# CUENTAS CLARAS

## PORTAFOLIO DE TÍTULO

Duoc UC

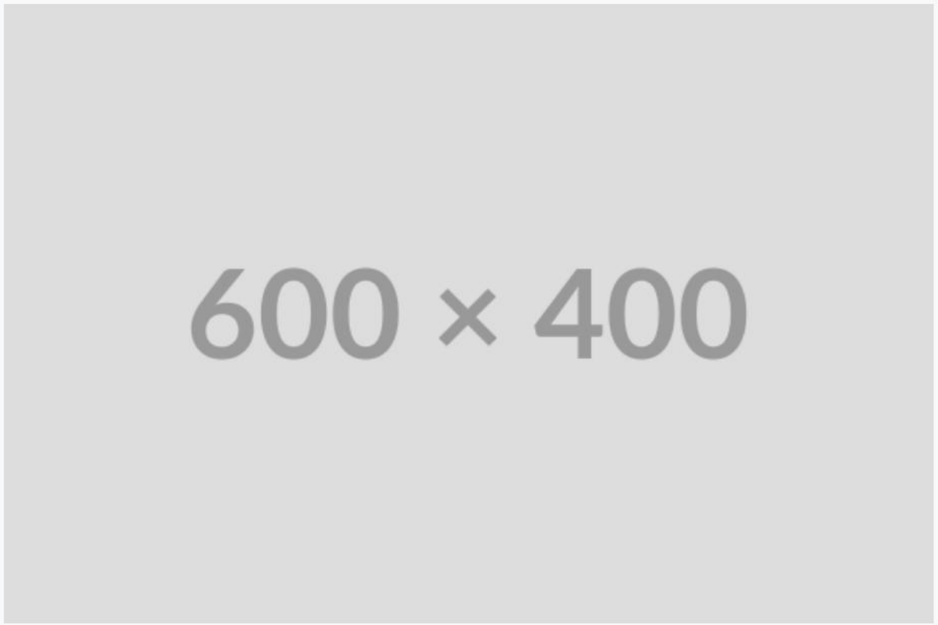
# Introducción

Visión general del proyecto, el equipo y sus objetivos.

# ¿Qué es Cuentas Claras?

Cuentas Claras es una aplicación web integral diseñada para resolver la gestión de cobros, cargos y conciliaciones en comunidades y condominios.

En esta presentación, exploraremos sus componentes técnicos (frontend, backend, seguridad), la integración de autenticación segura, el uso de bases de datos relacionales y la creación de entornos reproducibles con Docker.



# El Equipo

## Matías Román

**QA & Coordinador:** Pruebas funcionales y automatizadas, control de calidad y coordinación.

## Patricio Quintanilla

**Desarrollador Frontend:** Desarrollo de interfaz web, integración de APIs y autenticación.

## Frank Vogt

**Desarrollador Backend:** Implementación de API REST, modelado de base de datos y seguridad.

# El Contexto de Negocio

Definiendo el problema, nuestra solución y el modelo de negocio.

# El Problema

## Gestión Manual Ineficiente

- Dificultad en la gestión manual de cobros y conciliaciones.
- Errores frecuentes en registros contables.
- Falta de trazabilidad en movimientos financieros.

## Sistemas Desconectados

- Uso de múltiples herramientas no integradas (Excel, Email, Banco).
- Aumento del tiempo de gestión y riesgo de inconsistencias.
- Falta de un sistema centralizado, seguro y transparente.

# La Solución Propuesta

## Plataforma Centralizada

- Plataforma web segura (SaaS) y centralizada para la gestión.
- Automatización del registro de cargos, pagos y conciliaciones.
- Integración con pasarelas de pago (Webpay/Khipu).

## Seguridad y Transparencia

- Autenticación segura (JWT, bcrypt, 2FA).
- Separación de datos por comunidad (Arquitectura Multi-Tenant).
- Interfaz intuitiva para crear y editar cargos fácilmente.

# Objetivos del Proyecto

## Objetivo General

Desarrollar una plataforma web segura y eficiente para la gestión de comunidades, que permita emitir cargos, registrar pagos y realizar conciliaciones automáticas, optimizando los procesos y garantizando transparencia.

## Objetivo Específico

Implementar un sistema de autenticación y control de acceso mediante JWT, bcrypt y 2FA, con el fin de asegurar la protección de la información y mantener la correcta separación de datos entre comunidades.

# Modelo de Negocio: SaaS

La arquitectura de la base de datos permite una estrategia de "Software as a Service" (SaaS) flexible y escalable.

- **Modelo de Suscripción:** Un cobro mensual (MRR - Monthly Recurring Revenue) basado en la cantidad de unidades (viviendas) registradas por la comunidad.
- **Modelo Freemium:** Ofrecer un plan básico gratuito (ej. hasta 20 unidades) y cobrar por módulos avanzados (conciliación bancaria, pago en línea).
- **Licenciamiento B2B:** Venta de licencias de la plataforma a grandes empresas administradoras de propiedades para que gestionen toda su cartera.
- **Modelo Transaccional:** Cobrar una pequeña comisión (ej. 1-2%) por cada pago procesado exitosamente a través de las pasarelas integradas.

