UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO CIENCIAS E INGENIERIAS

Proceso de reingeniería para el desarrollo de un software administrativo contable web usando una base de datos multivalor y servicios RESTful

Proyecto de investigación

Cristhian Vinicio Mejía Pérez

Ingeniería en Sistemas

Trabajo de titulación presentado como requisito
para la obtención del título de
Ingeniero en Sistemas

Quito, 14 de mayo de 2019

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO CIENCIAS E INGENIERIAS

HOJA DE CALIFICACIÓN
DE TRABAJO DE TITULACIÓN

Proceso de reingeniería para el desarrollo de un software administrativo contable web usando una base de datos multivalor y servicios RESTful

Cristhian Vinicio Mejía Pérez

Calificación: _____

Nombre del profesor, Título académico: Daniel Riofrío, Doctor of Philosophy in Computer Science

Firma del profesor: _____

Quito, 14 de mayo de 2019

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante:	_____
Nombres y apellidos:	Cristhian Vinicio Mejía Pérez
Código:	00127749
Cédula de Identidad:	1725251290
Lugar y fecha:	Quito, 14 de mayo de 2019

RESUMEN

El software administrativo contable SIP, de la empresa Admindysad Cía, Ltda., necesita una actualización tecnológica y se propone el desarrollo de una aplicación web. El requerimiento principal es mantener la misma base de datos multivalor Rocket D3, utilizada en SIP, e incluirla en la aplicación propuesta, SIPweb, utilizando API RESTful. Para realizar una implementación adecuada de la nueva tecnología, se realiza un proceso de reingeniería sobre SIP que involucra procesos de reconstrucción, transformación, refinación e implementación. Para actualizar la tecnología heredada a un contexto web es necesaria una reconstrucción de la arquitectura y su adaptación a un patrón de diseño Modelo-Vista-Controlador. Utilizando casos de uso y diagrama de clases se diseña la funcionalidad de un prototipo de gestión de proyectos que permita validar la nueva arquitectura con un sistema de autorización y autenticación de usuarios mediante los roles de Tesorero y Director Financiero. La implementación del prototipo se realiza en el framework Django 2.1 con Python 3.7 que a su vez maneja convenientemente un patrón de diseño equivalente a MVC, denominado Modelo-Vista-Template. Se establece una arquitectura del prototipo en base a las características de Django, y se incluye la conectividad con API RESTful que permite la conexión con la base de datos D3. Implementando vistas de formularios, catálogos, detalles y reportes, se llega a un prototipo que demuestra con eficiencia el funcionamiento de la arquitectura a nivel de desarrollo y su interactividad con D3, con el módulo de autenticación de usuarios y gestión de proyectos. El trabajo futuro incluye el desarrollo completo de la aplicación SIPweb y un enfoque a nivel visual.

Palabras clave: *Rocket D3, sistema operativo Pick, multivalor, base de datos no relacional, desarrollo web, API RESTful, Django, Python, patrón de diseño MVC, software administrativo contable.*

ABSTRACT

The business accounting software SIP by Admindysad Cía. Ltda. needs a technologic update and the development of a web application is proposed. The main requirement is to keep the same Rocket D3 multivalue database, already used in SIP, and include it on the proposed application, SIPweb, using RESTful API. To achieve an adequate implementation of the new technology, a reengineering process is needed over the SIP software that involves processes of reconstruction, transformation, refining and implementation. In order to update the legacy technology into a web context, it is needed a reconstruction of the architecture and its adaptation into a Model-View-Controller design pattern. Through use cases and class diagrams, the functionality of a project management prototype is designed so it can validate the reengineered architecture with an authentication and authorization system for users with Treasurer and CFO roles. The implementation of the prototype is developed with the Django 2.1 framework and Python 3.7 that conveniently uses a design pattern like MVC, called Model-View-Template. According to the Django functionality, a prototype architecture is established, and a RESTful API connectivity is included for the D3 database connection. Employing forms, catalogues, detail and report views, a prototype that shows efficiently the functionality at a development level and an interactivity with D3 is achieved, with the corresponding module for user authentication and project management. Future work includes the complete development of the SIPweb application and a more prioritized advance in the visual layer.

Keywords: Rocket D3, Pick operating system, multivalue, non-relational database, web development, RESTful API, Django, Python, MVC design pattern, business accounting software.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Descripción del problema

En los últimos años, las expectativas de las empresas para procesos de contabilidad y administración han ido creciendo en función de la evolución tecnológica emergente. Con el auge de aplicaciones web es natural que una organización las prefiera en relación a un programa de escritorio. Bajo estos términos, se reconoce la necesidad de actualización tecnológica del software administrativo contable SIP que la empresa de desarrollo de software Admindysad Cía. Ltda. ha comercializado en la última década.

Enfocándose en el desarrollo de software personalizado en función de las necesidades de cada cliente, SIP (Sistema Integrado Premium) se destaca por el uso de la base de datos no relacional, multidimensional y multivalor Rocket D3 para manejar sus procesos de back-end, beneficiándose de las extensas características de este tipo de base de datos.

Admindysad ha examinado opciones de procesos de desarrollo web para varios de sus clientes con el objetivo de acoplar nuevas tecnologías y reinventar su producto, manteniendo la confiabilidad que los identifica y renovando su imagen.

Se propone un proceso de reingeniería completo que incluye la etapa de reconstrucción para analizar la arquitectura original y la tecnología heredada, y para recolectar o generar toda la documentación que describa el software a nivel funcional. La etapa de transformación involucra la redefinición de la arquitectura para adoptar un nuevo patrón de diseño MVC (Modelo-Vista-Controlador) y el análisis de la nueva tecnología que lo pueda afiliar. Posteriormente se revisa y corrige cualquier documentación necesaria que se adapte a la nueva arquitectura. Sin embargo, una vez que la fase teórica está completa se requiere una etapa de implementación de una sección del software completo para asesorar la validez del modelo propuesto.

Para el proceso de implementación, se propone el desarrollo de un módulo de gestión de proyectos que pueda explotar la arquitectura renovada bajo el mismo sistema de gestión de base de datos Rocket D3. La compatibilidad entre esta base de datos y servicios RESTful requiere que se creen recursos web, llamados API RESTful, para manejar la comunicación con la base de datos. Además, se utilizará Python 3.7, ya que es una herramienta emergente para desarrollo web, cuya inclusión con D3 hace del prototipo propuesto una aplicación innovadora en el mercado. Con respecto a la programación front-end, se usará el framework Django por su relación con Python, y debido a que permite un desarrollo eficiente, rápido y pragmático, y así promover un diseño moderno e intuitivo.

Se puede notar la complejidad a nivel de diseño de sistemas que requiere un proyecto esquematizado de reingeniería bajo el nivel funcional descrito anteriormente. La importancia está definida bajo los requerimientos y necesidades de la empresa y patrocinador Admindysad Cía. Ltda., que, al necesitar un producto renovado, solicitan un software y un rediseño que les permita recuperar competitividad en el mercado.

Objetivo General

Rediseñar el software administrativo contable de la empresa Admindysad, SIP (Sistema Integrado Premium), manteniendo el modelo de base de datos existente, a una plataforma web.

Objetivos Específicos

- Analizar el diseño y la implementación de la aplicación actual.
- Analizar el gestor de base de datos Rocket D3 en el que se encuentra implementada la aplicación SIP.
- Analizar las tecnologías web back-end Python con servicios RESTful y front-end con el framework Django.
- Diseñar la nueva aplicación web, SIPweb, bajo el patrón de diseño MVC.

- Diseñar los módulos de la nueva aplicación web considerando un bajo acoplamiento y alta cohesión.
- Documentar el modelo de la nueva aplicación utilizando casos de uso y diagrama de clases.
- Implementar un prototipo del módulo de gestión de proyectos considerando los objetivos de diseño.

Organización del Documento

Se organiza el documento de manera que las distintas fases de reingeniería propuestas se acoplen a un proceso de desarrollo de software mediante un método en cascada. Se inicia con un marco teórico que distingue el estado del arte de la tecnología que forma parte del presente proyecto y se continúa con el proceso de desarrollo. La fase de análisis constituye la observación de los requerimientos, el proceso de reconstrucción de la arquitectura actual y demostración de la documentación importante. La etapa de diseño está conformada por la remodelación a nivel teórico de la nueva aplicación web. Para la fase de implementación, se incluye la fase de desarrollo de los respectivos módulos del software, complementado con una sección de pruebas. Al final se encuentra una descripción de recomendaciones, trabajo futuro y conclusiones del proyecto.

ANTECEDENTES HISTÓRICOS

Estado del arte

Durante la década de 1960, el mundo de la computación tiene un auge introducido por la aparición de circuitos integrados, multiprogramación y entornos de tiempo real. El sistema operativo UNIX se empieza a desarrollar para optimizar estas características y convertirse en un producto innovador y conveniente para la época (Ritchie & Thompson, 1974). Simultáneamente, en el campo de almacenamiento de datos, se realizan muchos cambios para reemplazar las cintas magnéticas por el uso de discos (Christensen, 1993). Al tener la capacidad de manejar estructuras y jerarquías fácilmente dentro de un disco, los programadores empiezan a diseñar el concepto de bases de datos bajo un modelo relacional (Codd, 1970). La popularización de este concepto provoca que el enfoque a nivel comercial esté manipulado principalmente por bases de datos de esta categoría (Bachman, 1973).

Más allá de generar un modelo de gestión de datos intuitivo, era crítico optimizar el espacio de almacenamiento. A pesar de que el uso de discos se volvió un estándar después de 1970 (Christensen, 1993), el costo por megabyte aún era un lujo que no todas las empresas tenían acceso, por lo que existía una discusión bastante amplia en función de la economía de recursos de procesamiento de datos (Phister, 1976). En la Figura 1 se observa cómo ha ido disminuyendo el costo por megabyte en distintas tecnologías de almacenamiento, y se puede observar claramente que los precios disminuyen en cuestión de orden de magnitud. Como referencia, en la década de 1970, el costo por megabyte en almacenamiento en discos duros podía llegar hasta $2550, en comparación al 2019, en el que el costo por megabyte es de $0.0000187, una diferencia de ocho órdenes de magnitud (McCallum, 2019).

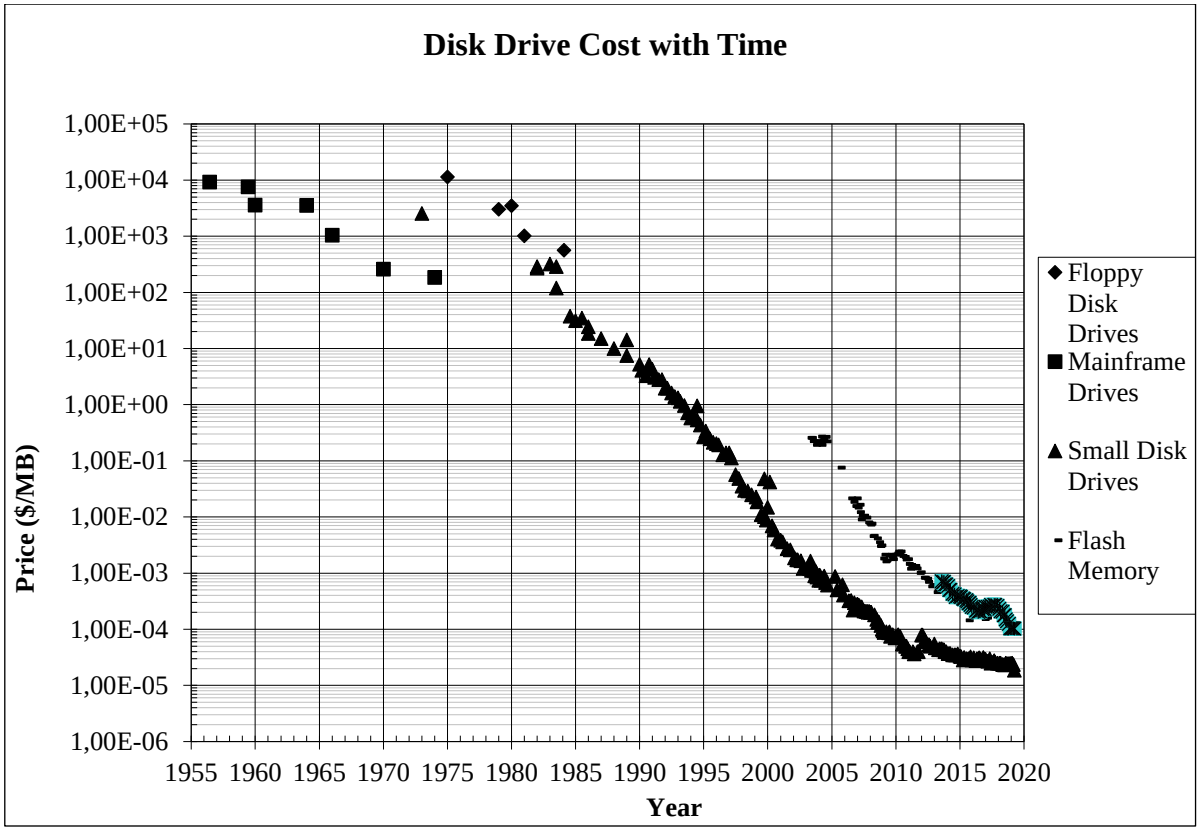

Figura 1. Precio de almacenamiento en disco por MB a través del tiempo (McCallum, 2019).

Bajo estas condiciones, es justificable entender que los modelos de bases de datos se vuelvan relacionales en lugar de almacenar datos no relacionados, e intenten implementar medidas que permitan garantizar que se utilice la menor cantidad de almacenamiento posible. Durante las siguientes décadas distintas empresas desarrollaron aplicaciones de base de datos implementando un esquema relacional debido a la necesidad de ahorrar espacio en disco, como por ejemplo la base de datos DB2 de IBM, sacada al mercado a inicios de los 80 (Saracco & Haderle, 2013).

