Nombre y apellido del autor: **Kern Matías Nicolas**

Título del Proyecto Final: **Implementación de Sistema de Proveedores (SP) para las empresas "Servicios Agropecuarios Amanecer S.R.L" y "Otermin S.A"**

Grado académico alcanzado: **Ingeniero en Sistemas**

Unidad Académica a la cual pertenece: **Facultad de Ingeniería de la UNLPam**

Fecha de aprobación: **13/11/2021**

Nombre y apellido del director: **Hugo Alfonso**

Cátedra: **Base de Datos**

Nombre y apellido y co-director: **-**

Nombre, apellido y filiación institucional del Jurado: **Carolina Salto, Gabriela Minetti, María de los Angeles Martín**

Resumen en español de hasta 200 palabras: **El presente trabajo final contempla el desarrollo de un software realizado a medida, para las empresas Amanecer Servicios Agropecuarios S.R.L y Otermin S.A. Dicho sistema, actualmente, está siendo utilizado por las firmas, reduciendo los tiempos de operabilidad en la carga de datos, cálculos y confección de informes. Para la realización del mismo, se utilizó la Metodología Iterativa Incremental en conjunto con los frameworks Django y Bootstrap, PostgreSQL como sistema gestor de base de datos y GitHub.**

Palabras claves (4 a 6): **Metodología Iterativa Incremental, GIT, Django, Bootstrap, PostgreSQL.**

Palabras claves (4 a 6): **This final work contemplates the development of a software made to measure for the companies Amanecer Servicios Agropecuarios S.R.L and Otermin S.A. This system is currently being used by them, reducing operability times in data loading, calculations and reports making. To carry it out, the Iterative Incremental Methodology was used in conjunction with Django and Bootstrap frameworks, PostgreSQL as database management system and GitHub.**

Key Words (4 a 6): **Incremental Iterative Methodology, GIT, Django, Bootstrap, PostgreSQL.**

PROYECTO FINAL

IMPLEMENTACIÓN DE SISTEMA DE PROVEEDORES (SP) PARA LAS EMPRESAS “SERVICIOS AGROPECUARIOS AMANECER S.R.L” Y “OTERMIN S.A”

Universidad Nacional de La Pampa

Facultad de Ingeniería

Ingeniería en Sistemas Plan 2004

Autor: Kern, Matías Nicolas

Director: Alfonso, Hugo

Fecha: 07/11/2020

Agradecimientos y dedicatoria

Llegar hasta este punto significó transitar un camino lleno de desafíos tanto académicos, como personales. Haber dedicado tantas horas de estudio tuvieron su recompensa, la cual quiero compartir con todas aquellas personas que me acompañaron en esta etapa.

Primer quiero darle las gracias a mi familia, en especial a mis padres, que hicieron todo lo posible para que pueda estudiar una carrera en la universidad pública, gratuita y de calidad. Gracias por guiarme y aconsejarme durante mi vida de estudiante.

Agradecer a mis amigos/as que tuve antes, durante y después de mi paso por la universidad, por tantas horas de estudio, acompañadas con mates y risas de por medio. Hicieron que el camino sea mucho más alegre y divertido.

También, darle las gracias a Marina Capponi y su familia, por dejarme desarrollar este proyecto con su empresa Amanecer Servicios Agropecuarios S.R.L.

A todos los docentes y personal de la facultad también les doy las gracias. De ellos, me llevo el conocimiento y buenas prácticas que definen a un egresado de la Facultad de Ingeniería. Mi gratitud, en especial, a Hugo Alfonso por el apoyo y los consejos brindados a lo largo del desarrollo de este proyecto.

Por último, pero no menos importante, agradecer y dedicarle este trabajo a una persona muy especial en mi vida y que ya no está conmigo para compartir este logro. A mi abuela, que con su alegría y palabras logró sacarme una sonrisa en momentos difíciles.

Índice

Índice de ilustraciones

Resumen

El presente proyecto contempla el desarrollo de un software realizado a medida para las empresas Amanecer Servicios Agropecuarios S.R.L y Otermin S.A, ambas radicadas en la ciudad de General Pico, La Pampa. Dichas empresas presentaron en su momento la necesidad de automatizar ciertos procesos de carga de información de naturaleza contable y viajes realizados por su flota de camiones, confección de informes sobre movimientos y transporte; y obtener estadísticas para procesos de toma de decisiones de tipo gerencial.

Utilizando herramientas de Ingeniería de Software es que se da solución a las problemáticas mencionadas en el párrafo anterior. A través de la metodología iterativa incremental es que se dividió el desarrollo del sistema en iteraciones que fueron desarrollándose conforme avanzaba el proyecto. También utilizando diferentes tecnologías web se construyó un sistema que responde correctamente a los requisitos de las empresas, que en gran medida eran coincidentes.

Actualmente, el sistema se encuentra en producción y siendo utilizado por los actores involucrados a este, logrando reducir los tiempos de carga de datos, cálculos y confección de informes.

Las empresas Servicios Agropecuarios Amanecer S.R.L y Otermin S.A contemplan en su mayoría, los mismos requerimientos solicitados para el desarrollo correspondiente, con la particularidad de que se establecieron una serie de funcionalidades adicionales para trabajar con Servicios Agropecuarios Amanecer S.R.L. Esto deriva en que el mismo desarrollo logra ser aplicado a dos empresas con dominios de problemas donde el sistema de Otermin S.A. se reduce a un conjunto de funcionalidades comunes, motivo por el cual el presente informe se realiza sobre Servicios Agropeciarios Amanecer SRL.

Capítulo 1 – Introducción

1.1 Problema

A modo introductorio se enunciará el problema, los objetivos propuestos, la metodología para alcanzar los mismos y los resultados esperados del presente trabajo. Servicios Agropecuarios Amanecer S.R.L es una empresa radicada en la ciudad de General Pico, La Pampa, cuenta con más de 14 años de experiencia en el rubro agropecuario. Gestionada por el núcleo familiar, quien además está a cargo de su administración. En cuanto a su actividad comercial, se destacan los servicios de siembras, labranzas y fletes de distintos tipos de cargas.

Toda la gestión de la empresa es llevada a cabo por una de sus dueñas, la cual deja constancia de la contabilidad del negocio en planillas de cálculos alojados localmente en una de las computadoras que posee la firma, con el consecuente problema de pérdida de información en caso de que se dañe el disco. Tales archivos, almacenan el flujo de ingresos y egresos de dinero que presenta la empresa, a la vez de que son utilizados como detalle adjunto a facturas de clientes y proveedores, que luego son enviadas al contador. Además, resulta sumamente laborioso obtener información detallada acerca de los ingresos y egresos a lo largo del tiempo, siendo que toda la información importante requiere previamente ser procesada de forma manual para luego ser volcada en gráficos estadísticos. En cuanto al servicio de fletes que la empresa provee, la forma de trabajar presenta la misma dinámica dificultosa que la mencionada anteriormente, con la salvedad de que en dichas planillas se registran los viajes realizados por los choferes y a partir de allí, se calcula un adicional salarial. Por último, es de vital importancia generar informes mensuales sobre los movimientos que avalen las facturas presentadas durante el período, siendo esa tarea, también desarrollada de forma manual por la dueña de la firma.

Frente a este contexto, es que surge la necesidad de implementar una solución de software, creada a la medida para responder a las necesidades de la empresa. De esta manera, muchas de las operaciones de carga de información, generación de informes detallados y de cálculos de adicionales, se realizarán de forma automatizada y detallada.

1.2 Objetivos de trabajo

1.2.1 Objetivos de la empresa

Los objetivos que propone la empresa son amplios, pero con un nivel sencillo de dificultad, donde el punto más importante para lograr los objetivos propuestos radica en la instalación y configuración del entorno de desarrollo y también en la comprensión de los procesos de negocio que esta empresa posee.

Por lo expuesto anteriormente, a continuación, se detallan los siguientes objetivos a cumplir:

- Relevar, analizar y mejorar los procesos de carga de información contable por parte de los usuarios al sistema. Además de familiarizarse con cuestiones propias de contabilidad y sistemas de organizaciones.
- Analizar y sugerir cambios en los modos de trabajo y carga de información relevante para la empresa.
- Diseñar cómo se llevará a cabo la implementación de la solución de software, previamente avalada por la empresa.
- Relevar, analizar e implementar procesos para generar información contable útil para la toma de decisiones de la empresa.
- Diseñar un sistema multiusuario y multiplataforma escalable, a la vez de poder otorgar permisos para ciertos tipos de usuarios.

1.2.2 Objetivos académicos

Basado en la baja complejidad de las tareas que se desarrollaran para cubrir las expectativas que se desprenden de los objetivos planteados por la empresa, se decidió complementar el trabajo con una memoria descriptiva que sea de utilidad para estudiantes que estén enfrentándose a sus primeros desarrollos de sistemas, y explicar detalladamente las tecnologías, la metodología y los conocimientos necesarios para la correcta resolución del problema planteado.

Así, la elaboración de este informe pretende ser una guía práctica para la/s materia/s que así lo requiera/n, dicha guía incluirá la instalación de un conjunto de herramientas útiles para un correcto desarrollo de mediana envergadura, utilización de sistemas de control de versiones, deploy en un servidor que tendrá la función de alojar la solución de software del presente proyecto. Todos estos temas, serán tratados en mayor profundidad conforme se desarrolle el presente informe.

Además, se pretende integrar multitud de contenidos adquiridos durante la cursada de las diversas materias correspondientes a la carrera Ingeniería en Sistemas Plan 2004.

1.3 Metodología de trabajo

La modalidad seleccionada para este trabajo es "Proyecto de desarrollo", ya que es la indicada para llevar a cabo este trabajo que responde a una problemática real de una empresa de la ciudad. Y como tal, la primera actividad relacionada al proyecto fue la de realizar una entrevista con la dueña de la empresa donde esta persona explicó de manera breve y concisa el problema que se le presentaba para desempeñar su actividad. Luego de este primer contacto, se procedió a una serie de entrevistas más formales, donde se tomó conocimiento detalladamente acerca de la actividad que realiza la empresa Servicios Agropecuarios Amanecer S.R.L, sus procesos de manejo de información, áreas críticas y generación de información para la toma de decisiones. Además, la dueña aportó planillas a modo de ejemplo, dejando constancia de los datos que guardan, su ordenamiento y fórmulas para realizar cálculos sobre las mismas.

Luego, se seleccionó un framework de desarrollo web para implementar el sistema; además, se realizó una investigación previa para seleccionar el motor de base de datos que mejor se integrara al framework seleccionado y posteriormente el servicio de hosting a contratar para alojar el proyecto. Todo lo mencionado anteriormente, se acopló en su momento a un sistema de control de versiones a modo de resguardar el código generado y poder realizar actualizaciones en el momento que sea necesario.

En cuanto a la elección del framework, cabe mencionar que se tuvieron que adquirir las nociones y conocimientos básicos acerca de su instalación, configuración y utilización para posteriormente comenzar con el proceso de desarrollo, amparado en la documentación oficial como guía para dar los primeros pasos.

Dada la naturaleza del proyecto, y la dificultad relativamente sencilla que posee, es que se elige la metodología iterativa incremental, donde el desarrollo del sistema en su totalidad se dividió en varias iteraciones, donde cada una de estas comprende el análisis, desarrollo, prueba e implantación para una vez finalizada, se procede con la siguiente iteración, y así hasta finalizar el proyecto en su totalidad. Dicha metodología será tratada con mayor detalle en el Capítulo 3.

Luego de varias reuniones con representantes de las empresas, se dio inicio al desarrollo del sistema que presentaba el conjunto de funcionalidades mayor, comenzando por la primera iteración. Una vez finalizada cada iteración, se daba inicio a la iteración siguiente y así sucesivamente. Luego de completar cada iteración, se procedía a realizar una verificación con la dueña.

Cabe mencionar que, a lo largo del desarrollo del sistema, también, se tuvieron que incorporar conocimientos específicos de contabilidad y por tal motivo, fueron necesarias las reuniones entre la dueña de la empresa y su contadora, para explicar en qué consistiría el desarrollo del sistema, qué problema resolvería y en caso de que surgiese alguna duda de naturaleza contable, la misma sería consultada con la especialista en el tema.

1.4 Resultados esperados

Al finalizar el desarrollo del presente proyecto, se esperaba obtener un sistema estable y listo para reemplazar la utilización de planillas de Excel. Obtener información para la toma de decisiones por parte de los dueños de la empresa y generar una correcta visualización de las estadísticas de su negocio. Por otra parte, también se esperaba que el módulo desarrollado para llevar el control de transportes sea el esperado y sirviera de soporte para realizar un correcto cálculo de los adicionales de los choferes de la empresa. Además, de obtener un

conocimiento especializado acerca de los procesos contables, comerciales y administrativos que Servicios Agropecuarios Amanecer S.R.L posee; y tener una comunicación fluida con sus dueños para realizar modificaciones o nuevos proyectos en un futuro.

Por otra parte, a partir del conocimiento adquirido a lo largo de la formación académica brindada por la FI se ha tratado de incorporar nuevas tecnologías, herramientas y procesos de trabajo que son utilizados por diversas empresas del medio, en este caso de la rama agropecuaria. Varias de las tecnologías y metodologías fueron mencionadas alguna vez en diferentes materias de la carrera, pero es un mundo amplio y no se llega a cubrir todo el espectro de variedades, por lo que la elección de qué herramienta, proceso, o tecnología a utilizar, queda a cargo del desarrollador, en base al problema que se encuentra resolviendo.

1.5 Organización del documento

El presente documento se encuentra dividido en 4 capítulos. En el primer capítulo, llamado "Introducción", se detalla la información propia al problema en cuestión; los objetivos esperados, tanto por la empresa, como académicos; la metodología utilizada para llevar a cabo el presente trabajo, como así también, los resultados que se esperan obtener una vez finalizado el mismo.

En el Capítulo 2 "Tecnologías utilizadas" se describe en detalle todo el set de herramientas que serán utilizadas para el desarrollo, versionado y despliegue del proyecto en producción.

En el Capítulo 3 "Metodología de trabajo" se explicará cual es la metodología seleccionada para llevar a cabo el proyecto.

En el Capítulo 4 "Desarrollo del sistema", se detallará en qué consiste la implementación de cada una de las iteraciones que se deben desarrollar a lo largo del proyecto. Se mostrarán diagramas realizados durante las etapas de diseño, los cuales describen cómo se implementará el sistema.

A modo de conclusión, se realizará una síntesis del presente trabajo.

Por último, se enunciará la bibliografía consultada para la realización del informe y se añadirá un conjunto de anexos que recopilan detalles de cómo secuenciar la incorporación del software utilizado.

Capítulo 2 – Tecnologías utilizadas

En el presente capítulo se realizará una explicación detallada de las tecnologías utilizadas en la etapa de desarrollo. Iniciando con la configuración del entorno para desarrollo [22]; dentro de las primeras tareas, se encuentra la instalación del lenguaje de programación Python [20], y de la herramienta para crear entornos virtuales o VirtualEnvs [25], posteriormente la instalación de los frameworks Django [4] y Bootstrap [1,23] (Para detalle de instalación y uso ver Anexo 1, 2 y 3). Además, se expondrán diversos plugins que fueron incluidos en diferentes módulos de la aplicación a fin de facilitar el procesamiento y visualización de información, tal es el caso de DataTables [3], Select2 [21], Datepicker [13], por mencionar algunos de los más utilizados. También, se hará mención del sistema de control de versiones GIT [7,8], el cual permitió una mejor organización del proyecto.

