Departamento de Informática
Universidad Técnica Federico Santa María

Informe de Proyecto – INF-225-2018-1-CSJ
Proyecto "Maestros chasquillas"
29-08-2018

Objetivo

La empresa GPI carece de un sistema que organice el proceso de solicitudes de materiales desde una obra hasta la empresa, por lo cual se nos ha pedido implementar un sistema que pueda cumplir con las expectativas de la empresa en torno a este ambiente. Para esto se nos solicitó poder generar un manejo del inventario de la empresa y de las solicitudes que se generan durante la comunicación interna, también se nos solicitó generar una comunicación más eficiente entre proveedores y empresa, tener un registro de plazos de entrega, el cual maneja e informa atrasos en cualquier parte del flujo del proceso, virtualizar el trabajo del Centro de negocios al generar solicitudes RE-ADQ-101, ya que todos estos procesos se hacen a mano actualmente, y finalmente automatizar el proceso de generar órdenes de compra.

Para desarrollar este proyecto escogimos un framework de desarrollo basado en el patrón MVC (modelo-vista-controlador), pues los resultados que se pueden obtener a partir del uso de este pueden cumplir de manera satisfactoria los requisitos del proyecto. El framework de desarrollo escogido es Django, pues se basa en el lenguaje de programación Python. Lenguaje muy potente que podría en un futuro agregar módulos complejos tales como análisis de datos o inteligencia artificial que son proveídos por sus librerías. Algo que también hay que comentar es que Django cuenta con una buena documentación y disponemos de un libro de buenas prácticas para construir sistemas robustos.

También escogimos la base de datos Postgres porque es una base de datos relacional (esto se ajusta al proyecto) de fácil instalación y tiene un buen comportamiento.

Cabe destacar también que para facilitar el despliegue del proyecto en un vps se han instalado las tecnologías anteriormente descritas en Dockers y se han orquestado con ayuda de Docker compose. Lo anterior significa que podemos levantar el proyecto con una simple conexión por ssh al vps más un "git clone" más el comando único "docker-compose up" facilitando el proceso de dev-ops.

Integrantes:

Nombres y Apellidos	Email	ROL USM
Gabriel Guzmán	gabriel.guzman.13@sansano.usm.cl	201360563-k
Macarena Hidalgo	macarena.hidalgo.14@sansano.usm.cl	201473608-8

1. Requisitos clave (Final)

Tabla 1: Requisitos funcionales (Finales)

Req. funcional	**Descripción y medición (máximo 2 líneas)**
Generar solicitudes.	El centro de negocios podrá generar solicitudes REQ-ADQ-101.
Comprobar stock en bodega para solicitudes.	Se podrá comparar las existencias de ciertos materiales en la bodega con los pedidos en REQ-ADQ-101.
Registros de solicitudes.	Seguimiento de materiales solicitados en REQ-ADQ-102, organizados por ítems.
Asignar fechas de entrega estimadas a las solicitudes.	Bodeguero podrá asignar fechas estimadas de entrega de materiales para REQ-ADQ-101.
Asignar prioridad a materiales.	Bodeguero podrá asignar prioridades de materiales para una solicitud.
Cotizar con proveedores.	Encargado de compras podrá enviar solicitudes de cotizaciones por email a proveedores.
Órdenes de compra.	Encargado de compras genera solicitudes REQ-ADQ-103.
Registro de usuario.	El administrador tendrá acceso a crear usuario, al cual se le enviará un mail de confirmación para crear contraseña.
Control de usuarios.	El administrador podrá ver todos los usuarios activos en el sistema y podrá eliminarlos de ser necesario.
Creación de bodegas.	El administrador podrá crear bodegas y asignarles un bodeguero.
Creación de obras.	El centro de negocios podrá crear una obra.
Reservar material de bodega.	El bodeguero podrá reservar material que se encuentre en el inventario.
Liberar material de bodega.	El bodeguero podrá liberar un material ya reservado.

