1. Introducción

"Ticket Splitter" es una aplicación para la gestión de proyectos en los que los usuarios pueden registrar tickets y realizar pagos compartidos. El sistema calcula cuánto debe cada miembro a los demás según las contribuciones y gastos registrados.
Objetivo: Simplificar y automatizar la gestión financiera colaborativa en proyectos grupales, ofreciendo recordatorios automáticos, cálculos precisos y funcionalidades de colaboración.

---

2. Arquitectura

Backend:

- **Lenguaje:** TypeScript.
- **Framework:** Node.js con Express.
- **Base de datos:** Google Cloud SQL (MySQL), gestionada mediante Prisma.
- **Otros:**
    - Sistema de autenticación seguro para usuarios.
    - Envío de correos transaccionales y recordatorios con SendGrid.
    - Gestión y almacenamiento de imágenes con Cloudinary.

Frontend:

- **Librería:** React.
- **Diseño:** Tailwind CSS.

---

3. Instalación y Configuración

Requisitos Previos:

- Node.js
- MySQL
- Prisma CLI (`npm install prisma -g`)

Pasos para Clonar y Configurar:

1. Clona el repositorio:

   ```
   git clone https://github.com/lisandroaloo/ticket-spliter.git
   ```

```
cd ticket-spliter
```

2. Instala las dependencias:

    - **Backend:**

      ```
      npm install
      ```

    - **Frontend:**

      ```
      cd frontend
      npm install
      ```

3. Configura las variables de entorno:

   Crea un archivo `.env` con las siguientes claves:

   ```
   DATABASE_URL="mysql://<USER>:<PASSWORD>@<HOST>/<DATABASE>"
   SENDGRID_API_KEY=<api_key>
   FRONTEND_URL=http://localhost:3000
   ```

4. Ejecuta las migraciones de la base de datos:

   ```
   npx prisma migrate dev
   ```

---

4. Diseño del Sistema

Fases de Desarrollo:

1. **Análisis:**
    - Requisitos funcionales: manejo de proyectos, tickets y cálculos de pagos.
    - Requisitos no funcionales: seguridad, escalabilidad, usabilidad.
2. **Diseño**
    - Diseño de interfaces en figma https://www.figma.com/design/I1IRe2jqXdaeF6e8ImiTgR/TPO?node-id=0-1&t=AsnODFMutIGxPQDM-1
    - Diagrama de base de datos.
    - Modelado de datos: Prisma define los modelos `Usuario`, `Proyecto`, `Ticket`, `Pago`, `UsuarioXProyecto` y `TicketXProyecto`.
    - Modularidad: backend y frontend están separados para facilitar el mantenimiento.
3. **Implementación:**
    - Desarrollo iterativo con pruebas continuas.
    - Uso de TypeScript para tipado estático y reducción de errores.
4. **Pruebas:**
    - Unitarias y de integración para el backend.
    - Pruebas funcionales y visuales en el frontend.

---

5. Documentación de la API REST

Estructura de la API:
Base URL: `http://localhost:3000/api`

Endpoints Principales:

Autenticación:

- `POST /auth/login` : Iniciar sesión.
- `POST /auth/register` : Registrar un nuevo usuario.
- `POST /auth/closeAccount` : Cerrar cuenta permanentemente.
- `POST /auth/checkEmail` : Verificar disponibilidad de correo electrónico.
- `POST /auth/logout` : Cerrar sesión.

Gestión de Proyectos:

- `POST /projects` : Crear un nuevo proyecto.
- `GET /projects/:userId` : Obtener proyectos de un usuario específico.
- `PATCH /projects` : Editar un proyecto.
- `POST /projects/detail/:prId` : Agregar usuario al proyecto.
- `GET /projects/detail/:prId` : Obtener detalles de un proyecto específico.
- `PATCH /projects/detail/:prId` : Cerrar un proyecto específico.
- `GET /projects/tickets/:prId` : Obtener tickets de un proyecto específico.
- `GET /projects/users/:prId` : Obtener usuarios de un proyecto.
- `GET /projects/usersNotInProject/:prId` : Obtener usuarios que no pertenecen a un proyecto.

Tickets:

- `POST /tickets` : Registrar un ticket.
- `GET /projects/getList/:userId` : Ver tickets asociados a un usuario.
- `POST /tickets/upload` : Subir imagen de un ticket a Cloudinary.

Pagos:

- `GET /payments/byEmisorUserId/:usId` : Obtiene pagos realizados por un usuario.
- `GET /payments/byReceptorUserId/:usId` : Obtiene pagos recibidos por un usuario.
- `GET /payments/byProyectoId/:prId` : Obtiene los pagos de un proyecto.
- `GET /payments/markAsSent/:paId` : Marca un pago como enviado.
- `GET /payments/markAsRecieved/:paId` : Marca un pago como recibido.

Ejemplo de Solicitud:

Crear un Proyecto:

- **Endpoint:** `POST /api/projects`

- **Body:**

```
{
  "pr_nombre": "Viaje de Trabajo",
  "pr_descripcion": "Gastos compartidos para el viaje",
  "us_email": "example@example.com"
}
```

- **Respuesta:**

```
{
 "pr_id": "3",
  "pr_nombre": "Viaje de Trabajo",
  "pr_descripcion": "Gastos compartidos para el viaje",
 "pr_abierto": true
}
```

---

6. Casos de Uso

Caso de Uso: Crear Proyecto

- **Actor Principal:** Usuario registrado.
- **Precondición:** El usuario está autenticado en la aplicación.
- **Flujo Principal:**

1. El usuario accede al panel de proyectos.
2. Selecciona "Crear Proyecto".
3. Completa el formulario con:
    - Nombre del proyecto.
    - Descripción.
4. Guarda el formulario.
5. El sistema:
    - Valida los datos.
    - Crea el proyecto en la base de datos.

Caso de Uso: Agregar Miembro

- **Actor Principal:** Usuario registrado.
- **Precondición:**
    1. El usuario está autenticado en la aplicación.
    2. El proyecto existe.
    3. El usuario a agregar está registrado.
- **Flujo Principal:**
    1. El usuario accede al panel de proyectos.
    2. Selecciona un proyecto.
    3. Selecciona “+”.
    4. Completa el formulario con el nombre del miembro a agregar.
    5. Selecciona “Confirmar”.
    6. El sistema:
        - Valida los datos.
        - Guarda el nuevo usuario junto al proyecto en la base de datos.
        - Envía una notificación por correo.
    -