UNIVERSIDAD SANTO TOMAS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
T U N J A

DESARROLLO DE UN SISTEMA DE GESTIÓN DE INVENTARIO PARA EL CONTROL DE PRODUCTOS EN LAS FASES DE SOLICITUD, APRESTAMIENTO, ENVÍO, RECEPCIÓN Y RETORNO

PROPONENTE(S)
Oscar Javier Gómez Sánchez
CC. 1069079235
Cod. 2251136
Luis Guillermo Gómez Galeano
CC. 1007141532
Cod. 2251130

DIRECTOR
Jennifer Eliana Correa Ussa

Tunja
18 de junio de 2021

CONTENIDO

ÍNDICE DE TABLAS

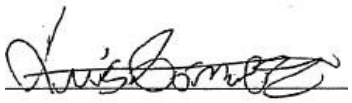
ÍNDICE DE FIGURAS

ANEXOS

1. FICHA TÉCNICA DE LA PASANTÍA

Título	Desarrollo de un Sistema de Gestión de Inventario para el Control de Productos en las Fases de Solicitud, Aprestamiento, Envío, Recepción y Retorno.
Autor (es)	Luis Guillermo Gómez Galeano, Oscar Javier Gómez Sánchez
Director del proyecto	Jennifer Eliana Correa Ussa
Entidad donde se realizó la pasantía	Ide@soft
Lugar de ejecución de la pasantía	Tunja, Boyacá
Duración	4 meses

Los abajo firmantes confirman que todos los datos incluidos en el presente informe son correctos y verídicos, que las actividades aquí descritas fueron completadas satisfactoriamente. Asimismo, se constata que la estudiante ha cumplido con las **640** horas exigidas por la universidad.



Firma pasante 1
Luis Guillermo Gómez Galeano

Firma directora
Jennifer Eliana Correa Ussa

Firma pasante 2
Oscar Javier Gómez Sánchez

Firma representante legal de la empresa
Iván Fernando Fonseca Barinas

Título	Desarrollo de un Sistema de Gestión de Inventario para el Control de Productos en las Fases de Solicitud, Aprestamiento, Envío, Recepción y Retorno.
Autor (es)	Luis Guillermo Gómez Galeano, Oscar Javier Gómez Sánchez
Director del proyecto	Jennifer Eliana Correa Ussa
Palabras claves	Software, Inventario, API, informática
Descripción	
Desarrollo de un software para servir una API Graphql para el control de inventario como también los procesos de una empresa cliente usando principalmente el framework Django	

2. PLANTEAMIENTO DEL PROBLEMA

Empresas que basan su negocio en grandes cantidades de inventario presentan problemas en la gestión y traslado de sus productos en sitio de trabajo. Al aumentar sus actividades se han presentado falencias en el control del inventario, involucrando pérdidas considerables de dinero por el incremento de los tiempos desde la solicitud de productos hasta su préstamo, y el deterioro de estos (Castañeda y Silva, 2013).

Las empresas que han implementado sistemas de inventario como Alegra Colombia, Siigo o Microsoft Dynamics les han permitido mejorar el control de la información de sus productos, sin embargo, presentan algunas limitaciones en los procesos de solicitud, aprestamiento, envío, recepción y retorno de cada producto dificultando la trazabilidad de cada uno de los elementos, el control de estos y la productividad.

De igual forma, una de las problemáticas en las empresas se presenta cuando la administración no se preocupa por realizar una planificación adecuada del mantenimiento incumpliendo los estándares de desempeño de sus procesos debido a la baja confiabilidad en los activos de producción, repercutiendo en los costos globales de operación, en la vida útil de la maquinaria y en el nivel competitivo de la industria. (Mago et al., 2020)

Finalmente, estas empresas no tienen una manera de indicar las tareas que pueden realizar cada uno de los empleados sobre el inventario, ni hacer seguimiento del cumplimiento de estas pese a que maneja diferentes niveles de responsabilidad para el desarrollo de las operaciones.

El siguiente diagrama de espina de pescado [Figura 1] muestra las causas de la situación problema.

Figura 1. Diagrama de pescado de las causas de por qué hay pérdidas en el inventario

Fuente: Autor

3. JUSTIFICACIÓN

La necesidad de las empresas en Colombia por consolidarse en el mercado requiere el mejoramiento continuo de sus procesos, el desarrollo constante de habilidades de los trabajadores y el uso de soluciones tecnológicas innovadoras. Tanto pequeñas como grandes empresas requieren tener un control sofisticado de su negocio por lo que los inventarios son un aspecto importante porque implica poseer una mayor supervisión del stock, reducir costos y acelerar el cumplimiento de la demanda (Pintelon y Van, 2015).

El desarrollo del sistema y su implementación permitirá a las empresas facilitar el registro y seguimiento de los productos, el histórico de mantenimientos, la gestión de eventos con sus respectivas órdenes, el control de acceso a la información mediante roles de usuario. Además, el sistema facilitará a las empresas el seguimiento y la trazabilidad de los productos en cada una de las fases de solicitud, aprestamiento, envío, recepción y retorno de los elementos, ofreciendo información oportuna y confiable sobre el lugar, el estado y el responsable de cada producto.

Una de las características con la que contará el sistema es la generación y lectura de códigos QR. Esta codificación permitirá capturar información relacionada con el identificador de cada producto, conocer las unidades logísticas y localizaciones de manera automática, de esta forma se cuenta con un control adecuado de las existencias.

La implementación del sistema contribuirá con la reducción de tiempos y gastos económicos puesto que se mejora la gestión sobre el control de inventario, aumentando la productividad, eficiencia en la prestación de los servicios y agiliza el tiempo de respuesta a los clientes así mismo contribuye a centrarse en los procesos misionales del negocio como la producción y la adquisición de nuevos mercados.

4. OBJETIVOS

4.1. Objetivo General

Desarrollar un sistema de control de inventario para la gestión de los productos en las fases de solicitud, aprestamiento, envío, recepción y retorno.

4.2. Objetivos específicos

A continuación, se presentan los objetivos específicos.

Tabla 1. Objetivos específicos

Nro.	Objetivo específico
1	Determinar el alcance del sistema para definir las necesidades funcionales mediante documento de historias de usuario.
2	Implementar una base de datos que permita gestionar la información del inventario por medio del sistema gestor de bases de datos PostgreSQL.
3	Desarrollar el sistema para el cumplimiento de las historias de usuario utilizando buenas prácticas de programación orientada a objetos.

Fuente: Autor

5. MODELO DE DESARROLLO

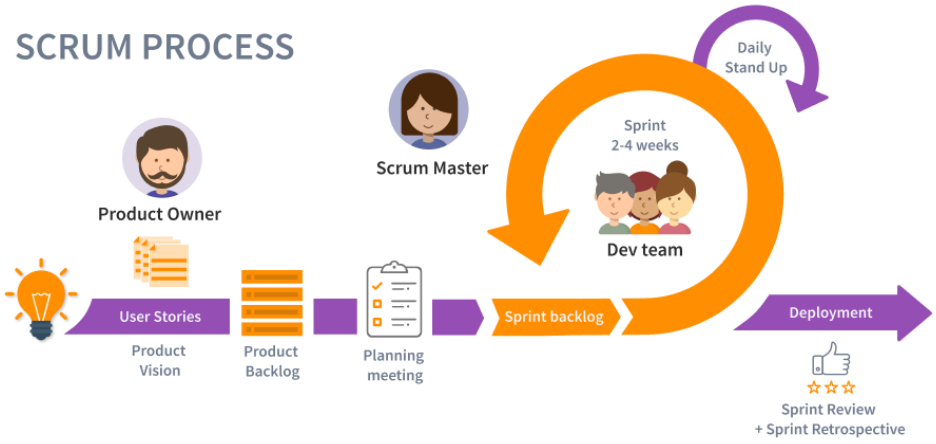
En el ciclo de vida del software se aplicaron las fases: análisis de requisitos; diseño y arquitectura; implementación; pruebas y validación, y documentación.

