UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Ingeniería Informática

Tesis de Diploma

Título: La Inteligencia del Negocio Contable aplicada en el BI-Versat para la toma de decisiones

Autores:

Yasiel Valdés González

Daniel Izquierdo Hernández

Tutor: Ing. Luis Alberto Jiménez Martínez

Curso: 2020-2021

UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Software Engineering

Diploma Thesis

Title: Accounting Business Intelligence applied in BI-Versat for decision making

Authors:

Yasiel Valdés González

Daniel Izquierdo Hernández

Tutor: Ing. Luis Alberto Jiménez Martínez

Course: 2020-2021

Dedicatoria

A mis padres, mi hermano, mi novia y a toda mi familia, a los que están y a los que desgraciadamente ya no nos acompañan, por todo su apoyo y comprensión en momentos difíciles.

Yasiel Valdés González

Dedicatoria

El presente trabajo va dedicado a Dios, que ha sido exageradamente bueno conmigo todos estos años. A mi familia que, con apoyo incondicional, amor y confianza permitieron que logre culminar mi carrera profesional.

Daniel Izquierdo Hernández

Agradecimientos

A mi mamá, mi principal fuente de inspiración y apoyo y a quien debo gran parte de mis éxitos, gracias por tu inmenso e infinito cariño.

A mi papá, por sus sacrificios para que nada me faltase en la vida y por ese apoyo constante para que persevere en mis metas personales.

A mi hermano, por su apoyo y cariño a su hermano más pequeño y por ser ejemplo de profesional y de que nada en la vida es fácil y muchos menos sin esfuerzo.

A mis abuelos Antonio, Idalia, Pedro, Haydee, y también a mi tío-abuelo Evelio (Pally) por brindarme todo su cariño y apoyo.

A Lissi, por llenar de alegría y amor mi vida cuando menos lo esperaba y por acompañarme en toda esta travesía universitaria, llena de inolvidables momentos.

A Ramón, ese otro hermano mío, que siempre está en todo momento con su apoyo y su alegría. Y a todos aquellos que no están hoy con nosotros, como Yami e Ismany con las que siempre tuve su cariño, apoyo y alegría.

A Mary, Guille, Albe, y a toda esa gran familia que me han acogido como uno más y que en todo momento me han apoyado incondicionalmente.

A mis tíos Julio (Nino), María y Capiro, y a mi prima Massiel, por alentarme a ser un profesional.

A Javier Cárdenas y Dora Martín Brito (Dorita) por guiar mis pasos en el preuniversitario, haber hecho posible mi arribo a la universidad con sus enseñanzas, consejos.

A todos mis compañeros de aula por hacer de este transcurso una experiencia única.

A Juan Ibrahim Mesa (Juanito), por espantar mi miedos y preocupaciones cuando me sentí abrumado dentro de la carrera, gracias por cambiar mi perspectiva del asunto.

A Luis Alberto Jiménez Martínez, por confiar en sacar adelante esta investigación y por sus consejos oportunos.

Al personal de Datazucar, en especial a Luis Morel, por abrirnos las puertas del centro y poner a nuestra disposición todo aquello a su alcance para llevar a cabo la investigación.

A todo aquel que de una manera u otra ha influido positivamente en mi formación universitaria y como profesional, le doy Gracias.

Yasiel Valdés González

Agradecimientos

Quiero agradecerle primeramente a Dios, porque nunca hubo prueba lo suficientemente dura que no pudiera vencer con Él.

A mi padre, porque su ayuda fue vital todos estos años, y su vida ha sido para mí la mayor de las enseñanzas.

A mi madre, por su inmenso amor y cariño, por enseñarme la importancia del trabajo y de hacer las cosas bien.

A mi hermana, porque siempre estuvo disponible para escuchar, conversar, alentar y reír.

A mis abuelos maternos Arturo y Gina, por la fuerza que me transmiten, porque los años lejos de debilitarlos los fortalecen.

A mis abuelos paternos Hugo y Elsa, que hoy no están conmigo, pero marcaron valores en mi infancia.

A Amanda, por tantos momentos llenos de felicidad y ternura.

A todas las amistades que hice durante mi proceso educativo, nunca los olvidaré.

A Luis Alberto Jiménez Martínez nuestro tutor, porque no titubeó en prestarnos su ayuda cuando la pedimos.

A todo el equipo de trabajo de Datazucar, especialmente a Luis Morel por tratarnos como uno más y brindarnos todo lo que estuvo a su alcance para lograr este proyecto.

A todas las personas que de una forma u otra influyeron para que esto fuera posible, muchas gracias.

Daniel Izquierdo Hernández

Resumen

El avance tecnológico va de la mano con el crecimiento de la información. Actualmente las empresas disponen cantidades incuantificables de datos, sin prestar beneficio alguno. En tanto, las formas convencionales para interpretar información contable retardan la entrega de registros económicos, disminuyendo los tiempos de respuesta en la toma de decisiones. Los sistemas de inteligencia de negocio son sistemas que apoyan la toma de decisiones mediante análisis de la información en las empresas, basados en información del entorno y la información histórica de la misma. La información contable no es ajena a este tipo de análisis y es de gran utilidad a la empresa, permitiendo conocer la situación económica de la organización mediante indicadores financieros. La presente investigación tiene como objetivo presentar la Web Corporativa BI Versat como una solución de Inteligencia de Negocios que muestre los indicadores clave de desempeño de la empresa, su estado actual y desarrollo histórico. Esta manera de ver la información ahorra mucho tiempo al directivo y permite que no se pierda entre tablas y números, encontrando rápidamente lo que es esencial del dato. Esta herramienta permitirá a los directivos el seguimiento de los principales factores que contribuyen al funcionamiento y éxito del negocio, de forma que se pueda alcanzar el objetivo fijado, así como prescribir una línea de acción futura.

Palabras clave: toma de decisiones, inteligencia de negocios, información contable

Abstract

Technological advancement goes hand in hand with the growth of information. Currently companies have unquantifiable amounts of data, without providing any benefit. Meanwhile, the conventional ways to interpret accounting information, delay the delivery of economic records, reducing response times in decision making. Business intelligence systems are systems that support decision-making by analyzing information in companies based on information from the environment and its historical information. Accounting information is not alien to this type of analysis and is very useful to the company, allowing to know the economic situation of the organization through financial indicators. This thesis aims to present the BI Versat Corporate Website as a Business Intelligence solution that shows the key performance indicators of the company, its current status and its historical development. This way of seeing the information saves a lot of time for the manager and allows them not to get lost between tables and numbers, quickly finding what is essential in the data. This tool would allow managers to monitor the main factors that contribute to the operation and success of the business, so that the set objective can be achieved, as well as prescribe a future course of action.

Keywords: support decision-making, business intelligence, accounting information

Contenido

TABLA DE ILUSTRACIONES

Introducción

Los avances tecnológicos recientes han provocado una gran revolución informacional, incrementando la disponibilidad y las posibilidades de acceso a la información. Esto trae de la mano el acceso a grandes volúmenes de datos que generan un gran monto de operaciones internas, gastos, facturación, contabilidad, producción y personal.

A medida que la cantidad de datos acumulados se ha ido incrementando, ha proliferado también las necesidades de consultas más complejas. Los directivos se encuentran ante un gran volumen de información que es difícil de procesar con claridad. Pero tener datos no es igual a tener información.

Las soluciones de inteligencia de negocio (en lo adelante BI por sus términos en inglés) se basan en el tratamiento de los grandes volúmenes de datos que se encuentran distribuidos en una empresa con el fin de tomar la información realmente necesaria y así brindar información ventajosa para el proceso de toma de decisiones, además de ser capaz de recoger los datos que se generan diariamente y convertirlos en información activa.

Es de vital importancia para los directivos y quienes toman las decisiones en una empresa, el acceso cómodo y rápido a la información útil para encauzar, de mejor manera, las decisiones importantes para el futuro de la empresa e intentar sacar ventaja sobre los competidores en el sector.

En este sentido han sido desarrolladas varias investigaciones, tanto en el ámbito internacional como nacional, las cuales constituyen antecedentes de estudios similares realizados en universidades nacionales e internacionales. En la arena internacional resalta la investigación de (Agüero, 2019). De igual manera, se encuentra el trabajo de (Solórzano, 2018).

Las investigaciones anteriormente mencionadas constituyen un ejemplo de cómo la informática se encuentra vinculada al manejo de los procesos contables y financieros de las empresas en la actualidad. En nuestro país se cuenta con el sistema legado Versat Sarasola, software de Gestión Contable, que es utilizado por todas las entidades presupuestadas del país para efectuar el control económico desde los centros de gestión Contables y las Tesorerías creadas en todos y cada uno de los municipios del país. Este sistema se encuentra implementado en la empresa Datazucar, en la cual se enmarca el presente proyecto. Dicho sistema aún resuelve

funciones de vital importancia en la empresa y no puede suplantarse, constituye un punto crítico para un directivo que necesite tomar decisiones. Además de ello, la capacidad de funcionamiento del propio sistema se ve limitada, al no interactuar con otras aplicaciones, al no formar parte natural de las decisiones tomadas y de no disponer de mecanismos dinámicos para la retroalimentación del proceso.

Versat Sarasola brinda múltiples informes económicos, pero por sí solo, no es capaz de brindar la información oportuna que requieren los directivos de una empresa para tomar sus decisiones económicas. Ello trae consigo una demora para los directivos que deseen conocer información detallada de la empresa en tiempo real y útil para enfrentar los retos y disposiciones en el quehacer de la empresa.

La situación expuesta, evidencia la necesidad de una herramienta que a partir de los datos del sistema informático Versat Sarasola permita y facilite a los directivos tomar decisiones sobre el futuro desempeño empresarial.

A partir de lo planteado, se presenta como problema de investigación el siguiente:

¿Cómo aplicar la Inteligencia de Negocios en el sistema Bi-Versat de manera que ayude en el proceso de toma de decisiones de la empresa Datazucar?

Objetivo General: Implementar el software Bi-Versat aplicando Inteligencia de Negocio, de manera que brinde a los directivos de la empresa Datazucar, las herramientas para gestionar información económica, financiera y productiva en tiempo real para facilitar la toma de decisiones.

Objetivos Específicos:

- Realizar un análisis de la literatura especializada, abarcando los aspectos teóricos conceptuales y las experiencias prácticas existentes, relacionados a las metodologías para el desarrollo de soluciones de inteligencia de negocio.
- Describir el caso de estudio para el sistema Versat Sarasola.
- Desarrollar el sistema que ofrecerá a los directivos de la empresa Datazucar las herramientas para crear su propia información relevante.

- Desarrollar una API REST mediante el uso de Django Rest Framework para brindar escalabilidad y soporte a la aplicación.
- Definir diversos roles y permisos, especificando la información que será mostrada por cada perfil de usuario.
- Fundamentar el correcto funcionamiento del software mediante pruebas.

Estructura de Documento:

El trabajo consta de cuatro capítulos estructurados de la siguiente manera:

En el capítulo 1 se describen las principales características del sistema legado Versat Sarasola y las tecnologías y herramientas utilizadas en el desarrollo del sistema.

En el capítulo 2 se abordan aspectos relacionados al modelado del negocio, aplicando la metodología ágil de desarrollo XP. Se definen las historias de usuario, las tareas de ingeniería y las tarjetas CRC, con el objetivo de lograr que la aplicación sea confiable y fácil de manipular.

En el capítulo 3 se presenta el diseño de la base de datos y se describe la arquitectura del sistema. Además, se definen el diagrama de clases de diseño, así como el diagrama de despliegue y el de componentes.

En el capítulo 4 se describe la estrategia de pruebas a realizar y se muestran los resultados de estas aplicadas a la herramienta informática desarrollada, con el objetivo de validar sus funcionalidades.

Capítulo 1. Marco Teórico Referencial de la investigación

En el presente capítulo se describen las principales características del sistema legado Versat Sarasola y las tecnologías y herramientas utilizadas en el desarrollo del sistema.

El Business Intelligence (BI - por sus siglas en inglés) está presente en casi todas las grandes multinacionales alrededor del mundo. Netflix es una de las empresas más importantes del mundo en la actualidad, y también uno de los ejemplos más claros del éxito del Business Intelligence. Netflix tiene la capacidad para realizar análisis de millones de datos, lo que le permite conocer los gustos y preferencias de cada usuario al detalle, en función del uso que haga de la plataforma, gracias al análisis de datos. De esta forma, puede recomendar series y películas del modo más eficaz, siendo esta su principal ventaja competitiva (Fernández, Neira, y Clares, 2016).

Otro de los mejores ejemplos de aplicación de Business Intelligence es el de Coca-Cola Bottling Company, el socio de embotellado independiente más grande de Coca Cola. Esta compañía tuvo dificultad para acceder a datos de ventas y operaciones en tiempo real, debido a las restricciones de los procesos de informes manuales. Actualmente cuenta con una plataforma de Business Intelligence que automatiza los procesos de informes manuales, la cual ha permitido ahorrar más de 260 horas de trabajo al año, manejando los informes de todas las operaciones de ventas y entregas en la empresa (Sadiku, y Musa, 2021).

Actualmente en nuestro país la inteligencia de negocios se emplea con éxito en diferentes ámbitos y empresas. Existen sistemas de gestión contable financiera, entre los que podemos citar: (Datazucar, 2020), empleado en la mayoría de las empresas y entidades presupuestadas, RODAS (Citmatel, 2020) empleado en cadenas de tiendas, SISCONT5 (Cupet, 2020) mantenido por la Empresa Tecnomática, SICEMA PLUS SQL de la Empresa ALIMATIC (Alimatic, 2020) aplicado en la industria alimentaria y muchos otros que no emplean BI. Estos sistemas no automatizan la cadena de valor de la empresa. En su lugar automatizan funciones por separado: registros de clientes, facturas, compras, inventarios, finanzas, etc. Los usuarios deben realizar muchas tareas manuales, por ejemplo: consultando lo establecido en un contrato de venta para elaborar una factura, consultando un listado de precios establecido por un proveedor para proceder a confeccionar un pago (García, 2020). Estos son problemas

frecuentes en el ámbito empresarial cubano, que llevan a los directivos a buscar un software que utilice BI para solucionar el manejo de grandes volúmenes de datos en las empresas.

Por otra parte, se encuentra que el Versat Sarasola es un sistema integrado de gestión económica y financiera, diseñado para ser utilizado de acuerdo con las características de cada entidad. Dicho sistema es configurable por las entidades al momento de su instalación y tiene como objetivo fundamental permitirles a los directivos: analizar, controlar y evaluar los resultados de su negocio o actividad, en tiempo real, al contar con un instrumento seguro, rápido, eficaz y de fácil manejo para la Planificación, Control y el Análisis de la gestión económica y financiera (Hernández, 2013).

El sistema legado Versat Sarasola, aún resuelve funciones de vital importancia en la empresa y no puede suplantarse, constituye un punto crítico para un directivo que necesite tomar decisiones. Además de ello, la capacidad de funcionamiento del propio sistema se encuentra limitada, al no interactuar con otras aplicaciones, al no formar parte natural de las decisiones tomadas y de no disponer de mecanismos dinámicos para la retroalimentación del proceso.

Constituyen requerimientos de modernización del sistema, la implementación de mecanismos para suministrar sus datos a otras aplicaciones y a su vez, retroalimentarse desde agentes externos de forma regulada.

A partir de lo planteado, se presenta como problema de investigación el siguiente:

¿Cómo aplicar la Inteligencia de Negocios en el sistema Bi-Versat de manera que ayude en el proceso de toma de decisiones de la empresa Datazucar?

Justificación de la investigación

Hoy en día la información es el activo más importante en todas las empresas por más grande o pequeña que sean. El conocimiento del cliente, la comprensión de los procesos internos y la efectividad con que se realicen todas las operaciones que demanda la entidad son los pilares sobre los cuales descansa el éxito empresarial.

Las empresas y/o negocios en gran parte cuentan con muchos datos e información, pero a la vez cuentan con carencia de conocimiento. La mayoría de estos negocios y/o empresas

cuentan con sistemas aislados, cada uno con características, datos e infraestructura propios. Ello incide de manera negativa en el mantenimiento de información oportuna y actual.

Mientras una empresa sea más organizada y esté más integrada, será más fácil para cualquier usuario y/o colaborador, obtener la información que necesite y a su vez encontrándose en la mejor oportunidad para tomar decisiones al respecto.

Las primeras aproximaciones a lo que en hoy en día se conocen como soluciones de inteligencia de negocios fueron los almacenes de datos relacionales, a partir de diseños multidimensionales, basados en dimensiones para los hechos. La intención consistió en desarrollar sistemas que permiten unificar datos que se obtienen desde múltiples fuentes, para luego contar con poderosas herramientas de visualización donde se permita el consumo masivo, orientado al usuario, y la distribución automática de información para la toma de decisiones obtenida del procesamiento de los datos primarios almacenados en las fuentes.

La inteligencia de negocios es la habilidad de transformar los datos en información, y dicha información en conocimiento, de manera que se pueda optimizar el proceso de la toma de decisiones en los negocios de cualquier organización.

1.1. Bases teóricas-científicas

1.1.1. Business Intelligence

Inteligencia de negocios o Business Intelligence es el conjunto de soluciones tecnológicas que integran y facilitan la transformación de los datos de la compañía en conocimiento para obtener una ventaja competitiva (Agüero, 2019).

El Business Inteligence es una amplia categoría de aplicaciones y tecnologías para recoger, almacenar, analizar y proveer acceso a datos para ayudar a los usuarios de las empresas a tomar mejores decisiones de negocio. Las aplicaciones de BI incluyen las actividades de los sistemas de soporte a las decisiones (DSS por sus siglas en inglés), consultas e informes, tecnologías OLAP, análisis estadístico y datamining (Cano, 2007).

En el libro Business intelligence: Competir con información, Josep Lluís Cano sostiene que el objetivo básico de inteligencia de negocios es apoyar, de forma sostenible y continuada, a las

organizaciones para mejorar su competitividad, facilitando la información necesaria para la toma de decisiones (2007).

El Business Intelligence, es una solución que se compone por una plataforma tecnológica que integra todas las fuentes de información tanto internas como externas de la organización para que se vean como una sola y luego ser visualizadas mediante herramientas de reporteo, así como también de las preguntas clave del negocio y quizás lo más importante, la inteligencia analítica del usuario. Sin embargo, el usuario final no debe conocer la complejidad de los sistemas que almacenan los datos para poder obtener reportes e indicadores en el momento apropiado para una óptima toma de decisiones. BI permite monitorear toda la empresa desde una pantalla, integrando todos los departamentos y asegurando que la información que se presenta sea veraz y confiable. Con una solución de BI ya no es necesario esperar a final del año, ni al final del trimestre, ni siquiera al final del día para ver cómo va la empresa de acuerdo a sus objetivos.

1.1.2. Fuentes de información

Las fuentes de información son todo aquello que proporciona datos para reconstruir hechos y las bases del conocimiento. Son un instrumento para el conocimiento, la búsqueda y el acceso a la información.

Luis Antonio Domínguez (2016) define que la información es un conjunto organizado de datos procesados. Constituyen un mensaje que pasa al conocimiento del sujeto o de quien recibe el mensaje. Durante años, el ser humano ha utilizado diferentes métodos para el manejo de la información y toma de decisiones. Actualmente la transformación de la gestión administrativa de las empresas está basada en la información que se tiene del desempeño de los indicadores en los diferentes procesos, siendo necesario recurrir a un procedimiento técnico aplicable para el desarrollo del Sistema de Información que se utilice como guía para el correcto seguimiento de los pasos a seguir que facilite una información útil.

Andreu, Ricart y Valor (1991), definieron a los sistemas de información como:

> conjunto formal de procesos que, operando sobre una colección de datos estructurada de acuerdo a las necesidades de la empresa, recopila, elabora y distribuyen selectivamente la información necesaria para la operación de dicha empresa y para las actividades de dirección y control

> correspondientes, apoyando, al menos en parte, los procesos de toma de decisiones necesarios para desempeñar funciones de negocio de la empresa de acuerdo con su estrategia. (citado en Merencio, 2019).