Tabla 2: Requisitos extra-funcionales (Finales)

Req. extra-funcional	**Descripción y medición (máximo 2 líneas)**
Usabilidad.	programa sencillo de usar, adaptación < 2 semanas.
Eficiencia.	El sistema debe ser capaz de procesar N solicitudes por segundo.
Seguridad.	Los permisos de acceso al sistema podrán ser cambiados solamente por el administrador.
Dependibilidad.	El sistema debe tener una disponibilidad del 99,99% de las veces en que un usuario intente accederlo.

2. Árbol de Utilidad (Final)

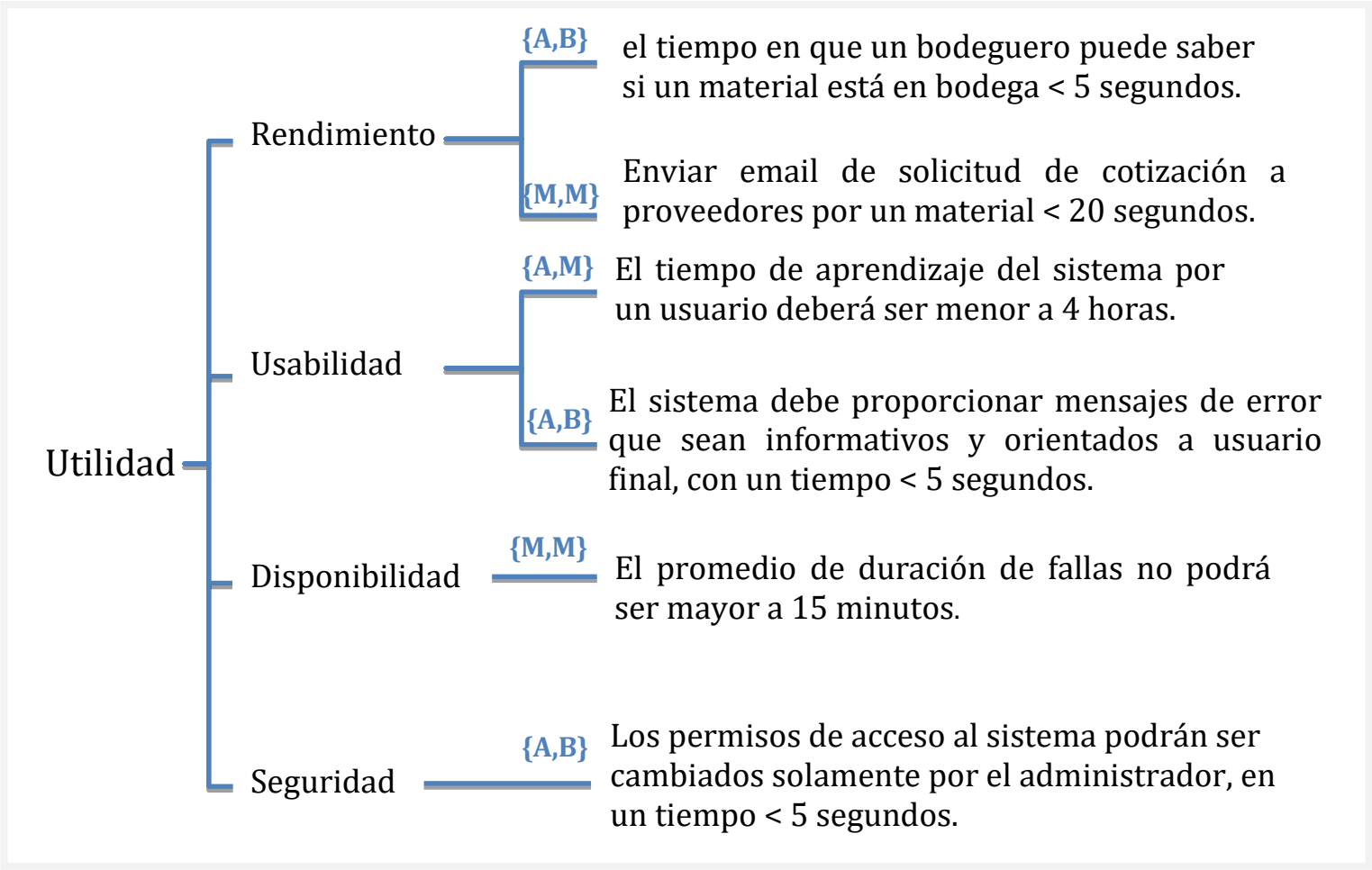

Figura 1: Árbol de utilidad Final. {A: Alta, M: Media, B: Baja}.