Se usó la metodología *Scrum* [Figura 2] donde el *product backlog* consistió en las historias de usuario elaboradas, el s*print backlog* el cual es un subconjunto de tareas del product backlog se organizaron y gestionaron usando la herramienta *Trello* y posteriormente *Gitkraken Boards* las cuales son herramientas que facilitan el trabajo en equipo y la organización de tareas.

Además, se desempeñaron los siguientes roles:

- **Product owner (PO):** desempeñado por el gerente de la empresa Iván Fonseca encargado de la comunicación con el cliente, gestionar los presupuestos, decidir qué actividades son prioritarias y maximizar el valor del producto.
- **Scrum master:** desempeñado por la mánager de equipo encargada de liderar el equipo de desarrollo para que cumpla con la metodología y evitar impedimentos que puedan afectar a la entrega del producto.
- **Equipo de desarrollo:** desempeñado por Oscar Gómez y Luis Gómez encargados de desarrollar el backend del producto dando cumplimiento a las historias de usuario de cada sprint.

Figura 2. Diagrama de la metodología Scrum



Tomada de tuleap [Gráfico], 2021, https://www.tuleap.org/wp-content/uploads/2020/06/Scrum-process-schema-EN-small.png

A continuación, se listan (Tabla 2) las fases del ciclo de vida de software junto con las actividades y evidencias de cada una.

Tabla 2. Resumen general de las fases

Fase	Actividades	Evidencias
1. Análisis de historias de usuario	• Análisis de historias de usuario • Enumeración de historias de usuario	• Documento de historias de usuario
2. Diseño y Arquitectura	• Diseño de la base de datos • Diseño de la arquitectura	• Archivo del diseño de la base de datos • Diagrama de la arquitectura del software
3. Implementación	• Desarrollo del Backend • Pruebas • Gestión de cambios • Despliegue del software en servidor de desarrollo	• Software el cual cumple con las necesidades y requerimientos especificados por la empresa
4. Pruebas y validación	• Pruebas de calidad y cumplimiento	• Resultados de análisis SonarQube
5. Documentación	• Documentación de la API	• Documentación final de la API/Software
6. Despliegue	• Despliegue del software en producción	• Software ejecutándose en modo producción

6. DESARROLLO DEL PROYECTO

A continuación, se detalla cada una de las actividades que tienen cada fase del proyecto.

6.1. Fase 1. Análisis de requisitos

El desarrollo de la fase de análisis de requisitos estuvo compuesto por dos actividades con el objetivo de comprender los requisitos que debía cumplir el software.

6.1.1. Análisis de historias de usuario

La empresa facilitó acceso a una carpeta compartida en Google Drive para compartir documentos dentro del equipo de trabajo que constaba de dos desarrolladores Backend (autores), un desarrollador frontend y el gerente de la empresa.

Se socializó el documento de historias de usuario dispuesto en un documento de Google Docs. En el documento se listan las historias de usuario acompañadas de figuras (diseños de interfaz de usuario de baja fidelidad) que el desarrollador frontend debía dar cumplimiento [Figura 3].

Figura 3. Documento de historias de usuario

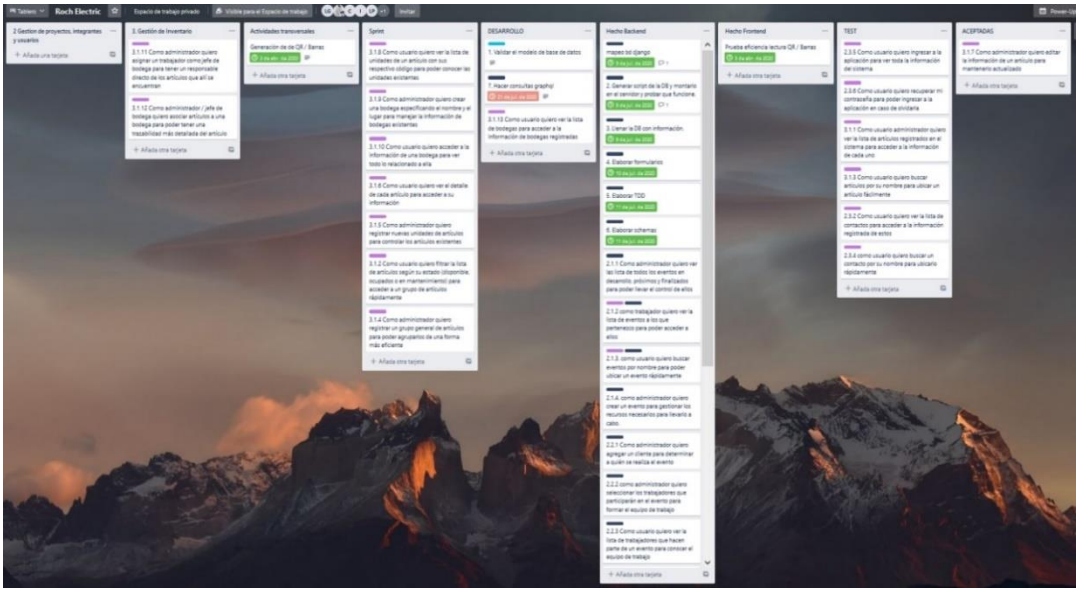
Fuente: Autor

Como compromiso se determinó realizar la lectura de cada una de las historias contenidas en el documento para comprender los requisitos funcionales que el software debía cumplir para luego en la siguiente fase (diseño y arquitectura) ser verificadas en el diseño de la base de datos preexistente.

6.1.2. Clasificación de historias de usuario

La empresa dispuso un tablero compartido en la aplicación Trello donde se organizaron las historias de usuario en diferentes columnas para facilitar su gestión [Figura 4] bajo la metodología SCRUM.

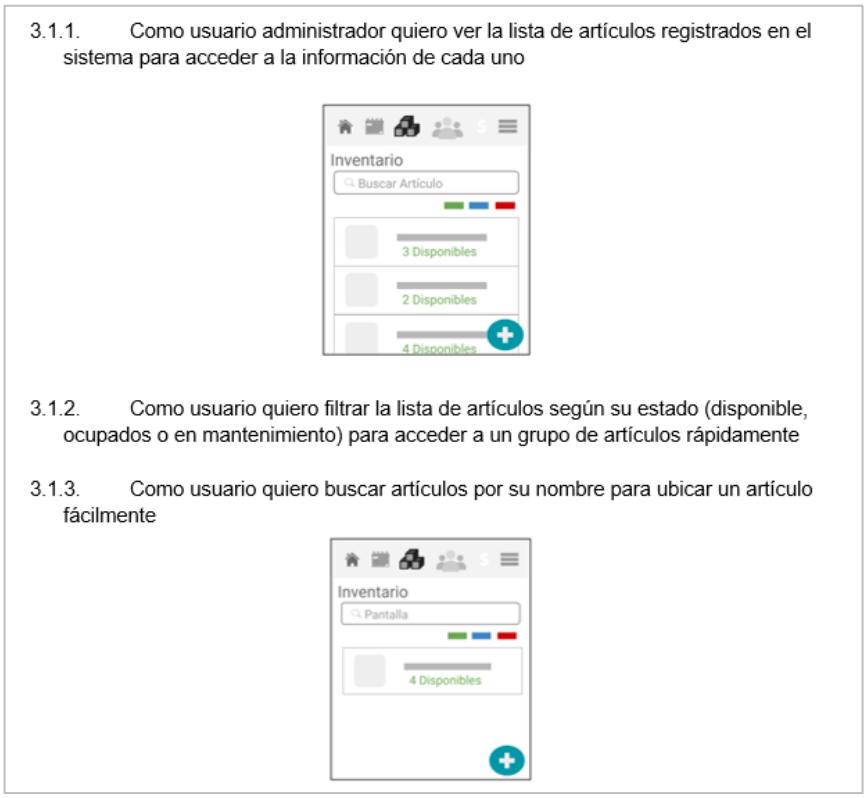
Figura 4. Tablero de actividades en la herramienta Trello



Fuente: Autor

Se necesitaba identificar cada una de las historias de usuario para gestionarlas en el tablero y no perderlas de vista en el documento inicial que contenía mayor detalle de cada una por lo que se enumeraron todas [Figura 5].