Eventualmente, llega un punto a finales del siglo XX en el que el costo por megabyte disminuye tanto que no existe la necesidad de optimizar el uso de recursos de almacenamiento y se empiezan a desarrollar nuevos esquemas de bases de datos ajenos al modelo relacional que se popularizó y perfeccionó en los 30 años anteriores, con varias ventajas y desventajas (Kaur, Kaur, & Kaur, 2013). Años después surge el concepto de Big Data que, mezclado con un rápido

avance de tecnología web, requiere un manejo más eficiente de información que ingresa en enormes cantidades y más continuamente que información tradicional (Miloslavskaya & Tolstoy, 2016) (Grolinger, Higashino, Tiwari, & Capretz, 2013).

Una solución propuesta en relación con la gestión de Big Data es la integración del concepto de data lake que se define como un repositorio de almacenamiento masivamente escalable en arquitectura completamente plana sin jerarquía. Todos los datos que ingresan a un data lake se encuentran en un formato nativo hasta que son necesitados y capturados por sistemas de procesamiento que no alteren la estructura bruta de los datos. A diferencia de las bases de datos con sistemas analíticos estáticos, un data lake implementa un modelo dinámico que hace uso de la semántica contextual de los datos para realizar funciones de análisis. Debido a la rapidez de procesamiento y a las capacidades de almacenamiento sin incurrir en altos costos, este esquema permite redundancia de datos de forma eficiente (Miloslavskaya & Tolstoy, 2016).

Por otro lado, en el sector de la tecnología web creciente, uno de los conceptos que emergen a inicios de siglo es la Transferencia de Estado Representacional, o REST por sus siglas en inglés. Este concepto es un estilo de arquitectura que se basa en distribuir hipermedia en función de estados que representan algún contenido en específico, y pueden ser accedidos por identificadores de recursos únicos y representados en varios formatos legibles (Fielding, 2000). Más allá de contribuir a una arquitectura básica para la implementación de protocolos web, REST permite también diseñar un enlace entre modelos de datos y aplicaciones cuyo único canal de comunicación son los recursos web, o RESTful API, y no necesitan de almacenamiento local para la base de datos (Kumar, 2018).

Con tantas propuestas innovadoras emergentes en el mercado tecnológico, es difícil mantener un registro de todas las tecnologías existentes. En la década de 1965 apareció el sistema operativo Pick que manejaba varios de los conceptos no relacionales observados

décadas después, pero no tuvo tanta apertura popular dentro del ecosistema de gestión de bases de datos como su contraparte relacional, a pesar de tener muchas otras ventajas completamente innovadoras para la época.

Sistema operativo Pick

Historia

Alrededor de 1965, Richard "Dick" Pick y Don Nelson desarrollan un sistema de gestión de base de datos escrito en lenguaje de ensamblador y denominado GIRLS (Generalized Information Retrieval Language System) con el objetivo de comercializarlo al gobierno de Estados Unidos para su uso en la Armada. Años después, se lanza al mercado bajo el nombre Pick Operating System (o simplemente Pick) y durante las siguientes dos décadas, el sistema es reimplementado decenas de veces, inicialmente en minicomputadoras y manteniendo su gran portabilidad a lo largo de los años (Girap, 2018).

La propuesta inicial del sistema Pick era tener un lenguaje de recuperación similar al inglés que se pueda utilizar en un computador sin especificar, definiéndose entonces como un sistema operativo de memoria virtual y multiusuario. Alrededor de dos décadas después, el sistema evolucionado adquirió una complejidad tan grande que fue comparado, en sofisticación, con el sistema UNIX. La diferencia principal es que Pick era mucho más sencillo de usar y se enfocaba en la creación de aplicaciones, en contraste a UNIX, que se enfocaba en la programación de sistemas. Esta disimilitud hace que Pick resalte en contextos empresariales y de manejo de datos transaccionales (Cook & Brandon, 1984).

A pesar de las evidentes ventajas que el sistema operativo Pick demuestra, ha gozado de muy poco reconocimiento público durante su tiempo de vida. Además del poco esfuerzo en marketing, la causa principalmente se asocia a que la tendencia en bases de datos estaba en modelos relacionales, y el sistema de base de datos de Pick no se define bajo este concepto

(Pick M. , 2008). Razones adicionales implican la dificultad de operar bajo el sistema operativo en sí. Si bien la creación de aplicaciones es sencilla, éstas deben regirse estrictamente a la estructura del sistema sin opción a modificarlo. Pick, además, no tuvo un incremento tecnológico notable en comparación a sistemas operativos más avanzados (Cook & Brandon, 1984). No obstante, la lista de productos que han implementado el sistema operativo Pick a través de las últimas décadas no ha disminuido, y se ha popularizado en ciertos sectores en donde la flexibilidad de la gestión de datos es prioritaria (Rocket Software, Inc.).

Características

Richard Pick, en un reporte de 1987 define a su sistema operativo como un producto fácil de usar, multiusuario, con memoria virtual. En paralelo con UNIX, Pick describe un sistema multiusuario caracterizado por permitir múltiples usuarios compartir los mismos recursos dentro de un sistema (Wong & Seltzer, 1999). Esto se evidencia como clara ventaja al implementar un sistema de bases de datos en el que se puede distinguir distintos usuarios para acceder a las mismas cuentas en el sistema.

Pick, además, promociona un gestor de memoria virtual que permite movimiento dinámico de datos y programas dependiendo de las necesidades. Existe un sistema de diccionarios que manejan toda la información de la jerarquía de archivos existente en el sistema, además de información de cómo presentar estos datos al usuario final. Esto, sumado al gestor de memoria virtual, genera una ventaja significativa en velocidad y facilidad de uso (Pick R. , 1987). Cada archivo está compuesto de varios campos, o atributos, y cada instancia de un archivo se llama ítem, o récord. El diccionario de archivos, o definición de archivos, guarda información de cada uno internamente para buscarlos con rapidez en el futuro. Encima del sistema de archivos eficiente de Pick, el sistema operativo guarda un identificador único para cada uno de sus elementos, por lo que utiliza hashing para añadir rapidez a las búsquedas (Sisk, 2000). Es interesante resaltar que el concepto de utilizar hashing para cada elemento en

el sistema de archivos es similar al análisis contextual que un data lake implementa, exceptuando la jerarquización de archivos como se describió en secciones anteriores.

El gestor de archivos maneja una estructura que Pick llama matemáticamente tridimensional, y se relaciona al concepto de multivalor. Multivalor se define como la propiedad de un campo para poder almacenar distintos valores sin la necesidad de requerir un espacio de almacenamiento para cada uno, los guarda en una misma cadena ahorrando espacio, y solo los divide al momento de requerir visualizarlos mediante un carácter separador (Pick R. , 1987).

Uno de los aspectos que más cabe destacar del sistema operativo Pick es su eficiente uso del lenguaje BASIC, adaptado al sistema operativo y renombrado PICK/BASIC. La implementación de este lenguaje para realizar operaciones rápidas de bajo nivel con un lenguaje simple permite que el desarrollo de programas que hagan uso de la base de datos se vuelva bastante óptimo. El sistema de archivos es el mismo tanto para los programadores, como para usuarios, e incluso para el sistema operativo, haciendo que las operaciones programadas en BASIC accedan directamente a éste, sin capas ocultas de información (Sisk, 2000).

Toda aplicación desarrollada para el sistema operativo Pick requiere de implementar un patrón de diseño en el que toda la aplicación esté montada sobre el mismo ambiente y exista comunicación directa entre todos los módulos. Para implementar este concepto, Pick utiliza un Lenguaje de Control de Terminal, TCL por sus siglas en inglés, y así poder acceder, desde cualquier punto, al sistema de archivos compartido de la base de datos, siempre que se respete la estructura de usuarios y cuentas (Pick R. , 1987).

Con estas características siendo las principales y más relevantes para el desarrollo del presente trabajo de titulación, se demuestra la utilidad de un sistema como Pick, que se

desarrolló inicialmente para no depender de hardware en la década de 1960, y sus conceptos se han mantenido estables y robustos hasta el día de hoy.

Gestor de base de datos Rocket D3

Una de las implementaciones más conocidas del sistema operativo Pick es D3, un producto de base de datos que porta el sistema operativo Pick en otros sistemas operativos host, como UNIX, Linux o Windows. Comercializado a través de los últimos 30 años por varias empresas, empezando por Pick Systems, actualmente D3 es licenciado y distribuido por Rocket Software bajo el nombre Rocket D3 (Girap, 2018). Rocket D3 ahora es parte del amplio portafolio de productos que ofrece Rocket Software, junto con UniData, UniVerse y mvBase, que también manejan conceptos del sistema operativo Pick para gestión de base de datos multivalor (Rocket Software, Inc.).

Rocket D3 hace uso de todas las ventajas del sistema operativo Pick que se trataron en la sección anterior. Implementa el mismo sistema de archivos para manejar la base de datos. El valor agregado, sin embargo, se relaciona a la gama de productos adicionales y secundarios que complementan a la base de datos. A través de los años, Rocket, y varias de sus predecesoras, han desarrollado varios estándares de ODBC (Conectividad abierta de base de datos) y complementos que se relacionan a la arquitectura REST para su uso en aplicaciones web mediante la programación de API para hacer de D3 una base de datos implementada por cientos de empresas alrededor del mundo (TigerLogic Corporation, 2007).

DESARROLLO DEL TEMA

Formulación del problema

La empresa Admindysad Cía. Ltda. requiere de una plataforma web que permita desempeñar las mismas funciones que el software SIP y que utilice el gestor de base de datos D3 ya existente. Sin embargo, el cambio de tecnología en front-end no implica solamente una migración de procesos a una plataforma compatible, sino el desarrollo completo de una arquitectura renovada. Se debe aplicar un proceso de reingeniería para poder determinar el mejor diseño que pueda implementar los requerimientos específicos de una manera escalable. Admindysad no presenta requerimientos específicos relacionados a la tecnología a utilizar, por lo que se realizará un análisis sobre una tecnología actual que será descrita más adelante.

Desarrollo del prototipo

Análisis de requerimientos

Actualmente, el software SIP funciona bajo el sistema operativo Pick, bajo el cual desempeña los roles de front-end y back-end con las funcionalidades propias del sistema operativo. Se accede a la aplicación mediante un cliente que se vincula a un servidor remoto mediante conexión telnet. El usuario elige una cuenta específica a la cual acceder y finalmente, mediante un sistema de autenticación básico con nombre de usuario y contraseña, ingresa a la plataforma de escritorio. Ésta, a su vez, tiene una conexión directa con la base de datos D3 asignada a la cuenta detallada.

Las vistas que se muestran al iniciar sesión dependen de la cuenta y del nivel de autorización y permisos que se le hayan otorgado al usuario. Con una interfaz sencilla de menús de texto, el usuario puede acceder a todas las opciones autorizadas, tal como se puede apreciar en la Figura 2. Cada menú está asignado a un módulo del sistema respectivo, y cada módulo contiene los procesos necesarios para la funcionalidad completa del módulo. Se puede dividir

a todos los procesos en las categorías de creación/modificación, consulta de catálogos, generación de reportes. Los procesos de creación/modificación se muestran como un formulario secuencial en donde se ingresan datos que son validados inmediatamente y al final se guardan o actualizan directamente en un archivo en la base de datos, ejemplificado en la Figura 3. La consulta de catálogos se presenta como una tabla con los valores respectivos del objeto perteneciente al Catálogo, también se puede filtrar la consulta. La generación de reportes se efectúa de acuerdo con la recopilación de datos requeridos y la presentación en un formato que le sirva al Usuario para imprimir o detallar a nivel de auditoría. Se puede apreciar que, en muchos casos, los procesos no son completamente intuitivos al usuario, lo cual implica un reto para la nueva aplicación.