2.1 Consideraciones previas

En primer lugar, se mencionan las herramientas de hardware y software utilizadas para las tareas de desarrollo y despliegue del sistema:

- Notebook Asus k53e, I3 2.7 Ghz, 4 Gb de memoria Ram, disco ssd 240 gb.
- Monitor LG Flatron W2243T, 21.5 pulgadas Full HD.
- TV Samsung 40 pulgadas.
- Teclado Genius usb y mouse Logitech m317 inalámbrico.
- Windows 7 Ultimate 64 bits.
- Sublime Text 3.2.2 [24].
- Google Chrome Versión 81.0.4044.138.
- Firefox 75.0.
- Cmd.exe - Símbolo del sistema.
- Git bash v2.26.0.
- Python versión v3.8.1.
- Django 3.0.3.
- PostgreSQL v11.6 [17].

- MySQL 5.7.28 [16]
- StarUML v2.1.2.
- Servicio de Host PythonAnywhere [19].

En segundo lugar, es importante mencionar las herramientas de comunicación utilizadas, en el marco del aislamiento preventivo, social y obligatorio que decretó el gobierno nacional por la pandemia de covid-19; las mismas son:

- WhatsApp Messenger. Aquí se incluyen videollamadas, audios, chats e imágenes.
- Correos electrónicos Gmail y Hotmail.
- Zoom.
- Google Hangouts.
- Google Drive.
- Skype.

2.2 Entorno de desarrollo

2.2.1 Python

Python [20] es un lenguaje de programación multiparadigma que soporta programación orientada a objetos, programación imperativa y programación funcional. Además, presenta las características de ser interpretado, dinámico y multiplataforma.

Este lenguaje de programación, se considera de propósito general, además, de ser sumamente popular por su facilidad de aprendizaje. Dentro de sus principales características se destacan:

- El gran número de librerías que posee, los tipos de datos y funciones incorporadas dentro del propio lenguaje.
- La facilidad de instalación y creación de programas.
- El gran número de plataformas que lo soportan. (Unix, Windows, Mac, por mencionar los más conocidos).

- Código simple de entender, con la principal ventaja de que son necesarias menos líneas de código para escribir un algoritmo.
- Python corre sobre un intérprete, lo que significa que el código puede ser ejecutado inmediatamente después de ser escrito.
- Posee una sintaxis sencilla.
- Es gratuito.

Entre los principales desarrollos que se pueden llevar a cabo con Python se encuentran: la creación de aplicaciones web, manejo de bases de datos, creación de algoritmos para resolución de problemas matemáticos complejos, prototipado rápido de software.

2.2.2 Entornos virtuales o "Virtualenvs"

Los entornos virtuales son herramientas escritas en Python, y utilizadas para crear entornos de desarrollo aislados para proyectos, lo que posibilita la instalación de paquetes sin interferir con otros Virtualenvs [25] creados, ni tampoco con los paquetes de Python instalados en el sistema de manera global. El uso de Virtualenvs se justifica ya que Python se encuentra disponible en versiones 2.X y 3.X. por lo tanto, existen múltiples herramientas que están disponibles para una versión y para la otra no. En este punto, es donde se aprecia la utilización de entornos virtuales, ya que permiten trabajar en distintos proyectos con versiones diferentes de paquetes y Python. Tomando como ejemplo el presente proyecto de desarrollo, se creó un entorno virtual, que contiene todas las herramientas necesarias para la implementación del sistema. Por otra parte, y de forma previa, se crearon otros entornos virtuales que contienen proyectos particulares de desarrollo ajenos al presente trabajo. De esta manera se mantiene la independencia de paquetes y versiones entre distintos proyectos de software.

2.2.3 Django

Django [4] es un framework de desarrollo orientado a la web, de código abierto, gratuito y escrito en Python. Cabe mencionar que, un framework es un conjunto de componentes y herramientas que ayudan a desarrollar sistemas de manera ágil y sencilla. Entre las posibilidades que permite Django, se encuentran las de lograr un software:

- Completo: Django provee por defecto un gran número de herramientas que facilitan la tarea de desarrollo.
- Versátil: Django permite la creación de variedad de sistemas orientados a la web, desde redes sociales hasta sistemas de ventas online. Puede integrarse con variedad de frameworks del lado del cliente y puede devolver contenido en diferentes formatos como puede ser HTML, RSS feeds, JSON [15], entre otros. Internamente, puede trabajar con distintos motores de bases de datos y sistemas de plantillas.
- Seguro: provee soluciones para la gestión de usuarios, permisos y contraseñas de forma nativa, sin que el desarrollador deba implementar módulos para resolver estas tareas. Además, ofrece protección por defecto frente a vulnerabilidades como SQL-injection, scripts maliciosos, falsificación de certificados, etc.
- Escalable: el producto software logrado permite modularidad de cada parte de su arquitectura. Por lo tanto, cada pieza puede ser reemplazada o modificada sin repercutir en otras.
- Mantenible: el código es escrito respetando patrones de diseño, los cuales facilitan su mantenimiento y reutilización. Se utiliza el principio “Don’t repeat yourself” (DRY), para que no exista la duplicación innecesaria del código. Y también, se promueve la agrupación de las funcionalidades relacionadas a través de “Apps”, las cuales son reutilizables. Y a un nivel menor, agrupa código relacionado en módulos, siguiendo el patrón “Modelo Vista Controlador” (MVC). Aunque en el caso de este framework, a las vistas se las denomina templates y al controlador se lo denomina vista.
- Portable: Django al estar escrito en Python, se ejecuta en multiplicidad de plataformas y es soportado por diversidad de servicios de hosting.

2.2.3.1 Patrón de diseño en el cual está basado Django (MTV)

El patrón de diseño en que está basado Django se denomina *"Model Template View" o MTV* en el cual se destacan los siguientes módulos:

- *Plantillas o Templates*: en este módulo se da la interacción entre el usuario y el sistema. En este framework, la función la realiza el motor de plantillas y el cargador de plantillas, los cuales reciben y muestran información al usuario.
- *Vista o View*: en esta capa se encuentra la lógica de la aplicación. En Django, una vista es una función o un Clase que recibe y devuelve peticiones HTTP. Las vistas acceden a los datos para cumplir con las peticiones por medio de *modelos* y delegan el formateo y presentación de los datos a los templates.
- *Modelo o Model*: Los *modelos* son clases que definen la estructura de los datos de una aplicación Django. Proporcionan la funcionalidad de añadir, modificar y borrar registros en una base de datos integrada con el framework.

Además de los tres módulos mencionados previamente, Django cuenta con un *mapeador URL*, el cual se usa para redirigir las peticiones HTTP a una *View* específica que resuelva el requerimiento (request). Asimismo, el *mapeador* puede incluir patrones de cadenas o dígitos que son incluidos en la URL, y los envía a la vista como si se tratase de datos.

El funcionamiento del conjunto MVT junto con el *mapeador URL* se describe en la siguiente imagen:

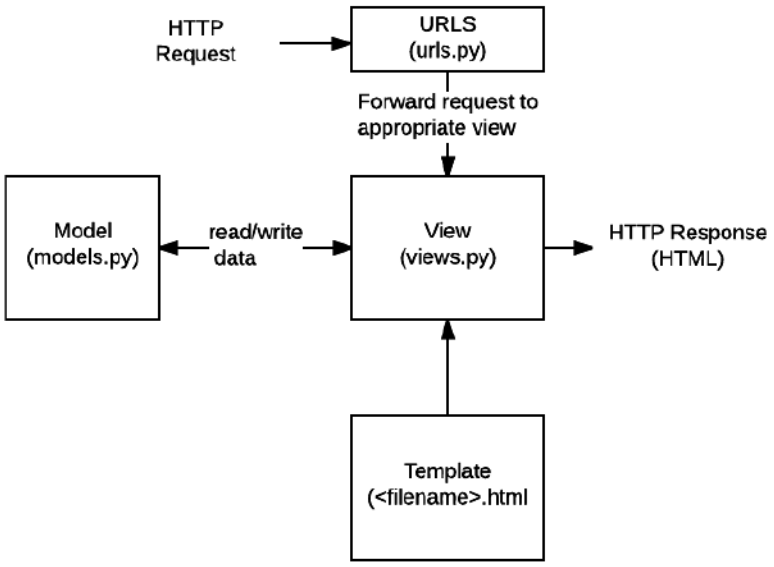

Ilustración 1: Mapeador URL y conjunto MTV. [6]

Esta imagen describe la interacción del *mapeador*, el cual recibe una *petición HTTP*; *la vista*, que recibe la petición proveniente del *mapeador* y contiene la lógica para realizar consultas al *modelo*, procesar los datos provenientes del *modelo* acorde a la petición que recibió y, presentarlos en un *template* para luego devolver una respuesta *HTTP Response* al navegador.

2.2.3.2 Mapeador URL

El *mapeador URL* se encuentra contenido en ficheros llamados urls.py localizados a lo largo de toda la estructura del proyecto. En este archivo se define una lista de mapeos entre URLS específicas y su correspondiente vista asociada a ellas. En caso de recibir una petición HTTP que tiene una URL que coincida con las cargadas en la lista, se pasa la petición a la vista asociada.

A continuación, se muestra una captura del archivo *urls.py* perteneciente a la App *"Transporte"* correspondiente al proyecto de desarrollo:

```python
from django.urls import path
from apps.transporte.views import  listarChoferes, nuevoChofer,generarListadoChoferes, editarChofer, \
                                   eliminarChofer,nuevoViaje,  listarViajes, editarViaje, eliminarViaje
from django.contrib.auth.decorators import login_required

app_name = "transporte"
urlpatterns = [
    path('nuevoChofer/',login_required(nuevoChofer.as_view()), name='nuevoChofer'),
    path('listarChofer/',login_required(listarChoferes.as_view()), name='listarChoferes'),
    path('editarChofer/<pk>',login_required(editarChofer.as_view()), name='editarChofer'),
    path('eliminarChofer/<pk>',login_required(eliminarChofer.as_view()), name='eliminarChofer'),
    path('pdf_listado_choferes/', login_required(generarListadoChoferes.as_view()), name='pdf_listado_choferes'),
    path('nuevoViaje/',login_required(nuevoViaje.as_view()), name='nuevoViaje'),
    path('listarViajes/',login_required(listarViajes.as_view()), name='listarViajes'),
    path('editarViaje/<pk>',login_required(editarViaje.as_view()), name='editarViaje'),
    path('eliminarViaje/<pk>',login_required(eliminarViaje.as_view()), name='eliminarViaje')
]
```

Ilustración 2: Archivo urls.py de App transporte.

En este archivo se pueden observar las importaciones realizadas desde el archivo vistas de la App de transporte, al igual que la inclusión de la función *path* y *login_required*. El primero, es útil para relacionar la URL cargada en la lista *urlpatterns* y el segundo, es la función a la que llamará. Además, se chequea si el usuario que está enviando la petición al *mapeador* se encuentra logeado en el sistema.

Luego, *app_name =”transporte”*, sirve para referenciar la App cuando se incluye la URL en la vista o en el template.

Dentro de la lista *urlpatterns*, se pueden observar numerosos ítems, los cuales corresponden a las vistas relacionadas con los distintas URLS. El primer argumento de la función *path*, es una cadena que define un patrón URL a relacionar. El segundo argumento, se trata de una vista a la cual se le asocia el primer argumento, y el tercer argumento de la función, corresponde al alias que tomará esa asociación entre ruta y vista, que sirve para identificar la URL de manera única y en caso de necesitar realizar cambios en la dirección, no tener que modificar nada más que la URL.

2.2.3.3 Vistas

En la vista se encuentra la lógica del proyecto, esta recibe y envía peticiones HTTP de los clientes web. También se encarga de gestionar el resto de los recursos con los que cuenta el framework, como consultas a bases de datos, manejo de plantillas y demás. Las vistas se

encuentran almacenadas en archivos llamados *views.py* localizados dentro del proyecto de software.

Dentro de un archivo *views.py* podemos encontrar funciones y clases, estas últimas reciben el nombre de vistas basadas en clases o "CBV", más adelante se profundizará sobre este tipo.

En cuanto a las funciones, estas reciben como argumento un objeto *HttpRequest*, en donde se encuentra la información correspondiente a la solicitud que se realiza, por ejemplo, si se trata de un *POST* o un *GET*, y devuelve un objeto *HttpResponse* con el template de la página solicitada.

El siguiente código sirve de ejemplo y tiene la finalidad de expresar lo expuesto previamente. En el desarrollo del sistema, las implementaciones de las vistas se realizaron a través de vistas basadas en clases.

```
def index(request):
    return render(request,'cuentaCorrienteCliente/index.html')
```

Ilustración 3: Función básica.

Cabe mencionar que *render* es una función que proporciona Django para devolver un objeto *HttpResponse* como primer argumento, como segundo argumento una plantilla, la cual en este caso es *'cuentaCorrienteCliente/index.html'* y como tercer argumento el *contexto*, el cual representa todo el set de información dinámica que se visualizará en el browser.

Como se mencionó previamente, durante el desarrollo del proyecto, se optó por trabajar con vistas basadas en clases. Este tipo particular de vistas son clases genéricas definidas dentro de Django que facilitan la labor de escribir código que se sabe que será reiterativo. Siguiendo con el ejemplo anterior, lo que devuelve la función *index*, no es más que una página estática, la cual se obtiene a partir de la plantilla enviada. La manera de realizar esto, pero con CBV es la siguiente:

```python
from django.views.generic import TemplateView

class Index(TemplateView):
    template_name = "index.html"
```

Ilustración 4: Class Based View básica (CBV).

A simple vista, parece no ser demasiado útil reemplazar una función por una clase para obtener una página estática, sin embargo, el verdadero potencial de CBVs radica en la posibilidad de realizar *ABMs* de manera casi inmediata, centrándose en qué se quiere lograr y no en cómo. Es en este punto donde el verdadero potencial de un framework de desarrollo se destaca.

En cuanto al tipo de CBVs, cumplen diferentes funciones, a continuación, se exponen algunas de las utilizadas en el proyecto:

- *CreateView*, este tipo de CBV cumple la función de mostrar un formulario para crear un objeto, volver a mostrar el formulario con los errores de validación en caso de que los haya y de guardar el objeto.
  Un ejemplo de su utilización puede verse en la siguiente imagen:

```python
class nuevoCliente(PermissionRequiredMixin,CreateView):
    permission_required = 'cuentaCorrienteCliente.add_cliente'
    model= Cliente
    form_class = ClienteForm
    template_name= 'cuentaCorrienteCliente/cliente_form.html'
    success_url= reverse_lazy('cuentaCorrienteCliente:listarCliente')
```

Ilustración 5: CBV CreateView de Cliente.

Esta clase es la encargada de la creación de un nuevo Cliente en el sistema, la clase hereda de *"PermissionRequiredMixin"* el cual se usa para gestión de permisos de usuarios y se explicará con mayor detalle posteriormente; y de *CreateView* que

determina la funcionalidad de la clase. *Permission_required* también es utilizado para permisos de usuarios.

Los atributos importantes en este momento son: *model*, el cual indica el *modelo* con el que la vista interactuará realizando consultas de inserción a la base de datos; *form_class*, el cual indica el formulario que está enlazado con el *modelo*; *template_name* corresponde a la plantilla que contendrá el *form_class* y *success_url*, indica la dirección URL que debe ser redirigida en caso de que la carga de un nuevo Cliente resulte exitosa.

- *FormView*, esta vista muestra un formulario. En caso de un error, muestra nuevamente el formulario, pero con los errores de validación hallados y en caso satisfactorio redirige a una nueva dirección. En este punto, puede parecer que *CreateView* y *FormView* son lo mismo, puesto que, realizan la misma función en cuanto a formularios se trata, pero la diferencia radica en que *CreateView* es adecuado para crear objetos mientras que *FormView* es adecuado para trabajar con formularios sencillos. La utilización de uno u otro depende del contexto donde se necesiten aplicar.

- *UpdateView*, esta vista muestra un formulario para editar un objeto ya existente, vuelve a mostrar el formulario con errores de validación en caso de que los haya y guarda, los cambios del objeto modificado. Un ejemplo de su utilización puede apreciarse en la siguiente figura:

```
class editarCliente(PermissionRequiredMixin,UpdateView):
  permission_required = 'cuentaCorrienteCliente.change_cliente'
  model=Cliente
  form_class=ClienteForm
  template_name='cuentaCorrienteCliente/cliente_form.html'
  success_url= reverse_lazy('cuentaCorrienteCliente:listarCliente')
```

Ilustración 6: CBV UpdateView de Cliente.