Figura 5. Documento de historias de usuario enumeradas



Fuente: Autor

A continuación, se muestran las entradas, instrumentos/herramientas, entregables y tiempo de la fase de análisis de historias de usuario.

Tabla 3. Fase 1. Análisis de historias de usuario

Tipo	Elementos
Entradas	• Documentos de historias de usuario
Instrumentos y/o herramientas	• Computador • Conexión a internet • Navegador Web • Cuenta de Google (acceso a la carpeta compartida)
Entregables	Documento de historias de usuario
Tiempo	1 semana

Fuente: Autor

6.2. Fase 2. Diseño y Arquitectura

El desarrollo de la fase de diseño y arquitectura estuvo compuesto por dos actividades: el diseño de la base de datos y el diseño de la arquitectura.

6.2.1. Diseño de la base de datos

La empresa dispuso un archivo .ldm (herramienta *Power Designer)* el cual estaba disponible en la carpeta compartida que se entregó en la primera fase (Fase 1. análisis de requisitos). Este archivo era el borrador del diseño entidad–relación de la base de datos.
Se revisó y modificó el diseño inicial obteniéndose las siguientes tablas:

- project: eventos o proyectos donde van a estar o estuvieron los equipos prestados.
- order: solicitud de equipos o materiales para un determinado proyecto.
- accounting_movement: ingresos y egresos de un proyecto u orden de un proyecto.
- category_movement: conceptos de pagos y cobro que puede tener un movimiento.
- contact: usuarios del sistema y clientes.
- detail: seguimiento de cada uno de los productos en las fases que puede tener un proceso (venta, prestamos, etc.) definido por la empresa.
- document: documentos digitalizados cualesquiera que se elaboran al reparar un producto.
- document_reference: documentos digitalizados cualesquiera que se elaboran al realizarse un movimiento.
- track: lugares en los que han estado los diferentes productos como proyectos o bodegas.
- group: agrupación de productos similares según como aparece en el inventario existente.
- message: notas cualesquiera que se pueden hacer a una reparación.
- order_document: documentos digitalizados cualesquiera que se elaboran en cada fase del flujo asociado a una orden.
- product: también llamado artículo, equipo, elemento o material que la empresa posee para vender o prestar.
- phase: fases que puede tener determinado flujo de un orden.
- phase_order: registro de la fase en la que una determinada orden esta.
- repair: registro de reparación de los productos.

- repair_type: tipos de reparaciones que puede tener un producto.
- site: registro de un lugar.
- unit: registro de unidades, ejemplo: litros, metros, unidades.

En la posterior fase (Fase 3. Implementación) se fueron realizando mejoras al diseño de la base de datos y los respectivos cambios en el código (mapeo relacional de objetos de Django) como la eliminación y adición de tablas, y adicción de campos a tablas existentes. Las tablas eliminadas fueron:

- track: la empresa no le interesaba hacer seguimiento exhaustivo del historial de la ubicación de los productos.
- category_movement: se eliminó a petición del cliente debido a que solo necesitaba un control básico para determinar si fue un ingreso o egreso y no una gran cantidad de conceptos como ventas, salario, adelantos, etc.
- repair_type: se eliminó por los pocos registros y la nula variabilidad que estos tendrían optando por usar una enumeración.

Las tablas añadidas fueron:

- product_item: era la tabla *product* existente pero especializada para identificar cada producto individualmente de un grupo de productos (tabla *product*) con las mismas características.
- product: agrupación general de productos que comparten las mismas características.
- kit: determina la composición de un producto a partir de otros productos.
- product_item_kit: productos que componen un kit.
- order_detail: productos que se requieren en una orden ya sea de un proyecto o venta con las cantidades requeridas.
- entity: empresa u organización a la que pertenece un usuario.
- line: agrupación de productos por grupos (tabla *group*) según el inventario existente.
- member: persona que pertenece a un proyecto.
- product_history: cantidad que se adiciona o quita de un producto consumible existente.
- order_history: ordenes finalizadas junto con el usuario que las finalizó.

Algunas tablas con modificaciones en sus campos fueron:

- accounting_movement: se agregaron los campos *description* para agregar información que detalle el movimiento; *is_paid* y *movement_date* para determinar si el movimiento está pago, es un futuro pago o es una deuda pendiente; *payment_method* para el registro del método de pago*; type* para determinar si es un ingreso o egreso; *made_by* para saber que usuario realizó el movimiento y *tag* para clasificar libremente los movimientos.
- site: se agregaron los campos *is_warehouse* para determinar si el sitio es una bodega; *manager_warehouse* (en caso de que el lugar sea una bodega) para determinar qué persona está a cargo de esta; *lat* y *lon* para el registro opcional de coordenadas y con ello facilitar el uso posterior de APIs como Google Maps.
- unit: se agregó el campo *symbol* para registrar el símbolo de la unidad.
- member: se agregó el campo *is_customer* para determinar el cliente de un proyecto.
- contact: se agregó el campo *picture* para la adición de la ruta de la foto de perfil de un usuario.
- phase_order: se agregaron los campos *notes* para el registro de información no estructurada en una fase y *is_complete* para registrar si una fase de una orden fue completada sin necesidad de hacer cálculos complejos.
- order: se agregaron los campos *request_by* y *receiver* para determinar quién hace la solicitud y quién es el destinatario de la orden respectivamente.
- order_detail: se agregó el campo *inflow_quantity* para verificar que los productos prestados o vendidos sean los solicitados e igualmente para verificar que la cantidad a retornar o entregar al comprador sea igual a la cantidad aprobada en la orden.
- repair: se agregó el estado NOT_REP (enumeración) para usarse cuando un producto no puede ser reparado.
- product_item: se agregaron los campos *description* para agregar notas a un producto en específico; *serial* para registrar el serial de los productos que lo tienen; *return_date* para saber la fecha en que un producto estará disponible y con ello hacer planeación de proyectos/ventas futuras y al campo *state* las enumeraciones DELETE para la eliminación lógica y DAG (dañado permanentemente) para inhabilitar el producto en proyectos o ventas.

