Tarea1 – MyTask

Pablo campos - pablo.camposa@usm.cl
Luis Zegarra - luis.zegarra@usm.cl

Enlace repositorio en GitHub: Repositorio de Equipo 13

¿Cómo especificarías mejor el requerimiento? (Validación)

El punto principal es que el cliente requiere una aplicación "sencilla" para la gestión de sus tareas. Por "sencilla", hemos acordado implementar una solución que cumpla con los principios de usabilidad, sea fácil de usar y útil para el usuario final, conforme a los requerimientos del cliente. Todo esto será desarrollado en una arquitectura monolítica con un almacenado de datos de manera local.

Gestión de Tareas

En este apartado se solicitan las implementaciones básicas para el manejo de datos, comúnmente conocidas como operaciones CRUD (Crear, Consultar, Actualizar, Borrar). A continuación, se detallan cada una de las funcionalidades:

- **Crear**: Se debe permitir la creación de una TAREA que contenga los siguientes campos en formato string:
  - **Título**: Campo de texto obligatorio. No puede estar vacío.
  - Descripción: Campo opcional. Puede estar vacío.
  - E**stado**: Los estados predefinidos serán "pendiente", "en proceso", "completada" y "atrasada" para mantener la organización.
  - **Fecha de vencimiento**: Se debe ingresar en formato de fecha `%Y-%m-%d`. Puede estar vacío, pero se le asignará un valor predeterminado para facilitar la comprobación de tareas "atrasadas".
  - **Etiqueta**: Las etiquetas estarán preestablecidas y el usuario no podrá modificarlas ni editarlas.
  - F**echa de creación**: Se almacenará la fecha de creación de la tarea, ya que es un dato útil para el usuario.
  - Usuario: Nombre del usuario que crea la tarea, lo que permitirá mostrar solo las tareas correspondientes a dicho usuario.

- **Consultar**: Se debe permitir obtener la información de una tarea almacenada para ser utilizada en las funciones necesarias y mostrada por consola al usuario.
- **Actualizar**: Se debe permitir la actualización de todos los campos de las tareas almacenadas, excepto la fecha de creación (campo inalterable).
- **Borrar**: Se debe permitir eliminar una tarea del sistema de almacenamiento utilizado. Esto implica borrado total, sin posibilidad de recuperación, y la eliminación de la visualización de la tarea.

En cuanto a la interacción con estas funciones, se acordó que se realizará mediante la línea de comandos. Como equipo, hemos decidido implementar las funcionalidades de manera que:

- **Aprendizaje Rápido**: El usuario pueda aprender a utilizar las funciones de manera rápida e intuitiva.
- **Eficiencia**: La distribución del programa debe permitir al usuario ejecutar sus actividades de la manera más rápida posible y con la menor cantidad de pasos.
- **Memorización**: En relación con la eficiencia, la organización del programa debe permitir que el usuario no olvide cómo utilizarlo, incluso si deja de hacerlo por un tiempo prolongado.
- **Disminución de Errores**: Se busca minimizar los errores cometidos por el usuario al ejecutar las tareas.
- **Satisfacción**: El sistema debe cumplir con las expectativas del cliente, generando una experiencia satisfactoria.

Filtrado y Búsqueda

Los usuarios podrán filtrar las tareas en función de los siguientes criterios:

- **Título**: Permite buscar tareas por título, siendo insensible a mayúsculas y aceptando una sola palabra que pertenezca al título para mostrar todas las tareas que la contengan.
- **Estado**: Se ofrecerán opciones para elegir el estado por el cual se desea buscar, ingresando la opción por consola.
- **Etiqueta**: Se proporcionarán opciones de etiquetas, y el usuario deberá ingresar por cuál desea filtrar, mediante la consola.
- **Rango de Fechas**: Permite realizar la búsqueda desde la fecha actual hasta la fecha ingresada por el usuario.

Opción de Activar o No los Filtros:

Caso de Activar los Filtros: Si se desea filtrar por título, una vez ingresado, se realizará la búsqueda. Si no se desea buscar por título, se presentarán las demás opciones de filtrado, ya que el título apunta directamente a la tarea específica, mientras que los otros criterios pueden contener más tareas y permitir una búsqueda más amplia o acotada.
Caso de No Activar los Filtros: Se mostrarán todas las tareas existentes para ese usuario, pero de manera más detallada.