Aquí, la clase de la que hereda "*editarCliente*", corresponde a la especificación de que se trata de una CBV de tipo *UpdateView*, los atributos que se utilizan son los mismos que *"nuevoCliente"*, pero la diferencia está en el *mapeador URL*, donde la dirección asociada a esta vista contiene el id del cliente que se desea editar.

- *DeleteView*, esta vista muestra una página de confirmación y elimina un objeto existente. El objeto sólo será eliminado si se trata de un request *POST*. En caso de que el request sea *GET*, la vista mostrará una página de confirmación que debe contener un formulario que realice un POST hacia la misma dirección URL. Un ejemplo de su utilización puede apreciarse a continuación:

```
class eliminarMovimientoCliente(PermissionRequiredMixin,DeleteView):
    permission_required = 'cuentaCorrienteCliente.delete_movimiento_cliente'
    model= Movimiento_cliente
    template_name= 'cuentaCorrienteCliente/movimiento_cliente_borrar.html'
    success_url=reverse_lazy('cuentaCorrienteCliente:listarMovimientoCliente')
```

Ilustración 7: CBV DeleteView de Cliente.

El funcionamiento de este código es similar al de *UpdateView*, solo que, en vez de editar, elimina, y para eso debe recibir el id del objeto que se desea borrar, el cual se obtiene a través de la dirección URL pasada por parámetro desde el *mapeador URL*.

- *TemplateView*, esta vista solamente renderiza un template correspondiente a lo solicitado en el navegador junto con el contexto que contiene los parámetros enviados en la URL.

- *View*, todas las vistas mencionadas anteriormente, heredan desde esta clase padre. No es una vista genérica por lo que puede importarse y utilizarse en caso de ser necesario.

En este punto es importante hacer una aclaración respecto a las vistas utilizadas en el proyecto, ya que, en varios puntos de este, fue necesaria la redefinición de métodos de ciertas CBVs para poder cumplir con los requisitos del proyecto. Tal es el caso de la vista para listar

clientes ya registrados en el sistema, como es de esperar, se podría haber utilizado una vista de tipo *ListView* la cual solamente retorna una lista con clientes que se quieren mostrar, sin embargo, en el template correspondiente al listado de clientes, también, fue necesario incorporar un formulario para generar listados de los clientes registrados, se podría esperar el uso de *FormView*. Con lo cual, se tiene, por un lado, una funcionalidad de listar, y por el otro, la funcionalidad de enviar información para luego consultar en la base de datos y finalmente reflejarla en el listado. Este requisito fue resuelto a través de la redefinición de métodos de clase correspondiente a *FormView*, en donde, además de mostrar un formulario, se envía el contexto cargado con los objetos correspondientes a los clientes.

El siguiente código refleja lo expresado anteriormente:

```
class listarMovimientoCliente(PermissionRequiredMixin,FormView):
    permission_required = 'cuentaCorrienteCliente.view_movimiento_cliente'
    template_name = 'cuentaCorrienteCliente/movimiento_cliente_listar.html'
    form_class = FormularioFechasInforme

    def get_context_data(self, **kwargs):
        context = super(listarMovimientoCliente, self).get_context_data(**kwargs)
        context['object_list'] = self.get_queryset()
        return context

    def get_queryset(self):
        self.clientes = self.kwargs.pop(" clientes","")
        return Movimiento_cliente.objects.all()

    def form_valid(self, form):
      datos = form.cleaned_data
      fecha_inicio=datos['fecha_inicio']
      fecha_fin=datos['fecha_fin']
      fechas={
      'fecha_inicio':fecha_inicio,
      'fecha_fin':fecha_fin
      }
      datos_empresa= ControlPanel.objects.all().values()
      data= Movimiento_cliente.objects.filter(fecha_comprobante__range=[str(fecha_inicio), str(fecha_fin)])
            .order_by('fecha_comprobante').order_by('fecha_comprobante')
      total= Movimiento_cliente.objects.all()
            .filter(fecha_comprobante__range=[str(fecha_inicio), str(fecha_fin)]).
            .aggregate(Sum('monto_neto'),Sum('monto_exento'),Sum('iva_ins'),Sum('monto_total'))
      pdf= render_to_pdf('cuentaCorrienteCliente/pdf_movimientos_clientes.html',
                         {'datos_empresa':datos_empresa,'fechas':fechas,'data':data,'total':total})
      return HttpResponse(pdf,content_type='application/pdf')
```

Ilustración 8: CBV FormView modificado de Cliente.

Los métodos redefinidos fueron *get_context_data()*, *get_queryset()* y *form_valid()*. En primer lugar, la funcionalidad de la vista no se modificó, por lo que sigue mostrando un formulario, en este caso *FormularioFechasInforme*, pero la redefinición de los métodos, resulta necesaria para poder enviar, además, del formulario mencionado, un contexto donde

se incluye una lista de objetos correspondientes a los clientes registrados en el sistema. El primer método, es el encargado de devolver ese contexto y para ello debe ser cargado haciendo uso de *get_queryset()*, el cual es el encargado de realizar la consulta a la base de datos, la cual devuelve los clientes. Finalmente el tercer método *form_valid()*, controla que la información enviada por POST en el *form* mostrado sea correcta y además realiza la consulta para confección del informe con los datos de los clientes.

2.2.3.4 Modelos

Las aplicaciones implementadas en Django manejan y consultan datos a través de clases llamadas *modelos*. Estas definen una estructura de datos almacenados, donde se incluyen los tipos de campos, su tamaño, valores por defecto, listas de opciones, textos de ayuda, entre otros. La definición de los *modelos* es independiente de la base de datos enlazada al proyecto de desarrollo, pudiendo seleccionar cualquiera de las bases de datos soportadas por el framework realizando las configuraciones correspondientes. Una vez seleccionada y configurada la base de datos, no es necesario interactuar directamente con esta.

Un *modelo* corresponde a una única tabla de la base de datos, y sus atributos se asignan a las columnas de la tabla. Una instancia de un *modelo*, corresponde a una fila de la tabla en la base de datos. Los *modelos*, heredan de *django.db.models.Model* la cual proporciona funcionalidades para agregar, modificar y borrar registros de la base de datos.

Una de las partes más importantes de los *modelos* son los campos o *fields*, los cuales definen los campos de las tablas en la base de datos. Cada field en el *modelo* debe ser una instancia de la clase de campo que se desea. Django utiliza los tipos de clases de campo para determinar:

- El tipo de columna, la cual determina qué tipo de dato almacenar en la base de datos (Integer, Varchar, Date, Etc.).
- El widget HTML predeterminado que se utiliza para representar un campo de un formulario.

- Requisitos de validación tanto en el administrador de Django como en formularios generados dinámicamente.

Además, cada field puede presentar distintas opciones en su declaración, algunos de los más utilizados en el desarrollo del presente proyecto fueron:

- *Null*: si se encuentra seteado en TRUE, Django entiende que se pueden guardar valores nulos en la base de datos. Por defecto, es FALSE.
- *Blank*: si se encuentra seteado en TRUE, se permite a guardar el campo vacío. Por defecto se setea en FALSE.
- *Unique*: si se encuentra seteado en TRUE, el campo debe ser único en toda la tabla. No confundir con clave primaria.
- *Max_length*: indica la longitud máxima que puede tener un campo.
- *Max_digits*: indican la cantidad máxima de dígitos que puede tener un número.
- *Decimal_places*: indican la cantidad de decimales que puede tener un número.
- *Error_messages*: despliegan un mensaje de error en caso de que el campo donde se define esta opción no pase la validación, dicho mensaje se muestra en el template.
- *Validators*: es un invocable que evalúa ciertos criterios de validación y si el campo no los cumple, arroja un ValidationError.

En cuanto a las bases de datos relacionales, una de las características más importantes que poseen es la de relacionar tablas entre sí. Django por su parte, ofrece formas de definir los tres tipos más comunes de relaciones en bases de datos:

- Muchos a uno, la cual se define a través de *ForeignKey*, la misma se utiliza como si se tratase de otro tipo de campo, incluyéndolo como un atributo en la clase *modelo*. La *ForeignKey* requiere de un argumento posicional que indica la clase con la que está relacionado el *modelo*.
- Muchos a muchos, esta relación utiliza el campo *ManyToManyField* y funciona de la misma forma que ForeignKey.
- Uno a uno, esta relación se utiliza a través del campo *OneToOneField* y funciona de la misma manera que las dos mencionadas anteriormente.

El siguiente *modelo* expone todos los temas explicados anteriormente:

```
class Viaje(models.Model):
    chofer = models.ForeignKey(Chofer, on_delete=models.PROTECT)
    fecha_viaje=models.DateField()
    procedencia= models.CharField(max_length=50)
    destino = models.CharField(max_length=50)
    detalle=  models.CharField(max_length=50)
    duenio_mercaderia = models.ForeignKey(Cliente, on_delete=models.PROTECT)
    kg_descargados=models.DecimalField(max_digits=10, decimal_places=4, blank=False, validators=[MinValueValidator(0.01)] )
    tarifa= models.DecimalField(max_digits=10, decimal_places=2, validators=[MinValueValidator(0.01)])
    total_viaje=models.DecimalField(max_digits=20, decimal_places=4)
    porcentaje_a_pagar=models.DecimalField(max_digits=10, decimal_places=2,validators=[MinValueValidator(0.01)])
    anticipo=models.DecimalField(max_digits=10, decimal_places=2,validators=[MinValueValidator(0)])
    pendiente_pago=models.DecimalField(max_digits=20, decimal_places=4)

    def __str__(self):
        return '{}'.format(self.chofer)
```

Ilustración 9: Modelo correspondiente a Viaje.

Una vez creados los *modelos*, Django proporciona una *API* que permite crear, recuperar, editar y borrar objetos. Dicha API se utiliza en las vistas para realizar acciones sobre la base de datos sin la necesidad de ejecutar consultas sobre esta. Por ejemplo, la línea de código *Viaje.objects.**all(),*** retorna todos las filas de la tabla viaje, las cuales en Django se trataría de un objeto por cada fila de la base de datos. De esta manera, las consultas a la base de datos son mucho más sencillas, rápidas de escribir y ejecutar.

Otro ejemplo más complejo, es un fragmento de código utilizado para calcular el total de los movimientos de un cliente en un informe solicitado de acuerdo a un rango de fechas:

```
total= Movimiento_cliente.objects.all()
       .filter(fecha_comprobante__range=[str(fecha_inicio), str(fecha_fin)]).
       .aggregate(Sum('monto_neto'),Sum('monto_exento'),Sum('iva_ins'),Sum('monto_total'))
```

Ilustración 10: Consulta para obtener totales.

La lógica detrás de esta línea es la de traer todos los objetos de *Movimiento_cliente*, filtrarlos por un rango de fechas que se ingresan por formulario, y realizar la suma de los campos monto_neto, monto_exento, iva_ins y monto_total. Cabe mencionar que, la función *aggregate* devuelve un diccionario en Python con pares de nombre y valor. Este se genera a partir del nombre del campo.

2.2.3.5 Templates o Plantillas

Como definición general, un *Template* es un archivo que se puede utilizar para representar información diferente de manera consistente. El formato de una plantilla en Django se escribe utilizando HTML. Sin entrar en demasiados detalles sobre este lenguaje de marcado, se destaca que una plantilla deberá tener las etiquetas <head> y <body> y otras, para dar formato al documento.

Las plantillas en Django son archivos utilizados por las vistas para mostrar información proveniente de los *modelos*. Los templates se almacenan en un directorio específico del árbol de documentos del proyecto, y las vistas saben dónde encontrarlas.

Una plantilla se renderiza junto con un contexto, el renderizado reemplaza las variables que son devueltas por la vista y en su lugar, se muestran los valores que estas toman. Para ello, Django provee un lenguaje de plantillas. Este lenguaje, otorga cierta lógica a la hora de representar información en un template ya que permite condiciones, bucles, formateo de información y demás.

Cuando el template recibe el contexto desde la vista, debe cambiar la variable por su valor asociado, para esto se utilizan las doble llaves *{{nombre_variable}}*. Si, por ejemplo, recibe una lista de objetos, esta debe recorrerse utilizando un bucle, lo cual se representa de la siguiente manera:

```
{%for config in object_list%}
    <td>{{config.razon_social}}</td>
    <td class="text-center" id="id_cuit">{{config.cuit}}</td>
    <td class="text-center">
    <div class="btn-group">
    <a class="btn btn-primary " href="{% url 'gestion:editarConfig' config.id %}">
        <i class="fas fa-edit"></i>
        Editar
    </a>
    </div>
    </td>
{%endfor%}
```

Ilustración 11: Recorriendo lista de objeto en template.

El código anterior, corresponde a la visualización de información básica de la empresa en una tabla, se utiliza un bucle *for* para recorrer una lista de objetos que es parte del contexto enviado desde la vista. Para acceder a los atributos de cada objeto se utiliza el punto y nombre del atributo.

Otra ventaja que provee el framework es que permite la extensión de plantillas, esto significa que se pueden reutilizar fragmentos de código HTML para diferentes páginas del sitio web que se está desarrollando, lo cual es favorable a la hora de realizar modificaciones, ya que, al no haber repetición de código, será necesario realizar un solo cambio.

Cabe aclarar que, los siguientes ejemplos, son a modo explicativo sobre las características de templates, su utilización dentro del proyecto de desarrollo fue bastante más compleja y con una cantidad muy amplia de propiedades.

Lo primero que se realiza es la creación de una plantilla base que será el padre de todas las otras plantillas a utilizar. En esta, se definirá la estructura básica del sitio y se incluirán etiquetas para crear bloques de contenido como se muestra a continuación:

```
<body>
    <div class="page-header">
        <h1>Template Base</h1>
    </div>
    <div class="container">
        <div class="row">
            <div class="col-md-12">
            {% block content %}
            {% endblock %}
            </div>
        </div>
    </div>
</body>
```

Ilustración 12: Estructura básica template base.

Luego, se debe crear el resto de las plantillas que heredarán de la plantilla base, las cuales redefinirán el bloque declarado en el template padre:

```
{% extends 'base.html' %}
{% block content %}
    {%for config in object_list%}
        <td>{{config.razon_social}}</td>
    <td class="text-center" id="id_cuit">{{config.cuit}}</td>
        <td class="text-center">
        <div class="btn-group">
        <a class="btn btn-primary " href="{% url 'gestion:editarConfig' config.id %}">
            <i class="fas fa-edit"></i>
        Editar</a>
        </div>
    </td>
    {%endfor%}
{% endblock %}
```

Ilustración 13: Plantilla que hereda de template base.

De esta forma se evita la reescritura de código, lo cual facilita su legibilidad y permite su reutilización.

2.2.3.6 Otros elementos de Django

En la sección anterior se hizo una introducción sobre las principales características que presenta este framework y que serán las más utilizadas durante el proceso de desarrollo de un proyecto. Sin embargo, Django proporciona una variedad más amplia de elementos para integrar al proyecto. Algunos de ellos son:

- *Formularios*: como ya bien se sabe, se utilizan para recolectar datos del usuario y ser procesados en el servidor. Django simplifica su creación, validación y procesamiento al proporcionar un marco de trabajo que permite definir formularios y campos a través de programación, y luego utilizar estos objetos para generar el código HTML del formulario y manejar su validación e interacción con el usuario.
  Los formularios creados se encuentran en archivos llamados *forms.py* dentro de cada App creada en el proyecto. Un ejemplo de un formulario es:

```
class Tipo_Comprobante_Form(forms.ModelForm):
            class Meta:
                model=Tipo_Comprobante
                fields= [
                        'descripcion',
                ]
                labels ={
                        'descripcion' : 'descripcion',
                }
                widgets = {
                        'descripcion': forms.TextInput(attrs={'class':'form-control'}),
                }
```

Ilustración 14: Formulario Tipo_Comprobante_Form incluido en archivo forms.py.

