NPM Deploy

Paso #0:
Pulir el código lo más posible, prepararlo para la migración, asegúrate de nombres, css, casos extremos, etc.

Usualmente cuando nos damos cuenta que un código puede convertirse en un paquete de NPM por si solo, es porque ya lo creamos en una aplicación, ahí es donde me refiero a realizar la pulida inicial.

Paso #1: Crear paquete
npx tsdx create <nombre del paquete>

Si les pregunta, React + TypeScript, posiblemente la primera vez les pida instalar de forma global TSDX (aceptar)

Dejar todo lo que formará parte del paquete en la carpeta SRC

tsdx.io - JavaScript y React

Paso #2: Optimizar index.tsx
El archivo **src/index.tsx**, es el punto de entrada de todas las importaciones, por lo que ahí debe de tener las exportaciones que las personas u otros desarrolladores importarán.

Podemos crear exportaciones de esta forma para no perder nuestra estructura.

```
export * from './components';
```

Paso #3: (Opcional) Módulos
Si tu código que estás creando tiene imágenes importadas de esta forma y/o CSS modularizado de esta forma:

```
import noImage from '../assets/no-image.jpg';
import styles from '../styles/styles.module.css';
```

Necesitaremos crear un archivo de configuración de TSDX "**tsdx.config.js**" en la raíz que nos ayudará en el proceso de construcción de nuestro paquete.

Ver ejemplo del archivo **tsdx.config.js** en la siguiente página:

tsdx.config.js

```
const postcss = require('rollup-plugin-postcss');
const images = require('@rollup/plugin-image');

module.exports = {
  rollup(config, options) {

    config.plugins = [
      postcss({ modules: true }),
      images({ incude: ['**/*.png', '**/*.jpg'] }),
      ...config.plugins,
    ];

    return config;
  },
};
```

Realizar las instalaciones respectivas:
yarn add -D rollup-plugin-postcss
yarn add -D @rollup/plugin-image

Mas información sobre tsdx.config.js aquí

Paso #4: Build

Ejecutar el comando
`yarn build`

Corregir cualquier error que aquí aparezca.

Paso #5: Example

Crear un ejemplo de cómo se usa el código, usualmente personas que tengan curiosidad y necesidad de un ejemplo, irán a verlo.

Paso #6: GitHub Repo

Este paso aunque suene opcional, es importante para la longevidad del proyecto, puede que eventualmente decidas dejarlo y heredarlo a otra persona que lo continuará o invitar colaboradores que puedan realizar actualizaciones o bien aceptar mejoras que otras personas puedan hacer a tu paquete.

Adicionalmente tratar de mantener release tags acorde a la versión del paquete que puedes observar en el package.json

Colocar la referencia de tu repositorio en el package.json que se encuentra en el root del proyecto.

```
"repository": {
  "url": "https://github.com/<tu repositorio>",
  "type": "git"
},
```

Homepage - Opcional:

De tener un sitio web personal (puede ser GitHub Pages), agregar la llave **homepage**, en el package.json

```
"homepage": "https://fernando-herrera.com",
```

Keywords - Opcional:

Si quieres hacer un paquete que pueda ser fácilmente visible por la comunidad y que sea indexado por los bots de Google (y otros), añadir la llave **keywords** dentro del package.json

```
"keywords": [
  "product",
  "card",
  "fernando",
  "herrera"
]
```

Paso #7: Pruebas automáticas

Es importante asegurarnos que nuestro paquete tenga pruebas automáticas para asegurarnos que funciona como esperamos el día de mañana, esto no asegura que el paquete funcionará sin errores, pero por lo menos tendremos la seguridad de que las funcionalidades principales están probadas y siguen funcionando en cada Release. Esto reduce enormemente el estrés de que nuevas versiones puedan tener breaking changes no intencionales.

Preferencia personal: **Jest**
1- Instalar las dependencias:

yarn add --dev jest babel-jest @babel/preset-env @babel/preset-react react-test-renderer

2- Instalar la contraparte de TypeScript
yarn add @types/react @types/react-dom @types/react-test-renderer

Ignorar imágenes y css cargados como módulos de JavaScript que darán error. **Añadir la siguiente configuración en el package.json**

```
"jest":{
    "moduleNameMapper": {
     "\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|woff2|mp4|webm|wav|mp3|
m4a|aac|oga)$": "identity-obj-proxy",
     "\\.(css|less|scss|sass)$": "identity-obj-proxy"
   }
 }
```

3- Preferencia personal, configurar modo observador de cambios en testing, crear el siguiente script en el **package.json**:

```
"test:watch": "tsdx test --watch",
```

Paso #8: Publicar

1- Crear una cuenta en NPM

2- Realizar el login en la consola o terminal (Colocar toda la información solicitada)

```
npm login
```

3- Ejecutar el siguiente comando para publicar la aplicación:

```
yarn publish
```

NPM Update - Actualización

Una actualización se resume:

1. Actualizar la carpeta SRC - Si aplica
2. Actualizar el example - Si aplica
3. Subir la versión en el package.json
4. Realizar la actualización en el repositorio local y remoto
5. Se recomienda crear un nuevo release tag
6. Ejecutar nuevamente el yarn publish