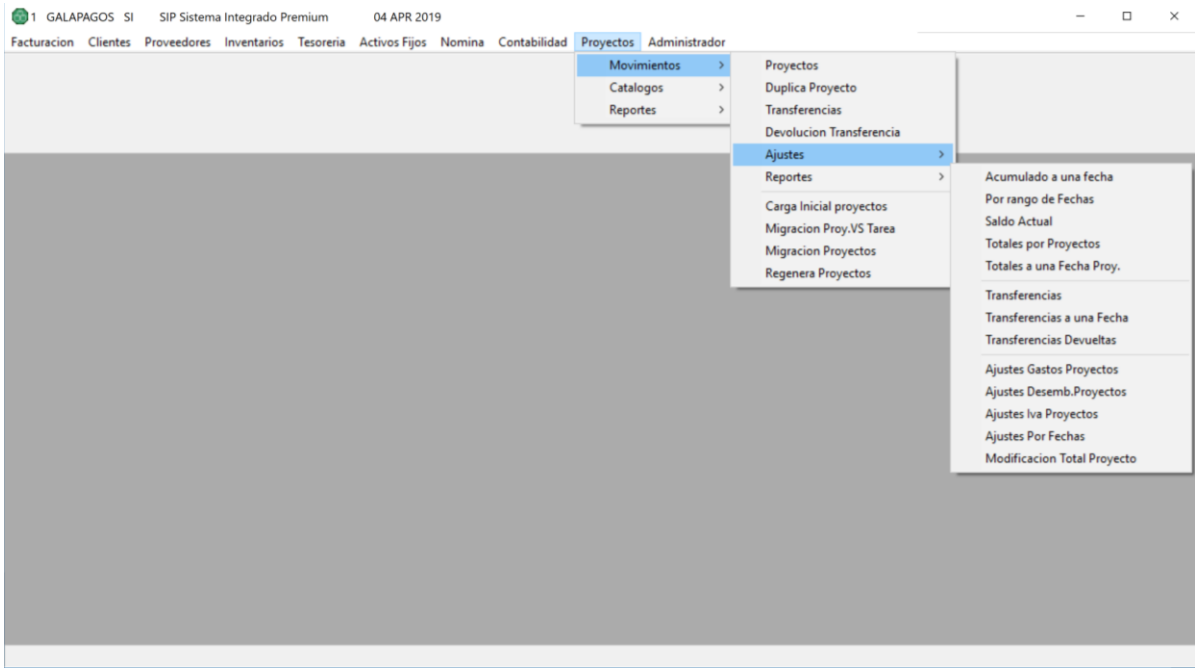

Figura 2. Interfaz gráfica del software SIP

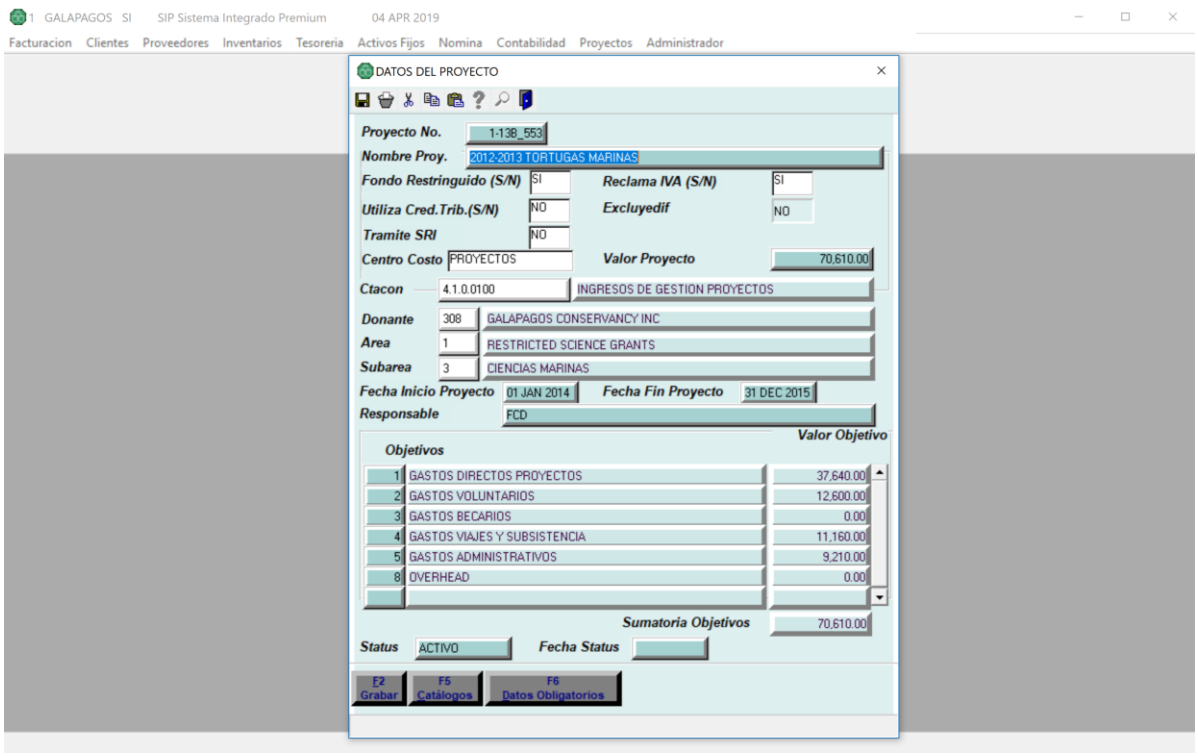

Figura 3. Interfaz gráfica de creación y modificación de un Proyecto en el software SIP

Para la implementación del sistema en una plataforma web, denominado SIPweb, se pretende realizar los procesos de manera que mejoren la experiencia de usuario, y mantengan la funcionalidad de los distintos módulos. Se requiere, además, que se conserve el sistema de base de datos D3 para no involucrar migración de datos como parte del proceso. Para demostrar la funcionalidad de la arquitectura a diseñar se requiere de un prototipo que demuestre el ciclo de creación, gestión, consulta de catálogos y reporte de un proyecto. Proponemos, entonces, el análisis en función de los siguientes procesos que constituyen la secuencia mencionada:

- Creación y modificación de un proyecto.
- Registro de pagos a proveedores.
- Consulta de catálogos de áreas, bancos, donantes, proveedores, proyectos, objetivos y tareas.
- Gestión de catálogos de áreas, objetivos y tareas.

- Reporte del estado de un proyecto.

Se realiza una documentación de los casos de uso mencionados para poder determinar la funcionalidad completa que debe cumplir el prototipo. En la Figura 4 se demuestra un diagrama de los casos de uso relevantes.

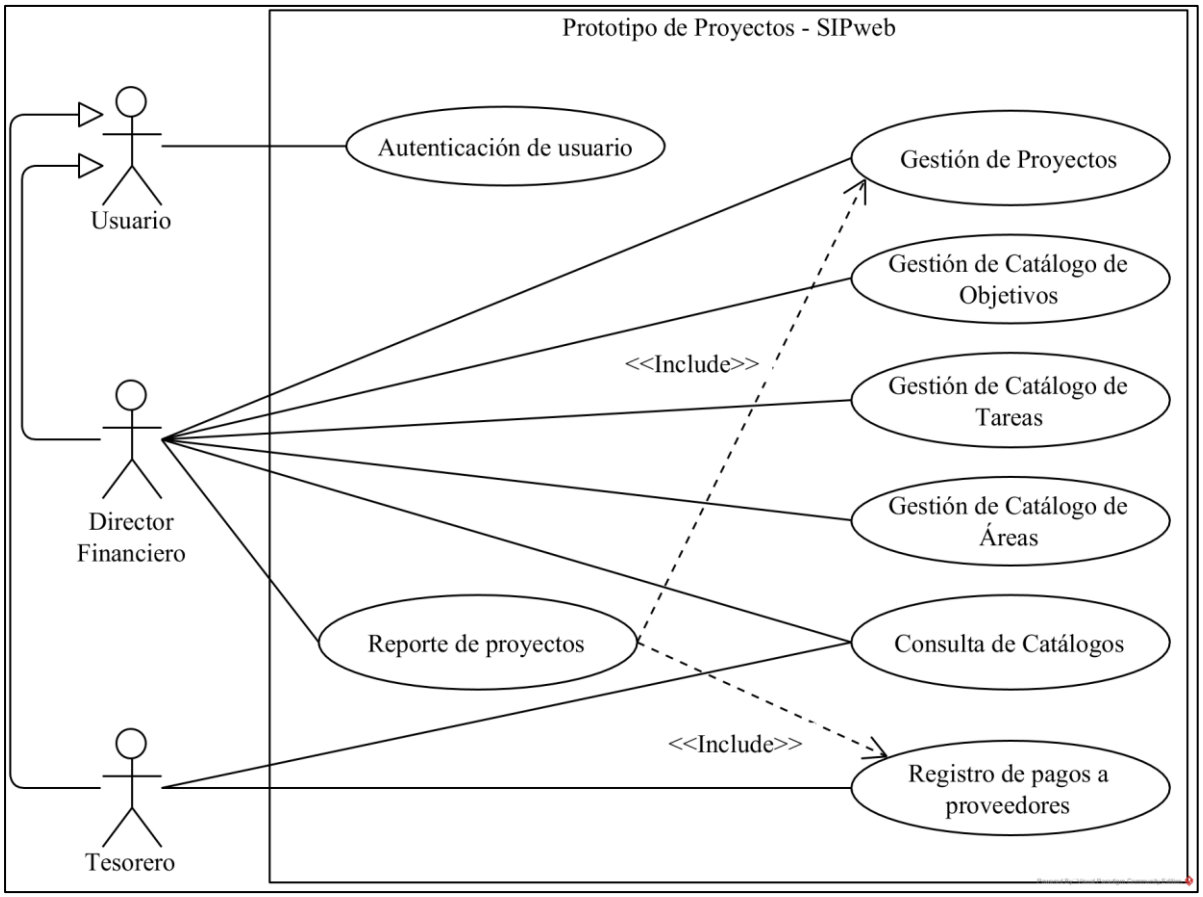

Figura 4. Diagrama de casos de uso para prototipo de gestión de proyectos de SIPweb

Actores

Se define un número de actores que puedan desempeñar los roles principales de uso del sistema, enfocados en los procesos que contendrá el prototipo. Generalmente, cada actor está asociado con un grupo de usuarios específicos, sin embargo, en casos reales, pueden existir usuarios que cumplan el rol de más de un actor simultáneamente. Esta funcionalidad se la describe bajo un sistema de autorización. A continuación, se listan los actores importantes para la implementación del prototipo.

- Usuario: Actor que puede acceder al Sistema y tiene un nivel de autorización específico dentro del mismo.
- Director Financiero: Creador de proyectos, encargado de la inclusión, modificación y mantenimiento de las características de un proyecto durante su ciclo de vida. Tiene acceso a los procesos de reporte de proyectos y a los distintos catálogos.
- Tesorero: Reporta los pagos y administra recursos económicos de proyecto.

Prerrequisitos generales

De la misma forma, se definen prerrequisitos generales que deben cumplirse previo a la implementación de los casos de uso que se describirán más adelante:

- El Usuario debe identificarse mediante el sistema de autenticación previo al ingreso a las funcionalidades del Sistema.
- Cada Usuario tiene acceso solamente a las funcionalidades definidas en su rol respectivo y no puede interferir con los procesos que no pertenezcan a su rol.

Casos de uso

Autenticación de usuario

Se utiliza un sistema simple pero necesario para permitir el ingreso de un Usuario a la aplicación.

1. El Sistema despliega un formulario para iniciar sesión con nombre de usuario y contraseña.
2. Una vez que el Usuario ingrese los datos, el Sistema verifica que el Usuario exista y que la contraseña coincida con el Usuario.
    - Si el Usuario no existe, el Sistema restringe el ingreso y envía un mensaje de error.
    - Si el Usuario existe, el Sistema permite el ingreso normalmente.

3. Finalmente, el Sistema reconoce al Usuario y su nivel de autorización. Despliega las funcionalidades de la aplicación dependiendo de esto.
4. El Usuario ingresa al Sistema con sus procesos designados.

Creación de un proyecto

Para la implementación del prototipo, se considerará el caso de uso de Creación de proyectos. Este módulo requiere solamente del actor Director Financiero.

1. Director Financiero identifica al proyecto mediante un sistema de codificación único de proyecto propio de la empresa.
2. Director Financiero asigna un nombre al proyecto, que corresponde a la actividad principal por los fondos que se reciben del Donante.
3. Director Financiero determina si los fondos asignados al proyecto son restringidos.
4. Director Financiero ingresa el valor del proyecto, equivalente al total que el Donante aprobó entregar para que se ejecute el proyecto.
5. Director Financiero escoge una Cuenta contable que se utilizará para el registro de cada compra.
6. Director Financiero escoge al Donante que entrega los fondos para el proyecto.
7. Director Financiero escoge el Área donde se utilizarán los fondos.
8. Director Financiero define la fecha de inicio y fecha de finalización del proyecto que consta en el presupuesto aprobado.
    - El Sistema controla que no existan inconsistencias en las fechas.
9. Director Financiero define el nombre de un Responsable que estará a cargo del proyecto.

10. Director Financiero ejecuta la opción de guardar los datos iniciales del proyecto. El Sistema valida la información y lo dirige a la pantalla de creación de Objetivos y Tareas.
11. Director Financiero especifica los Objetivos y Tareas del proyecto.
    - Director Financiero escoge un Objetivo del Catálogo.
    - Director Financiero determina todas las Tareas para cumplir el Objetivo.
12. Director Financiero registra un Banco (Cuenta bancaria) en donde se depositarán los fondos del proyecto.
13. Director Financiero realiza la acción de grabar los datos.
14. El Sistema almacena información del Usuario que registró los datos, incluyendo fecha, hora y dirección IP.

Gestión de Catálogo de Objetivos

Se requiere disponer de un acceso al catálogo de Objetivos para proyectos. Esta función incluye procesos de consulta y creación de Objetivos. Esta tarea se la asigna al Usuario con rol de Director Financiero. Un Objetivo se define como un enunciado que permite asesorar el fin para el cual se financia un proyecto específico. Cada proyecto dispone de uno o más Objetivos.

1. El Sistema despliega una lista con todos los Objetivos existentes, con varias columnas que representan los atributos del elemento.
2. Director Financiero puede escoger uno de la lista, o agregar un Objetivo nuevo.
    - Si escoge agregar uno nuevo, el Sistema permite ingresar los datos solicitados para agregar y guardar un nuevo Objetivo.
    - Si escoge un Objetivo de la lista se muestra un detalle del mismo.

Gestión de Catálogo de Tareas

Se requiere disponer de un acceso al catálogo de Tareas para proyectos. Esta función incluye procesos de consulta, creación, modificación y eliminación de Tareas. Esta labor se la asigna al Usuario con rol de Director Financiero. Una Tarea se define como un enunciado específico que se debe realizar para garantizar el cumplimiento de un Objetivo. Cada Objetivo dispone de distintas Tareas.

1. El Sistema despliega una lista con todas las Tareas existentes, con varias columnas que representan los atributos del elemento.
2. Director Financiero puede escoger una Tarea de la lista, o agregar una nueva.
    - Si escoge agregar una nueva Tarea, el Sistema permite ingresar y guardar los datos de una Tarea.
    - Si escoge una Tarea de la lista se muestra un detalle de la misma.

Gestión de Catálogo de Áreas

Se requiere disponer de un acceso al catálogo de Áreas para proyectos. Esta función incluye procesos de consulta, creación, modificación y eliminación de Áreas. Esta tarea se la asigna al Usuario con rol de Director Financiero. Un Área se define como una división que permite identificar de manera general en dónde se ubicará el desarrollo del proyecto. Un proyecto se asigna a un Área específica.

1. El Sistema despliega una lista con todas las Áreas existentes, con varias columnas que representan los atributos del elemento.
2. Director Financiero puede escoger un área de la lista, o agregar una nueva.
    - Si escoge agregar una nueva, el Sistema permite ingresar los datos necesarios para agregar y guardar una nueva Área.

- Si escoge un Área de la lista se muestra un detalle de la misma.

Consulta de Catálogos

Se requiere disponer de un acceso a todos los catálogos del sistema que complementen la creación de un proyecto. Esta función constituye solamente la acción de consulta de un elemento. Los catálogos deben estar disponibles para Director Financiero y Tesorero. Los catálogos disponibles para consulta son: áreas, bancos, donantes, proveedores, proyectos, objetivos y tareas.

1. El Sistema despliega una lista con todos los elementos existentes, con varias columnas que representan los atributos del elemento.
2. El Usuario puede escoger uno de la lista y se muestra un detalle del elemento.

Registro de pagos a proveedores

Como proceso complementario a la creación de un proyecto, se expone implementar un proceso que permita registrar pagos a proveedores que hagan uso de los recursos entregados por el Donante y tener un control de estos registros. Esta tarea se la asigna al Usuario con rol de Tesorero.

