservas de productos y nuevas las promociones.

f) Empleado

Un empleado es otro tipo de usuario, cuyos permisos dependen del tipo de privilegios que tenga asignado. Los cuales podrán ser de dos tipos: básico o encargado, pero ambos actores tienen control directo sobre la aplicación.

i. **Encargado:** Es capaz de registrar nuevos empleados o nuevos clientes, y organizar el resto de los módulos dentro de la aplicación. Aparte, este tipo de empleados solo pueden ser dados de alta por los administradores del sistema o por otros encargados. Podrán recibir notificaciones referentes al *stock*.

ii. **Básico:** Puede crear y asignarles citas a los clientes, pero tiene un acceso restringido con el resto módulos, es decir, solo podrá consultarlos. No tendrá acceso para crear o modificar elementos (pedidos, promociones, maquinas, etc). Puede notificar a los encargados de la falta algún producto en el *stock*.

Todos los usuarios son controlados por el administrador del sistema. Quienes son asignados por medio de un nombre de usuario, correo electrónico y una contraseña para acceder a sus datos y poder autenticarse en la aplicación. El usuario siempre que quiera podrá ser dado de baja y sus datos serán borrados de forma permanente de la aplicación.

g) Tratamiento

Esta entidad hace referencia a los tratamientos disponibles que tiene la empresa. Estos tendrán un nombre, descripción, duración, la zona (corporal, facial), la maquina necesaria para ese tratamiento y el producto que se va a emplear (lo que permitirá llevar un *stock* de cada producto).

h) Cita

Esta entidad puede ser utilizada por los módulos de cliente y empleado, el uso de este módulo es básicamente el de asignar una cita de un cliente con un empleado que será el encargado de realizar el tratamiento, junto con la fecha y hora que se realizará la cita y el tratamiento a realizar. Esta cita podrá ser anulada tanto por el cliente como por un empleado. Una vez que la cita se dé por finalizada se generará una factura.

i) Inventario de Pedidos

La función de esta entidad es la de llevar un registro de todos los pedidos que hace la empresa. Cuando un pedido se realiza, automáticamente se guarda en esta entidad con la fecha en la que se realizó y el pedido a que pertenece. Una vez que el pedido tiene como estado entregado, de forma automática se actualiza la fila de este pedido, asignado la fecha en la que fue entregado. De esta manera se puede llevar un control de los pedidos. Además, esta entidad cuenta con un filtro para listarlos, a través de la fecha y/o proveedor.

j) Pedidos

El objetivo principal de esta entidad es el de crear pedidos de productos para la empresa, cada pedido solo puede ser realizado por los encargados y cuando se realiza, se registra de forma automática la fecha y el estado del pedido como encargado; ya que cada pedido puede tener tres posibles estados: encargado, recibido, cancelado. Una vez que el pedido fue entregado, se actualiza la información del *stock*.

k) Máquina

Esta entidad nos sirve para tener un control de las maquinas del centro, que son las que permiten realizar los tratamientos que se ofrecen. Las maquinas tendrán por defecto la fecha de alta del día que se crean en el sistema e indican en que zona se utilizan para trabajar en los tratamientos: fácil o corporal. Identificar la maquina permite optimizar su uso y evita crear citas por salas en las que se use la misma máquina.

l) Sala

Con esta entidad damos de alta a las salas donde tendrán lugar citas, y donde se van a realizar los tratamientos. Cada sala se identifica por un nombre único, estas solo pueden ser gestionadas por los encargados, los básicos solo podrán realizar consultas de ellas.

m) Notificaciones

La principal función de esta entidad es poder mantener una comunicación entre los empleados y clientes de forma fácil. Solo los empleados podrán mandar notificaciones a otros empleados y a los clientes, como, por ejemplo, notificaciones sobre el estado del *stock* de un producto o a los clientes para confirmar citas. Estas notificaciones serán recibidas en el correo de cada uno de los usuarios que se han almacenado en la BBDD.

2.2.- Requisitos Funcionales

En esta sección vamos a describir en detalle las herramientas utilizadas para llevar a cabo la gestión de los requisitos de la aplicación.

2.2.1.- Trello

Los requisitos que acordamos con el cliente para cada uno de los usuarios que pueden navegar en CLEO fueron registradas en la aplicación de Trello. Estos requisitos fueron desarrollados de forma similar a historias de usuarios. Que representan a cada módulo de la aplicación además de hacer distinción con cada uno de los posibles roles (administrador, encargado, básico, cliente y anónimo). Cada historia de usuario es única y al ser épicas, están compuestas por funciones de ese modulo, que, a su vez, para poder realizar estas funciones, debían dividirse en tareas más pequeñas.

A continuación, mostraremos capturas de nuestro tablero en Trello para explicar cómo la utilizamos esta herramienta para definir los requisitos funcionales de la aplicación.

Ilustración 1 Tablero de requisitos y como esta agrupado por prioridad y por el tipo de usuario

El método de prioridad que hemos implementado para las historias de usuario es KANO, el cual nos permite agrupar las historias de usuario en cuatro categorías:

a) Obligatorias: son requisitos que son esperados por el cliente y conducen a la insatisfacción si no se encuentran en la aplicación.

b) **Necesarias:** requisitos que al estar en la aplicación conducen a un incremento de satisfacción del cliente.

c) No Esperadas: requisitos que no son esperados por el cliente y que puede resultar en una gran satisfacción si están disponibles.

d) **Indiferentes:** el cliente no está interesado en estos requisitos.

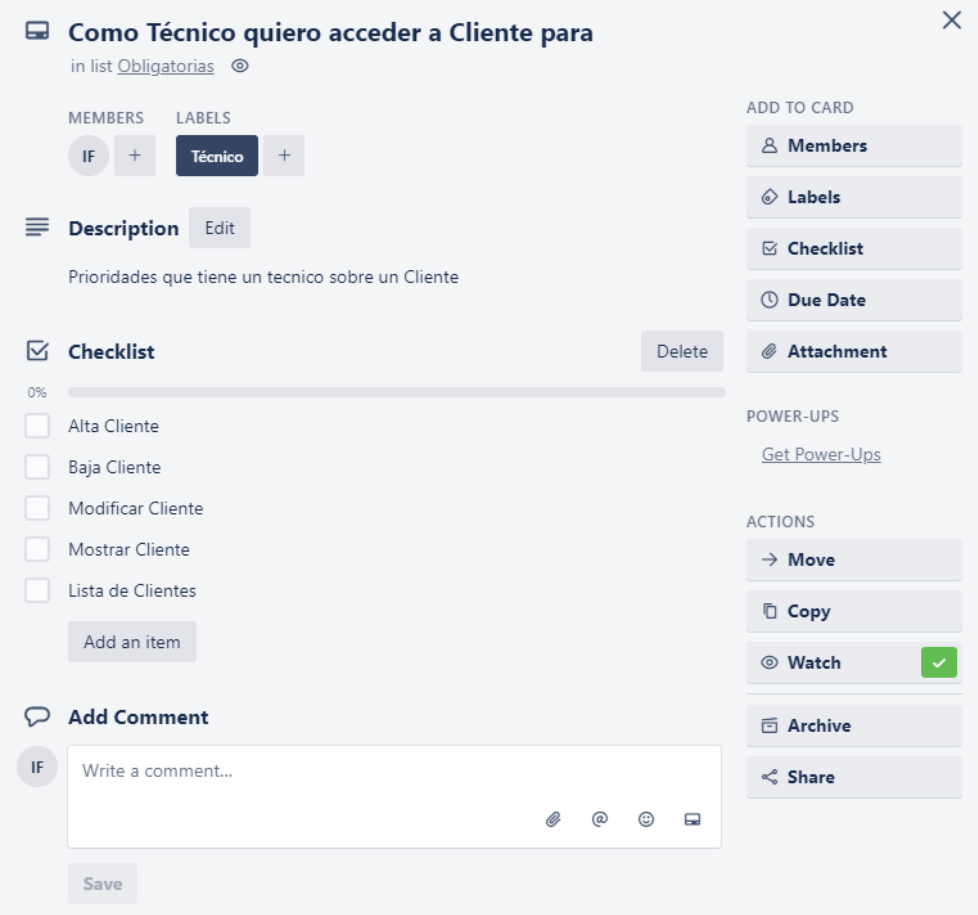


Ilustración 2 Ejemplo de una Historia de Usuario Épica

La ilustración 2 muestra cómo se ve una historia de usuario detalladamente; muestra a quién se le asigno esa HU, las etiquetas que indicaban a que rol al que se le añadía y el checklist es una lista de funcionalidades que debe cumplir y que a medida que se avanzaba la íbamos actualizando para tener claro las funciones que faltaban por hacer.

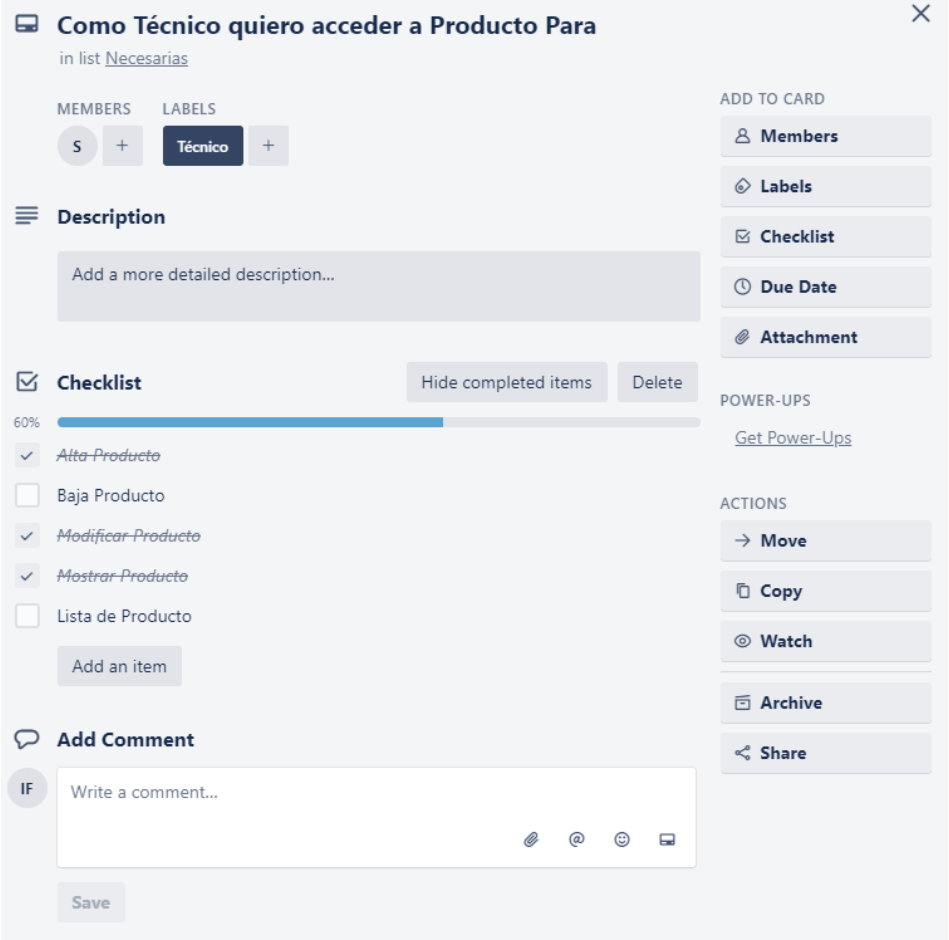


Ilustración 3 Barra de Avances de una HU Épica

La ilustración 3 muestra la barra de avances realizados en esa historia de usuarios y las funciones que faltan por hacer, este caso solo dar de baja un producto y listar los productos que haya en el sistema.

Ilustración 4 Etiquetas de Roles

Las etiquetas eran la forma más visual para saber que usuario teníamos que trabajar y es por ello cada uno estaba representado con un color diferente.

Luego, cuando nos tocaba evaluar la historia de usuario que debíamos trabajar, la dividíamos en tareas más pequeñas que se iban adaptando a las de la configuración, para añadirlas a la aplicación y tareas que consistían en realizar procesos de validación y verificación, como es el caso de la ilustración 5.

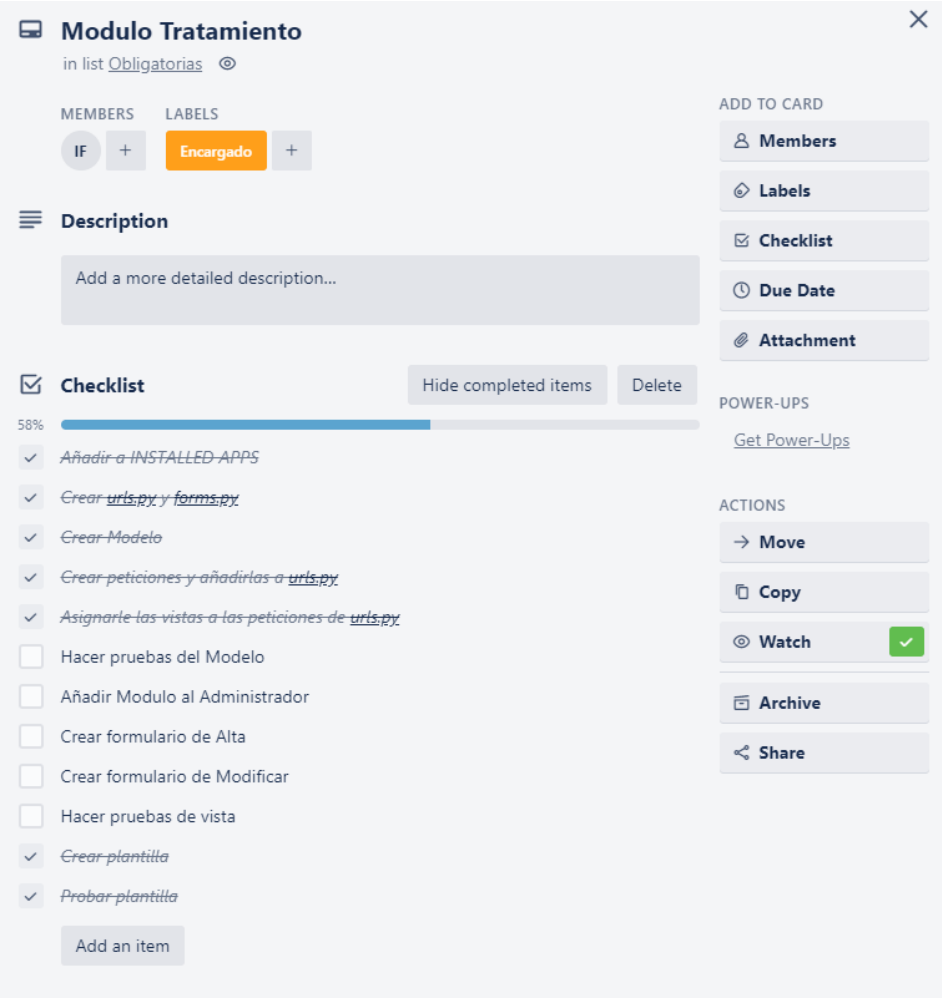


Ilustración 5 Dividir Tareas pequeñas de las HU

Esta imagen describe las tareas en las que dividíamos la HU en tareas pequeñas para ir recordando los pasos que debíamos implementar en cada módulo del sistema.

