

Manual Técnico

app-inclusiva

Tecnologías utilizadas

NPM

Para manejar los paquetes de la aplicación utilizamos NPM

Github

Utilizamos github para manejar el proyecto y mantener sincronizado todo el proyecto entre los integrantes del equipo, también se usa para que **Vercel** tome el repositorio y cree una página que todos pueden ver y usar. Se usan el manejador de proyectos de github para mantener una lista de issues que se deben ir completando en el sprint.

Firebase

Se utiliza Firebase como principal método para la autenticación de los usuarios y dentro de Firebase se utilizan los métodos de autenticación de Google y Facebook. Para Facebook se necesita crear una cuenta de desarrollador para poder obtener un App ID y un App Secret.

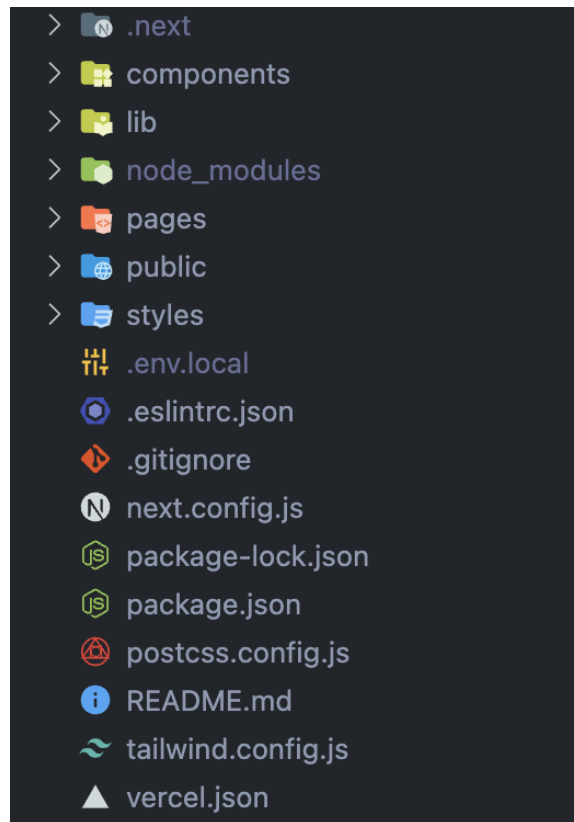
Se utiliza Firebase Storage para almacenar las fotos de los juegos y también para los archivos como apk, exe y html para que los usuarios puedan descargarlo.

Se utiliza Firebase Database para poder guardar el catálogo de juegos y los usuarios autenticados, los usuarios autenticados tienen un username que es otra tabla relacionada al uid del usuario.

Nextjs

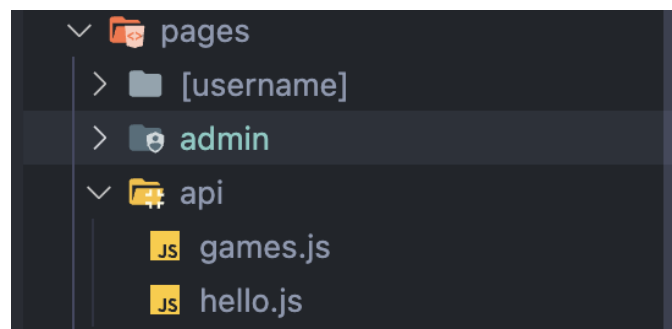
Utilizamos el framework Nextjs basado en Reactjs, este framework facilita el routing de la aplicación, el consumo de APIs y también es más rápido y seguro que Reactjs.

El proyecto utiliza una estructura en donde se tiene una carpeta especial para crear **components** reutilizables. Una carpeta **lib** para guardar métodos y lógica de librerías requeridas como Firebase. La carpeta **pages** que es muy importante para Nextjs, este framework toma el nombre de los archivos dentro de esa carpeta y los mapea a rutas, en este caso si hacemos un componente catalogo.js, el framework lo hace ruta y si entramos a **/catalogo**, se renderiza ese componente.



Estructura general de la app

Dentro del folder pages tenemos un folder api, esto también es importante porque aquí se manejan las apis para obtener información dentro de nuestros componentes, se hace un método que llame a una api externa y luego se utiliza en los componentes que queramos.