1. Tesorero identifica el proveedor al cual se dirige el pago.
    - El Sistema despliega una lista con todos los proveedores existentes.
2. Tesorero introduce número de la factura a emitir.
3. Tesorero identifica el proyecto para el cual se realizará el pago respectivo.
    - El Sistema despliega una lista con todos los proyectos existentes.
4. Tesorero especifica el valor total de la factura a pagar.
5. Tesorero especifica la fecha de emisión y fecha de vencimiento.
6. Tesorero especifica el detalle del pago.

7. El Sistema guarda la información y registra un número de ingreso secuencial.

Diseño

En función de la documentación y el análisis presentado anteriormente, se establece un diagrama de clases, mostrado en la Figura 5, que permite distinguir la relación entre los componentes incluidos en el prototipo a desarrollar. El mismo permitirá entender un esquema general del funcionamiento y podrá evolucionar a futuro para incluir la documentación de la aplicación completa. Una versión más detallada y extendida del diagrama de clases se adjunta en el CD, bajo el directorio *"Anexos\Anexo 1\Diagrama de Clases Completo.png"*.

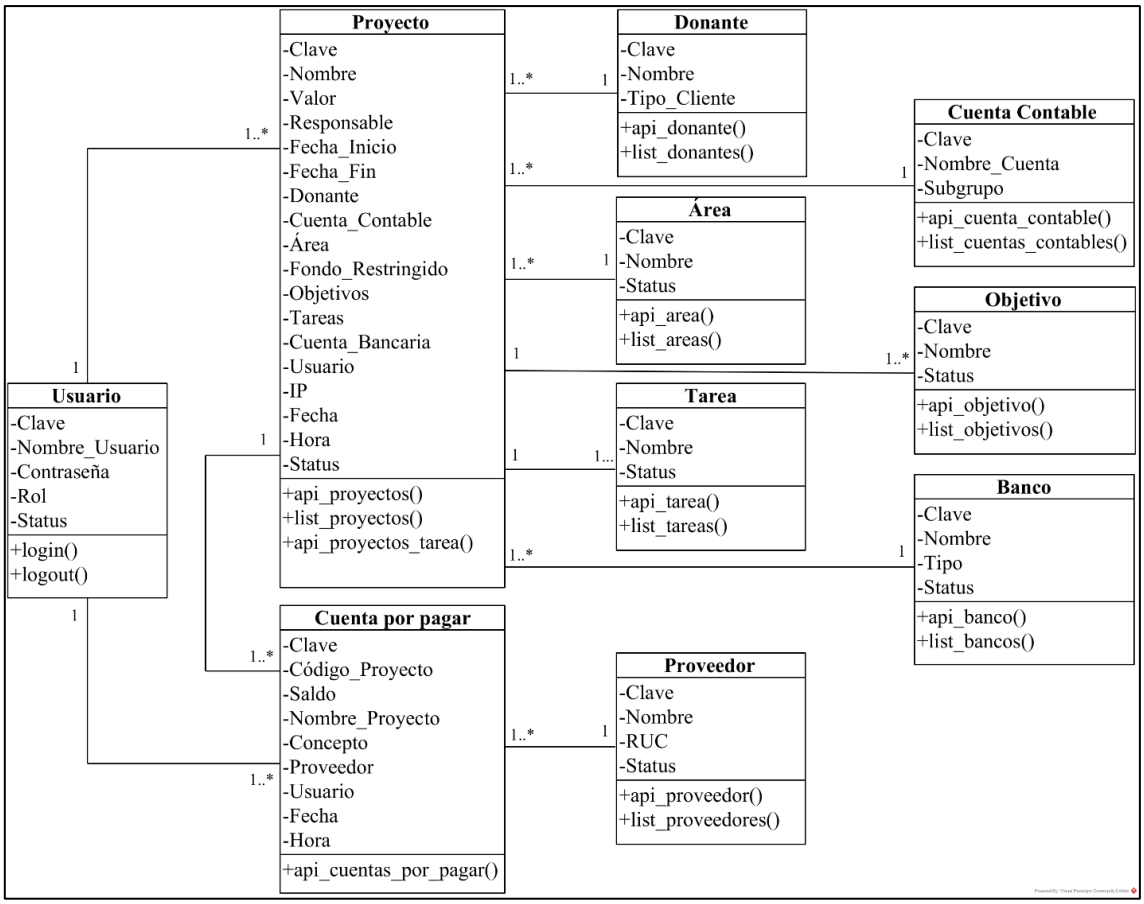

Figura 5. Diagrama de clases para prototipo de gestión de proyectos de SIPweb

La clase de Usuario está compuesta por los atributos que identifican a un Usuario dentro de la aplicación. Para propósitos de autenticación, se requiere una contraseña y un nombre de usuario, y para el sistema de autorización se añade un atributo de Rol para que se puedan

cumplir los casos de uso como se definieron en la sección de análisis. Finalmente, se incluye un atributo *status* que permite llevar un control a nivel administrativo para determinar si un Usuario sigue activo en la empresa o no. Las operaciones de esta clase se limitan a iniciar sesión (*login*), y cerrar sesión (*logout*), dentro de la aplicación.

Un Usuario con rol de Director Financiero tiene autoridad de gestión sobre la clase Proyecto. Esta clase, a su vez, requiere de la interacción con distintos elementos de otros catálogos, que se describen a la derecha de la clase Proyecto. Las clases Área, Banco, Cuenta Contable y Donante tienen una relación de uno a muchos con respecto a un Proyecto; las clases Objetivo y Tarea tienen una relación de muchos a uno. Además, se tienen diferentes atributos individuales que detallan todos los campos requeridos para cumplir los casos de uso descritos. De igual manera, una instancia de la clase Proyecto tiene atributos que permiten un monitoreo de actividad, incluyendo el Usuario, IP, Fecha y Hora de creación de un Proyecto, tal como se describió en el caso de uso Creación de un Proyecto. Las operaciones de un Proyecto se dividen en una consulta de catálogo de todos los Proyectos (*list_proyectos*), una operación con distintos atributos que permite recuperar el detalle de un proyecto específico y crear uno nuevo (*api_proyectos*), y una operación que permite recuperar las tareas para un proyecto existente (*api_proyectos_tareas*).

En tercer lugar, un Usuario con rol de Tesorero tiene la autoridad de registrar pagos a proveedores para un proyecto específico. Estos pagos se registran como una instancia de la clase Cuenta por pagar. Existe una relación de uno a muchos entre Proyecto y Cuenta por pagar. Además, existe una relación entre Cuenta por pagar y la clase Proveedor, ya que un pago se efectúa a un proveedor específico. De igual manera, una instancia de la clase Cuenta por pagar tiene atributos que permiten un monitoreo de actividad, incluyendo el Usuario, IP, Fecha y Hora de creación de una Cuenta por pagar, tal como se describió en el caso de uso Registro de pagos a proveedores. Las operaciones en este caso se limitan a la creación de una nueva cuenta

por pagar y a la consulta de todas las cuentas por pagar asignadas a un proyecto (*api_cuentas_por_pagar*).

Finalmente, como especificación para las demás clases incluidas en la Figura 5 (Área, Banco, Cuenta Contable, Donante, Objetivo, Proveedor y Tarea), existe un atributo status que define que un elemento en el catálogo no esté disponible para ser utilizado. Es un atributo implementado en la base de datos que representa una forma auditable de deshabilitar elementos sin eliminarlos.

Arquitectura del prototipo

Una vez diseñado un esquema de clases que permita observar una relación entre los elementos que se manejarán dentro del prototipo, es importante definir una arquitectura completa que sea manejada por toda la aplicación para demostrar las distintas comunicaciones, y poder definir cómo éstas se relacionan al patrón de diseño MVC. En la Figura 6 se define dicha arquitectura.

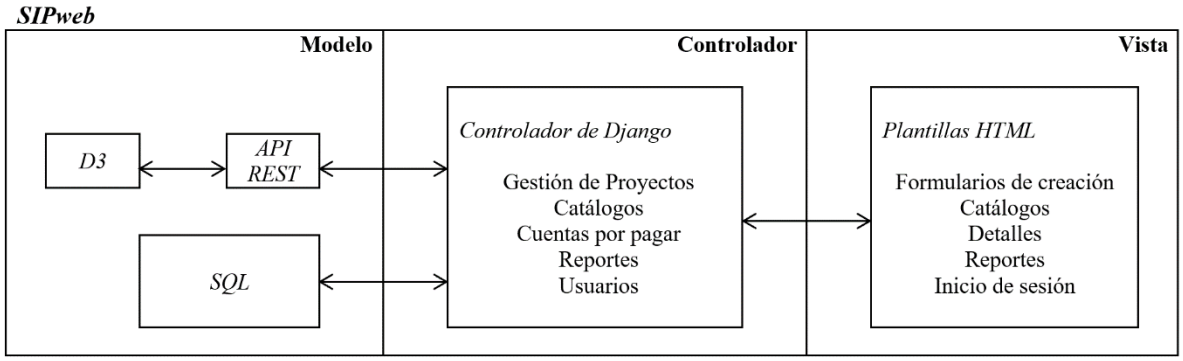

Figura 6. Diseño de la arquitectura inicial del prototipo

La arquitectura se acopla a cada uno de los tres componentes del patrón MVC. Dentro de lo que corresponde al modelo se divide en dos partes fundamentales. La base de datos D3 maneja toda la información y toda transacción realizada, como se ha especificado en los requerimientos. Ésta se comunica con el controlador solamente mediante API REST. Por otro lado, se reconoce que existen instancias en las que es preferible utilizar una base de datos

relacional, por ejemplo, para el manejo de usuarios y permisos, sobre todo debido a que no existirían intermediarios externos entre la base de datos y el controlador.

El controlador se encarga del manejo de toda comunicación con la base de datos, del procesamiento de la información y de la renderización para que ésta se muestre adecuadamente en las vistas. Esto incluye todas las llamadas generadas hacia las API, tanto de consulta como de escritura. Finalmente, el componente visual presenta toda la información recibida por el controlador al usuario final. No existe un enlace directo entre la vista y el modelo debido a que así se caracteriza el patrón de diseño MVC. En función de esta arquitectura base se puede empezar con el proceso de implementación.

Implementación

Para el desarrollo del prototipo de gestión de proyectos se utiliza Django, un framework web de alto nivel y código abierto basado en el lenguaje Python. Se adapta correctamente a los objetivos de diseño ya que maneja un patrón de diseño MVT (Modelo-Vista-Template) muy similar al propuesto MVC. La capa de Modelo se refleja idénticamente en ambos patrones, la capa de Controlador se denomina Vista en Django, y la capa de Vista se denota como Template (o Plantilla). Tomando en cuenta esto, se puede adaptar fácilmente la arquitectura descrita en la Figura 6 para una aplicación desarrollada en Django. Además, permite la inclusión de cualquier librería de Python, permitiendo una gran flexibilidad y escalabilidad a futuro. Se utilizará el IDE Pycharm de JetBrains con el fin de gestionar el proyecto en un ambiente que facilite los procesos de desarrollo, compilación y ejecución del código y de los recursos web.

Considerando que mantener una base de datos Rocket D3 existente consta en los requerimientos, Admindysad Cía. Ltda. gestionará con Rocket Software la provisión de las licencias para el correcto funcionamiento de la base de datos y de todos los procesos requeridos para una integración con servicios RESTful. Además, proveerá la instalación de las mismas

para un acceso ininterrumpido en una instalación local de máquina virtual Windows Server 2016. Finalmente, en la misma máquina virtual se instalará cualquier herramienta o respaldo de información suficiente para procesos de desarrollo y pruebas, como el software MVS Toolkit que permite gestionar la creación de API REST.

Herramientas por utilizar

En la Tabla 1 se muestra la versión de los principales elementos identificados anteriormente que fueron parte del proceso de implementación del prototipo y una breve descripción de las funcionalidades relacionadas con el proyecto.

Herramienta	Versión	Descripción
Python	3.7.0	Lenguaje de programación interpretado multiparadigma de código abierto que tiene gran apertura para fines de desarrollo web.
Django	2.1.7	Framework de desarrollo web que implementa un patrón de diseño Modelo-Vista-Template y funciona bajo el lenguaje de programación Python.
JetBrains PyCharm	2018.2.2	Ambiente de desarrollo integrado para Python con compatibilidad para una variedad de frameworks, incluyendo Django; integración de sintaxis complementaria para HTML, CSS, JavaScript y JSON; y ejecución y depuración de errores en tiempo real del servidor web.
Rocket D3	10.2.0	Base de datos multivalor correlacional que es utilizado por el software SIP para su modelo de datos y se usará para la implementación web.
Rocket MVS Toolkit	2.2.3.0122	Software que permite la creación y despliegue de distintas API REST para la conexión con la base de datos. Tiene un enlace directo entre los servicios web de Rocket D3 y la base de datos.

Tabla 1. Descripción de las herramientas utilizadas para la implementación del prototipo.

Además, en el CD adjunto, bajo el directorio *"Anexos\Anexo 2\Licencias Utilizadas.jpeg"* se incluye una tabla con un detalle específico de la transacción realizada entre Rocket y Admindysad Cía. Ltda. para cada tipo de licencias y las versiones manejadas.

Organización del proyecto en Django

Para la ejecución del prototipo de gestión de proyectos se utiliza Django, un framework web de alto nivel y código abierto basado en el lenguaje Python. Se adapta correctamente a los objetivos de diseño ya que maneja un patrón de diseño MVT (Modelo-Vista-Template) muy similar al propuesto MVC. La capa de Modelo se refleja idénticamente en ambos patrones, la capa de Controlador se denomina Vista, y la capa de Vista se denota como Template (o Plantilla). Tomando en cuenta esto, se puede adaptar fácilmente la arquitectura descrita en la Figura 6 para un contexto de Django. En la figura 6 se muestra cómo se adaptan las características de Django en relación con la arquitectura de la Figura 6.

