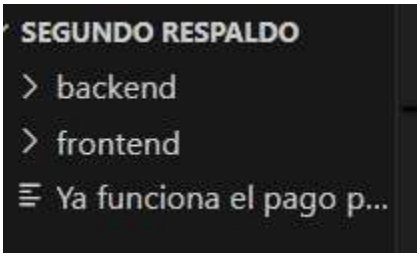
DOCUMENTATION

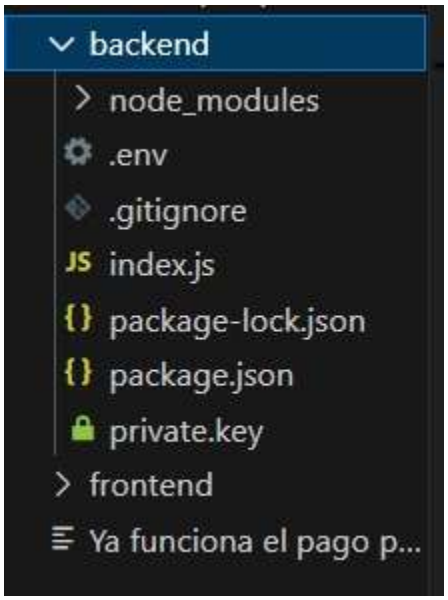
The present system is designed to process tip payments. These are generated by service offers such as taxi services like Uber, food delivery, restaurant service, among others.

To start the project, there is a web application which contains the following structure for its development.



The main folder where the project is developed is called “SEGUNDO RESPALDO”, which in turn contains two folders that host each of the fundamental elements for the functioning of the web application.

We start with the backend, which is responsible for executing the logic that the program will follow to carry out the transactions



Inside the backend folder, there is an “env” folder where the banking data required for the transaction is stored, such as the client, the sending account, the receiving account, and also the key_id.

```
PORT=3000
WALLET_URL=https://ilp.interledger-test.dev/sfia
SENDING_WALLET_URL=https://ilp.interledger-test.dev/rdlfo
RECEIVING_WALLET_URL=https://ilp.interledger-test.dev/mrco
PRIVATE_KEY_PATH=./private.key
KEY_ID=9f374a21-08db-476a-bcc6-02cfc9331cc6
```

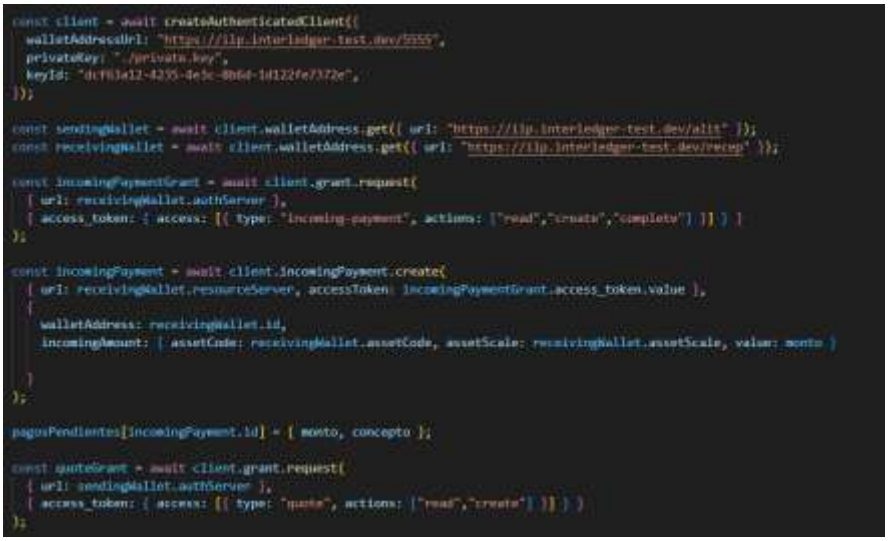
Next is “node_modules”, where all the files that Open Payments uses to instantiate the necessary resources for establishing the connection are located.