En este caso, la clase *Tipo_Comprobante_Form* hereda de *forms.ModelForm*, la cual indica que el formulario que se crea está relacionado a un *modelo* previamente creado. La clase *Meta*, indica los campos que se crearán y que se relacionarán con el *modelo*, es por ello que se define el *modelo* a través del atributo *model*, también se definen los campos o *fields* a través de una lista, las etiquetas o *labels* que tendrán esos campos a través de un diccionario y finalmente los widgets, lo cuales establecen el tipo de etiqueta HTML para cada campo.

- *Administrador de Django*: el sitio de administración de este framework está incluido por defecto desde el momento en que se instala Django. La principal función de este sitio es gestionar *modelos* creados, dar de alta usuarios al sistema y proporcionar permisos.

- *Autenticación y permisos de usuarios*: Django proporciona un sistema de autenticación y permisos que permite verificar credenciales de usuarios y definir qué acciones pueden realizar sobre el sistema. El framework incluye *modelos* para gestión de usuarios y grupos, estos últimos, son una forma genérica de aplicar permisos a más de un usuario a la vez.

  La forma de otorgar permisos a un usuario es a través del administrador de Django, y lo recomendable es realizarlo a través de la creación de grupos de permisos que luego son otorgados a un usuario en particular. Además, los permisos que posee un

usuario pueden ser utilizados dentro del código generado en las vistas, *mapeador URL*, y también, en los templates que utilizan las vistas, la razón de esto es la de restringir los accesos a lugares del sitio a las que el usuario no debe acceder.

Tomando como ejemplo el fragmento de código presentado en el *mapeador URL*, ver Ilustración 2: Archivo urls.py de App transporte, particularmente la línea:

```
path('nuevoChofer/',login_required(nuevoChofer.as_view()), name='nuevoChofer')
```

Puede verse que el segundo parámetro de la función *path*, hace uso de un método llamado *login_required()*, el cual chequea si el usuario está logeado en el sistema o no. Si efectivamente se encuentra logueado, llamará a la vista solicitada, caso contrario redirigirá a la página de login para iniciar sesión.

El control de permisos en las vistas puede realizarse de diferentes formas dependiendo de si se trata de una vista basada en función o una vista basada en clase CBV. En el presente informe, se explicará la utilización básica de permisos aplicados a CBV. La forma de trabajar con permisos sobre las vistas definidas, se realiza a través de Mixins. Estos son clases que agregan comportamiento a otra clase a través de la herencia de esta última. En este caso, permite el control de permisos por parte del usuario logueado al sistema y que quiere hacer uso de una vista en particular.

El siguiente ejemplo, es el mismo que se utilizó en la explicación de vistas, ver 2.1.2.4 Vistas, pero sirve traerlo nuevamente a la explicación:

```
class nuevoCliente(PermissionRequiredMixin,CreateView):
    permission_required = 'cuentaCorrienteCliente.add_cliente'
    model= Cliente
    form_class = ClienteForm
    template_name= 'cuentaCorrienteCliente/cliente_form.html'
    success_url= reverse_lazy('cuentaCorrienteCliente:listarCliente')
```

Ilustración 15: Uso de permisos en CBV.

Como se mencionó previamente, la función de esta vista es la de crear un nuevo cliente en el sistema, pero la particularidad es que no cualquier usuario puede realizar esta función, sino los usuarios que posean ese permiso, el cual fue otorgado a través del administrador de Django, incluyendo el permiso en un grupo al cual pertenece o no el usuario registrado. Ahora bien, la forma de corroborar los permisos, se realiza a través de *permission_required*, el cual evalúa si el usuario se encuentra autorizado para crear o no un cliente. Es importante mencionar que, la clase *nuevoCliente* hereda de la clase *PermissionRequiredMixin*, y provee todas las funcionalidades para manejar este tipo de situaciones. De esta manera, es como se tratan los permisos de usuario en las vistas.

Por otro lado, los permisos dentro de los templates se realizan chequeando si el usuario logueado puede o no realizar cual o tal acción, si está autorizado se muestra el acceso, gráfico o elemento correspondiente; en caso de no estar autorizado, no se muestra nada o se muestra otro elemento acorde al perfil de este usuario.

El siguiente ejemplo corresponde a los expresado anteriormente, en cuanto a manejo de permisos en templates:

```
{%for elemento in object_list%}
        <tr>
            <td>{{elemento.chofer}}</td>
            <td class="text-center">{{elemento.fecha_viaje|date:'d/m/Y'}}</td>
            <td>{{elemento.procedencia}}</td>
            <td>{{elemento.destino}}</td>
            <td class="text-right">{{elemento.detalle}}</td>
            <td class="text-right">$ {{elemento.pendiente_pago}}</td>
            {% if 'transporte' in perms %}
                {% if 'transporte.change_viaje' in perms and  'transporte.delete_viaje' in      perms %}
                <td class="text-center">
                    <div class="btn-group">
                        <a class="btn btn-primary " href="{% url 'transporte:editarViaje' elemento.id %}">
                            <i class="fas fa-edit"></i>Editar
                    </a>
                    </div>
                    <div class="btn-group">
                        <a class="btn btn-danger " href="{% url 'transporte:eliminarViaje' elemento.id %}">
                            <i class="fas fa-trash"></i>Eliminar
                    </a>
                    </div>
                </td>
            {%else%}
            {% if 'transporte.delete_viaje' in perms %}
                    <td class="text-center">
                    <div class="btn-group">
                            <a class="btn btn-danger " href="{% url 'transporte:eliminarViaje' elemento.id %}">
                                <i class="fas fa-trash"></i> Eliminar
                            </a>
                    </div>
                    </td>
            {%else%}
            {% if 'transporte.change_viaje' in perms %}
                    <td class="text-center">
                        <div class="btn-group">
                                <a class="btn btn-primary" href="{% url 'transporte:editarViaje' elemento.id %}">
                                    <i class="fas fa-edit"> </i> Editar
                                </a>
                        </div>
                    </td>
                {%endif%}
                {%endif%}
        {%endif%}
        {%endif%}
    </tr>
   {%endfor%}
```

Ilustración 16: Uso de permisos en templates.

Este código corresponde a un fragmento del template para listar viajes, en este caso y dependiendo de los permisos que tenga el usuario, este podrá editar y borrar, editar solamente, borrar solamente, o no realizar ninguna acción sobre los registros de la lista.

Para cerrar esta sección, cabe aclarar que, el manejo de permisos se da a través de las distintas interacciones entre los elementos que conforman el framework, donde el encargado de gestionar los permisos es el administrador de Django. Estos son consultados en el *mapeador URL*, las vistas y los templates para conformar la página que es devuelta al navegador.

2.2.3.7 Estructura del proyecto

A continuación, se puede observar la estructura del proyecto de desarrollo actual:

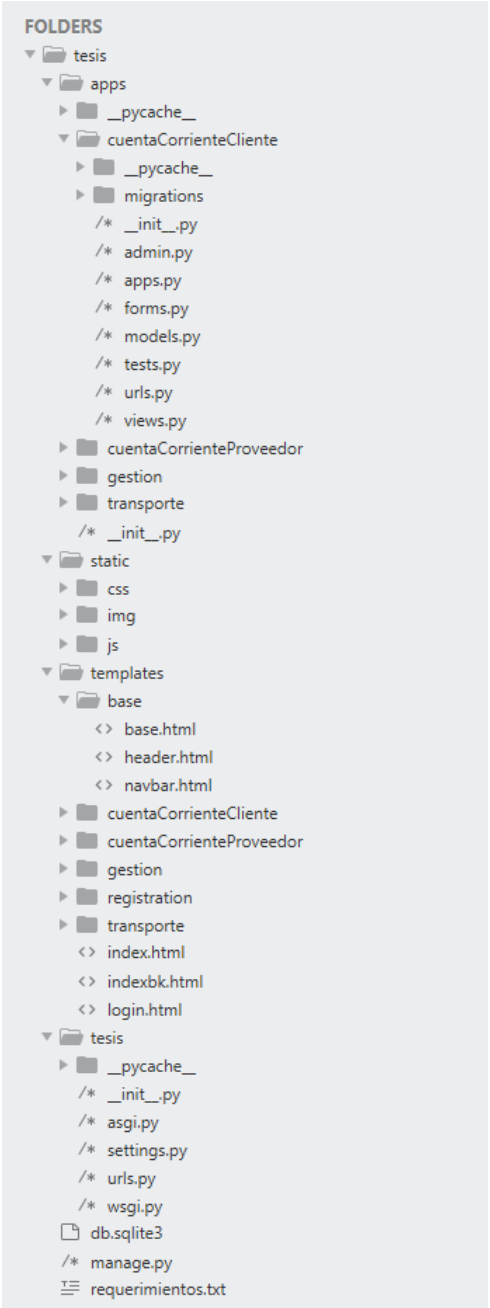

Ilustración 17: Estructura directorios de proyecto en Django.

En esta imagen, se puede ver la estructura de directorios, subdirectorios y archivos que conforman el proyecto de desarrollo abordado en el presente informe. Los directorios a destacar en esta sección son:

- *Apps*: aquí se alojan las Apps creadas que conforman el proyecto. Cada App contiene su propio archivo de vista, *modelo*, *mapeador URL*, formularios, entre otros. Cabe destacar que, las Apps son paquetes de Python que cumplen una determinada función, y tiene una estructura predefinida que facilita la codificación.

  En este punto, es necesario explicar que un proyecto de Django es una colección de configuraciones y Apps para un sitio web en particular. Un proyecto puede contener múltiples aplicaciones.

  En este caso se crearon 4 aplicaciones que responden a lógicas distintas, *cuentaCorrienteCliente*, se encarga de la gestión de clientes y sus movimientos; *cuentaCorrienteProveedor*, se ocupa de proveedores y sus movimientos. La App *gestión*, se encarga de las configuraciones pertinentes al sitio en cuestión y; *transporte* se encarga de la gestión de choferes y los viajes realizados por estos.
- *Static*: aquí se encuentran los archivos correspondientes a hojas de estilos, imágenes y scripts; es decir, archivos estáticos.
- *Templates*: en este directorio se encuentra toda la estructura de archivos correspondientes a los templates mencionados con anterioridad. Se organiza de tal manera, de que cada directorio se identifique con el nombre de la App que contiene la vista que los llama.
- *Tesis*: es el nombre que se le otorgó al proyecto al momento de su creación, allí se encuentran varios archivos de configuración, donde el más importante es *settings.py*, este archivo contiene todas las configuraciones en cuanto a bases de datos se refiere, manejo de rutas, declaración de Apps, configuraciones de idioma y hora, passwords y demás.

2.2.4 Bootstrap

Bootstrap es un *framework front-end* de código abierto para desarrollo con HTML, CSS y JS. Permite implementar rápidamente prototipos y aplicaciones web. Se caracteriza por aportar adaptabilidad a los sitios que lo utilizan. Además, es soportado por las últimas versiones de los navegadores modernos.

La principal característica de este framework es que contiene todos los elementos necesarios para crear páginas que sean adaptativas y agradables para el usuario. Entre sus componentes más importantes se destaca, el sistema de grilla o grid, que aporta la funcionalidad para implementar sitios web adaptables. La misma funciona utilizando contenedores, filas y columnas para el diseño y la alineación del contenido.

Los contenedores proporcionan un espacio para centrar y rellenar el contenido de forma horizontal. Para ello se utilizan las clases *.container*. Las filas se incluyen dentro de los contenedores y se encargan de establecer el espacio para las columnas que contendrán. Las filas se definen utilizando la clase *.row*. En Bootstrap, todo el contenido debe incluirse dentro de columnas, y las filas solo deben contener columnas. Se pueden incluir hasta un máximo de doce columnas por fila y se utilizan a través de la clase *.col*.

Además de su sistema de grillas, Bootstrap provee una amplia serie de componentes para facilitar el desarrollo del sitio. Algunos de los componentes utilizados dentro del presente proyecto son:

- *Alerts*: como su nombre lo indica, las alertas son pequeños fragmentos de código que pueden incluirse en el proyecto para mostrar un error, una confirmación exitosa, un mensaje de alerta, entre otras.
- *Buttons*: los botones personalizados con los estilos de Bootstrap se utilizan para acciones en formularios, cuadros de diálogo, dentro de tablas.
- *Cards*: este elemento se trata de un contenedor que proporciona adaptabilidad al contenido que se coloque dentro. Incluye opciones para encabezados, cuerpos y pie del contenido. En el proyecto fueron utilizados para dar un estilo visual ordenado a los gráficos estadísticos y tablas.

- *Forms*: es tal vez uno de los componentes más importantes que provee este framework front end, el uso de formularios Bootstrap aporta un gran número de posibilidades de personalización y disposición de cada uno de los elementos que lo conforman. Además, provee la opción para visualizar de forma agradable, los errores que presente la validación de los campos que contiene.

2.2.4 Otros plugins utilizados

Se utilizaron otras tecnologías a fin de facilitar la carga y consulta de información, dichos plugins ofrecen facilidades en cuanto a su instalación. Entre los más utilizados en el desarrollo del proyecto se encuentran:

- *Datatables*: es un complemento de la librería *jQuery* [12]. Es una herramienta adaptable a las necesidades particulares de un proyecto, a la vez de agregar características avanzadas a una tabla HTML. El uso de este plugin, radica en que los usuarios pueden obtener información útil de una tabla de manera casi instantánea.

- *Select2*: es un plugin que utiliza la librería *jQuery*. Este se utiliza para aportar funcionalidades extras a los select, tales como filtros, scrolling infinito, soporte *Ajax*, entre otros. En el presente proyecto se utilizó para el filtrado tanto de clientes, proveedores y choferes a la hora de ingresar movimientos o viajes.

- *Datepicker*: es un plugin que utiliza *jQuery* y que aporta la funcionalidad de seleccionar fechas a través de una interface gráfica amigable con el usuario. Lo normal en cuanto a su utilización, es hacerlo a través de formularios, donde uno de sus campos contiene un *Datepicker*, en el instante en que el usuario realiza click sobre él, se abre un cuadro donde podrá seleccionar la fecha deseada. De esta manera, se evitan fechas inválidas o con errores en sus dígitos.

- *Mask* [14]: el uso de este plugin también depende de la librería *jQuery* y se utiliza para aportar caracteres que den formato a cadenas de texto ingresadas por el usuario,

tales como teléfonos, números de documentos, números de comprobantes, etc. Su uso dentro del proyecto se puede observar en formularios y listados, donde es preciso que el usuario visualice datos correctamente formateados.

- *Charts* [2]: este plugin permite crear diferentes tipos de gráficos interactivos de manera sencilla. Entre sus principales características se destacan las de ser adaptable a múltiples pantallas, utiliza el elemento *canvas* propio de HTML5, soportado por una amplia variedad de navegadores. También, todos los gráficos que se pueden crean, son animados y personalizables.

2.2.5 GIT

Git es un sistema de control de versiones gratis y de código abierto, diseñado para gestionar todo tipo de proyectos, con la particularidad de que es sencillo de utilizar. Su funcionalidad radica en llevar el registro de los cambios y coordinar grupos de personas desarrollando en el mismo proyecto.

Para utilizar git, este proporciona una serie de comandos básicos:

- *git init*: crea un subdirectorio llamado .git, el cual contiene todos los archivos necesarios del repositorio.
- *git fetch*: descarga los cambios realizados en el repositorio remoto.
- *git merge <nombre_rama>*: realiza cambios en la rama actual.
- *git pull:* realiza la función de fetch y merge
- *git commit –m <mensaje>:* confirma los cambios realizados. El mensaje es una breve descripción de la modificación hecha.
- *git push origin <nombre_rama>:* sube la rama "nombre_rama" al servidor remoto.
- *git status*: muestra el estado actual de la rama, con los cambios que hay sin "commitear".
- *git add <nombre_archivo>:* comienza el seguimiento del archivo.

