

Carga Plus

Mitre 870
Ciudad de Mendoza, Mendoza 5500

Bill-e

Objetivo

El objetivo del proyecto es desarrollar una billetera la cual se podrán realizar transacciones monetarias virtuales de manera segura y efectiva, tanto orientado a la red de PDV de CargaPlus como de usuarios finales.

Alcance

1. **Cuenta Corriente:** Cada usuario va a contar con una cuenta corriente virtual para llevar el control simple y detallado de movimientos como depósitos, extracciones, transferencias y consumos en diferentes productos y servicios.
2. **Acceso a la Información:** Podrán acceder y ver sus datos cuando le sea necesario, como información personal o sus movimientos.
3. **Interfaz:** Proponemos una interfaz intuitiva y fácil de manejar, con datos relevantes a la vista y atajos más usados para que el usuario se sienta lo más cómodo posible.
4. **Identificación:** Incluye métodos biométricos de identificación a la hora de iniciar la aplicación, esto garantiza que solo la persona que coincida con la información biométrica pueda ingresar a su cuenta.
5. **Publicidad:** Banners con diferentes productos que ofrece la empresa.

Límite

Será capaz de realizar transacciones proporcionadas por la api de BIN, como así con otros usuarios de la billetera.

También se podrán consumir servicios asociados a la empresa, ya sean préstamos o cargas virtuales.

Especificaciones técnicas

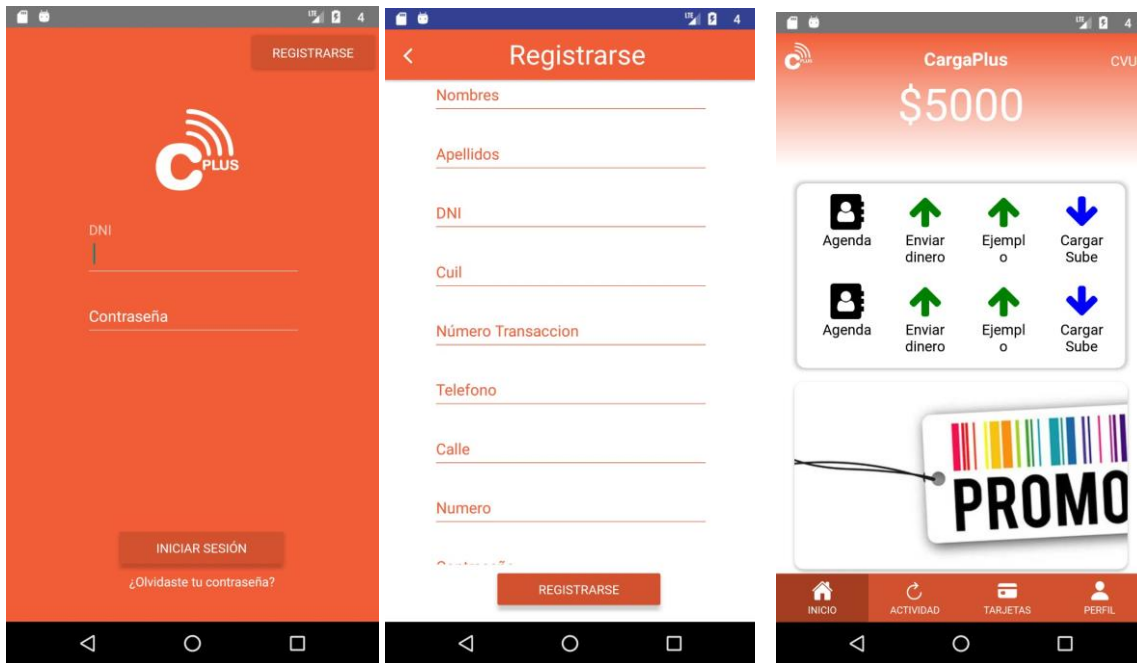
Abordaje:

Para desplegar la aplicación, vamos a realizar una arquitectura orientada a microservicios, donde cada componente es independiente uno de otro. Los beneficios de este modelo son: agilidad, implementación sencilla, libertad tecnológica, escalabilidad, resiliencia, código reutilizable.