Then we have the “index.js” file, which has the function of executing the transactions between Open Payments users.

```
import express from "express";
import cors from "cors";
import {
  createAuthenticatedClient,
  OpenPaymentsClientError,
  isFinalizedGrant,
} from "@interledger/open-payments";

const app = express();
const PORT = 4000;

app.use(cors());
app.use(express.json());


app.get("/", (req, res) => {
  res.send("✅ Backend Open Payments activo! Usa POST /pago para iniciar pagos.");
});

let lastOutgoingGrant = null;
const pagosPendientes = {};

app.post("/pago", async (req, res) => {
  try {
    const { monto, concepto } = req.body;
    if (!monto || !concepto) {
      return res.status(400).json({ error: "Debes enviar monto y concepto" });
    }
```

In this section, the packages from “node_modules” are imported to create a new user, who will either receive or send payments from their Open Payments account.

Likewise, the port on which the web page will be displayed is defined, as well as the fields required to make a payment, such as amount and description.



In this section, constants are created to define our client. Additionally, a grant and an incoming payment are created to carry out the transaction.

Now, a Quote is created for authentication and to execute the payment to be made.

```
    lastOutgoingGrant = { client, outgoingPaymentGrant, sendingWallet, quote };

    res.json({
      message: "Grant interactivo generado. Abre la URL para aceptar el pago",
      url: outgoingPaymentGrant.interact.redirect,
      concepto,
      monto
    });

  } catch (err) {
    if (err instanceof OpenPaymentsClientError) res.status(400).json({ error: err.description || err });
    else res.status(500).json({ error: err.message });
  }
});

app.post("/finalizar-pago", async (req, res) => {
  try {
    if (!lastOutgoingGrant) return res.status(400).json({ error: "No hay grant pendiente" });

    const { client, outgoingPaymentGrant, sendingWallet, quote } = lastOutgoingGrant;

    const finalizedGrant = await client.grant.continue({
      url: outgoingPaymentGrant.continue.uri,
      accessToken: outgoingPaymentGrant.continue.access_token.value,
    });
```

Payment errors within the app are verified in order to complete the transaction, which will be displayed on-screen in a message indicating whether the payment was successfully sent or not.

```
    if (!isFinalizedGrant(finalizedGrant)) return res.status(400).json({ error: "Grant no finalizado, acepta el enlace en el navegador." });

    const outgoingPayment = await client.outgoingPayment.create(
      { url: sendingWallet.resourceServer, accessToken: finalizedGrant.access_token.value },
      { walletAddress: sendingWallet.id, quoteId: quote.id }
    );

    lastOutgoingGrant = null;

    res.json({
      message: "Pago realizado correctamente",
      outgoingPayment
    });

  } catch (err) {
    if (err instanceof OpenPaymentsClientError) res.status(400).json({ error: err.description || err });
    else res.status(500).json({ error: err.message });
  }
});

app.listen(PORT, () => {
  console.log(`🚀 Servidor Open Payments corriendo en http://localhost:${PORT}`);
});
```

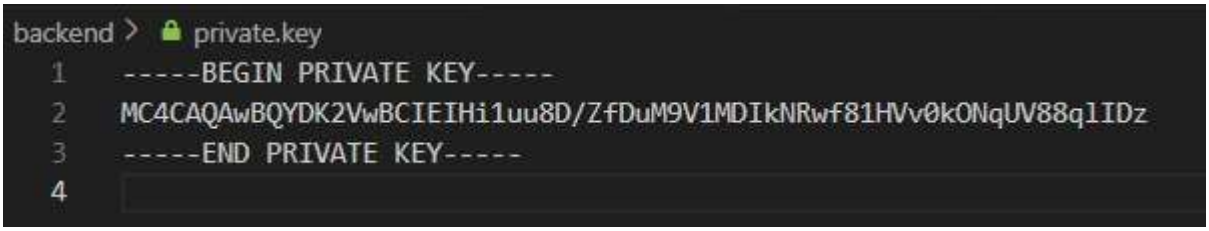
Finally, the server is launched on port 4000, where, if the API is running correctly, this is displayed in the console. Otherwise, an error will be shown.

Now we move on to the “package.json” file.

In this section, it is specified that we are working with modules and the entry points are defined. The Open Payments dependencies are also created.

To finish with the backend, we have the private key of our user who will make the payment through Open Payments.

.



Next, the frontend folder is defined, which has the function of showing, through an interface, how the application works to make payments using Open Payments.

It begins with the “App.css” file, in which the root styles for the main file are implemented, along with styles for logos and spinning animations.