Ilustración 6 Historias de Usuario Completadas

Una vez finalizada la tarea la colocábamos en una lista de HU completadas que luego nos ayudaría a evaluar lo que estaba hecho y mostrárselo al cliente.

2.2.2.- Casos de Uso

En los casos de uso podremos ver las distintas acciones que puede hacer un actor (los usuarios de la aplicación) en diversos escenarios. Entendemos como actor a una persona anónima que desea ser cliente, un cliente registrado, a los empleados y al administrador de la aplicación.

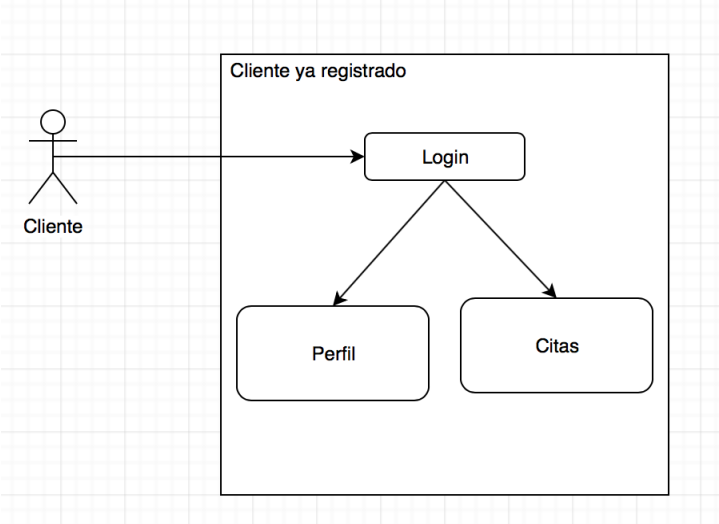


Ilustración 7 Caso Uso Cliente ya autenticado

En este caso de uso vemos la secuencia de un cliente ya dado de alta en la aplicación. Una vez que se autentica, tienes dos vistas para elegir, el apartado de citas o sus datos. (Ver imagen a la derecha).

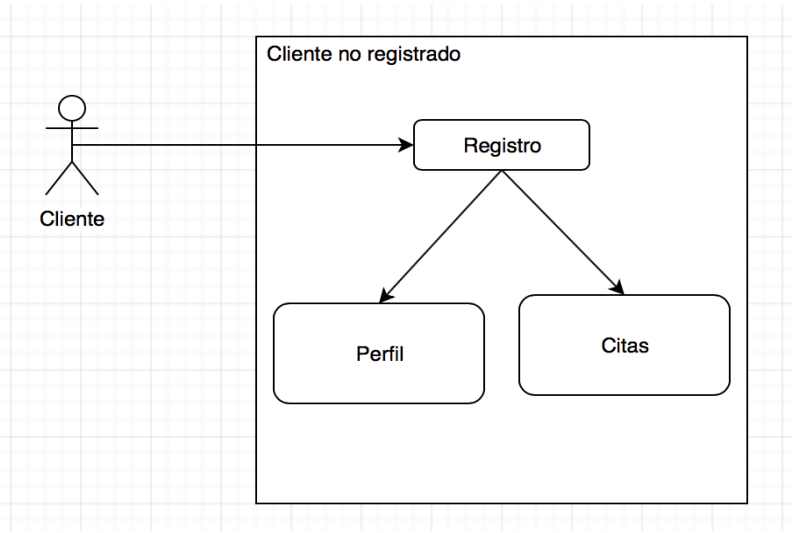
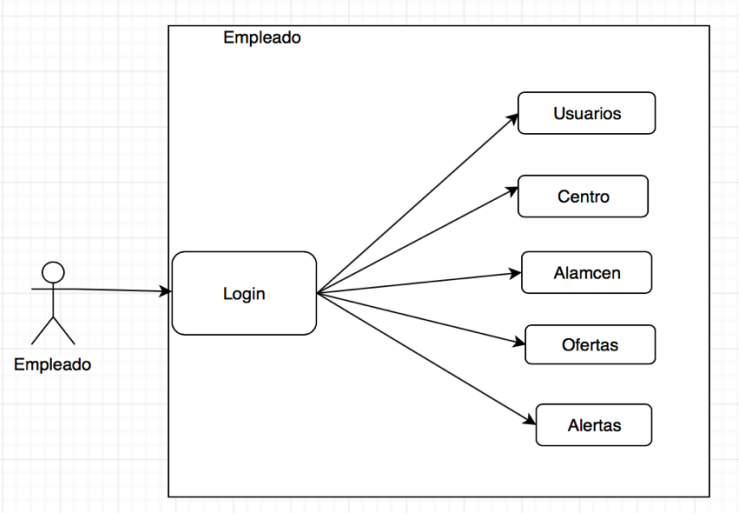


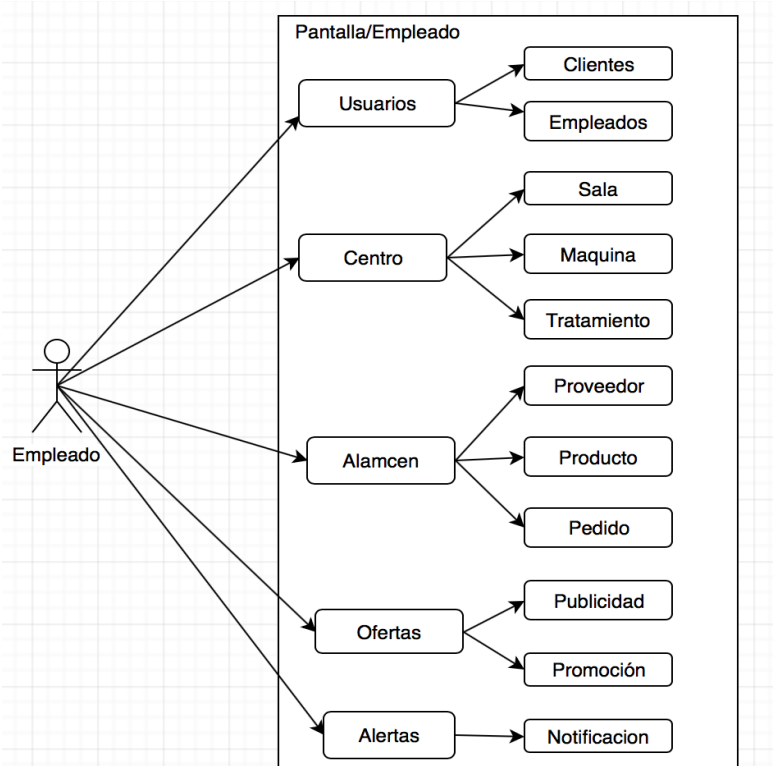
Ilustración 8 Cliente Anónimo

Si, por el contrario, es un usuario anónimo que no está registrado se le muestra las opciones para registrarse primero y después tendrá la opción de ver sus datos.
(Ver Ilustración 8)



Por parte de los empleados, tenemos el siguiente caso de uso. Una vez que se autentica, el empleado tiene acceso a todas las opciones que se ven, ya sea básico o encargado. La distinción se encuentra en la propia vista.

Ilustración 9 Empleado accediendo a CLEO



En este caso, vemos las distintas opciones que tienen los empleados dentro de cada menú. Las diferencias según el rol del empleado se aprecian una vez se accede a la vista, donde se podrá eliminar o crear dependiendo del rol.

Ilustración 10 Caso de Uso Empleado autenticado

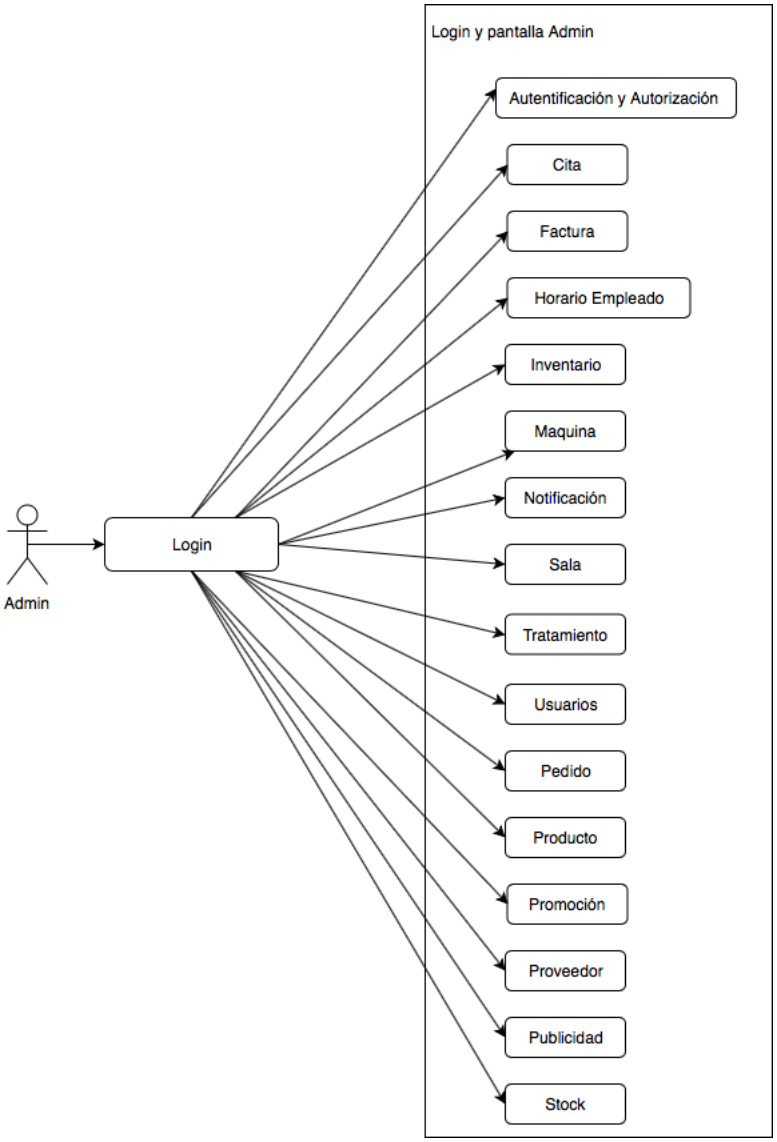


Ilustración 11 Caso de uso administrador CLEO

En este caso, vemos las distintas opciones que tienen los administradores dentro del sistema. Además de tener acceso a todos los módulos del sistema, también tienen control sobre un módulo que les permite manejar los permisos que tienen los usuarios y asignarlo a grupos se usuarios que navegan en la aplicación.

2.4.- Impedimentos

En el siguiente punto, detallamos los impedimentos con los que nos hemos encontrado a lo largo del desarrollo de la aplicación y cómo los hemos solventado:

a) Nuevo lenguaje

El uso del nuevo lenguaje, Python, ha sido desde el principio uno de los primeros problemas para nosotras, porque nos hemos tenido que formar y practicar para poder trabajar con él. Esto dio pie a que, al ser una herramienta nueva

tuviésemos más errores y dudas a la hora de programar. Al ser una herramienta open source, nos facilitó la búsqueda de documentación. Para los impedimentos que se nos presentaban en el momento de desarrollo. En cualquier caso, la curva de aprendizaje no se ha cerrado y seguimos aprendiendo.

b) Nuevo Framework:

Django es una herramienta bastante completa, y que para el desarrollo de aplicaciones web requiere de un tiempo de investigación y prototipado, antes de implementar parte de sus herramientas en las aplicaciones web. En nuestro caso, al ser una herramienta con la que teníamos muy poco tiempo trabajando, las primeras entregas funcionales tardaron más de lo esperado, debido al tiempo de investigación que tuvimos que implementar, pero el resto de las entregas ese porcentaje de investigación decreció y fue aumentando en niveles de producción:

i. **Formularios:** Django facilita la creación de formularios a partir de los modelos. Solo hay que asignarle el tipo de dato que debe obtener el formulario y este se encargara de comprobar si el usuario ha llenado los campos del formulario correctamente.

   El principal problema que nos dio es que, al cargarse el sistema, los formularios cargaban las opciones de los combos una sola vez. De manera que, si había una nueva opción para el combo, esta no salía reflejada en el formulario; para ello tuvimos que hacer una consulta extra en las peticiones que implementaban formularios y que requerían ese tipo de combos, luego llamarlos manualmente en las plantillas, y haciéndolos coincidir con los definidos en el formulario para poder utilizar la validación de datos que trae Django por defecto.

   Esto a veces era complicado porque simplemente un espacio podría hacer que no se validaran bien los formularios, ya que no coincidían los definidos en el forms.py con los que hacíamos manualmente, y para solucionar esto teníamos que generar del forms.py y luego lo copiábamos en una plantilla HTML.

ii. **Modelos:** Una de las características de Django es que, al crear los Modelos, defines como se manejan los datos en la aplicación y se trabaja con las BBDD de forma independiente. Debido al poco conocimiento que teníamos para trabajar con modelos, nuestra solución fue hacer el diseño de las entidades desde nuestra BBDD en SQL y migrarlas al modelo con el comando:

```
>>python manage.py inspectdb > models.py
```

El fichero models.py hacia una copia de todas las clases que tenía la aplicación, incluyendo las de la configuración de Django. De modo que esto nos aseguró que los modelos fuesen según los requisitos del cliente y nos permitió tener mayor agilidad a la hora del desarrollo.

Por otra parte, cuando se realizó la configuración de los usuarios que pueden acceder a CLEO, trabajamos con los módulos que tiene Django por defecto. Debimos investigar y modificar las clases de usuarios, haciendo las adaptaciones necesarias para que finalmente pudiésemos migrar del modelo a la BBDD, con los comandos:

```
>>python manage.py makemigrations
>>python manage.py migrate
```

Esto no causo ningún problema con las tablas existentes en la BBDD porque las migraciones de Django soportan control de versiones sobre las tablas de las Bases de Dato, modificando la tabla, sus atributos y su contenido sin problema alguno.

c) Elaboración de la Base de Datos:

i. Relaciones: La principal duda que tuvimos fue como representar ciertas relaciones N:M, como es el caso de Producto-Proveedor:

Ilustración 12 Relación de N:M Producto y Proveedor

Al ver que teníamos muchas relaciones de este tipo y ya teníamos muchas tablas. Decimos fusionar este tipo de relaciones en terceras tablas que formaban parte del diseño inicial. Como este caso, que empleamos Pedido donde se ve la relación de producto y proveedor. Pedido.

ii. Tablas Tipo o Estado: Otra duda surgió con la implementación de las tablas tipo o estado, como por ejemplo tipoProducto; ya que anteriormente habíamos trabajado con enumerados para definir los tipos. Sin embargo, nuestro tutor nos sugirió que la mejor solución era crear una tabla que definiese los tipos, de modo que al modificar el tipo o añadir otros tipos,

el coste fuese menor ya que solo afectaría a una fila de la tabla tipo y no todos los valores que dependen de los tipos.

iii. **Diseño e Implementación BBDD a Django:** La configuración de la BBDD en la aplicación nos llevó algún tiempo porque al trabajar con diferentes SO (Windows & MAC), no valía para ambos el tipo de configuración y la librería oficial que sugiere Django utilizar *mysqlclient*, tiene problemas para utilizarlo en Windows. Nuestra solución fue trabajar con una librería independiente llamada *mysql-connector-python* que funciona igual para los dos Sistemas Operativos.

iv. **Docker:** Hemos tenido problemas a la hora de crear el Dokcerfile y los docker-compose en los SO (Windows y MAC). Por el lado de MAC fueron los permisos de acceso a la BBDD y que unos permisos que requiere de más para el entorno, mientras que por el lado de Windows para poder trabajar con Docker requiere tener licencia Pro o Education, ya que de ser la licencia home bloquea ciertas variables de entorno que Docker necesita utilizar para configurar nuestro entorno.