Las fuentes de información a la que se puede acceder son:

- De los sistemas operacionales o transaccionales, que incluyen aplicaciones desarrolladas a medida de ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), SCM (Supply Chain Management), entre otros.

- De Sistemas de información de los departamentos: contabilidad, tesorería, presupuesto, hojas de cálculo, etc.

- De fuentes externas: como estudios de mercado realizados o investigaciones hechas**.**

Las fuentes de información externas son básicas y fundamentales para enriquecer la información que tenemos de nuestros clientes. En algunos casos es interesante incorporar información referente, por ejemplo, a población, número de habitantes, entre otros. Acceder a varias bases de datos requiere habilidades y el conocimiento de distintas sintaxis de SQL (Cano, 2007).

1.1.3. Toma de decisiones

Las decisiones pueden tomarse en un contexto de certidumbre, incertidumbre o riesgo. Tomar buenas decisiones es algo que todo gerente se esfuerza por lograr, ya que la calidad de las decisiones administrativas influye poderosamente en el éxito o fracaso de una organización (Agüero, 2019).

La toma de decisiones en una organización se circunscribe a todo un colectivo de personas que están apoyando el mismo proyecto.

1.2 Definición de términos básicos

1.2.1 Aplicación Web

Una aplicación web, es una aplicación distribuida, la cual es accedida por los clientes finales mediante la conexión con un servidor vía web por una determinada red, ya sea Internet o una Intranet. Es posible en término general, describirla como un programa informático que es

ejecutado en un navegador del cliente, debido a que la codificación de esta es con algún lenguaje soportado por estos, como, por ejemplo, HTML con JavaScript, entre otros. Dichos navegadores serán capaces de renderizar la aplicación para mostrarla a los usuarios y que puedan interactuar con ella.

La comunicación existente entre los servidores y los clientes finales se realiza mediante el protocolo HTTP (*Hypertext Transfer Protocol*). Dicho protocolo de la capa de aplicación sigue el esquema petición – respuesta entre los clientes y servidores, el cual, permite la transferencia de información en la *World Wide Web*. La forma que tiene el usuario de realizar peticiones al servidor es mediante las URLs (Caldera, 2017).

1.2.2 Framework Web

Se refiere a la plataforma sobre la cual se desarrolla un proyecto de software, incluyendo sus técnicas y metodologías. Es un conjunto de componentes de software que construyen un diseño reutilizable que facilita y agiliza el desarrollo de una aplicación web robusta. Se puede considerar como una aplicación web incompleta y configurable a la que se le pueden añadir nuevas funcionalidades para conseguir el comportamiento deseado por el grupo de programadores. Estas herramientas y funcionalidades pueden ser: librerías de clases, funciones, scripts, etc. (Solórzano, 2018).

Los frameworks promueven buenas prácticas de desarrollo mediante el uso de patrones de diseño. También, permiten una mayor fiabilidad de las páginas de una aplicación web, ya que el framework ha sido previamente sometido a diferentes pruebas.

1.3 Fundamentación del entorno de desarrollo

1.3.1 Python

Python es un lenguaje de scripting independiente de plataforma que se encuentra orientado a diversos objetos, este cuenta con la capacidad para llevar a cabo cualquier tipo de programa. Aquí se pueden incluir las aplicaciones del sistema operativo Windows y servidores de red, y además diversas páginas en línea. (Llerena, 2020)

En la actualidad, este lenguaje de programación ha ganado gran popularidad, esto se debe a una serie de particularidades que ofrece a sus usuarios. Entre estas:

- Python cuenta con una gran diversidad de librerías, distintos tipos de datos y funciones que se encuentran incorporadas en su propio lenguaje de programación. Lo cual, contribuye en la realización de diversas tareas habituales. Esto es posible sin la necesidad de tener que programarlas desde el inicio.
- Posee una gran sencillez y rapidez para desarrollar nuevos programas.
- Es totalmente gratuito, e incluso para el cumplimiento de diversos propósitos empresariales.

1.3.2 Django

Django es un framework web de alto nivel que permite el desarrollo rápido de sitios web seguros y mantenibles. Desarrollado por programadores experimentados, Django maneja de forma “nativa” una serie de soluciones a situaciones comunes del desarrollo web, facilitando una buena cantidad de herramientas al programador para que no tenga que “reinventar la rueda”. Es gratuito y de código abierto, tiene una comunidad próspera y activa, una gran documentación y muchas opciones de soporte gratuito y de pago (Vincent, 2021).

Django posee algunas directrices para el desarrollo de software de calidad. Para ello establece que el código debe ser:

Completo

Se sigue la filosofía "Baterías incluidas" y provee casi todo lo que los desarrolladores desean tener "de fábrica". Porque todo lo que se necesita es parte de un único "producto", todo funciona a la perfección, sigue principios de diseño consistentes y tiene una amplia y actualizada documentación.

Versátil

Django puede ser (y ha sido) usado para construir casi cualquier tipo de sitio web, desde sistemas manejadores de contenidos y wikis, hasta redes sociales y sitios de noticias. Puede

funcionar con cualquier framework en el lado del cliente, y puede devolver contenido en casi cualquier formato (incluyendo HTML, RSS feeds, JSON, XML, etc). Internamente, mientras ofrece opciones para casi cualquier funcionalidad que se desee (distintos motores de base de datos, motores de plantillas, etc.), también puede ser extendido para usar otros componentes si es necesario.

Seguro

Django ayuda a los desarrolladores a evitar varios errores comunes de seguridad al proveer un framework que ha sido diseñado para "hacer lo correcto", para proteger el sitio web automáticamente. Por ejemplo, Django, proporciona una manera segura de administrar cuentas de usuario y contraseñas, evitando así errores comunes como colocar informaciones de sesión en cookies donde es vulnerable (en lugar de eso las cookies solo contienen una clave y los datos se almacenan en la base de datos) o se almacenan directamente las contraseñas en un hash de contraseñas.

Django permite protección contra algunas vulnerabilidades de forma predeterminada, incluida la inyección SQL, scripts entre sitios, falsificación de solicitudes entre sitios y clickjacking.

Escalable

Django usa un componente basado en la arquitectura shared-nothing (cada parte de la arquitectura es independiente de las otras, y por lo tanto puede ser reemplazado o cambiado si es necesario). Teniendo en cuenta una clara separación entre las diferentes partes significa que puede escalar para aumentar el tráfico al agregar hardware en cualquier nivel: servidores de caché, servidores de bases de datos o servidores de aplicación. Algunos de los sitios más concurridos han escalado a Django para satisfacer sus demandas (por ejemplo, Instagram y Disqus).

Mantenible

El código de Django está escrito usando principios y patrones de diseño para fomentar la creación de código mantenible y reutilizable. En particular, utiliza el principio No te repitas "Don't Repeat Yourself" (DRY) para que no exista una duplicación innecesaria, reduciendo la cantidad de código. Django también promueve la agrupación de la funcionalidad relacionada en

aplicaciones reutilizables y en un nivel más bajo, agrupa código relacionado en módulos, siguiendo el patrón Model View Template (MVT) (Pino, 2021).

1.3.3 Api (Interfaz de Programación de Aplicaciones)

Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.

Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

A veces, las API se consideran como contratos, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

Las necesidades comerciales suelen cambiar rápidamente en respuesta a los mercados digitales en constante cambio, donde la competencia puede modificar un sector entero con una aplicación nueva. Para seguir siendo competitivos, es importante admitir la implementación y el desarrollo rápidos de servicios innovadores. El desarrollo de aplicaciones nativas de la nube es una forma identificable de aumentar la velocidad de desarrollo y se basa en la conexión de una arquitectura de aplicaciones de microservicios a través de las API.

Las API son un medio simplificado para conectar su propia infraestructura a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las API públicas representan un valor comercial único porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden

rentabilizar sus datos (un ejemplo conocido es la API de Google Maps) (Fernández, 2018).

Ilustración 1 Ejemplo de Funcionamiento de una API
Fuente: (Viniegra, 2020)

1.3.4 Api Rest

El término REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP. Un servicio REST no es una arquitectura software, sino un conjunto de restricciones que se tienen en cuenta en la arquitectura software que usaremos para crear aplicaciones web respetando HTTP.

Las restricciones que definen a un sistema RESTful serían:

- Cliente-servidor: El servidor se encarga de controlar los datos mientras que el cliente se encarga de manejar las interacciones del usuario. Esta restricción mantiene al cliente y al servidor débilmente acoplados (el cliente no necesita conocer los detalles de implementación del servidor y el servidor se “despreocupa” de cómo son usados los datos que envía al cliente).
- Sin estado: cada petición que recibe el servidor debería ser independiente y contener todo lo necesario para ser procesada.

- Cacheable: debe admitir un sistema de almacenamiento en caché. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.
- Interfaz uniforme: define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección o “URI”.
- Sistema de capas: el servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.

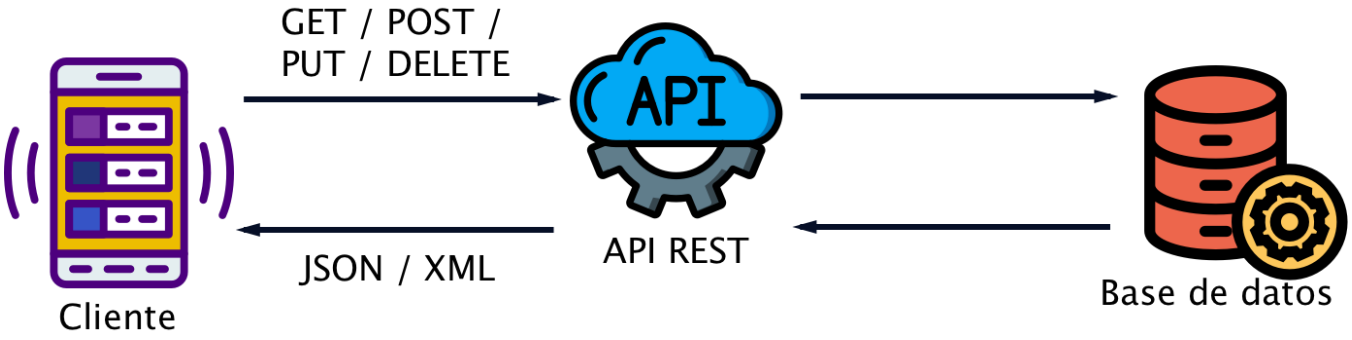


Ilustración 2 Estructura de comunicación de una Api Rest
Fuente: (Infante, 2020)

La mayoría de las empresas utilizan API REST para crear servicios web. Esto se debe a que es un estándar lógico y eficiente. Por poner algún ejemplo tenemos los sistemas de identificación de Facebook o también la autenticación en los servicios de Google (hojas de cálculo, Google Analytics, Google Maps).

Métodos en una API REST

Las operaciones más importantes que permiten manipular los recursos son:

- GET es usado para recuperar un recurso.

- POST se usa la mayoría de las veces para crear un nuevo recurso. También puede usarse para enviar datos a un recurso que ya existe para su procesamiento. En este segundo caso, no se crearía ningún recurso nuevo.
- PUT es útil para crear o editar un recurso. En el cuerpo de la petición irá la representación completa del recurso. En caso de existir, se reemplaza, de lo contrario se crea el nuevo recurso.
- PATCH realiza actualizaciones parciales. En el cuerpo de la petición se incluirán los cambios a realizar en el recurso. Puede ser más eficiente en el uso de la red que PUT ya que no envía el recurso completo.
- DELETE se usa para eliminar un recurso.

Otras operaciones menos comunes, pero también destacables son:

- HEAD funciona igual que GET, pero no recupera el recurso. Se usa sobre todo para testear si existe el recurso antes de hacer la petición GET para obtenerlo (un ejemplo de su utilidad sería comprobar si existe un fichero o recurso de gran tamaño y saber la respuesta que se obtendría de la API REST antes de proceder a la descarga del recurso).
- OPTIONS permite al cliente conocer las opciones o requerimientos asociados a un recurso antes de iniciar cualquier petición sobre el mismo.

Dos términos a tener en cuenta son los de métodos seguros y métodos idempotentes. Los métodos seguros son aquellos que no modifican recursos (GET, HEAD y OPTIONS), mientras que los métodos idempotentes son aquellos que se pueden llamar varias veces obteniendo el mismo resultado (GET, PUT, DELETE, HEAD y OPTIONS).

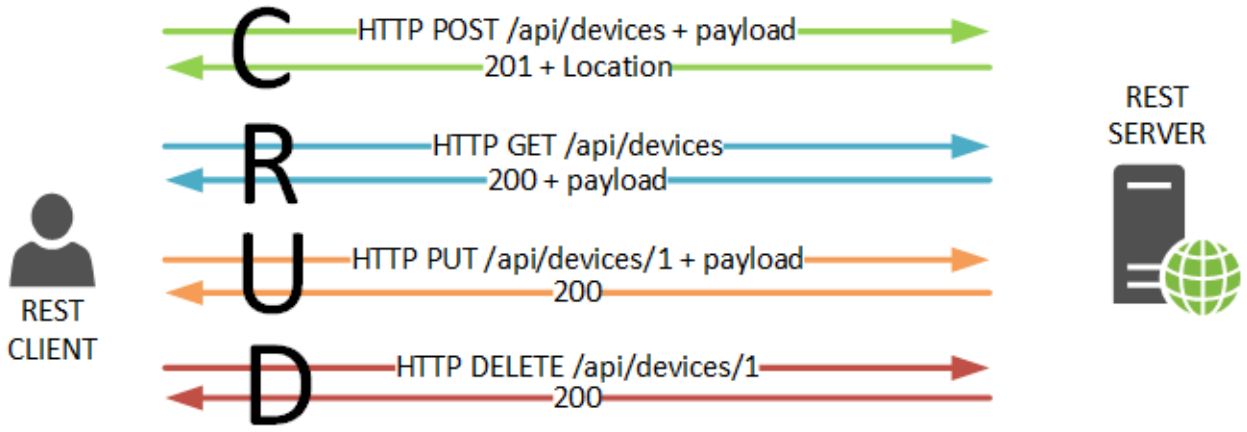


Ilustración 3 Métodos en una Api Rest

Fuente: (Mircha, 2020)

Características de una API REST

- El uso de hipermedios (procedimientos para crear contenidos que contengan texto, imagen, video, audio y otros métodos de información) para permitir al usuario navegar por los distintos recursos de una API REST a través de enlaces HTML (principio HATEOAS, Hypermedia as the engine of application state o hipermedia como el motor del estado de la aplicación).

- Independencia de lenguajes. La separación en capas de la API permite que el cliente se despreocupe del lenguaje en que esté implementado el servidor. Basta a ambos con saber que las respuestas se recibirán en el lenguaje de intercambio usado (que será XML o JSON).

- Los recursos en una API REST se identifican por medio de URI. Será esa misma URI la que permitirá acceder al recurso o realizar cualquier operación de modificación sobre el mismo.

- Las APIs deben manejar cualquier error que se produzca, devolviendo la información de error adecuada al cliente. Por ejemplo, en el caso de que se haga una petición GET sobre un recurso inexistente, la API devuelve un código de error HTTP 404.

¿Cuál es la principal ventaja de una API REST?

La principal ventaja del uso de una API REST reside en la independencia que proporciona frente a cualquier consumidor, sin importar el lenguaje o plataforma con el que se acceda a ella. Esto permite que una misma API REST sea consumida por infinidad de clientes, sea cual sea la naturaleza de estos y que el cambio a cualquier otro tipo de consumidor no provoque impacto alguno en ella. Esta característica proporciona fiabilidad, escalabilidad y una fácil portabilidad a cualquier otra plataforma, ya que aísla, por completo, al cliente del servidor. Solo se requiere que el intercambio de información de las respuestas se haga en un formato soportado, por lo general JSON o XML. Dicha separación entre el cliente y el servidor hace que se pueda migrar a otros servidores o bases de datos de manera transparente, siempre y cuando los datos se sigan enviado de manera correcta. Esto convierte a las APIs REST en una de las arquitecturas web más utilizadas por la flexibilidad que aportan a cualquier entorno de trabajo sea cual sea su naturaleza (Fernández, 2018).

1.3.5 Django REST Framework

A continuación, se aborda sobre algunas definiciones básicas que se hacen necesarias para comprender su uso:

- API (Interfaz de programación de aplicaciones): están pensadas para ser accedidas por otros programas.

Ventajas de ofrecer una API como servicio:

- Control de la información que se entrega.
- Información actualizada.
- Flexibilidad del manejo interno del servicio.
- Volumen de datos.
- Facilidad de filtrar información.
- Datos normalizados.

Desarrollo orientado a microservicios:

- Equipos de desarrollo pequeños y especializados.

❖ CRUD

Se refiere a las operaciones básicas (Crear, Leer, Actualizar y Eliminar) de los objetos de la base de datos.

❖ REST

▪ Estilo de arquitectura de software para la creación de APIs.

▪ Métodos HTTP explícitos: GET, POST, PUT, PATCH, DELETE.

❖ Serializers

▪ Convierten objetos de Python a formatos de datos más simples como JSON y XML (serialización) y viceversa (deserialización).

▪ Validan los datos que recibe la aplicación, como los Forms en Django.

❖ Vistas (views) especializadas

Vistas basadas en clases que se corresponden con los métodos de HTTP utilizados para CRUD:

- CreateAPIView → POST
- RetrieveAPIView y ListAPIView → GET
- UpdateAPIView → PUT + PATCH
- DestroyAPIView → DELETE

❖ ModelSerializer: Basado en el modelo, genera automáticamente los campos y validaciones del serializer. Muy similar a ModelForm de Django.

❖ Viewsets:

- Encapsulan la lógica de varias vistas relacionadas en una sola clase.
- Permiten utilizar actions y routers.

- GenericViewSet, ModelViewSet y ReadOnlyModelViewSet.

❖ Routers:

- Generan automáticamente estructuras de URLs típicas. Si no se especifica el basename, se genera automáticamente en base al queryset de la viewset.

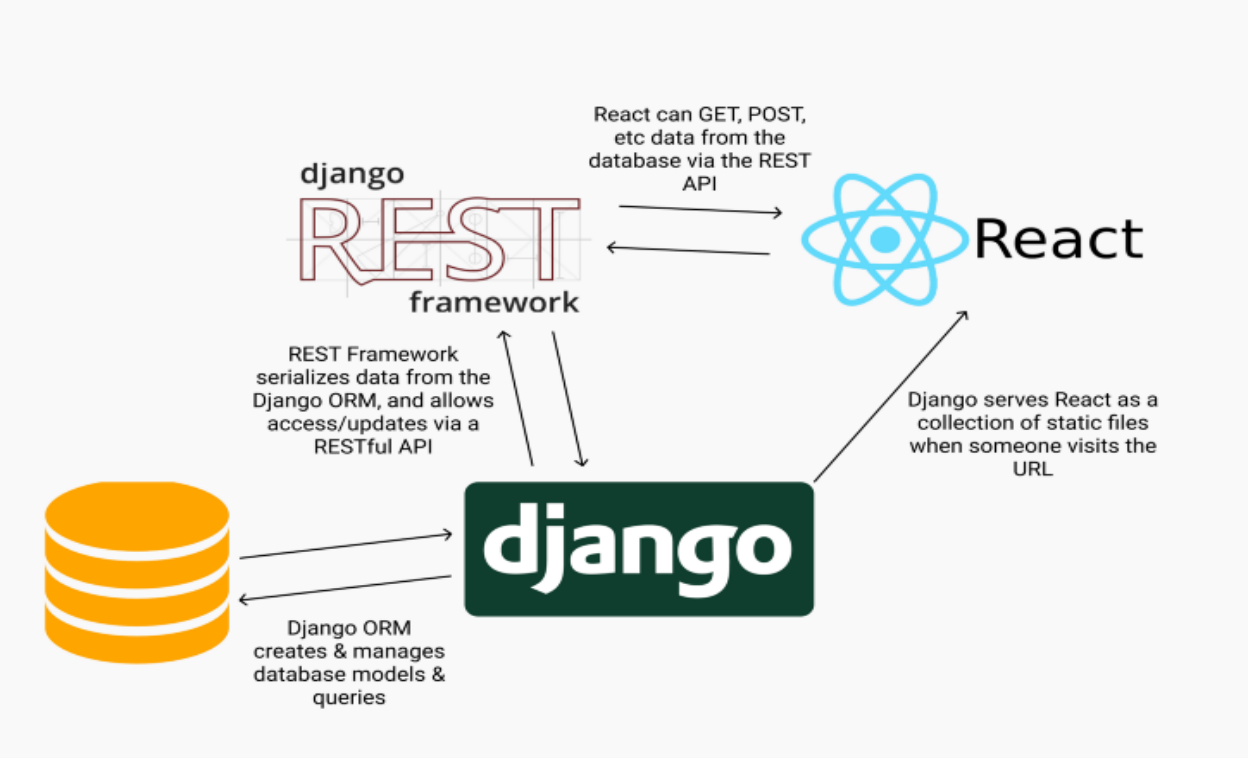


Ilustración 4 Comunicación entre Django, Django Rest, base de datos y React.
Fuente: (Garner, 2019).