This file is complemented with styles for the cards shown in the app and for the secondary text style.

Then we continue with the “App.jsx” file, where the functions for the initial fields are defined, such as the type of currency to be sent and received, the amount, description, type of service, as well as the types of currencies available for conversion.

```jsx
import { useState } from "react";

function App() {
  const [step, setStep] = useState(1);
  const [grantUrl, setGrantUrl] = useState("");
  const [message, setMessage] = useState("");

  const [monto, setMonto] = useState("");
  const [concepto, setConcepto] = useState("");
  const [tipoServicioOtro, setTipoServicioOtro] = useState("");

  const [origen, setOrigen] = useState("USD");
  const [destino, setDestino] = useState("MXN");
  const [tipoServicio, setTipoServicio] = useState("Transporte");

  const monedas = [
    { code: "PKR", label: "PKR (Pakistán)" },
    { code: "PEB", label: "PEB" },
    { code: "CAD", label: "CAD (Canadá)" },
    { code: "SGD", label: "SGD (Singapur)" },
    { code: "MXN", label: "MXN (México)" },
    { code: "GBP", label: "GBP (Reino Unido)" },
    { code: "ZAR", label: "ZAR (Sudáfrica)" },
    { code: "EUR", label: "EUR (Europa)" },
    { code: "USD", label: "USD (Estados Unidos)" }
  ];

  const tiposServicio = ["Transporte", "Comida", "Venta", "Alojamiento", "Otro"];

  const crearPago = async () => {
    const conceptoFinal = tipoServicio === "Otro" ? tipoServicioOtro : concepto;
    if (!monto || !conceptoFinal) {
      setMessage("Debes ingresar monto y concepto");
      return;
    }
```

Here the methods for processing the payment are defined, so that if it is completed or rejected, the result is shown within the app interface.

```
  setMessage("Generando pago...");
  try {
    const res = await fetch("http://localhost:4000/pago", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ monto, concepto: conceptoFinal, origen, destino, tipoServicio }),
    });
    const data = await res.json();
    if (data.url) {
      setGrantUrl(data.url);
      setMessage("Abre el enlace para aceptar el pago y luego presiona 'Finalizar pago'.");
    } else {
      setMessage(data.error || "Error al crear pago");
    }
  } catch (err) {
    setMessage("Error de conexión con el servidor");
  }
};

const finalizarPago = async () => {
  setMessage("Finalizando pago...");
  try {
    const res = await fetch("http://localhost:4000/finalizar-pago", { method: "POST" });
    const data = await res.json();
    if (data.outgoingPayment) {
      setMessage("✅ Pago completado!");
      setStep(1); // Reinicia el wizard
    } else {
      setMessage(data.error || "Error al finalizar pago");
    }
  } catch (err) {
    setMessage("Error de conexión con el servidor");
  }
};
```

Next is the “main.js” file, where React libraries are imported, as well as the creation of entries for the project’s root folder and the rendering of the web application. Finally, the “index.html” file, which is responsible for giving presentation and structure to the web page that will be the Open Payments app. This file contains the page title, the styles used, and the display of the project’s input fields.

```jsx
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Finally, the “index.html” file, which is responsible for giving presentation and structure to the web page that will be the Open Payments app. This file contains the page title, the styles used, and the display of the project’s input fields.

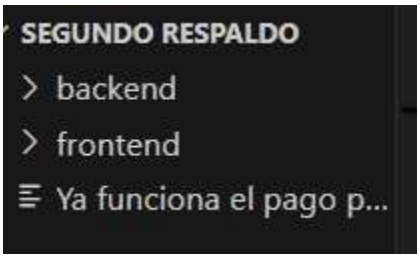
```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Open Payments</title>
    <style>
      body {
        font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;
        margin: 0;
        padding: 0;
        background: linear-gradient(135deg, #ff6ec7, #6ec1ff, #fffa6e);
        background-size: 400% 400%;
        animation: gradientShift 15s ease infinite;
        color: #fff;
      }

      @keyframes gradientShift {
        0% {background-position: 0% 50%;}
        50% {background-position: 100% 50%;}
        100% {background-position: 0% 50%;}
      }