Nuestra solución fue trabajar con otras herramientas en el proceso de desarrollo mientras conseguíamos una solución a los errores causados.

Capítulo 3: Diseño y organización

El capítulo 3 explica el diseño de la base de datos de CLEO, adaptándola con las entidades que trae Django en sus proyectos para un mejor manejo de los usuarios del sistema. También, explicamos cómo es la arquitectura de Django, exponiendo cómo funciona cada una de las capas que tiene y mostrando implementaciones con ejemplos de nuestra aplicación. Finalmente explicamos cómo Django trabaja las autenticaciones y autorizaciones para las peticiones que se realizan en el proyecto.

3.1.- Diseño de la Base de Datos

Para el manejo de los datos en nuestra aplicación trabajamos con MySQL y utilizando MySQL Workbench como interfaz de usuario para un mejor control durante el proceso de desarrollo. Lugo, para la configuración, el nombre de nuestra base de datos se llama: cleodb. Consta de diez tablas que forman parte de Django para el manejo de usuarios, grupos, autenticación y permisos, sesiones, migraciones, etc. También existen otras veintiséis tablas demás que forman parte del sistema de CLEO, la cuales, hemos diseñado y añadido nosotras según los requerimientos.

Una característica que tienen nuestras tablas es que las tablas de tipo y de estado son tablas sencillas que permiten modificar o añadir nuevas opciones si en un futuro el sistema lo requiere. En las tablas de Producto y Proveedor, hemos añadido un valor booleano que nos permitiera realizar bajas lógicas; ya que al tener registros de los pedidos necesitamos consultar Productos y Proveedores independientemente de si se han dado de baja en el sistema. Para el resto utilizamos bajas físicas, es decir, desaparece totalmente de nuestra BBDD.

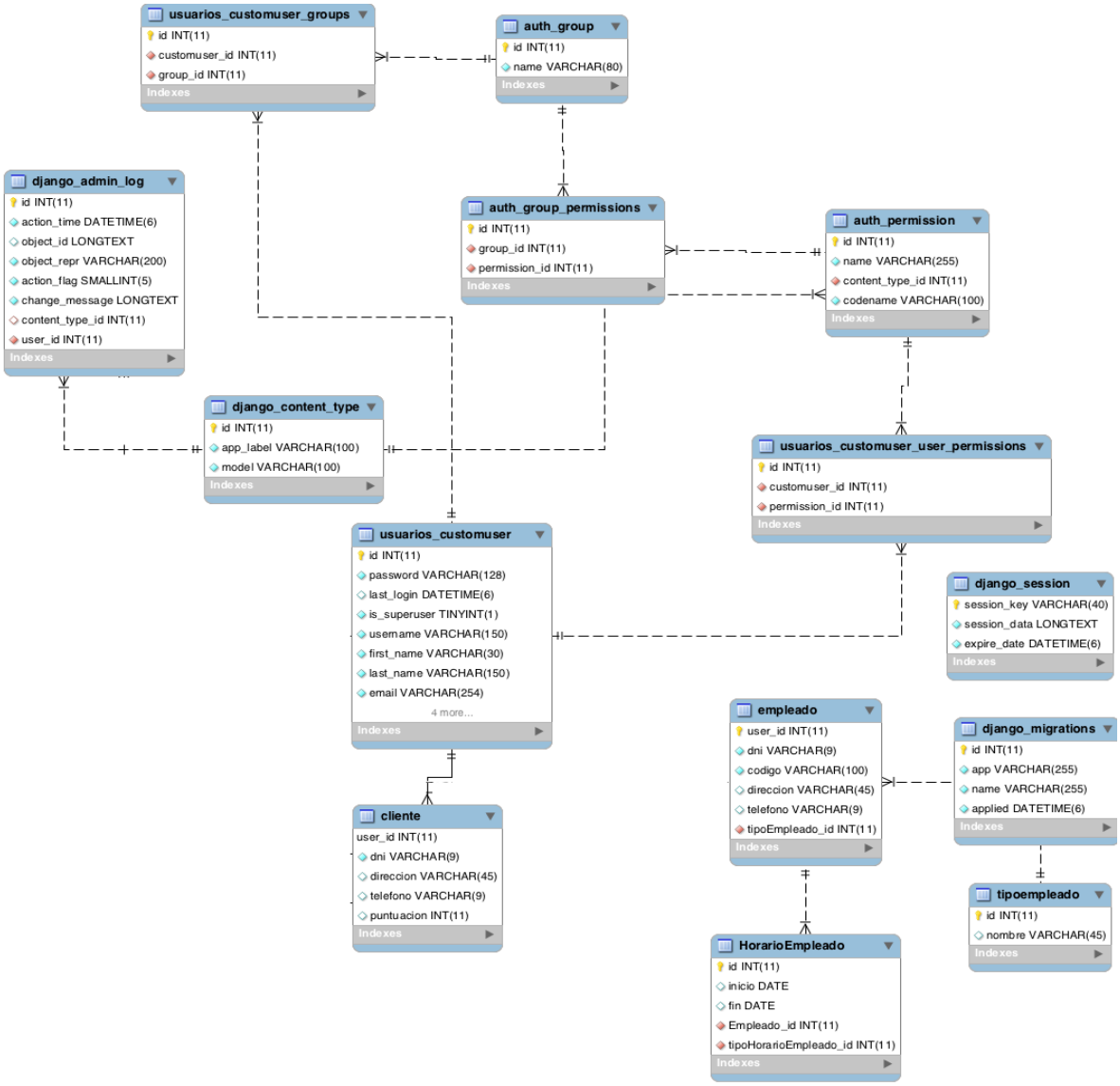


Ilustración 13 Tablas de BBDD que son creadas por Django

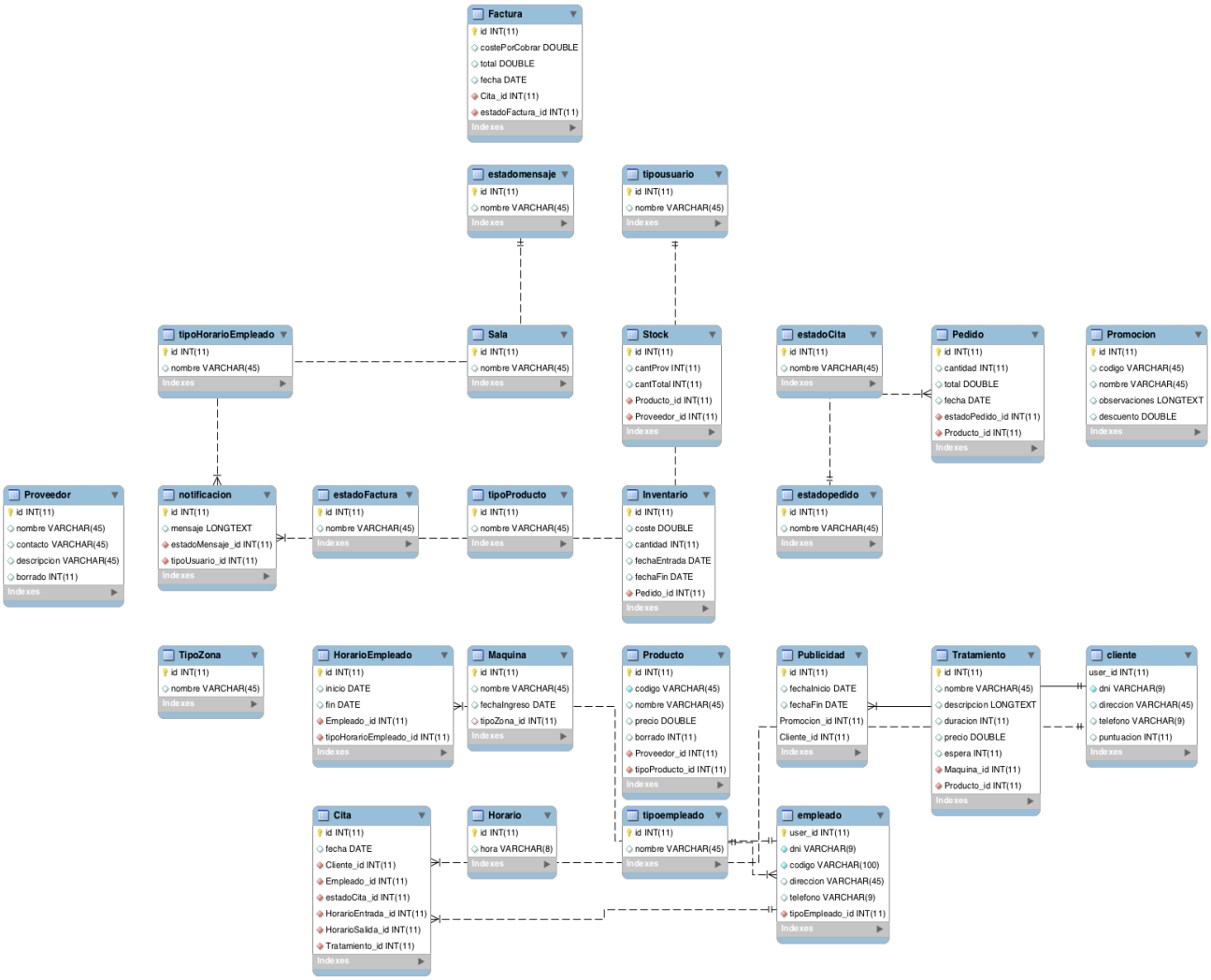


Ilustración 14 Tablas de Django acopladas a las de la aplicación

Como vemos en las imágenes anteriores **(**ilustración 13 e ilustración 14**)**, muestran el diseño de la BBDD, en la ilustración 13 solo mostramos las tablas creadas por Django y las relaciones que tiene con las del sistema CLEO. En la segunda, ilustración 14, se muestras solo las propias del sistema y las relaciones entre ellas. Estos diagramas han sido generados por la aplicación Workbench.

3.2.- Arquitectura

Django utiliza una versión del patrón Modelo-Vista-Controlador (MVC), llamada MTV (o también conocida como Modelo-Plantilla-Vista. La capa modelo es la que se encarga todos los datos que genera la aplicación, la capa vista se encarga de manejar la lógica de negocio y respuestas que genera la aplicación y la capa plantillas definen como se verán los datos en el navegador.

d) ¿Cómo funciona el patrón MTV?

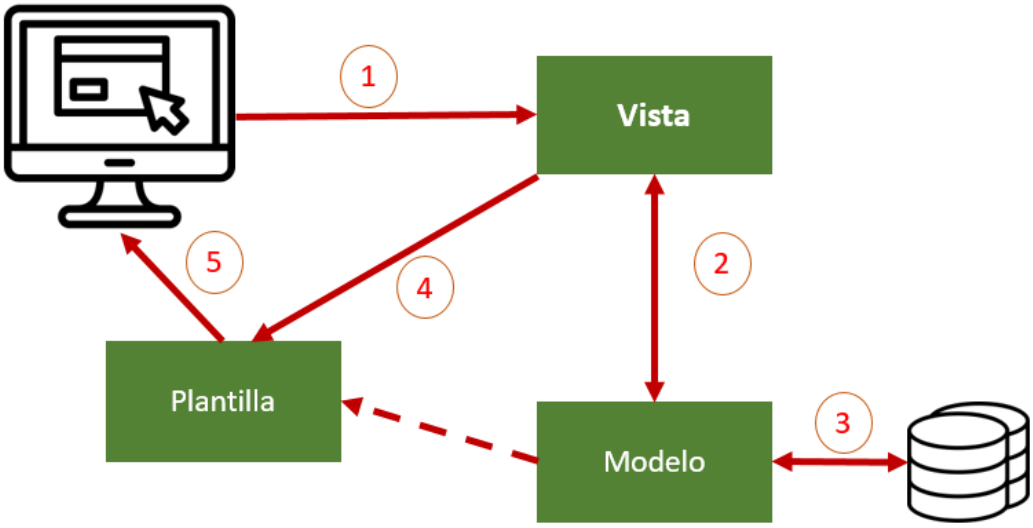


Ilustración 15 Como funciona el MVT en Django

Una petición del navegador envía una URL, la cual tiene una vista configurada (Paso 1), la Vista interactúa con el Modelo para obtener los datos mediante interacciones con la BBDD (Pasos 2 y 3) y una vez que el Modelo le responde a la vista, esta transforma la plantilla (paso 4) y finalmente la muestra en el navegador (Paso 5).

3.2.1- Capa Vista

Esta capa se encarga de evaluar que vista gestiona los modelos y cómo serán presentados al usuario, a través de las URL. Las vistas en Django pueden definirse en funciones o en métodos de una clase, su principal propósito es determinar qué datos serán visualizados.