1.3.6 PostgreSQL

En la actualidad existen disímiles SGBD (Sistemas Gestores de Bases de Datos), los más conocidos son SQLServer, PostgreSQL, Oracle, MySQL de los cuales se elige PostgreSQL en su versión 8.4 por ser orientado a objetos, libre, código abierto, de fácil instalación y cómodo para la conexión. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (Schönig, 2019).

1.3.7 Bootstrap

Bootstrap permite crear interfaces web con CSS y JavaScript, este permite adaptar la interfaz del sitio web al tamaño del dispositivo en que se visualice. Es decir, el sitio web se adapta automáticamente al tamaño de una PC, una Tablet u otro dispositivo. Esta técnica de diseño y desarrollo se conoce como responsive design o diseño adaptativo. Este framework evita preocuparse por las *media queries* y los porcentajes en los CSS para hacer una web responsive, facilitando la programación del sitio (Moreto, Lambert, Jakobus, y Marah, 2017).

Además, se basa en la simplicidad de sus interfaces, lo cual es una tendencia del mercado, en las que emplea un diseño plano, botones grandes para facilitar la usabilidad en los dispositivos más pequeños, atendiendo a la experiencia de usuario.

Ventajas

- Facilidad de diseñar una web usando elementos compuestos por diferentes combinaciones de HTML, CSS y JavaScript.
- Utiliza HTML5, CSS3, jQuery o GitHub, entre otros.
- Incluye Grid System: muy útil para maquetar por columnas.
- Sus plantillas son de sencilla adaptación (responsive).
- Se integra con librerías JavaScript.
- Usa Less: un lenguaje de las hojas de estilo CSS preparado para enriquecer los estilos de la web.
- Cuenta con una documentación completa.
- Facilita enormemente el diseño de interfaces e incluye por defecto una plantilla bastante optimizada.
- Soporte: Hay una gran comunidad de desarrolladores con implementación externa como WordPress.
- Rapidez y comodidad: Sencilla herramienta de construcción para sitios web e interfaces.

- Componentes: Mucha variedad de temas y plantillas a utilizar. (Hernández, 2020)

Desventajas de Bootstrap

- Aunque la curva de aprendizaje es leve, se debe familiarizar con sus estructuras y nomenclaturas.
- El diseño gráfico debe estar adaptado a las 12 columnas.
- Por defecto ya tiene anchos, márgenes y altos de líneas.
- Es complicado cambiar de versión cuando se han realizado modificaciones profundas sobre su núcleo.
- Cuando se necesita añadir componentes que no existan, se debe personalizar para mantener la coherencia con el diseño. (Sarzosa, 2018)

1.3.8 JavaScript

JavaScript surgió con un propósito muy claro: mejorar la navegación del usuario directamente desde el navegador. JavaScript es la tercera pieza fundamental del desarrollo web frontend, junto con los lenguajes HTML y CSS (Ferguson, 2019). Cada uno de estos tres lenguajes tiene una función muy concreta en el desarrollo web:

- El HTML se utiliza para conformar el esqueleto y la estructura de los contenidos de una página web.
- El CSS define el estilo y la apariencia web.
- JavaScript rompe con la rigidez del HTML y permite crear elementos dinámicos e interactivos, mejorando ampliamente la interacción de los usuarios con una página web.

Características de JavaScript

Lenguaje del lado del cliente:

Cuando se dice que un lenguaje es del lado del cliente, se refiere a que se ejecuta en la máquina del propio cliente a través de un navegador. Algunos de estos lenguajes son el propio JavaScript, HTML, CSS.

Lenguaje orientado a objetos:

JavaScript es un lenguaje orientado a objetos, es decir, utiliza clases y objetos como estructuras que permiten organizarse de forma simple y son reutilizables durante todo el desarrollo. Otros lenguajes orientados a objetos son Java, Python y C++.

Lenguaje interpretado:

JavaScript es un lenguaje interpretado porque utiliza un intérprete que permite convertir las líneas de código en el lenguaje de la máquina. Esto tiene un gran número de ventajas como la reducción del procesamiento en servidores web al ejecutarse directamente en el navegador del usuario, o que es apto para múltiples plataformas permitiendo usar el mismo código.

Muy utilizado por desarrolladores:

JavaScript es, en la actualidad, uno de los lenguajes más demandados de los últimos años por su versatilidad y su infinita capacidad para crear plataformas cada vez más atractivas.

1.3.9 jQuery

La librería jQuery es una de las librerías más conocidas para programar en JavaScript, y cuenta con una gran comunidad de usuarios y desarrolladores. Una de sus principales características es que es se trata de una librería open source, es decir, de código abierto. JQuery ayuda a desarrolladores y diseñadores a agregar elementos interactivos y dinámicos a sus sitios, limando inconsistencias en los navegadores y reduciendo grandemente el tiempo de desarrollo (Lara, 2019).

Su filosofía se basa en realizar órdenes de codificación simples y escuetas, programando en una o dos líneas lo que en JavaScript llevaría 20 líneas. Esta característica simplifica enormemente el trabajo de desarrollo, haciéndola muy popular en el sector.

Esta librería cuenta con una gran cantidad de extensiones o plugins que permiten añadir más funcionalidades a su núcleo, dotando al desarrollador de una gran flexibilidad y capacidades a la hora de afrontar un proyecto en JavaScript. Empresas tan importantes como Google, WordPress e IBM confían en jQuery para varios de sus proyectos.

1.3.10 Chart JS

Chart JS es un plugin JavaScript simple, flexible y muy completo para los diseñadores gráficos y desarrolladores que desean incrustar gráficas en las páginas web. No requiere jQuery. Tiene 6 tipos de tablas: gráficos de curvas, gráficos de barras, gráficos de radar, gráficos circulares, gráficos de área polar y gráficos de anillos. La librería Chart.js permite la opción de personalizar todos los aspectos de las gráficas. Por ejemplo, se puede cambiar el color y anchura de los bordes de las barras. También se puede modificar las descripciones de texto y la leyenda cambiando el tamaño de fuente y color (Da Rocha, 2019).

1.4 Consideraciones finales del capítulo

En este capítulo se estudiaron las tecnologías con las cuales se pudo realizar la presente investigación. De las herramientas utilizadas se especifican las principales características y las ventajas que brindan. La implementación del sistema informático se basó en el modelo cliente/servidor, desarrollando una aplicación web con las ventajas que esta arquitectura ofrece. Brinda la posibilidad de que los cambios solo se realicen en el servidor, que se pueda acceder a ella desde cualquier máquina conectada a la red mediante un navegador web sin necesidad de instalarse, y que la puedan emplear varios usuarios a la vez. Se utilizará Django como framework para el desarrollo de la aplicación, por las grandes facilidades que brinda a los programadores y como gestor de bases de datos se utilizará PostgreSQL.

Capítulo 2. Modelo del Negocio y Requisitos

En este capítulo se abordan aspectos relacionados al modelado del negocio, aplicando la metodología ágil de desarrollo XP. Se definen las historias de usuario, las tareas de ingeniería y las tarjetas CRC, con el objetivo de lograr que la aplicación sea confiable y fácil de manipular.

El desarrollo de software no es una tarea fácil, prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Entre ellas se encuentran las tradicionales, que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que deben ser usadas en un proyecto.

Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Esta forma de desarrollo no resulta ser la más adecuada para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante y se exige reducir drásticamente los tiempos de desarrollo, pero manteniendo una alta calidad.

Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, en este escenario emergen las Metodologías Ágiles (AMs) como una posible respuesta para llenar este vacío metodológico.

2.1 Metodologías Ágiles (Ams)

Las Metodologías Ágiles son una supuesta solución a las falencias e inconvenientes que presentan los tradicionales métodos de desarrollo. Los procesos ágiles ofrecen un enfoque diametralmente opuesto para gestionar y controlar el desarrollo de productos y proyectos de software.

Las metodologías ágiles de desarrollo de software son una agrupación de las prácticas tradicionales pero llevadas al extremo, tomando la esencia y aplicándolas, buscando la calidad en el desarrollo desde el inicio, entregas oportunas y la entrega final del sistema, teniendo en cuenta el soporte, mantenimiento, auditoría y capacitaciones al usuario final (Caballero, 2009).

Estas metodologías poseen una serie de principios fundamentales para el desarrollo de un software de calidad. Dentro de sus máximas se encuentra el desarrollo de software que funcione más allá de conseguir una buena documentación. Se sobrepone el minimalismo al modelado y la documentación del sistema. La colaboración con el cliente cobra mayor importancia que la negociación de un contrato, así como la respuesta a los cambios sobre la planificación estricta que dictan las metodologías tradicionales.

Programación Extrema o Extreme Programming (XP)

Es un enfoque ágil que tiene como objetivo fundamental brindar un código sin errores, donde el cliente pueda probar todas las funcionalidades y esté al tanto de lo que está ocurriendo en el proyecto en ese tiempo. Entre sus principales características se encuentran: posee un desarrollo iterativo e incremental, la programación se hace en parejas, se hacen las pruebas unitarias continuas para evitar los errores, lo que facilita las correcciones por las entregas frecuentes, se puede refactorizar y simplificar el código (Salazar, 2015). XP se enmarca en las metodologías ágiles por su enfoque en el trabajo en equipo y la forma en la que permite la rápida integración de las actividades del grupo. Esta metodología permite una revisión continua del código elaborado por el grupo de trabajo, permite la elaboración de prototipos mostrados al cliente en intervalos de tiempo más cortos.

Ilustración 5 Metodología XP
Fuente: (Muradas, 2018)

2.2 Metodología de desarrollo seleccionada para la implementación: XP

Luego de analizar las características esenciales que presentan las metodologías ágiles y las tradicionales, la investigación hará uso de XP, ya que puede existir una retroalimentación continua, comunicación fluida entre los participantes y el desarrollo de la aplicación en pareja. Esta presenta un tipo de desarrollo iterativo e incremental que facilita que se puedan implementar requisitos no identificados en el inicio del proyecto.

2.3 Herramientas de la Metodología XP

2.3.1 Historias de Usuario

Las Historias de Usuario representan una breve descripción del comportamiento del sistema, se realizan por cada característica principal del mismo y son utilizadas para cumplir estimaciones de tiempo. Así mismo reemplazan un gran documento de requisitos y presiden la creación de las pruebas de aceptación.

Cada Historia de Usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan implementarlas en unas semanas (Meléndez, Gaitan, y Pérez, 2016).

2.3.2 Tareas de Ingenierías (Task Card)

Una Historia de Usuario se descompone en varias tareas de ingeniería, las cuales describen las actividades que se realizarán. Así mismo las tareas de ingeniería se vinculan más al desarrollador, ya que permite tener un acercamiento con el código (Meléndez et al., 2016).

2.3.3 Pruebas de Aceptación

Las pruebas de aceptación son de vital importancia para el éxito de una iteración y el comienzo de la siguiente, con lo cual el cliente puede conocer el avance en el desarrollo del sistema y a los programadores lo que les resta por hacer. Además, permite una retroalimentación para el desarrollo de las próximas historias de usuarios a ser entregadas. Estas son comúnmente llamadas pruebas del cliente, por lo que son realizadas por el encargado de verificar si las

historias de usuarios de cada iteración cumplen con la funcionalidad esperada (Meléndez et al., 2016).

2.3.4 Tarjetas CRC

Las Tarjetas CRC (Clase-Responsabilidades-Colaboradores), permiten conocer qué clases componen el sistema y cuáles interactúan entre sí. Se dividen en tres secciones: Nombre de la Clase, Responsabilidades y Colaboradores (Meléndez et al., 2016).

2.4 Roles de la Metodología XP

Programador: El programador escribe las pruebas unitarias y produce el código del sistema.

Cliente: Escribe las historias de usuarios y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración, centrándose en aportar mayor valor al negocio.

Encargado de Prueba (*Tester*): Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

Encargado de Seguimiento (*Tracker*): Proporciona retroalimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

Entrenador (*Coach*): Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

Consultor: Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

Gestor (*Big Boss*): Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

2.5 Fases de la Programación Extrema

2.5.1 Planeación

La metodología XP plantea la planificación como un diálogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, a los programadores y a los coordinadores. El proyecto comienza recopilando las historias de usuarios, las que constituyen a los tradicionales Casos de Uso. Una vez obtenidas estas historias de usuarios, los programadores evalúan rápidamente el tiempo de desarrollo de cada una.

2.5.2 Diseño

La metodología XP hace especial énfasis en los diseños simples y claros. Los conceptos más importantes de diseño en esta metodología son los siguientes:

Simplicidad: Un diseño simple se implementa más rápido que uno complejo. Por ello, XP propone implementar el diseño más simple posible que funcione.

Soluciones “Spike”: Cuando aparecen problemas técnicos, o cuando es difícil de estimar el tiempo para implementar una historia de usuario, pueden utilizarse pequeños programas de prueba (llamados “Spike”), para explorar diferentes soluciones.

Recodificación (“Refactoring”): Consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de crearlo más simple, conciso y entendible. Las metodologías de XP sugieren recodificar cada vez que sea necesario.

2.5.3 Codificación

Disponibilidad del Cliente: Uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo. Involucrar al cliente es fundamental para que pueda desarrollarse un proyecto con la metodología XP. Al comienzo del desarrollo del software, este debe proporcionar las historias de usuarios, pero dado que estas historias son expresamente cortas y de “alto nivel”, no contienen los detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los programadores, durante la etapa de desarrollo.

Uso de Estándares: XP promueve la programación basada en estándares, de manera que sea entendible por todo el equipo, y que facilite la recodificación.

Programación Dirigida por Pruebas (Test-Driven Programming): En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los test, es usualmente realizada en la última etapa del desarrollo del software, o al final de cada módulo. La metodología XP propone un modelo inverso, primero se escriben las pruebas que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas. Las pruebas a los que se refiere esta práctica son los test unitarios, realizados por los desarrolladores. La definición de estas al comienzo condiciona o “dirige” el desarrollo.

Programación en Pares: XP propone que se desarrolle en pares de programadores, ambos trabajando juntos. Si bien parece que esta práctica duplica el tiempo asignado al proyecto (y, por ende, los costos en recursos humanos), al trabajar en pares se minimizan los errores y se logran mejores diseños, compensando la inversión en horas. El producto obtenido es por lo general de mejor calidad que cuando el desarrollo se realiza por programadores individuales.

Integraciones Permanentes: Todos los desarrolladores necesitan trabajar siempre con la “última versión”. Realizar cambios o mejoras sobre versiones antiguas causan graves problemas, y retrasan al proyecto. Debido a ello, XP promueve publicar lo antes posible las nuevas versiones, aunque no sean las últimas, siempre que estén libres de errores. Idealmente, todos los días deben existir nuevas versiones publicadas. Para evitar errores, solo una pareja de desarrolladores puede integrar su código a la vez.

Propiedad Colectiva del Código: En un proyecto XP, todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto. Asimismo, una pareja de programadores puede cambiar el código que sea necesario para corregir problemas, agregar funciones o recodificar.

Ritmo Sostenido: La metodología XP indica que debe llevarse un ritmo sostenido de trabajo. El concepto que se desea establecer con esta práctica es planificar el trabajo de forma a mantener un ritmo constante y razonable, sin sobrecargar al equipo (Meléndez et al., 2016).

2.5.4 Pruebas

Pruebas Unitarias: Todos los módulos deben de pasar las pruebas unitarias antes de ser liberados o publicados. Por otra parte, las pruebas deben ser definidas antes de realizar el

código (*Test-Driven Programmming*). Que todo código liberado pase correctamente pruebas unitarias, es lo que habilita que funcione la propiedad colectiva del código.

Detección y Corrección de Errores: Cuando se encuentra un error (Bug), este debe ser corregido inmediatamente, y se deben tener precauciones para que errores similares no vuelvan a ocurrir. Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto.

2.6 Prácticas de la metodología XP

XP es para pequeños y medianos equipos basándose en la comunicación continua entre todos los participantes, la simplicidad en las soluciones implementadas y coraje para enfrentar los cambios.

Esta metodología recomienda seguir las siguientes prácticas:

- Comunicación**:** Conversación continua entre el equipo de desarrollo y el cliente, para implementar cambios lo antes posible.
- Entregas pequeñas: Entrega en versiones operativas.
- Pruebas: Se realizan pruebas unitarias por parte de los programadores.
- Refactorización (refactoring): Remover código duplicado para facilitar los posteriores cambios.
- Programación en parejas: Se realiza para contar con menor tasa de errores, mejor diseño y mayor satisfacción de los programadores.
- Continua: Cuando un fragmento de código esté listo, puede ser integrado al sistema.
- Cliente in-situ: El cliente debe estar presente y disponible para el equipo de desarrollo.
- Estándares de programación: Normas definidas por los desarrolladores para tener un código legible.
- Juego de la planificación: Desde el comienzo del desarrollo se requiere que el grupo y el cliente tengan una visión general del proyecto. En el transcurso de este se realizan diferentes reuniones, con el fin de organizar las tareas e ideas que surgen tanto por parte del cliente como del equipo.
- Propiedad colectiva del código: El código no es conocido por una sola persona del grupo de trabajo, esto facilita implementar cambios al programa por parte de otros integrantes del grupo.

- 40 horas por semana: Se debe de trabajar un máximo de 40 horas por semana (Meléndez et al., 2016).

2.7 Puesta en práctica de la metodología XP al sistema

Tomando en cuenta las ventajas que ofrece XP no es necesario adoptarla en su forma completa, sino que pueden utilizarse varias de sus prácticas en forma independiente. Esto hace que el costo de su implementación sea mucho más accesible que el de otras metodologías. Es decir, solo se toma en cuenta las prácticas que serán de utilidad para el desarrollo del problema planteado.

2.7.1 Interacción con el cliente

El cliente se encuentra más cercano al proceso de desarrollo, tanto así que se considera como parte del equipo. Se elimina la fase inicial de captura de requisitos y se permite que estos se vayan definiendo de una forma ordenada durante el tiempo que dura el proyecto. El cliente puede cambiar de opinión sobre la marcha y debe encontrarse siempre disponible para resolver dudas del equipo de desarrollo y para detallar los requisitos especificados cuando sea necesario.

El juego de la planificación consiste en una reunión entre el programador y el cliente, en este caso los estudiantes con el jefe del sector de desarrollo de aplicaciones en la empresa Datazucar. Este encuentro se lleva a cabo con el objetivo de realizar las historias de usuario, que en sí son la captura de requisitos, ya sea funcionales o no. El cliente plantea a los programadores las tareas que quiere ver realizadas en cada iteración, buscando aportar mayor valor al negocio. También los programadores deben reunirse con el entrenador (tutor) quien aporta la idea general para la implementación. Luego, queda conformada la historia de usuario.

Entregas Pequeñas: Estas entregas son realizadas por los programadores (estudiantes) los cuales determinan el tiempo aproximado de duración para la realización. Para la implementación de la entrega, los programadores deben reunirse con el consultor (Especialista en Contabilidad y Finanzas) quien posee dominio sobre el perfil económico de la empresa. El cliente juega un papel importante pues selecciona qué construir en la siguiente entrega, teniendo en cuenta el tiempo de duración que los programadores determinaron para la tarea en

cuestión. Después de esto, los programadores vuelven a retomar el control y entonces realizan la implementación. En todo momento, estas tareas son seguidas por el Encargado de Seguimiento (tutor), el cual verifica las estimaciones realizadas y el tiempo real dedicado para mejorar futuras estimaciones.