Adicionalmente se realizaron configuraciones y mejoras generales como las siguientes:

- Validación de longitudes: se verificaron las longitudes de cadenas, decimales y flotantes para que fueran adecuadas a la información a guardar.
- Generales: nombres de tablas y campos en inglés, uso de minúsculas y guión bajo donde se use más de una palabra, por ejemplo, *order_detail*.
- Constraints:
    - Llaves primarias: todas las tablas tienen llave primaria *id* de tipo entero autoincremental.
    - Campos únicos: se validó que determinados campos no permitan registros repetidos por ejemplo el correo, el nombre de un proyecto, el código de una línea, etc.
    - Llaves foráneas: se evita que la eliminación de un registro padre elimine indiscriminadamente los registros hijos mediante el uso de la propiedad *DELETE RESTRICT,* por ejemplo, las tablas *product* y *order. M*ientras que las otras tablas tienen la propiedad *DELETE CASCADE.*
    - Campos nulos y obligatorios: según la lógica del negocio se dejaron campos obligatorios y no obligatorios, por ejemplo, el campo *serial* en la tabla *product_item* que no es obligatorio.

Finalmente, se listan las 10 tablas creadas automáticamente por el framework Django:

- Gestión de usuarios, grupos y permisos de estos:
    - auth_group
    - auth_permission
    - auth_group_permission
    - django_content_type
    - django_session
    - contact_group
    - contact_user_permissions
    - graphql_auth_userstatus
- Registro de actividades del panel de administración de Django.
    - django_admin_log

- Gestión de migraciones de la base de datos.
    - django_migrations

Se obtiene el diseño final [Figura 6] de la base de datos con un total de 35 tablas de las cuales 25 son propias y 10 son creadas automáticamente por el framework Django.

Figura 6. Diagrama entidad-relación de la base de datos final

Fuente: Autor

6.2.2. Diseño de la arquitectura

La empresa previamente había determinado exponer el servicio web mediante una API GraphQL ya que era la tecnología que han estado usando en el desarrollo de sus productos por la flexibilidad y facilidad para acceder e ingresar a los datos.

La empresa determinó emplear un lenguaje de programación y framework que facilitara la implementación y desarrollo ágil de este tipo de APIs por lo que se usó el lenguaje de programación Python y el framework Django.

En la página oficial de Django este se define como:

> Django es un framework web de alto nivel en Python que fomenta el desarrollo rápido y el diseño limpio y pragmático. Construido por desarrolladores experimentados, se encarga de gran parte de las molestias del desarrollo web, para poder centrarte en escribir la aplicación sin necesidad de reinventar la rueda. Es gratuito y de código abierto (Django Framework, 2021).

La empresa eligió el motor de base de datos PostgreSQL. Según ellos, se eligió este motor de base de datos debido a que es relacional, gratuita, segura, ampliamente usada en el mundo empresarial, cuenta con buena documentación y soporte.

A continuación, se muestran las entradas, instrumentos/herramientas, entregables y tiempo de esta fase.

Tabla 4. Fase 2. Diseño y Arquitectura

Tipo	Elementos
Entradas	• Archivo *.ldm* del diseño entidad-relación de la base de datos
Instrumentos y/o herramientas	• Computador • Conexión a internet • Navegador Web • Cuenta de Google (acceso a la carpeta compartida) • Power Designer
Entregables	• Diseño final entidad-relación de la base de datos • Listado de tecnologías a usar para la creación del software
Tiempo	3 semanas

Fuente: Autor

6.3. Fase 3. Implementación

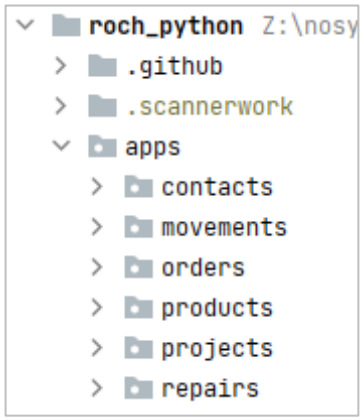
El desarrollo de la fase de implementación estuvo compuesto de las siguientes actividades:

6.3.1. Desarrollo del Backend

Inicialmente se modularizó el software en aplicaciones de Django que es la arquitectura propuesta por este framework. En la documentación oficial de Django (Django Framework, 2021), una aplicación describe un paquete de Python que proporciona un conjunto de características y que estas aplicaciones se pueden reutilizar en varios proyectos.

Se definieron las aplicaciones necesarias para el software las cuales fueron *contacts, movements, orders, projects, products y repairs* como se muestra en la captura de pantalla [Figura 7].

Figura 7. Lista de aplicaciones del software



Fuente: Autor

Una vez definidas las aplicaciones del software se procedió a mapear por cada una de estas las tablas correspondientes del diseño de la base de datos a un modelo de Django.

Un modelo es la fuente única y definitiva de información sobre sus datos. Contiene los campos y comportamientos esenciales de los datos que se almacenan. Por lo general, cada modelo se asigna a una sola tabla de la base de datos (Django Framework, 2021).

Por ejemplo, el mapeo de la tabla *unit* [Figura 8] especifica el nombre y los campos de esta con sus respectivos tipos de datos, validaciones y tamaños.

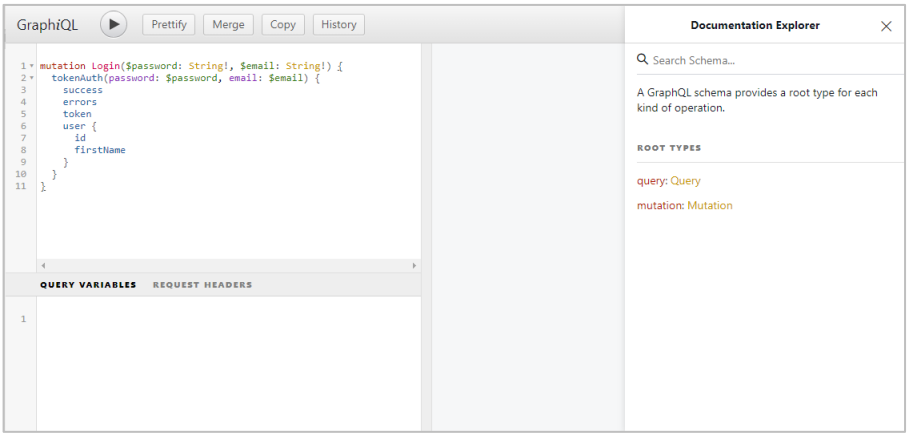
Figura 8. Mapeo de la tabla *Unit*

```python
class Unit(models.Model):
    """Unit model"""

    name = models.CharField(
        _('nombre de la unidad de medida'),
        max_length=60,
        unique=True,
        error_messages={
            'unique': _('Ya existe una unidad con ese nombre.')},
        validators=[
            MinLengthValidator(limit_value=3, message=_('El nombre debe tener al menos 3 caracteres.')),
        ]
    )
    symbol = models.CharField(
        _('Símbolo de la unidad'), max_length=5, unique=True,
        error_messages={
            'unique': _('Ya existe una unidad con ese símbolo.')
        }
    )
    description = models.TextField(_('descripción de la unidad'), max_length=700, null=True, blank=True)

    class Meta:
        db_table = 'unit'
        verbose_name = _('unidad')
        verbose_name_plural = _('unidades')

    def __str__(self):
        return self.name
```

Fuente: Autor

Ya con el mapeo de todas las tablas a los modelos de Django se procedió a implementar las funcionalidades de la lógica de negocio por cada una de las aplicaciones y expuesta como servicio web a través de API GraphQL la cual fue usada mediante la interfaz gráfica de GraphiQL el cual es un editor, explorador e IDE [Figura 9] interactivo para construir consultas y explorar la API.

Figura 9. Editor, explorador e IDE GraphiQL



Fuente: Autor

Finalmente, se aplicaron correcciones generales a la lógica de negocio de algunas aplicaciones para dar cumplimiento a lo indicado en las historias de usuario.

