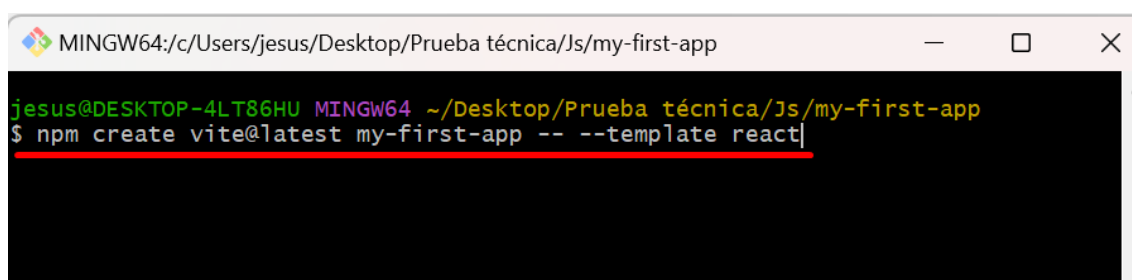


EXPLICACIÓN DE LA APLICACIÓN POR PASOS

CREACIÓN

Lo primero fue determinar el lenguaje que mejor se adaptase a la petición que había recibido. Debido al objetivo marcado, determiné que lo idóneo era usar **React** como marco principal para la aplicación, debido a que gracias al uso de **estados** se nos facilita la labor de gestionar con data externa y la presentación de la misma.

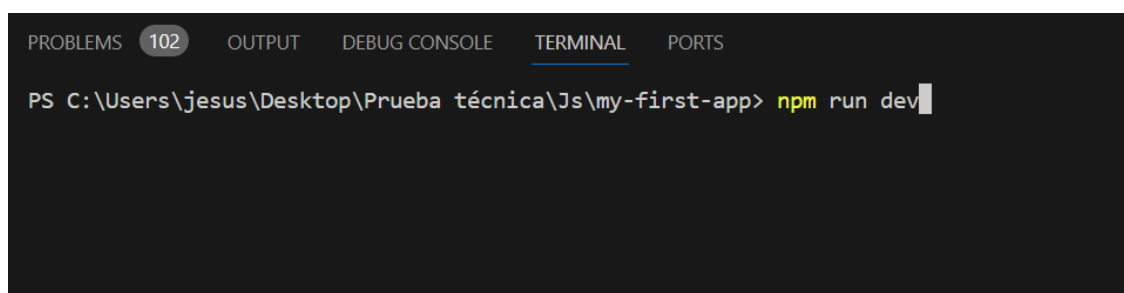
Una vez determinados el framework que usaremos, el siguiente paso era crear un repositorio base desde el cual trabajar para lo que usamos **Vite** (permitiéndonos generar mediante su macro un repositorio por el cual comenzar):



```
MINGW64:/c/Users/jesus/Desktop/Prueba técnica/Js/my-first-app
jesus@DESKTOP-4LT86HU MINGW64 ~/Desktop/Prueba técnica/Js/my-first-app
$ npm create vite@latest my-first-app -- --template react
```

Lo siguiente sería abrir el repositorio e instalar dependencias con el comando “npm install” o “npm i”. Una vez instaladas las dependencias, determinamos qué otras podríamos necesitar para el proyecto, como **Axios**¹ (para la conexión con la Api), **React-spinner** (para introducir un spinner mientras se carga la información) o **React-bootstrap** (para facilitar la configuración de la visualización de la web).