Diseño simple: Es realizado por los programadores en un momento determinado del proyecto y es la solución más simple, es una entrega eficiente y rápida al cliente, totalmente funcional.

Pruebas: Son elaboradas por el cliente en conjunto con el encargado de pruebas (tutor), quien ayuda al cliente a elaborarlas y es el responsable de las herramientas para estas. Las pruebas son ejecutadas constantemente antes de cada modificación del sistema.

Cliente in-situ: Esto consiste en la participación constante del cliente**,** jefe del sector de desarrollo de aplicaciones en la empresa Datazucar, en el proyecto. Es decir, este siempre tiene que estar presente y disponible todo el tiempo para el equipo, esto debe funcionar ya que es uno de los principales factores para el éxito de XP. Esto permite que los programadores siempre tengan a su alcance a la persona que determina lo que se realizará en cada historia de usuario.

2.7.2 Captura de requisitos

El proceso de captura de requisitos en XP gira en torno a una lista de características que el cliente desea que existan en el sistema final. Cada una de estas características recibe el nombre de Historias de Usuarios**,** y su definición consta de dos fases:

1. El cliente describe con sus propias palabras las características y el responsable del equipo de desarrollo le informa de la dificultad técnica de cada una de ellas y, por lo tanto, de su coste. A través del diálogo resultante el cliente deja por escrito un conjunto de historias y las ordena en función de la prioridad que tienen para él. Entonces, se definen algunos hitos y fechas aproximadas para ellas.

2. La segunda fase consiste en coger las primeras historias que serían implementadas (primera iteración) y dividirlas en las tareas necesarias para llevarlas a cabo. El cliente también participa, pero hay más peso por parte del equipo de desarrollo, que daría como resultado una planificación más exacta. En cada iteración se repite esta segunda fase para las historias planificadas para ella.

2.7.3 Historias de usuario

El gestor del proyecto prevé la reunión entre los programadores (estudiantes) y el cliente (jefe del sector de desarrollo de aplicaciones en la empresa Datazucar), donde el cliente define el valor del negocio a implementar, es decir, la historia de usuario a realizar, luego los programadores estiman el esfuerzo necesario para su implementación, él decide el tiempo de duración de esta y de las tareas planteadas en la historia de usuario cual será implementada.

Luego de esta reunión ya queda conformada la historia de usuario, quedaría de la siguiente manera:

Una historia de usuario en general contiene como ítems los siguientes atributos: Número de historia, prioridad técnica, referencia a otra historia previa, riesgo, descripción, notas y una lista de seguimiento con las fechas, estado, cosas por terminar y comentarios (Caballero, 2009).

Historia de Usuario: 1

Nombre Historia: Gestión de Usuarios en el sistema
Usuario: Administrador
Prioridad en negocio: Alta
Riesgo en desarrollo: (Alta / Media / Baja)
Iteración asignada: 1
Puntos estimados: 5
Programador responsable: Equipo XP
Observaciones**:** Confirmado con el cliente.

Según sus prioridades y las restricciones de tiempo el cliente es quien decide la tarea a implementar. Es decir, toma la tarea teniendo en cuenta la restricción de tiempo puesta por los programadores para la implementación de esta. A partir de este momento se desglosan en tarea(s), siempre recordando que varias tareas conforman una historia de usuario. Surgen

entonces las llamadas tarjetas de tareas, que son las que se reparten entre los programadores, es decir lo que se modela entre ellos.

Tarea: 1

Nombre Tarea: Insertar Usuario	
Usuario: Administrador	
Tipo de Tarea: Desarrollo	
Puntos estimados: 3	
Fecha inicio: 16/03/21	Fecha fin: 21/03/21
Programador responsable: Equipo XP Descripción: Comprobar que el usuario a insertar no está en la base de datos	

Para el desarrollo y puesta en práctica del próximo requisito funcional se debe realizar el mismo proceso, lo que cambian son las tarjetas en algunos de sus aspectos. Para obtener las tarjetas para el requisito siguiente se deberá realizar nuevamente el ciclo de desarrollo.

Historia de Usuario: 2

Nombre Historia: Listar Balance de Comprobación
Usuario: Administrativo Empresarial
Prioridad en negocio: Alta
Riesgo en desarrollo: Alto
Puntos estimados: 2
Iteración asignada: 1
Programador responsable: Equipo XP

Historia de Usuario: 3

Nombre Historia: Gestionar Inventarios
Usuario: Administrativo Empresarial
Prioridad en negocio: Alta
Riesgo en desarrollo: Media
Puntos estimados: 3
Iteración asignada: 1
Programador responsable: Equipo XP

Tarea: 2

Nombre Tarea: Listar Inventarios-Existencias, Listar Inventarios-Entradas, Listar Inventarios-Salidas	
Usuario: Gestor Administrativo	
Tipo de Tarea: Desarrollo	
Puntos estimados: 3	
Fecha inicio: 25/03/21	Fecha fin: 10/04/21
Programador responsable: Equipo XP	

Debido a que es bien engorroso el proceso de listar todas las tareas que contiene el desarrollo del proyecto, solo se listarán las historias de usuario (requisitos funcionales en otras metodologías)

Historia de Usuario: 4

Nombre Historia: Listar Cuentas por Pagar
Usuario: Administrativo Empresarial
Prioridad en negocio: Alta
Riesgo en desarrollo: Media
Puntos estimados: 2
Iteración asignada: 1
Programador responsable: Equipo XP

Historia de Usuario: 5

Nombre Historia: Listar Cuentas por Cobrar
Usuario: Administrativo Empresarial
Prioridad en negocio: Alta
Riesgo en desarrollo: Media
Puntos estimados: 2
Iteración asignada: 2
Programador responsable: Equipo XP

Historia de Usuario: 6

Nombre Historia: Listar Resumen de Ventas Detalladas
Usuario: Administrativo Empresarial

Prioridad en negocio: Alta
Riesgo en desarrollo: Alta
Puntos estimados: 4
Iteración asignada: 2
Programador responsable: Equipo XP

Historia de Usuario: 7

Nombre Historia: Listar Ejercicios
Usuario: Administrativo Empresarial
Prioridad en negocio: Media
Riesgo en desarrollo: Baja
Puntos estimados: 2
Iteración asignada: 2
Programador responsable: Equipo XP

Historia de Usuario: 8

Nombre Historia: Listar Periodos
Usuario: Administrativo Empresarial
Prioridad en negocio: Media
Riesgo en desarrollo: Baja

Puntos estimados: 3
Iteración asignada: 2
Programador responsable: Equipo XP

2.7.4 Pruebas de Aceptación

CASO DE PRUEBA	
Código: 1	N.º Historia de Usuario: 1
Historia de Usuario: Gestionar Usuarios.	
Condiciones de Ejecución: Cada usuario debe contar con un perfil de usuario y su contraseña para poder acceder a las funciones del sistema.	
Entrada/Pasos de Ejecución: Llenar el formulario usuario introduciendo su nombre de usuario y contraseña Luego pulsar el botón Autenticar.	
Resultado Esperado: Acceso a las funcionalidades del sistema.	
Evaluación de la Prueba: La prueba se concluyó satisfactoriamente.	

CASO DE PRUEBA	
Código: 2	N.º Historia de Usuario: 2
Historia de Usuario: Listar Balance de Comprobación.	
Condiciones de Ejecución: El usuario debe estar autenticado para poder listar el balance.	
Entrada/Pasos de Ejecución: El usuario se autentifica y puede acceder al balance mediante la barra lateral donde se encuentran.	

Resultado Esperado: Listar el Balance de Comprobación.
Evaluación de la Prueba: La prueba se concluyó satisfactoriamente.

CASO DE PRUEBA	
Código: 3	N.º Historia de Usuario: 3
Historia de Usuario: Gestionar Inventarios	
Condiciones de Ejecución: El usuario debe estar autenticado para poder gestionar los inventarios.	
Entrada/Pasos de Ejecución: El usuario se autentifica y puede acceder a Listar Inventarios-Existencias, Listar Inventarios-Entradas, Listar Inventarios-Salidas.	
Resultado Esperado: Gestionar los inventarios.	
Evaluación de la Prueba: La prueba se concluyó satisfactoriamente.	

2.7.5 Tarjetas CRC

Por último, se hacen las Tarjetas CRC (Clases, Responsabilidades y Colaboración), confeccionadas para facilitar la comunicación y documentar los resultados. Estas permiten que el equipo completo contribuya en la tarea de diseño. Una tarjeta CRC representa un objeto, por lo tanto, es una clase, cuyo nombre se pone en forma de título en la tarjeta, los atributos y las responsabilidades más significativas se colocan a la izquierda y las clases que están implicadas con cada responsabilidad a la derecha, en la misma línea que su requerimiento correspondiente. Para una mejor comprensión de estas se decide agruparlas por historias de usuarios.

Nombre de la clase: Balance.

Tipo de la clase: Lógica del negocio.	
Responsabilidades:	Colaboradores:
Listar los Balances.	models.BalanceContable obj_balance
ImportacionBalancePlan	guardar_a_bd

Nombre de la clase: ChartCuentas	
Tipo de la clase: Recolecta de datos de cuentas para mostrar en los gráficos	
Responsabilidades:	Colaboradores:
Obtener datos de la Api (Cuentas de la Empresa). Validar datos de forma correcta y prepara una estructura de datos compatible con la librería ChartJS.	getCuentasXCobrar obj_getCuentasXPagar

Nombre de la clase: ChartVentas	
Tipo de la clase: Recolecta de datos de ventas para mostrar en los gráficos.	
Responsabilidades:	Colaboradores:

Obtener datos de la Api (Ventas por Facturación). Validar datos de forma correcta y prepara una estructura de datos compatible con la librería ChartJS.	getventas_facturacion

Nombre de la clase: Filterset	
Tipo de la clase: Filtrado de datos.	
Responsabilidades:	Colaboradores:
Se encarga de construir el filtrado de los datos mediante la construcción de una estructura que permite el uso de las clases proporcionadas por el framework para el filtrado de datos.	InvetarioFilter BalanceContableFilter CtasXEdadesFilter VentasResumenFilter VentasResumenDetallesFilter MovimientosInventarioFilter VentasPtosVentaFilter

La mayoría de las prácticas propuestas por XP no son novedosas, sino que en alguna forma ya habían sido propuestas en ingeniería de software e incluso demostrado su valor en la práctica. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

Muchas personas ven esta metodología como algo innovador, las ideas que propone no son nuevas, pero XP agrupa un conjunto de buenas prácticas y las lleva al extremo.

La innovación de XP consiste en agrupar todas las prácticas, que se lleven a cabo y se soporten entre sí en el mayor grado posible.

Si bien XP es la metodología ágil de más renombre en la actualidad, se diferencia de las demás metodologías que forman este grupo en un aspecto en particular: el alto nivel de disciplina de las personas que participan en proyecto.

2.8 Consideraciones finales del capítulo

En el capítulo se definen elementos fundamentales en el proceso de creación del sistema como son la arquitectura usada y los requisitos funcionales del sistema, presentados en forma de Historias de Usuario, tal como dicta XP, metodología elegida para el proyecto. Mediante el empleo de XP se potenciaron las relaciones interpersonales, clave para el éxito para el desarrollo de software, elevando los conocimientos de los desarrolladores, y propiciando un buen clima de trabajo.

Capítulo 3. Descripción de la propuesta de solución

En este capítulo se presenta el diseño de la base de datos y se describe la arquitectura del sistema. Además, se definen los diagramas de clases de diseño y tanto el diagrama de despliegue como el de componentes.

Para el desarrollo de la aplicación web BI-Versat se involucra el uso de tecnologías de desarrollo ágil, paradigmas orientados a objetos y filosofías inherentes al software libre. Se hace uso de herramientas disponibles en la internet de forma gratuita; Postman, Django, Python, Bootstrap, y jQuery. El proceso de desarrollo con dichas herramientas se reduce considerablemente ya que cubren muchos de los aspectos que hasta hace algunos años dependían de mucho esfuerzo de programación y costos elevados en aspectos como la robustez, seguridad y compatibilidad.

3.1 Diseño de la Base de Datos

Los modelos de datos definen la manera en que se modela la estructura lógica de una base de datos. Son entidades fundamentales para introducir la abstracción en una base de datos. Definen cómo los datos se conectan entre sí y cómo se procesan y almacenan dentro del sistema. Un modelo de datos puede ser concreto o abstracto, y están representados por la notación de modelado de datos, que a menudo se presenta en formato gráfico. Su enfoque principal es apoyar y ayudar a los sistemas de información mostrando el formato y la definición de los diferentes datos involucrados. También ayudan a evitar la redundancia de datos. La información almacenada en los modelos de datos es de gran importancia para las empresas porque dicta las relaciones entre las tablas de la base de datos, las claves externas y los eventos involucrados (Govea, 2020).

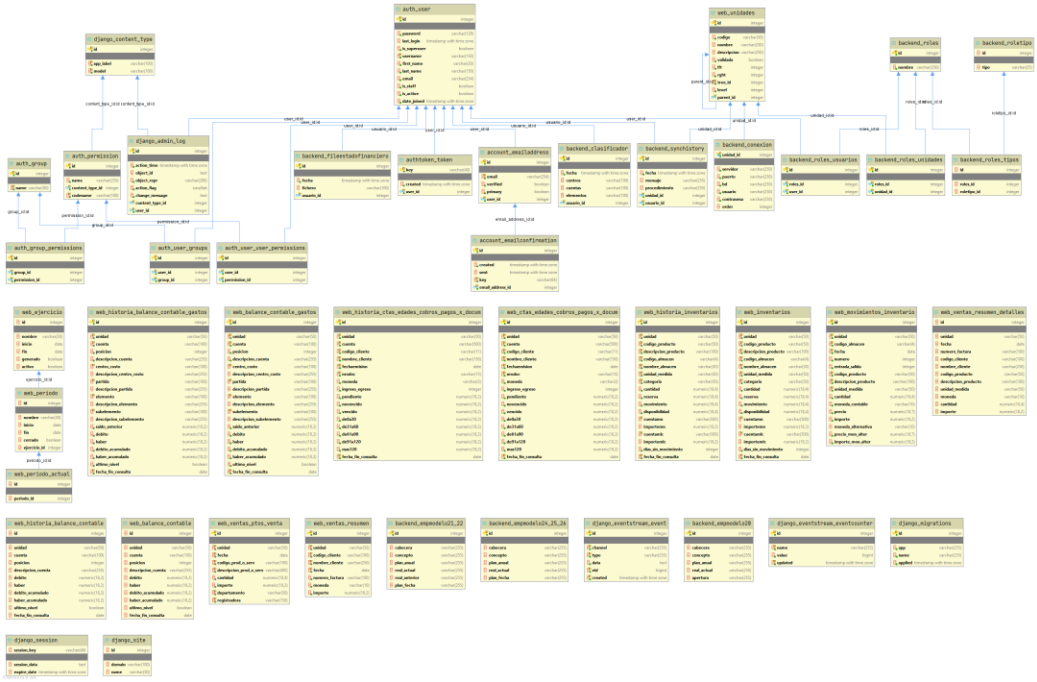


Ilustración 6 Diseño de la Base de Datos

Fuente: Elaboración propia

3.2 Diagrama de Clases del Diseño

Un Diagrama de Clases de Diseño muestra la especificación para las clases de software de una aplicación. Incluye la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Navegabilidad.
- Dependencias.

Ilustración 7 Diagrama de Clases del Diseño

Fuente: Elaboración propia

A diferencia del Modelo Conceptual, un Diagrama de Clases de Diseño muestra definiciones de entidades de software más que conceptos del mundo real.(Govea, 2020)

3.3 Capturas de pantalla de fragmentos de la aplicación

Las gráficas siguientes muestran como la información contable relevante de la empresa puede ser apreciada por los directivos. Estos esquemas permiten una mejor evaluación de los indicadores claves para el desarrollo de la empresa Datazucar.

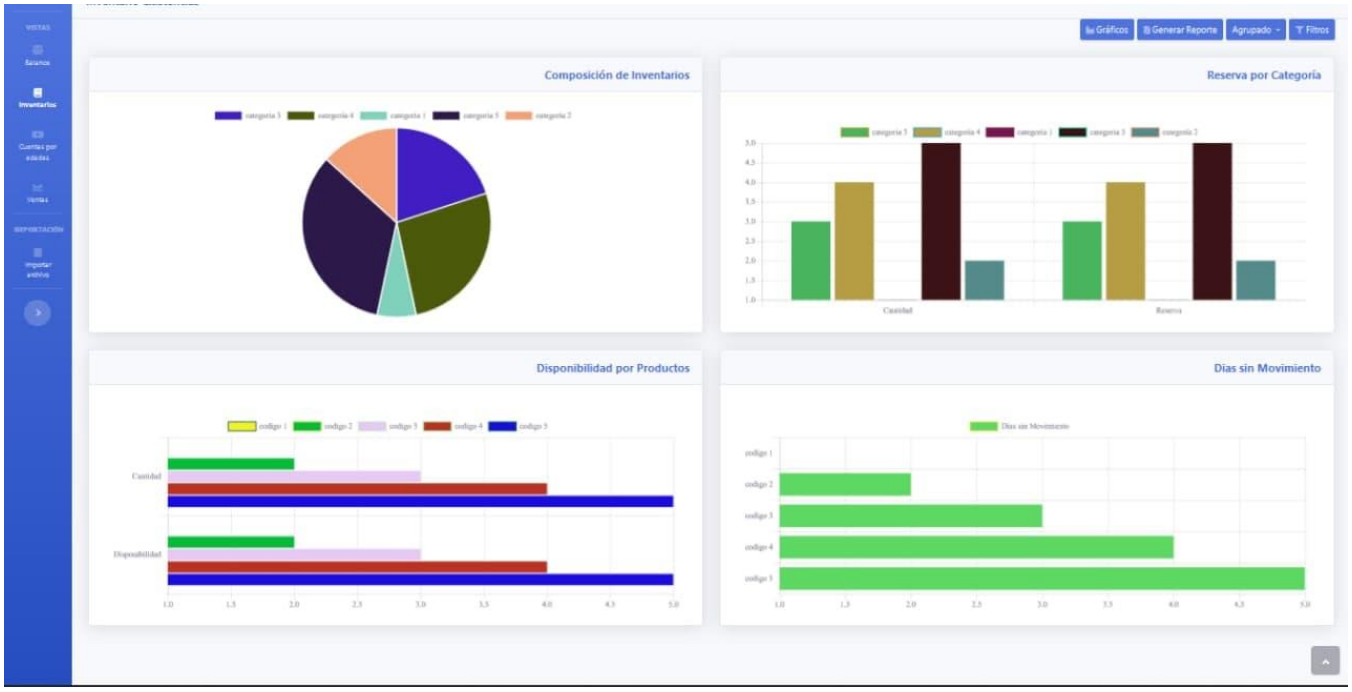


Ilustración 8 Gráficos Inventario-Existencias
Fuente: Elaboración propia

3.3.1 Balance de Comprobación

El balance de comprobación es un instrumento financiero que se utiliza para visualizar la lista del total de los débitos y de los créditos de las cuentas, junto al saldo de cada una de ellas (ya sea deudor o acreedor). De esta forma, permite establecer un resumen básico de un estado financiero.

El balance de comprobación refleja la contabilidad de una empresa u organización en un determinado período. Por eso, el balance actúa como base a la hora de preparar las cuentas anuales.

La elaboración de un balance de comprobación comienza con la realización de las sumas de las anotaciones de cada cuenta, tanto del debe como del haber. En el paso siguiente, se obtiene el saldo de cada cuenta (la diferencia del debe y el haber). Por último, las sumas y los saldos obtenidos se trasladan al balance.

Por lo general, el balance de comprobación es un documento voluntario para el empresario, aunque recomendable para que este pueda conocer con precisión el estado financiero de la

empresa sin necesidad de arrastrar errores hasta la elaboración de las cuentas anuales (Ramírez, 2018).