Es importante decir que la capa vista es la que se encarga del procesamiento de datos obtenidos por la capa Modelo. Dichos datos serán mostrados al usuario en forma de listas, campos informativos o formularios. Estos últimos envían los datos al sistema a través de las plantillas. Estos datos, son recogidos por la capa vista donde finalmente se validan y almacenan los datos en la aplicación.

```python
from django.urls import path
from . import views

urlpatterns = [
    path('nuevo', views.nuevo, name='app_nuevo'),
    path('lista', views.listar, name='app_listar'),
    path('<int:pk>/eliminar', views.eliminar, name='app_eliminar'),
    path('<int:pk>/detalle', views.detalle, name='app_detalle'),
    path('<int:pk>/modificar', views.modificar, name='app_modificar'),
]

```

Ilustración 16 Ejemplo de peticiones en el url.py

En la imagen (Ilustración 16) podemos ver como se ha implementado el despachador URL, este se encarga de redirigir las peticiones HTTP a la vista apropiada. En el path indicamos la ruta que se va a asociar, con la plantilla que se devuelva en el método que indicamos como segundo parámetro. El despachador URL también puede emparejar patrones de cadenas o dígitos específicos que se encuentran en la ruta y los pasa a la función de visualización (vista) como datos.

```python
def nuevo(request):
   if request.method == "POST":
      form = FormTratamientoInsert(request.POST)

      if form.is_valid():
         datos = form.cleaned_data

         # recogemos los datos
         nomTra = datos.get("nombre")
         desc = datos.get("descripcion")
         tipo = datos.get("maquina")
         prod = datos.get("producto")
         dur = datos.get("duracion")
         prec = datos.get("precio")
         esp = datos.get("espera")

         if not Tratamiento.objects.filter(nombre=nomTra):  # todavia se puede guardar una maquina mas

            instMaquina = Maquina.objects.get(id=tipo)
            instProducto = Producto.objects.get(id=prod)
            t = Tratamiento(nombre=nomTra, descripcion=desc, maquina=instMaquina, producto=instProducto,
                            duracion=dur, precio=prec, espera=esp)
            t.save()
            return HttpResponseRedirect("/tratamiento/lista")

         else:
            messages.error(request, 'El tratamiento ya existe.')
            messages.error(request, '')
      else:
         messages.error(request, 'Datos no validos.')

   return render(request, 'tnuevo.html', {'elem': 'Añadir', 'titulo': 'Añadir Tratamiento',
            'datosMaquina':listaMaquinas(""), 'datosProducto':listaProducto("")})


```

Ilustración 17 Ejemplo de la vista nuevo proveniente del view.py

Por otro lado, la imagen 17 representa a la vista de un nuevo tratamiento, es decir, cada vez que se llame a la ruta de nuevo, por ejemplo, ***localhost:8000/tratamiento/nuevo,*** esta será la función que gestione dicha petición, según el tipo de petición solicitara (GET) los datos o los obtendrá (POST) para que luego sean trabajado por la capa Modelo.

3.2.3- Capa Modelo

El modelo es la capa que se encarga de definir los campos que almacenan cada una de las entidades que requiere la aplicación. Se definen mediante clases que están compuestas por los atributos que componen la entidad, métodos que permiten definir las propiedades que posee y relaciones que tienen entre ellas (ORM).

Esta capa, al definir las entidades como clases, nos permite indicar y controlar el comportamiento de los datos, obteniendo como resultado objetos de esa entidad, lo cuales hacen que el trabajo en las Vistas sea más sencillo. También es importante destacar que al trabar con ORM, Django nos proporciona mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la Base de Datos desde esta capa.

```
1  from django.db import models
2  from ..Maquina.models import Maquina
3  from ..Producto.models import Producto
4
5
6  class Tratamiento(models.Model):
7      id = models.AutoField(primary_key=True)
8      nombre = models.CharField(unique=True,max_length=45, blank=True, null=True)
9      descripcion = models.TextField(blank=True, null=True)
10     duracion = models.IntegerField(blank=True, null=True)
11     precio = models.FloatField(blank=True, null=True)
12     espera = models.IntegerField(blank=True, null=True)
13     maquina = models.ForeignKey(Maquina, models.DO_NOTHING, db_column='Maquina_id')
14     producto = models.ForeignKey(Producto, models.DO_NOTHING, db_column='Producto_id')
15
16     class Meta:
17         managed = False
18         db_table = 'Tratamiento'
19         unique_together = (('id', 'nombre','maquina','producto'),)
```

Ilustración 18 Ejemplo modelo Tratamiento

Esta imagen es un ejemplo de cómo representamos la entidad Tratamiento en un modelo. Indicamos cada uno de sus atributos en una variable y asignando su tipo a partir de ***django.db.models , así*** como también las relaciones que pueden existir (claves foráneas) entre modelos, como es el ejemplo de las líneas 13 y 14.

3.2.4.- Plantillas

Las plantillas son la capa de presentación en Django, se presentan en ficheros HTML con marcadores de posición (contenedores con formatos previos), esto permite generar paginas dinámicas, rellenando estos marcadores con información que es proporcionada por la vista y consultada o gestionada por el modelo.

Trabajan como los ficheros HTML normales, en donde tienen asociadas sus hojas de estilos CSS, y sus extensiones a otras librerías como lo pueden ser JQuery y Bootstrap. También permiten utilizar directamente información obtenida en las peticiones, como es el caso de la información del usuario que proporciona Django (request.user) y que permite hacer extensiones sobre ellas, utilizando una plantilla base de la cual extenderíamos y solo tendríamos que definir los bloques genéricos que se fueren definido en la plantilla base.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1, maximum-scale=1">
    {% if title %}
        <title>CLEO - {{title}}</title>
    {% else %}
        <title>CLEO</title>
    {% endif %}

    {% load static %}
    <link rel="stylesheet" href="{% static 'bootstrap/css/bootstrap.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/styles_gral.css' %}">
    {% if user.is_staff %}
        <link rel="stylesheet" href="{% static 'css/style2.css' %}">
    {% else %}
        <link rel="stylesheet" href="{% static 'css/style3.css' %}">
    {% endif %}

    <script src="{% static 'js/jquery-3.4.0.min.js' %}"></script>
    <script src="{% static 'bootstrap/js/bootstrap.min.js' %}"></script>

</head>
<body class="empleado">
    <header>
        <div class="head-img row">

        </div>
        <nav class="navbar navbar-inverse">
            <div class="container-fluid">
                <div class="navbar-header">
                    <a class="navbar-brand" href="/nosotros">CLEO</a>
                </div>
                <ul class="nav navbar-nav">
                    <li  class="home">
                        <a href="/"><span class="glyphicon glyphicon-home"></span></a>
                    </li>


                </ul>
                <ul class="nav navbar-nav navbar-right">

                    {% if user.is_authenticated %}
                    <li>
                        <a href="/perfil"> <span class="glyphicon glyphicon-user"></span> Mi Perfil </a>
```

Ilustración 19 Ejemplo de una plantilla base

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

{% block content %}
<div class="container form-login">
        <h3>Entrada</h3>
        <div class="row">
            <form method="post">
               {% csrf_token %}
               {{ form.as_p }}
               <div>
                   <button class="btn col-md-12 " type="submit">Entrar</button>
                   <small class="text-muted col-md-12">Aun no te has registrado?
                            <a href="/registrar">Registar</a>
                   </small>
               </div>
            </form>
        </div>
    </div>
{% endblock %}
```

Ilustración 20 Ejemplo de una plantilla que hereda de la plantilla base

En ambas imágenes podemos ver un ejemplo de cómo desarrollamos las plantillas de Django, aquí se siguen utilizando las etiquetas de HTML, pero interpreta el lenguaje Python al encapsularlo mediante la siguiente sintaxis: **{% *<código>* %}**. También este tipo de sintaxis nos permite generar una plantilla base y extender de ella como es el caso de la primera imagen (plantilla base), siendo extendida en la imagen dos desde otro fichero HTML.

3.3- Autenticación y autorización

Las sesiones en Django son un mecanismo para almacenar información arbitraria por navegador, para poder consultarla para el sitio web cuando el navegador se conecta. Cada dato asociado con una sesión se conoce como una clave, que se usa tanto para guardar como para recuperar la información de una aplicación web.

En el caso del sistema de autenticación y autorización que, proporciona Django, éste está construido sobre el framework de sesión que el mismo ofrece. Este sistema permite verificar credenciales de usuario y definir qué acciones puede realizar, de tal manera que permite a los desarrolladores restringir las tareas en cada aplicación. En nuestro proyecto, tuvimos que adaptar el modelo de autenticación que trae Django por defecto y adaptarlo a los diferentes usuarios acceder a contenido restringido de la aplicación.

```python
from django.contrib.auth.models import AbstractUser
from django.db import models



class CustomUser(AbstractUser):
    is_client = models.BooleanField(default=False)

```

Ilustración 21 Adaptación del modelo AbstractUser

La imagen () muestra cómo creamos una clase **CustomUser** que hereda de **AbstractUser**; la clase por defecto que tiene Django para el sistema de Autenticación de usuarios y que tiene un nuevo atributo llamado **is_client** nos permitirá saber si ese usuario es un cliente. En el caso de que el usuario sea empleado o administrador, la clase **AbstractUser** posee dos variables por defecto llamadas: **is_staff, is_superuser,** que permiten saber fácilmente cuál es el tipo de usuario registrado en la aplicación.

```python
class EmpleadoRegistrationForm(UserCreationForm):
    nombre = forms.CharField(max_length=45, label="Nombre ",
                             widget=(forms.TextInput(attrs={"id": "nombre"})))

    apellidos = forms.CharField(max_length=45, label="Apellidos ",
                                widget=(forms.TextInput(attrs={"id": "apellidos"})))

    dni = forms.CharField(max_length=9, label="Dni ",
                          widget=(forms.TextInput(attrs={"id": "dni"})))

    email = forms.EmailField(max_length=45, label="Email ",
                             widget=(forms.TextInput(attrs={"id": "email"})))

    direccion = forms.CharField(max_length=45, label="Direccion",
                                widget=(forms.TextInput(attrs={"id": "direccion"})))

    telefono = forms.CharField(max_length=45, label="Telefono ",
                               widget=(forms.TextInput(attrs={"id": "telefono"})))

    tipo = forms.ChoiceField(choices=tipoChoice())

```

Ilustración 22 Formulario de crear un empleado adaptado a CLEO

En esta otra imagen, vemos cómo tuvimos que adaptar el formulario que trae Django para el registro de usuarios, este caso en concreto es un formulario de Empleados, el cual tiene siete atributos más que el que viene por defecto en la clase **UserCreationForm**.

```
46  @login_required
47  def perfil (request):
48
49      if request.user.is_staff and (not request.user.is_superuser):
50              empleado=Empleado.objects.get(user_id=request.user.id)
51              request.session['type_staff']= empleado.tipoempleado.nombre
52              return render(request, 'empleado/perfil.html')
53
54      elif request.user.is_staff and  request.user.is_superuser:
55          request.session['type_staff'] = 'Admin'
56          return HttpResponseRedirect('/admin')
57
58      elif request.user.is_client:
59              return render(request, 'cliente/perfil.html')
60
```

Ilustración 23 Comprobación de usuario y variable de sesión de usuario

Una vez modificado el sistema de usuarios de Django y adaptado a nuestra aplicación, podemos ver en la imagen que se crea una variable de sesión llamada **request.user**. la cual almacena todos los atributos de la clase **CustomUser** y en la función se utiliza para comprobar que tipo de usuario es y en caso de ser empleado almacena otra llamada **type_staff** que nos ayuda a saber cuál es el tipo de empleado que acaba de acceder a la aplicación.

Capítulo 4: Implementación

Este capítulo contamos qué requisitos se necesitan del sistema, cómo debe ser es el proceso de instalación de la aplicación en él servidor y también se muestran las guías de uso, según los tipos de usuario que existen en ella (administrador, empleado encargado, empleado básico y cliente). Finalmente se comentan las contribuciones que cada una de nosotras aporto para la elaboración de este proyecto.

4.1.- Guía Instalación

Una vez ya desarrollada la aplicación (https://github.com/iiventura/CLEO), hemos seguido los siguientes tutoriales para poder desplegarla:

- Despliegue con Heroku: https://bit.ly/30sFMDy
- Clear DB Heroku: https://bit.ly/2IpBNmi

4.3.- Guía de Uso

CLEO es una aplicación web con diseño responsive que permite acceder a él desde cualquier dispositivo con acceso a Internet. Según entras a la página de inicio, aparece la pantalla de bienvenida con un menú en la cabecera de la aplicación, en él, podrás crear una cuenta como cliente, conocer un poco de la historia de cómo nació CLEO o autenticarte para acceder al contenido principal. El proceso de autenticación es el mismo para los tres tipos de usuarios que existen en la aplicación.

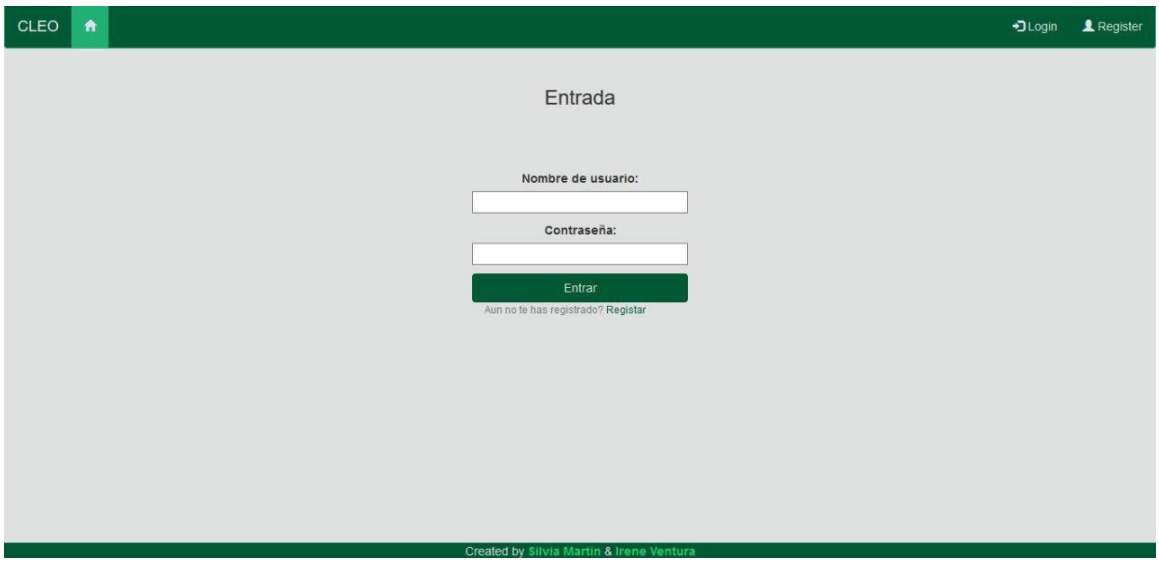


Ilustración 24 Interfaz de autenticación a CLEO

4.3.1.- Guía de Cliente

Un cliente puede registrarse el mismo desde la aplicación, con el siguiente formulario:

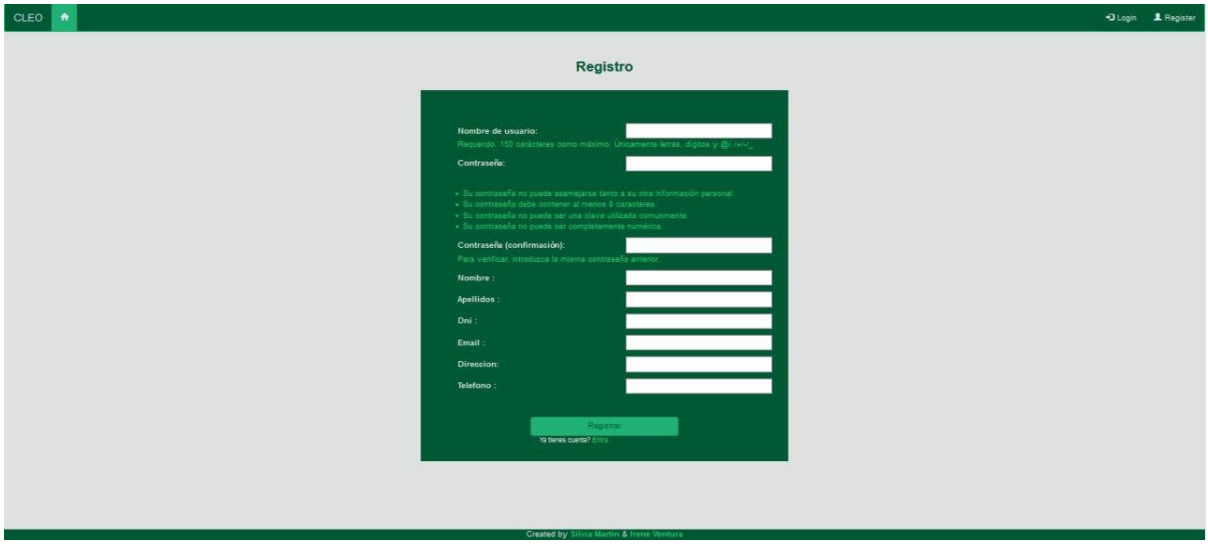


Ilustración 25 Interfaz de un usuario registrándose como cliente

Una vez se ha registrado le sugiere acceder como en la ilustración 24 que muestra la interfaz de autenticación. Luego, si el cliente accede al sistema vera la siguiente interfaz:

Ilustración 26 Vista principal del cliente al haber hecho login

En este caso el cliente tiene disponible un apartado Cita y dos botones directos: Facturas y Productos.