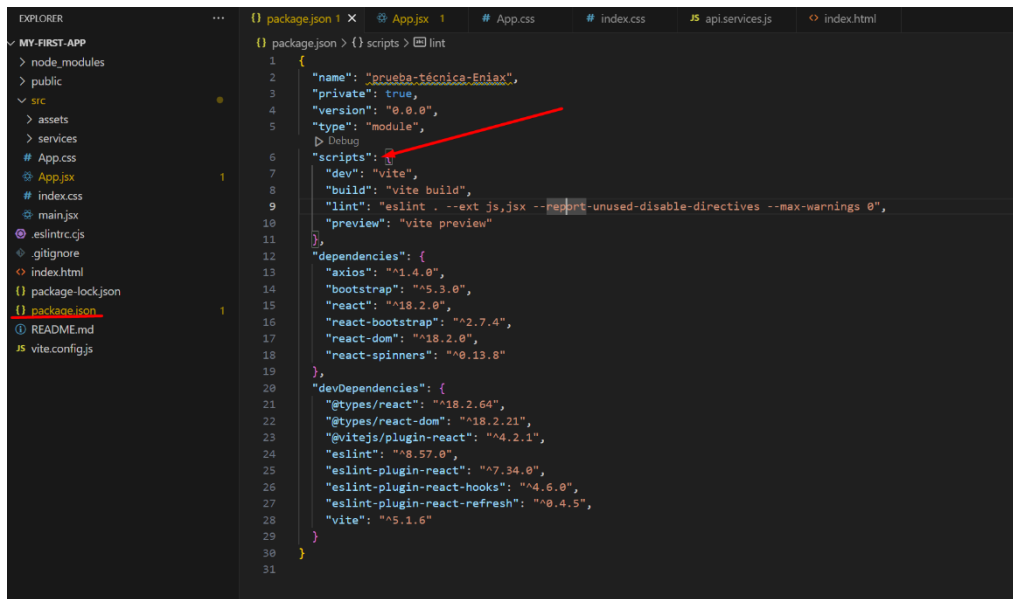
Una vez instalados todas las dependencias necesarias, arrancamos nuestra aplicación con el comando “npm run dev” (que nos permite ir modificando la aplicación y ver los cambios directamente con cada guardado):



```
PROBLEMS 102 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\jesus\Desktop\Prueba técnica\Js\my-first-app> npm run dev
```

¹ Hemos elegido Axios en lugar de Fetch por motivos de optimización. Dado que generábamos la app desde 0 y sin restricciones, parecía la opción más correcta.

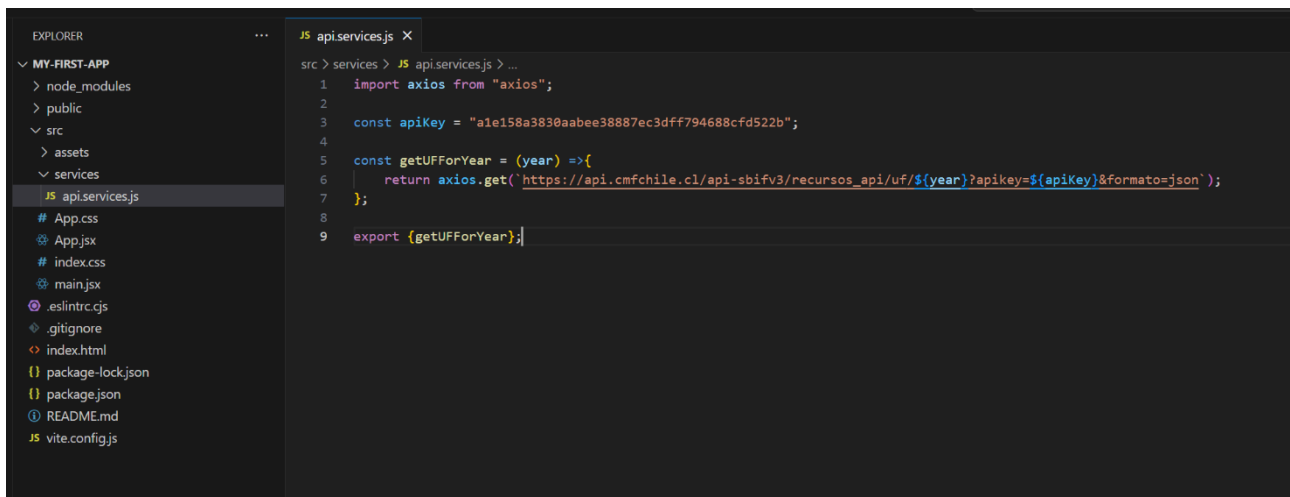
Estos comandos podemos contemplarlos en el “package.json”:



```
1 {
2   "name": "prueba-técnica-Eniax",
3   "private": true,
4   "version": "0.0.0",
5   "type": "module",
6   "scripts": {
7     "dev": "vite",
8     "build": "vite build",
9     "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",
10    "preview": "vite preview"
11  },
12  "dependencies": {
13    "axios": "^1.4.0",
14    "bootstrap": "^5.3.0",
15    "react": "^18.2.0",
16    "react-bootstrap": "^2.7.4",
17    "react-dom": "^18.2.0",
18    "react-spinners": "^0.13.8"
19  },
20  "devDependencies": {
21    "@types/react": "^18.2.64",
22    "@types/react-dom": "^18.2.21",
23    "@vitejs/plugin-react": "^4.2.1",
24    "eslint": "^8.57.0",
25    "eslint-plugin-react": "^7.34.0",
26    "eslint-plugin-react-hooks": "^4.6.0",
27    "eslint-plugin-react-refresh": "^0.4.5",
28    "vite": "^5.1.6"
29  }
30 }
```

