Proyecto de Integración con Khipu API v3

Este proyecto es una aplicación web construida con Astro que se integra con la API v3 de Khipu para procesar pagos. Permite a los usuarios crear solicitudes de pago y verificar el estado de los pagos existentes.

Estructura del Proyecto

```
Khipu/
├── astro.config.mjs        # Configuración de Astro
├── package.json            # Dependencias y scripts del proyecto
├── README.md               # Este archivo
├── tsconfig.json           # Configuración de TypeScript
├── .env                    # Variables de entorno (contiene KHIPU_API_KEY)
├── public/
│   └── favicon.svg         # Favicon del sitio
└── src/
    ├── env.d.ts            # Definiciones de tipos para variables de entorno
    ├── assets/             # Directorio para assets estáticos (ej. imágenes)
    │   ├── astro.svg
    │   └── background.svg
    ├── components/         # Componentes reutilizables de la UI
    │   ├── Welcome.astro
    │   ├── React/          # Componentes de React
    │   │   ├── PaymentForm.jsx     # Formulario para crear pagos
    │   │   └── PaymentStatus.jsx   # Componente para mostrar el estado del pago
    │   └── Svelte/
    │       └── PaymentForm.svelte # (Ejemplo de componente Svelte, no usado
activamente para Khipu)
    ├── layouts/            # Plantillas base para las páginas
    │   ├── BaseLayout.astro  # Layout principal con navegación y estilos globales
    │   └── Layout.astro      # Otro layout (podría ser para páginas específicas)
    ├── pages/              # Páginas y endpoints de la aplicación
    │   ├── index.astro     # Página principal con formularios de pago y estado
    │   ├── payment-cancelled.astro # Página de redirección para pagos cancelados
    │   ├── payment-success.astro   # Página de redirección para pagos exitosos
    │   └── api/            # Endpoints de la API backend
    │       ├── create-payment.ts # Endpoint para crear un nuevo pago en Khipu
    │       └── payment-status/
    │           └── [id].ts   # Endpoint para verificar el estado de un pago por
su ID
```

Funcionalidades

El proyecto implementa las siguientes funcionalidades principales relacionadas con la API de Khipu:

1. **Creación de Pagos**:
    - Los usuarios pueden ingresar los detalles de un pago (monto, descripción, etc.) a través de un formulario.

- Al enviar el formulario, se realiza una solicitud al endpoint local `/api/create-payment.ts`.
- Este endpoint se comunica con la API de Khipu (v3) para generar una nueva orden de pago.
- Utiliza autenticación HMAC-SHA256 para asegurar la comunicación con Khipu.
- Si la creación es exitosa, Khipu devuelve una URL de pago a la cual el usuario es redirigido.

2. **Verificación de Estado de Pago**:

- Los usuarios pueden ingresar el ID de una transacción de Khipu para consultar su estado.
- Al enviar la solicitud, se llama al endpoint local `/api/payment-status/[id].ts`.
- Este endpoint consulta la API de Khipu (v3) para obtener el estado actual del pago especificado.
- Utiliza autenticación HMAC-SHA256.
- El estado del pago (ej. "pendiente", "pagado", "rechazado") se muestra al usuario, con indicaciones visuales y posibles acciones.

Archivos Clave y Descripción

Variables de Entorno

- `.env`:
  - `KHIPU_API_KEY`: Tu clave secreta de la API de Khipu (Merchant Secret). Es crucial para la autenticación HMAC-SHA256.
  - `KHIPU_MERCHANT_ID`: Tu ID de comerciante de Khipu.

Endpoints de API (Backend - `src/pages/api/`)

- `create-payment.ts`:
  - **Función**: Maneja las solicitudes POST para crear un nuevo pago.
  - **Lógica Principal**:
    - Recibe los datos del pago desde el frontend (monto, asunto, cuerpo, correo del pagador, URLs de retorno/cancelación).
    - Construye el payload para la API de Khipu v3 (`/payments`).
    - Genera la firma HMAC-SHA256 utilizando `KHIPU_MERCHANT_ID` y `KHIPU_API_KEY`.
    - Realiza la solicitud `fetch` a la API de Khipu.
    - Devuelve la URL de pago (`payment_url`) o un mensaje de error.
- `payment-status/[id].ts`:
  - **Función**: Maneja las solicitudes GET para verificar el estado de un pago existente, usando el ID del pago como parámetro dinámico en la URL.
  - **Lógica Principal**:
    - Extrae el `id` del pago de los parámetros de la URL.
    - Construye la URL para la API de Khipu v3 (`/payments/{id}`).
    - Genera la firma HMAC-SHA256.
    - Realiza la solicitud `fetch` a la API de Khipu.
    - Devuelve los detalles del estado del pago (ej. `status`, `status_detail`) o un mensaje de error.