La implementación requiere de servidores donde correrán los diferentes servicios, y un “maestro” que funcione como orquestador de estos (servidor Linux). Además cuenta con un servidor aparte (Windows) en el cual se encontrará la base de datos, allí se podrá ver toda la información de la aplicación.

Los microservicios a desarrollar serán los expuestos en los casos de uso, que en resumidas cuentas son las diferentes funcionalidades de la aplicación.

Ejemplos de pantallas:



Tecnologías usadas:

Micro-Servicios


La arquitectura de microservicios es un método de desarrollo de aplicaciones software que funciona como un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, proporcionando una funcionalidad de negocio completa. En ella, cada microservicio es un código que puede estar en un lenguaje de programación diferente, y que desempeña una función específica. Los microservicios se comunican entre sí a través de APIs, y cuentan con sistemas de almacenamiento propios, lo que evita la sobrecarga y caída de la aplicación.



```

317 // Node stopping error is caught below in the select.
318 if err := t.rpcContext.Stopper.RunTask(
319     stream.Context(), "storage.RaftTransport: processing
320     func(ctx context.Context) {
321         t.rpcContext.Stopper.RunWorker(ctx, func(ctx context
322         errCh <- func() error {
323             var stats *raftTransportStats
324             stream := &LockedRaftMessageResponseStream(Multi
325             for {
326                 batch, err := stream.Recv()
327                 if err != nil {
328                     return err
329                 }
330                 if len(batch.Requests) == 0 {
331                     continue
332                 }
333                 if stats == nil {
334                     stats = t.getStats(batch.Requests[0].FromRe
335                 }
336                 for i := range batch.Requests {
337                     batch.Requests[i]

```

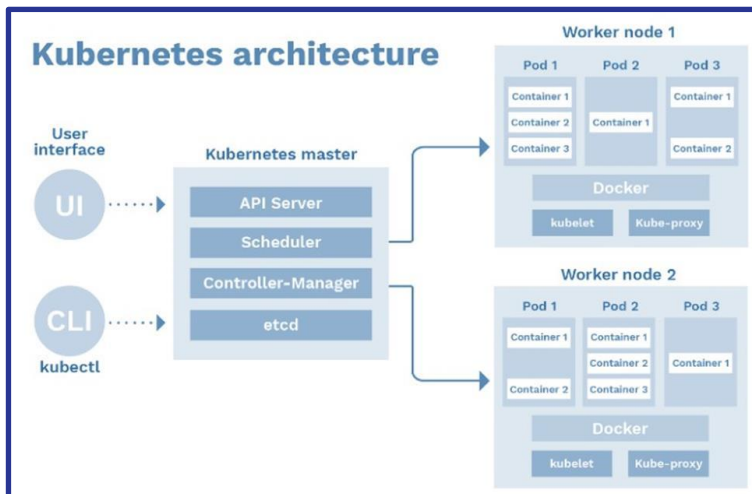
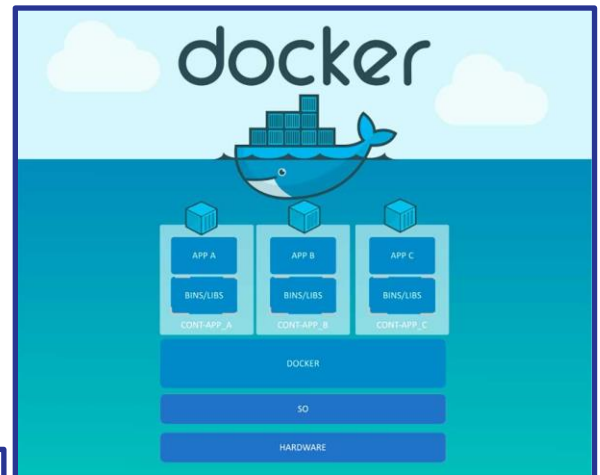


Go

Go es un lenguaje de programación concurrente y compilado inspirado en la sintaxis de C, que intenta ser dinámico como Python y con el rendimiento de C o C++. Ha sido desarrollado por Google y es de los más utilizados en desarrollo BackEnd. <https://golang.org/>