CONEXIÓN CON LA API

Para realizar la conexión con la api estableceremos una estructura de servicios, lo cual nos permitirá escalar la aplicación de forma sencilla en un futuro. Para ello creamos en la carpeta “src” una carpeta que llamamos “services” en la que crearemos un primer servicio para establecer la conexión con la api:



```
1 import axios from "axios";
2
3 const apiKey = "a1e158a3830aabee38887ec3dff794688cfd522b";
4
5 const getUfforYear = (year) =>{
6   return axios.get('https://api.cmfchile.cl/api-sbifv3/recursos_api/uf/${year}?apikey=${apiKey}&formato=json');
7 };
8
9 export {getUfforYear};
```

Previamente hemos tenido que ponernos en contacto con el servicio para que nos proporcionasen la key. Generalmente (por motivos de seguridad), este tipo de datos se deberían de poner en un archivo encriptado y usarse como una variable global (que no sería accesible para el usuario), pero dado que estamos en una prueba técnica, he optado por dejar la variable en el mismo archivo para facilitar la labor.

Una vez definidas la función por el cual obtenemos los datos (función a la que le pasamos el valor del año que deseamos buscar), la exportamos para poder hacer uso de la misma.

DESARROLLO DE LA APP

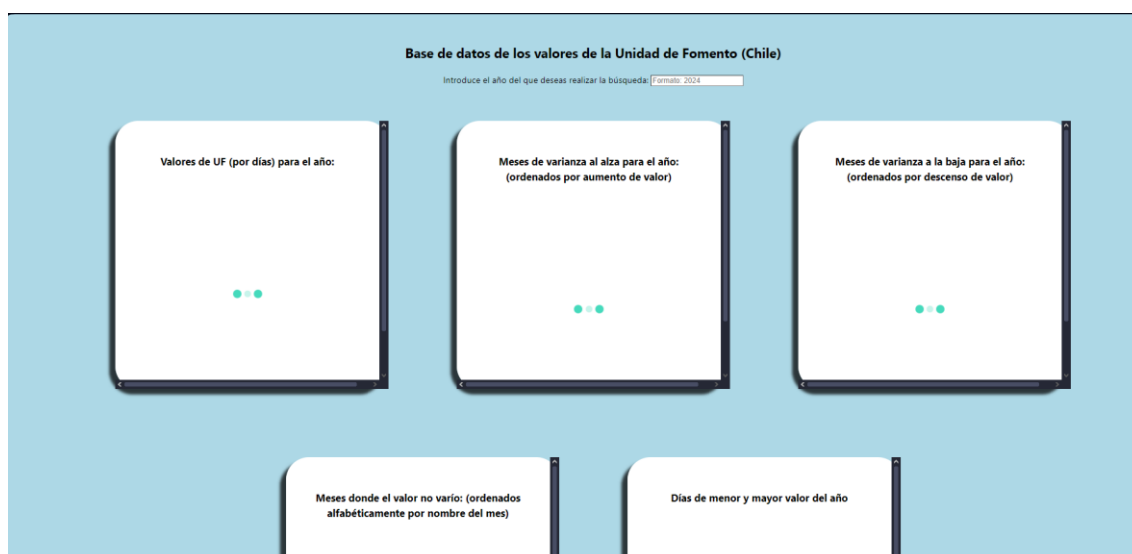
Todo el código principal se encuentra en el archivo **“App.jsx”**. Para conseguir nuestro objetivo, dividiremos nuestro código en “Estados” (donde definiremos nuestros estados de react), “Funciones” (desde funciones para actualizar la data en tiempo real como, por ejemplo, el año, que llamaremos “handlers”, el `useEffect` para gestionar el reload de la página y funciones asíncronas para la gestión de recepción de dato) y el “return” (donde realizaremos la presentación de la app).