6.3.1.1 Buenas prácticas de programación

Durante el desarrollo se aplicaron buenas prácticas de programación orientada a objetos, específicamente aplicando los principios SOLID.

SOLID es un acrónimo inventado por Robert C. Martin para establecer los cinco principios básicos de la programación orientada a objetos y diseño. Este acrónimo tiene bastante relación con los patrones de diseño, en especial, con la alta cohesión y el bajo acoplamiento (Rubira, 2021). Los principios aplicados fueron:

- **S – Single Responsibility:** consiste en que cada clase debe tener una única responsabilidad, trabajo o propósito. Por ejemplo, la clase *Product* [Figura 10] tiene la única responsabilidad de abstraer un producto de la vida real.
- **O – Open-Close:** establece que una entidad de software (clase, módulo, función, etc.) debe quedar abierta para su extensión, pero cerrada para su modificación. Es decir, se debe poder extender el comportamiento de tal entidad, pero sin modificar su código fuente (Álvarez et al., 2021). Por ejemplo, la clase *Contact* [**Error! Reference source not found.**] extiende de las clases *AbstractBaseUser* y *PermissionsMixin* de Django.
- **L – Liskov Substitution:** las clases derivadas o secundarias deben ser sustituibles por sus clases base o primarias. Este principio garantiza que cualquier clase que sea hija de una clase principal se pueda utilizar en lugar de su principal sin ningún comportamiento inesperado (GeeksforGeeks, 2021). En este caso se cumplió con este principio por defecto sin presentarse comportamientos inesperados al extender clases de Django como es la clase *Model*.
- **I – Interface Segregation:** Es similar al principio de responsabilidad única. Afirma que no debemos obligar a ningún cliente a implementar una interfaz que sea irrelevante para ellos. El objetivo principal de este concepto es centrarse en evitar las interfaces pesadas y dar preferencia a muchas interfaces pequeñas específicas del cliente (GeeksforGeeks, 2021). No fue necesario la segregación de las clases debido al amplio uso del principio de responsabilidad única.
- **D – Dependency Inversion:** los módulos/clases de alto nivel no deberían depender de módulos/clases de bajo nivel, sino que deberían depender de abstracciones. También se debe asegurar de que la abstracción no debe depender de los detalles, sino que los detalles deben depender de las abstracciones

(GeeksforGeeks, 2021). Por ejemplo, para la creación de un registro de *order_history* [Figura 12Figura 5] se desacopla y flexibiliza la dependencia de las clases *Order_History* y *User* al pasar un objecto *user* como parámetro.

Figura 10. Aplicación del principio de Single Responsibility en la clase *Product.*

```
class Product(models.Model):
    """Product model"""

    group = models.ForeignKey(Group, verbose_name=_('categoría'), on_delete=models.CASCADE)
    unit = models.ForeignKey(Unit, verbose_name=_('unidad'), on_delete=models.CASCADE)
    name = models.CharField(_('nombre del producto'), max_length=100, unique=True, error_messages={'unique': _('Ya existe un producto
    description = models.TextField(_('descripción del producto'), null=True, blank=True)
    total_quantity = models.FloatField(_('cantidad inicial ingresada'), default=0.0)
    quantity_available = models.FloatField(_('cantidad disponible'), default=0.0)
    image = models.ImageField(_('imagen'), upload_to='products/images', null=True, blank=True, validators=[...])
    unit_price = models.IntegerField(_('precio unitario'), null=True, blank=True)
    sales_price = models.IntegerField(_('precio de venta'), null=True, blank=True)
    brand = models.CharField(_('marca'), max_length=100, null=True, blank=True)
    code = models.PositiveIntegerField(_('código de roch asignado al producto'))
    is_consumable = models.BooleanField(_('es de un solo uso?'), default=False)
    qrcode = models.ImageField(_('ruta de la imagen'), upload_to='products/qrcode', null=True, blank=True, validators=[...])
    created_at = models.DateTimeField(_('fecha de registro'), auto_now_add=True)
    updated_at = models.DateTimeField(_('fecha de modificación de registro'), auto_now=True)

    class Meta:...

    def __str__(self):...

    def get_image(self):
        return settings.PROXY_PASS + self.image.url if self.image else None
```

Fuente: Autor

Figura 11. Aplicación de la herencia de clases de acuerdo con el principio Open-Close

```
class Contact(AbstractBaseUser, PermissionsMixin):
    """User model"""

    class IdentificationType(models.TextChoices):
        CC = 'CC', _('Cédula de Ciudadanía')
        CE = 'CE', _('Cédula de Extranjería')

    entity = models.ForeignKey(Entity, verbose_name=_('Entidad/Organización'), on_delete=models.CASCADE)
```

Fuente: Autor

Figura 12. Aplicación del principio Dependency Inversion.

```
OrderHistory(state=Order.States.FULL_DELIVERY, order=phase_order.order_id,
             user=info.context.user).save()
```

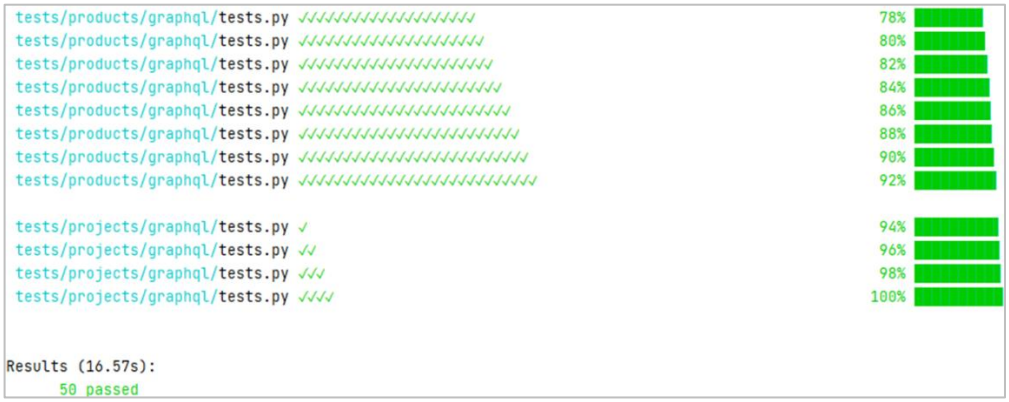
Fuente: Autor

Además de aplicar los principios SOLID se usaron otras buenas prácticas de programación como el uso de nombres significativos en clases y métodos; métodos con una funcionalidad específica; reducción de declaraciones condicionales para disminuir la complejidad del código mejorando la comprensión y mantenibilidad; documentación, reutilización y optimización de código; uso de la convención de sintaxis del lenguaje Python y pruebas unitarias.

6.3.2. Pruebas

Se elaboraron pruebas unitarias de las funcionalidades de cada aplicación a medida que se iban desarrollando. En la siguiente figura [Figura 13] se muestra los resultados finales de la ejecución de las pruebas obteniéndose un resultado exitoso del 100% de 50 pruebas existentes.

Figura 13. Ejecución exitosa de pruebas unitarias



Fuente: Autor

6.3.3. Gestión de cambios

Se realizaron correcciones generales a la lógica de negocio en diferentes aplicaciones a medida que se realizaba la implementación de esta con ayuda de las pruebas.

6.3.4. Despliegue del software en servidor de desarrollo

Se desplegó el software desarrollado en un servidor remoto de desarrollo para hacer las respectivas configuraciones de la tal manera que funcione correctamente cuando se despliegue a producción como también para que los desarrolladores *frontend* fueran consumiendo la API.