# Modelo de Negocio: Métricas Clave (KPIs)

El éxito del modelo SaaS se mediría con indicadores clave de rendimiento:

- **MRR (Ingreso Mensual Recurrente):** La métrica más importante. El ingreso predecible generado por las suscripciones.
- **CAC (Costo de Adquisición de Cliente):** Cuánto cuesta (marketing, ventas) adquirir una nueva comunidad (cliente).
- **LTV (Valor de Vida del Cliente):** El ingreso total esperado que un cliente generará durante su tiempo en la plataforma. (Idealmente LTV > 3 * CAC).
- **Churn Rate (Tasa de Abandono):** El porcentaje de clientes (comunidades) que cancelan su suscripción en un período. Mantenerlo bajo es vital.

# Proceso y Metodología

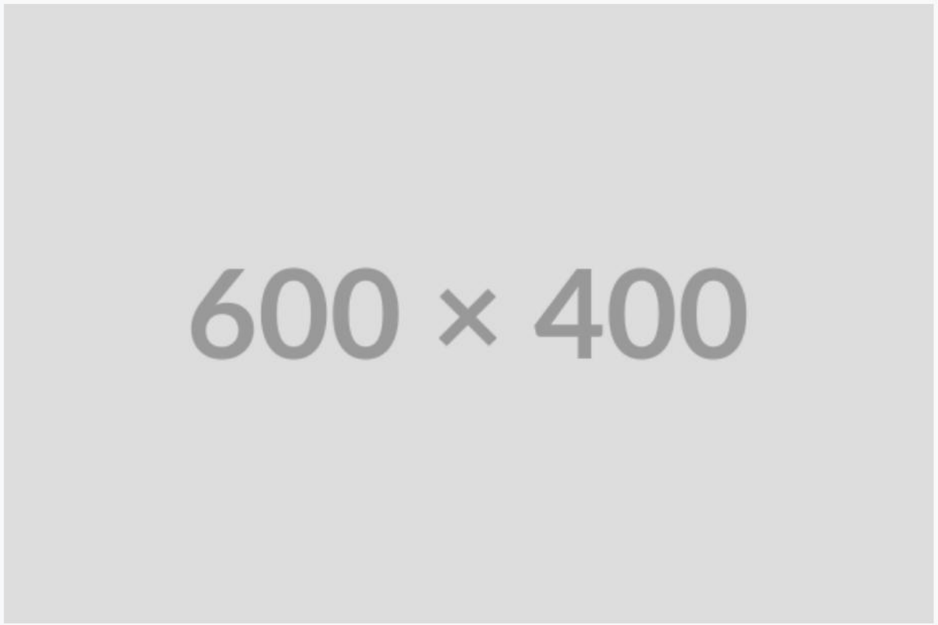
Cómo construimos el producto de forma ágil y controlada.

# Metodología Ágil: Scrum

Utilizamos Scrum para gestionar el proyecto, dividiendo el trabajo en Sprints con tareas y objetivos claros.

Esta forma de trabajo nos permitió:

- Organizar mejor al equipo y asignar responsabilidades.
- Adaptarnos rápidamente a los cambios y nuevos requerimientos.
- Avanzar de manera constante e incremental.
- Realizar revisiones y ajustes en cada etapa.



# Herramientas de Gestión y Calidad


Jira: Gestión de tareas, historias de usuario y seguimiento de Sprints.


GitHub: Control de versiones (Git), Pull Requests y revisión de código.


Postman: Pruebas funcionales y automatizadas de la API REST.

# Arquitectura del Sistema

El esqueleto técnico de Cuentas Claras.