Gestión de Estados

Se debe permitir a los usuarios definir los estados de sus tareas. Los estados predefinidos son "pendiente", "en progreso", "completada" y "atrasada". Al crear una tarea, se le asignará automáticamente el estado "pendiente". Posteriormente, el usuario podrá cambiar el estado a "pendiente", "en progreso" o "completada". El estado "atrasada" se asignará automáticamente si la fecha de vencimiento de la tarea es menor que la fecha actual.

Autenticación

Para cumplir con este requerimiento, se definió la creación de un sistema de usuarios para acceder a las funcionalidades de la aplicación. El usuario deberá registrarse, y su información será almacenada en una base de datos. Las tareas estarán vinculadas al usuario que las crea, para evitar mezclarlas con las de otros usuarios. Es decir, se pueden registrar varios usuarios, y cada uno tendrá acceso personal a sus tareas.

Validación

Para asegurar que estamos resolviendo el problema correcto, se tendrán en cuenta las especificaciones dadas y las respuestas proporcionadas por el cliente. Se desarrollará un producto mínimo viable con las características solicitadas de manera funcional, escalando progresivamente el programa para hacerlo más atractivo y completo.

¿Cómo asegurarías que el programa cumpla el requerimiento? (Verificación)

Una vez implementado una base sólida de la aplicación, habrá que verificar que estamos resolviendo el problema de la manera correcta. Para esto, se realizarán pruebas unitarias para cada una de las funcionalidades implementadas, asegurándonos de que cada funcionalidad opere de manera independiente. Posteriormente, se integrarán las funcionalidades para que trabajen de manera conjunta dentro de la aplicación.

Estas pruebas se registrarán en una hoja de cálculo de Excel, donde se documentarán los pasos seguidos y los resultados obtenidos. A medida que se obtengan los resultados, podremos comprobar si las implementaciones cumplen con los requerimientos solicitados. Esto nos permitirá corregir errores o buscar una mejor implementación si es necesario.