A continuación, se muestran las entradas, instrumentos/herramientas, entregables y tiempo de la fase de Implementación.

Tabla 5. Fase 3. Implementación

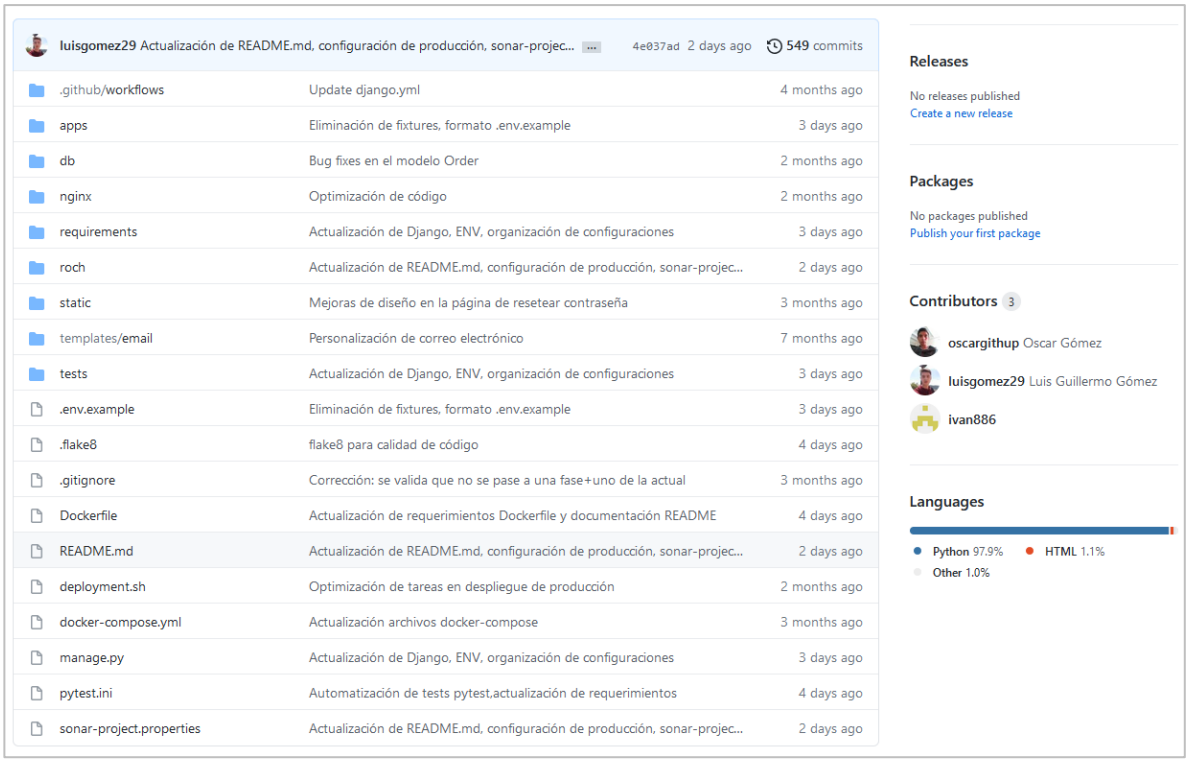
Tipo	Elementos
Entradas	• Archivo *.ldm* del diseño entidad-relación de la base de datos
Instrumentos y/o herramientas	• Computador • Conexión a internet • Navegador Web • Cuenta de Google (acceso a la carpeta compartida) • Power Designer • PyCharm • Git

	• Postman
Entregables	Software el cual cumple con las necesidades y requerimientos especificados por la empresa.
Tiempo	9 semanas

Fuente: Autor

Esta fase tiene como resultado el código fuente dispuesto en el repositorio remoto (GitHub) gestionado por la empresa [Figura 14].

Figura 14. Repositorio en GitHub del código fuente del software



Fuente: Autor

6.4. Fase 4. Pruebas y Validación

El desarrollo de la fase de pruebas y validación estuvo compuesto de las siguientes actividades:

6.4.1. Pruebas de calidad y cumplimiento

Para el cumplimiento de las pruebas de calidad se realizaron por medio de la herramienta de SonarQube, en el cual se arrojaron los resultados.

Atributos internos

Los atributos internos son características propias del software que son medidos examinando el proceso, producto o proyecto mismo.

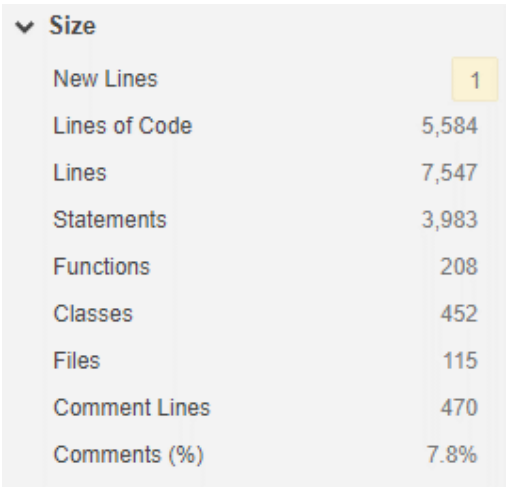
A continuación, se da a conocer la descripción de cada uno de los atributos internos para la evaluación del software:

- **Tamaño del sistema y código fuente**

  La aplicación SonarQube cálculo líneas de código, funciones, clases, archivos, líneas comentadas, declaraciones, entre otros [Figura 15] propios y no externos como dependencias indicando el tamaño del software.

  El proyecto tiene una densidad total de comentarios de 7.8% debido a que solo las funcionalidades complejas se documentaron mientras las otras no ya que tenían código legible.

Figura 15. Tamaño de sistema y código fuente del software



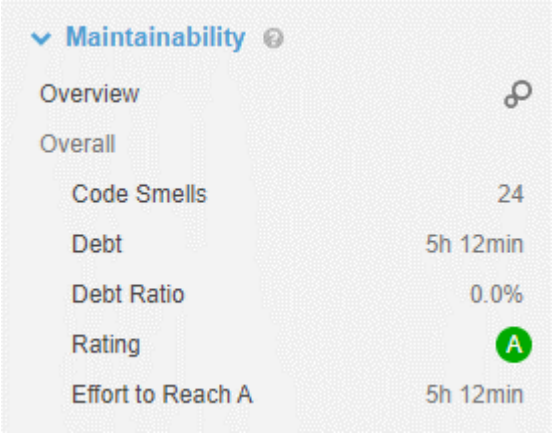
Fuente: Autor

- **Mantenibilidad**

  La mantenibilidad es la característica que “representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas”. (ISO25000, 2021)

  Se obtuvo una calificación positiva A indicando que no es demasiado complejo aplicar procesos correctivos.

Figura 16. Mantenibilidad del código



Fuente: Autor

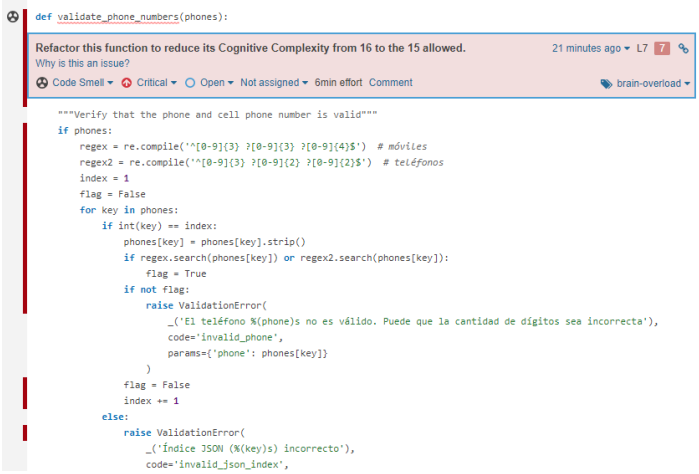
- **Code Smells**

  SonarQube define a Code Smells (2021) como:

  > Un problema de mantenimiento del código. Dejarlo como está significa que, en el mejor de los casos, los mantenedores tendrán más dificultades de las que deberían para hacer cambios en el código. En el peor de los casos, estarán tan confundidos por el estado del código que introducirán errores adicionales al realizar los cambios.