Docker

Docker automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. <https://www.docker.com/>



Kubernetes >> Docker Swarm

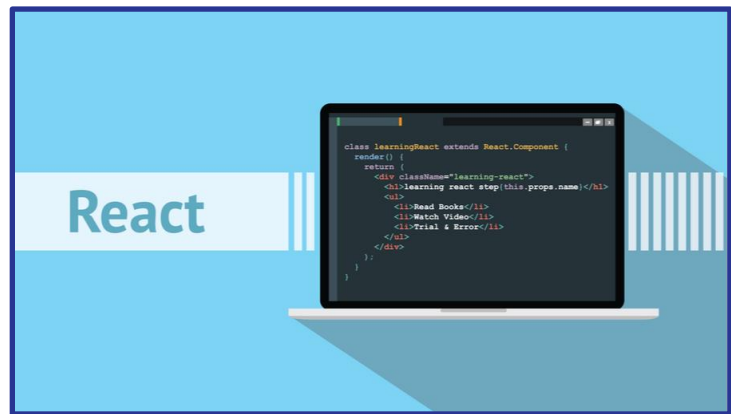
Kubernetes es un sistema de código libre para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores.

<https://kubernetes.io/>

React Native

Es un framework JavaScript para crear aplicaciones reales nativas para iOS y Android, basado en la librería de JavaScript React para la creación de componentes visuales.

<https://reactnative.dev/>



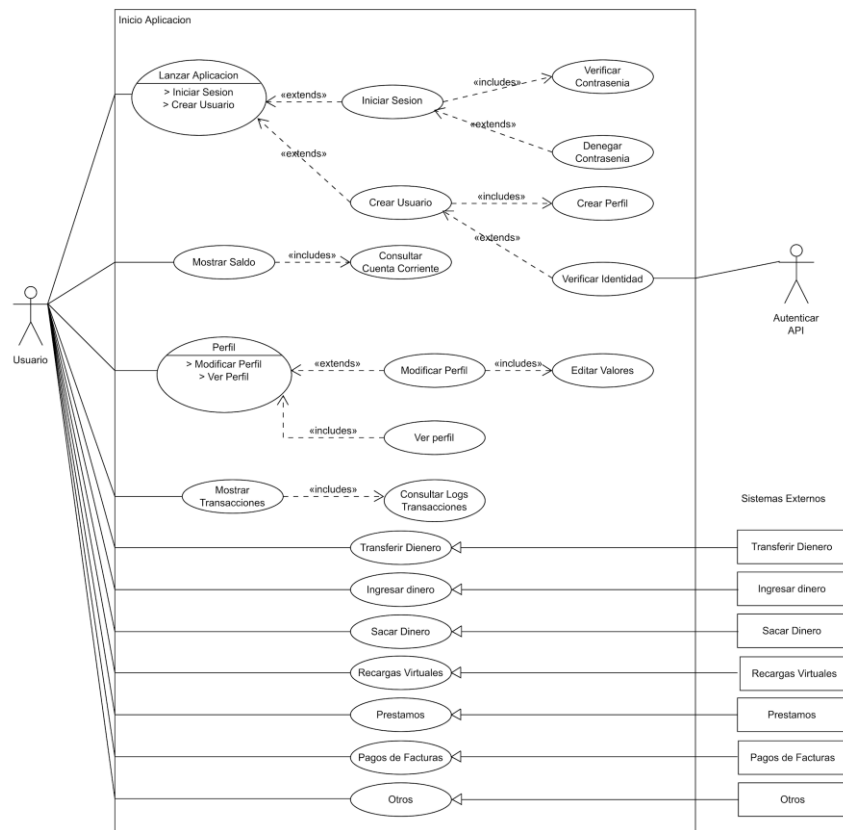
Diseño del sistema:

Casos de Uso:

Inicio de aplicación

Este caso refleja lo que sucedería al iniciar la aplicación, como paso principal es iniciar la sesión del usuario. Si no posee una cuenta, debe crearse una y crear una contraseña para esta, además se debe verificar su identidad a través de la API Autenticar; en caso de tener una cuenta solo deberá identificarse en la aplicación.