Organización del Proyecto y Flujo de Trabajo

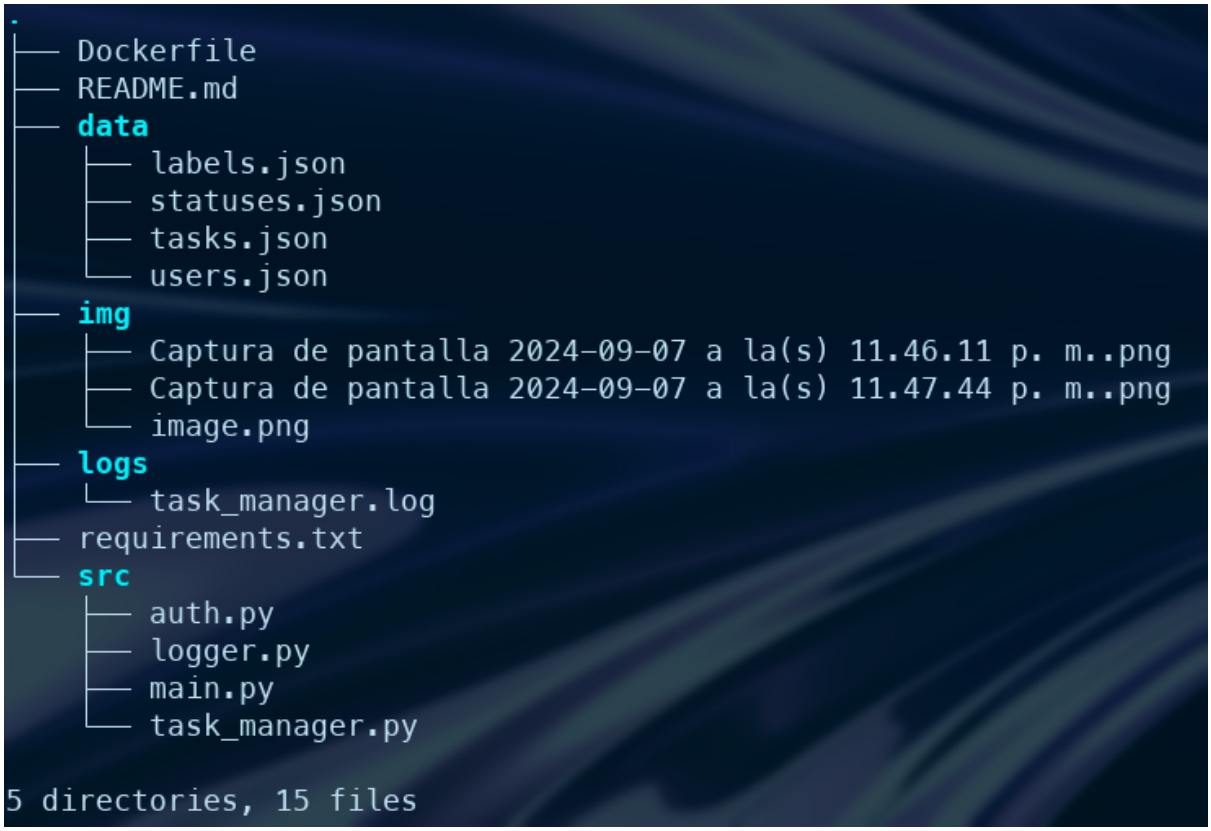
Organización del Proyecto

Al ser una aplicación con poca complejidad en el manejo de archivos y funciones, decidimos implementar una arquitectura monolítica por su simplicidad en el desarrollo y la gestión. Además, al estar todo centralizado, es más fácil realizar pruebas integradas y corregir errores. Al ser monolítica, también ofrece la facilidad de implementar un único punto de despliegue, como podría ser la ejecución manual del `main.py` o mediante un contenedor Docker.

Como equipo, dividimos las tareas para asegurar una buena organización, utilizando las herramientas de Jira y Slack para mantener el orden en el desarrollo. No asignamos roles específicos, permitiendo que cada miembro aportara según su disponibilidad, ya que contábamos con diferentes bloques de tiempo de trabajo. Para el desarrollo del código, utilizamos un repositorio en GitHub, lo que permitía que cada uno mantuviera sus cambios en ramas distintas. Cuando teníamos una versión funcional final, se sincronizaba con la rama

principal (main). Realizamos pruebas independientes para verificar cómo se implementaban los requisitos de forma aislada, y luego pruebas integrales para comprobar el funcionamiento general del programa.

La estructura final de la aplicación se ha organizado con la siguiente manera:



Directorio `data`

Esta carpeta contendrá los archivos JSON utilizados para el almacenamiento de datos:

- `labels.json`: Contiene las etiquetas predeterminadas disponibles para las tareas. Si se desea agregar o eliminar una etiqueta, esto deberá hacerse en este archivo.
- `statuses.json`: Contiene los estados posibles para las tareas, los cuales están predefinidos.
- `task.json`: Almacena las tareas creadas por los usuarios, funcionando como la "base de datos" de las tareas. Las tareas se almacenan en un formato estructurado.
- `users.json`: Almacena la información de los usuarios registrados y se utiliza para la autenticación y verificación necesarias durante el registro.

Directorio `logs`

Esta carpeta se creará en la primera ejecución del programa en caso de que no exista y almacenará los logs generados durante la ejecución de las funciones dentro de `task_manager.log`. Existen tres tipos de logs:

- Información: Detalla las acciones realizadas por el usuario y las operaciones internas del programa.
- Advertencia (Warning): Registra intentos de realizar acciones no permitidas o que no se pueden ejecutar debido a ciertos problemas. Estos logs son útiles para detectar posibles errores en el código.
- Error: Describe los errores ocurridos, incluyendo un breve mensaje y el tipo de error generado.

El archivo `task_manager.log` se actualizará con cada ejecución del programa y solo puede eliminarse de forma manual.

Directorio `src`

Contiene los archivos principales para la ejecución del programa:

- `auth.py`: Contiene las funciones de registro y autenticación de usuarios, así como funciones para la lectura y escritura en el archivo `users.json`. Utiliza la **librería bcrypt** para la encriptación de las contraseñas.
- `logger.py`: Define las funciones para el registro de logs en `task_manager.log`.
- `main.py`: Archivo principal del programa que coordina la ejecución de las funciones definidas en los otros módulos.
- `task_manager.py`: Define las funciones relacionadas con la gestión y verificación de tareas.

Directorio `img`: Contiene imágenes para el README.md

README.md: Contiene información de problema desarrollado e instrucciones para compilar el programa.

DOCKERFILE: Archivo para crear a imagen en Docker. Instrucciones de como ejecutarlo están en el README.

Flujo de Trabajo

GitHub

Se utilizó un Repositorio de Equipo 13 en GitHub para un control de versiones eficiente y continuo. Inicialmente, se partió con un producto mínimo viable que contenía las funciones principales. Posteriormente, mediante el uso de ramas, se fueron refinando las funcionalidades. Esto permitió implementar una versión que trabaja con la base de datos MongoDB y otra que utiliza archivos JSON para el almacenamiento.

