En este documento, se analizan las decisiones de diseño que fueron realizadas para el proyecto antes de nuestras modificaciones, y a qué QAs (Quality Attributes) afectan, y cómo los afectan. También se realiza una explicación de los cambios realizados por nosotros, qué mejoras hicimos, qué QAs se vieron mejorados por nuestros cambios, y cómo.

Incorporación de herramientas de Log y Monitoreo

Con el añadido de herramientas de monitoreo como cAdvisor, Graphite y Grafana, incrementa la **observabilidad** del sistema, ya que es más fácil visualizar cómo se comporta, y su capacidad de respuesta ante fallos.

Uso de Docker y Nginx

Al usar Docker y el load balancer Nginx, estamos teniendo buena **disponibilidad** ya que se permite manejar multiples instancias de los servicios. Cuando hay alta carga o fallos en una instancia, Nginx redirige el tráfico a otras disponibles, y Docker facilita la recuperación y escalamiento de contenedores.

Validaciones de campos en peticiones

El sistema no tiene validaciones, lo que afecta a la **seguridad** del sistema pudiendo enviarse datos erróneos, realizar operaciones inválidas o enviar muchas peticiones erróneas que tiren abajo el sistema. Potencialmente también se podría ver afectada la **disponibilidad** ya que si no hay control de los datos que se envían en las requests, y se envían muchas para atacar al sistema, el sistema caerá debido a un ataque DoS (Denial-of-Service)

Se agregaron **validaciones** a los campos enviados a los endpoints de PUT Rate, POST Exchange y PUT Balance. Se validó que se enviaran al request los campos obligatorios, que tuvieran los valores, tipos y/o formatos correctos. Además, se muestra el mensaje de error correspondiente que señala el campo que está incorrecto al intentar enviar el request. Los mensajes de éxito o de error en la request también se mejoraron y se hicieron mas descriptivos.
Todos estos aspectos mejoran la **fiabilidad** del sistema, haciendo que sea mas sólido al tener un sistema de validación de peticiones y no permitiendo cualquier petición. Sobre todo, mejora la **seguridad**, ya que evita que se pasen datos incorrectos o erróneos, y también permite defenderse de ataques maliciosos de envío de muchas peticiones rechazándolas si estas no tienen los valores correctos en los campos.

Modificación de endpoint PUT de Cuentas

En el sistema se creó el endpoint **/cuentas/<id>/balance**. De base, este endpoint puede **romper las reglas de negocio**, ya que arbitrariamente se cambia el balance de una cuenta,

sin ninguna transacción en el medio ni dinero que sale de algún lado y va hacia otro. Y eventualmente puede romper el sistema.
Este endpoint afecta negativamente a algunos QAs, como por ejemplo la **seguridad**, porque este endpoint tiene más permisos de los que deberia tener (Afectar el balance de una cuenta), y si se tuviera control de este endpoint, se podrían alterar arbitrariamente los balances de todas las cuentas. Por la misma linea, al tener más permisos y responsabilidades de los que debería, y poder alterar fácilmente el sistema, es más difícil mantener y modificarlo ante futuros cambios o agregación de nuevas funcionalidades, y puede afectar a aspectos como la **mantenibilidad**, la **confiabilidad**, la **modificabilidad**, la **testeabilidad**, etc.

Por ello, se reemplazó el endpoint por el endpoint /**transfer**, el cual se encarga de realizar transferencias, sacando dinero de una cuenta e ingresandolo a otro, manteniendo la consistencia del sistema. Además, este endpoint tiene nuevas validaciones que el otro endpoint no tenía, como que se envíen los campos obligatorios, que las cuentas existan, que la cuenta que envía dinero tenga suficientes fondos, etc.
Este nuevo endpoint mejora la **seguridad** sacandole responsabilidades al endpoint anterior, por lo que el sistema no puede romperse y ya no se puede manipular el endpoint para modificar balances de forma arbitraria. También se mejoran los aspectos de **mantenibilidad**, **confiabilidad**, **modificabilidad**, entre otros, al tener un endpoint más consistente que permita agregar más funciones a futuro o modificar existentes.