En el actual proyecto se trabajó sobre la rama *Master*. La manera de trabajar fue realizando el desarrollo en local para luego subir los cambios al repositorio ubicado en GitHub, dicha acción se realizó a través de los comandos *git add . , "git commit –m <descripcion>" y git push –u origin master*. Al momento de subir a producción se utiliza el comando *git pull*.

Capítulo 3 – Metodología de trabajo

Para llevar a cabo este proyecto de desarrollo, se realizó un relevamiento sobre distintas metodologías de software que podían aplicarse al presente trabajo. En primer lugar, se analizaron las metodologías más tradicionales como cascada, el desarrollo evolutivo y desarrollo basado en componentes. En segundo lugar, se tomaron en cuenta, para su análisis, metodologías tales como la iterativa incremental o la metodología en espiral. Finalmente, se realizó un estudio sobre la factibilidad de utilizar metodologías ágiles tales como Scrum o Programación Extrema (XP).

La utilización de una metodología u otra se dio por la naturaleza del proyecto y del cliente, también influyó en la elección el hecho de que, al tratarse de un proyecto individual, las metodologías ágiles no serían las apropiadas ya que se necesitan de varios participantes en el proyecto. Procesos como cascada, evolutivo y desarrollo basado en componentes, tampoco fueron acordes a este caso, ya que representan dificultades a la hora de reestructurar el proyecto en caso de que algún requisito cambie. Además, en este proyecto fue sumamente importante obtener una entrega funcional de tipo prototipo desde etapas iniciales. Por estas razones, las primeras metodologías no fueron tomadas en consideración. No obstante, pueden ser utilizadas como complemento de otras metodologías tales como iterativa incremental.

Finalmente, la elección decantó por iterativa incremental ya que se consideró que sería la que mejor se ajustaría con el desarrollo del sistema expuesto en el actual documento. La idea principal fue, desarrollar un sistema por módulos, donde en cada módulo se realizó una captura de requisitos, análisis, diseño, implementación, prueba e instalación. También, fue necesario obtener desde el inicio del proyecto, un sistema con características de prototipo para compartir con el cliente y que éste solicite realizar los cambios y aclaraciones convenientes. De esta forma, se generaba retroalimentación a medida que el proceso de desarrollo avanzaba.

3.1 Metodología Iterativa Incremental

Para la descripción de esta metodología se tomó como referencia el Capítulo 5 “Un Proceso Iterativo e Incremental” del libro “El proceso unificado de desarrollo de software”. En ese capítulo se explica que la metodología iterativa e incremental en el desarrollo de software, divide al proyecto en partes más pequeñas. Cada una de estas partes es una iteración que resulta en un incremento. Las iteraciones, corresponden a los flujos de trabajo y los incrementos, determinan el crecimiento del producto. Las iteraciones se deben seleccionar y ejecutar de forma planificada, donde la elección que se implementará en cada una, se da a través del conjunto de casos de uso y también por los riesgos que representan para el proyecto. En las primeras iteraciones, se pueden observar los riesgos más importantes del proyecto.

En cada iteración, el desarrollador identifica y especifica los casos de uso relevantes, crea un diseño utilizando la arquitectura seleccionada como guía para implementar dichos casos de uso. Si la iteración cumple con los objetivos planteados, se pasa a la siguiente. Por el contrario, se debe revisar lo hecho hasta ese momento.

La metodología iterativa incremental proporciona la estrategia para desarrollar un producto software en pequeños pasos manejables:

- Planificar un poco.
- Especificar, diseñar e implementar un poco.
- Integrar, probar y ejecutar un poco en cada iteración.

Si se está conforme con un paso, se avanza al siguiente. En cada paso, se obtiene retroalimentación que permite ajustar los objetivos para el siguiente paso. Después, se avanza al siguiente paso, y después al siguiente y así sucesivamente. Una vez que se dieron todos los pasos planificados, se obtiene un producto desarrollado que se puede entregar al cliente.

Las iteraciones en las primeras fases tratan con la determinación del ámbito del proyecto, la eliminación de riesgos críticos y creación de la línea base de la arquitectura. Luego, a medida que avanza el proyecto, se reducen gradualmente los riesgos restantes y se implementa el resto de los componentes, la forma de las iteraciones cambia, resultando en incrementos. En

palabras más sencillas, se divide el proyecto en un número variable de mini proyectos, donde cada uno de ellos corresponde a una iteración. Cada iteración, está conformada de la misma forma que un proyecto de software: planificación, desarrollo (requisitos, análisis y diseño, implementación y prueba) y una preparación para la entrega. Por último, cabe resaltar que el control de cada iteración está a cargo del encargado del proyecto, es responsabilidad de éste conformar las primeras iteraciones de manera de mitigar riesgos conforme avance el proyecto.

3.1.1 Motivaciones para realizar un desarrollo iterativo – incremental

Entre las principales motivaciones para llevar a cabo un proyecto de desarrollo se destacan:

3.1.1.1 Atenuación de riesgos

En el desarrollo de software se puede definir un riesgo como un asunto que tiene cierto grado de probabilidad de poner en peligro el éxito de un proyecto. Entonces, atacar los riesgos al comienzo del proceso de desarrollo de software es el primer paso para obtener un proyecto exitoso.

En el desarrollo iterativo incremental, los riesgos importantes se identifican y se reducen en las primeras fases y conforme avanzan las iteraciones, los riesgos se van reduciendo. Sobre el final del proceso de desarrollo, los riesgos fueron reducidos casi por completo.

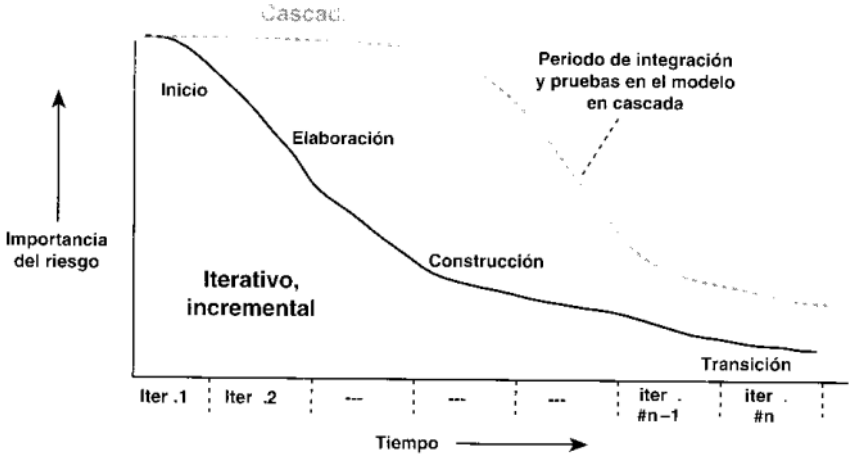

Ilustración 18: Riesgos durante iteraciones del proyecto. Ivar Jacobson et al. [11]

3.1.1.2 Obtener una arquitectura robusta

La obtención de una arquitectura robusta es el resultado de las iteraciones sobre el inicio del proyecto. En estas, se puede encontrar una arquitectura mínima que satisface los requisitos claves, evita los riesgos críticos y resuelve los principales problemas de desarrollo.

La idea es estabilizar una arquitectura lo más robusta posible al principio del ciclo de vida, cuando los costos aún son bajos.

3.1.1.3 Gestionar requisitos cambiantes

Desde la perspectiva de los usuarios, es importante hacer evolucionar el producto a lo largo de una serie de versiones ejecutables. Cada una de estas versiones, es una parte operativa del sistema y el conjunto de éstas constituyen iteraciones, las cuales progresan hasta alcanzar un resultado esperado o incremento.

Tener un sistema con un funcionamiento parcial en una instancia inicial, concede que los afectados por este, realicen aportes y sugerencias. Lo cual, permite en caso de ser necesario, realizar las modificaciones correspondientes con un pequeño costo en el presupuesto o calendario. Entonces, la respuesta a una retroalimentación o incluir una revisión sólo significará un cambio incremental.

3.1.1.4 Permitir cambios tácticos

Con esta metodología, se pueden resolver problemas y aspectos no contemplados en las primeras construcciones e incluir cambios para corregirlos casi de manera instantánea. Con esta técnica, los problemas se van descubriendo poco a poco con lo cual es mucho más sencillo lidiar con las partes que conforman el proyecto, tales como clientes, directivos, programadores y demás; también permite adaptar el calendario en base al tipo de problema que se trate.

3.1.1.5 Conseguir una integración continua

Al finalizar cada iteración queda en evidencia que se han reducido algunos riesgos. Se entregan nuevas funcionalidades en cada iteración, que son evidentes para los usuarios, los cuales pueden observar el avance del proyecto. Además, al realizarse integraciones de forma regular, muchos de los problemas pueden ser resueltos de forma rápida y seguir avanzando conforme se vayan solucionando.

3.1.1.6 Conseguir un aprendizaje temprano

Luego de una pequeña cantidad de iteraciones, al principio del proyecto, todos los integrantes del equipo tienen una correcta comprensión del mismo. Así, es fácil incorporar personal al proyecto debido que, pueden formarse con el propio trabajo y cuentan con compañeros

capacitados. En el caso de que alguien nuevo no aprenda lo suficientemente rápido o cometa errores, su fallo no representa un problema mayor para el progreso a largo plazo del proyecto.

El método iterativo incremental también, ayuda al proyecto a tratar riesgos de naturaleza no técnica, como los riesgos de la organización, por ejemplo, capacitar a los programadores en el uso de las tecnologías que afectan al proyecto.

Mediante el trabajo dividido en iteraciones, el personal es cada vez más capaz de cumplir con las necesidades de los clientes y reducir los riesgos. En la construcción por incrementos, todos los implicados pueden observar el nivel de progreso. La reducción de los problemas de último momento, aceleran los tiempos de puesta en producción del producto.

3.1.2 Iteración e incremento

En proyectos de software grandes, los requerimientos cambian cuando el negocio se enfrenta a factores externos, cuando cambian las prioridades de gestión, tecnologías y demás. Todo esto hace que, el proceso de desarrollo no sea único y que las actividades del proceso se repitan regularmente. Frente a esto, se planifican nuevas funcionalidades y cambios a través de iteraciones, donde cada una proporciona un subconjunto de funcionalidades nuevas o correcciones al sistema. La asignación de funcionalidades a los incrementos depende del nivel de prioridad, donde los de prioridad más alta son los que se entregan primero.

Una vez que un incremento se completa y entrega, los clientes pueden utilizar esa funcionalidad del sistema. Esto significa que, tienen una entrega temprana desde el inicio del proyecto. Pueden experimentar con el sistema, lo cual ayuda a clarificar sus requerimientos para los incrementos posteriores y para las últimas versiones del incremento actual. Tan pronto como se completan los nuevos incrementos, se integran a los existentes, de tal forma que, la funcionalidad del sistema mejora con cada incremento entregado. Los servicios comunes se pueden implementar al inicio del proceso o de forma incremental tan pronto como sean requeridos.

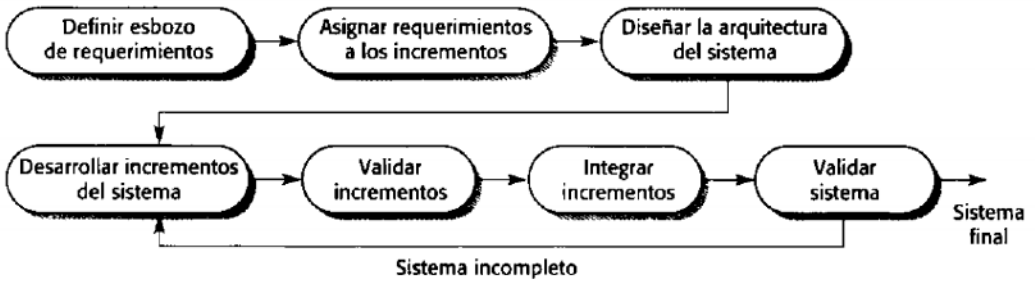

Ilustración 19: Entrega por incrementos. Iam Sommerville [10]

Cada iteración es una pasada por cinco flujos de trabajo fundamentales, donde se inicia con la captura de requisitos, se realiza un análisis, diseño, implementación, prueba y finaliza con un módulo en producción el cual puede ser utilizado por el cliente. Es para destacar que, estos flujos de trabajo se corresponden con la metodología en cascada, presentan similitudes en cuanto a su organización y trabajo.

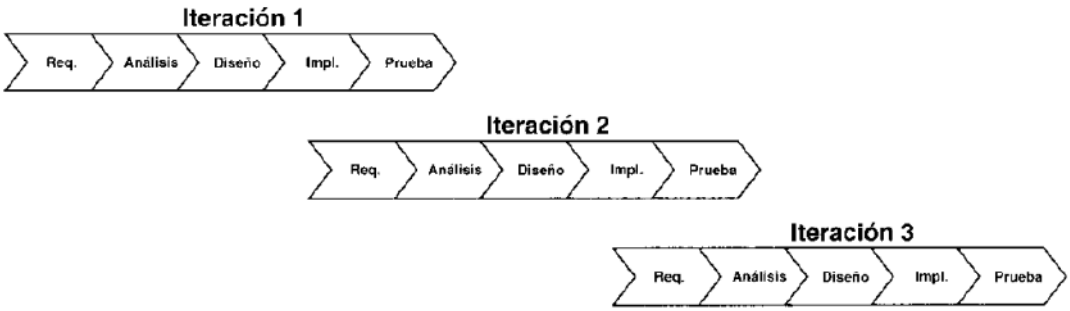

Ilustración 20: Iteraciones que conforman el proyecto.

3.2 Ventajas del proceso incremental

El proceso de desarrollo incremental presenta varias ventajas:

1. Los clientes no tienen que esperar hasta que el sistema completo se entregue para utilizarlo. El primer incremento, satisface los requerimientos más críticos de tal forma que pueden hacer uso de parte del software prácticamente desde el inicio.
2. Los clientes pueden utilizar los incrementos iniciales como prototipos y obtener experiencia sobre los requerimientos posteriores del sistema.
3. Existe un bajo riesgo de un fallo total del proyecto. Aunque, se pueden encontrar problemas en algunos incrementos, lo normal es que el sistema se entregue de forma satisfactoria al cliente.
4. Puesto que los servicios de mayor prioridad se entregan primero, y los incrementos posteriores se integran a ellos, resulta inevitable que los servicios más importantes del sistema sean a los que más pruebas se les realicen. Esto significa que, es poco probable que el cliente encuentre fallos en las partes más importantes del sistema.

3.3 Desventajas del proceso de desarrollo incremental

Existen algunas desventajas en el proceso de desarrollo incremental:

1. Los incrementos deben ser relativamente pequeños, no deben superar las 20.000 líneas de código y cada uno debe entregar alguna funcionalidad del sistema.
2. Resulta difícil adaptar los requerimientos del cliente a incrementos de tamaño apropiados.
3. Puede ocurrir que el sistema necesite hacer uso de varios recursos que se encuentran en diferentes incrementos, por lo que pueden generarse tiempos de espera, hasta tanto no se completen los incrementos necesarios.

3.4 Utilización de esta metodología durante el proyecto

La utilización de la metodología mencionada previamente se dio desde el inicio del proyecto, donde fue necesaria la creación de un prototipo de software para mostrarle al cliente. Debido a que es un sistema desarrollado a medida, de acuerdo a las necesidades de la empresa, se optó por realizar entregas una vez finalizada cada iteración, tal cual indica el proceso iterativo incremental. Luego, se realizaba un incremento y se proseguía con la siguiente iteración.