3. Modelo de Software (Final)

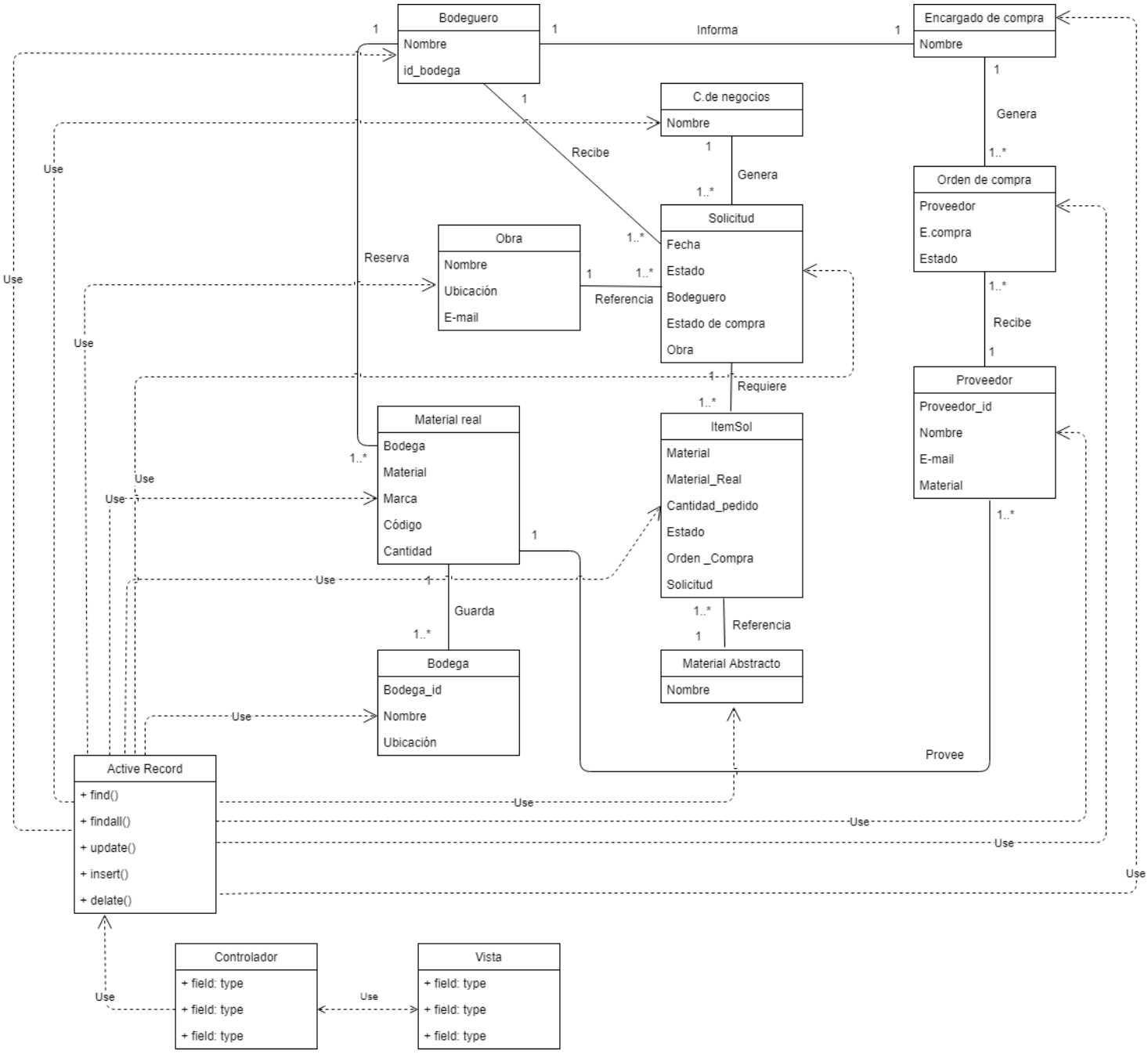

Tabla 3: Selección de Patrones

Intención	**Patrón de Diseño**	**Razonamiento**
Reflejar el estado de las solicitudes.	Model-View-Controller (MVC)	MVC permite cargar HTML dinámico que muestra el estado de las solicitudes según la información disponible en la base de datos.
Facilitar el uso de la base de datos a través de un lenguaje de mayor nivel que los comandos SQL.	ORM	Para disminuir la probabilidad de ocurrencia de errores durante la gestión de la base de datos se utilizan las funciones que provee Django.
Facilitar el despliegue del sistema en un vps (dev ops).	Microservicio en contenedores	Para implementar el proyecto será necesaria la contratación de un vps sobre el cual instalar las dependencias del proyecto a través de una conexión ssh. Para simplificar esta tarea se utiliza Docker con Docker-compose para levantar el proyecto con un único comando para instalar todas las librerías y gestionar todos los componentes: servidor, base de datos y sistema web tratando a cada uno de estos componentes como un microservicio individual.
Se desea implementar un controlador por vista.	Page controller.	Debido a que cada página web dinámica es manejada por un controlador específico, estos pueden seguir siendo simples.
Las clases del dominio deben poder comunicarse entre sí de manera adecuada, al mismo tiempo que el usuario activo debe poder buscar, visualizar y acceder a los distintos recursos de la plataforma.	Model-View-Controller (MVC)	MVC es práctico en este sentido, pues permite acceder fácilmente a los distintos objetos de la aplicación. Satisface distintos intereses de usuarios. Mismo modelo, pero con distintos públicos objetivos

4. Trade-offs entre tecnologías (Final)

Tecnologías elegidas:

1) Usamos Django en vez de Flask
2) Usamos Docker compose en vez de kubernete
3) Usamos Postgres en vez de MySQL