Ilustración 27 Menú de Cliente

En el caso de Cita al darle clic presenta dos apartados más, "Nueva" es para solicitar una cita y "Mis Citas" presenta la lista de citas que tiene pendiente.

Ilustración 28 Opciones de Citas

4.3.2.- Guía de Empleado

CLEO distingue en su aplicación dos tipos de empleado: Encargado y Básico. La interfaz de los empleados se diferencia de la de los clientes por tener tonos azules en las interfaces graficas de la aplicación y cambiar menú y funciones según el tipo de empleado.

4.3.2.1 Encargado

La pantalla inicial del Encargado se muestra de la siguiente manera:

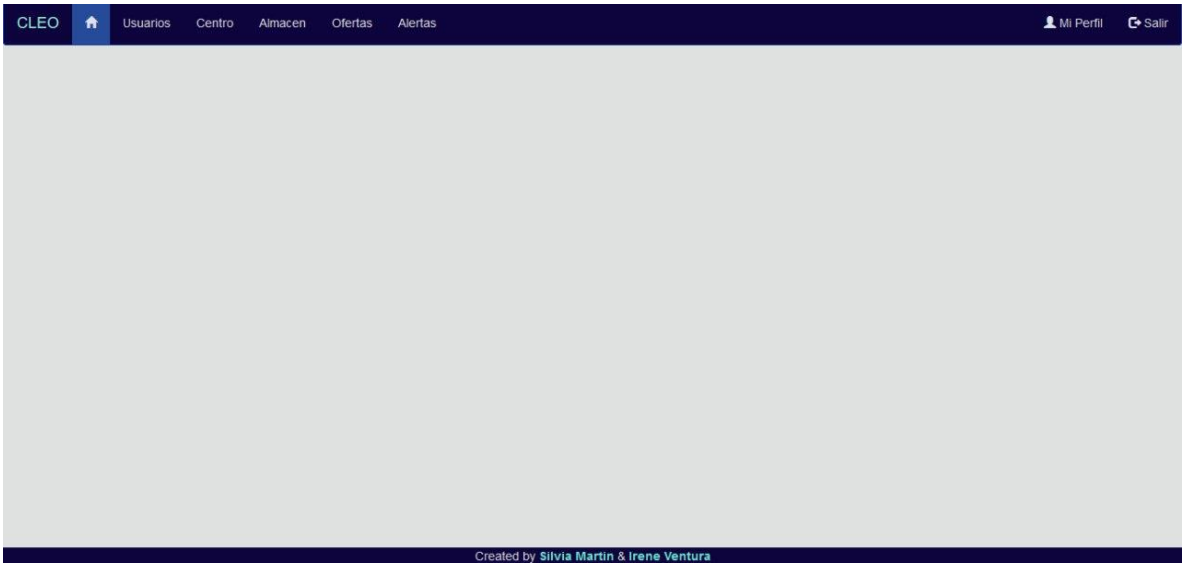


Ilustración 29 Vista principal empleado

La barra de menu para el encargado se muestra con cinco botones que le permiten ejercer su trabajo comodamente: Usuarios, Centros, Almacen, Ofertas y Alertas.

Ilustración 30 Menú de encargado

Si el encargado hace clic en alguno de los cinco botones, cada uno despliega una lista de botones con funciones más específicas.

Ilustración 31Opciones del encargado

Todos los links “Ver” en las listas, redirigen la aplicación a la lista del componente a la que se ha solicitado, Veamos el ejemplo con “Ver Maquinas”:

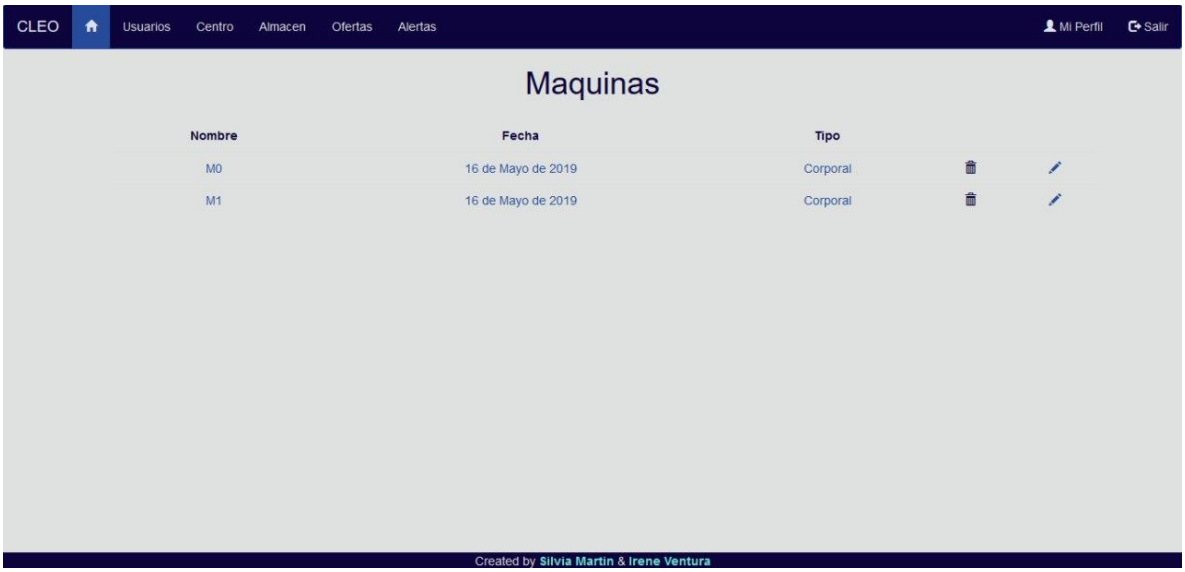


Ilustración 32 Ejemplo lista maquinas encargado

Todas las listas que el encargado puede consultar en la aplicación disponen de los siguientes iconos:

Ilustración 33 Botón de Editar y Eliminar

Son dos botones que permiten eliminar o editar los datos de un elemento que pertenecen a una entidad de la aplicación. Y, por último, los encargados pueden crear información en la aplicación mediante los botones “Nuevo”, los cuales le redirijan al formulario para crear un elemento, por ejemplo, veamos qué pasa si llamamos a “Nuevo Empleado”:

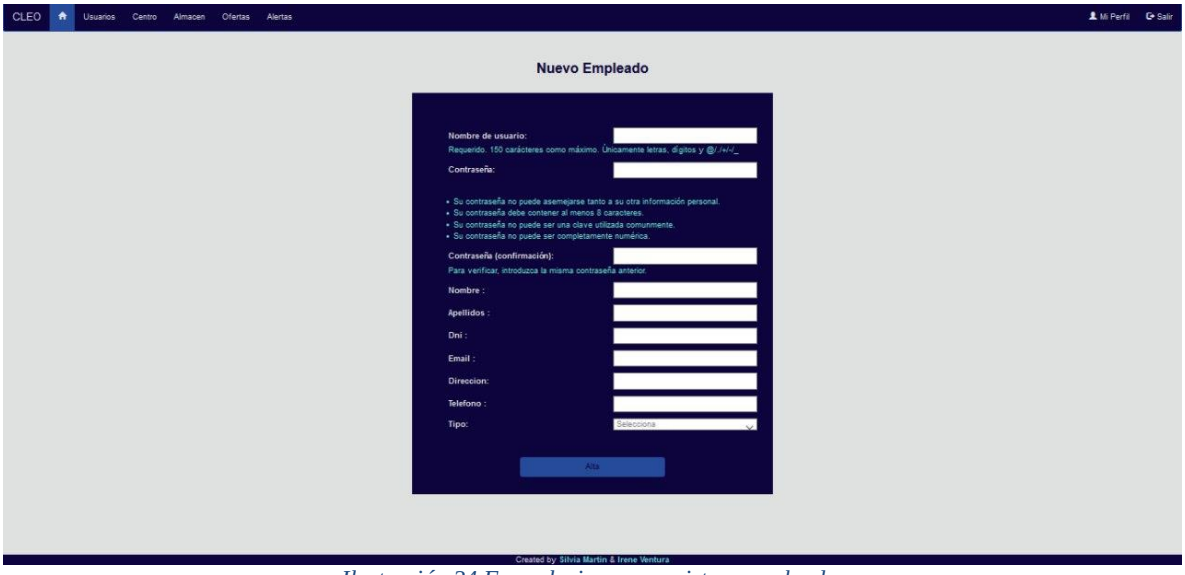


Ilustración 34 Formulario para registrar empleados

4.3.2.1 Básico

Cuando un empleado básico ha iniciado sesión, a diferencia del encargado, dispone de cuatro módulos: Usuarios, Centro, Almacén, Alertas, los cuales solo puede hacer modificaciones en el sistema para registrar nuevos clientes, eliminar clientes, realizar citas, facturar y para enviar notificaciones. Mientras que en el resto de las tareas que puede realizar son consultas al sistema.

Ilustración 35 Menú empleado básico

Ilustración 36 Opciones empleado básico

Otra característica importante es que al hacer solicitudes del tipo “Ver”, las listas muestran solamente el contenido, veamos el mismo ejemplo de “Ver Maquinas” desde la perspectiva de un empleado básico:

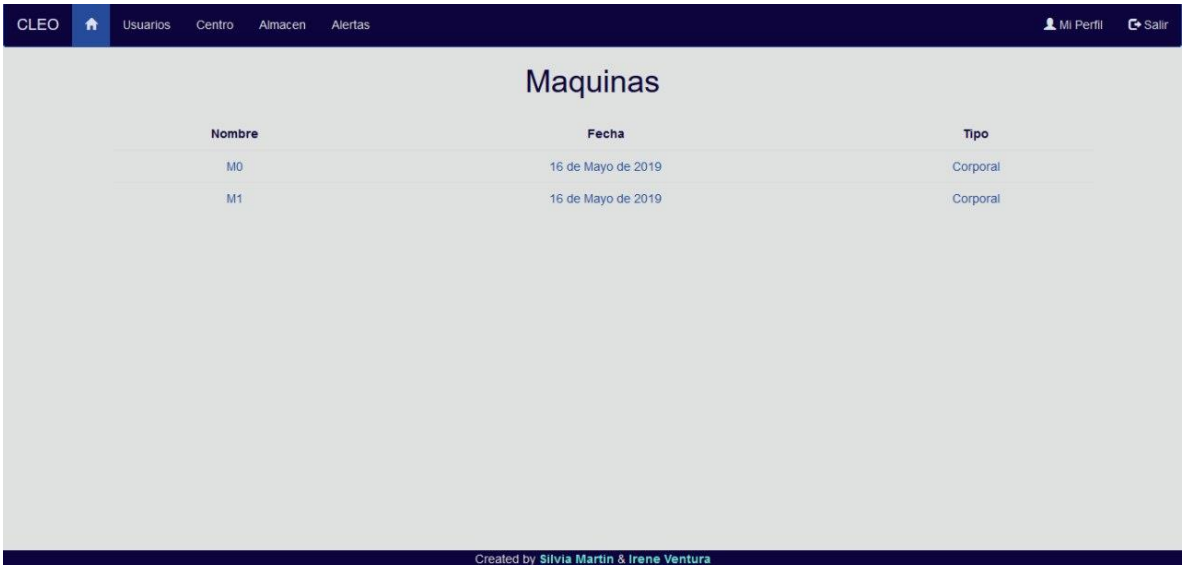


Ilustración 37 Vista lista de máquinas desde un empleado básico

4.3.3.- Guía de Administrador

Una vez se haya autenticado el administrador, se muestra la interfaz que le permite acceder a todos los componentes de que tiene la aplicación:

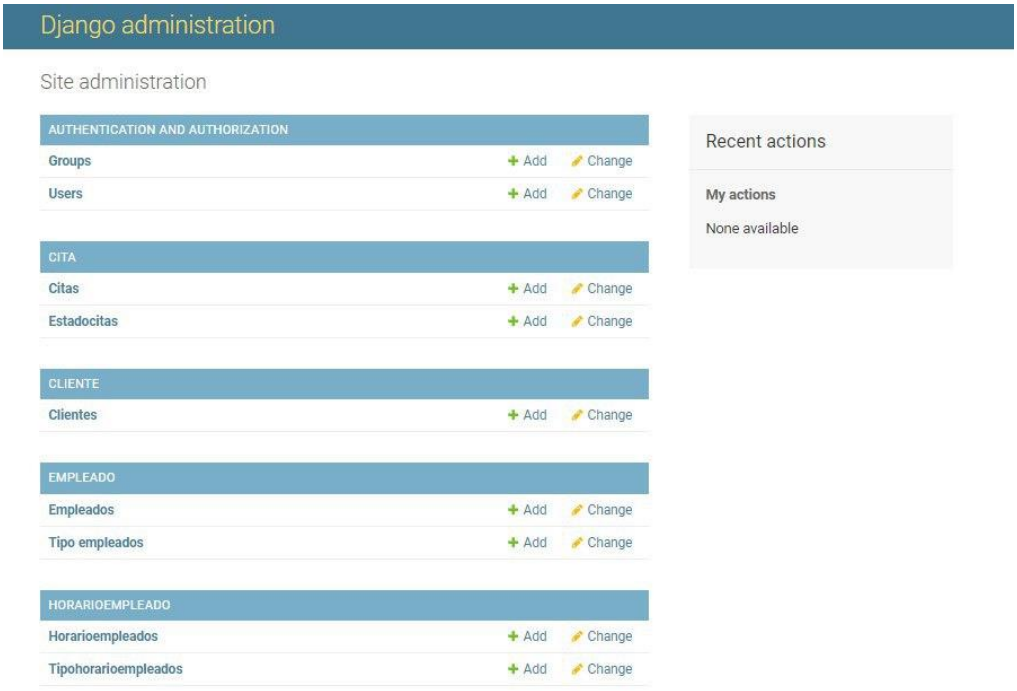


Ilustración 38 Vista principal administrador

Pero además de poder acceder y consultar todo en la aplicación, dispone de un módulo que le permite crear grupos de usuarios y asignarles permisos para que puedan o no realizar ciertas funciones, o incluso bloquear usuarios de la aplicación.