      #root {
        max-width: 600px;
        margin: 50px auto;
        padding: 20px;
        background: rgba(0,0,0,0.25);
        border-radius: 16px;
        box-shadow: 0 8px 25px rgba(0,0,0,0.3);
        text-align: center;
      }

      h1 {
        font-size: 2em;
        margin-bottom: 20px;
        text-shadow: 0 2px 5px rgba(0,0,0,0.5);
      }
```

```
      button {
        transition: all 0.3s ease;
        margin: 5px 0;
      }

      button:hover {
        background: linear-gradient(135deg, #ff4ecf, #4ebfff);
        transform: translateY(-3px) scale(1.05);
      }

      a {
        color: #ff6ec7;
        font-weight: 600;
        text-decoration: none;
      }

      a:hover {
        color: #ff4ecf;
      }

      .grant-section {
        margin-top: 15px;
      }

      .message {
        margin-top: 15px;
        font-weight: bold;
      }
    </style>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

DOCUMENTACIÓN

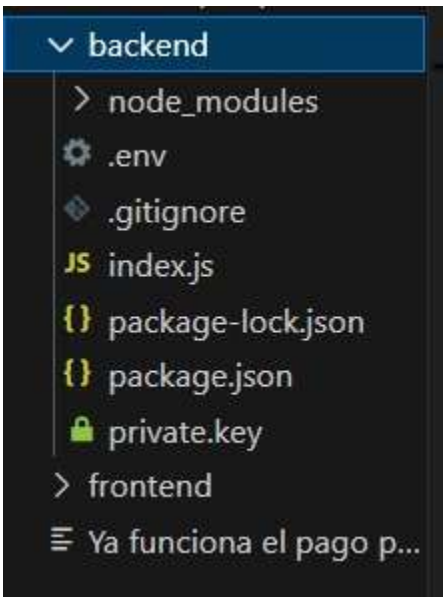
El presente sistema está diseñado para realizar los pagos de propinas, estos son generados por ofertas de servicios como lo es servicio de taxi como Uber, envió de alimentos, atención en restaurantes, entre otros.

Para iniciar el proyecto se tiene la aplicación web el cual contiene la siguiente estructura para su desarrollo.



La carpeta principal donde se desarrolla el proyecto se llama “SEGUNDO RESPALDO”, que a su vez contiene dos carpetas en las cuales esta alojada cada uno de los elementos fundamentales para el funcionamiento de la aplicación web.

Se comienza con el backend el cual es el encargado de ejecutar la lógica que seguirá el programa para realizar las transacciones.