La primera acción fue la de realizar un relevamiento de requisitos, donde se recopiló la mayoría de los requerimientos funcionales del sistema. Con esto se pudieron confeccionar las diferentes iteraciones que afectarían el desarrollo del sistema, también se realizó un cálculo estimativo del tiempo que tomaría el desarrollo completo de cada una de las iteraciones y de esta manera poder indicar una fecha posible de entrega. Se puede mencionar que las primeras iteraciones fueron pensadas especialmente con el foco puesto en la mitigación de riegos, recordando, que los riesgos más elevados deben colocarse en las primeras iteraciones del proceso de desarrollo, de modo que, para el final del mismo, los riesgos fueron mitigados casi por completo.

A medida que el desarrollo avanzó y se entregaban funcionalidades finalizadas al cliente, este se encargaba de testearlos, adaptarse al uso del sistema en un principio y posteriormente a las nuevas funcionalidades que iban sumándose. Con esto, el cliente realizaba una retroalimentación sumamente valiosa, ya que, de esta, se podía continuar con las siguientes iteraciones o se debía volver hacia atrás y realizar la modificación correspondiente.

Terminada la última iteración, el sistema se encuentra en un estado de finalización. En caso de que surgiese alguna nueva funcionalidad o cambio sobre lo que ya está hecho, se debe crear una nueva iteración y realizar la implementación o cambio correspondiente.

Por último, resta mencionar que este tipo de metodología funciona muy bien cuando el cliente está dispuesto a trabajar en conjunto con el desarrollador o equipo de desarrollo, ya que continuamente, se encuentra utilizando el sistema, evaluándolo y realizando devoluciones acerca del funcionamiento del mismo. Para llevar a cabo este proyecto, fue importante establecer pautas en cuanto a horarios de reuniones con la empresa, por lo general los

encuentros eran presenciales al inicio del proyecto y en horario de tarde. Luego, frente a la situación de público conocimiento por la pandemia de COVID-19, es que se optó por realizar encuentros virtuales a través de WhatsApp, Hangouts, Zoom o Skype, también por la tarde. Dichas reuniones, tenían una duración, en promedio, de una hora y en las mismas se hablaba sobre que se hizo, que se estaba haciendo en ese momento y que se haría después. En cuanto a lo que ya se encontraba implementado y testeado por el cliente, este realizaba devoluciones y sugerencias de cambios que consideraba oportunos. Lo que se estaba realizando durante las distintas iteraciones, era consultado tanto a la empresa como a su contadora, a fines de obtener una solución de software lo más adecuada posible al problema planteado. Por último, se indicaban en que consistirían los futuros módulos a desarrollar, indicando cuando estarían listas las nuevas funcionalidades y cuando se verían reflejados los cambios solicitados.

Es destacable considerar que la metodología incremental permite realizar cambios rápidos durante el proyecto sin mayores complicaciones, a diferencia de otras metodologías, tales como cascada, donde frente a un cambio o nuevo requisito en etapas avanzadas del proyecto, es necesario realizar un roll-back y repetir todas las etapas nuevamente. Claramente, una metodología de desarrollo tradicional hubiera implicado un costo alto, y posiblemente los ajustes a lo implementado requerirían más trabajo y por ende una mayor demora, con lo cual, el avance del proyecto se hubiese visto comprometido seriamente provocando costos extras en tiempo o dinero, por ejemplo.

4 Desarrollo del sistema

Este capítulo tratará sobre las diferentes etapas de desarrollo del sistema. Comenzando con la inicialización, donde se capturaron los requisitos del sistema y en base a diferentes criterios, se eligieron los de mayor necesidad y criticidad. Una vez terminado ese análisis previo, se confeccionaron las diferentes iteraciones que tuvo el sistema. Al final de cada una de las iteraciones, se obtuvo un incremento de funcionalidades, las cuales se acoplaban al sistema en producción. Al finalizar la última iteración, el sistema se encontraba testeado por el desarrollador y por la empresa, y los riegos disminuyeron casi por completo.

4.1 Inicio

Como bien se habló capítulos anteriores, el inicio del presente proyecto se dio por la necesidad de la empresa Amanecer Servicios Agropecuarios S.R.L de automatizar ciertas tareas de gestión contable, para ello, las partes implicadas se reunieron en varias entrevistas, donde la dueña de la empresa indicó la necesidad de automatizar la carga de información pertinente a las compras y ventas que su empresa realiza de forma mensual. Además, solicitó la creación de un panel donde se pudieran visualizar diferentes estadísticas acerca del estado de su negocio. Frente a estos requisitos, se realizó un relevamiento profundo de los procesos de manipulación contable de Servicios Agropecuarios Amanecer S.R.L. Se trabajó con el respaldo de la contadora de la empresa, quién también es usuaria en el sistema.

Por último, es válido aclarar, que conforme se iban desarrollando los objetivos propuestos, la empresa permaneció en contacto permanente, sugiriendo modificaciones, indicando nuevos requerimientos para desarrollar y evacuando las dudas que surgieron durante el proceso.

4.1.1 Desarrollo de diagrama de casos de uso

En ese momento se realizó un diagrama de casos de uso, el cual contempló dos tipos de actores diferentes entre sí, la dueña de la empresa y la contadora. Como casos de uso se destacan:

1. Realizar ABM Cliente.
2. Realizar ABM Proveedor.
3. Realizar ABM Movimiento de cliente.
4. Realizar ABM Movimiento de proveedor.
5. Generar reporte clientes.
6. Generar reporte proveedores
7. Generar reporte movimiento de clientes.
8. Generar reporte movimiento de proveedores.
9. Realizar ABM de chofer.
10. Realizar ABM de viaje.
11. Generar reporte viajes.
12. Visualizar estadísticas.
13. Ver listado de clientes.
14. Ver listado de proveedores.
15. Ver listado movimientos de clientes.
16. Ver listado movimiento de proveedores.

Es oportuno mencionar que los casos de uso números 9, 10 y 11 se expresan aquí en el conjunto total, pero fueron casos de uso de requisitos posteriores a los iniciales. Por lo tanto, su tratamiento fue realizado luego de iniciadas las iteraciones planificadas.

A continuación, se puede observar el diagrama de casos de uso final, obtenido después de completar la última iteración:

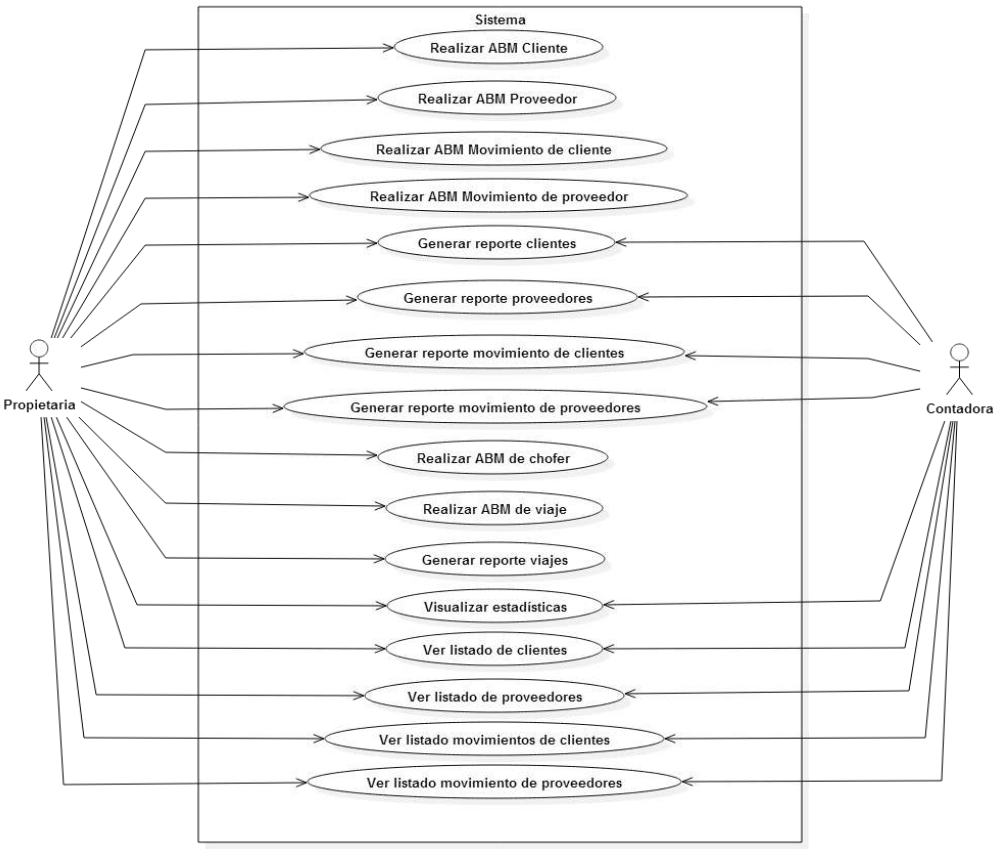

Ilustración 21: Diagrama de Casos de Uso

4.1.2 Planificación de las iteraciones y creación de un cronograma de trabajo

Para realizar una planificación de todas las iteraciones, se realizó una estimación sobre el tiempo que estas tomarían. Además, se analizó el riesgo que tendría la implementación de cada uno de los casos de uso y la necesidad de implementar uno de manera previa a otro. Los primeros casos de uso serían los que tienen relación con la gestión de clientes y proveedores; y la de movimientos de clientes y proveedores, para luego seguir por los casos de uso que respondan a elaboración de informes y finalmente, explotar los casos de uso para obtención de información para toma de decisiones.

En ese punto del proyecto, se realizó un diagrama de Gantt sobre el tiempo estimado que tomaría cada una de las iteraciones, el mismo puede observarse a continuación:

Ilustración 22: Diagrama de Gantt.

Siguiendo con este apartado, se enumeran las 9 iteraciones planificadas para desarrollar el sistema, cabe aclarar que, se planificaron 7 iteraciones, pero durante el transcurso de la iteración 3, la empresa requirió una funcionalidad extra al sistema, por lo tanto, la iteración 4 corresponde a esa nueva solicitud y también se añade una iteración extra para atender a una modificación sugerida por la dueña, la cual corresponde a la iteración 7:

1. Iteración 1: Aquí se incluye la instalación y aprendizaje de cada una de las tecnologías más importantes en este proyecto (Para detalle de instalación y uso ver Anexo 1, 2 y 3). Además, se implementó un primer prototipo con las principales interfaces con las que cuenta el sistema. En cuanto a las funcionalidades, se desarrollaron los ABM tanto de clientes como de proveedores y, los ABM simplificados de los movimientos de clientes y proveedores. La razón de implementar el módulo de movimientos de clientes y proveedores de forma mínima, fue la de esperar la aceptación por parte de la dueña, y en caso de solicitar algún cambio, no tener que volver a reestructurar todo lo realizado hasta ese momento.

2. Iteración 2: Se definieron perfiles de usuario, se implementaron las funcionalidades correspondientes al sistema de login y vistas de usuarios. Además, se llegó a la conclusión que el sistema tendría dos perfiles de usuarios distintos. Por un lado, el perfil “Control Total”, correspondiente a la dueña de la empresa. Este perfil da permisos para el control total sobre los distintos módulos que conforman el sistema. Por el otro lado, un perfil para usuarios que sólo requieran visualizar información y

generar informes, tal es el caso de la contadora. Ambos perfiles se crearon a través del administrador que provee Django, del cual se habló en Capítulo 2, sección 2.1.2.5 “Otros elementos de Django”.

3. Iteración 3: Se implementaron las funcionalidades restantes de cálculos en base a IVA tanto de los movimientos de clientes, como de los movimientos de los proveedores. Esta fue una de las funcionalidades de mayor riesgo en el sistema, ya que representó la principal necesidad que motivó la creación de un software para la empresa.

4. Iteración 4: En este punto el cliente agregó un requisito extra al proyecto, el cual fue la necesidad de tener un registro de los viajes realizados de manera mensual, para ello se implementó el módulo de carga, modificación y eliminación de choferes y también de los viajes que éstos realizan de manera mensual.

5. Iteración 5: Corresponde a la generación de informes sobre los movimientos de clientes y movimientos de proveedores; clientes, proveedores, choferes y viajes.

6. Iteración 6: Corresponde a la modificación de un requerimiento de la empresa, donde se solicitó que los movimientos tanto de clientes como de proveedores puedan contener en la misma operación diferentes alícuotas de IVA.

7. Iteración 7: Aquí se realizó la implementación del módulo que genera datos estadísticos, acerca de los ingresos y egresos de dinero de la empresa en un determinado período. Además, se creó un panel para visualizar dichos datos de manera ordenada a través de la utilización de diferentes tipos de gráficos.

8. Iteración 8: Corresponde a la implementación de un módulo de control el cual se encarga de la carga, modificación y eliminación de información interna que utiliza la empresa para gestionar clientes, movimientos de clientes, proveedores y movimientos de proveedores.

9. Iteración 9: Corresponde a cambios menores sugeridos por el cliente, tales como modificar el formato de la información mostrada e informes. Cambio de estilos en tablas, botones y gráficos.

4.1.3 Diagrama de Entidad Relación DER

La razón de aplicar la metodología iterativa incremental es la de dividir al sistema en pequeñas partes o iteraciones que resultan en incrementos. Dentro de cada una de esta iteración, se desarrolla una funcionalidad de software que termina incluyéndose dentro del sistema aprobado hasta ese momento. Para llevar a cabo esta tarea, de obtener un incremento, primero se debe realizar un relevamiento de requisitos como bien se dijo anteriormente, luego realizar el análisis y diseño correspondiente. En este punto es donde se construye un diagrama de entidad relación. La particularidad de la metodología iterativa incremental es que, va construyendo el DER a medida que se realizan iteraciones, por lo tanto, al inicio de las iteraciones, se obtiene un DER acotado, y conforme se itera, este va ampliándose y modificándose. Para el final de la etapa de iteración, se obtiene un diagrama completo, el cual presenta un alto grado de fidelidad con el análisis de los requerimientos relevados a lo largo del desarrollo del proyecto.

A continuación, se presenta el DER, obtenido después de realizar la última iteración planificada:

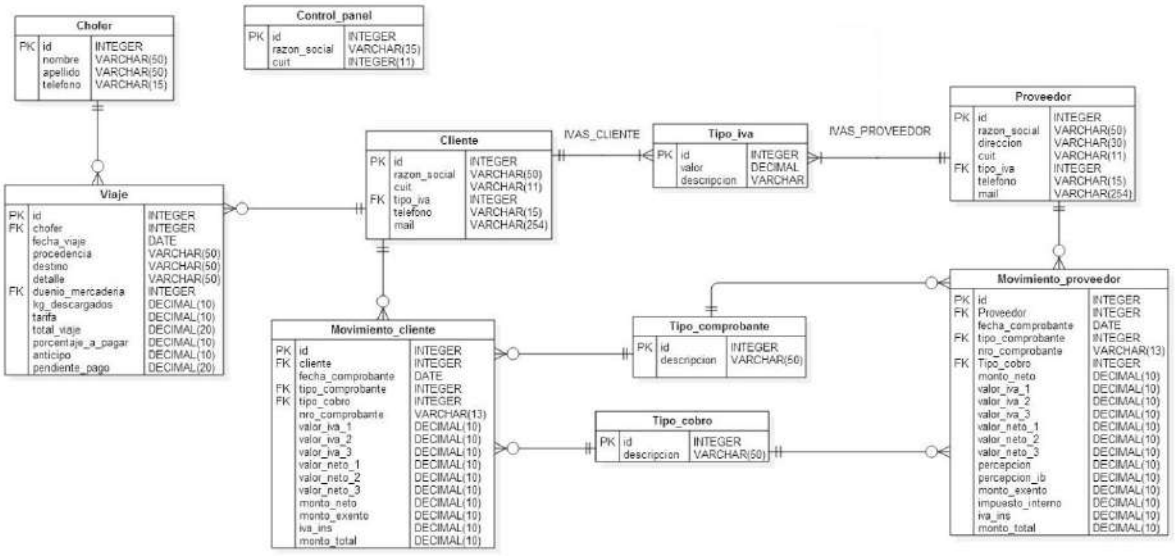

Ilustración 23: Diagrama de Entidad Relación (DER). Construido con StarUML V2.1.2.