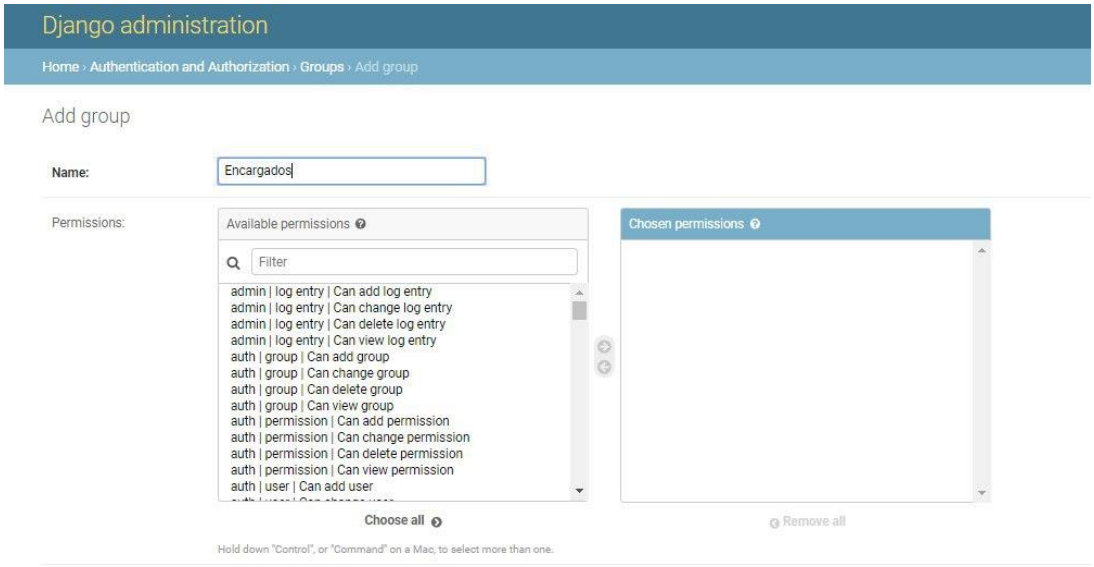


Ilustración 39 Formulario para crear grupos y conceder permisos

Y finalmente, si la organización decide ampliar opciones o modificar estados, el administrador puede modificar el contenido de etas tablas que tienen por defecto la estructura de CLEO, como es el caso de tipos los estados de cita:

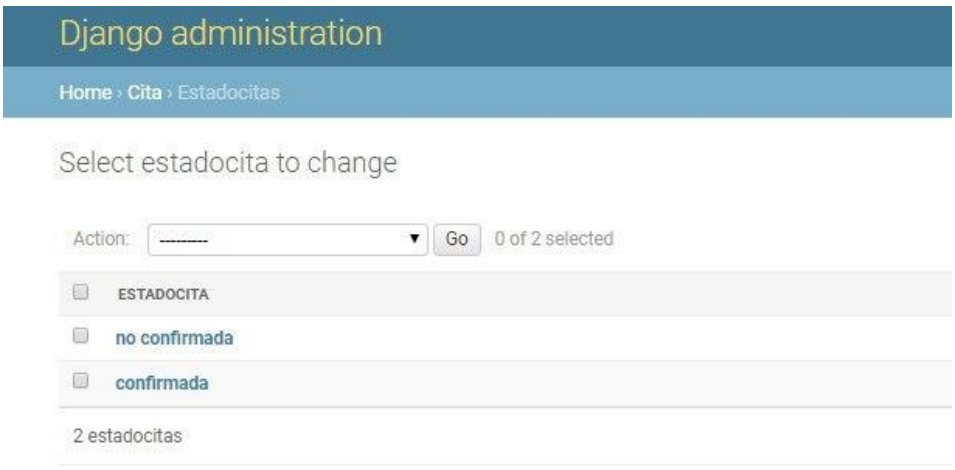


Ilustración 40 Lista Tabla de Tipo Cita

4.4.- Estimaciones

En este apartado vamos a mostrar, las entregas iterativas que hemos desarrollado a lo largo del proyecto y los tiempos que nos han ocupado cada una. Para cada entrega se ha considerado un periodo de dos semanas, a excepción del primero, que no lo tomamos como una entrega puesto que formó parte del proceso de investigación previo a empezar a programar.

Para una mejor compresión de los sprint, los íbamos coloreando siguiendo una leyenda:

Ilustración 41 Leyenda Estados Sprint

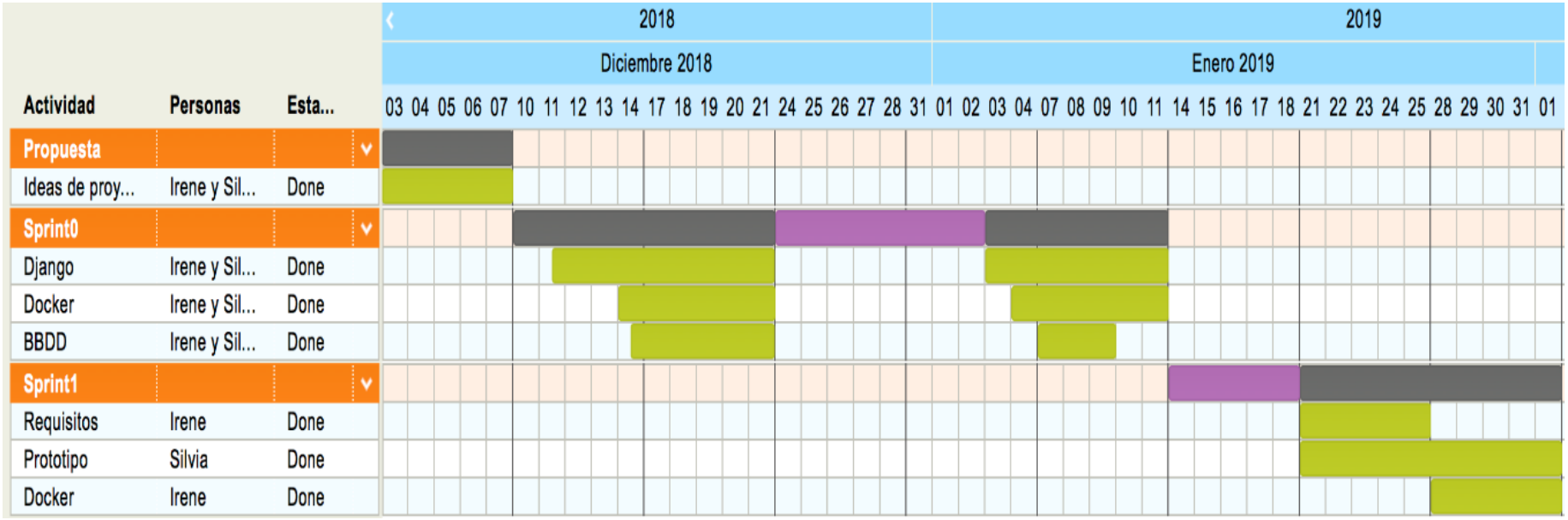


Ilustración 42 Estimación Diciembre y Enero

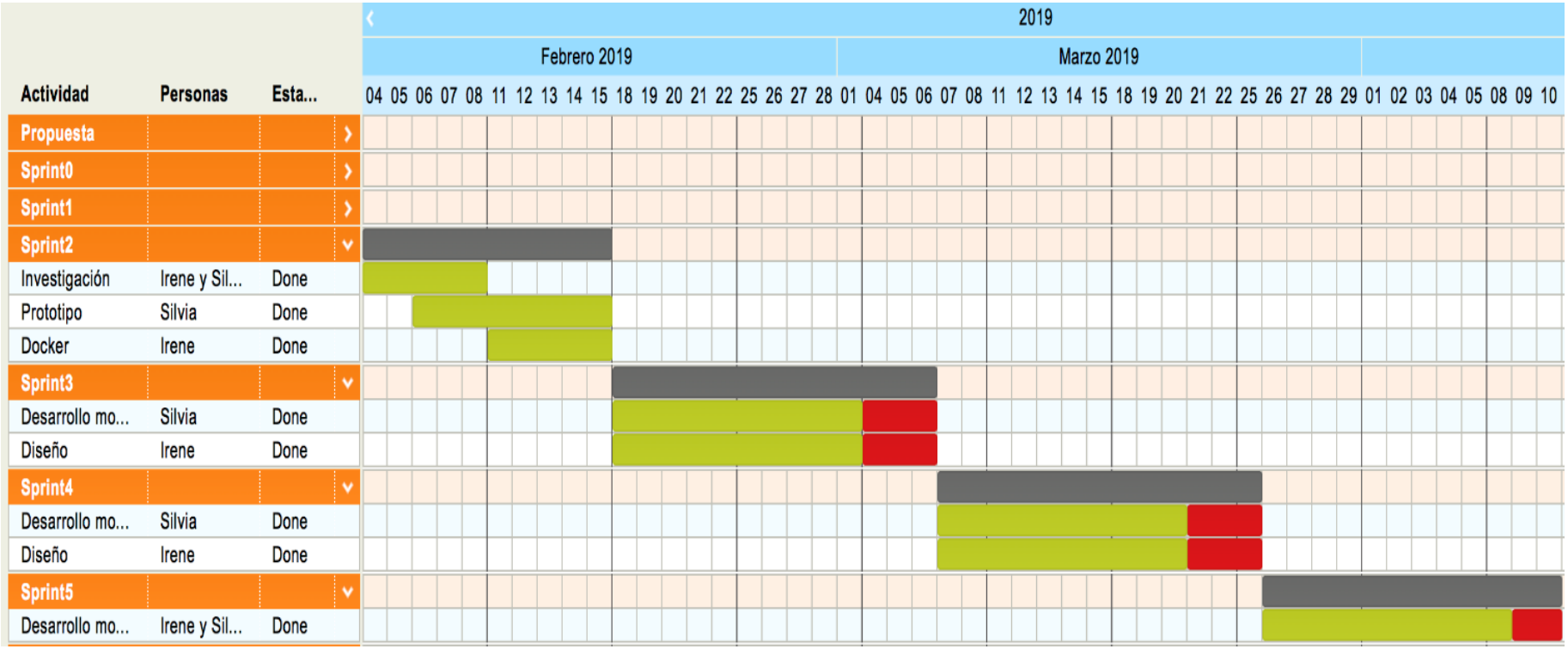


Ilustración 43 Estimación Febrero y Marzo

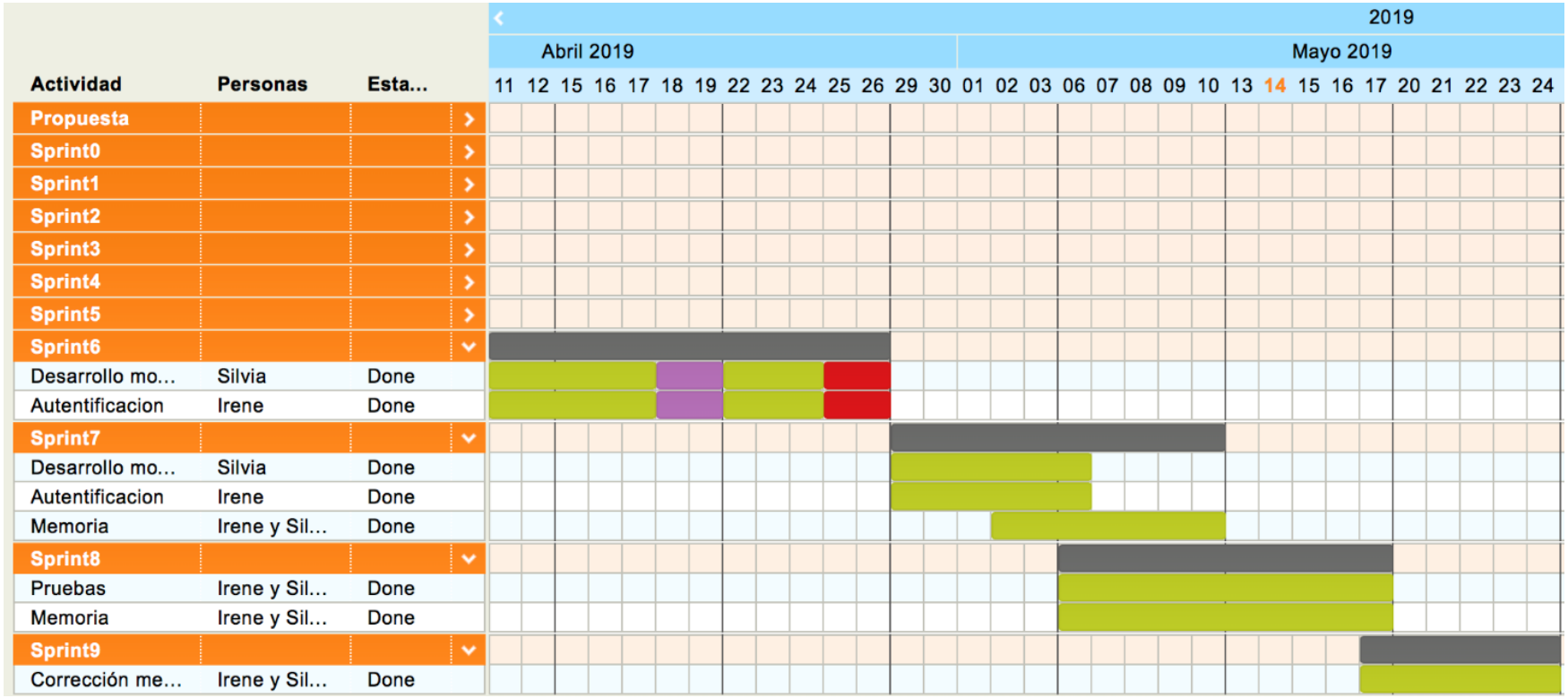


Ilustración 44 Estimación Abril y May

4.5.- Contribuciones al Proyecto

En este apartado vamos a exponer las contribuciones que hemos realizado en el proyecto. Cada una explica su experiencia y el proceso que ha realizado durante el desarrollo de la aplicación. Por lo general, ambas hemos estado implicadas en todas las partes del proyecto y las tareas han sido repartidas por igual.