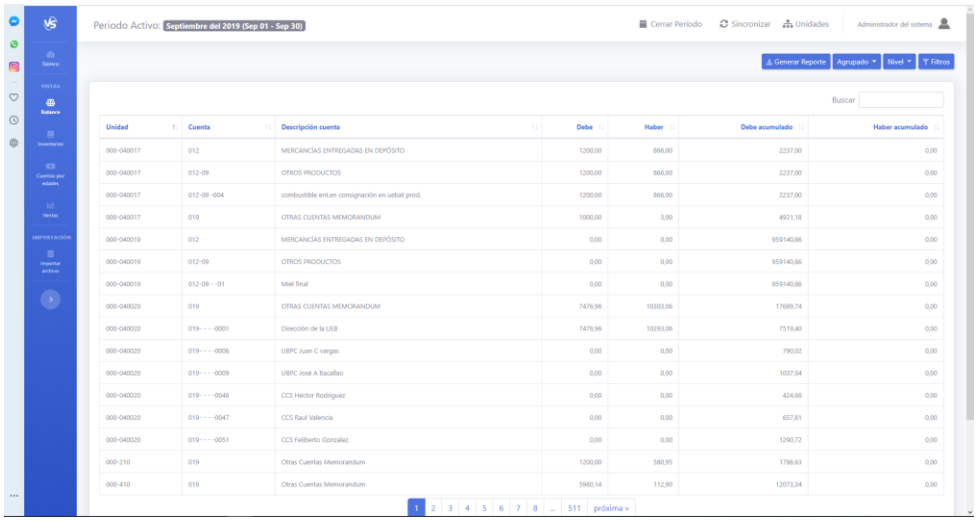
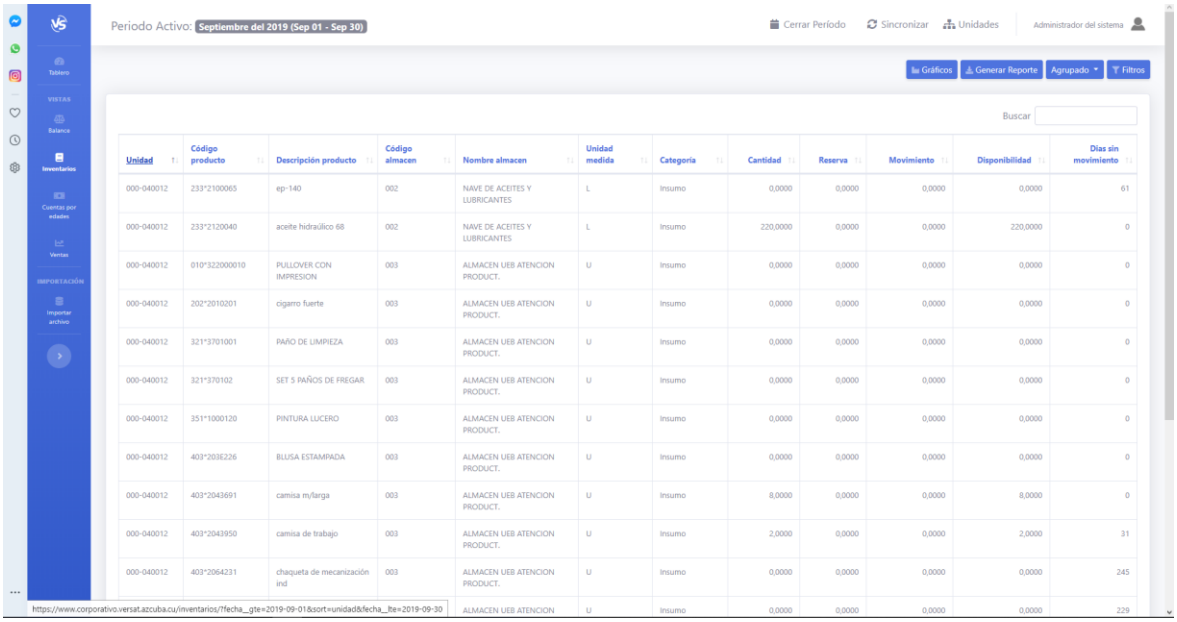


Ilustración 9 Balance de Comprobación

Fuente: Elaboración propia

3.3.2 Inventario-Existencias

El inventario de existencias es un registro de todas las mercancías que forman parte de la actividad económica de una empresa. Así, se incluyen los bienes disponibles para la venta, los insumos y los productos que aún no finalizan su fabricación.



Unidad	Código producto	Descripción producto	Código almacen	Nombre almacen	Unidad medida	Categoría	Cantidad	Reserva	Movimiento	Disponibilidad	Dias sin movimiento
000-040012	233*2100065	ep-140	002	NAVE DE ACEITES Y LUBRICANTES	L	Insumo	0,0000	0,0000	0,0000	0,0000	61
000-040012	233*2120040	aceite hidraúlico 68	002	NAVE DE ACEITES Y LUBRICANTES	L	Insumo	220,0000	0,0000	0,0000	220,0000	0
000-040012	010*322000010	PULLOVER CON IMPRESION	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	0,0000	0,0000	0,0000	0,0000	0
000-040012	202*2010201	cigarro fuerte	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	0,0000	0,0000	0,0000	0,0000	0
000-040012	321*3701001	PAÑO DE LIMPIEZA	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	0,0000	0,0000	0,0000	0,0000	0
000-040012	321*370102	SET 5 PAÑOS DE FREGAR	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	0,0000	0,0000	0,0000	0,0000	0
000-040012	351*1000120	PINTURA LUCERO	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	0,0000	0,0000	0,0000	0,0000	0
000-040012	403*203E226	BLUSA ESTAMPADA	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	0,0000	0,0000	0,0000	0,0000	0
000-040012	403*2043691	camisa m/larga	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	8,0000	0,0000	0,0000	8,0000	0
000-040012	403*2043950	camisa de trabajo	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	2,0000	0,0000	0,0000	2,0000	31
000-040012	403*2064231	chaqueta de mecanización ind	003	ALMACEN UEB ATENCION PRODUCT.	U	Insumo	0,0000	0,0000	0,0000	0,0000	245
				ALMACEN UEB ATENCION	U	Insumo	0,0000	0,0000	0,0000	0,0000	229

Ilustración 10 Inventario-Existencias

Fuente: Elaboración propia

Este inventario tiene como objetivo mantener la integridad de las propiedades de la compañía y anotarlas correctamente en su balance de situación.

3.3.3 Inventario Entradas-Salidas

Entradas

La entrada de mercancía es un proceso de inventario que le permitirá registrar en el sistema, todo aquel producto que llegue al centro de costo sea cual sea la razón de su llegada.

Tipos de entradas

Existen dos formas principales en que la mercancía puede entrar al sistema de inventario:

- Entradas de almacén por compras (Ingreso de facturas de compra a proveedores)
- Entradas por otros conceptos (Conceptos definidos por el usuario: entrada por obsequio, etc.)

Salidas

Las salidas de mercancías son una operación de inventario para el registro de salida de un producto del centro de costo. Esta salida puede darse por un concepto predefinido por el usuario. No sustituye el proceso de facturación por ventas o devoluciones a proveedores (Gómez, 2010).

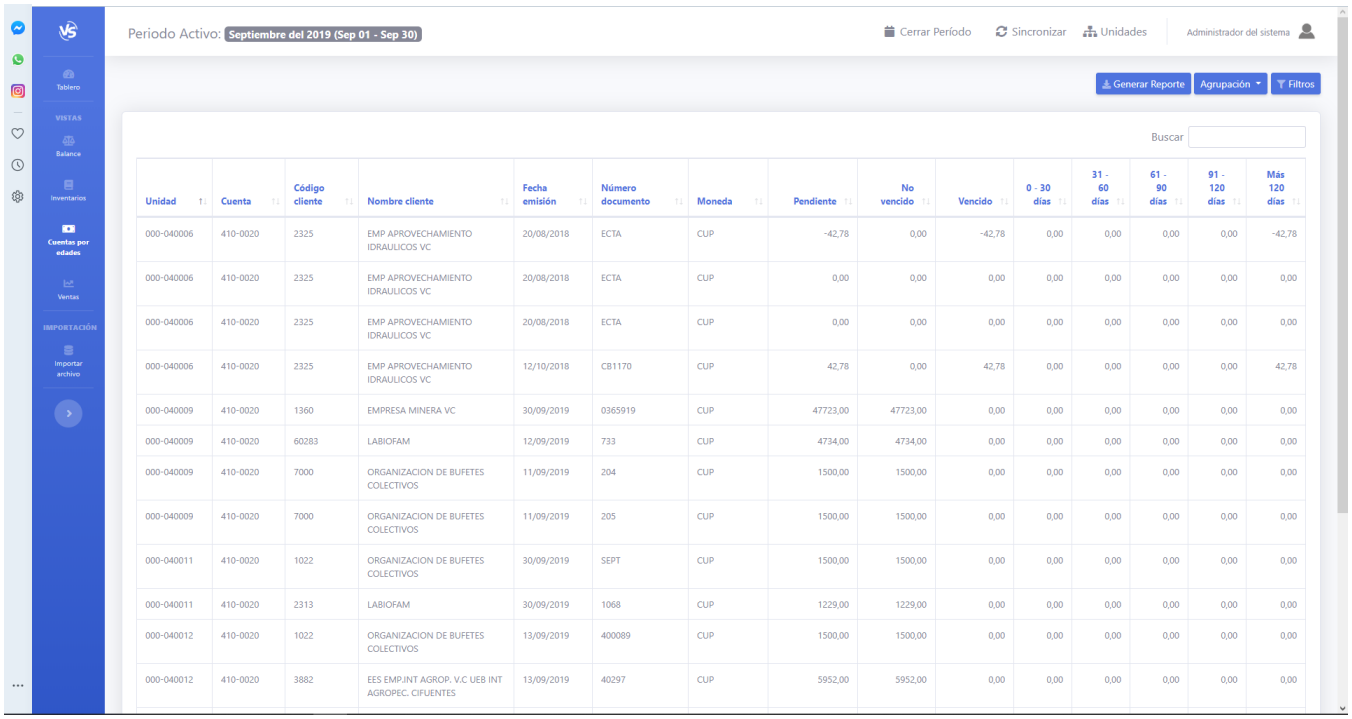
3.3.4 Cuentas por Edades

Cuentas por Cobrar

Las cuentas por cobrar representan derechos de cobro que se espera recibir en efectivo. Las cuentas por cobrar representan sumas que adeudan las entidades a una empresa por la venta de productos y servicios. En la mayoría de las entidades comerciales, las cuentas por cobrar normalmente se generan al emitir una factura y enviarla al cliente por correo o de manera electrónica, y el cliente, a su vez, debe liquidarla dentro de un período de tiempo establecido que se denomina términos de crédito o términos de pago.

Cuentas por Pagar

Las cuentas por pagar (CP) representan la suma que se adeuda en una fecha específica por la compra de productos o servicios. Las cuentas por pagar se registran al momento en que se aprueba una factura para pago y se registran en el Libro Mayor General (o en el libro auxiliar de las CP) como un pasivo, pendiente de pago o abierta, debido a que no ha sido liquidada. Las cuentas por pagar generalmente se clasifican como Cuentas por Pagar Comerciales y Cuentas por Pagar de Gastos Diversos. Algunos ejemplos comunes de Cuentas por Pagar de Gastos son publicidad, viajes, entretenimiento, suministros de oficina, y servicios. Las CP son una forma de crédito que los proveedores ofrecen a sus clientes al permitirles pagar por un producto o servicio después de que este haya sido recibido (Gómez, 2010).



Unidad	Cuenta	Código cliente	Nombre cliente	Fecha emisión	Número documento	Moneda	Pendiente	No vencido	Vencido	0 - 30 días	31 - 60 días	61 - 90 días	91 - 120 días	Más 120 días
000-040006	410-0020	2325	EMP APROVECHAMIENTO IDRAULICOS VC	20/08/2018	ECTA	CUP	-42,78	0,00	-42,78	0,00	0,00	0,00	0,00	-42,78
000-040006	410-0020	2325	EMP APROVECHAMIENTO IDRAULICOS VC	20/08/2018	ECTA	CUP	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040006	410-0020	2325	EMP APROVECHAMIENTO IDRAULICOS VC	20/08/2018	ECTA	CUP	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040006	410-0020	2325	EMP APROVECHAMIENTO IDRAULICOS VC	12/10/2018	CB1170	CUP	42,78	0,00	42,78	0,00	0,00	0,00	0,00	42,78
000-040009	410-0020	1360	EMPRESA MINERA VC	30/09/2019	0365919	CUP	47723,00	47723,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040009	410-0020	60283	LABIOFAM	12/09/2019	733	CUP	4734,00	4734,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040009	410-0020	7000	ORGANIZACION DE BUFETES COLECTIVOS	11/09/2019	204	CUP	1500,00	1500,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040009	410-0020	7000	ORGANIZACION DE BUFETES COLECTIVOS	11/09/2019	205	CUP	1500,00	1500,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040011	410-0020	1022	ORGANIZACION DE BUFETES COLECTIVOS	30/09/2019	SEPT	CUP	1500,00	1500,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040011	410-0020	2313	LABIOFAM	30/09/2019	1068	CUP	1229,00	1229,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040012	410-0020	1022	ORGANIZACION DE BUFETES COLECTIVOS	13/09/2019	400089	CUP	1500,00	1500,00	0,00	0,00	0,00	0,00	0,00	0,00
000-040012	410-0020	3882	EES EMP.INT AGROP. V.C UEB INT AGROPEC. CIFUENTES	13/09/2019	40297	CUP	5952,00	5952,00	0,00	0,00	0,00	0,00	0,00	0,00

Ilustración 11 Cuentas por Edades

Fuente: Elaboración propia

3.3.5 Ventas

Las ventas, en economía, son la entrega de un determinado bien o servicio bajo un precio estipulado y a cambio de una contraprestación económica en forma de dinero por parte de un vendedor o proveedor.

Las ventas suponen la obtención de una ganancia económica desde el punto de vista del vendedor. Este agente económico ofrece su producto a potenciales compradores, que se harán con el mismo tras compensarle con dinero sobre un precio conocido previamente.

La realización de ventas supone el núcleo de la actividad económica de un gran margen del espectro económico, donde los actores económicos obtienen ganancias dinerarias tras la entrega de un producto o servicio en el que se especializan (Gómez, 2010).

3.3.6 Puntos de Venta

Un punto de venta es aquel espacio, físico (tienda) o virtual (ecommerce), en el que una empresa establece contacto con su cliente potencial, pudiendo desarrollarse una transacción

de compraventa. Este espacio, puede existir físicamente, pudiendo el cliente acudir a él. Por otro lado, puede tratarse de un espacio virtual, un portal web, donde se halla un negocio online.

El punto de venta, además, suele ser el principal canal de distribución del producto o servicio brindado. Por ello, se dice que es un aspecto determinante para una empresa, ya que de este depende una gran parte del ingreso que la empresa recibe (Gómez, 2010).

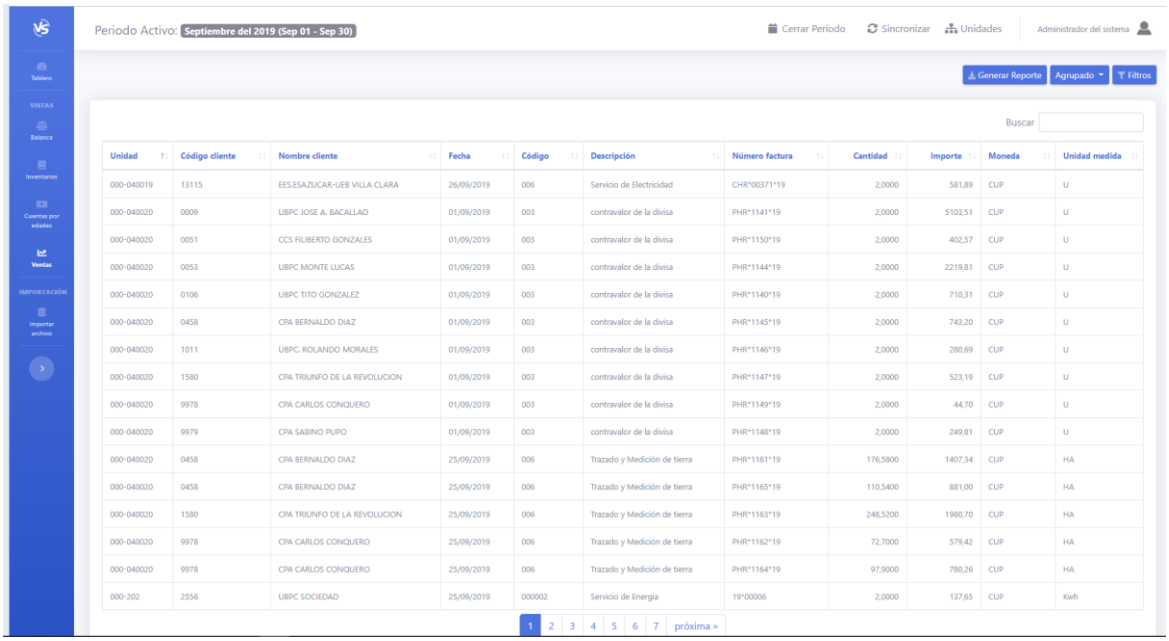


Ilustración 12 Puntos de ventas

Fuente: Elaboración propia

3.4 Errores comunes en el BI-Versat

El diseño de errores es uno de los aspectos que muchas veces queda olvidado en el diseño de plataformas digitales. ¿Qué verá el usuario cuando el sistema falle? ¿Cómo sabrá que hizo algo mal? Estos elementos de orientación y ayuda no son algo menor. Si un usuario se encuentra con un error y no sabe qué hacer, lo más probable es que abandone el sitio o aplicación en cuestión y no vuelva a visitarlo, quedándose con la idea de que la plataforma no funciona.

Los errores se pueden producir por una falla del sistema o por una acción del usuario, por lo que hay dos niveles o áreas en las que se deben diseñar las respuestas:

- Sitio web: Los mensajes de error aparecen en páginas y elementos no encontrados o no disponibles.
- Formularios: El mensaje se activa cuando se acaba el tiempo para enviar una información, se introducen datos incorrectamente o se dejan campos vacíos.

Cuando el sistema falla, lo más importante es ofrecer una solución alternativa al usuario. Esto se puede lograr incluyendo enlaces a otras páginas con contenido relacionado, ofreciendo un buscador y/o facilitando información de contacto.

En el caso de los formularios, el diseño y el mensaje deben orientarse a destacar el error y enseñarle al usuario cómo debe completar la información. Para lograr esto se debe indicar claramente dónde ocurrió el problema, destacando visualmente el área y agregando una alerta escrita.

Además, para prevenir que el usuario cometa errores, se pueden incluir instrucciones que indiquen cómo introducir los datos de forma correcta en cada campo (Savoy, 2015).

En el sistema BI-Versat es posible ver ciertas alarmas o mensajes de error en diferentes acciones que el usuario ejecute, ya sea por equivocación o indicaciones del sistema.

Una de las acciones más frecuentes es la autenticación de los usuarios dentro del sistema, por lo tanto, es común cometer errores:

Ilustración 13 Credenciales Inválidas de un usuario

Fuente: Elaboración propia

En la imagen anterior se puede ver un mensaje de error donde se le indica al usuario que su usuario o contraseña está incorrecto.