4.2 Iteraciones

Esta sección realiza una descripción en detalle sobre las distintas iteraciones que conformaron el desarrollo del proyecto. El detalle de cada una de las iteraciones se especifica a continuación:

1. Iteración 1: En la primera iteración se realizaron las tareas de descarga e instalación de los softwares y tecnologías que estuvieron implicadas en el desarrollo del sistema (Para detalle de instalación y uso ver Anexo 1, 2 y 3), se descargó e instaló el editor de texto Sublime, se actualizaron todas las herramientas de comunicación y navegadores web, se creó el entorno virtual VirtualEnv que contendría el proyecto desarrollado con Django, se realizó la descarga e instalación del framework mencionado y se procedió a estudiar su utilización, se descargó, instaló y configuró la base de datos PostgreSQL. También se instalaron tecnologías de FrontEnd como Bootstrap, de acuerdo a los pasos que establece Django para el manejo de archivos estáticos. Por último, en cuanto a configuraciones e instalaciones se refiere, se descargó e instaló un dashboard o tablero, a los fines de reducir los tiempos de implementación de interfaces de usuario.

Una vez que todo funcionó correctamente, se creó un usuario administrador de Django y un usuario de pruebas, ambos perfiles fueron creados utilizando el administrador de Django del cual se habló en Capítulo 2, sección 2.1.2.5 “Otros elementos de Django”. Se procedió a la creación de los primeros módulos del proyecto, ellos fueron la realización de alta, baja y modificación de clientes, proveedores, movimientos de clientes y movimientos de proveedores. Como se mencionó previamente, se realizó una implementación parcial sobre el módulo de movimientos ya que se esperaba de forma anticipada alguna sugerencia o modificación de la dueña. Para la creación de éstos, fue necesario crear las Apps *cuentaCorrienteCliente* y *cuentaCorrienteProveedor*, luego crear las URLS dentro del *mapeador URL* y enlazarlas, crear las vistas correspondientes a la gestión de las Apps mencionadas, y los *modelos* que responden a las vistas. Para la creación de los *modelos*, se tomaron del diagrama de entidad relación visto en la sección 4.1.3, las entidades correspondientes a *Cliente*, *Proveedor*, *Movimiento_cliente* y *Movimiento_proveedor*. En este punto, el módulo de movimientos no realizaba calculo alguno, sólo se limitaba a la carga de información. Para la visualización de información de manera ordenada, se utilizó el plugin *DataTables* el cual se instaló finalizados todos los ABM comprendidos en esta iteración.

Se creó un repositorio en *GitHub*, el cual fue utilizado para almacenar las actualizaciones que fueron surgiendo a medida que se iteraba. La manera de utilizarlo era desarrollar en pc local y actualizar el repositorio con las nuevas funcionalidades o modificaciones.

Se utilizó el servicio de host provisto por *PythonAnywhere*, en donde se creó un *VirtualEnv*, se procedió con la instalación y configuración de éste de manera análoga a la pc utilizada para desarrollo, una vez hecho esto, se realizó una clonación del repositorio en *GitHub* hacia el servidor para posteriormente realizar las configuraciones propias del servicio host.

Finalmente, se realizó un encuentro con la dueña de la empresa donde, se mostraron los avances hasta ese momento, se explicaron funcionalidades y formas de trabajo con el sistema. En este punto, se habló sobre finalizar con la implementación del módulo de movimientos de clientes y proveedores y por tal motivo, se planificó una

reunión con la dueña y su contadora. Por último, se le entregaron las credenciales de acceso correspondiente al usuario de prueba. A partir de aquí estuvo habilitada para ingresar al sistema, comenzar a utilizarlo y realizar las devoluciones fundamentales para la retroalimentación del desarrollo de la aplicación.

2. Iteración 2: Aquí se incluye la reunión con la contadora en primer lugar, de allí, se determinó que el sistema iba a tener dos perfiles de usuario diferentes, por un lado, el perfil de la dueña con control total sobre el sistema, y por el otro lado, el perfil de la contadora, con solo los permisos de visualización de clientes, proveedores y los movimientos correspondientes de cada uno. De dicha reunión, también se especificó la manera de trabajar con los movimientos de clientes y proveedores, y de cómo se realizan los cálculos para registrar los movimientos.

   En cuanto a desarrollo, se implementaron las funcionalidades correspondientes al sistema de login y vistas de usuarios. Se crearon los perfiles “Control Total”, correspondiente a la dueña de la empresa. Este perfil, da permisos para el control total sobre los distintos módulos que conforman el sistema. Por el otro lado, un perfil para estos usuarios que sólo requieran visualizar información y generar informes, tal es el caso de la contadora. Ambos perfiles, se crearon a través del administrador que provee Django.

3. Iteración 3: Los cálculos que se realizan tanto para movimientos de clientes y proveedores, determinan el monto total que corresponde a cada uno de los comprobantes que se ingresa al sistema. La siguiente Ilustración corresponde a la interface de carga de movimientos de clientes:

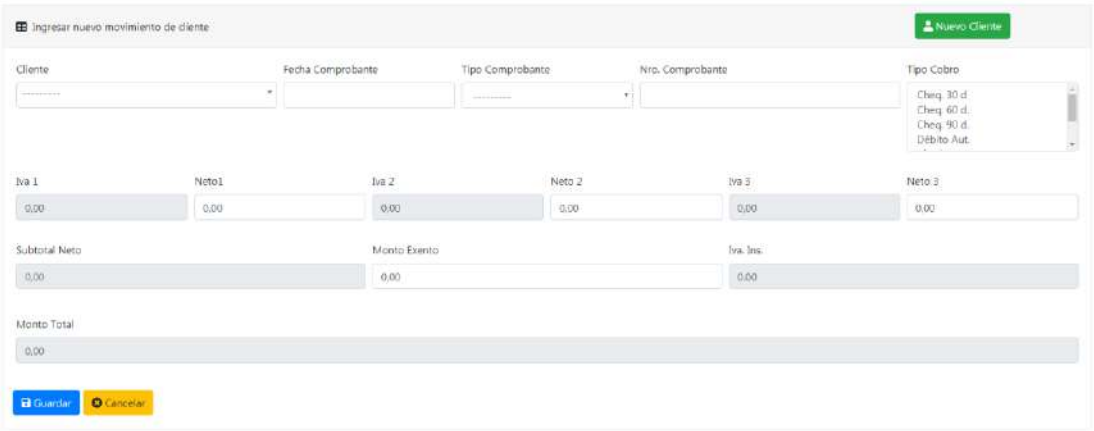

Ilustración 24: Interface de carga de movimientos de cliente.

En esta imagen se pueden apreciar el formulario de carga con los siguientes campos:

- *Cliente*: Es un selector con un buscador incluido, selecciona el cliente al cual pertenece el movimiento.
- *Fecha Comprobante*: Selecciona la fecha del comprobante a cargar.
- *Tipo Comprobante*: Selecciona el tipo de comprobante a cargar. Los tipos de comprobante pueden ser: factura A, factura B, factura C, nota de crédito, nota de débito y seguro.
- *Nro. Comprobante*: Indica el número de comprobante a registrar.
- *Tipo de cobro*: Indica el tipo de cobro a registrar, estos pueden ser: cheque 30,60 y 90 días, débito automático, efectivo, tarjeta de crédito o transferencia bancaria.
- *IVA1, IVA2, IVA3*: Estos campos se cargan automáticamente a través de una llamada *Ajax* [9] al servidor, en ellos se almacenan los diferentes tipos de IVA que tiene el usuario seleccionado.
- *Neto1, Neto2, Neto3*: En estos campos se completan los montos correspondientes al valor afectado por los distintos *IVAs*.
- *Subtotal Neto*: este campo contiene la suma de *Neto1, Neto2 y Neto3*.
- *Monto Exento*: Este campo permite el ingreso del dato valor exento en caso de que exista en el comprobante a cargar.

- *Iva Ins*: Este campo contiene el resultado de la suma de los cálculos de porcentaje de los *IVA*s a los valores *NETOs*.
- *Monto Total*: Este campo corresponde a la suma de los campos *Subtotal Neto*, *Monto Exento* e *Iva Ins*.

Para realizar los cálculos se utilizó *Javascript* y para completar los campos relacionados a los distintos *IVA* se utilizó *Ajax* mediante la selección de un cliente, el servidor procesa la petición en la vista correspondiente a la URL configurada y devuelve una respuesta con datos de tipo *JSON*, la cual es procesada en el cliente y posteriormente completa los campos con el *IVA* correspondiente.

De forma análoga se desarrolló la funcionalidad de movimientos de proveedores, con la diferencia de que el formulario para ingresar datos contiene algunos otros conceptos adicionales que puede tener discriminado el comprobante a registrar, en la siguiente imagen se puede observar:

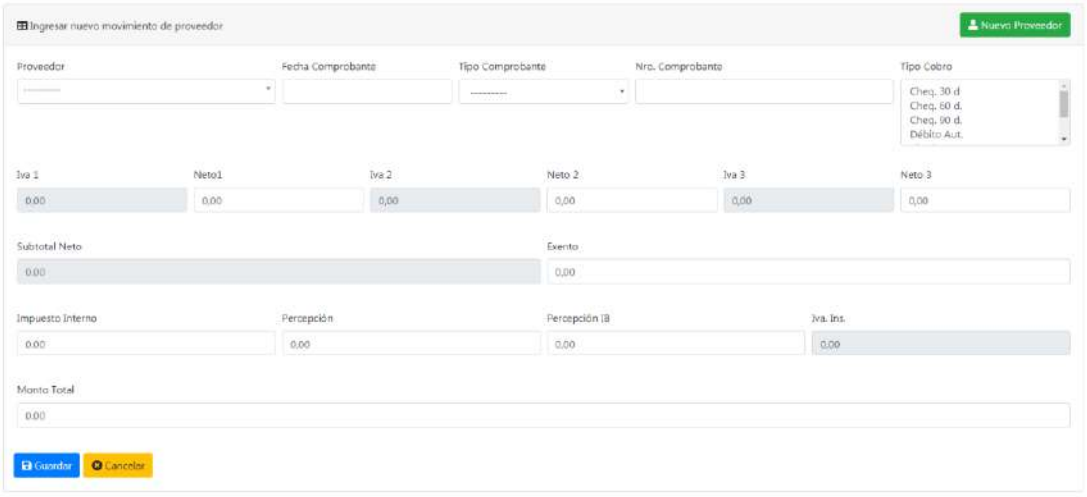

Ilustración 25: Interface de carga de movimiento de proveedor.

El desarrollo de estos dos módulos resuelve el principal problema de automatización que planteó la empresa al inicio del proyecto. Puede verse que, como representan un elevado riesgo, se colocaron dentro de las primeras iteraciones; pero ocurre que, para

su desarrollo, fue necesario haber implementado previamente las funcionalidades de carga de clientes y proveedores.

Además, la dueña de la empresa, estableció un requerimiento funcional extra, el cual fue poder llevar un registro de los viajes realizados por los choferes de la empresa. Dicho requerimiento se consideró para la iteración número 4.

Por último, resta mencionar que, esta iteración concluyó con la carga en el servidor de las nuevas funcionalidades implementadas y una reunión con la dueña de la empresa donde se trataron los puntos desarrollados.

4. Iteración 4: Esta iteración desarrolla el requerimiento funcional solicitado por la dueña durante el transcurso de la iteración número tres, la funcionalidad solicitada era poder llevar un registro de los viajes realizados por sus choferes de manera mensual, donde en el mismo, se registrara el asiento de la tarifa pagada al chofer de acuerdo a las toneladas transportadas y a la tarifa por el tipo de carga. Previamente al desarrollo del módulo de viajes, se debió crear un módulo de choferes donde se pudiera realizar el alta, baja y modificación de los conductores. Una vez completado ese desarrollo previo, se procedió con la implementación del módulo de viajes.

   En la siguiente imagen se puede apreciar la interface creada para la carga de viajes realizados:

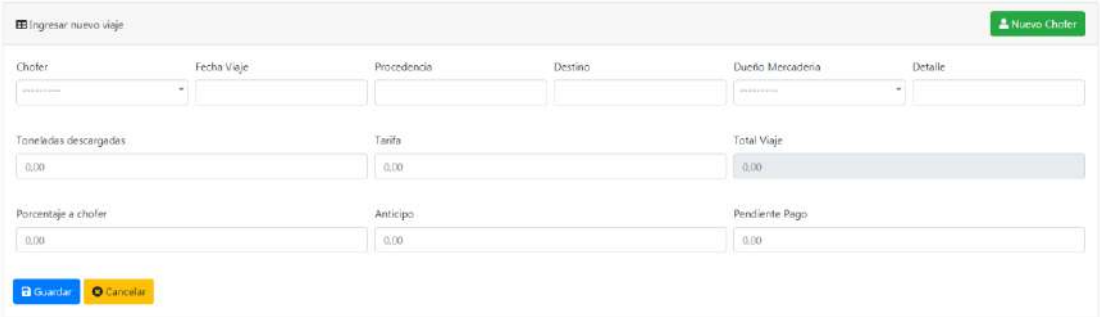

Ilustración 26: Interface de carga de viaje.

Los campos involucrados en la carga de viajes son:

- *Chofer*: Indica el chofer a cargo de realizar el viaje ingresado.
- *Fecha Viaje*: Indica la fecha en que se realizó el transporte.
- *Procedencia*: Indica el lugar origen de la mercadería transportada.
- *Destino*: Indica el lugar destino de la mercadería transportada.
- *Dueño mercadería*: Indica el dueño de la mercadería transportada, dato que se vincula a los clientes existentes.
- *Detalle*: Corresponde a un campo de opción en caso de que se desee agregar alguna observación al viaje.
- *Toneladas descargadas*: Indica la cantidad de toneladas descargadas en el viaje.
- *Tarifa*: Indica el valor por tonelada de carga transportada.
- *Total Viaje*: Indica el valor del viaje, este valor se calcula realizando el producto entre las Toneladas descargadas y la tarifa.
- *Porcentaje a chofer*: Indica el porcentaje del Total de Viaje pagado al chofer.
- *Anticipo*: Indica en caso de que el chofer lo requiera, un anticipo de dinero para el viaje.
- *Pendiente Pago*: Indica el monto de dinero a pagar al chofer por sus servicios.

Para la implementación de las funcionalidades mencionadas anteriormente, se utilizaron las entidades *Chofer* y *Viaje* del diagrama de entidad relación. Como tareas finales en esta iteración, se subieron a producción las funcionalidades nuevas y se pactó una nueva reunión con la dueña de la empresa.

5. Iteración 5: Las tareas de desarrollo involucradas en esta iteración fueron las de implementar los módulos para generar formularios en formato pdf de clientes, proveedores y choferes registrados en el sistema, movimientos tanto de clientes, como de proveedores, y también viajes realizados. Para implementar esta funcionalidad, se utilizó un módulo desarrollado en Python llamado *xhtml2pdf* [5], el cual permite generar documentos pdf a partir de contenido HTML además de permitir el control de flujos para poder realizar listados de manera sencilla.

Concluido el desarrollo se subieron las nuevas funcionalidades al servidor y se pactó una reunión virtual con la dueña, la razón de utilizar canales de comunicación virtuales se debió al decreto del aislamiento social, preventivo y obligatorio sancionado por el Gobierno Nacional. En esta reunión, se solicitó modificar el módulo de registro de movimientos de clientes y proveedores, ya que tantos los clientes como los proveedores, pueden presentar diferentes tipos de IVA.

