

Documentación

Front-end Xaxis Disco

Octubre 2022

Descripción

Proyecto front-end desarrollado en React@18, encargado de comunicarse con la api DISCO y mostrar los resultados según interfaz previamente revisada con Xaxis.

Instalación y aspectos técnicos

En este proyecto se utilizaron principalmente las siguientes librerías:

- **webpack@5.74.0**: Para compilar los archivos tanto en desarrollo como producción.
- **react / react-dom@18.2.0**: Como base para desarrollar todo el front.
- **typescript@4.8.2**: Principalmente para manejar el tipado del proyecto (componentes y datos).
- **axios@0.27.2**: Para manejar la comunicación con la api.
- **react-router-dom@6.3.0**: Para manejar las rutas del proyecto.
- **jest@27.5.1**: Agregar test a la aplicación.
- **stubby**: Para simular la api (si no tenemos todavía un ambiente de desarrollo)
- Es necesario tener instalado Node >= 14.

Para levantar la aplicación en modo desarrollo o producción se pueden ejecutar los siguientes comandos:

```
"scripts": {  
  "start": "webpack --config ./webpack/webpack.prod.js && node server.js",  
  "start:dev": "webpack serve --config ./webpack/webpack.dev.js",  
  "start:stubs": "concurrently --kill-others \"npm run start\" \"npm run start:stubby\"",  
  "start:stubby": "stubby -d stubs/stubby-config.yml -w -s 8200 -q",  
  "build": "webpack --config ./webpack/webpack.prod.js",  
  "test": "jest --config=tests/jest.config.js",  
  "test:watch": "jest --coverage --watchAll --config=tests/jest.config.js"  
},
```

Modo desarrollo:

1. **Agregar variables de entorno:** Se agrega un archivo .env en la raíz del proyecto con la siguiente información:

```
.env
1 DISCO_API=http://localhost:8200/disco/api
2 ACCOUNT_ID=6d1ea72d-8173-4aa2-b0f4-7cd574c5d24c
3 PORT=9000
```

2. **Instalar dependencias:** Necesitamos instalar todas las librerías del proyecto antes de levantarlo.

```
npm install
```

3. **Ejecutar en modo desarrollo:**

```
npm run start:dev
```

* Si no tenemos levantada la api desde el back, podemos simular una en otra terminal ejecutando `npm run start:stubby`.

* `ACCOUNT_ID`: Es una variable por defecto sólo para esta primera etapa.

Modo producción:

1. **Agregar variables de entorno:** Se configuran las siguientes variables, con la api productiva.

```
.env
1 DISCO_API=http://localhost:8200/disco/api
2 ACCOUNT_ID=6d1ea72d-8173-4aa2-b0f4-7cd574c5d24c
3 PORT=9000
```

2. **Instalar dependencias:** Necesitamos instalar todas las librerías del proyecto.

```
npm install
```

3. **Ejecutar en modo desarrollo:**

```
npm run build
```

Para ejecutar test:

1. **Instalar dependencias:** Necesitamos instalar todas las librerías del proyecto.

```
npm install
```

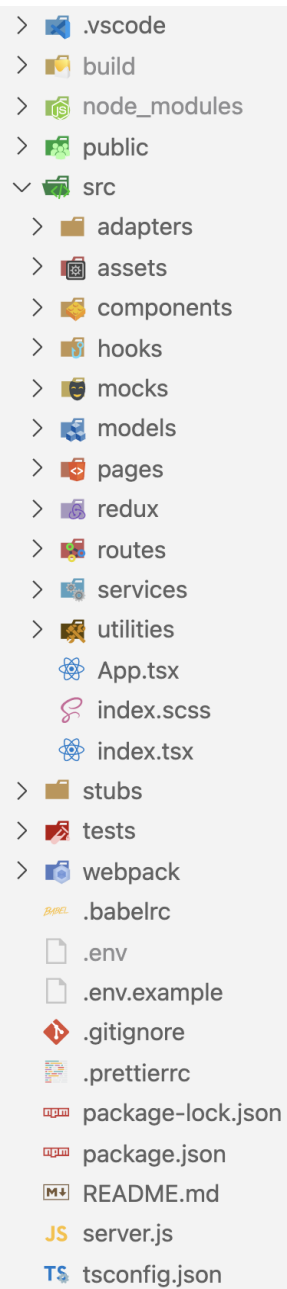
2. **Ejecutar test:**

```
npm run test
```

 o si queremos en modo desarrollo

```
npm run test:watch
```

Estructura del proyecto



A screenshot of a file explorer showing the project structure. The root directory contains the following items:

- > .vscode
- > build
- > node_modules
- > public
- ▼ src
 - > adapters
 - > assets
 - > components
 - > hooks
 - > mocks
 - > models
 - > pages
 - > redux
 - > routes
 - > services
 - > utilities
 - App.tsx
 - index.scss
 - index.tsx
- > stubs
- > tests
- > webpack
- .babelrc
- .env
- .env.example
- .gitignore
- .prettierrc
- package-lock.json
- package.json
- README.md
- server.js
- tsconfig.json

package.json

Contiene la definición de todas las librerías instaladas en el proyecto.

webpack

Carpeta con los archivos de configuración para el build de la aplicación, tanto en desarrollo como producción.

tests

Configuración para lograr ejecutar los test

stubs

Definiciones para simular api DISCO

src/adapters

Transformación de objetos de datos de entrada y salida.

src/assets

Recursos de imágenes, fuentes, íconos y estilos de la aplicación.

src/components

Contiene todos los componentes generales de la aplicación como botones, inputs.

src/hooks

Contiene todos los hooks generales de la aplicación, principalmente utilizados para el manejo de datos.

src/mocks

Aquí están todos los datos que NO se obtienen desde la api y son creados por el front para simular datos.

src/models

Aquí se encuentran principalmente las interfaces de los datos que se manejan en toda la aplicación,

src/pages

Contiene todas las vistas de la aplicación, aquí se manejan todos los módulos.

Cada componente puede tener distintas extensiones como:

- `AccountSetup.tsx` (componente)
- `accountSetup.scss` (estilos)
- `accountSetup.test.tsx` (test)

src/redux

Manejador de estados de toda la aplicación.

src/routes

Definición de rutas

src/services

Aquí es donde manejamos los endpoints para llamar al back

src/utilities

Funciones que se utilizan a nivel general en toda la aplicación