En un sistema contable como el BI-Versat es necesario hacer análisis dentro de un período específico, en un ejercicio contable específico y una vez concluido el estudio, el usuario puede pasar al siguiente período contable, pero si el usuario se encuentra en el último período se lanzaría el siguiente error:

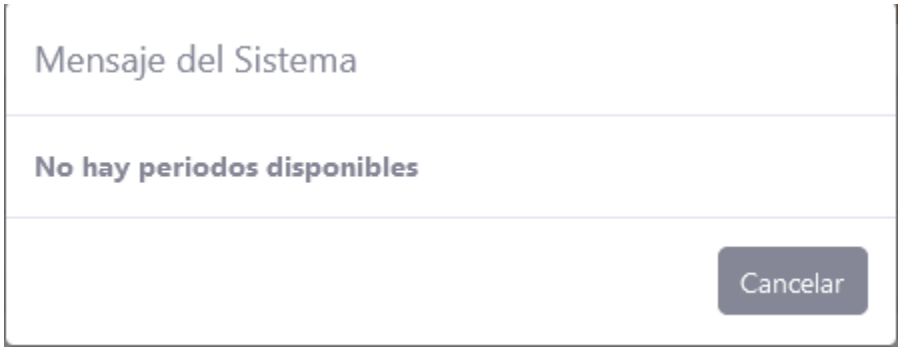


Ilustración 14 Períodos no Disponibles
Fuente: Elaboración propia

De esta forma el usuario puede estar consciente de que el período contable en el que se encuentra es el último dentro de la aplicación.

Una de las herramientas principales del sistema son sus búsquedas filtradas para obtener información necesaria para el usuario. Muchas veces no existen los resultados que el usuario solicita, si este es el caso, dentro de la tabla se puede visualizar un mensaje como se muestra en el ejemplo:

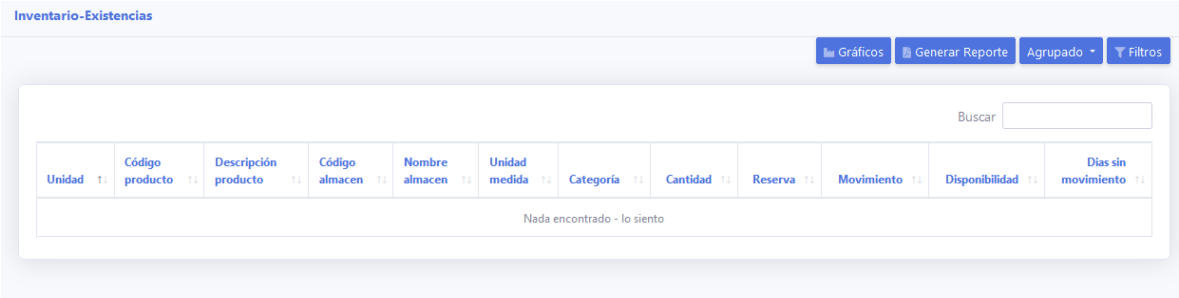


Ilustración 15 Tabla sin resultados

Fuente: Elaboración propia

Otra de las funciones básicas del BI-Versat es su capacidad para obtener información en tiempo real del sistema contable Versat Sarasola, para ello el usuario tiene la opción de sincronizar las unidades contables y así obtener la información actualizada dentro del sistema, pero en ocasiones existen errores de conexión que impiden esta actualización, en este caso el usuario verá un mensaje como el siguiente:

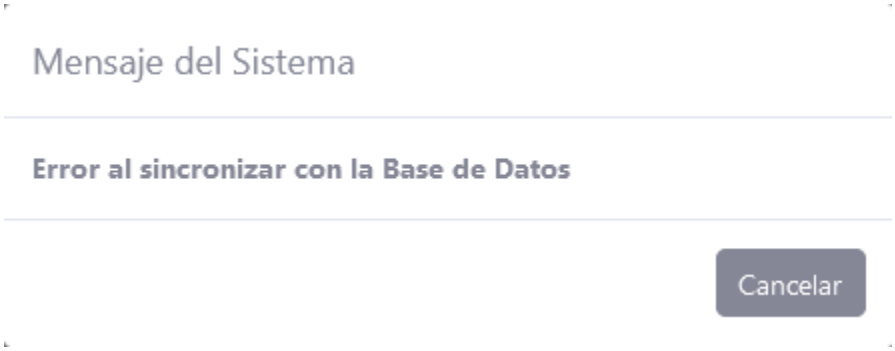


Ilustración 16 Error al sincronizar los datos
Fuente: Elaboración propia

El BI-Versat dispone de otra forma para obtener la información si por alguna razón específica el usuario no desea utilizar la opción de sincronizar las unidades contables, y es importándola directamente en el sistema con un archivo en formato csv, por tanto, el sistema está preparado para mandar un mensaje advirtiendo si el archivo no cumple con los requisitos para realizar la importación:

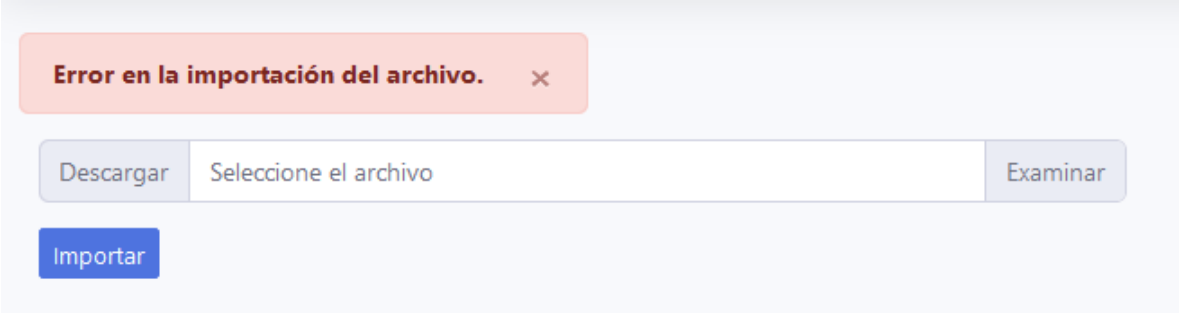


Ilustración 17 Importación Fallida
Fuente: Elaboración propia

Estos son algunos ejemplos de los errores más comunes que el sistema contable BI-Versat es capaz de detectar y de esta forma alertar y guiar al cliente.

Los mensajes de error son una oportunidad para educar a los usuarios y enseñarles aspectos de la tecnología o una interfaz particular. Además, muchas veces abren un espacio para mostrar otros servicios o contenidos como alternativas y/o complementos para solucionar el problema.

Las claves para el correcto diseño de estos apoyos están en la empatía, la experiencia y la retroalimentación que los mismos usuarios entregan, por lo que los estudios de usabilidad son de gran ayuda en este proceso. El usuario debe recibir ayuda al realizar tareas, aun cuando cometan errores o el sistema presente fallas (Savoy, 2015).

3.5 Arquitectura del Sistema. Principios y Patrones de Diseño utilizados

Los patrones de software aportan una solución previamente probada y documentada a problemas comúnmente dados en el ámbito de la programación. Las ventajas de su empleo son las siguientes: ahorro de tiempo a la hora de desarrollar un sistema, establecen un lenguaje común para que distintos desarrolladores puedan comunicarse fácilmente sabiendo en cada momento de qué elemento está hablando cada desarrollador y permiten al autor estar seguro de la validez del código implementado (Caldera, 2017).

3.5.1 Patrón Modelo – Vista – Controlador

El patrón Modelo – Vista – Controlador es un patrón de diseño que posee un objetivo claro: separar los datos de la aplicación y el estado de esta, frente al mecanismo usado para representar el modelo al usuario final.

Como patrón de diseño, se basa en la reutilización de código y en la separación de las distintas responsabilidades de la aplicación, con lo que se pretende un mejor mantenimiento del código, así como ayudar al programador con su tarea de implementación. Como el resto de los patrones, también tiene como ocupación simplificar la comunicación entre los desarrolladores de un equipo de trabajo, en el momento del desarrollo.

Cabe destacar que este patrón tiene tres componentes principales, que son: el modelo, la vista y el controlador. De tal manera que existe, por un lado, la encapsulación de los datos, la interfaz

o vista por otro y por último la lógica interna de la aplicación o el controlador. Se pasa a comentar estos tres componentes fundamentales:

Modelo

En el modelo se encuentra el núcleo de la funcionalidad de la aplicación y se encapsula el estado de esta. Esta capa trabaja con los datos, y tiene los mecanismos necesarios para acceder a la información y actualizar su estado. Esta parte es independiente de las otras dos (controlador y vista), ya que no contiene enlaces a ellas.

Vista

Esta capa es la encargada de la presentación de los datos del modelo. Para poder realizar correctamente la presentación de los datos, la vista puede acceder al modelo, mediante una referencia que tiene, pero no puede modificar su estado.

Controlador

El controlador es el encargado de reaccionar ante las peticiones realizadas por el usuario de la aplicación, ejecutando la acción adecuada y creando el modelo pendiente. Es una capa que sirve de enlace entre las vistas y los modelos, puesto que es el vínculo entre ellas para implementar las necesidades de la aplicación web.

En cuanto a la arquitectura de Django, cabe comentar que fue diseñado para que exista un acoplamiento débil entre las distintas partes de la aplicación y una clara separación de estas. Por tanto, es posible realizar cambios en una parte específica de la aplicación sin afectar a las demás piezas de esta. Django, por lo tanto, tiene una implementación especial del Modelo – Vista – Controlador (Caldera, 2017).

En las aplicaciones desarrolladas con Django, debido a que el Controlador es manejado por el mismo Framework, y la parte más significativa se la llevan los modelos (*Models*), vistas (*Views*) y plantillas (*Templates*), Django es conocido también como un framework MTV (*Models – Views – Templates*).

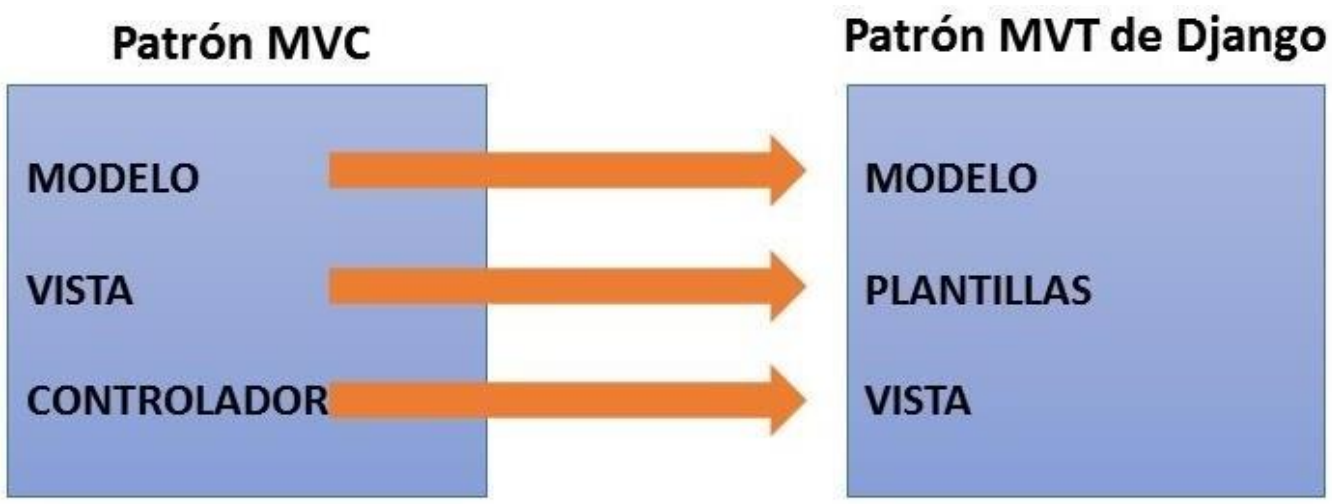


Ilustración 18 Modelo de correspondencia MVC-MVT
Fuente: (Caldera, 2017)

En Django, se conoce al Controlador como “vista” y a la vista como “plantilla”. La vista describe el dato que se muestra al usuario de la aplicación, por lo que se centran más en qué dato ve y no cómo lo ve. Por lo tanto, lo que decide qué dato ve el usuario es la vista, y cómo lo ve se delega a la plantilla. La vista, es la capa de la lógica de negocios, donde se encuentra la inteligencia que accede al modelo y la delega a la plantilla adecuada. Por último, el modelo no cambia con respecto a la explicación del patrón Modelo – Vista – Controlador convencional, sigue siendo la capa de acceso a la base de datos, que contiene cómo acceder a ellos, cuál es su comportamiento, cómo validarlos y las relaciones entre los datos (Caldera, 2017).

A continuación, se muestra un esquema de la colaboración existente entre los distintos componentes del patrón:

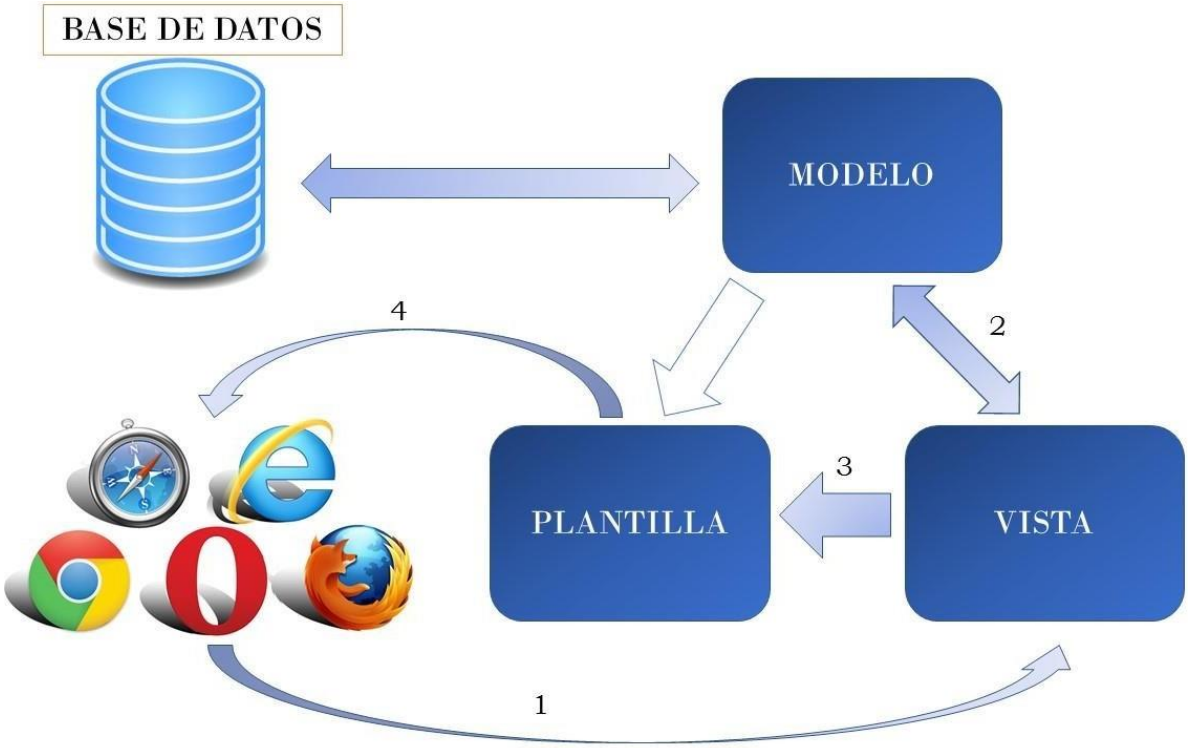


Ilustración 19 Diagrama de colaboración de capas

Fuente: (Caldera, 2017)

I. El cliente desde su navegador realiza una solicitud de un servicio web de la aplicación, por lo que se pone en contacto con la capa de la vista.

II. La capa de la vista necesita tener contacto directo con la del modelo ya que es de ella de donde cogerá los datos que solicita el usuario. Por lo que entra en acción la capa del modelo. Esta última, gracias a las consultas a la base de datos devuelve los objetos a la capa de la vista, que ella ya se encargará de filtrar y devolver explícitamente los que el usuario ha solicitado en la petición web.

III. El siguiente paso es que la capa de la vista envíe a la de la plantilla los datos requeridos por el usuario, tras previo procesamiento de estos. En este paso, ha actuado indirectamente el modelo, ya que es quien proporcionó los datos a la vista.

IV. En último lugar, la capa de la plantilla es la encargada de cómo se deben mostrar los datos al usuario. Por lo que, con los datos recibidos de la vista, los incluye en su

respectiva plantilla para que los navegadores puedan procesarlo y el usuario visualice correctamente la información.

3.6 Diagramas

El Lenguaje de Modelación Unificado, (UML, por sus siglas en inglés) es la norma ISO de aplicación general para el desarrollo de software y arquitecturas de sistemas complejos. Este lenguaje de modelado utiliza diferentes tipos de diagramas para los procesos de planificación y desarrollo en la programación orientada a objetos.

En la versión actual (UML 2.5) se clasifican 14 tipos de diagramas, que se dividen a grandes rasgos en de comportamiento y estructurales. Dentro de estos diagramas se encuentran el de despliegue y el de componentes (Menéndez, 2005).

3.6.1 Diagrama de Despliegue

Un diagrama de despliegue muestra la arquitectura de ejecución de un sistema, incluyendo nodos como entornos de ejecución de hardware o software, y el middleware que los conecta. Se utilizan normalmente para visualizar el hardware y el software físico de un sistema. Permite entender cómo el sistema se despliega físicamente en el hardware.

Los diagramas de despliegue ayudan a modelar la topología de hardware de un sistema en comparación con otros tipos de diagramas UML, que en su mayoría esbozan los componentes lógicos de un sistema (Menéndez, 2005).

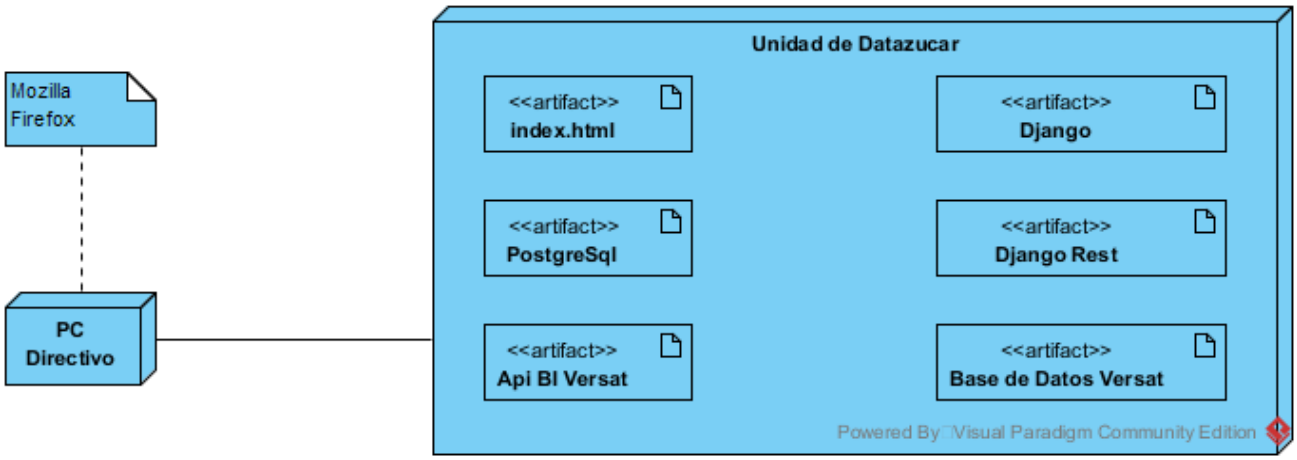


Ilustración 20 Diagrama de Despliegue

Fuente: Elaboración propia

3.6.2 Diagrama de Componentes

El diagrama de componentes muestra la relación entre componentes de software, sus dependencias, su comunicación, su ubicación y otras condiciones. Este diagrama proporciona una vista de alto nivel de estos dentro de un sistema. Pueden ser un componente de software, como una base de datos o una interfaz de usuario; o un componente de hardware como un circuito, microchip o dispositivo; o una unidad de negocio como un proveedor, nomina o envío.

Un componente está formado por numerosas clases y paquetes de clases internos. También se puede crear a partir de una colección de componentes más pequeños. Mientras que otros diagramas UML describen la funcionalidad de un sistema, estos se utilizan para modelar los componentes que ayudan a hacer esas funcionalidades, representando la forma en la que estos se organizan y sus dependencias (Govea, 2020).