Mejoras en búsquedas

Al tener que realizar búsquedas de cuentas por su id (Por ejemplo en el endpoint POST Transfer), se hacía la búsqueda en un arreglo, lo que hacía que su complejidad sea O(n). Ahora, al hacer la búsqueda, se hace en el **arreglo de cuentas convertido en una estructura Map**, y entonces la complejidad se convierte en O(1).
Esto ayuda a la **performance** del sistema. Si bien no se verá una mejora notable en performance en pruebas pequeñas, sí se notará la diferencia al tener muchas cuentas, y al realizar muchas operaciones.

Guardado en Archivos JSON

Toda la información de cuentas, log y rates se guarda en archivos JSON. Para el funcionamiento del sistema, esto tiene varios inconvenientes.
Primero, la **performance** es pobre, ya que los accesos a disco son más costosos que por ejemplo los accesos a memoria o a una BD, y además en el medio se lee, escribe, parsea, modifica y se realizan otras operaciones sobre los archivos. Además, las búsquedas son lineales ya que no hay una estructura de orden definida por el archivo, sino por el usuario. En particular, la **latencia** es pobre cuando el sistema tiene alta carga.
Segundo, la **escalabilidad** es baja, ya que no se pueden manejar muchas transacciones, porque múltiples datos pueden querer leer o escribir sobre un mismo recurso y que se bloqueen los archivos, y si alguno es modificado, no serán leídos los valores correctos en varias transacciones.

Tercero, es **vulnerable** por lo que no tiene buena **seguridad**, ya que la información está expuesta en archivos planos sin codificación en el medio ni alguna barrera de seguridad.
Cuarto, es poco **mantenible**, ya que se deben tocar los JSON manualmente en caso de algún agregado o modificación en toda la estructura, además que se evitan problemas de que se quiera acceder a un mismo recurso simultáneamente.
Por último, la **fiabilidad** no es buena, ya que las transacciones pueden no realizarse correctamente, y el sistema no es confiable.

Una solución a estos problemas, es guardar los datos en una **base de datos**, relacional o no relacional según la necesidad. En nuestro caso, se puede usar **Redis**.
Esto mejoraría enormemente varios aspectos. La **performance** mejora porque el acceso a datos ahora es en memoria, las búsquedas son mejores, y la **latencia** se reduce. La **escalabilidad** aumenta porque los datos se manejan desde un solo lado (Base de datos) y no un solo archivo, las transacciones se realizan sin conflicto y solo ocurrirían errores si pasa algo con la base de datos. La **seguridad** aumenta porque los datos son manejados dentro de la base de datos, la cual puede configurarse con contraseñas, claves y demás para que nadie la acceda. La **mantenibilidad** aumenta a largo plazo porque solo se deben añadir nuevos datos a la base de datos. La **fiabilidad** aumenta ya que las transacciones no se chocarán entre sí, ni darán errores constantemente.

Añadir tiers (Niveles)

El enunciado pedía que se añadan tiers de usuarios que haya un nivel de usuarios, y que los usuarios con mejor nivel tengan mejor tasa a la hora de hacer el intercambio de monedas. Para eso, se añadió un archivo **tiers.json** (Con una serie de tiers que tienen **nombre** y **spread**) en la carpeta **/app/state** y se implementó el método **GET Tiers** que devuelve las tiers del archivo json. Además, se añadió el campo **tiers** dentro de cada elemento de **accounts.json**, de manera que cada usuario tenía asignada una tier.

Añadido de manejo de seguridad con vaultSec

El enunciado pedía implementar un servicio de seguridad vaultSec, de tal forma que cuando llega al nginx, el request está autenticado y autorizado. Con el añadido de vaulSec, el sistema es más robusto y tiene mayor **seguridad**, ya que las peticiones no pueden ser enviadas por cualquiera, previene ataques DoS, que puedan realizarse operaciones que rompan el sistema por usuarios externos, y demás.

Añadido de métricas de volumen operado y neto de transacciones

El enunciado también pedía añadir métricas de volumen operado y neto de transacciones. Con este añadido, mejora la **observabilidad** del sistema, ya que permite visualizar más métricas que pueden ser relevantes para el análisis y comportamiento del sistema.