Todo el código está propiamente comentado, pero aún así destacaré un par de cosas para su aclaración. Primero, hemos decidido usar la estructura **“async/await”** y **“try/catch”** (para gestionar los errores²) para nuestras funciones asíncronas, como por ejemplo la función `“getData”`.

Por otro lado, con el uso de los estados de React hemos ido guardando las variables en los mismos de tal manera que se cumplan las condiciones requeridas (de orden, etc.) y los mismos se fuesen actualizando cuando el usuario introduce el año en el buscador.

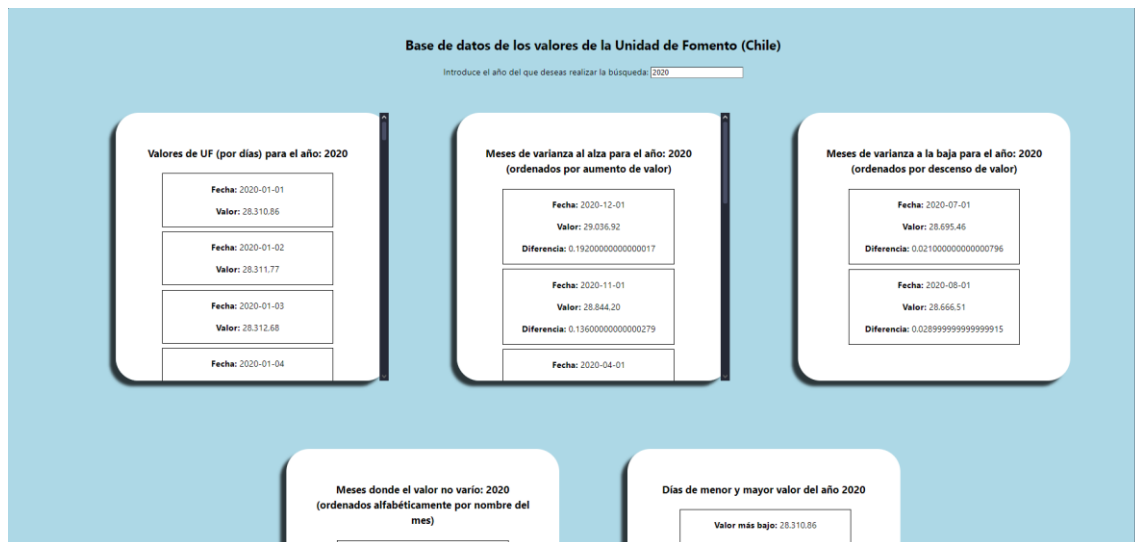
FUNCIONAMIENTO

El resultado final sería el siguiente:



Al inicio de la app no tenemos data incluida en los cuadrantes, por lo que el spinner estaría presente. Una vez insertamos el año de búsqueda en el input superior, el resultado sería el siguiente:

² Podríamos haber utilizado el `react-router-dom` y su `useNavigate` para hacer redirecciones en caso de ciertos errores o para establecer distribución por páginas del resultado, pero debido a la simplicidad de nuestro objetivo hemos decidido simplificar el resultado.



Se establecen 2 filas de cuadrantes para facilitar la visibilidad de los datos, siendo el resultado el ya mostrado.