4.5.1.- Parte Silvia Martín Gómez

Mis principales aportaciones a este proyecto han sido. Prototipo inicial, gestión de la base de datos, desarrollo de módulos (indicados en este punto) y gestión de envío de correos. Para poder llevar a cabo este proyecto, lo primero fue conocer nuestro framework de trabajo, Django, ya que lo desconocía por completo. Es por ello que mi primer contacto fue mirar la documentación oficial, para hacerme una idea de cómo trabaja y como se relacionan los distintos componentes de la aplicación; modelos, vistas, plantillas etc. A continuación, vi tutoriales de cómo se creaban los proyectos y dentro crear las aplicaciones que se van a ejecutar.

Otro punto fue el uso de la base de datos, en nuestro caso al trabajar con MySQL para la configuración en Django, tuvimos algunos problemas. En mi caso, el hecho de configurarlo con un MAC requiere opciones extras que hay que añadir, pero finalmente pude solucionarlo. Descargué la versión de mysqlclient directamente de la página https://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient y no importando el plugin desde Pycharm. Todo esto para poder hacerlo compatible con la base de datos. Además, tuve que cambiar las variables de entorno porque cogía por defecto Python2 y no Python3 y como interprete tengo Python3.7. Para manejar la base de datos, ambas acordamos que una sola persona debía ser la encargada de gestionar la base de datos y controlar las modificaciones en ella, en este caso, fue una de las tareas de las que me encargue de elaborar durante el desarrollo de la aplicación.

Una vez instalado el entorno y configurado la base de datos, comenzamos a investigar sobre los Dockers, para poder trabajar en un mismo sistema y obviar que las dos utilizamos sistemas operativos diferentes; esta idea fue una propuesta de nuestro tutor para evitarnos complicaciones. Sin embargo, esta investigación nos llevó varias semanas y como no veíamos avances, decidimos empezar a dividirnos el trabajo; yo comencé a realizar un prototipo de la aplicación y mi compañera siguió investigando acerca de los Dockers, pero finalmente, realizamos el proyecto usando PyCharm, un entorno de desarrollo que nos permite trabajar con Python y Django de forma sencilla, sin tener que hacer muchas configuraciones en nuestros sistemas operativos.

Como he mencionado anteriormente, el prototipo que diseñé contaba con las funciones CRUD de los módulos: clientes, empleados, salas, maquinas, tratamientos, proveedores y productos, evaluando la información mediante sesiones y realizando las interfaces graficas muy básicas, estas dos últimas tareas fueron mejoradas por mi compañera durante las siguientes entregas al cliente. Continuando con el desarrollo del prototipo, una de las principales complicaciones que tuve, fue el implementar los formularios de Django a nuestro proyecto, en especial aquellos que tienen comboBox y que sus opciones dependen de otros modelos de la aplicación, ya que no se actualizaban correctamente desde el form.py y es por ello que había que actualizar sus valores en la vista.

Ya hecho el prototipo con una parte de las entidades básicas, lo evaluamos de forma conjunta y apoyándonos en los requisitos que recogió mi compañera con el cliente, comenzamos a evaluar las entidades que nos faltaba añadir y también las modificaciones que había que hacer para adaptarnos a lo que el cliente nos estaba pidiendo.

A continuación, pasamos a publicar el prototipo en un repositorio en Github y poder trabajar las dos en el proyecto. El proceso de configuración lo realizó mi compañera, quien también me explicó cómo utilizar la herramienta y así poder sincronizar los cambios que íbamos haciendo. Este punto lo explicará ella en su apartado de contribuciones.

Las primeras tareas fueron bastante claras al asignarlas, Irene se encargó de la maquetación y diseño de toda la web y yo adapté los módulos que ya estaban con los nuevos requisitos. Al tener ya una base más sólida, nos repartimos los módulos y cada vez que los terminábamos, sincronizábamos los repositorios y revisábamos los módulos de la otra para validar y verificar si existían errores. Pero también es verdad que en las entregas finales Irene se encargó de hacer las pruebas y yo fui desarrollando los módulos que faltaban por añadir.

Durante el desarrollo de los módulos finales, noté que había ciertos requisitos de estos módulos que aún no estaban claros, por lo que coordiné una reunión con nuestro cliente para poder aclarar las dudas que teníamos en cuanto a lo que él quería de esos módulos. Esto me permitió diseñar los siguientes módulos tomando en cuenta lo siguiente:

a) Cita:

En este módulo era importante listar los tratamientos y que el cliente selecciona la fecha y el horario, ya que se toma cuenta la disponibilidad del centro para evaluar cuales son los tratamientos que pueden realizarse, según el tiempo que duran, como, por ejemplo: si un tratamiento dura tres horas la fecha máxima que puedes elegir son las 19:00 porque se cierra a las 22:00. Una vez seleccionado estos campos, el sistema nos indicará si es válida la cita o no. Si es válida la aplicación nos da la oportunidad de elegir al empleado y en el caso contrario él lo asigna de forma aleatoria. Finalmente, las citas se crean en un estado de no confirmadas y es el empleado quien la confirma y el sistema manda un correo al cliente indicando la fecha y la hora de la cita.

b) Factura:

En este módulo el empleado selecciona la cita de la cual quiere la factura. Solo aparecerán aquellas citas que tienen pendiente el pago. Puede ser la primera factura de la cita o porque el pago este fraccionado, donde aparece en cada factura el importe restante. Una vez que la factura se ha pagado (completa o parcialmente) no se puede modificar y tampoco eliminar.

c) *Stock*:

Para una mejor la comprensión del sistema se tuvo que añadir este módulo. Ya que el inventario puede ser modificado por los pedidos (añadir productos), por citas (productos utilizados en el tratamiento) y en la compra de productos (quitar productos del inventario y añadirlos en factura). Este módulo es fundamental a la hora de notificarle al empleado que tienen que reponer productos.

Finalmente, mi última tarea para el desarrollo de la aplicación fue crear los formularios y evaluarlos durante el periodo que desplegamos la aplicación, para luego analizarlos y reflejar en esta memoria los resultados y posibles elementos a añadir en versiones futuras.

4.5.2.- Parte Irene Ventura Farías

Mi parte de contribución a este proyecto, aparte del desarrollo de algunos módulos, fueron: el diseño (CSS) y maquetación (Bootstrap) de la aplicación, el proceso de autenticación de los usuarios (*login/register*) con el sistema de Django, la recolección de requisitos y pruebas de otros softwares similares en el mercado, la configuración de repositorios, las pruebas de los módulos y el despliegue de la aplicación para realizar las experiencias con los usuarios.

Mi proceso de investigación comenzó en cómo trabajar con Django. Una de las primeras diferencias que encontré con respecto a otras herramientas para desarrollo de aplicaciones web, es que Django te proporciona un servidor web, por lo que no necesité trabajar con Apache como lo habíamos hecho anteriormente con otras herramientas. Una vez teniendo el conocimiento de cómo crear un proyecto y aplicaciones en Django, realicé un par de tutoriales (DjangoGirls, Mozilla Django Tutorial, etc.) para entender la herramienta con más detalle y conocer el manejo de peticiones, como se elaboran los modelos, como se trabajan los datos en las vistas y como se elaboran los formularios y las plantillas dinámicas.

Seguidamente continúe mi investigación en cómo cambiar la configuración de la BBDD, ya que en Django por defecto está configurado para SQLite3 y nosotras debíamos configurarlo para trabajar con MySQL. Para esta configuración tuvimos ciertos impedimentos que hemos mencionado en el punto 2.4 de la memoria pero que pudimos resolver.

Como nuestro tutor nos sugirió que trabajásemos durante el desarrollo de CLEO con Docker, decidimos que nuestro siguiente paso sería investigar esta herramienta para poder trabajar desde un mismo entorno, ya que como hemos mencionado varias veces en la memoria, cada una trabajó desde un Sistema Operativo diferente. En esta fase, buscamos individualmente la información necesaria para crear el entorno desde un *docker-compose*. Debido a que era la herramienta que más desconocíamos, realizar la configuración fue bastante complicada porque Docker está desarrollada en Linux y como yo estuve programando desde Windows, para poder configurar Docker en este, necesitas una licencia Premium (Education o Pro) y mi conexión al proyecto de Django y la conexión de la BBDD fueron bastante complicados de configurar.

Ya que a Silvia también le surgieron errores al trabajar con Docker, decidimos que ella empezara haciendo un prototipo de lo que sería nuestra aplicación y que yo continuara investigando sobre la configuración de Docker. Finalmente, al darnos tantos problemas en ambos SO decidimos apartar la idea de trabajar con Docker durante el desarrollo y sustituirlo por PyCharm que es un IDE multiplataforma que proporciona trabajar con Django desde entornos virtuales de Python.

Para poder formalizar los requisitos de nuestra aplicación, me informe un poco de cómo trabajan otras aplicaciones similares a la que íbamos a desarrollar y ya que como la que tiene un parentesco con la persona que quería la aplicación era yo, pues tuve acceso a la licencia de software con la que en ese momento ella estaba trabajando (Flowww) así que me leí el manual de la licencia y conversé con ella un poco de los requisitos que debía tener la aplicación.

Una vez obtenidos los requisitos básicos de la aplicación, procedimos a hacer el primer diseño de la BBDD y adaptar el prototipo que Silvia había realizado con a los requisitos formales que el cliente nos proporcionó; sin tomar en cuenta por el momento las tablas que Django trae por defecto para el manejo de permisos, usuarios y autenticaciones.

A continuación, procedí a configurar en ambos SO, el repositorio con el que íbamos a trabajar para desarrollar la aplicación; un repositorio principal y un fork conectado remotamente. De manera que trabajábamos con dos repositorios que había que sincronizar cada cierto tiempo. Cada vez que el repositorio principal hacia cambios, el fork debía ser actualizado mediante un ***git pull upstream master*** y cada vez que había cambios en el remoto este mandaba una pull request al principal y se ejecutaba un rebase. Luego, le explique un poco a Silvia como había que manejar Git y Github para la sincronización de repositorios en caso de que tuviese que hacerlo ella, pero procurábamos, hacer estas sincronizaciones durante las reuniones.

A la hora del desarrollo de la aplicación la primera tarea que me toco realizar fue la estructura y maquetación (Bootstrap) de las plantillas de la aplicación, así como el diseño visual de la aplicación. Una vez aclarada el diseño de la interfaz, nos repartíamos el trabajo por módulos; empezando por los más sencillo y finalizando por los más grandes.

Al realizar un módulo, cada una de nosotras realizábamos las pruebas de unidad correspondientes para comprobar que funcionaban correctamente. Pero cuando ya teníamos un gran porcentaje de módulos realizados, empecé a hacer las pruebas de integración de forma manual, de manera que yo hacia las pruebas y Silvia iba corrigiendo los errores.

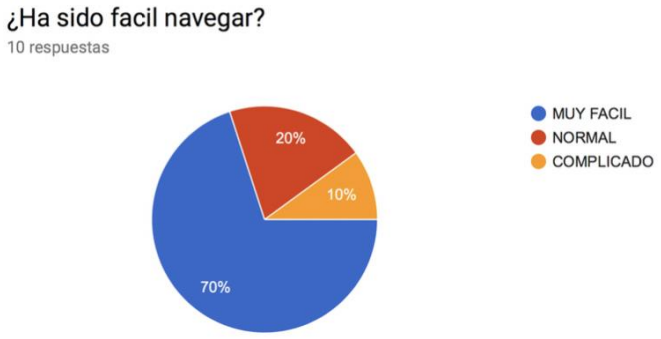
Finalmente me tocó realizar la configuración de usuarios en la aplicación, adaptando la estructura con las que trabaja Django por defecto a nuestra BBDD, modificando el proceso de registro y acceso de usuarios, así como también que se registren los valores por defecto de las tablas tipo y/o estado, lo grupos y permisos de usuario entre los módulos de la aplicación mediante decorators y el despliegue de esta al momento de hacer las pruebas de sistema y validaciones.

Capítulo 5: Experiencias de usuario

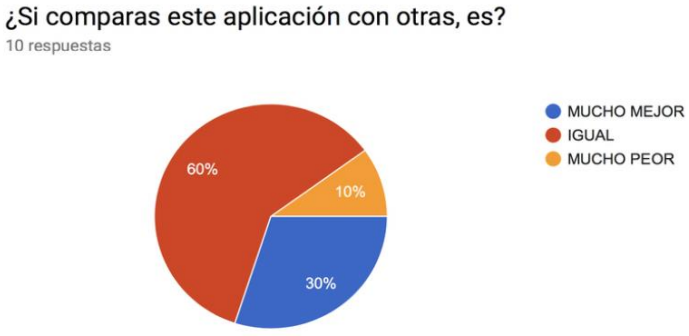
5.1.- Experiencias de Usuario

En este apartado mostramos las encuestas realizadas a los usuarios, tanto clientes finales, como usuarios de la aplicación. La encuesta se ha basado en experiencias reales de diez usuarios, cinco usuarios eran empleados y los otros cinco eran clientes; donde los usuarios hicieron pruebas durante un cierto tiempo y más tarde contestaron la encuesta. También es importante mencionar que al ser 10 personas las encuestadas, el número de personas corresponde con el porcentaje y que los resultados obtenidos serán tomados en consideración para las modificaciones en las versiones futuras.

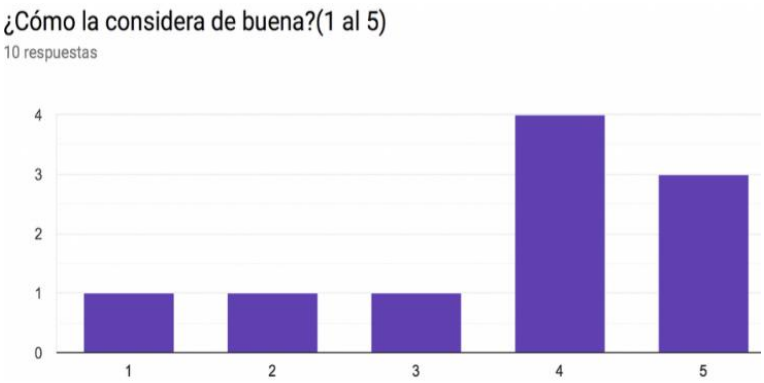
La información que a continuación explicamos se obtuvo a partir de esta encuesta: https://bit.ly/2We8Sao



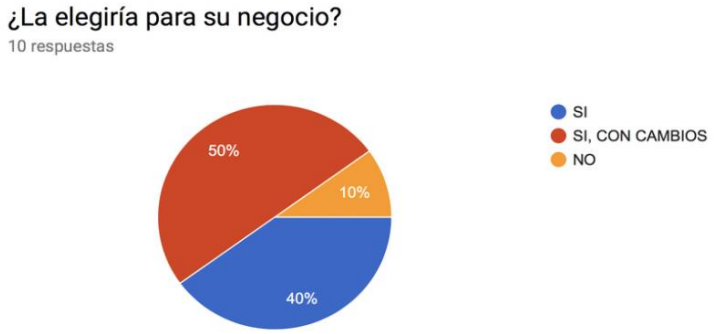
En esta pregunta hace referencia al entorno y diseño de la aplicación; si le ha resultado intuitiva y amigable al usuario. Como se ve en la encuesta, el entorno es amigable para los usuarios, ya que 7/10 personas han dicho que es muy fácil trabajar con la aplicación.