  La cantidad de *code smells* encontrados fueron 12, entre ellos comentarios, corrección de sintaxis y definición de constantes. En la figura [Figura 17] se muestra parte del código donde hay un *code smell* debido a que tiene una complejidad ciclomática de 16 donde se permite máximo 15.

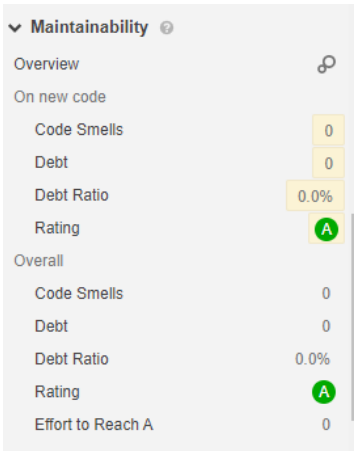
Figura 17. Código Smell donde la complejidad ciclomática es más alta de la permitida



Fuente: Autor

Posteriormente se realizaron las correcciones pertinentes de los *code smells* y se realizó nuevamente el análisis en *SonarQube* [Figura 18]

Figura 18. Inexistencia de *code smells* después de aplicar correcciones



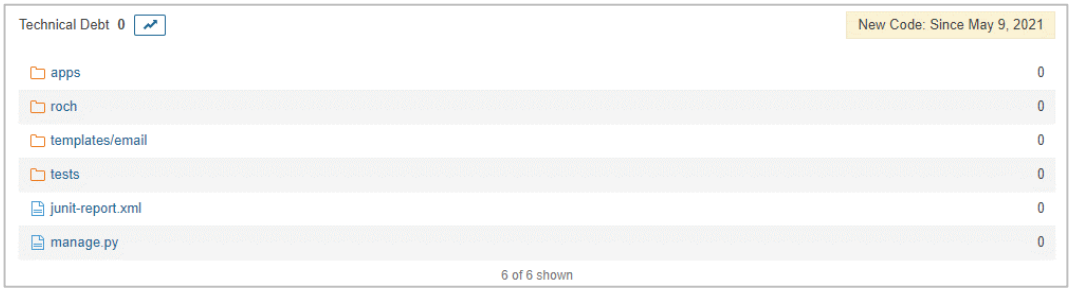
Fuente: Autor

- **Deuda técnica**

  Según SonarQube (2021) la deuda técnica "es el esfuerzo para corregir todos los *code smells*. La medida se almacena en minutos en la base de datos. Se asume un día de 8 horas cuando los valores se muestran en días".

  Inicialmente se obtuvo una deuda técnica de 5 horas con 12 minutos debido a doce *code smells* y posteriormente se redujo a cero con la corrección de estos [Figura 19].

Figura 19. Deuda técnica del software

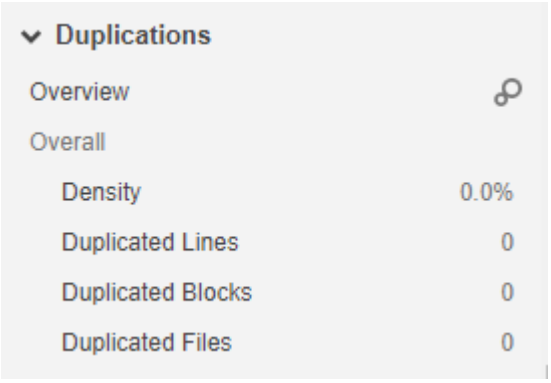


Fuente: Autor

- **Duplicidad de código**

  La duplicidad de código calculado por SonarQube es de 0 líneas lo cual indica que el software es de calidad.

Figura 20. Duplicidad de código del software



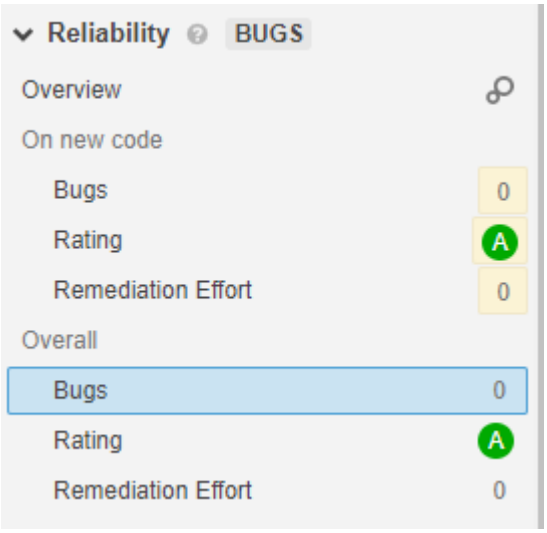
Fuente: Autor

- **Confiabilidad**

  La confiabilidad es la “capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados”. (ISO25000, 2021)

  Se obtuvo una calificación positiva A indicando que no se encuentra ningún *bug* en el software por lo que no es necesario aplicar procesos correctivos.

Figura 21. Fiabilidad del código



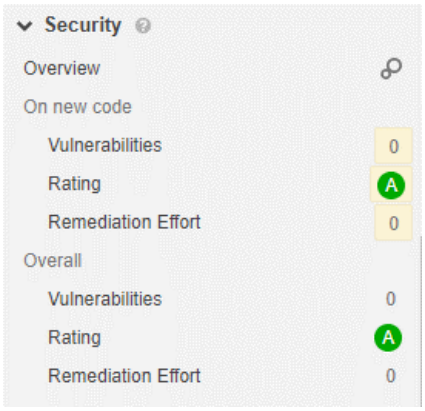
Fuente: Autor

- **Seguridad**

  La seguridad es la “capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos” (ISO25000, 2021).

Se obtuvo una calificación positiva A indicando que no se encuentra ninguna vulnerabilidad en el software por lo que no es necesario aplicar procesos correctivos.

Figura 22. Seguridad en el código



Fuente: Autor

A continuación, se muestran las entradas, instrumentos/herramientas, entregables y tiempo de la fase de pruebas y validación.

Tabla 6. Fase 4. Pruebas y validación

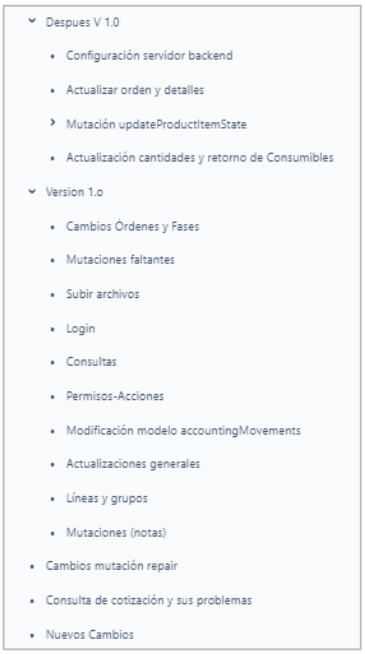
Tipo	Elementos
Entradas	• Software desarrollado
Instrumentos y/o herramientas	• Conexión a internet • Navegador web • Computador • SonarQube
Entregables	Resultados de análisis SonarQube.
Tiempo	4 días

Fuente: Autor

6.5. Fase 5. Documentación

Una vez implementado y probado el software se elaboró la documentación más importante de la API [Figura 23] relacionadas con los módulos de órdenes, proyectos y contactos como también la subida de archivos [Figura 24]. Esta documentación fue elaborada en la aplicación *Confluence (Atlassian)* debido a la estética, organización y trabajo en equipo que esta facilita.