Una vez que ingrese, podrá ver el saldo que tiene en su cuenta, así como ver las transacciones que ha realizado. Luego puede navegar a través de la aplicación, donde puede modificar su perfil así como realizar distintas acciones en su cuenta, ya sea ingresar dinero a la misma, transferir, retirar dinero.

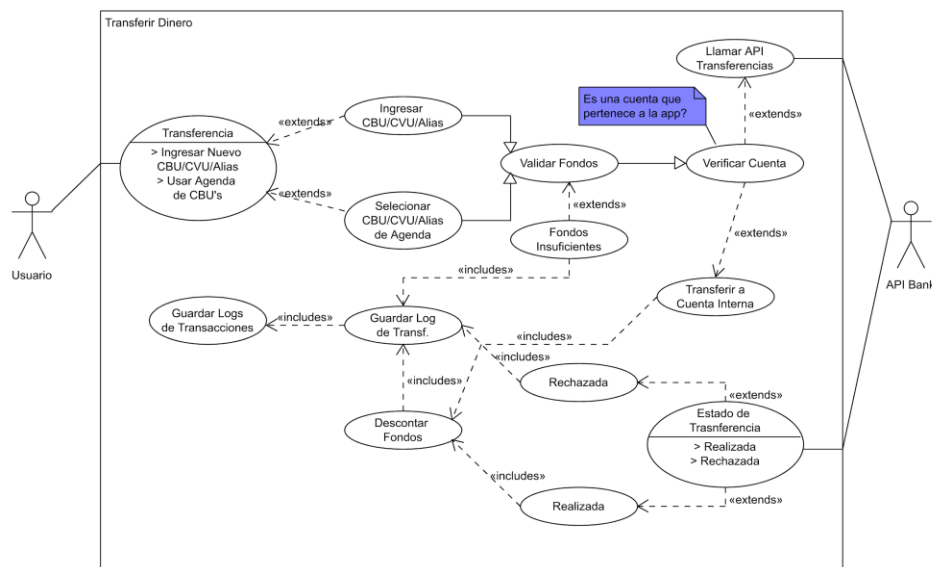


Transferir Dinero

En este caso tenemos la transferencia desde la cuenta del usuario a cualquier otra, que bien puede ser a otra cuenta dentro de la aplicación o a una cuenta externa al sistema.

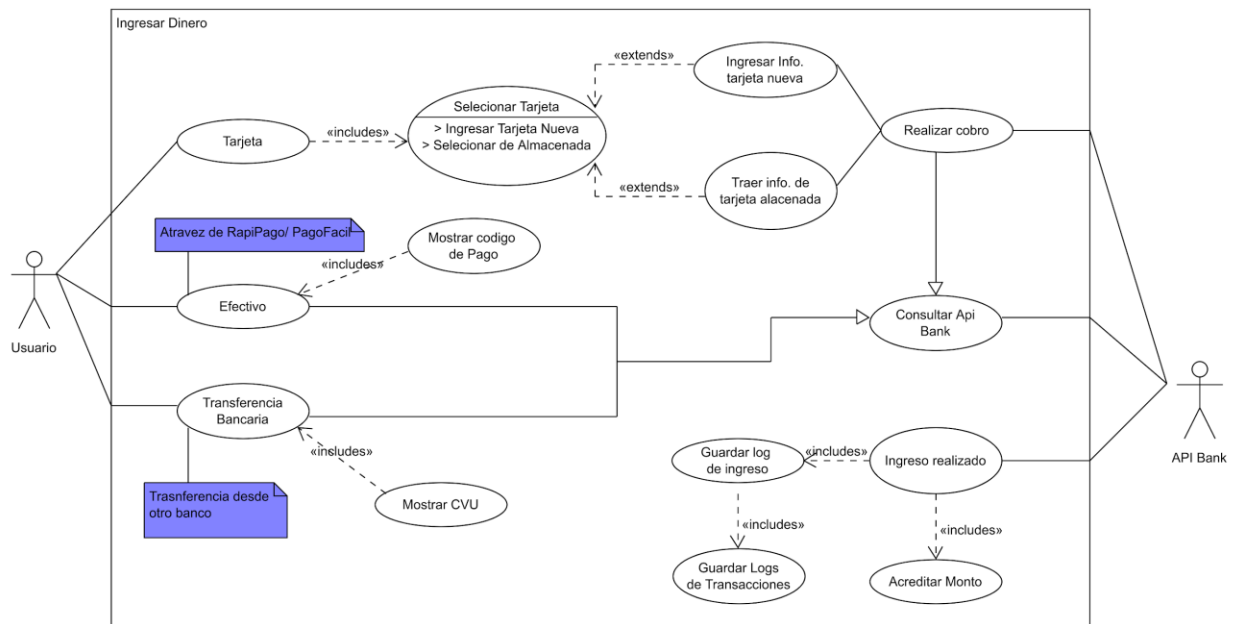
Para ambos casos hacemos uso de un CBU/CVU/Alias, que bien puede estar almacenado en una agenda del perfil. Luego el usuario ingresa la cantidad a transferir, que puede suceder que no tenga fondos suficientes, donde se le notifica al usuario y se genera un registro de la transacción denegada. En caso de que los fondos sean suficientes verificamos si la cuenta beneficiaria pertenece a una cuenta interna o externa.