# Arquitectura de Software (Alto Nivel)

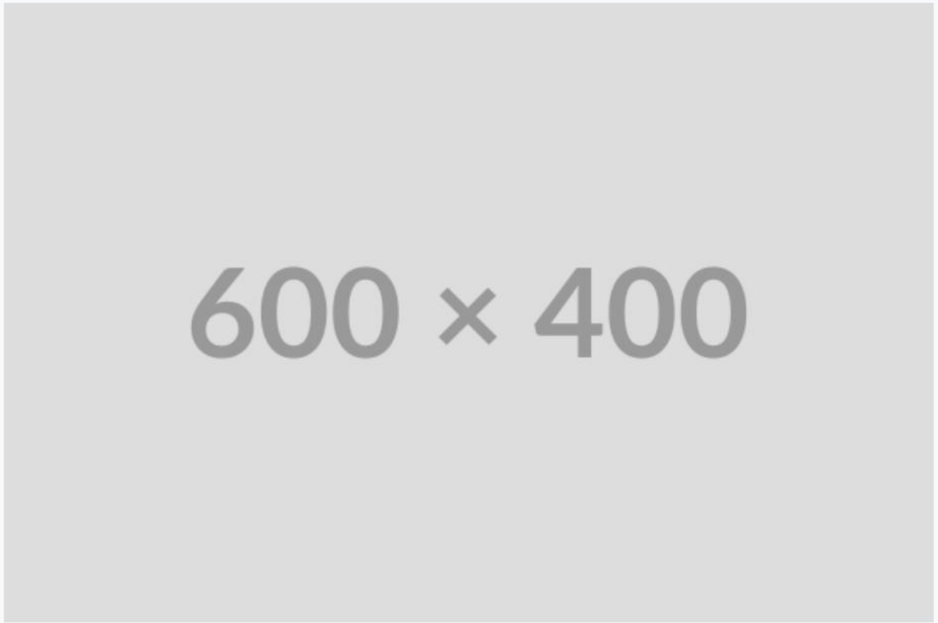
# Arquitectura de Despliegue (Cloud)

# Diseño Clave: Arquitectura Multi-Tenant

El desafío principal de un SaaS es manejar múltiples clientes (comunidades) en una sola instancia de la aplicación, asegurando que los datos de un cliente no sean visibles para otro.

**Nuestra Solución:** Aislamiento a Nivel de Base de Datos.

- Casi todas las tablas clave (unidades, pagos, gastos) tienen una columna **`comunidad_id`**.
- Cada consulta (Query) de la API filtra obligatoriamente por el `comunidad_id` del usuario autenticado.
- Esto garantiza que un administrador de la "Comunidad A" nunca pueda ver datos de la "Comunidad B".



# Modelo de Datos (Diagrama General)

# Módulos Clave del Modelo de Datos

## Identidad y Acceso

Controla quién accede a qué. (Tablas: `usuario`, `rol`, `usuario_comunidad_rol`).

## Ciclo de Gastos

Genera la deuda mensual. (Tablas: `emision_gasto_comun`, `cuenta_cobro_unidad`).

## Pagos y Aplicaciones

Registra ingresos y los aplica a deudas. (Tablas: `pago`, `pago_aplicacion`).

# Stack Tecnológico

Las herramientas que usamos para construir Cuentas Claras.

# Tecnologías Frontend


React: Biblioteca para construir interfaces de usuario.


Next.js: Framework para React (SSR, SSG).


TypeScript: JavaScript con tipado estático.

# Tecnologías Backend


Node.js: Entorno de ejecución de JavaScript en el servidor.


Express: Framework minimalista para APIs REST.


Sequelize: ORM para la comunicación con MySQL.

# Tecnologías de Datos y Caché


MySQL: Sistema de gestión de bases de datos relacional.


Redis: Almacén en memoria para caché (sesiones, etc.).


phpMyAdmin: Herramienta de administración de MySQL.

# Tecnologías DevOps (Infraestructura)


Docker: Creación de entornos reproducibles (contenedores).


Nginx: Proxy reverso, balanceo de carga y servidor de estáticos.


GitHub Actions: CI/CD para automatizar pruebas y despliegues.

# Tecnologías de Calidad y Documentación


Postman: Pruebas funcionales y de integración de la API.


Swagger (OpenAPI): Documentación interactiva de la API.


ESLint: Mantenimiento de la calidad y consistencia del código.

# Enfoque en Seguridad

Protegiendo la información sensible de las comunidades.

# Pilares de Seguridad

- **Autenticación (JWT):** Uso de JSON Web Tokens para gestionar sesiones de API seguras y sin estado, firmadas digitalmente.
- **Encriptación (bcrypt):** Almacenamiento de contraseñas de usuario usando hashing robusto e irreversible (bcrypt) con salt.
- **2FA (TOTP):** Implementación de "Time-based One-Time Password" (ej. Google Authenticator) para una capa adicional de seguridad en el login.
- **Aislamiento (Multi-Tenant):** Diseño de base de datos que aísla lógicamente los datos de una `comunidad` de otra a nivel de consulta.
- **Validación de API:** Uso de Swagger y validación de esquemas para prevenir inyecciones (ej. SQLi, XSS) y asegurar que solo lleguen datos limpios.

# Conclusión

Demostración y próximos pasos.

# Roadmap y Futuras Mejoras

- **Aplicación Móvil:** Desarrollo de una app (React Native) para residentes, permitiéndoles ver sus cargos y pagar directamente.
- **Conciliación Bancaria con IA:** Usar IA para leer cartolas bancarias y sugerir automáticamente la aplicación de pagos no identificados.
- **Módulo de Comunicación:** Añadir un "muro" o sistema de tickets para mejorar la comunicación entre administración y residentes.
- **Dashboards Avanzados:** Más gráficos y reportes de gestión financiera, morosidad y tendencias de gastos para administradores.

# Preguntas?

Muchas Gracias Por su atención.