Figura 23. Listado de documentación en Confluence



Fuente: Autor

Figura 24. Parte de documentación de como subir archivos con la API

Fuente: Autor

Adicionalmente se documentó cómo realizar el despliegue del software en desarrollo y producción [Figura 25].

Figura 25. Documentación para despliegue del software a producción

Tests

Se utiliza `pytest` como framework de prueba. Consultar la documentación de pytest para mas opciones al correr los tests.

Para correr los tests ejecutar:

```
pytest
```

Despliegue a producción con Docker

1. Ingresar a la carpeta del proyecto.

```
cd /home/roch_inventario/roch_python
```

2. Hacer backup de la base de datos.
3. Configurar variables de entorno en el archivo `.env` (Ver el archivo `.env.example`).
4. Ejecutar el archivo `deployment.sh`.

```
. deployment.sh
```

Iniciar contenedores Docker

Para iniciar los contenedores detenidos ejecutar el comando:

```
docker start $(docker ps -aqf "name=roch_python")
```

Fuente: Autor

A continuación, se muestra las entradas, instrumentos/herramientas, salidas (entregables) y tiempo de la fase de Documentación.

Tabla 7. Fase 5. Documentación

Tipo	Elementos
Entradas	• Software desarrollado
Instrumentos y/o herramientas	• Computador • Conexión a internet • Navegador Web • Cuenta de Google (acceso a la carpeta compartida) • Cuenta de Atlassian
Salidas/Entregable	Documentación final de la API/Software
Tiempo	2 semanas

Fuente: Autor

6.6. Fase 6. Despliegue

Se desplegó el software a producción en el servidor dispuesto por el cliente donde previamente se hicieron las respectivas configuraciones de variables de entorno como de Docker [Figura 26].

Figura 26. Contenedores Docker ejecutándose

Fuente: Autor

A continuación, se muestra las entradas, instrumentos/herramientas, salidas (entregables) y tiempo de la fase de Despliegue.

Tabla 8. Fase 6. Despliegue

Tipo	Elementos
Entradas	• Software
Instrumentos y/o herramientas	• Computador • Conexión a internet • Terminal de línea de comandos • Credenciales de acceso
Salidas/Entregables	Software ejecutándose en modo producción.
Tiempo	5 días

Fuente: Autor

7. CONCLUSIONES

- Se desarrolló con éxito el sistema que cumple con las historias de usuario utilizando buenas prácticas de programación orientada a objetos y del framework Django. Lo que más ayudó a desarrollar el software fue el uso de la metodología SCRUM porque permitió un mayor control de las historias de usuario, la asignación y priorización de actividades, comunicación y retroalimentación constante entre las partes interesadas (*stakeholders)* como también el uso de pruebas unitarias que aseguraron la calidad del software y agilizaron la implementación de mismo ya que al momento de hacer optimización de código, cambiar o agregar nuevas funcionalidades o actualización de dependencias es más fácil verificar que el software continúe funcionando como se espera y en caso de que haya errores permite detectarlos rápidamente.
- Se determinó el alcance del sistema para definir las necesidades mediante el uso de las historias de usuario las cuales fueron importantes para el desarrollo de este proyecto ya que nos facilitaron identificar y analizar los requisitos del software funcionales y no funcionales y las restricciones de estos permitiendo una compresión clara de las necesidades del cliente a resolver.
- Se implementó la base de datos que permite gestionar la información del inventario con base a las historias de usuario. Lo más importante para lograr la implementación en motor de bases de datos PostgreSQL fue el diseño entidad-relación porque define todo el esquema de la base de datos como también fue el uso del framework Django porque incluye un sistema de migraciones que permite modificar el esquema de la base de datos de forma reproducible y consistente.

8. REFERENCIAS

Pintelon , C., & Van, H. (2015). Development of a risk assessment selection methodology for asset maintenance decision making: An analytic network process (ANP) approach. *International Journal of Production Economics*.

Álvarez, L., Avendaño, H., García, Y., & Morales, S. (17 de Junio de 2021). *Programación Orientada a Objetos*. Obtenido de ferestrepoca.github.io: https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/philosophy.html

Castañeda Ramíres, Y. A., & Silva Vargas, D. A. (2013). *Implementación de un sistema de gestión de inventarios en Melexa S.A.* Recuperado el 17 de February de 2021, de https://repository.unilibre.edu.co/bitstream/handle/10901/9430/DOCUMENTO%20FINAL.pdf?sequence=1&isAllowed=y

Django Framework. (10 de Mayo de 2021). Obtenido de https://www.djangoproject.com/

Django Framework. (10 de Mayo de 2021). *Applications*. Obtenido de https://docs.djangoproject.com/es/3.2/ref/applications/#projects-and-applications

Django Framework. (10 de Mayo de 2021). *Models*. Obtenido de Documentation: https://docs.djangoproject.com/en/dev/topics/db/models/

GeeksforGeeks. (17 de Junio de 2021). *Best Practices of Object Oriented Programming (OOP)*. Obtenido de GeeksforGeeks: https://www.geeksforgeeks.org/best-practices-of-object-oriented-programming-oop/

ISO25000. (10 de May de 2021). *ISO25000*. Obtenido de Fiablidad: https://iso25000.com/index.php/normas-iso-25000/iso-25010/24-fiabilidad

ISO25000. (12 de Mayo de 2021). *ISO25000*. Obtenido de Seguridad: https://www.iso25000.com/index.php/normas-iso-25000/iso-25010/25-seguridad

ISO25000. (12 de Mayo de 2021). *ISO25000*. Obtenido de Mantenibilidad: https://www.iso25000.com/index.php/normas-iso-25000/iso-25010/26-mantenibilidad

Mago, M., Perea, B., & López, H. (2020). Implementación de mantenimiento preventivo y predictivo a los equipos del proceso de producción en la empresa EQUIACEROSSAS. *Ingenio Libre*, 10.

Rubira, J. (18 de Junio de 2021). *Solid, cinco principios básicos de diseño de clases*. Obtenido de Gembeta:dev: https://www.genbeta.com/desarrollo/solid-cinco-principios-basicos-de-diseno-de-clases

SonarQube. (10 de Mayo de 2021). *SonarQube*. Obtenido de Concepts: https://docs.sonarqube.org/8.9/user-guide/concepts/

SonarQube. (10 de Mayo de 2021). *SonarQube Docs*. Obtenido de Technical Debt: https://docs.sonarqube.org/latest/user-guide/metric-definitions/#header-4

9. ANEXOS

Anexo 1. Actas de reuniones

Enlace: https://1drv.ms/u/s!AkYpCgfYWnoJg8Bvh5hIaHvC9DvQZw?e=M8Udc8

Anexo 2. Historias de usuario

Enlace: https://1drv.ms/u/s!AkYpCgfYWnoJg8BwepwGXiYh72QmkQ?e=cp2ZXp

Anexo 3. Diagrama entidad-relación de la base de datos final
Enlace: https://1drv.ms/u/s!AkYpCgfYWnoJg8JUaPR77nZtVHvLOQ