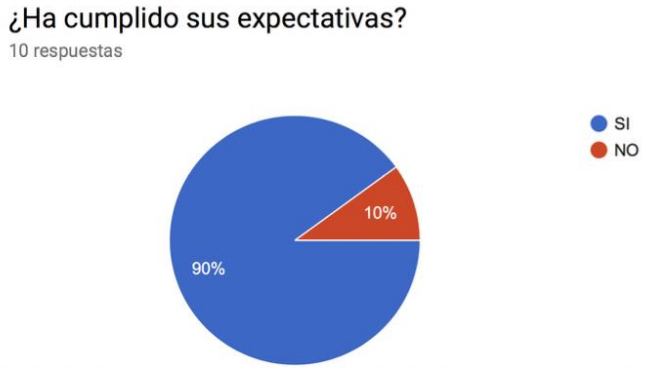
La pregunta es clara, y como se aprecia estamos, en general, al mismo nivel que el resto de las aplicaciones, puesto que 6/10 personas encuentran la aplicación similar a otras en el mercado.



Pregunta que recoge la visión general que ha tenido el usuario de nuestra aplicación. Siendo 1 mala y 5 muy buena. La cual permite concluir que ha gustado la aplicación, pues la mayoría de los usuarios (4/10 con puntuación de 4 y 3/10 con puntuación 5)



Con este resultado estamos muy satisfechas porque por lo general, con un 90%, nuestra aplicación seria elegida. Solo una persona no está convencida de usar CLEO en su negocio.



Al igual que la pregunta anterior, estamos muy contentas con este resultado. Significa que hemos podido desarrollar los requisitos y expectativas propuestos por el cliente, porque exceptuando a una sola persona, el resto de los usuarios se sienten satisfechos con nuestro trabajo.

Haciendo un estudio de todas las respuestas, se puede observar, que los requisitos se han llevado a cabo en un alto grado y que los usuarios en su mayor parte han quedado satisfechos. Esperamos, en versiones futuras, poder mejorar las encuestas y tener un mayor grado de satisfacción de los usuarios, atendiendo sobre a todo a los que ha puntuado de forma más baja.

Capítulo 6: Resultados y Conclusiones

Finalmente, el último capítulo de nuestro TFG detalla cómo evaluamos la experiencia de un grupo de clientes y empleados que hicieron uso de la aplicación. Consultando con cada uno su opinión acerca de CLEO, qué características les gusto, cuáles no y qué les gustaría que la aplicación tuviese en un futuro. Esto nos permitió a nosotras concluir que posibles cambios habría que realizar en versiones futuras y que otras funciones podríamos añadir. Ello nos permitió evaluar si cumplimos con los objetivos que nos planteamos en un principio. Y, finalmente, evaluar el tiempo total empleado en el desarrollo de la aplicación.

6.2.- Versiones Futuras

Los requisitos básicos de la aplicación fueron alcanzados, sin embargo, durante el desarrollo y en las pruebas de validación, se propusieron nuevas funcionalidades que pueden ser añadidas en versiones futuras, así como también un único objetivo que no pudimos realizar en esta primera versión que fue el de mejor uso a los Dockers:

a) Entornos Docker durante el desarrollo para versiones futuras

Ya que en esta versión no pudimos trabajar con Docker durante el desarrollo; puesto que fue la herramienta que más nos costó implementar debido a que era totalmente desconocida para nosotras. Esperamos añadirla al sistema de desarrollo en nuevas versiones, porque sabemos que nos facilitará el proceso de desarrollo al encapsular el entorno de producción e independizándolo de los SO.

b) Suscripción con doble confirmación

Como esperamos que la aplicación vaya creciendo, llegará un punto en que la información de nuestros usuarios sea sensible y para protegerlo de otras personas. Es por ello que nos gustaría que al crear una cuenta de usuario o al cambiar su contraseña, esta cuente con un proceso de verificación.

c) Autenticación delegada para registrarse en CLEO

Para facilitar el registro de nuevos clientes a la aplicación, nos gustaría implementar autenticación que sea delegada desde otras aplicaciones (Google, Twitter, etc.), esto también nos permitirá integrar funciones con estas aplicaciones (logros, obtener amigos, obtener imagen de perfil, etc.)

d) Integrar pasarelas de pago

Nos gustaría añadir la opción de pagar tratamientos y productos, facilitando y conectando el proceso de pago en nuestra aplicación. Ya que, con él, podríamos ofrecer diferentes métodos de pago, no habría limitaciones geográficas y podríamos hacer un seguimiento de las transacciones económicas.

6.3.- Conclusiones

Tras finalizar el desarrollo de la aplicación, podemos concluir los siguientes resultados:

- A pesar de que a nivel funcional hayamos alcanzado todos los objetivos que el cliente quería que tuviese la aplicación, se detectaron nuevas necesidades a medida que íbamos avanzando en el proyecto y en las pruebas realizadas con los usuarios. Algunas por ser sencillas fueron incorporadas al proyecto y otras por la complejidad que tenían, se dejaron para desarrollarlas y añadirlas en versiones futuras que son mencionadas en el punto anterior.

- Adquirimos nuevos conocimientos de nuevas tecnologías (Python, Django, Docker, etc.) que podrán servirnos para proyectos futuros y que nos puede ser útil a la hora de dedicarnos al trabajo autónomo o de buscar trabajo por cuenta ajena. Ya que Python es una herramienta que puede utilizarse en múltiples áreas y Docker puede ayudarnos a la configuración de diferentes entornos que necesitemos al trabajar.

- Haber gestionado el proyecto con metodologías ágiles nos ha permitido considerar cambios durante el desarrollo del proyecto, ya que, al realizar entregas iterativas, los cambios e impedimentos tendían a ser pequeños y fáciles de añadir o resolver.

- La buena comunicación entre ambas nos ha permitido que el nivel de errores e impedimentos fuese bastante pequeño, ya que ambas hemos mantenido reuniones constantes durante todo el proyecto y en cada reunión evaluábamos cual era la mejor solución para trabajar. Ambas hemos considerado que debíamos mantener el mismo nivel de conocimiento acerca del proyecto para poderlo sacar adelante.

- Poner en funcionamiento la aplicación por un largo periodo con usuarios reales, nos permitirá considerar que la aplicación es útil y que se pueden incluir cambios que aporten mejores soluciones para el negocio.

6.4.- Conclusions

After having ended the development of the application, we can conclude the next results:

- Despite that we have completed all the functionality required by the client, we were developing the modules and after the validation by real users. Some of them were added for their simplicity and others were added like objectives for future releases.

- We acquire new knowledge of new technologies (Python, Django, Docker, etc.) that can be useful for future projects or when we are looking for a job. Especially considering that Python is a tool that can be used in multiple areas and Docker can help us to configure different environments that we would need.

- Having managed the project with agile methodologies has allowed us to make changes in the development of the project easily Because we have made iterative deliveries and changes. Moreover, the impediments were tended to be small and easy to solve.

- The good communication between both of us has allowed the level of errors and impediments have remained quite small. There have been constants meeting throughout the project. In each meeting, we have evaluated which was the best solution to work. We have considered that both of us should have the same level of knowledge about the project in order to be able to move forward.

- Implementing the application for a long period with real users will allow us to consider if the application is useful and make the changes that provide better solutions for the business.

Bibliografía

[1] «Documentación Oficial de Python,» [En línea]. Available: https://www.python.org/.

[2] «Django Girls Tutorial,» [En línea]. Available: https://tutorial.djangogirls.org/es/django_start_project/.

[3] «Mozilla Django Tutorial,» [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/Server-side/Django.

[4] «Configuracion MySQL Django,» [En línea]. Available: https://medium.com/@a01207543/django-conecta-tu-proyecto-con-la-base-de-datos-mysql-2d329c73192a.

[5] «Django & Docker - Platzi,» [En línea]. Available: https://platzi.com/blog/django-docker/.

[6] «Encriptar contraseñas en Python- Codigo Facilito,» [En línea]. Available: https://codigofacilito.com/videos/encriptar_contrasenas_en_python_bytes.

[7] «Introduccion a Python - Cursos FDI,» [En línea]. Available: https://cursosinformatica.ucm.es/curso-introduccion-programacion.html.

[8] «Paginas Web con HTML5 - Curso FDI,» [En línea]. Available: https://cursosinformatica.ucm.es/curso-web-html5.html.

[9] "Testing a Pull Request - Github," [Online]. Available: https://github.com/TeamPorcupine/ProjectPorcupine/wiki/How-to-Test-a-Pull-Request.

[10] "Multiple Users on Django," [Online]. Available: https://simpleisbetterthancomplex.com/tutorial/2018/01/18/how-to-implement-multiple-user-types-with-django.html#practical-example.

[11] "Roles de Usuario -StackOverFlow," [Online]. Available: https://es.stackoverflow.com/questions/930/roles-de-usuarios-en-django.

[12] «Documentación Oficial Docker,» [En línea]. Available: https://www.docker.com/.

[13] "How To Restrict LoginView -StackOverFlow," [Online]. Available: https://stackoverflow.com/questions/41619368/how-to-restrict-login-view-after-user-logged-in.

[14] «Documentación Oficial Django,» [En línea]. Available: https://www.djangoproject.com/.

[15] «Docker Build Error en Mac,» [En línea]. Available: https://stackoverflow.com/questions/28996907/docker-build-requires-1-argument-see-docker-build-help.

[16] «Django with mysql and phpmyadmin in Docker -Medium,» [En línea]. Available: https://medium.com/@pierangelo1982/django-with-mysql-and-phpmyadmin-in-docker-30b9385c4247.

[17] "Custom User Model on Django," [Online]. Available: https://wsvincent.com/django-tips-custom-user-model/.

[18] "Herramienta para Evaluar Estimaciones - TomsPlanner," [Online]. Available: https://plan.tomsplanner.es .

Anexos

En la parte de anexos vamos a incluir información adicional acerca de la base de datos, donde expondremos las más destacadas y por último alguno de los acrónimos usados en este trabajo.

Anexo I. Tablas de la Base de Datos

En este punto, vamos a mostrar algunas tablas de la BBDD del sistema. Se mostrarán las tablas que consideramos más interesantes y el documento completo se podrá encontrar en repositorio en el capítulo 4 en la guía de instalación. El archivo .sql ha sido generado por Workbench ya que nosotras no hemos implementado las tablas, sino que hemos utilizado los modelos de Django (explicado en punto 2.4 apartado modelo)

Este Create corresponde a la tabla pedido, en ella se ve como un pedido guarda un producto (este tiene su proveedor) y el estado del pedido, que como se ha comentado antes, dependiendo del estado el pedido pasa a la tabla inventario o no. Tenemos como PK el id utilizado para filtrar mejor y como únicos el id, el estado y el producto.

Pedido		
Id	int(11)	(PK) Valor auto incrementado.
Cantidad	int(11)	Cantidad de los productos pedidos.
Total	double	Total del pedido realizado.
Fecha	Date	Fecha en la que se realiza el pedido.
estadoPedido_id	int(11)	(FK) Estado en el que se encuentra el pedido, cancelado, entregado, en proceso.
Producto_id	int(11)	(FK) Instancia del producto.

La tabla factura es una de las más completas porque engloba a cita, desarrollada en las próximas imágenes). Como se puede ver, factura tiene como foreign key a cita, que es la principal para completar la factura y el estado de la factura que puede ser pendiente o pagada. Si es pendiente, se irá restando el precio del tratamiento según vaya pagando el cliente. De nuevo un id como primary key para un mejor control.

Factura		
Id	int(11)	(PK) Valor auto incrementado.
costePorCobrar	double	Diferencia del precio total del tratamiento, se actualiza con cada pago del cliente.
Total	double	Total que paga el cliente en cada factura.
Fecha	Date	Fecha en la que se realiza la factura.
Cita_id	int(11)	(FK) instancia de la cita sobre la que se hace la factura.
estadoFactura_id	int(11)	(FK) estado de la factura, pendiente o pagada.

Empleado, tiene los datos básicos para registrar a un empleado. Como primary key tenemos user_id que es un id que hace referencia a la tabla usuarios_customer (detallada a continuación). Añadimos como unique el dni y el tipo de empleado.

Empleado		
User_id	int(11)	(PK) Instacia de usuario
Dni	varchar(9)	DNI del empleado
Código	varchar(100)	Código del empleado
Dirección	varchar(45)	Dirección del empleado
Teléfono	varchar(9)	Teléfono del empleado
tipoEmpleado_id	int(11)	(FK) Instancia del tipo de empleado, básico o encargado.

Tabla usuarios_customer, utilizada por las entidades empleado y clientes para hacer una base de los usuarios que utilicen la aplicación. Con esta entidad gestionamos los roles de los empleados.

Usuarios_customer		
Id	int(11)	(PK) Valor auto incrementado.
Password	varchar(128)	Password del usuario.
last_login	dateTime(6)	Fecha del último acceso.
is_superuser	Tinyint(1)	Nos indica si es super usuario o no ➔ admin
Usernmae	varchar(150)	Nombre que usará para entra en la aplicación.
first_name	varchar(30)	Nombre del usuario.
last_name	varchar(150)	Apellido del usuario
Email	varchar(254)	Email del usuario.
is_staff	Tinyint(1)	Nos indica si es staff ➔encargado
Is_Active	Tinyint(1)	Nos indica si está activo el usuario.
date_joined	dateTime(6)	Fecha en la que se dio d alta en el sistema.
is_client	Tinyint(1)	Nos indica si es cliente

Esta entidad posiblemente es la más completa de todas y quizá la más compleja también, ya que engloba varias entidades. Como vemos cita asocia cliente, empleado, tratamiento y los horarios de la cita (que también son entidades de Horario) la clave la forman el id (como siempre para un mejor manejo y control), el estado (puede ser confirmada o no confirmada), cliente, empleado, los horarios y el tratamiento que se realiza.

Cita		
Id	int(11)	(PK) Valor auto incrementado.
Fecha	int(11)	Fecha de la cita
Cliente_id	int(11)	Instancia del cliente que pide la cita.
Empleado_id	int(11)	Instancia del empleado que realiza la cita.

estadoCita_id	int(11)	Instancia del estado de la cita, confirmada no confirmada.
horarioEntrada_id	int(11)	Instancia de la hora de inicio de la cita.
horarioSalida_id	int(11)	Instancia de la hora de fin de la cita.
Tratamiento_id	int(11)	Instancia del tratamiento que se realiza en la cita.

Anexo II. Acrónimos

A continuación, detallamos los acrónimos que se han nombrado en la memoria:

a) **TFG:** Trabajo Fin de Grado.
b) **BBDD:** Base de datos.
c) **SO:** Sistemas operativo.
d) **HTML:** Hypertext Markup Language,
e) **CSS:** Cascading Style Sheets
f) **HU:** Historias de Usuario
g) **PK:** Primary Key
h) **FK:** Foreign Key