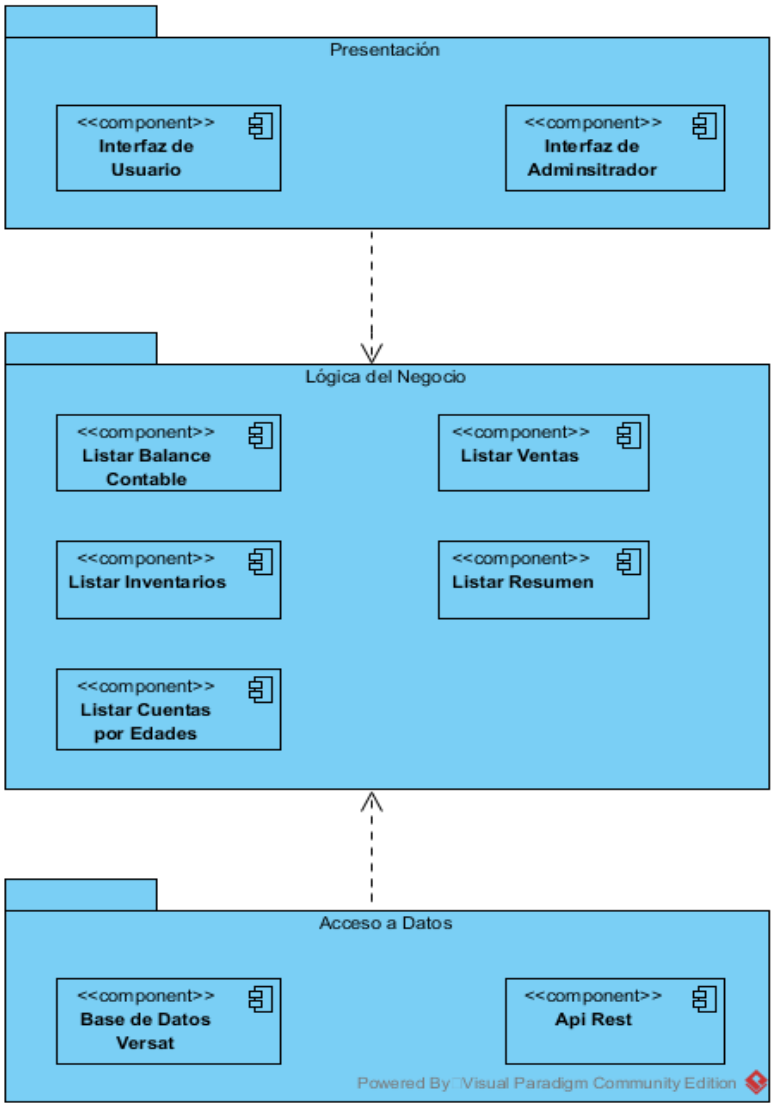


Ilustración 21 Diagrama de Componentes

Fuente: Elaboración propia

3.7 Seudocódigo de algunas funciones del proyecto

El siguiente seudocódigo muestra como el sistema BI Versat procesa la información para ser mostrada al usuario, para ello la función get_obj_inventario recibe la petición request donde se puede obtener la fecha específica, las unidades contables y la opción de recibir la información de forma agrupada o extendida. Teniendo en cuenta todas estas variables se hace una petición a la base datos en la tabla Inventarios donde se busca la información de estos por fecha y unidad, devolviendo una consulta que con las librerías de Django_filters y Django_tables, respectivamente, será procesada para retornar a la vista tres objetos de tipo table, filtro y queryset.

```
Algoritmo Obtener Inventario
    Funcion [table, filtro, queryset] <- get_obj_inventario (request,
unidades, agrupado)
        periodo_actual ← PeriodoActual.objects.first()
        Si periodo_actual y periodo_actual.periodo Entonces
            fecha_fin_per_act ← periodo_actual.periodo.fin.__str__()
        SiNo
            fecha_fin_per_act ← ''
        Fin Si
        Si agrupado Entonces
            Si request.GET contiene fecha__lte Entonces
                fecha_lte = request.GET['fecha__lte']
                Si fecha_fin_per_act ≠ fecha_lte Entonces
                    queryset ← HistoriaInventario.objects.filter(
                    unidad__in=unidades,
                    fecha_fin_consulta=fecha_lte
                    ).values(
                    'codigo_producto',
                    'descripcion_producto',
                    'unidad_medida',
                    'categoria',
                    ).annotate(
                    cantidad=Sum('cantidad'),
                    reserva=Sum('reserva'),
                    movimiento=Sum('movimiento'),
                    disponibilidad=Sum('disponibilidad'),
                    )
```

```
                SiNo
                    queryset ← Inventario.objects.filter(
                    unidad__in=unidades,
                    fecha_fin_consulta=fecha_lte
                    ).values(
                    'codigo_producto',
                    'descripcion_producto',
                    'unidad_medida',
                    'categoria',
                    ).annotate(
                    cantidad=Sum('cantidad'),
                    reserva=Sum('reserva'),
                    movimiento=Sum('movimiento'),
                    disponibilidad=Sum('disponibilidad'),
                    )
                Fin Si
            SiNo
                queryset ←
Inventario.objects.filter(unidad__in=unidades).values(
                'codigo_producto',
                'descripcion_producto',
                'unidad_medida',
                'categoria',
                ).annotate(
                cantidad=Sum('cantidad'),
                reserva=Sum('reserva'),
                movimiento=Sum('movimiento'),
                disponibilidad=Sum('disponibilidad'),
                )
            Fin Si
            filtro ← Funcion query <- InvetarioFilter ( request.GET,
queryset=queryset )
            FinFuncion
            table ← Funcion query <- InventarioAgrupadoTable ( filtro.qs,
order_by=request.GET.get('sort') )
            FinFuncion
            retornar [table, filtro, queryset]
        SiNo
            Si request.GET contiene fecha__lte Entonces
                fecha_lte ← request.GET['fecha__lte']
                Si fecha_fin_per_act ≠ fecha_lte Entonces
                    queryset ←
HistoriaInventario.objects.filter(unidad__in=unidades,
fecha_fin_consulta=fecha_lte)
                SiNo
```

```
                    queryset ←
Inventario.objects.filter(unidad__in=unidades, fecha_fin_consulta=fecha_lte)
                Fin Si
            SiNo
                queryset ← Inventario.objects.filter(unidad__in=unidades)
            Fin Si
            filtro ← Funcion query <- InvetarioFilter ( request.GET,
queryset=queryset )
        FinFuncion
        table ← Funcion query <- InventarioAgrupadoTable ( filtro.qs,
order_by=request.GET.get('sort') )
        FinFuncion
        retornar [table, filtro, queryset]
        Fin Si
    Fin Funcion
FinAlgoritmo
```

En este ejemplo el proceso es similar al anterior, con la diferencia que la función recibe una variable extra llamada tipo para indicar el movimiento de inventario que se solicita y una vez concluido el proceso de buscar dicha información se devuelven dos objetos de tipo table y filtro.

```
Algoritmo Obtener Inventario Movimiento
    Funcion [table, filtro] <- get_obj_inventario (request, unidades,
agrupado, tipo)
        Si agrupado Entonces
            Si request.GET contiene fecha__lte y request.GET contiene
fecha__gte Entonces
                f ← MovimientosInventario.objects.filter(
                unidad__in=unidades,
                entrada_salida=tipo,
                fecha__lte=request.GET['fecha__lte'],
                fecha__gte=request.GET['fecha__gte'],
                )
            SiNo
                Si request.GET contiene fecha__lte y no request.GET contiene
fecha__gte Entonces
                    f ← MovimientosInventario.objects.filter(
                    unidad__in=unidades,
                    entrada_salida=tipo,
                    fecha__lte=request.GET['fecha__lte'],
                    )
                SiNo
                    Si no request.GET contiene fecha__lte y request.GET
contiene fecha__gte Entonces
                        f = MovimientosInventario.objects.filter(
```

```
                        unidad__in=unidades,
                        entrada_salida=tipo,
                        fecha__gte=request.GET['fecha__gte'],
                        )
                    SiNo
                        f = MovimientosInventario.objects.filter(
                        unidad__in=unidades,
                        entrada_salida=tipo,
                        )
                    Fin Si
                Fin Si
            Fin Si
            queryset ← f.values(
                'codigo_producto',
                'descripcion_producto',
                'unidad_medida',
                'moneda_contable',
                'moneda_alternativa',
            ).annotate(
                cantidad=Sum('cantidad'),
                importe=Sum('importe'),
                importe_mon_alter=Sum('importe_mon_alter'),
            )
            filtro ← Funcion query <- MovimientosInventarioFilter (
request.GET, queryset=queryset )
            FinFuncion
            table ← Funcion query <- MovimientosInventarioAgrupadoTable (
filtro.qs, order_by=request.GET.get('sort', 'codigo_producto') )
            FinFuncion
            retornar [table, filtro]
        SiNo
            Si request.GET contiene fecha__lte y request.GET contiene
fecha__gte Entonces
                queryset ← MovimientosInventario.objects.filter(
                    unidad__in=unidades,
                    entrada_salida=tipo,
                    fecha__lte=request.GET['fecha__lte'],
                    fecha__gte=request.GET['fecha__gte'],
                )
            SiNo
                Si request.GET contiene fecha__lte y no request.GET contiene
fecha__gte Entonces
                    queryset ← MovimientosInventario.objects.filter(
                        unidad__in=unidades,
                        entrada_salida=tipo,
                        fecha__lte=request.GET['fecha__lte'],
```

```
                    )
                SiNo
                    Si no request.GET contiene fecha__lte y request.GET
contiene fecha__gte Entonces
                        queryset ← MovimientosInventario.objects.filter(
                            unidad__in=unidades,
                            entrada_salida=tipo,
                            fecha__gte=request.GET['fecha__gte'],
                        )
                    SiNo
                        queryset ←
MovimientosInventario.objects.filter(unidad__in=unidades,
entrada_salida=tipo)
                    Fin Si
                Fin Si
            Fin Si
            filtro ← Funcion query <- MovimientosInventarioFilter (
request.GET, queryset=queryset )
            FinFuncion
            table ← Funcion query <- MovimientosInventarioAgrupadoTable (
filtro.qs, order_by=request.GET.get('sort', 'fecha') )
            FinFuncion
            retornar [table, filtro]
        Fin Si
    Fin Funcion
FinAlgoritmo
```

3.8 Consideraciones finales del capítulo

En el capítulo fue posible profundizar en las funcionalidades que ofrece el sistema. De igual modo para el diseño se modelan varios diagramas, como son el diagrama de despliegue y de componentes, que sirven de guía para la implementación del software.

Capítulo 4. Pruebas y validación de la solución desarrollada

En este capítulo se describe la estrategia de pruebas a realizar y se muestran los resultados de estas aplicadas a la herramienta informática desarrollada, con el objetivo de validar sus funcionalidades.

Unas de las vías más importantes para determinar el estado de la calidad de un producto de software es el proceso de pruebas. Están dirigidas a componentes del sistema en su totalidad, con el objetivo de medir el grado en que cumple con los requerimientos. En ellas se usan casos de prueba, especificados de forma estructurada mediante técnicas. Sus objetivos, métodos y técnicas usadas se describen en el plan de prueba.

Las pruebas son una actividad fundamental en muchos procesos de desarrollo, incluido el de software. Estas permiten detectar la presencia de errores que pudieran generar las entradas o salidas de datos y comportamientos inapropiados durante su ejecución. Es una actividad en la cual el sistema es ejecutado bajo condiciones específicas para demostrar que no tiene la madurez necesaria para ser implantado. Dentro de las actividades que se practican para obtener un software con la madurez necesaria están:

- Revisiones: consiste en que cada integrante del equipo de desarrollo revisa el producto que va generando.
- Inspecciones: revisión de cada producto por parte de colegas.
- Validaciones: es el cliente quien revisa el producto para decir si cumple con sus necesidades.

Esta definición implica que se considera una prueba exitosa si se demuestran deficiencias en el software. Las fallas pueden ser en el código o en el modelado, en dependencia del tipo de pruebas que se le apliquen al software.

Las pruebas de software se pueden clasificar con base en su propósito, por la fase del ciclo de vida y por su alcance. Respecto al propósito de las pruebas de software, estas pueden clasificarse como: pruebas de corrección, de rendimiento, de fiabilidad y de seguridad. Con base al ciclo de vida, las pruebas de software pueden clasificarse como: pruebas de fase de

requisitos, de fase de diseño, de fase de programa, evaluación de resultados de pruebas, de fase de instalación, de aceptación y de mantenimiento (Bibián, 2017).

4.1 Pruebas API, ¿Por qué son importantes?

Las pruebas API brindan acceso a la aplicación sin la necesidad de utilizar la interfaz, y por ello proveen una evaluación más rápida y eficiente de una versión en el ciclo de pruebas. Exponen cualquier error y si algo está mal en la capa de negocio, ayudan a identificar problemas pequeños antes de convertirse en problemas grandes (Vargas, 2020).

4.1.1 Herramientas

La siguiente es una lista de herramientas recomendadas:

- Katalon (SOAP y REST)
- SoapUI (SOAP y REST)
- Postman (REST)
- JMeter (SOAP y REST)
- REST-assured (Java - REST)
- RestSharp (C# - REST)
- Karate DSL (Java - SOAP y REST)

4.1.2 Tipos de Pruebas API

Las pruebas API cubren lo siguiente:

- Pruebas de Unidad: evalúan un solo ‘endpoint’ con una sola solicitud y buscan una respuesta o un conjunto de respuestas.
- Pruebas de Funcionalidad: revisa si la API funciona y hace exactamente lo que debe hacer.

- Pruebas de Carga: verifica si la API puede manejar un número usual y/o más alto de solicitudes.
- Pruebas de Confiabilidad: confirman si la API brinda una conexión y resultados consistentes.
- Pruebas Negativas: verifica cómo se comportan los datos de salida cuando se brinda una entrada inválida.
- Pruebas de Documentación API: revisa si la documentación brinda una guía fácil para el usuario (Vargas, 2020).

4.1.3 ¿Qué verifican las Pruebas API?

Las pruebas API verifican lo siguiente:

- Autorización
- Parámetros de entrada
- Estructura de payload
- Estructura de respuesta
- Resultados esperados
- Código de status HTTP
- Manejo de errores
- Tiempo de respuesta
- Otros

4.1.4 ¿Qué tipos de bugs detectan las pruebas API?

Las pruebas API pueden ayudar a identificar varios tipos de bugs. La siguiente es una lista breve de los errores que se pueden hallar:

- Banderas que no se utilizan

- Funcionalidades ausentes o duplicadas
- Inhabilidad para manejar los errores
- Problemas de seguridad
- Respuestas de error inadecuadas
- Una estructura incorrecta de datos de respuesta
- Tiempo de respuesta prolongado (Vargas, 2020).

4.2 Métodos de Pruebas

Existen diversos métodos para realizar las pruebas de software, entre las más importantes se encuentran la prueba de Caja Blanca y prueba de Caja Negra.

El uso de la prueba de Caja Blanca es mejor para verificar que se recorran todos los caminos y detectar un mayor número de errores. La Caja Negra brinda la posibilidad de cubrir la mayor parte de las combinaciones de entradas y lograr así un juego de pruebas más eficaz.

Las pruebas mencionadas permiten probar cada una de las condiciones existentes en el programa, identificar claramente las entradas, salidas y estudiar las relaciones que existen entre ellas, permitiendo así maximizar la calidad de las pruebas y en dependencia del resultado se constará con un sistema más estable y confiable (Salazar, 2015).

4.2.1 Prueba de Especificación (Caja Negra)

Pruebas de Caja Negra: También suelen ser llamadas funcionales y basadas en especificaciones. En ellas se pretende examinar el programa en busca de que cuente con las funcionalidades que debe tener y cómo lleva a cabo las mismas, analizando siempre los resultados que devuelve y probando todas las entradas en sus valores válidos e inválidos.

Al ejecutar las pruebas de Caja Negra se desarrollan casos de prueba reales para cada condición o combinación de condiciones y se analizan los resultados que arroja el sistema para cada uno de los casos. En esta estrategia se verifica el programa considerándolo una caja negra. Las pruebas no se hacen en base al código, sino a la interfaz. No importa que se cubran

todas las rutas dentro del programa, lo importante es probar todas las entradas en sus valores válidos e inválidos y lograr que el sistema tenga una interfaz amigable (Salazar, 2015).

4.2.1.1 Beneficios de las pruebas de Caja Negra

Las pruebas de caja negra suelen ser aplicadas en fases posteriores de prueba. Este test intencionadamente ignora la estructura de control, concentra su atención en el dominio de la información. El evaluador no necesita conocimientos de ningún lenguaje de programación específico. La prueba se realiza desde el punto de vista del usuario, no del diseñador. Los casos de prueba se pueden diseñar tan pronto como se completen las especificaciones (Salazar, 2015).

4.2.1.2 Postman

Postman es una herramienta que se utiliza, sobre todo, para el testing de API REST, aunque también admite otras funcionalidades fuera del testing de este tipo de sistemas. Esta herramienta, además de testear, consumir y depurar API REST, posibilita monitorizarlas, escribir pruebas automatizadas para ellas, documentarlas y simularlas. Postman puede generar una documentación interesante y atractiva con ejemplos y fragmentos de código, de forma que hace muy fácil entender cómo funciona una API determinada (López, 2019).

A partir del análisis realizado, la herramienta a utilizar para el estudio y desarrollo de las pruebas de caja negra es Postman, debido a las facilidades que ofrece en la operatividad. Para ello se presentan diversos escenarios posibles en cuanto a valores que pueden obtenerse en las peticiones Rest a la API.

4.3 Obtener Resumen de Ventas

GET ventas-resumen

GET http://127.0.0.1:8000/api/ventas-resumen/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers (1)

KEY	VALUE	DESCRIPTION
Authorization	token c03aedc99f86b39ea70e67f5ae15e423f05a2616	
Key	Value	Description

Temporary Headers (8)

Body Cookies (2) Headers (5) Test Results (4/4) Status: 200 OK Time: 104ms

Pretty Raw Preview Visualize JSON


```
{
    "count": 5,
    "next": null,
    "previous": null,
    "results": [
        {
            "unidad": "U1-U2-U3-U4-U5-U6-U7-U8",
            "codigo_cliente": "codigo 1",
            "nombre_cliente": "nombre 1",
            "fecha": "2021-07-30",
            "numero_factura": "numero 1",
            "moneda": "cup",
            "importe": "1.00"
        },
        {
            "unidad": "U1-U2-U3-U4-U5-U6-U7-U8",
            "codigo_cliente": "codigo 2",
```


Ilustración 22 Obtener Resumen de Ventas

Fuente: Elaboración propia

En la ilustración anterior se muestra el caso de uso GET ventas-resumen donde se observa la respuesta obtenida en formato JSON respaldado por el código 200, equivalente a que todas las pruebas fueron realizadas sin errores.