6. Iteración 6: Corresponde a una modificación solicitada por la empresa, donde pidió que los movimientos tanto de clientes, como de proveedores, puedan discriminar tipos diferentes de IVA. Frente a este cambio, es que se debió realizar una reestructuración que afectó a varios módulos del sistema. Una modificación de esta naturaleza, donde se trata con áreas sensibles del sistema, debería surgir durante las primeras iteraciones. Afortunadamente, el enfoque iterativo incrementa permite este tipo de cambios y brinda la posibilidad de solucionarlos sin mayores complicaciones ni demoras de tiempo.
   Las entidades modificadas del diagrama de entidad relación fueron las de *Cliente*, *Proveedor*, *Movimiento de cliente*, *Movimiento de proveedor*. Además, para facilitar la carga de distintos datos, tanto de clientes, proveedores y sus movimientos, se decidió crear las entidades *Tipo_iva*, *Tipo_comprobante y Tipo_cobro.*
   Finalizada la tarea de desarrollo, se subieron los cambios a producción y se contactó a la empresa mediante video llamada para testear en conjunto los cambios realizados. Una vez aprobadas las modificaciones se dio por concluida esta iteración.

7. Iteración 7: Esta iteración corresponde al desarrollo de un módulo para obtener información para la toma de decisiones gerenciales. Dicha información, corresponde a ingresos y egresos de la empresa discriminados de manera mensual y la confección de un gráfico de estado de resultado. Para el manejo de gráficos se utilizó un plugin jQuery llamado *Charts.js*, el cual provee todas las funcionalidades necesarias para la creación de gráficos de barra, torta, lineales, etc.
   En la siguiente imagen se puede observar la interface creada para visualizar información utilizando el plugin mencionado:

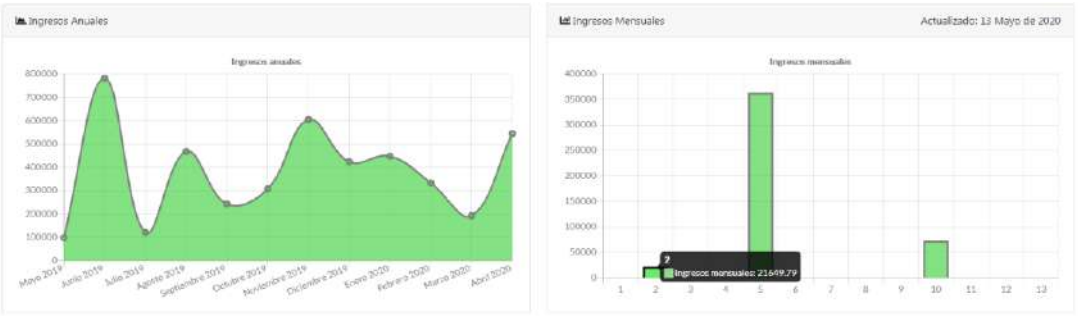

Ilustración 27: Gráfico estadísticas de ingresos.

De manera análoga se utilizó *Charts.js* para representar información de los flujos de egreso de la empresa y de estado de resultado.

8. Iteración 8: Esta iteración corresponde al desarrollo de un módulo de control, al que solamente tiene acceso el usuario con los permisos adecuados. Este módulo tiene las funcionalidades para realizar altas, bajas y modificaciones de los diferentes alícuotas de IVA que pueden tener los clientes y proveedores. También, gestiona los diferentes tipos de comprobantes y cobros que pueden realizarse en un movimiento tanto de clientes, como de proveedores. Además, contiene una sección para modificar el nombre y cuit de la empresa en caso de que sea necesario.
   Con respeto a la implementación para la empresa Otermin S.A. se puso el sistema en un host distinto haciendo el recorte de las funcionalidades pautadas con la dueña.
   Finalizada la tarea de desarrollo, se realizó una reunión virtual con la dueña donde sugirió realizar algunos cambios menores a nivel global del proyecto. Tales cambios, contemplan modificaciones de estilos, agregado de guiones a teléfonos, comprobantes, cuits y detalles en los diferentes listados y registros que existen en el sistema.

9. Iteración 9: Esta corresponde a una de las últimas iteraciones planificadas y contempla los cambios sugeridos por la dueña de la empresa. Los mismos se realizaron acorde a sus sugerencias de agregar guiones a cuits, teléfonos y números de comprobantes a fines de mejorar la interface acorde al estilo de trabajo del usuario.

Para ello, se utilizó un plugin jQuery llamado *mask.js*, este provee las funciones necesarias para realizar un enmascarado de los datos para su correcta visualización. Por último, se realizaron mejoras en los estilos de botones y tablas.

Conclusión

Para concluir con el presente informe, resta decir que se lograron los objetivos planteados en la sección 1.2 a través del uso de diversas tecnologías y utilizando como base una metodología reconocida en la ingeniería de software. Además, se aplicó una gran cantidad de conocimientos adquiridos a lo largo de estos años como estudiante de la carrera Ingeniería en Sistemas y también logrados durante mi incipiente carrera profesional.

A través de este proyecto, se obtuvo un software desarrollado a medida, el cual resuelve los problemas presentados por la empresa. Este sistema se encuentra en un estado donde los riesgos son mínimos debido a la constante interacción con la dueña, la cual siempre estuvo presente para realizar aportes que permitieron una retroalimentación sobre el sistema. Cabe decir que ya se han planificado nuevas iteraciones para agregar nuevas funcionalidades al software en producción.

La utilización de este sistema se aprecia en los cierres de mes de la empresa, en donde la generación de informes de movimientos resulta provechoso a la hora de ahorrar tiempo que antes se destinaba a realizar cálculos de forma manual. También gracias a que se guarda información contable, se puede tener de forma instantánea un estado de resultado y de esta manera saber cuál es la situación en que se encuentra la empresa. También se agilizó el procesamiento de información para realizar las liquidaciones de viajes de los distintos choferes que conforman la flota de Amanecer Servicios Agropecuarios S.R.L.

Por último, resta mencionar que este desarrollo representó un desafío tanto de tiempo como técnico al tener que aprender a utilizar tecnologías en parte desconocidas y en parte saber cómo integrarlas a las ya aprendidas. Y sobre todo resaltar que el framework Django es sumamente versátil a la hora de desarrollar un software, ya que permite realizar cualquier tipo de sistema de manera rápida y segura gracias a sus librerías y patrón MTV.

Considero oportuno destacar que si bien el desarrollo ha sido una solución para la empresa, esto no revistió gran complejidad en sí mismo, motivo por el cual se decidió cubrir dos objetivos adicionales inicialmente no previstos; en primer lugar explicar detalladamente todas los recursos y tecnologías usadas, y en segundo elaborar un conjunto de anexos

independientes que sirvan de guía para la instalación de todo el software utilizado para que desde la cátedra de Bases de Datos se pudiese usar como guía en el proceso de elaboración de proyectos sencillo y ágiles.

Anexo 1 – Instalación y uso de VirtualEnvs

La instalación de *Virtualenv*, se realiza a través de *pip*, el cual es un sistema de gestión de paquetes, que se utiliza para instalar otros paquetes de software escritos en Python. Para ello, desde el terminal global del sistema se ingresa el comando *pip install virtualenv*, tras este paso, el sistema ya se encuentra listo para la creación de entornos virtuales.

Para la creación de un *Venv*, se debe elegir en primera medida, la ubicación donde se alojará el proyecto, en este caso se utilizó la ruta *C:/Users/Matias-PC/Dev*, el segundo paso es, ingresar por consola la instrucción *virtualenv NOMBRE_ENTORNO_VIRTUAL*. Una vez creado el entorno virtual, este debe ser activado, con lo cual se debe ingresar al directorio del proyecto, y dirigirse al subdirectorio /*Scripts* y escribir por consola *activate*. Con esto, el entorno virtual ya se encuentra activado y listo para recibir la instalación de paquetes como Django. Para desactivarlo y salir del entorno virtual, se debe ingresar por consola el comando *deactivate*.

Anexo 2 - Instalación y configuración de Django

En este anexo se dará una explicación sobre la instalación y configuración básica del framework dentro de un *virtualenv*.

Para comenzar con la instalación es necesario haber creado un entorno virtual, el cual contendrá el proyecto. Una vez creado e iniciado el entorno virtual se procede con la instalación de Django en la versión acorde a las necesidades del proyecto, para esto se utiliza el siguiente comando:

python -m pip install Django

Una vez instalado el framework en el entorno virtual, ya se encuentra disponible la posibilidad de crear un proyecto en Django, para esto, se utilizan una serie de comandos propios de este framework, los cuales son:

- *django-admin.py startproject "nombre_proyecto"*, este comando crea el proyecto de Django, y la estructura de directorios y elementos para trabajar en el desarrollo del sistema.
- *django-admin.py startapp "nombre_app"*, este comando se encarga de la creación de las Apps que conforman el proyecto.
- *django-admin.py createsuperuser*, este comando crea el usuario admin, con el cual se podrá ingresar al administrador de Django.

Otra serie de comandos muy utilizados para trabajar dentro del proyecto de desarrollo son:

- *manage.py makemigrations*, crea las migraciones cuando se crea o modifica un *modelo*.
- *manage.py migrate*, luego de creadas las migraciones de un *modelo*, es necesario reflejar estos cambios en la base de datos, para esto se utiliza migrate.
- *manage.py runserver*, sirve para iniciar el servidor interno que provee Django.

En cuanto a la base de datos, se utilizaró *PostgreSQL* en el ambiente de desarrollo y *MySQL* en ambiente de producción. Nótese que el uso de una u otra base de datos es transparente a

la hora de trabajar con Django ya que este resuelve de manera casi automática la interacción con la base de datos.

Para configurar la conexión entre *Django* y *PostgreSQL*, primero se debe instalar *psycopg2* [18]. Este es un adaptador entre el lenguaje de programación *Python* y *PostgreeSQL*. Su instalación requiere del comando: *pip install psycopg2*, una vez instalada la librería, se debe realizar la instalación de *PostgreSQL*. La instalación de la base de datos en local, se realiza de forma sencilla, descargando el instalador desde el sitio oficial y siguiendo los pasos de su instalación.

Una vez instalada la base de datos, el siguiente paso es modificar el archivo encargado de contener todas las configuraciones del proyecto Django. Este archivo es *settings.py*, contiene la configuración de la base de datos y Django. Se puede observar en el siguiente fragmento extraído del archivo mencionado:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'nombre_base_de_datos',
        'USER':'usuario',
        'PASSWORD':'contraseña',
        'HOST':'127.0.0.1',
        'DATABASE_PORT':'5432',
    }
}
```

Ilustración 28: Configuración con Base de Datos.

Este código contiene la configuración correspondiente para enlazar la base de datos con el framework, en él se especifica el motor de base de datos utilizado, el nombre de usuario y contraseña creados en el instante de la instalación de la base de datos, el host y puerto donde se conectará.

De manera análoga, se realiza la configuración con otro tipo de base de datos, en este caso, *MySQL*. La única diferencia es el engine, que será el correspondiente con *MySql*. El nombre, usuario y contraseña serán los creados en la base de datos de producción. Host y puerto corresponden, también, a la base de datos de producción.

Una vez realizadas las configuraciones correspondientes, se debe ejecutar el comando *manage.py migrate*, esto creará las tablas por defecto que proporciona Django. Luego, se debe crear un *superusuario* para poder ingresar al administrador de django, la creación se realiza utilizando el comando *django-admin.py createsuperuser*.

Finalmente para ejecutar el servidor se debe utilizar el comando *manage.py runserver*, una vez ingresado este comando, se deberá abrir un navegador e ingresar a *http://127.0.0.1:8000*, si todo es correcto, debe mostrar una página de bienvenida. En caso de necesitar ingresar al administrador, se hace a través de *http://127.0.0.1:8000/admin*.

El siguiente paso es crear las Apps necesarias para comenzar a desarrollar el sistema. Se utiliza el comando *django-admin.py startapp "nombre_app"*, donde "nombre_app" corresponde a *cuentaCorrienteCliente, cuentaCorrienteProveedor, gestion y transporte*.

Anexo 3 – Instalación de Bootstrap

Para instalar *Bootstrap*, se deben descargar los archivos correspondientes desde el sitio oficial e incluirlos dentro del proyecto. Por una parte, se cargan los archivos *css* y por el otro lado, los archivos *js*. Para realizar esto, se deben incluir dentro del archivo HTML base, el cual incluye un archivo llamado *header.html*, en este, se encuentran las llamadas a todos los *css* del proyecto. Con sólo referenciar un archivo *css* en este documento, basta para que sea accedido por todas las páginas que conforman el proyecto. De manera similar se procede con los archivos *js*, pero con la diferencia, de que los archivos utilizados por Bootstrap se cargan en el archivo base, mientras que el resto se carga en bloques declarados dentro de cada una de las páginas que los utilizan. Esto es para evitar una sobrecarga de archivos dentro del base con lo cual genera demora en tiempos de carga.

Referencias Bibliográficas

1. Boostrap - getbootstrap.com - https://getbootstrap.com/ - [Última vez accedido 19/05/2020]
2. Chart.js - www.chartjs.org - https://www.chartjs.org/ - [Última vez accedido 19/05/2020]
3. Datatables - datatables.net - https://datatables.net/ - [Última vez accedido 19/05/2020]
4. Django - www.djangoproject.com - https://www.djangoproject.com/ - [Última vez accedido 19/05/2020]
5. "Django — render HTML to PDF" - codeburst.io - https://codeburst.io/django-render-html-to-pdf-41a2b9c41d16 - [Última vez accedido 19/05/2020]
6. Framework Web Django (Python) - developer.mozilla.org - https://developer.mozilla.org/es/docs/Learn/Server-side/Django - [Última vez accedido 19/05/2020]
7. Git - git-scm.com - https://git-scm.com/ - [Última vez accedido 19/05/2020]
8. GitHub - github.com - https://github.com/ - [Última vez accedido 19/05/2020]
9. "How to Work With AJAX Request With Django" - simpleisbetterthancomplex.com - https://simpleisbetterthancomplex.com/tutorial/2016/08/29/how-to-work-with-ajax-request-with-django.html - [Última vez accedido 19/05/2020]
10. Iam Sommerville, "Ingeniería de Software, Séptima edición", 2005
11. Ivar Jacobson, Grady Booch, James Rumbaugh, "El Proceso Unificado de Desarrollo de Software",2000
12. JQuery - jquery.com - https://jquery.com/ - [Última vez accedido 19/05/2020]

13. JQuery Datepicker - jqueryui.com - https://jqueryui.com/datepicker/ - [Última vez accedido 19/05/2020]

14. jQuery Mask Plugin - igorescobar.github.io - https://igorescobar.github.io/jQuery-Mask-Plugin/ - [Última vez accedido 19/05/2020]

15. JSON - www.json.org - https://www.json.org/json-es.html - [Última vez accedido 19/05/2020]

16. MySQL - mysql.com - http:// mysql.com - [Última vez accedido 19/05/2020]

17. "PostgreSQL: The World's Most Advanced Open Source Relational Database" - www.postgresql.org - https://www.postgresql.org/ - [Última vez accedido 19/05/2020]

18. Psycopg - psycopg.org - https://www.psycopg.org/ - [Última vez accedido 19/05/2020]

19. PythonAnywhere - pythonanywhere.com - https://www.pythonanywhere.com/ - [Última vez accedido 19/05/2020]

20. Python - python.org - https://www.python.org/ - [Última vez accedido 19/05/2020]

21. Select2 - select2.org - https://select2.org/dropdown - [Última vez accedido 19/05/2020]

22. Stackoverflow - s.stackoverflow.com - https://es.stackoverflow.com/ - [Última vez accedido 19/05/2020]

23. SB Admin - startbootstrap.com - https://startbootstrap.com/templates/sb-admin/ - [Última vez accedido 19/05/2020]

24. Sublime Text - sublimetext.com - https://www.sublimetext.com/ - [Última vez accedido 15/01/2020]

25. Virtual environment - wiki.archlinux.org - https://wiki.archlinux.org/index.php/Python_(Espa%C3%B1ol)/Virtual_environment_(Espa%C3%B1ol) - [Última vez accedido 19/05/2020]