Folder api con archivos para obtener información

Se hace fetch a la api de itch.io con la librería de superagent y se regresa el response

```
const superagent = require("superagent");

// Api para obtener los juegos de itch.io que tiene el usuario tiene creados
export default function handler(req, res) {
  superagent.get(`https://itch.io/api/1/${process.env.NEXT_PUBLIC_API_KEY}/my-games`).then(
    (response) => {
      res.status(200).send(response.text);
      return response.text;
    }
  );
}
```

```
useEffect(() => {
  const getGamesInfo = async () => {
    setLoading(true);
    const response = await fetch("/api/games");
    if (response) {
      const data = await response.json();
      setLoading(false);
      setGamesInfo(data.games);
    }
    setLoading(false);
  };


  getGamesInfo();
}, []);
```

Vercel

Utilizamos la plataforma Vercel para el hosting de la aplicación, esto nos permite de forma rápida sincronizar los cambios que se realizan en nuestro repositorio de github y los cambios se ven en la web en pocos segundos. Gracias a esto podemos hacer pruebas más rápidas con el usuario para que no tenga que instalar nada de forma local y con el link que nos genera Vercel podemos acceder a la aplicación Web. También nos permite configurar nuestra aplicación para que otras personas no nos roben nuestras claves de apis, gracias a las variables de ambiente que se pueden crear y utilizar con `NEXT_PUBLIC_{nombre de la variable}`



```
(`https://itch.io/api/1/${process.env.NEXT_PUBLIC_API_KEY}/my-games`)
```

 <https://app-inclusiva.vercel.app/>

Link a la aplicación web

Itch.io


Para poder jugar en la Web, decidimos utilizar la plataforma Itch.io, ahí se pueden publicar de forma segura y rápida los juegos tipo HTML y para los demás juegos la gente puede descargar los archivos, esto también nos permite llevar un mejor control de cuantas personas han entrado a jugar el ese juego.

Utilizamos la API de itch.io para poder obtener la el juego y la cantidad de personas que han interactuado con el.

<https://itch.io/api/1/KEY/my-games>

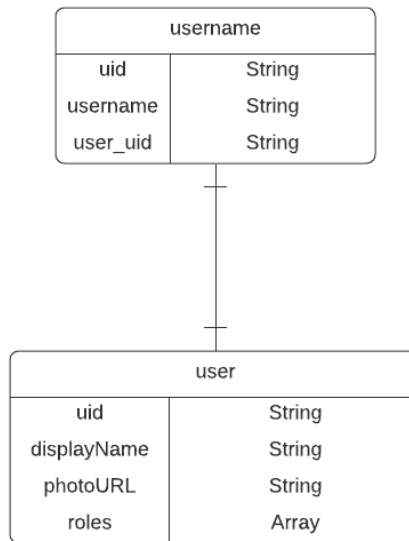
Serverside API reference

The itch.io server-side API lets you query the information about your games by making HTTP requests to API URLs. All API endpoints require authentication. There are two types of credentials you can use: API keys are long-lasting credentials that can be revoked by users JWT tokens are short-lived, expiring credentials They both

 <https://itch.io/docs/api/serverside>

Modelo de la base de datos

juego	
uid	String
image	String
publicado	Boolean
tipo	String
titulo	String
url	String
descripcion	String



Patrones utilizados

Provider Pattern (Patrón de proveedor): Este tipo de patrón se utiliza para que cierta información sea compartida con todos los componentes de la aplicación. Esto lo usamos principalmente para que todos los componentes tuvieran acceso a la información del usuario autenticado y puedan ser renderizados dependiendo de esa información. Se utiliza el createContext() de React y toda la aplicación se encierra con un proveedor (Provider).

```
1 import "../styles/globals.css";
2 import Navbar from "../components/Navbar";
3 import { UserContext } from "../lib/context";
4
5 import { useUserData } from "../lib/hooks";
6
7 function MyApp({ Component, pageProps }) {
8   const userData = useUserData();
9
10  // Se utiliza un hook y el contexto para que la app siempre tenga la información sobre el usuario autenticado
11
12  return (
13    <UserContext.Provider value={userData}>
14      <Navbar />
15      <Component {...pageProps} />
16    </UserContext.Provider>
17  );
18 }
19
20 export default MyApp;
```

Component pattern (Patrón de componentes): Este tipo de patrón es utilizado para crear componentes reutilizables, en nuestro caso lo usamos para mostrar el menu de navegación y las tarjetas de los juegos que tenemos guardados en la base de datos (Firebase).