Componentes React (Frontend - `src/components/React/`)

- `PaymentForm.jsx`:
  - **Función**: Renderiza un formulario para que el usuario ingrese los detalles del pago.

    - **Lógica Principal**:
        - Maneja el estado de los campos del formulario (monto, asunto, etc.).
        - Al enviar, realiza una solicitud `fetch` (POST) al endpoint `/api/create-payment.ts`.
        - Si la creación es exitosa, redirige al usuario a la `payment_url` proporcionada por Khipu.
        - Muestra mensajes de error si la creación falla.
- `PaymentStatus.jsx`:
    - **Función**: Renderiza un formulario para ingresar un ID de pago y muestra el estado resultante.
    - **Lógica Principal**:
        - Maneja el estado del campo de ID de pago.
        - Al enviar, realiza una solicitud `fetch` (GET) al endpoint `/api/payment-status/[paymentId]`.
        - Muestra el estado del pago de forma legible, con colores y mensajes según el estado (ej. "Pagado" en verde, "Pendiente" en amarillo).
        - Puede incluir botones para acciones adicionales (ej. "Volver a intentar el pago" si es aplicable).

Páginas Astro (Frontend - `src/pages/`)

- `index.astro`:
    - **Función**: Página principal de la aplicación.
    - **Contenido**: Integra los componentes `PaymentForm.jsx` y `PaymentStatus.jsx` para permitir a los usuarios crear y verificar pagos.
- `payment-cancelled.astro`:
    - **Función**: Página a la que Khipu redirige al usuario si cancela el pago.
    - **Contenido**: Muestra un mensaje indicando que el pago fue cancelado.
- `payment-success.astro`:
    - **Función**: Página a la que Khipu redirige al usuario después de un pago exitoso.
    - **Contenido**: Muestra un mensaje de confirmación de pago exitoso.

Layouts (`src/layouts/`)

- `BaseLayout.astro`:
    - **Función**: Plantilla base para la mayoría de las páginas.
    - **Contenido**: Incluye la estructura HTML común (head, body), estilos globales, y la barra de navegación.

Configuración y Puesta en Marcha

1. **Clonar el Repositorio**:

```
git clone <URL_DEL_REPOSITORIO>
cd Khipu
```

2. **Instalar Dependencias**: Asegúrate de tener Node.js y npm (o pnpm/yarn) instalados.

```
```

```
npm install
# o
# pnpm install
# o
# yarn install
```

3. **Configurar Variables de Entorno**:
    - Crea un archivo `.env` en la raíz del directorio `Khipu/` (al mismo nivel que `astro.config.mjs`).
    - Añade tus credenciales de Khipu:

      ```
      KHIPU_MERCHANT_ID="TU_ID_DE_COMERCIANTE"
      KHIPU_API_KEY="TU_LLAVE_SECRETA_API"
      ```

    - Reemplaza `"TU_ID_DE_COMERCIANTE"` y `"TU_LLAVE_SECRETA_API"` con tus valores reales.
4. **Desarrollo Local**: Para iniciar el servidor de desarrollo de Astro:

   ```
   npm run dev
   # o
   # pnpm dev
   # o
   # yarn dev
   ```

   Esto iniciará la aplicación generalmente en `http://localhost:4321`.
5. **Build para Producción**: Para compilar la aplicación para producción:

   ```
   npm run build
   # o
   # pnpm build
   # o
   # yarn build
   ```

   Los archivos compilados se encontrarán en el directorio `dist/`.

Consideraciones Adicionales

- **Seguridad**: La `KHIPU_API_KEY` es sensible. Asegúrate de que el archivo `.env` esté en tu `.gitignore` y nunca se suba a repositorios públicos.
- **Manejo de Errores**: El proyecto incluye manejo básico de errores, pero podría expandirse para ofrecer una mejor experiencia de usuario y logging más detallado.
- **Estilos**: Los estilos globales se manejan en `BaseLayout.astro` y los componentes pueden tener sus propios estilos encapsulados o utilizar clases globales.

Este README proporciona una guía completa para entender, configurar y ejecutar el proyecto de integración con Khipu.