Las ramas se dejaron abiertas para evidenciar el trabajo realizado, y pueden ser revisadas en el mismo repositorio. Sin embargo, la rama que contiene el trabajo principal es `main`, mientras que las demás son de apoyo.

Jira

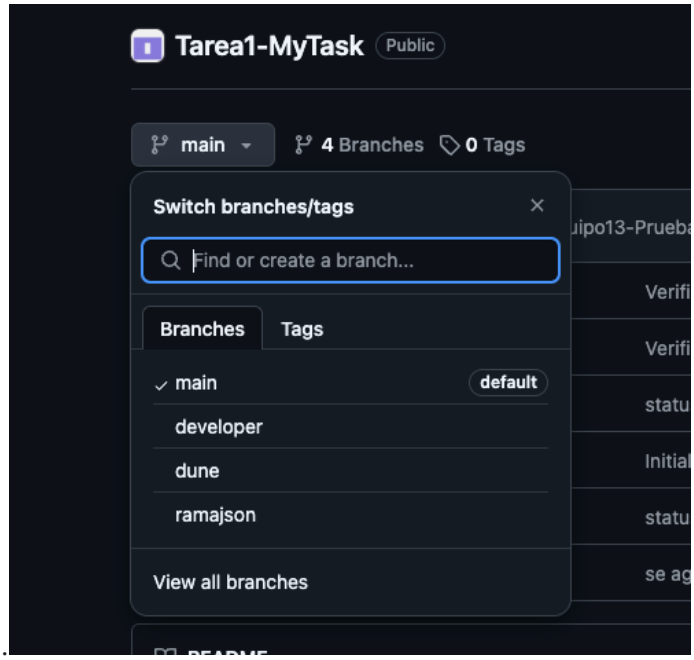
En paralelo, se utilizaron las herramientas de Jira para mantener una lista de tareas (TO-DO list) de las partes que debíamos realizar. Esta lista puede ser consultada en el siguiente enlace: Jira USM TEAM 13

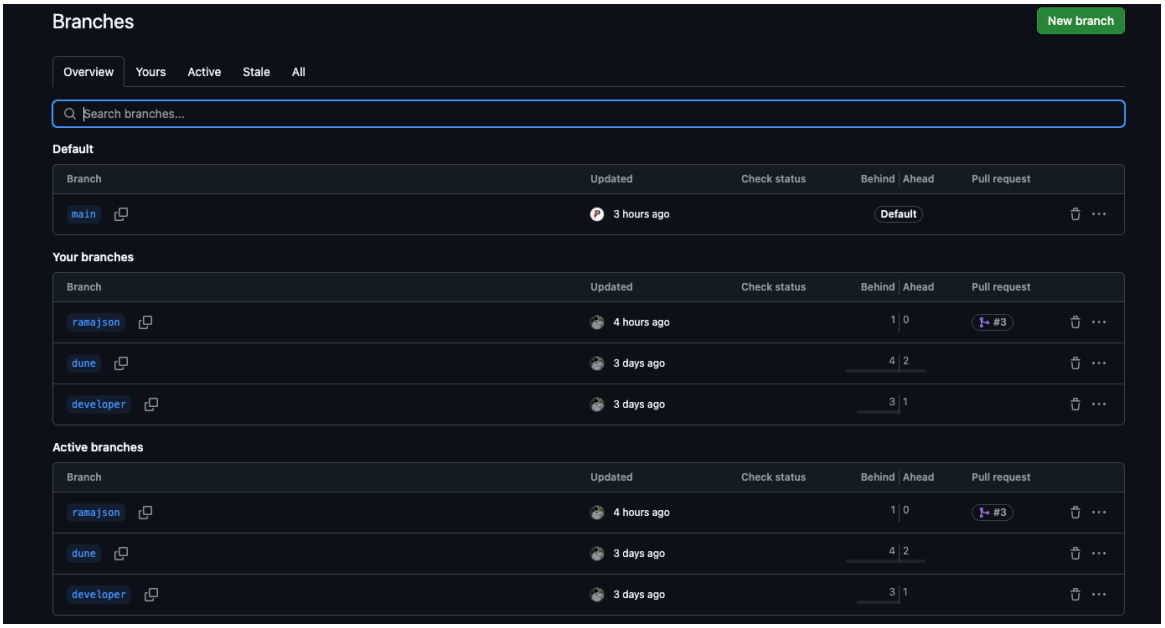
Slack

La herramienta Slack nos ayudó a mantener un registro de los cambios realizados dentro del repositorio de GitHub (New commits, New pull request, code reviews y merges) y cambios efectuados en la tabla de trabajo de Jira entre columnas de estado. Se dispusieron canales separados para ambas conexiones, lo que permitió un conocimiento en tiempo real de los cambios efectuados en cualquiera de las dos plataformas.

