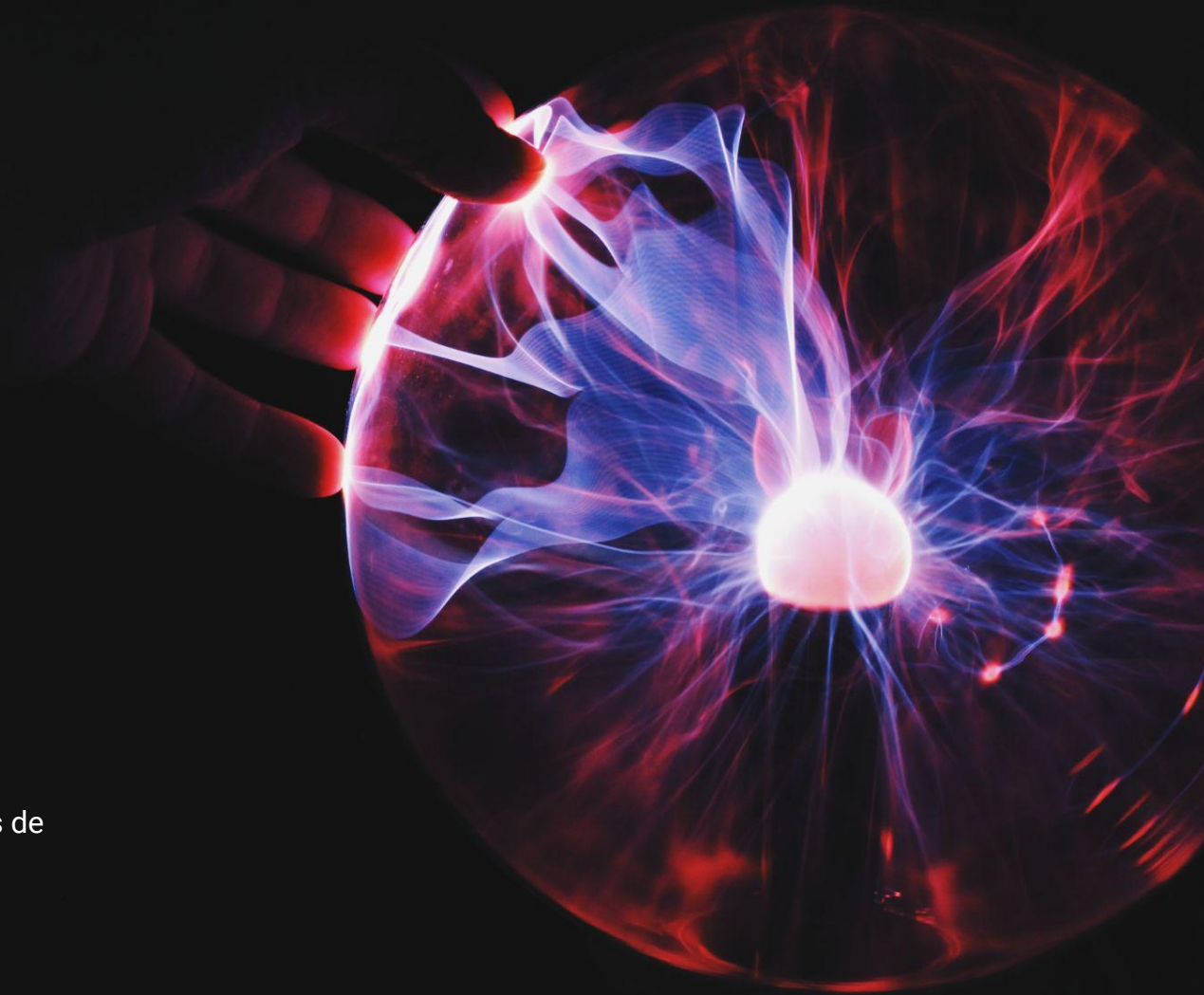


INPUT | OUTPUT

Web2 -> Web3

Cómo transformar un proyecto NEXTjs de
Web2 a Web3



+ WEB 1, 2, y 3 _

Web 1, 2, y 3: Diferencias entre Web 1, 2, y 3 - A GRANDES RASGOS

WEB 1	WEB 2	WEB 3
Centrado alrededor de “leer”	Centrado alrededor de “leer”, “interactuar”, y “publicar”	Centrado alrededor de “leer”, “interactuar”, “publicar”, y “poseer”
Páginas estáticas	Páginas y aplicaciones dinámicas	Páginas y aplicaciones dinámicas y “descentralizadas”
Páginas renderizadas en el servidor (el cliente no hacía mucho)	Tanto el servidor como el cliente trabajan	Idem Web 2, pero con mayor descentralización
HTML es 👑	HTML + JS + JSON comparten la 👑	Idem Web 2