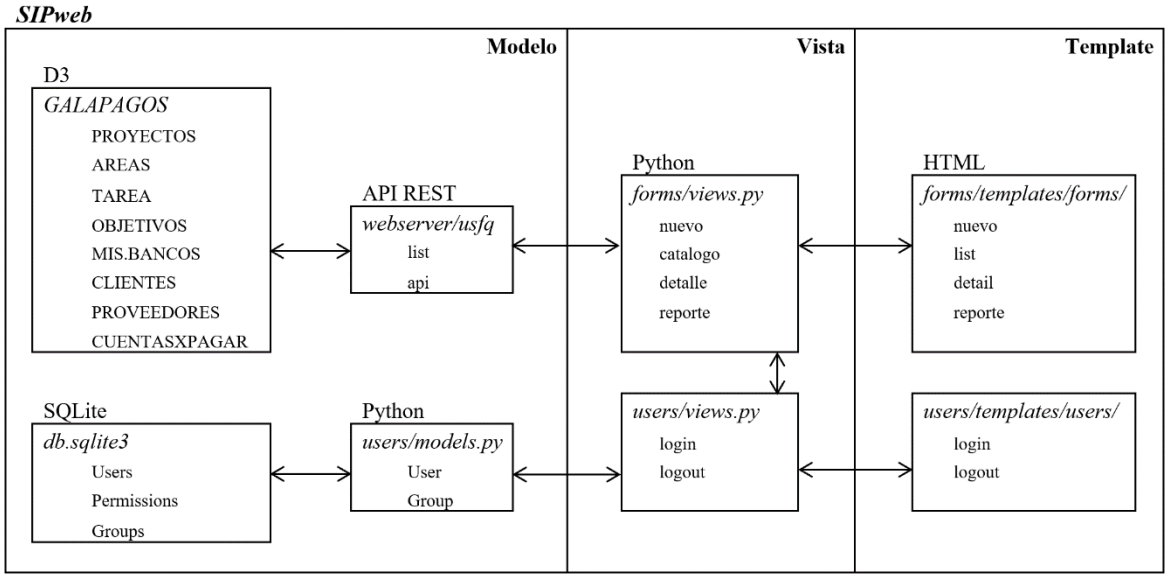

Figura 7. Arquitectura en Django para prototipo de SIPweb.

Django maneja un diseño en el que un proyecto está compuesto de distintos módulos independientes llamados *apps*, los cuales tienen su propio directorio de ficheros que Django reconoce e incluye al proyecto acorde a cómo se definan. Esto permite que el proyecto pueda escalar fácilmente sin tener que modificar o interrumpir la funcionalidad de otra *app*.

El módulo de autenticación y autorización de usuarios se maneja bajo las herramientas provistas por Django, y se enlaza a la app *users* en los directorios del proyecto. *Users* maneja el modelo de datos de Usuarios mostrado en la Figura 5 bajo una instancia de tablas de la base de datos SQLite que viene por defecto en Django, la definición de este modelo se encuentra en *users/models.py*. El fichero *users/views.py* representa el controlador, desde el cual se manipula la redirección y todos los procesos involucrados con la autenticación y autorización de un Usuario. Finalmente, el controlador se conecta a las plantillas HTML del directorio *users/templates/users* que se muestran al usuario.

Por otro lado, la *app forms* incorpora a la creación de proyectos, gestión de catálogos, registro de pagos a proveedores y creación de reportes de proyectos. En este caso, Django no maneja directamente ningún modelo de datos por sí solo. El modelo se define como la base de datos externa D3 con sus API respectivas. En el fichero *forms/views.py* se definen los procesos del controlador, que incluyen tanto la interacción con las API para conexión con el modelo de datos, como la redirección de vistas propia de Django. Similarmente a la *app users*, *forms* accede a todas sus plantillas HTML desde el directorio *forms/templates/forms*.

Una ventaja adicional de utilizar Django es su compatibilidad con las librerías de Python, que resulta especialmente útil para el caso de uso de reporte de proyectos, pues permite generar informes de manera gráfica con librerías de Python como Matplotlib, agregando valor al prototipo implementado.

Base de datos D3

Otro recurso que fue otorgado por Admindysad para el desarrollo del prototipo es el respaldo de datos de una cuenta ya existente desde la cual se analizará el modelo de base de datos. El prototipo estará directamente influenciado por éste y las diferentes definiciones de conceptos se basan en las manejadas en esta cuenta. La cuenta pertenece a la Fundación Charles

Darwin, y está poblada con datos reales desde el 2014, por lo que permite validar de mejor manera la implementación en un caso real.

Sin embargo, los campos usados en la implementación no están correctamente documentados en el diseño de los archivos de la base de datos, encontrando atributos sin utilizar, repetidos o innecesarios para la aplicación. Las características de la base de datos D3 permiten que estas fallas de diseño no afecten al producto final o a la rapidez de transacciones dentro de la base de datos, pero sí genera limitaciones que evitan escalabilidad a largo plazo. Sobre todo, en lo que respecta al diseño del modelo a utilizar en el prototipo de esta aplicación, se tuvo que consultar con expertos para definir las tablas y campos a utilizar, ralentizando el proceso de desarrollo. Esto presenta retos que se incluyen en la sección de recomendaciones y trabajo futuro.

Para ejemplificar el tipo de archivos utilizados de la base de datos, en la Tabla 2 se muestra la definición del archivo *OBJETIVOS* con cada uno de sus campos, y se muestra una columna adicional para mostrar aquellos usados en el prototipo para contrastar con el contenido completo del archivo.

Nombre del campo	**Descripción**	**Posición**	**Tipo**	**Tamaño**	**Utilizado**
CLAVE	Clave	0	A	7	Si
NOMBRE	Nombre del Objetivo	1	A	30	Si
TAREAS	Tareas	3.M	N	2	No
FECHA	Fecha	4	D	11	No
STATUS	Status	4	A	15	No
STATUS1	Status	4	A	1	Si
HORA	Hora	5	A	8	No
USUARIO	Usuario	6	A	10	No
PUERTO	Puerto	7	N	2	No

Tabla 2. Definición de archivo OBJETIVOS en D3.

Como se puede apreciar, tres de los nueve campos de la definición de archivos son utilizados para el prototipo. Asimismo, existen algunos campos que comparten posición, como los campos *FECHA*, *STATUS* y *STATUS1*, y de éstos, solo *STATUS1* tiene valores válidos. Por otro lado, el campo *TAREAS* es un campo multivalor.

API RESTful

Como parte de las herramientas otorgadas por Rocket para el desarrollo del prototipo, e instaladas por Admindysad en una máquina virtual, se encuentra MVS Toolkit, que permite gestionar recursos web, también llamados API, y probarlos en tiempo real. El software permite crear una conexión entre el servidor web de las API y la base de datos D3. Las API generan una respuesta de la base de datos en forma de archivo JSON, que puede ser leído y procesado con facilidad. El servidor web de las API está vinculado a la dirección IP de la máquina virtual que, para procesos de implementación, es 192.168.150.130 en el puerto 8181. Estas son las únicas dos variables que se alteran al momento de migrar las API a otro servidor, cumpliendo el propósito de bajo acoplamiento y alta cohesión.

De igual manera, dentro del servidor web se crea un servicio web con un identificador único. Por decisión de Admindysad, este servicio se denomina *usfq*. El URL de los recursos web utilizados para la implementación tienen, entonces, una estructura similar, y todas empiezan con el prefijo de la Figura 8.

http://192.168.150.130:8181/usfq/

Figura 8. Prefijo de las URL para los recursos web del prototipo SIPweb.

Existen dos tipos de recursos que se pueden crear en base a la información en la base de datos. La primera es la categoría *data*, que itera todos los elementos de un archivo en particular y devuelve los campos especificados. Este tipo de API es útil para la función de consulta de catálogos. Como convención, a los recursos que sirven este propósito se los reconocerá con el prefijo “list”, por ejemplo, para recuperar todos los elementos del archivo

TAREAS de la base de datos, con los campos CLAVE, NOMBRE y STATUS1, se debe llamar a la URL mostrada en la Figura 9.

http://192.168.150.130:8181/usfq/list_tareas?attributes=CLAVE+NOMBRE+STATUS1

Figura 9. URL para el API de llamada a la lista de elementos existentes del archivo TAREAS.

Un segundo tipo de recursos son de la categoría *catalogued subroutine*, que permiten ejecutar una subrutina, o proceso, dentro de la base de datos, con parámetros que pueden ser pasados en la URL. A pesar de que los parámetros dependen bastante de la subrutina a la cual se acopla el recurso web, se mantiene una convención en la que todas las API utilizadas en el prototipo tienen los mismos parámetros: tabla, tipo, user, ip, id, datos. Tabla reconoce el archivo desde el cual se saca la información en la base de datos, tipo representa la clase de operación que se va a realizar (C para consulta, A para adición, M para modificación y E para eliminación), user representa el usuario que realiza el llamado al API, ip la dirección IP desde donde se hace la petición, id es un parámetro que identifica un elemento en específico -si así fuese requerido-, y datos recopila el grupo de información que se envía para procesos de adición o modificación, aunque es un dato opcional para procesos de consulta. El principal objetivo de tener recursos web ligados a subrutinas catalogadas es poder operar sobre algún elemento específico, en lugar de iterar sobre todo el archivo. Además de hacer una petición más ligera y rápida, el procesamiento posterior es más sencillo. Para este caso, se utilizará el prefijo "api" para identificar a los recursos de esta categoría, por ejemplo, un API de consulta del elemento de AREAS con CLAVE igual a 2 tendría un URL como el de la Figura 10.

http://192.168.150.130:8181/usfq/api_area?tabla=AREAS&tipo=C&usr=&ip=&id=2&datos=

Figura 10. URL para el API de consulta del elemento con CLAVE 2 del archivo AREAS.

Como convención adicional, las subrutinas catalogadas ligadas a un recurso web tendrán el mismo nombre que el recurso. Así, el ejemplo descrito estaría ligado a una subrutina

llamada *api_area(tabla, tipo, usr, ip, id, datos)*, escrita en BASIC y almacenada dentro del sistema operativo Pick.

Finalmente, para facilitar la gestión de las URL generadas para todos los recursos que se van a utilizar en el prototipo, se utiliza un fichero JSON para almacenar tanto la porción inicial conteniendo la IP, el puerto y el servicio web, como un detalle de cada API con un identificador único. Para mantener un estándar de nombres dentro del proyecto, los recursos de categoría *data* son identificados con el prefijo "list", los recursos ligados a subrutinas y que son de tipo consulta se identifican con el prefijo "consulta" y los que son de tipo "adición" con el prefijo "add". Este método permite añadir y modificar las URL de las API de manera sencilla, sin necesidad de afectar de gran manera al código del controlador. La Tabla 3 muestra un extracto de la lista completa de las API con su respectivo identificador.

Nombre de recurso	**URL de recurso**
list_areas	list_areas?attributes=CLAVE+AREAS+STATUS1
list_donantes	list_donantes?attributes=CLAVE+TIPO.CLIENTE1+EMPRESA
consulta_proyecto	api_proyectos?tabla=PROYECTOS&tipo=C
consulta_tarea	api_tarea?tabla=TAREA&tipo=C
add_objetivo	api_objetivo?tabla=OBJETIVOS&tipo=A
add_cuenta_x_pagar	api_cuentasxpagar?tabla=CUENTASXPAGAR&tipo=A

Tabla 3. Extracto de archivo api_file.json con identificadores para las URL de recursos web.

Conexión con API desde Django

Para efectuar la conexión con los distintos recursos web desde el proyecto de Django se puede utilizar la librería *requests* nativa de Python. Esta conexión se lleva a cabo desde el controlador *forms/views.py*, pues la conexión con la base de datos D3 es exclusiva de la *app forms* en lo que respecta al prototipo. Cada función en ese fichero debe realizar todos los llamados a las API necesarias para un correcto funcionamiento de la aplicación.

Un detalle que se debe considerar al momento de hacer peticiones de este tipo es el manejo de excepciones, y en este caso, *ConnectionError* es una excepción que puede surgir de manera imprevista si el servidor web o la base de datos no se encuentran corriendo normalmente. Considerando este detalle, una implementación ejemplar en Python de una petición a un recurso llamado *api_url* con su respectivo manejo de excepciones se muestra en la Figura 11.

```
try:
    response = requests.get(api_url)
except ConnectionError:
    messages.error(request, error_message, extra_tags="alert-danger")
    return redirect('home')
data = response.json()
```

Figura 11. Implementación de llamada a un servicio web con la librería requests en Python 3.7.

Cabe resaltar que una manera aceptable de manejar la excepción sería mandar un mensaje visual de error de conexión al usuario y redirigirlo a una página que sí pueda cargar, como la página de inicio. Si es que la petición genera respuesta, la información puede ser procesada con formato JSON mediante la función *.json()* y ser manipulada con una estructura de datos simple en Python con la variable *data*. Se motiva fuertemente que se utilice la convención de código anterior para manejar excepciones y recibir los datos respectivos.

Definición de una vista

Como se mencionó anteriormente, cada vista en Django está asociada a una función en el archivo *views.py* de cada *app*. Esta función debe tomar como argumento un método de petición HTTP para que la vista actúe de diferente forma si se quiere discernir entre una petición POST y una petición GET, como en el caso de llenar un formulario, y poder adquirir información de la petición, como el usuario o la dirección IP. La función puede tomar atributos adicionales si se desea un comportamiento más dinámico. Después de realizar el procesamiento de datos respectivo en la función, ésta debe devolver una plantilla renderizada con atributos

que se constituyen como el contexto de la página. En la Figura 12 se muestra la definición de una función *detail_tarea*, usada en el prototipo, que genera una vista de detalle de un elemento Tarea.

```
def detail_tarea(request, id_tarea):
    username = request.user.username
    ip = get_client_ip(request)
    item_type = 'Tarea ' + id_tarea

    api_name = 'consulta_tarea'
    api_header = 'api_tarea'
    api_secondary = 'tarea'

    api_url = set_detail_url(api_name, username, ip, id_tarea)
    try:
        response = requests.get(api_url)
    except ConnectionError:
        messages.error(request, err_message, extra_tags="alert-danger")
        return redirect('home')
    data = response.json().get(api_header).get(api_secondary)

    context = {
        'title': item_type,
        'data': data,
        'ip': ip,
        'status': get_status(data.get('status'))
    }
    return render(request, 'forms/detail_tarea.html', context)
```

Figura 12. Implementación de la vista detail_tarea que muestra el contenido de api_tarea.