Si la cuenta es interna se 'transfiere' de una cuenta a otra, acreditando el monto en la cuenta beneficiaria y descontando el mismo de la cuenta que envía, y se realiza un registro de la transferencia. En caso de ser externa hacemos un llamado a la API de Bind, donde con los datos del usuario y la cuenta beneficiaria realizamos una transferencia bancaria en nombre del usuario, como PSP (Proveedores de Servicios de Pago). Luego puede suceder que la transferencia sea rechazada, en tal caso se genera un registro y se le avisa al usuario, si es realizada se descuentan los fondos a la cuenta y se genera un registro.

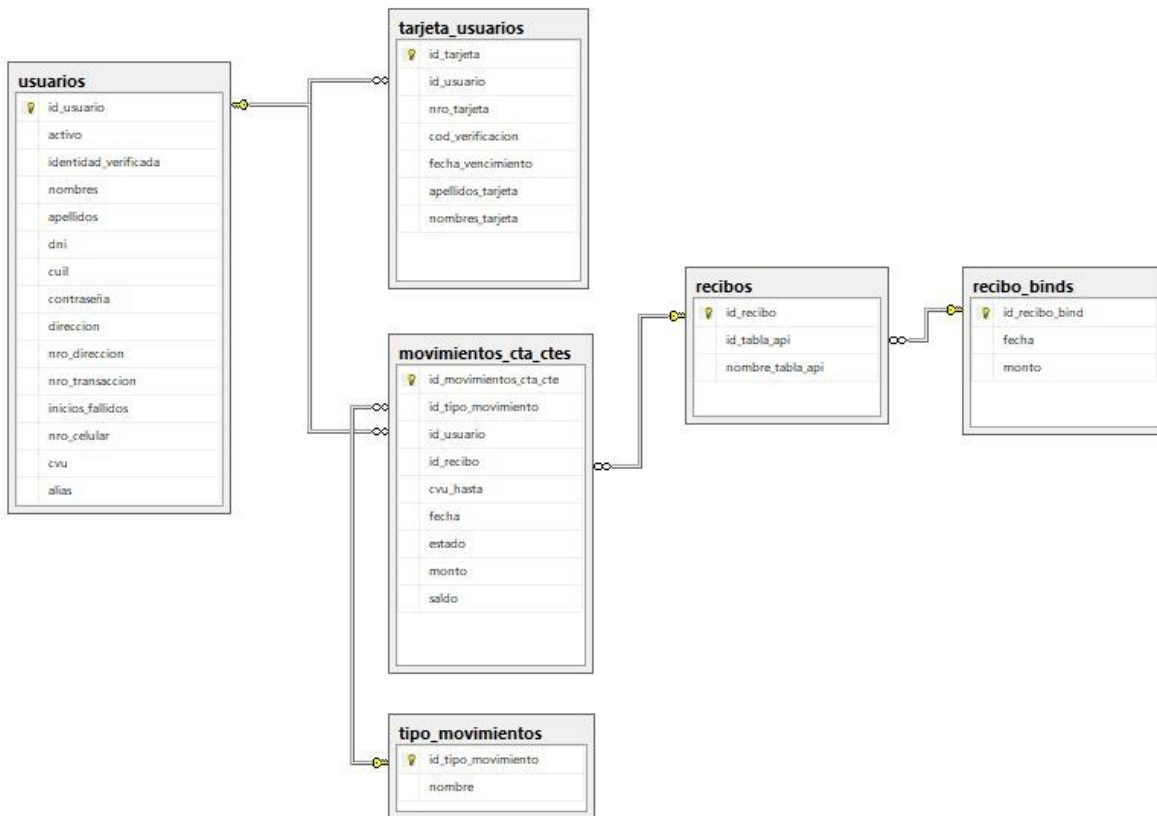


Ingresar Dinero

A la hora de ingresar dinero podrá elegir entre una tarjeta de crédito/débito, efectivo o por una transferencia bancaria; en el caso que elija como método de ingreso una tarjeta, podrá agregar una tarjeta, ingresando su información, que podrá ser usada cuando lo necesite, en caso de que ya tenga agregadas, podrá elegir la que desee. Cuando el usuario desee ingresar a través de un pago en efectivo, se le brindará un código que deberá presentar en un RapiPago/PagoFácil. Si se le realizará una transferencia a la cuenta tiene que acceder a su CVU/Alias para poder compartirlo en cualquier momento y recibir el dinero. Una vez se haya realizado la transferencia podrá ver su dinero y los detalles de la transacción.



Diseño de base de datos:



● Tabla Usuario

- idUsuario: Número de cliente del usuario.
- activo: Bit que identifica si la cuenta está activa.
- identidadVerificada: Bit que identifica si los datos son reales.
- nombres: Nombres del cliente.
- apellidos: Apellidos del cliente.
- dni: DNI del cliente.
- cuil: Cuil del cliente.
- contraseña: Contraseña codificada de la cuenta.
- direccion: Calle de la localidad del cliente.