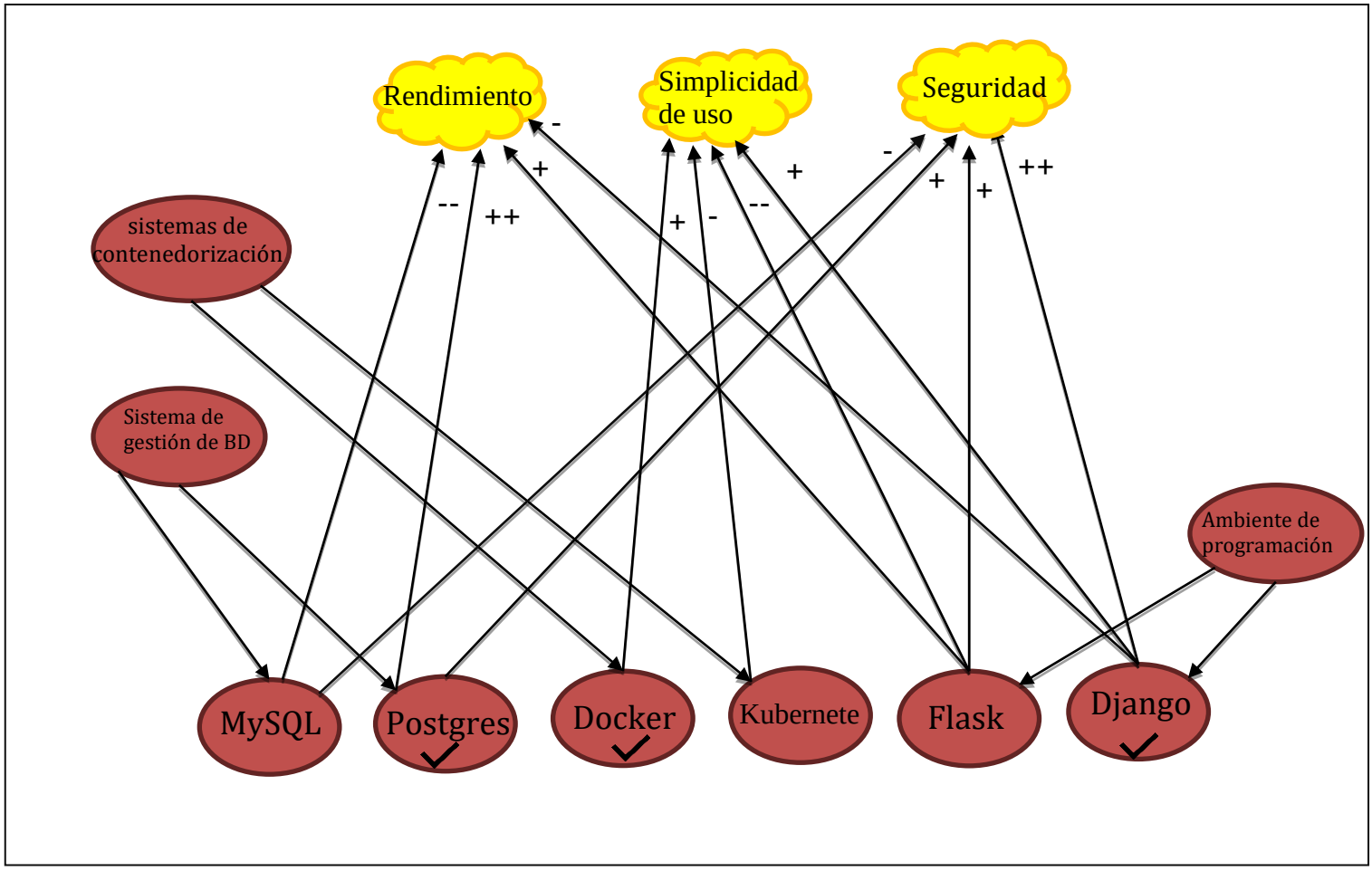

Figura 3: Softgoals.

Tabla 9: Trade-offs entre opciones tecnológicas

Decisión	Softgoal	Evaluación	Razonamiento
Django	Rendimiento	-	Lento a la hora de renderizar.
Django	Seguridad	++	Django provee de protección de secuencias de comandos entre sitios (XSS), protección contra falsificación de solicitudes cruzadas (CSRF), protección de inyección SQL, protección de clickjacking, seguridad de la sesión y validación de encabezado de host
Django	Simplicidad	+	Fantástica documentación y mil ejemplos en la red. Además, se basa en el lenguaje de programación Python.
Flask	Rendimiento	+	Rápido. A la hora de renderizar puede ser hasta 3 veces más rápido que Django.
Flask	Seguridad	+	Flask nos da la posibilidad de tener sesiones, algo similar al $_SESSION[variable] de PHP. La diferencia es que en PHP la cookie de sesión es un identificador aleatorio, y en el servidor se guardan las variables de sesión correspondientes a ese y otros identificadores. En flask, la cookie de sesión tiene el contenido de todas estas variables en JSON, comprimido y cifrado simétricamente con una clave secreta que se elige en la configuración de la aplicación. Esto significa que si uno conoce o averigua la clave secreta sería capaz de modificar todos los datos de la sesión, es decir, el usuario con el que se logueó, sus privilegios, etc. Datos que no son menores y que podrían causar severos fallos de seguridad.
Flask	Simplicidad	--	Pocos libros y complicado realizar tareas sencillas. Como migraciones, Unit Test o internalizaciones.
Docker	Simplicidad	+	Con un simple comando “docker-compose up” se crea todo lo necesario para que la aplicación corra. Además de ser muy simple de instalar.
Kubernete	Simplicidad	-	Es de más alto nivel, por lo cual se hace más complicado su entendimiento.