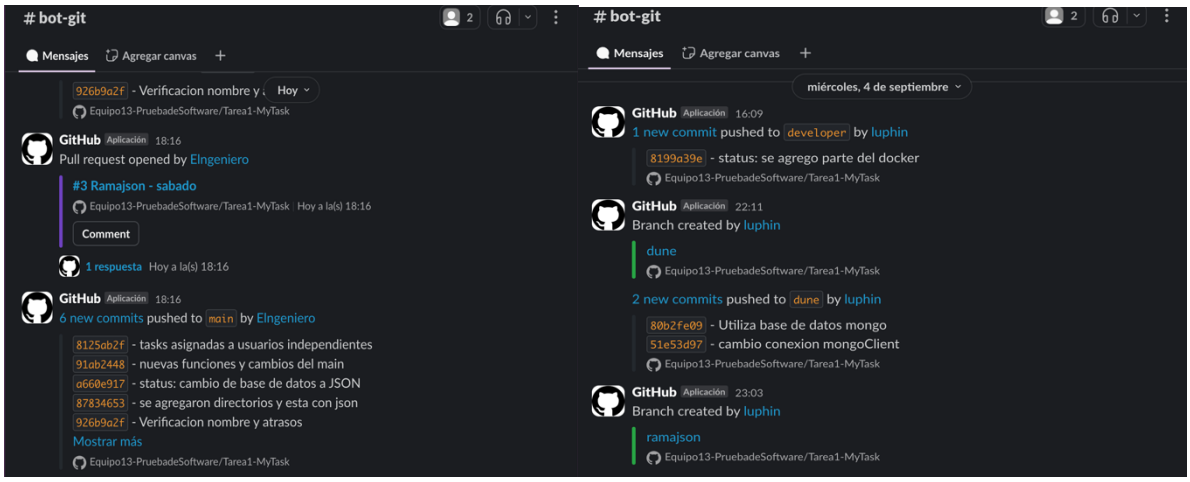
Incluir evidencia de flujo de trabajo y configuraciones realizadas (Imágenes de pantalla).

Ramas creadas en el directorio de GitHub. `main` es la principal y la demás son de desarrollo