- nroDireccion: Numero de la localidad del cliente.
- nroTransaccion: Número de transacción que del cliente.
- iniciosFallidos: Veces que se intentó de manera errónea entrar en la cuenta.
- nroCelular: Número de contacto del cliente.

Se guardarán los datos del perfil del cliente.

- **Tabla CuentaCorriente**

- idCuentaCorriente: Número de identificación de la cuenta.
- idUsuario: Número de cliente del usuario. Clave foránea.
- cvu: CVU del cliente.
- alias: Alias del cliente.

Se guardarán los datos relacionados a la cuenta virtual del cliente.

- **Tabla MovimientosCtaCte**

- idMovimientosCtaCte: Número de identificación del movimiento.
- idTipoMovimiento: Número de identificación del tipo de movimiento. Clave foránea.
- idCuentaCorriente: Número de identificación de la cuenta. Clave foránea.
- idRecibo: Número de identificación del recibo. Clave foránea.
- cvuDesde: CVU del remitente de la transferencia.
- cvuHasta: CVU del destinatario de la transferencia.
- fecha: Fecha y hora en que se realizó el movimiento.
- estado: Estado en tiempo real del movimiento.
- monto: Cantidad de dinero transferida.
- saldo: Saldo final de la cuenta.

Se guardarán los movimientos con los datos principales.

- **Tabla TipoMovimiento**

- idTipoMovimiento: Número de identificación del tipo de movimiento.
- nombre: Descripción del tipo de movimiento.

Se guardarán los tipos de movimientos posibles.

- **Tabla Recibo**

- idRecibo: Número de identificación del recibo.
- idTablaApi: Número de identificación en la tabla donde se encuentran los detalles.
- nombreTablaApi: Nombre de la tabla donde se encuentran los detalles.

Se guardarán los datos que relacionan las respuestas de las API's con el movimiento generado así se podrán ver más detalles del mismo.

- **Tablas ReciboBIND/ReciboVISA**

Estas tablas hacen referencia a la respuestas de las API's que se usarán.