Postgres	Rendimiento	++	El rendimiento de PostgreSQL se aprovecha mejor en sistemas que requieran ejecución de consultas complejas. PostgreSQL rinde bien en sistemas OLTP/OLAP cuando se requiere velocidad en lectura/escritura y se necesita análisis extenso de datos.
Postgres	Seguridad	+	PostgreSQL tiene ROLES y roles heredados para establecer y mantener los permisos. PostgreSQL tiene soporte nativo para SSL en conexiones para cifrar la comunicación cliente/servidor. También tiene seguridad a nivel de registros. Además de esto, PostgreSQL viene con una mejora integrada llamada SE-PostgreSQL, la cual provee controles de acceso adicionales basados en las políticas de seguridad de SELinux. (más completo que MySQL).
MySQL	Rendimiento	--	MySQL no rinda bien cuando se somete a cargas pesadas o al intentar completar consultas complejas.
MySQL	Seguridad	-	MySQL implementa seguridad basada en Listas de Control de Acceso (ACLs) para todas las conexiones, consultas y otras operaciones que un usuario pudiera intentar realizar. También hay algo de soporte para conexiones cifradas con SSL entre clientes y servidores MySQL.

5. Deuda técnica incurrida

Ítem deuda técnica	Razonamiento	Impacto	Enfoque de corrección
URL's inseguras entre bodegueros.	No existe una validación en la sección de URL's de Django que permita que cada bodeguero pueda modificar únicamente las solicitudes que les son asignadas.	Permite que se puedan modificar los estados de las solicitudes e inventario entre bodegueros a través de comandos URL. (EJ: bodeguero 1 puede liberar material X reservado por bodeguero 2).	En un día se puede generar una función de validación que se utilice en la sección de URL's de Django para permitir la ejecución de funciones de gestión de solicitudes solo a los bodegueros que tienen dichas solicitudes asignadas.
Escasa documentación en código.	En las líneas de código no hay mucha documentación sobre la forma en que se ejecutan las funciones más complejas.	Para modificar componentes es necesario volver a leer la documentación de ciertas funciones que no son auto explicativas.	En un día se pueden buscar las funciones más complejas con sintaxis o lenguaje no auto explicativo y documentar.
No uso de variables de entorno.	No se usan variables de entorno en la raíz del proyecto.	El nombre de la base de datos, superusuario, claves, emails de sistema, etc. Se encuentran escritas de forma directa en el código dificultando el refactoring y la visualización de información.	En un día se pueden definir las variables generales del sistema e incluirlas en el código por medio de variables de entorno (env).
Reuso de código css.	Muchas veces el código css se repite. (EJ: el código .css para generar tablas es el mismo para todos los tipos de usuarios, sin embargo, se vuelve a definir una y otra vez en cada carpeta de templates de cada tipo de usuario).	Si se quieren modificar una componente visual genérica para todos los tipos de usuarios cómo por ejemplo los colores y proporciones de los botones del dashboard, será necesario modificar el código .css en diferentes partes del software.	En un día se puede implementar la utilización de un template base para todos los tipos de usuarios. En este template base se deben incluir los archivos .css generales compartidos.

Tabla 10: Deudas Técnicas.