GET http://127.0.0.1:8000/api/ventas-resumen/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});
pm.test("Verify response body is json", function () {
    pm.expect(pm.response.json());
});
```

Body Cookies (2) Headers (5) Test Results (3/3) Status: 200 OK Time: 11

All Passed Skipped Failed

PASS Status code is 200

PASS Response time is less than 200ms

PASS Verify response body is json

Ilustración 23 Postman Detalles

Fuentes: Elaboración propia

Se muestran, además, las pruebas características de Postman donde se puede chequear entre otros status, velocidad y formato. Se evidencia un resultado satisfactorio, se recibe un status 200 a una velocidad por debajo de los 200 ms y en formato JSON.

Así mismo, se hicieron pruebas para peticiones no autorizadas para usuarios no registrados, lanzando el siguiente error en formato JSON.

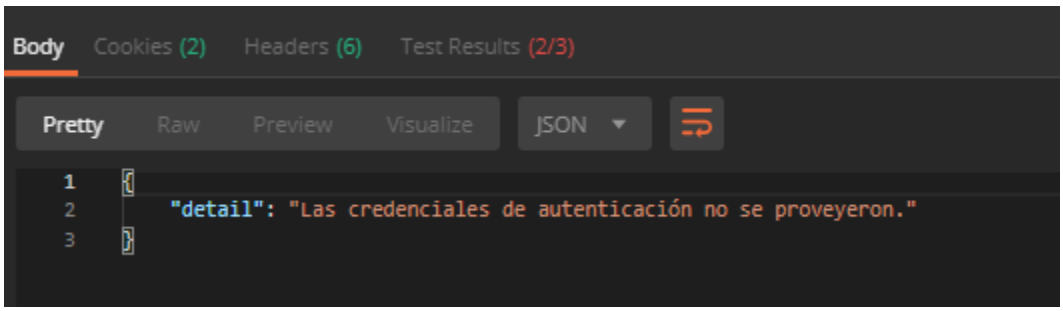


Ilustración 24 Credenciales Inválidas

Fuente: Elaboración propia

Debido a la naturaleza y funcionalidad del sistema BI-Versat, no se poseen peticiones de tipo POST, PUT, DELETE, ya que este sistema solo recibe información en tiempo real del Versat Sarasola. Por lo tanto, es información que debe permanecer intacta y que ningún usuario del BI-Versat puede modificar.

Las pruebas anteriores se aplicaron para todas las peticiones GET del Api BI-Versat:

- GET Inventarios
- GET Cuentas por edades
- GET Ventas resumen
- GET Ventas resumen detalles
- GET Ventas puntos de venta
- GET Movimientos de inventario

- GET Balance contable
- GET Balance contable más gastos

4.4 Prueba de Código (Caja Blanca)

Las pruebas de Caja Blanca suelen ser llamadas estructurales o de cobertura lógica. En ellas se pretende investigar sobre la estructura interna del código, exceptuando detalles referidos a datos de entrada o salida, para probar la lógica del programa desde el punto de vista algorítmico. Realizan un seguimiento del código fuente según se va ejecutando los casos de prueba, determinándose, de manera, concreta las instrucciones, bloques que han sido ejecutados por los casos de prueba.

En las pruebas de Caja Blanca se desarrollan casos de prueba que produzcan la ejecución de cada posible ruta del programa o módulo, considerándose una ruta como una combinación específica de condiciones manejadas por un programa. Cabe destacare que no todos los errores de software se pueden descubrir verificando todas las rutas de un programa, hay errores que se descubren al integrar unidades del sistema y pueden existir errores que no tengan relación con el código específicamente.

4.4.1 Diferencia entre pruebas de caja blanca y caja negra

Las pruebas de caja blanca se basan principalmente en la estructura interna del software y son muy efectivas en la validación del diseño y en la búsqueda de errores de programación y de implementación. Mientras, que las pruebas de caja negra se enfocan en los requerimientos funcionales del software y se llevan a cabo desde la perspectiva del usuario final (Bibián, 2017).

4.4.2 Beneficios de las pruebas de caja blanca

Entre los beneficios de las pruebas de caja blanca resaltan:

- Optimización de código al encontrar errores ocultos.
- Automatizar fácilmente los casos de prueba para el cuadro blanco.
- La prueba es más exhaustiva, porque generalmente se verifica todas las rutas de código.

- Las pruebas pueden comenzar temprano en el proyecto, incluso si la GUI (Interfaz de Usuario) no está disponible (Hoogenraad, 2018).

4.5 Test GET Ventas Resumen

```
class Test_Ventas_Resumen(TestCase):

    def setUp(self):
        # Creamos un usuario y generamos el acceso a la api para hacer pruebas de forma general
        user = User(
            email='testing_login@cosasdedevs.com',
            first_name='Testing',
            last_name='Testing',
            username='testing_login'
        )
        user.set_password('admin123')
        user.save()

        client = APIClient()
        response = client.post(
            '/api/token-auth/', {
                'username': 'testing_login',
                'password': 'admin123',
            },
            format='json'
        )
        result = json.loads(response.content)
        self.access_token = result['token']
        self.user = user
```

Ilustración 25 GET Ventas Resumen

Fuente: Elaboración propia

Se comienza creando un usuario de pruebas con una base de datos temporal que la librería DjangoTest crea para este tipo de situaciones. Se hace el proceso de autenticar al usuario creado para obtener su token y poder proceder con las pruebas.

En la siguiente ilustración se muestra la función que refleja si el usuario creado tiene acceso a la petición GET ventas resumen. Primero se crea un objeto de pruebas en la tabla VentasResumen para después comprobar si el usuario tiene acceso a él y, de ser así, va a lanzar una respuesta de status 200 y finalmente se comprueba que cada uno de los datos obtenidos cumplen con las validaciones específicas de cada campo.

```python
    def test_get_ventas_resumen(self):
        client = APIClient()
        client.credentials(HTTP_AUTHORIZATION='Token ' + self.access_token)

        VentasResumen.objects.create(
            unidad='U1-U2-U3-U4-U5-U6-U7-U8',
            codigo_cliente='codigo 1',
            nombre_cliente='nombre 1',
            fecha='2021-07-30',
            numero_factura='numero 1',
            moneda='cup',
            importe='1.00'
        )

        response = client.get('/api/ventas-resumen/')
        result = json.loads(response.content)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(result['count'], 1)

        for vr in result['results']:
            self.assertIn('unidad', vr)
            self.assertIn('codigo_cliente', vr)
            self.assertIn('nombre_cliente', vr)
            self.assertIn('fecha', vr)
            self.assertIn('numero_factura', vr)
            self.assertIn('moneda', vr)
            self.assertIn('importe', vr)
            break
```

Ilustración 26 Función que valida la información

Fuente: Elaboración propia

Cuando se hace la comprobación se aprecia que las pruebas pasaron correctamente ya que no solo el usuario tenía permiso para realizar esta petición, sino que, además, los datos fueron correctamente validados y no hubo ningún tipo de inconveniente.

```
✔ Tests passed: 1 of 1 test – 481 ms
```

Ilustración 27 Prueba pasada exitosamente

Fuente: Elaboración propia

4.5 Test GET Ventas Resumen Permision

La siguiente imagen muestra lo contrario, se evidencia la respuesta de la API cuando el usuario actual no tiene permisos para realizar esta petición en específico.

Al igual que en el ejemplo anterior, se crea un usuario de pruebas con la diferencia que este usuario no tendrá permisos de administrador y por lo tanto no puede recibir la petición GET ventas resumen de la API.

```python
class Test_Ventas_Resumen_Permision(TestCase):

    def setUp(self):
        # Creamos un usuario reclutador para comprobar que no tiene acceso
        user = User(
            email='testing_login@cosasdedevs.com',
            first_name='Testing',
            last_name='Testing',
            username='testing_login',
            is_staff=False,
            is_superuser=False
        )
        user.set_password('admin123')
        user.save()

        client = APIClient()
        response = client.post(
            '/api/token-auth/', {
                'username': 'testing_login',
                'password': 'admin123',
            },
            format='json'
        )
        result = json.loads(response.content)
        self.access_token = result['token']
        self.user = user
```

Ilustración 28 GET Ventas Resumen Permision

Fuente: Elaboración propia

En la segunda parte del código se comprueba que el usuario no tiene permisos para acceder a esta petición y de ser así se lanza un mensaje ratificando el error.

```python
def test_get_ventas_resumen(self):
    client = APIClient()
    client.credentials(HTTP_AUTHORIZATION='Token ' + 'other_token')

    response = client.get('/api/balance-contable/')

    result = json.loads(response.content)

    self.assertEqual(response.status_code, status.HTTP_401_UNAUTHORIZED)
    self.assertEqual(result['detail'], 'You do not have permission to perform this action.')
```

Ilustración 29 Usuario Sin Autorización

Fuente: Elaboración propia

En esta ilustración se muestra la respuesta en la terminal que, efectivamente, el usuario actual no tiene permisos de administrador para poder recibir la petición GET ventas resumen.

```
Tests failed: 1 of 1 test – 362 ms
Actual   :'You do not have permission to perform this action.'
<Click to see difference>
```

Ilustración 30 Salida por Consola del error de autenticación

Fuente: Elaboración propia

Las pruebas anteriores fueron realizadas a cada una de las peticiones GET de la API BI-Versat, pero por la similitud de estas solo se presenta como ejemplo la petición GET ventas resumen.

4.6 Estimación de Costos del proyecto

La estimación de costo y esfuerzo del software nunca será una ciencia exacta. Demasiadas variables (humanas, técnicas, ambientales, políticas) pueden afectar el costo final del software y el esfuerzo aplicado para su desarrollo. Sin embargo, la estimación del proyecto de software puede transformarse de un arte oscuro a una serie de pasos sistemáticos que proporcionen estimaciones con riesgo aceptable (Pressman, 2010).

En el momento de la planificación del proyecto, es casi imposible determinar el número de líneas de código que poseerá la aplicación, siendo el método tradicional para calcular los costos COCOMO (Modelo Constructivo de Costos, por su acrónimo del inglés COnstructiveCOstMOdel) poco idóneo en este caso, ya que está orientado a la magnitud del

producto final, midiendo el tamaño del proyecto, en líneas de código principalmente. Dentro de sus principales inconvenientes se pueden citar:

- Los resultados no son proporcionales a las tareas de gestión ya que no tiene en cuenta los recursos necesarios para realizarlas.
- Se puede desviar de la realidad si se indica mal el porcentaje de líneas de comentarios en el código fuente.
- Es un tanto subjetivo, puesto que está basado en estimaciones y parámetros que pueden ser "vistos" de distinta manera por distintos analistas que usen el método.
- Se miden los costes del producto, de acuerdo con su tamaño y otras características, pero no la productividad.

Sin embargo, ya que se utiliza XP como metodología para el desarrollo del software se tiene estimado el tiempo de desarrollo de este, aprovechando que la fórmula de Bohem comprende este parámetro (Costo = Cantidad de Hombres * Salario Medio * Tiempo de Desarrollo), es posible obtener una estimación del salario de los autores. Haciendo los cálculos pertinentes se tiene, considerando como valor promedio para el salario mensual por hombres $3000.00, dos desarrolladores y un tiempo de desarrollo estimado de 7 meses, al sustituir y calcular se obtiene un costo de $ 42.000.00.

4.7 Consideraciones finales del capítulo

En este capítulo se realizaron las pruebas tanto de caja negra como de caja blanca a uno de los casos de uso significativos donde se confirma la efectividad en el manejo de las transacciones a partir de que las respuestas esperadas coinciden con las obtenidas. De esta forma, se cumplen con las expectativas ya que la solución ofrecida satisface los resultados necesarios.

Conclusiones

A partir de la investigación realizada, se concluye que:

- El uso del framework Django ayudó a optimizar significativamente la codificación del sistema, además de la facilidad de configurar nuevos módulos internos o externos, todo ello se traduce en una reducción en el tiempo de producción. El desarrollo del software permite a los directivos determinar su propia información relevante, al tener control sobre lo que se muestra en los gráficos de la aplicación de manera que les permite gestionar información económica, financiera y productiva en tiempo real.
- Se realizó un análisis de la literatura especializada, abarcando aspectos relacionados a las metodologías para el desarrollo de soluciones de inteligencia de negocio. Esto conllevó a seleccionar la metodología de desarrollo de software XP para su aplicación al proyecto BI-Versat.
- El caso de estudio del sistema Versat Sarasola no es capaz de brindar la información oportuna que requieren los directivos de una empresa para tomar sus decisiones económicas.
- El sistema BI-Versat ofrece a los directivos de la empresa Datazucar las herramientas para crear su propia información relevante, cumpliendo con los requerimientos para la puesta en práctica de la herramienta.
- Se desarrolló una API REST mediante el uso de Django Rest Framework que permite una futura escalabilidad y soporte al software mediante el desarrollo de una aplicación Android que muestre los datos de BI-Versat a sus directivos desde sus dispositivos móviles.
- Se definieron diversos roles y permisos, especificando la información que será mostrada por cada perfil de usuario, permitiendo un eficiente control sobre los usuarios que acceden al sistema.
- El desarrollo del software cumple con las expectativas ya que la solución ofrecida satisface los resultados necesarios, lo cual se demuestra con la realización satisfactoria de pruebas.

Bibliografía

Agüero, J. D. (2019). *Aplicación de la inteligencia de negocios para la toma de decisiones en las pequeñas y medianas empresas de la Provincia de Pasco.* (Para optar el título profesional de: Ingeniero de Sistemas y Computación), Universidad Nacional Daniel Alcides Carrión, Cerro de Pasco, Perú.

Alimatic. (2020). Sistema Integral de Gestión Empresarial SICEMA PLUS SQL. Recuperado de: https://www.alimatic.alinet.cu

Bibián, J. (2017). Modelo para la ejecución de pruebas de software. *Revista Internacional de Investigación e Innovación Tecnológica, 5*. Recuperado de: http://www.riiit.com.mx/

Caballero, J. L. (2009). *Metodologías Ágiles para el desarrollo de Software.* (Trabajo de diploma para optar por el título de Ingeniería en Informática), Instituto Superior Minero Metalúrgico, Moa, Holguín.

Caldera, R. (2017). *Estudio del framework de desarrollo web Django.* (Grado en Ingeniería Informática), Universidad de Alcalá, España.

Cano, J. L. (2007). *Business intelligence: Competir con información*. Madrid, España: Fundación Cultural Banesto.

Citmatel. (2020). Sistema Integral Económico Administrativo RODAS XXI. Recuperado de: http://www.rodasxxi.cu

Cupet. (2020). SisCont5. Recuperado de: https://www.cupet.cu/footer/informatica-automatica-y-comunicaciones

Da Rocha, H. (2019). *Learn Chart.js. Create interactive visualizations for the Web with Chart.js 2*. Birmingham, Mumbai: Packt Publishing Ltd.

Datazucar. (2020). Servicios Informáticos Villa Clara, Versat Sarasola. Recuperado de: https://www.versat.azcuba.cu

Domínguez, L. A. (2016). *Análisis de sistemas de información*. México: Red Tercer Milenio

Ferguson, R. (2019). *Introduction to JavaScript In: Beginning JavaScript.* (Third Edition ed.). Berkeley, CA: Apress.

Fernández, E. P., Neira, E., y Clares, J. (2016). Gestión de datos en el negocio audiovisual: Netflix como estudio de caso. *Revista internacional de Información y Comunicación, 25*. http://www.elprofesionaldelainformacion.com/contenidos/2016/jul/06.pdf

Fernández, Ó. (2018). *Aplicación de metodología Scrum para el desarrollo de una API web de gestión médica con MEAN Stack.* (Tesis en opción al Grado en Ingeniería Informática), Universidad de Valladolid, España.

García, A. M. (2020). Aplicación de técnicas de inteligencia de negocios y análisis de datos en el entorno empresarial cubano: retos y perspectivas. *Revista Cubana de Ciencias Informáticas, 14*.

Garner, B. (2019). Build a REST API in 30 minutes with Django REST Framework. Recuperado de: https://medium.com/swlh/build-your-first-rest-api-with-django-rest-framework-e394e39a482c

Gómez, N. (2010). *Principios de Contabilidad* (4ta ed.). Colombia.

Govea, R. (2020). *Sistema para la gestión de alertas y avisos vía SMS para los usuarios de la Plataforma Bienestar.* Universidad Central "Marta Abreu" de Las Villas, Santa Clara, Villa Clara.

Hernández, E. A. (2020). *Desarrollo de una aplicación web con el framework Bootstrap y el precompilador Sass para la gestión de pedidos de productos agrícolas de la empresa El Chagra.* (Presentado para optar al grado académico de: Ingeniero en Sistemas Informáticos), Escuela Superior Politécnica De Chimborazo, Riobamba, Ecuador.

Hernández, L. (2013). *Mercado de Datos para el proceso de inventario del Sistema de Gestión Contable VERSAT Sarasola.* Universidad Central "Marta Abreu" de las Villas, Santa Clara, Villa Clara.

Hoogenraad, W. (2018). Prueba de caja blanca bajo el microscopio. Recuperado de: https://es.itpedia.nl/2018/02/05/white-box-testing-onder-de-loep/

Infante, U. (2020). Conectando una aplicación de Angular con una Api Rest. Recuperado de: https://rd.rocktech.mx/publicaciones/entrada/conectando-una-aplicacion-de-angular-con-una-api-rest

Lara, J. J. (2019). *Aplicación web para la gestión de solicitudes de almacén en la Dirección Provincial de Bufetes Colectivo Matanzas.* Universidad de Matanzas, Matanzas.

Llerena, J. (2020). *Codifica en Python* (Vol. 1). Quito,Ecuador: Universidad Politécnica Salesiana.

López, A. (2019). Qué es Postman y para qué sirve. Recuperado de: https://openwebinars.net/blog/que-es-postman/

Meléndez, S. M., Gaitan, M. E., y Pérez, N. N. (2016). *Metodología Ágil de desarrollo de software Programación Extrema.* Universidad Nacional Autónoma de Nicaragua, Nicaragua.

Menéndez, R. (2005). *Construcción de software orientado a objetos con el proceso unificado y el UML, un punto de vista práctico*. Huancayo, Perú: Universidad Continental.

Merencio, Y. (2019). *Procedimiento para la gestión de la información en el proceso de producción de azúcar en la Empresa Azucarera Holguín.* (Tesis presentada en opción al Título Académico de Máster en Dirección), Universidad de Holguín, Holguín, Cuba.

Mircha, J. (2020). Api Rest. Recuperado de: https://jonmircha.com/img/blog/rest-crud.png

Moreto, S., Lambert, M., Jakobus, B., y Marah, J. (2017). *Bootstrap 4 – Responsive Web Design* (1ra ed.). Birmingham, Reino Unido: Packt Publishing.

Muradas, Y. (2018). Conoce las 3 metodologías ágiles más usadas. Recuperado de: https://openwebinars.net/blog/conoce-las-3-metodologias-agiles-mas-usadas/

Pino, J. d. (2021). Introducción a Django. Recuperado de: https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction

Pressman, R. (2010). *Ingeniería del software. Un enfoque práctico* (3ra ed.). México: Mcgraw-Hill Interamericana Editores.

Ramírez, M. (2018). *Cómo entender contabilidad sin ser contador* (1ª ed.). Ciudad de México, México: Instituto Mexicano de Contadores Públicos, A.C.

Sadiku, M. N. O., y Musa, S. M. (2021). *Business Intelligence. In: A Primer on Multiple Intelligences* Recuperado de: https://doi.org/10.1007/978-3-030-77584-1_14

Salazar, E. (2015). Procedimiento para realizar pruebas de Caja Blanca. Recuperado de: https://www.informatica-juridica.com/trabajos/procedimiento-realizar-pruebas-caja-blanca/

Salazar, E. G. (2015). *Sistema de seguimiento de participantes y su vinculación con el sector productivo para la Unidad de Educación continua y el Instituto de Investigación y Posgrados de la Facultad de Ciencias Administrativas.* (Trabajo de Graduación previo a la obtención del Título de Ingeniero Informático), Universidad Central del Ecuador, Ecuador.

Sarzosa, C. E. (2018). *Estudio del framework opensource Bootstrap para la implementación de un Sistema de Seguimiento de actividades administrativas de la carrera de Ingeniería en Sistemas Computacionales de la Universidad Técnica del Norte.* (Opta por el título de Ingeniero en Sistemas Computacionales), Universidad Técnica Del Norte, Ecuador.

Savoy, S. (2015). Consejos para el diseño de mensajes de error. Recuperado de: https://blog.ida.cl/diseno/consejos-diseno-mensajes-error/

Schönig, H.-J. (2019). *Mastering PostgreSQL 12: Advanced Techniques to Build and Administer Scalable and Reliable PostgreSQL Database Applications* (3rd ed.). India: Packt Publishing.

Solórzano, J. A. (2018). *Desarrollo de una aplicación web multiplataforma usando el framework Django, para publicitar eventos sociales, aplicado en el municipio del Cantón Morona* (Para optar al Grado Académico de Ingeniero en Sistemas Informáticos), Escuela Superior Politécnica de Chimborazo., Macas, Ecuador

Vargas, R. (2020). Todo acerca de APIs y Pruebas API. Recuperado de: https://www.encora.com/es/blog/todo-acerca-de-apis-y-pruebas-api

Vincent, W. (2021). *Django for Beginners: Build websites with Python and Django*. Traverse City, Michigan: Independently Publisher.

Viniegra, J. S. (2020). Estrategias de Sincronización. Recuperado de: https://proasw.wordpress.com/2020/09/12/estrategias-de-sincronizacion/