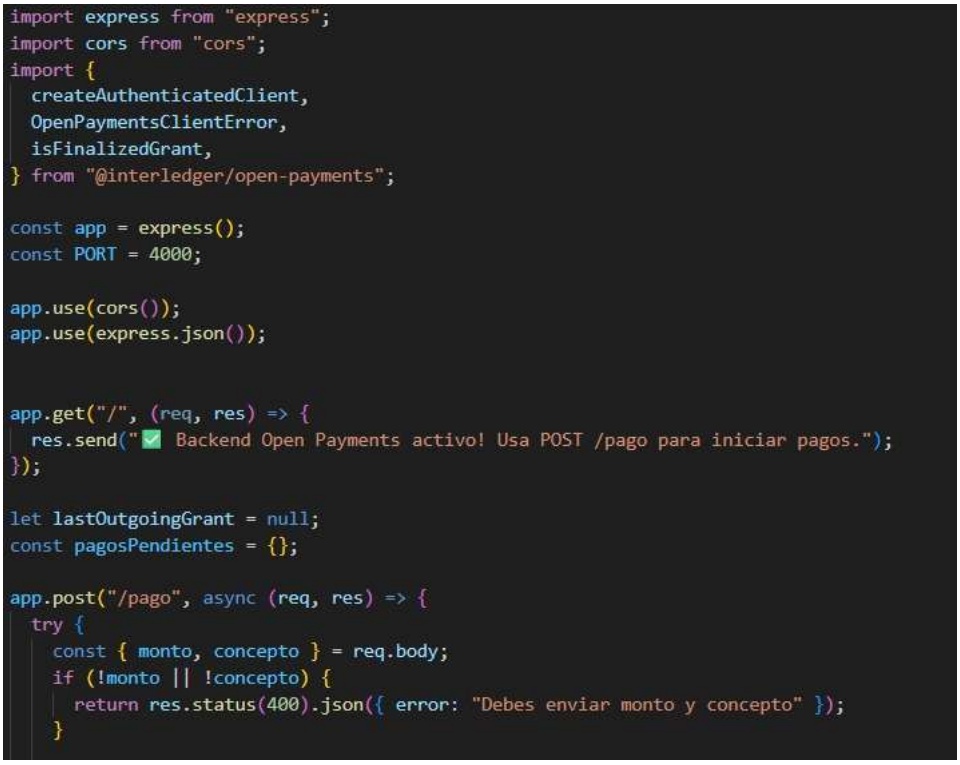


Dentro de la carpeta del backend, se encuentra alojada la carpeta de “env” que es donde se guardan los datos de la banca para realizar la transacción como lo es el cliente, la cuenta de envió y la cuenta que recibe, también la key_id

```
PORT=3000
WALLET_URL=https://ilp.interledger-test.dev/sfia
SENDING_WALLET_URL=https://ilp.interledger-test.dev/rdlfo
RECEIVING_WALLET_URL=https://ilp.interledger-test.dev/mrco
PRIVATE_KEY_PATH=./private.key
KEY_ID=9f374a21-08db-476a-bcc6-02cfc9331cc6
```

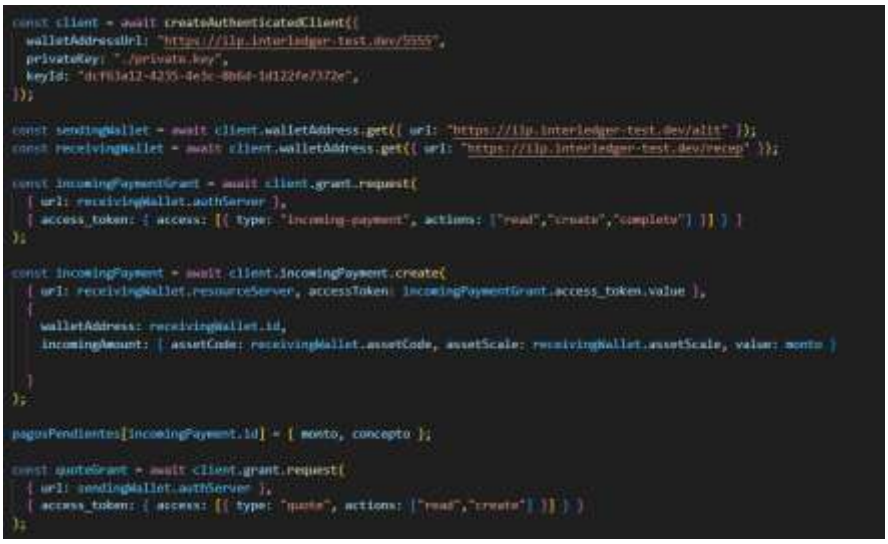
Ahora sigue “node_modules” que es donde se encuentran todos los archivos que open payments al instanciar todos los recursos para realizar la conexión.

Posteriormente se tiene el archivo “index.js” el cual tiene la función de ejecutar las transacciones entre los usuarios de open payments.



```
import express from "express";
import cors from "cors";
import {
  createAuthenticatedClient,
  OpenPaymentsClientError,
  isFinalizedGrant,
} from "@interledger/open-payments";

const app = express();
const PORT = 4000;

app.use(cors());
app.use(express.json());


app.get("/", (req, res) => {
  res.send("✅ Backend Open Payments activo! Usa POST /pago para iniciar pagos.");
});

let lastOutgoingGrant = null;
const pagosPendientes = {};

app.post("/pago", async (req, res) => {
  try {
    const { monto, concepto } = req.body;
    if (!monto || !concepto) {
      return res.status(400).json({ error: "Debes enviar monto y concepto" });
    }
```

En esta sección de importan los paquetes de “node_modules” para crear un nuevo usuario el cual es el que recibirá/enviara pagos desde su cuenta de open payments, así mismo se define el puesto por el que se visualiza la pagina web, de igual forma se tienen los campos para realizar el pago. Como lo es el monto y concepto.



En esta sección se crean las constantes para definir a nuestro cliente, al igual se crea un grant y incoming payment para realizar la transacción.