Integración de notificaciones de Jira y GitHub en Slack



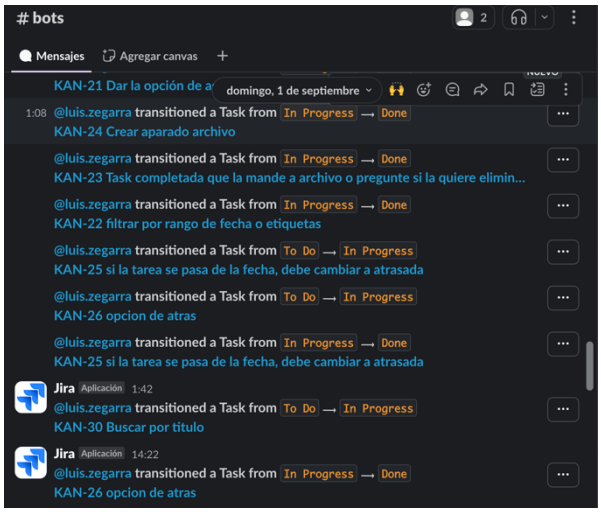
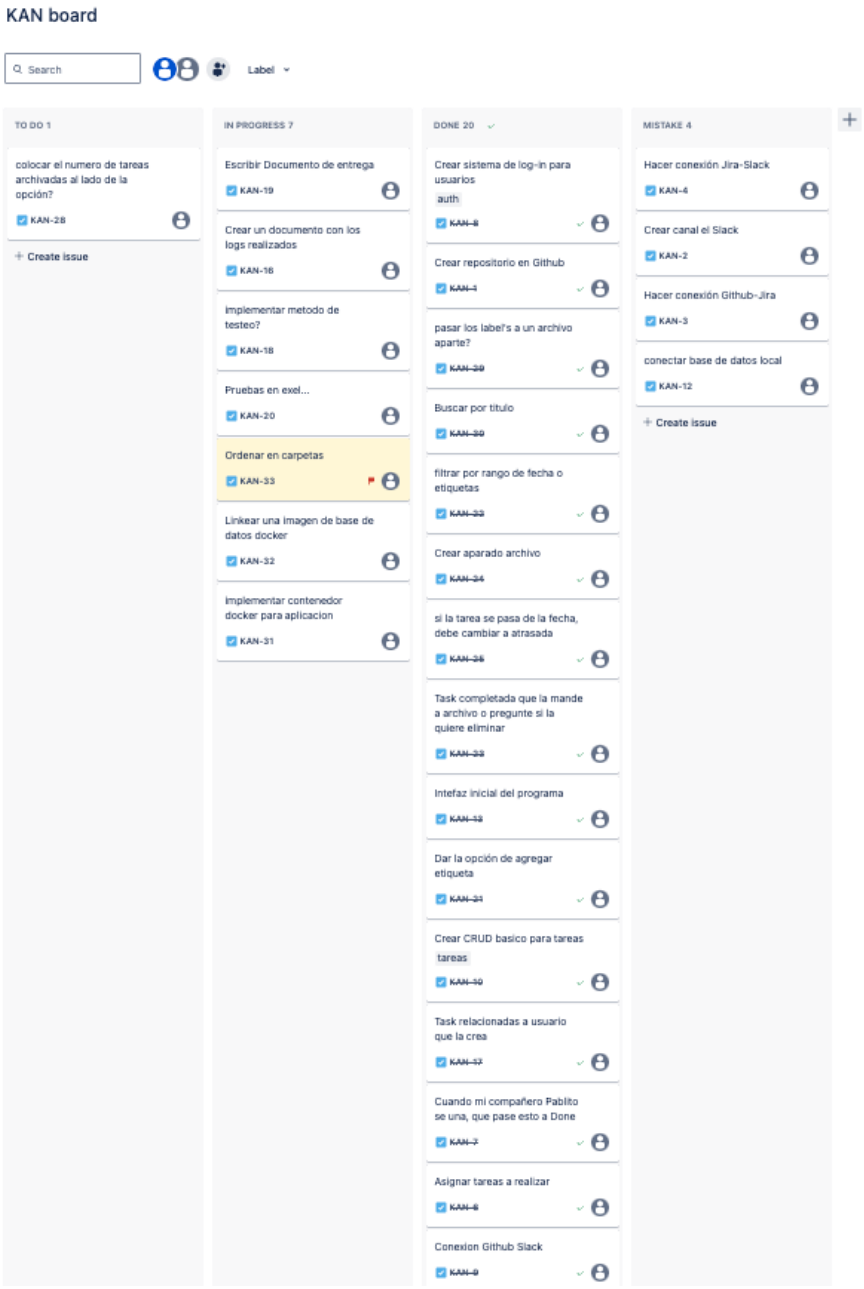


Tabla de Jira con las tareas creadas por el equipo (no es la tabla final).



Problemas Encontrados y Cómo se Solucionaron

Desde el inicio, surgieron dudas acerca de si debíamos trabajar con una base de datos como MongoDB o utilizar archivos de datos como JSON, especialmente en lo que respecta a la validación. Aunque el cliente solicitó un programa sencillo, nos preguntamos qué tan sencillo debía ser. ¿Bastaba con que se ejecutara localmente, o era necesario que pudiera funcionar desde varios dispositivos diferentes? Finalmente, decidimos implementarlo de manera local, y si el tiempo lo permitía, lo adaptaríamos para trabajar con una base de datos. Esta decisión se tomó para preparar el programa si el cliente quisiera usarlo desde diferentes dispositivos y mantener los datos actualizados.

Durante la verificación del programa, encontramos problemas en algunas funcionalidades que presentaban errores o bugs. Sin embargo, estos problemas fueron solucionados gracias a las pruebas fueron realizados por el Equipo (programadores y testers), quienes identificaron y corrigieron los fallos.

Finalmente, tuvimos dificultades con el manejo del contenedor en Docker. Este problema se presentó principalmente debido a la falta de experiencia en la dockerización de programas en Python. Un desafío específico fue la creación del archivo `requirements.txt`, cuyo funcionamiento no entendíamos del todo al principio. Sin embargo, logramos ejecutar un entorno virtual en python, lo que nos facilitó mucho las cosas para poder ejecutar el programa y crear la imagen en Docker.