A partir del código presentado surgen varias nociones de cómo se ha propuesto el manejo de vistas. Lo que la anterior porción de código devuelve es la renderización de la plantilla *forms/detail_tarea* con un contexto compuesto por distintas variables que se decidan pasar a la plantilla. La vista descrita en la función anterior pertenece a un grupo de vistas de detalle y para promover características de bajo acoplamiento y alta cohesión se sugiere utilizar la misma convención para vistas similares en donde el único cambio son las variables iniciales de la información de la API a utilizar. Existen otros tipos de vistas que se implementan en el prototipo como la vista de creación de un nuevo elemento denotada por el prefijo *nuevo*, la vista de un catálogo denotada por el prefijo *catalogo*, y la vista de reporte denotada por el prefijo *reporte*.

Cabe mencionar que, para toda instancia de nombramiento de variables, archivos o recursos de todo tipo se utiliza la convención de nombres de variables globales de Python, es decir, el nombre en minúsculas separado por guion bajo. La única excepción se da en el nombre de clases, en donde se utiliza la convención Camel case.

Manejo de formularios

Existen algunas vistas vinculadas al manejo de formularios que el usuario debe llenar para enviar a la base de datos. Para motivos del prototipo, la única acción que requiere de formularios, por el momento, es la creación de un nuevo elemento, pero eventualmente puede evolucionar a formularios de modificación y eliminación de datos, con el respectivo soporte de las API que se generen en conjunto con subrutinas catalogadas diseñadas para tal motivo.

Django permite un modelo automático de manejo de formularios mediante clases definidas en un archivo *forms.py*. Cada *app* puede tener un grupo de clases de formularios que permitan el ingreso de campos de una forma sistemática. Una clase en el archivo *forms.py* está compuesto por un grupo de variables, llamadas campos, que pueden ser pasadas como contexto desde la vista hacia las plantillas. Estos campos, para renderizarse en las plantillas, deben heredar de la clase *django.forms,* propia del framework. Luego, se debe vincular el campo con un widget para que tenga un aspecto visual coherente con el campo, y éste debe heredar también de *django.forms*. Dentro de los parámetros del widget se pueden especificar varios parámetros que se renderizan en la plantilla HTML. Un ejemplo de un campo *detalle* se visualiza en la Figura 13.

```
detalle = forms.CharField(label="Detalle del Pago:",
                          widget=forms.TextInput(attrs={
                              'placeholder': 'Ingrese detalle del pago',
                              'class': 'form-control-sm col-sm-6'}
                              )
                          )
```

Figura 13. Implementación del campo detalle para un formulario Django en Python 3.7.

En el parámetro *attrs* se pueden incluir atributos HTML para que se incluyan cuando se renderice la plantilla. Existen muchas más funcionalidades de validación e inicialización que se pueden crear en *forms.py* para explotar todas las funcionalidades de estas clases, y se exploran opciones para recomendaciones futuras cuando sean más necesarias.

Para integrar un formulario en una vista basta con instanciar un objeto de la clase del formulario que se creó. Finalmente, para mostrarlo con facilidad en la plantilla se pasa el objeto como parte del contexto para que sea renderizado. Como medida de seguridad importante, en las plantillas se instancia una etiqueta de un token CSRF (Cross-Site Request Forgery) para complementar a cada formulario creado y así evitar falsificación de peticiones que pueden afectar gravemente a la funcionalidad de la aplicación (Barth, Jackson, & Mitchell, 2008).

Plantillas HTML

Dentro de las plantillas HTML, ya en el componente de vistas en el patrón MVC (o template en el patrón de Django), se interpretan distintas etiquetas mediante herramientas que Django provee. Existe una sintaxis que permite la inclusión dinámica de las variables de contexto. Para mostrar el contenido de una variable se la coloca dentro de dos pares de llaves, como `{{variable}}`. Además, existe sintaxis que permite crear lazos condicionales o iterativos, por ejemplo `{% if condicion %}` o `{% for element in list %}`. Operaciones adicionales permiten extender código HTML de otros archivos, o recuperar la URL específica de una vista sin tener que describir el vínculo exacto. Utilizando esta sintaxis se pueden crear plantillas que

se pueblen con datos de forma dinámica y dependan de la información desplegada en archivos del controlador.

```
{% extends "forms/base.html" %}
{% block content %}
    <div class="content-section">
        <h5>Catálogo de Bancos</h5>
        <table>
            <tbody>
                {% for banco in data %}
                    <tr>
                        <th scope="row”>{{ banco.CODIGO }}</th>
                        <td>{{ banco.BANCO }}</td>
                        <td>{{ banco.CUENTA }}</td>
                        <td>{{ banco.STATUS1 }}</td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
{% endblock content %}
```

Figura 14. Plantilla HTML que devuelve una lista de los bancos de la vista catalogo_bancos.

Un ejemplo que permite visualizar claramente la facilidad que presenta la sintaxis utilizada se puede apreciar en la Figura 14, en el código HTML que genera una tabla con la información de los bancos recuperados en la variable *data*. Sin necesidad de escribir campo por campo se genera una lista con cualquier cantidad de elementos.

Autenticación y autorización

Uno de los casos de uso más indispensables de implementar es el de autenticación y autorización de usuarios. Esta tarea presenta diversos retos de seguridad que pueden ser motivo de una investigación por sí solos. En lugar de implementar un sistema de autenticación desde cero, se utiliza una funcionalidad propia de Django que ha sido perfeccionada para presentar diversas características de seguridad, como el almacenamiento de contraseñas como código hash en lugar de texto plano.

En primer lugar, se tiene un usuario administrador que permite la gestión de usuarios sencillamente desde la interfaz de un sitio administrativo propio de Django. Un usuario

registrado tiene distintos atributos, como nombre de usuario, contraseña, e-mail, grupos, permisos, etc. Dependiendo de las necesidades del prototipo, se pueden crear más o eliminar algunos. Para el caso del prototipo actual, simplemente se necesita nombre de usuario, contraseña y permisos.

El primer paso para establecer un sistema de autenticación es determinar que los distintos elementos a nivel de plantillas y de vistas no estén disponibles para los visitantes de la aplicación que no hayan iniciado sesión. En lo que respecta a las plantillas, Django guarda la información del usuario actual en una variable *user*, y se puede saber si ha iniciado sesión con la sintaxis `{% if user.is_authenticated %}`. Para restringir acceso en las vistas, Django tiene el decorador `@login_required` que impide el acceso a la vista a menos que el usuario haya iniciado sesión. Los decoradores se colocan antes de una función en Python.

Para la implementación y pruebas posteriores, se crearán dos usuarios inicialmente, Tesorero y Financiero, quienes constituirán los actores de los casos de uso mostrados en la Figura 4. Lógicamente, cada uno poseerá permisos de acuerdo con los casos de uso que deba desempeñar. Siendo así, se atribuye un permiso *is_tesorero* al usuario Tesorero y un permiso *is_financiero* al usuario Financiero para simular cómo sería un caso real.

Una vez otorgados los permisos, se toman decisiones de cómo controlar la autorización de usuarios a las diversas características de la aplicación. Se implementa la seguridad en dos áreas importantes, dentro de las plantillas HTML y dentro de las vistas en *views.py*. Para la primera se utiliza la sintaxis vista en la sección anterior, y con un lazo condicional se puede mostrar elementos solo si el usuario tiene un permiso específico. Los permisos de un usuario ingresado se almacenan en una variable *perms.auth*. Por ejemplo, para saber si un usuario tiene rol de financiero se incluiría una etiqueta `{% if perms.auth.is_financiero %}`. Por el lado de las vistas, Django provee otro decorador `@permission_required(permission, redirect)` en

donde se especifica un permiso específico y, opcionalmente, una URL a la que redirigir si se accede por error a la vista sin permisos adecuados.

Módulo de reportes

Una gran ventaja del uso de Python para desarrollo web es poder aprovechar el repertorio de librerías que dispone el lenguaje para aumentar funcionalidades en la aplicación web que no sean nativamente implementadas por Django. Una de estas librerías que mejor se adapta al contexto transaccional de SIPweb es Matplotlib que permite realizar diagramas y gráficos en base a información numérica que puede ser provista directamente de los datos recibidos de las API.

Un reporte se define como un detalle visual que permita asesorar, a través del tiempo, el avance de un proyecto en relación con todos los pagos a proveedores efectuados. Por esto, un reporte incluye el detalle de un proyecto, una tabla con todos los pagos a proveedores realizados, y un gráfico que permita tener una idea del avance de los pagos en función de las fechas. Se propone, para este prototipo, un gráfico de barras que presente los pagos de manera acumulativa de acuerdo con las distintas fechas.

```
fig, ax = plt.subplots(1, 1, figsize=(15, 10), dpi=220)
plt.bar(uz_tick, uz_totales, color=(0.09, 0.635, 0.722, 1))
plt.xticks(uz_tick, uz_fechas_string, rotation='vertical')
plt.tick_params(axis='x', labelsize=10)
plt.title('Pagos a proveedores', fontsize='18', **font)
plt.xlabel('Fecha de Pago', fontsize='14', **font)
plt.ylabel('Valor de Pago ($)', fontsize='14', **font)
plt.grid(True)
buf = BytesIO()
plt.savefig(buf, format='png', dpi=100)
image_base64 = base64.b64encode(buf.getvalue()).decode('utf-8')
                    .replace('\n', '')
buf.close()
```

Figura 15. Implementación de un reporte gráfico con la librería Matplotlib en Python 3.7.

En la Figura 15 se puede apreciar cómo se hace uso de la librería de Python y de los datos de la API para generar el gráfico que se renderizará en la plantilla HTML del reporte del

proyecto. Se usa la fecha y el valor de pago como variables independientes y dependientes, respectivamente.

Además, para mostrar la imagen rápidamente, se envía el gráfico como un búfer de base 64 bits que luego se renderiza en la plantilla HTML y se muestra con el tamaño que se pueda ajustar directamente en la etiqueta HTML para inserción de imágenes.

Con la inclusión del módulo de reportes, el prototipo SIPweb se encuentra completo y ha culminado el proceso de desarrollo. El código completo del proyecto en Django se encuentra en la sección de anexos del CD adjunto, bajo el directorio *Anexos\Anexo 3*. Ahí se halla la definición de todas las vistas, URL, formularios de cada *app*. Sin embargo, por motivos de privacidad de información, no se incluye ninguna información privada de la base de datos de Admindysad.

Pruebas

Para la etapa de pruebas, se ejecutan los procesos diseñados en las diferentes vistas para demostrar la funcionalidad del prototipo. Se mostrará el ciclo de vida de un proyecto alrededor de las distintas fases, compuestas por los casos de uso de la Figura 4. La página de inicio muestra un formulario sencillo de inicio de sesión, como se observa en la Figura 16. Ya se han creado usuarios con roles específicos, tal como fue descrito en la sección de implementación, así que se utiliza los mismos para probar las funciones.

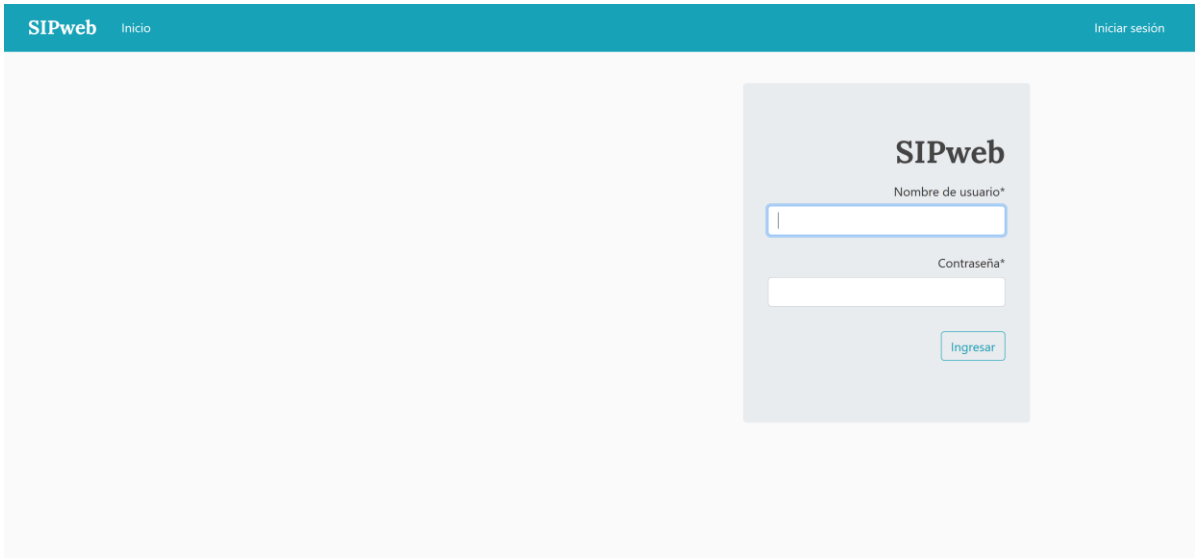

Figura 16. Pantalla de inicio del prototipo de SIPweb.

En primer lugar, se inicia sesión con el usuario Financiero, y al ingresar sus credenciales, se redirige a la misma pantalla de inicio, pero con un mensaje de inicio de sesión exitoso, tal como se aprecia en la Figura 17. Además, se puede apreciar que este usuario, con el rol de Director Financiero, tiene acceso al módulo Proyectos, desde el cual puede crear un nuevo proyecto asignando información de éste.

Figura 17. Página de inicio de SIPweb para el usuario Financiero.

En la Figura 18 se muestra la primera parte del formulario de creación de un proyecto. En esta sección se ingresan los datos principales de un nuevo proyecto. Cada campo es requerido, y algunos tienen validación adicional, como la fecha de finalización que no debe ser

inferior a la fecha de inicio, o el valor total que debe ser un número mayor a 0. Algunos campos son listas desplegables que realizan consultas independientes a la base de datos y devuelven un registro de todos los elementos de ese campo.