Ahora se crea un Quote para la autenticación y realización del pago que se va a realizar.

```
    lastOutgoingGrant = { client, outgoingPaymentGrant, sendingWallet, quote };

    res.json({
      message: "Grant interactivo generado. Abre la URL para aceptar el pago",
      url: outgoingPaymentGrant.interact.redirect,
      concepto,
      monto
    });

  } catch (err) {
    if (err instanceof OpenPaymentsClientError) res.status(400).json({ error: err.description || err });
    else res.status(500).json({ error: err.message });
  }
});

app.post("/finalizar-pago", async (req, res) => {
  try {
    if (!lastOutgoingGrant) return res.status(400).json({ error: "No hay grant pendiente" });

    const { client, outgoingPaymentGrant, sendingWallet, quote } = lastOutgoingGrant;

    const finalizedGrant = await client.grant.continue({
      url: outgoingPaymentGrant.continue.uri,
      accessToken: outgoingPaymentGrant.continue.access_token.value,
    });
```

Se verifica los errores de pago dentro de la app para poder completar la transacción, el cual se mostrará en un mensaje, en pantalla si se realizó él envió o no.

```
    if (!isFinalizedGrant(finalizedGrant)) return res.status(400).json({ error: "Grant no finalizado, acepta el enlace en el navegador." });

    const outgoingPayment = await client.outgoingPayment.create(
      { url: sendingWallet.resourceServer, accessToken: finalizedGrant.access_token.value },
      { walletAddress: sendingWallet.id, quoteId: quote.id }
    );

    lastOutgoingGrant = null;

    res.json({
      message: "Pago realizado correctamente",
      outgoingPayment
    });

  } catch (err) {
    if (err instanceof OpenPaymentsClientError) res.status(400).json({ error: err.description || err });
    else res.status(500).json({ error: err.message });
  }
});

app.listen(PORT, () => {
  console.log(`🚀 Servidor Open Payments corriendo en http://localhost:${PORT}`);
});
```

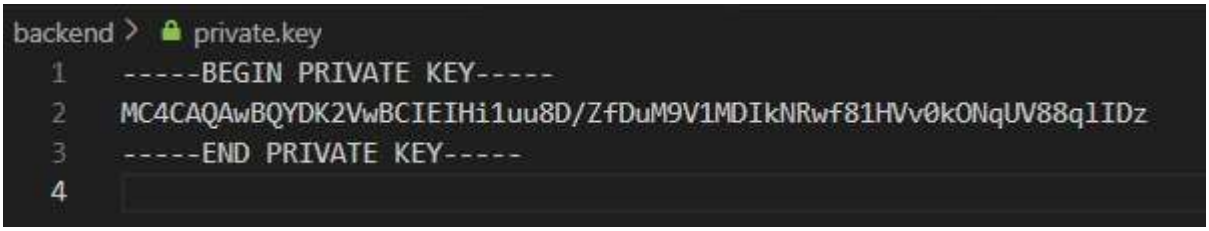
Finalmente, el servidor se carga al puerto 4000 en el cual, si la API esta corriendo correctamente se muestra en la consola, de lo contrario mostrara error-

Ahora pasamos con el archivo “pakage.json”

```
{
  "name": "open-payments-backend",
  "version": "1.0.0",
  "type": "module",
  "main": "index.js",
  ▷Debug
  "scripts": {
    "start": "node ./index.js"
  },
  "dependencies": {
    "@interledger/open-payments": "^7.1.3",
    "express": "^4.19.0",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "uuid": "^9.0.1"
  }
}
```

En esta sección se indica que se está trabajando con módulos y define los puntos de entradas, se crean las dependencias de open payments.

Para finalizar con el backend, se tiene la llave privada de nuestro usuario que realizara el pago desde open payments.



```
backend > private.key
1   -----BEGIN PRIVATE KEY-----
2   MC4CAQAwBQYDK2VwBCIEIHi1uu8D/ZfDuM9V1MDIkNRwf81HVv0kONqUV88qlIDz
3   -----END PRIVATE KEY-----
4
```

A continuación, se define la carpeta del frontend la cual tiene la función de mostrar mediante una interfaz como es que funciona el programa de la app para realizar los pagos mediante open payments.

Para ello comienza por el archivo “App.css” en el cual se implementan los estilos de root para el archivo raíz, los estilos de los logotipos y animaciones de giros.

