**Retos principales**

**Problema con el reconocimiento de voz**:

Creemos que cuando se utiliza el reconocimiento de voz, principalmente cuando se tenga que reconocer cantidades de dineros, nombres de bancos y hasta números de cuenta o de factura puede haber equivocaciones, tanto por parte del usuario como del reconocimiento. A esto hay que agregarle si el lugar donde la persona esta haciendo sus transacciones tiene mucho ruido, si la conexión de internet del usuario no es optima, o temas con la pronunciación de alguno de los elementos. El sistema va a requerir mucho entrenamiento para funcionar al 100%.

**Seguridad y normas bancarias**:

Cumplir con ciertas normas, principalmente la PCI DSS (que es una certificación), esta aparte de tener que cumplir con una serie de pasos y funciones de infraestructura requiere que un auditor de la entidad audite la aplicación. Además que se debe de volver a evaluar anualmente. Esto presenta un reto en cuanto a cumplir con cada una de las solicitudes que esta presenta.

**Conexión con diferentes bancos**:

Cada banco tiene su aplicación para hacer pagos, entonces para agregar una se requiere de un tiempo, al empezar con 3 (Banco Popular, BAC y Sinpe) hay que configurar y programar cada una de las plataformas, además de tener que estar en constante mantenimiento de estas ante posibles cambios que tengan. Cada vez que haya que agregar una pasarela nueva habrá un reto de como se manejan los pagos, los pagos recurrentes e información de cada plataforma.

**Manejo de muchos usuarios a la vez**:

La aplicación va a tener picos altos de uso en ciertas fechas y bajos en otros, el manejo de esto debe de ser muy bien configurado y monitoreado en nuestro proveedor de infraestructura para que se pueda manejar correctamente, después de algunos meses puede haber ya información de como utilizarlo, pero siempre será un reto el que la aplicación tenga tiempos de respuesta cortos y eficientes.

**Funcionalidades que No Estarán Listas Desde el Inicio**

**Interacción por Voz – 60%**

En principio podríamos contar con comandos básicos, por ejemplo: “Pagar Luz, Cuenta Banco Popular, Colones”. Esto hará que el usuario pueda pagar su cuenta de luz (ya registrada en el sistema), con la cuenta del Banco Popular y la moneda de la misma (que también ya estaría registrada en el sistema). Pero para que pueda reconocer números grandes como el IBAN que contiene 16 caracteres y pedir que se haga una transacción con ese monto puede ser mas complicado y va a requerir cierto entrenamiento y validaciones manuales para evitar que se comentan errores.

**Seguridad y Cumplimiento Normativo – 65%**

Aunque se implementaran todas las necesidades de cifrado, autenticación y contraseñas, además de seguridad en los servidores y en la programación para hacer de la herramienta algo seguro, la certificación PCI DSS va a requerir de un tiempo adicional para ser obtenida.

**Ejecución de Pagos – 70%**

Aunque fue parte de lo conversado en las preguntas que se le hicieron al cliente, empezaremos con algunos bancos en específico: BAC y Banco Popular, así como integrar algún pago por medio de SINPE. Pero la integración debería hacerse con mas bancos o entidades que nos permitan hacer los bancos. Es por esta razón que lo ponemos como que no va a estar al 100% ya que quisiéramos agregar mas opciones disponibles.

**Stack Tecnológico:**

Trabajaremos en el ecosistema de AWS.

**Arquitectura General**

**Lenguajes y Frameworks:**

Front End: React Native 🡪 consistencia para plataformas IOS y Android / React 🡪 versión accesible desde navegador (con algunas funcionalidades no presentes)

Backend: NodeJS + NestJS 🡪 nos funciona tanto en microservicios como en funciones lambda que se puedan requerir en el desarrollo.

**Bases de Datos y Almacenamiento**

Bases de Datos SQL: PostgreSQL 🡪 utilizaremos la opción de Amazon Aurora, nos permite ser escalables y al ser relacional nos permite guardar información correcta que se relaciona entre usuarios, pagos, entidades bancarias y distintos elementos.

Bases de Datos NoSQL: DynamoDB 🡪 es la base de datos NoSQL del ecosistema de AWS, nos permite trabajar con logs, historiales e inclusive con eventos y funcionalidades en tiempo real.

Almacenamiento: Amazon S3 🡪 nos permite almacenar archivos de voz, logs, historiales, pdf.

**Seguridad**

Autenticación: Amazon Cognito 🡪 Nos permite confirgurar el 2FA y el OAuth que el sistema necesita de manera sencilla.

Almacenamiento de Credenciales: AWS Secrets Manager 🡪 nos permite almacenar credenciales en un ambiente muy seguro.

Protección: AWS WAF 🡪 es un firewall que nos va a permitir proteger la aplicación de ataques.

**Notificaciones**

Envio de mensajes SMS: Amazon SNS 🡪 envia mensajes SMS y notificiaciones PUSH.

**Inteligencia Artificial**

Conversión voz a texto: Amazon Transcribe 🡪 se especializa en transformar cualquier audio o video a texto, nos permitirá entender lo que el usuario desea hacer en la aplicación mas fácilmente.

Procesamiento de lenguaje natural: Amazon Lex 🡪 nos permitirá entender las ordenes que obtiene el Amazon Transcribe, en el lenguaje natural que el cliente las envie, y ejecutarlas correctamente.

Aprendizaje de la herramienta: Amazon Bedrock 🡪 el uso de uno de los LLM ligados a Bedrock (anthropic, Meta, Mistral, entre otros) nos permitirá ensenarle a la IA como comportarse en un futuro.

**DevOps y Monitoreo:**

CI/CD: AWS CodePipeline 🡪 nos permite automatizar todas las etapas del “lanzamiento” del software en cada uno de sus ambientes (dev, staging, production) / AWS Codebuild 🡪 compilamos el código que hemos creado eficientemente, también nos permite testear y empacar el código para su lanzamiento.

Infraestructura: Cloudformation 🡪 para que nos permita escalar la aplicación cada que se necesite / Terraform 🡪 nos permite tener nuestra infraestructura como código, por si debemos recrearla rápidamente en algún otro ambiente, zona o servidor de AWS. Permitiendo que el tiempo para levantar nuestra infraestructura sea menor a hacerlo servicio por servicio.

Monitoreo: Amazon Cloudwatch 🡪 nos permite auditar como esta funcionando la infraestructura, crear alertas y configuraciones adicionales para monitorear correctamente nuestra aplicación.

Hosting: AWS Lambda 🡪 eventos pequenos y en tiempo real (tiene un pequeño delay cuando no se utiliza mucho cada función) / AWS Fargate 🡪 para tareas largas y pesadas. Ambos servicios son serverless lo que nos permite costos mas baratos y que sean fácilmente escalables de acuerdo a las necesidades de la misma aplicación.

QA

Aparte de las mencionadas Cloudwatch que nos permite evaluar y monitorear nuestra aplicación y de Codebuild que nos permitiría crear pruebas unitarias dentro del sistema se podrían utilizar las siguientes herramientas de AWS:

UI/UX: AWS Device Farm 🡪 nos permite evaluar el diseño de la aplicación tanto para móviles como para web. Fallos en el UI y en el UX, probar el rendimiento en distintos dispositivos nos permite esta aplicación.

Determina cargas altas o caídas del sistema: AWS XRay 🡪 nos permite medir el rendimiento de la aplicación cuando se conecta a un banco, o en algún momento especifico dentro del flujo de la aplicación.