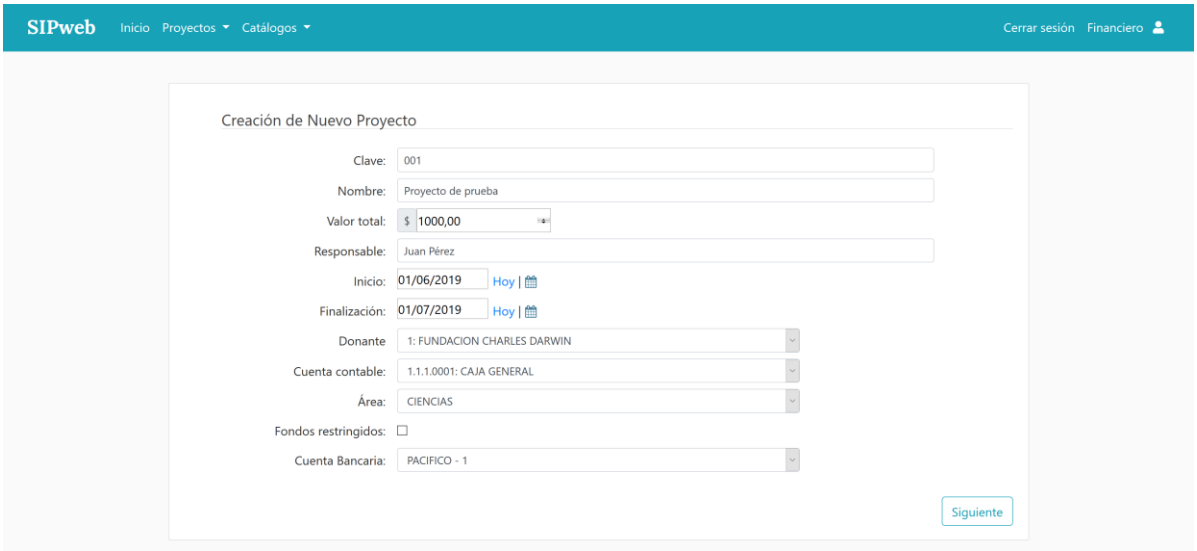

Figura 18. Ejemplo de inserción de información en el formulario de creación de proyectos.

Una vez ingresados los datos, Financiero pulsa “Siguiente”, y se dirige a la pantalla de asignación de objetivos y tareas para el proyecto, mostrada en la Figura 19. Cada objetivo contiene distintas tareas como se especificó en la descripción de casos de uso.

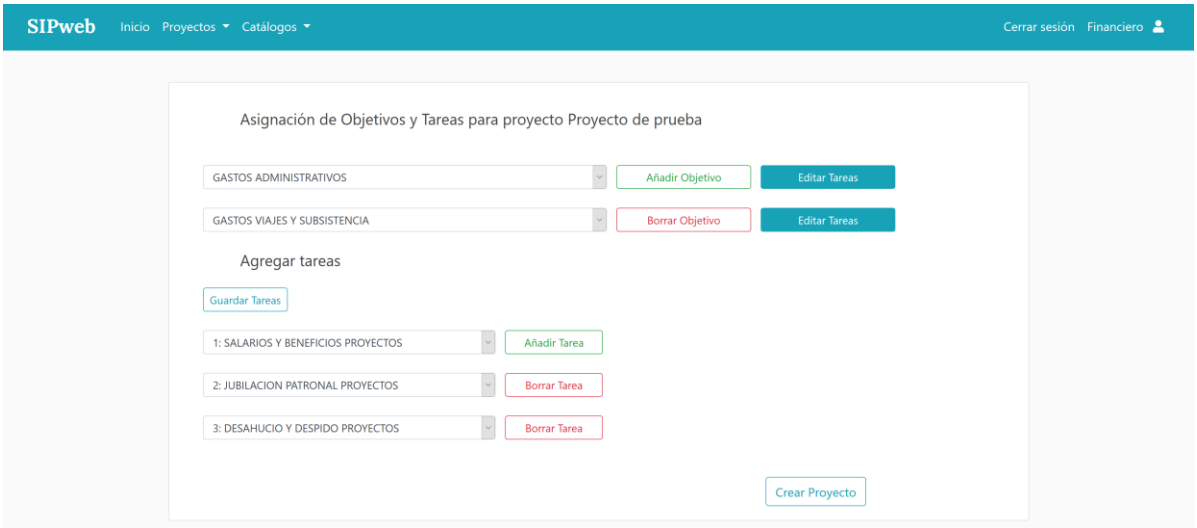

Figura 19. Ejemplo de asignación de objetivos y tareas para el proyecto creado en la figura 17.

Después de asignar objetivos y tareas, Financiero crea el proyecto, y éste es enviado a la base de datos con la información ingresada. Además de la gestión de proyectos, Financiero puede acceder a distintos catálogos, bajo el menú Catálogo en la barra superior de la Figura 19. Todos los catálogos se manejan de una manera bastante similar, por ejemplo, parte del catálogo de objetivos mostrado en la Figura 20 permite evidenciar todas las funcionalidades de un catálogo, desde la paginación hasta la opción de crear un nuevo elemento. Cabe destacar que un elemento tipo Área, Objetivo o Tarea puede ser gestionado solamente por un usuario Financiero, así que la opción de crear un nuevo elemento depende del usuario con los permisos adecuados.

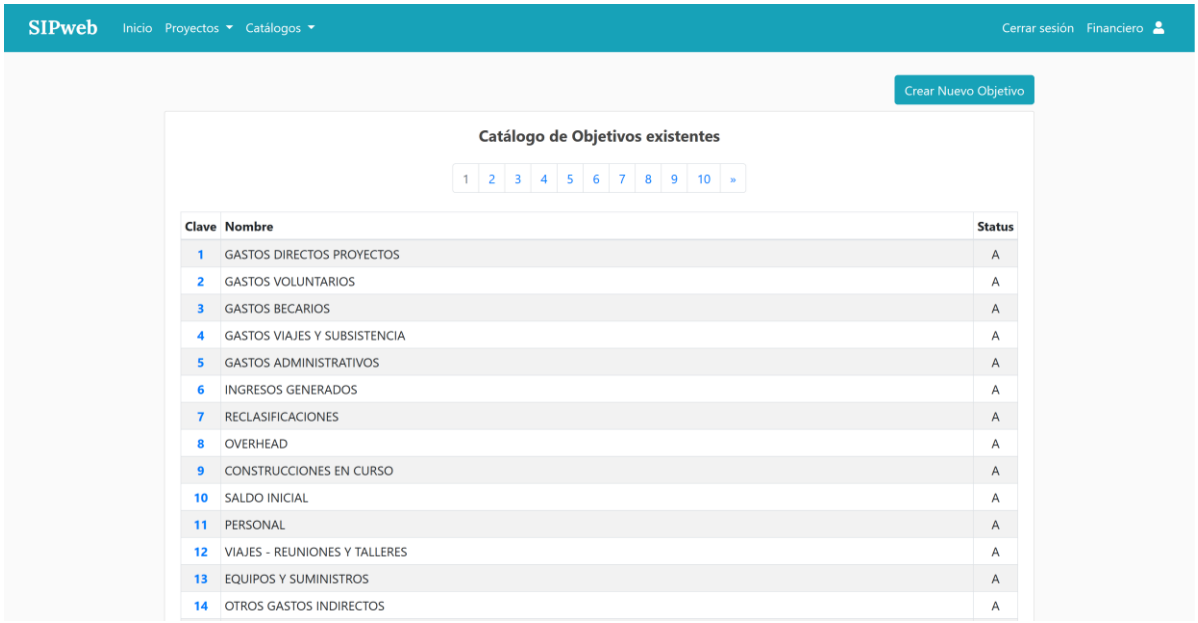

Figura 20. Pantalla de catálogo de objetivos para el usuario Financiero.

El resto de los catálogos se obvian en esta sección de pruebas ya que todos se basan en el mismo concepto descrito y demostrado en la Figura 20, además de que algunos catálogos muestran información de la base de datos de Admindysad que no debe mostrarse públicamente.

Finalmente, el usuario Financiero también tiene acceso a un módulo de reporte de proyectos, en el cual, además de mostrar la información del proyecto existente en la base de datos, permite visualizar un control de todas las cuentas por pagar a proveedores realizadas

hasta el momento. Esta información se despliega en una lista y en un gráfico de barras con valores acumulativos, tal como se describió en la sección de implementación. Un ejemplo de cómo se visualiza este gráfico en la pantalla de reportes se puede observar en la Figura 21.

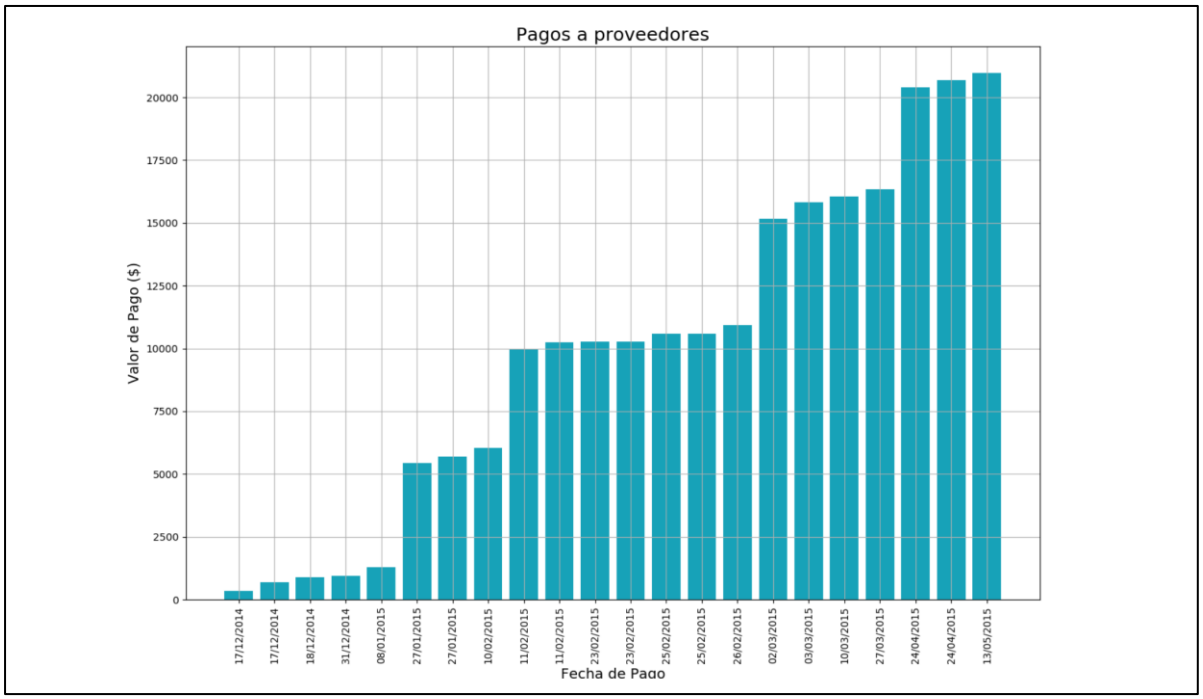

Figura 21. Gráfico de barras de las cuentas por pagar para el reporte de un proyecto existente.

Por otro lado, si el usuario Tesorero inicia sesión, su pantalla de inicio se observa como en la Figura 22. Cabe remarcar que el usuario Tesorero no tiene acceso visual al menú Proyectos, como el usuario Financiero en la Figura 16. Sin embargo, sí tiene acceso al módulo de Tesorería que permite el ingreso de cuentas por pagar a proveedores.

Figura 22. Pantalla de inicio de SIPweb para el usuario Tesorero.

Cuando el Tesorero accede a la pantalla de ingreso de cuentas por pagar, se muestra un formulario con información básica de facturación para un pago a un proveedor designado con los fondos respectivos de un proyecto. Este formulario se lo puede observar en la Figura 23.

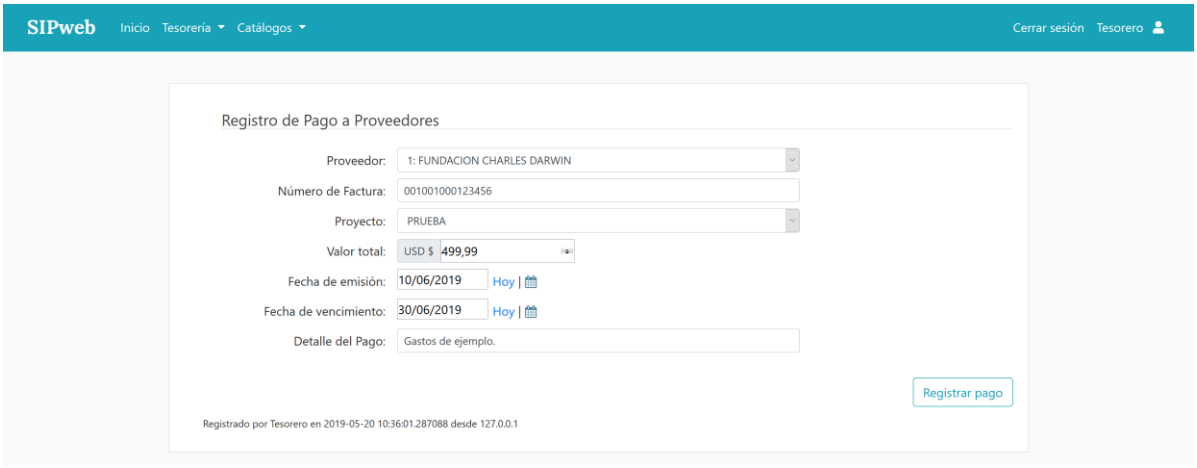

Figura 23. Ejemplo de ingreso de datos en formulario de registro de pago a proveedores.

Por último, para contrastar el diseño visual del catálogo observado en la Figura 20 con el mismo catálogo que se presenta al usuario Tesorero, se lo puede visualizar en la Figura 24. Cabe mencionar que, en este caso, no existe la opción de creación de un nuevo objetivo, debido a que ese caso de uso no está relacionado a un usuario Tesorero. Aun así, este usuario tiene la opción de visualizar y acceder a los catálogos y al detalle de cada elemento, sin la única opción de gestionarlos. La misma dinámica se presenta para el resto de los catálogos no mostrados en las figuras, y puede implementarse con facilidad en una manera estandarizada si es que surgen nuevos tipos de usuarios y roles en actualizaciones futuras.