Este archivo se complementa con los estilos de las tarjetas que se muestran en la app y el estilo de texto secundario.

Posteriormente se continua con el archivo “App.jsx”, en el cual se definen las funciones para los campos iniciales como el tipo de moneda que se quiere enviar y se quiere recibir, el monto, concepto y el tipo de servicio, así como los tipos de monedas que se pueden hacer la conversión.

```jsx
import { useState } from "react";

function App() {
  const [step, setStep] = useState(1);
  const [grantUrl, setGrantUrl] = useState("");
  const [message, setMessage] = useState("");

  const [monto, setMonto] = useState("");
  const [concepto, setConcepto] = useState("");
  const [tipoServicioOtro, setTipoServicioOtro] = useState("");

  const [origen, setOrigen] = useState("USD");
  const [destino, setDestino] = useState("MXN");
  const [tipoServicio, setTipoServicio] = useState("Transporte");

  const monedas = [
    { code: "PKR", label: "PKR (Pakistán)" },
    { code: "PEB", label: "PEB" },
    { code: "CAD", label: "CAD (Canadá)" },
    { code: "SGD", label: "SGD (Singapur)" },
    { code: "MXN", label: "MXN (México)" },
    { code: "GBP", label: "GBP (Reino Unido)" },
    { code: "ZAR", label: "ZAR (Sudáfrica)" },
    { code: "EUR", label: "EUR (Europa)" },
    { code: "USD", label: "USD (Estados Unidos)" }
  ];

  const tiposServicio = ["Transporte", "Comida", "Venta", "Alojamiento", "Otro"];

  const crearPago = async () => {
    const conceptoFinal = tipoServicio === "Otro" ? tipoServicioOtro : concepto;
    if (!monto || !conceptoFinal) {
      setMessage("Debes ingresar monto y concepto");
      return;
    }
```

Ahora se definen los métodos para realizar el pago, en caso que se realice o se rechace se muestre dentro de nuestra interfaz de la app.

```
  setMessage("Generando pago...");
  try {
    const res = await fetch("http://localhost:4000/pago", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ monto, concepto: conceptoFinal, origen, destino, tipoServicio }),
    });
    const data = await res.json();
    if (data.url) {
      setGrantUrl(data.url);
      setMessage("Abre el enlace para aceptar el pago y luego presiona 'Finalizar pago'.");
    } else {
      setMessage(data.error || "Error al crear pago");
    }
  } catch (err) {
    setMessage("Error de conexión con el servidor");
  }
};

const finalizarPago = async () => {
  setMessage("Finalizando pago...");
  try {
    const res = await fetch("http://localhost:4000/finalizar-pago", { method: "POST" });
    const data = await res.json();
    if (data.outgoingPayment) {
      setMessage("✅ Pago completado!");
      setStep(1); // Reinicia el wizard
    } else {
      setMessage(data.error || "Error al finalizar pago");
    }
  } catch (err) {
    setMessage("Error de conexión con el servidor");
  }
};
```

Ahora continua el archivo “main.js” que es donde se importan las librerías de react así como la creación de entradas para la carpeta raíz del proyecto y el renderizado de la aplicación web.

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Ahora continua con el archivo “index.html” el cual es el archivo encargado de darle presentación y orden a la página web que será la app de pagos de open payments, el cual contiene el título de la página, los estilos que utiliza, la visualización de los recuadros de entradas del proyecto.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Open Payments</title>
    <style>
      body {
        font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;
        margin: 0;
        padding: 0;
        background: linear-gradient(135deg, #ff6ec7, #6ec1ff, #fffa6e);
        background-size: 400% 400%;
        animation: gradientShift 15s ease infinite;
        color: #fff;
      }

      @keyframes gradientShift {
        0% {background-position: 0% 50%;}
        50% {background-position: 100% 50%;}
        100% {background-position: 0% 50%;}
      }

      #root {
        max-width: 600px;
        margin: 50px auto;
        padding: 20px;
        background: rgba(0,0,0,0.25);
        border-radius: 16px;
        box-shadow: 0 8px 25px rgba(0,0,0,0.3);
        text-align: center;
      }

      h1 {
        font-size: 2em;
        margin-bottom: 20px;
        text-shadow: 0 2px 5px rgba(0,0,0,0.5);
      }
```