Web 1, 2, y 3: Diferencias entre Web 1, 2, y 3 - LA REALIDAD

MARKETING!

La evolución de la web es constante y continua (no discreta):

<https://thehistoryoftheweb.com/timeline/>

Y no hay una razón técnica real para trazar la línea en ningún lado

Web 1, 2, y 3: Qué le digo al jefe/inversor/gerente no técnico?

- **Web1:** Página web con tecnología vieja y estática
- **Web2:** Página o aplicación web moderna e interactiva
- **Web3:** Web2 + interactúa con alguna blockchain

+ CREAR

PROYECTO Web3 🕶️

Crear proyecto Web 3 🕶️: Proyecto Web 2 + librerías que interactúan con

1. Crear proyecto Web2:
 - a. HTML + JS + CSS
 - b. Tu Framework favorito: **NextJS**, Vue, Flutter, Django, IHP, Play, ~~Laravel~~, ~~Rails~~, etc.
2. Instalar librerías para interactuar con la blockchain (Cardano):
 - a. JS: **MeshJS**, Plu-ts, Lucid Evolution, Blaze, etc.
 - b. Python: PyCardano
 - c. Haskell: Atlas
 - d. Scala: Scalus

Crear proyecto Web 3 🕶️: Proyecto Web 2 + librerías que interactúan con

1. Crear proyecto NextJS 14 (sin **app** router):

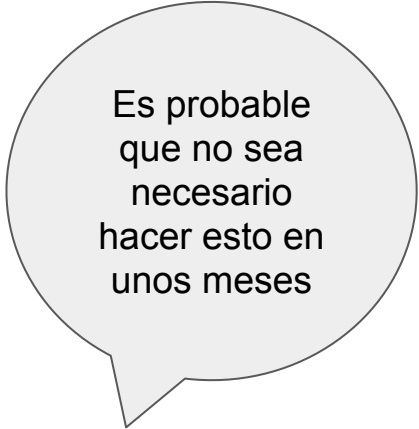
```
npx create-next-app@14 --typescript
```

2. Instalar MeshJS:

```
npm install @meshsdk/core @meshsdk/react
```


Crear proyecto Web 3 🕶️: Configurar Webpack (next.config.mjs)

```
/** @type {import('next').NextConfig} */  
const nextConfig = {  
  reactStrictMode: true,  
  webpack: function (config, options) {  
    config.experiments = {  
      asyncWebAssembly: true,  
      layers: true,  
    };  
    return config;  
  },  
};  
export default nextConfig;
```



Es probable
que no sea
necesario
hacer esto en
unos meses

Crear proyecto Web 3 🕶️: Agregar MeshProvider context

- `<MeshProvider>` provee contexto relacionado con Mesh.
- Esto permite mantener el estado de nuestra Dapp sincronizado entre páginas
- Para utilizarlo, envolvemos el root de nuestro proyecto con el provider en `_app.tsx`:

```
export default function App({ Component, pageProps }: AppProps) {  
  return (  
    <MeshProvider>  
      <Component {...pageProps} />  
    </MeshProvider>  
  );  
}
```

Crear proyecto Web 3 🕶️: LISTO!

LISTO!

Ahora podemos usar funciones y componentes siguiendo los docs en:

<https://meshjs.dev/>

Crear proyecto Web 3 🕶️: Pro Tip

- Ahora que entendemos cómo llegar a este punto, podemos adaptar cualquier proyecto web2 a Web3.
- Pero para proyectos nuevos, es más rápido usar la CLI:

```
npx create-mesh-app starter-next-ts-template
```

Fin

Preguntas?



INPUT | OUTPUT