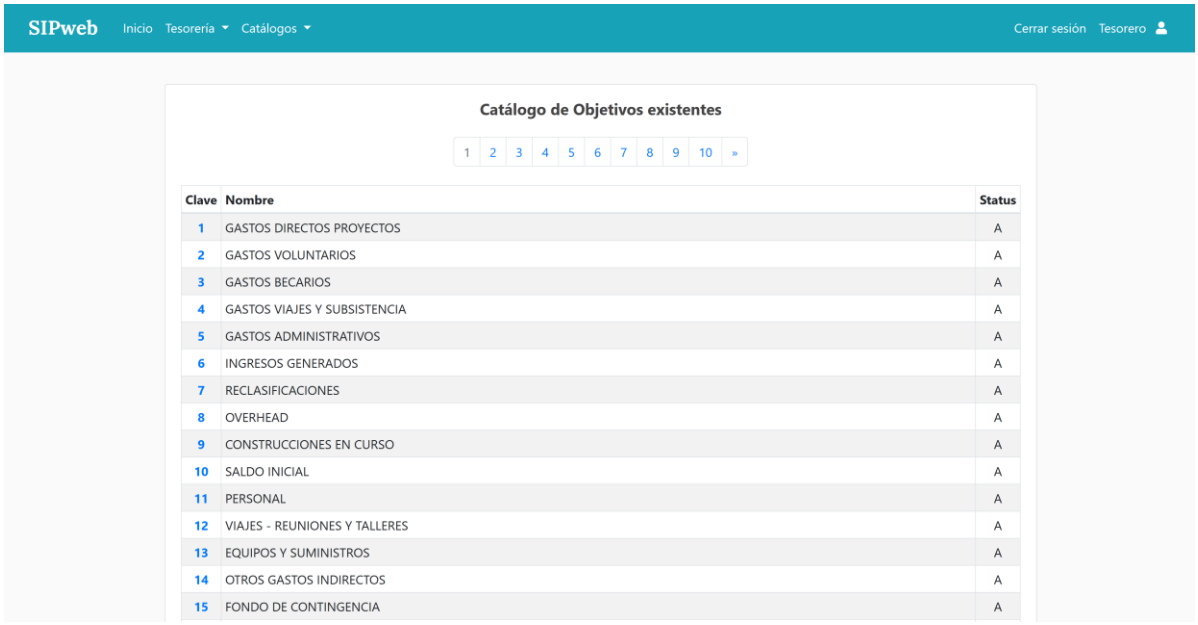

Figura 24. Pantalla de catálogo de objetivos para el usuario Tesorero.

Con esto se puede cerrar las diferentes pruebas de cada proceso evaluado en el análisis y diseño de la arquitectura, y desarrollado durante la fase de implementación del prototipo de SIPweb para gestión de proyectos.

Resultados

Cada uno de los procesos realizados durante la etapa de pruebas dio los resultados esperados. En primer lugar, el módulo de autenticación y autorización funciona como se lo propuso en donde los usuarios solamente pueden acceder a sus funciones, sin intervenir con las del resto. En el caso del usuario Financiero, que cumple el rol de Director Financiero, se puede observar que su participación se limita a la creación de proyectos, gestión de catálogos y visualización de reportes de proyectos, mas no tiene acceso a la vista relacionada con el registro de pagos a proveedores. Por otro lado, Tesorero solo puede acceder a la visualización de catálogos y registro de pagos a proveedores. Para complementar el módulo de autenticación, la funcionalidad de Django de encriptación de contraseñas provee un nivel de seguridad muy alto y permite cumplir con mayor cabalidad los requerimientos esperados de seguridad de un sistema de autenticación básico.

Por otro lado, en cuestión de los demás casos de uso, fue evidente cómo se pudo crear y manipular información desde las distintas vistas, sea de creación de proyectos o de ingreso de pagos a proveedores. Estos procedimientos se comunicaron efectivamente con la base de datos, agregando la información adecuada en las tablas requeridas. Se lo puede comprobar directamente al agregar un proyecto y evidenciar su existencia en el catálogo que presenta una lista a partir de una llamada a la base de datos. Al promover el concepto de bajo acoplamiento, tanto la creación de nuevos elementos como la consulta en catálogos son procesos separados e independientes que se comunican directamente al modelo de datos, y no entre sí, por lo que tener consistencia de datos alrededor de todo el proceso valida a la perfección los objetivos propuestos.

En cuestión de velocidad de llamada a las distintas API se encontraron varios puntos en los que la latencia de respuesta era alta. Esto, a pesar de influir directamente en la fluidez de la aplicación, no permite generar conclusiones directas en relación con la velocidad de consulta en la base de datos ni con la conexión con las API. El justificativo más probable alrededor de este dilema es la latencia otorgada por el sistema operativo en el que se montó la base de datos. Al ser una máquina virtual dentro del ordenador portátil donde corre el servidor web de Django, ésta carece de los suficientes recursos de memoria como para proveer una experiencia completa de la aplicación. La comunicación entre la máquina host y la máquina virtual presenta demoras considerables que permitirían explicar tiempos de espera altos dentro del prototipo, y evitan que estos se conviertan en factores que invaliden al prototipo.

CONCLUSIONES Y TRABAJO FUTURO

Conclusiones

En conclusión, se cumplieron todos los objetivos y se consiguió construir un prototipo que demuestra la posibilidad de trabajar en una aplicación web que haga uso del framework Django junto a un modelo de datos basado en D3. El uso de recursos web API permitió una comunicación fluida tanto para llamadas GET como para llamadas POST que demostraron su compatibilidad con un área de desarrollo ajeno como lo es Django. El hecho de que las API devuelvan la información de D3 en un formato JSON permite que el análisis y procesamiento de ésta se realice de manera bastante intuitiva, lo cual ayudó bastante al desarrollo del prototipo bajo un diseño que pueda generalizarse y escalar para abarcar más módulos de la aplicación SIP completa.

No se pudo concluir acerca de la velocidad de las API. Cuando el prototipo crezca y migre a recursos de hardware más potentes, se podrá evaluar con mayor precisión la rapidez de carga y descarga de la base de datos Rocket D3. Se recomienda, incluso, un análisis más exhaustivo y comparativo entre D3 y otras bases de datos más conocidas y populares para poder tener estadísticas y métricas de referencia que permitan responder si en realidad las ventajas de D3 se evidencian completamente en un contexto como el de la aplicación SIPweb y asesorar la necesidad del consumo de licencias de esta base de datos en comparación a recursos de código abierto o de menor costo.

Más allá de las ventajas propias de la base de datos D3, una recomendación que se propone para el eventual avance del prototipo SIPweb es un análisis exhaustivo del diseño de la implementación actual de la base de datos de Admindysad. Para garantizar que el prototipo pueda escalar con mayor facilidad y el equipo de trabajo pueda crecer independientemente es importante realizar un estudio sobre el diseño de los archivos en la base de datos que incluya

una refactorización de los nombres de los archivos, de los campos de cada archivo y de las posiciones. Regularmente se encontraron campos repetidos que apuntaban a una misma dirección. La razón principal para esta falla de diseño se debe a la característica de SIP de ser personalizado para cada cliente. A pesar de que la base de datos es suficientemente flexible para permitir errores de diseño, no es recomendable para una aplicación escalable. Para el prototipo de SIPweb se propone un esquema de trabajo más generalizado en donde se maneje un diseño de base de datos más intuitivo.

Una consideración adicional que vale la pena resaltar alrededor del proceso de desarrollo es la adquisición de licencias para el manejo de la base de datos Rocket D3 y el uso de todas las funcionalidades incluidas y necesarias para el desarrollo del prototipo. Esta gestión dependía completamente de la disponibilidad y disposición de las empresas Rocket Software, que distribuye y comercializa la base de datos Rocket D3, y Admindysad Cía. Ltda., que tiene contacto directo con Rocket y tiene los recursos para la adquisición e instalación de las licencias. Esta dependencia, de no ser tratada con la anticipación adecuada, puede resultar en retrasos inevitables en el proceso de desarrollo. Se recomienda realizar las comunicaciones pertinentes con antelación al desarrollo con todos los interesados del proyecto para tener la disponibilidad de recursos a tiempo.

Trabajo futuro

A pesar de que se cumplieron todos los objetivos en función del prototipo desarrollado en este trabajo, aún está lejos de ser una aplicación independiente en estado de producción. Se ha demostrado la validez de una arquitectura web para el proceso de gestión de proyectos, pero un software administrativo contable se constituye por mucho más que este módulo. El trabajo por realizar a futuro incluye, como objetivo principal, el extender la funcionalidad del prototipo para igualar y superar la de la aplicación SIP.

Por otro lado, también es necesario destacar que, por propósitos del desarrollo del prototipo, no se priorizó el aspecto visual de la aplicación. Un trabajo a futuro muy significativo al momento de presentar un prototipo completo es tener un mejor diseño visual. Por el momento se utiliza la versión predeterminada de Bootstrap 4 como librería de CSS. No obstante, existe la posibilidad de compilar y descargar una versión customizada de Bootstrap, ya que se trata de código abierto, con una gran cantidad de variables visuales que deberán ser analizadas a cabalidad para promover una presentación única para el prototipo SIPweb. Además, se espera un comportamiento más dinámico e intuitivo al usuario final, para lo cual se contempla un uso más avanzado de JavaScript dentro de cada pantalla, que puede ser complementado con la librería jQuery.

Para el prototipo actual se ha hecho uso de varias funcionalidades propias de Django, sin embargo, existen muchas más que permiten tener una aplicación más robusta, estable y escalable. En primer lugar, se puede implementar herramientas de testeo automatizado para validar las páginas que se vayan a crear mientras el prototipo crezca. Todo se gestiona convenientemente bajo un fichero *tests.py*, haciendo del proceso bastante sencillo. Además, se puede implementar procesadores de contexto, que permiten generalizar una vista y solo alterar la información que se renderiza. Esto permitiría generar estándares de programación a futuro.

Con respecto a cuestiones de seguridad, se propone, como recomendación fuerte, el migrar el modelo de usuarios a una base de datos más segura que SQLite. A pesar de que Django garantiza la seguridad en SQLite, es recomendable migrar datos a una base de datos más robusta que permita escalar y gestionar de mejor manera el eventual crecimiento de complejidad de roles y usuarios. Finalmente, medidas de seguridad básicas como la recuperación de contraseñas o el ingreso de información de usuario más completa, permitirá tener un modelo de usuarios más robusto que no dependa de un administrador para gestionarse.

Como consideraciones adicionales para trabajo futuro se incluye la incorporación de características nuevas al prototipo, como la posibilidad de exportar reportes para que puedan ser descargados en distintos formatos de archivos, o generar una interfaz que permita al usuario escoger cómo desea que se vea su reporte. Todo esto se puede conseguir con una mezcla de JavaScript y librerías de Python.

REFERENCIAS BIBLIOGRÁFICAS

Bachman, C. W. (1973, November). The programmer as navigator. *Communications of the ACM, 16*(11), 653-658.

Barth, A., Jackson, C., & Mitchell, J. C. (2008). *Robust Defenses for Cross-Site Request Forgery.* Stanford University, Alexandria. Retrieved from https://seclab.stanford.edu/websec/csrf/csrf.pdf

Christensen, C. M. (1993). The Rigid Disk Drive Industry: A History of Commercial and Technological Turbulence. *The Business History Review, 67*(4), 531-588. doi:10.2307/3116804

Codd, E. F. (1970, June). A relational model of data for large shared data banks. *Communications of the ACM, 13*(6), pp. 377-387. doi:10.1145/362384.362685

Cook, R., & Brandon, J. (1984, October). The Pick Operating System Part 1: Information Management. *BYTE Magazine, 9*(11), pp. 177-198.

Fielding, R. (2000). *Representational State Transfer (REST)*. Obtenido de ics.uci.edu: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Girap, S. (2018, January 8). *Pick operating system.* Retrieved from Alchetron: https://alchetron.com/Pick-operating-system

Grolinger, K., Higashino, W., Tiwari, A., & Capretz, M. (2013, December 18). Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing, 2*(22). doi:https://doi.org/10.1186/2192-113X-2-22

Johnson, W., & Anonymous. (2011, February 8). *Richard A "Dick" Pick.* (C. C. License, Ed.) Retrieved from County Historian: http://www.countyhistorian.com/knol/4hmquk6fx4gu-596-richard-a-dick-pick-d-19-oct-1994.html

Kaur, H., Kaur, J., & Kaur, K. (February de 2013). A Review Of Non Relational Databases, Their Types, Advantages And Disadvantages. *International Journal of Engineering Research & Technology (IJERT), 2*(2).

Kumar, D. (2018). *Best Practices for Building RESTful Web Services.* Obtenido de Infosys: https://www.infosys.com/digital/insights/Documents/restful-web-services.pdf

McCallum, J. (2019, April 21). *Disk Drive Prices (1955-2019)*. Retrieved from jcmit: https://jcmit.net/diskprice.htm

Miloslavskaya, N., & Tolstoy, A. (2016, December). Big Data, Fast Data and Data Lake Concepts. *Procedia Computer Science, 88*, 300-305. doi:10.1016/j.procs.2016.07.439

Phister, M. J. (1976). *Data Processing Technology and Economics.* Santa Monica CA: Santa Monica Publishing Co.

Pick, M. (16 de March de 2008). History of the PICK System #MultiValue. Obtenido de https://www.youtube.com/watch?v=6ms0yvJAUAk

Pick, R. (1987). *An overview of the Pick Operating System.* Irvine, California: Pick Systems.

Ritchie, D. M., & Thompson, K. (1974, July). The UNIX Time-Sharing System. *Communications of the ACM, 17*(7), 365-375. doi:10.1145/361011.361061

Rocket Software, Inc. (s.f.). MultiValue Application Platform. Waltham, Massachusetts, United States of America. Obtenido de https://www.rocketsoftware.com/product-categories/dbms-and-application-servers

Saracco, M., & Haderle, D. (April-June de 2013). The History and Growth of IBM's DB2. *IEEE Annals of the History of Computing, 35*, 54-66. doi:10.1109/MAHC.2012.55

Sisk, J. (January de 2000). *Pick/BASIC: A Programmer's Guide.* Irvine. Obtenido de jes.com.

TigerLogic Corporation. (2007). *D3 User's Guide.* Irwin: TigerLogic.

Wong, A., & Seltzer, M. (1999). *Operating System Support for Multi-User, Remote, Graphical Interaction.* Cambridge: Harvard Computer Science Group.