

FRONTEND:

NextJS: framework para React

Next.js es un framework de desarrollo web basado en React que permite la renderización del lado del servidor (SSR) y la generación de sitios estáticos (SSG). Desarrollado por Vercel, Next.js ofrece una configuración mínima y automática, facilitando el desarrollo de aplicaciones web optimizadas para rendimiento y SEO. Entre sus características destacadas se incluyen el enrutamiento automático basado en el sistema de archivos, la carga automática de componentes y la optimización de imágenes. Además, Next.js soporta API routes, lo que permite crear API's directamente dentro de la aplicación (aunque esta parte de API routes, back-end, no la veremos).

<https://nextjs.org/docs>

Creando un proyecto NextJS

Crear un proyecto con Next.js es sencillo y rápido. Sigue estos pasos para empezar:

🔗 Asegúrate de tener Node.js y npm instalados (Node.js version \geq v20.9+ is required)
<https://nodejs.org/en>

1. Crear una nueva aplicación Next.js:

Abre tu terminal y ejecuta el siguiente comando para crear una nueva aplicación:

`npx create-next-app my-app` 🔗 selecciona "No, customize settings"

```
Ok to proceed? (y)
? Would you like to use the recommended Next.js defaults? > - Use arrow-keys. Return to submit.
  Yes, use recommended defaults
  No, reuse previous settings
> No, customize settings - Choose your own preferences
```

```
✓ Would you like to use the recommended Next.js defaults? > No, customize settings
✓ Would you like to use TypeScript? ... No / Yes
✓ Which linter would you like to use? > ESLint
✓ Would you like to use React Compiler? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like your code inside a `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to use Turbopack? (recommended) ... No / Yes
✓ Would you like to customize the import alias (`@/*` by default)? ... No / Yes
```

Esto generará un nuevo directorio llamado 'my-app' con todos los archivos necesarios.
Navega al directorio del proyecto:

```
cd my-app
```

Inicia el servidor de desarrollo:

```
npm run dev
```

Esto iniciará el servidor y podrás ver tu aplicación en el navegador en <http://localhost:3000>

Explora la estructura de directorios del proyecto:

src/app/: Aquí se definen las rutas y páginas de tu aplicación.

public/: Contiene archivos estáticos como imágenes.

Archivos de configuración: .eslintrc.json, .gitignore, jsconfig.json, next.config.js, package.json

Eliminamos src/app/**page.js** por defecto y creamos **src/app/page.jsx**

```
export default function HomePage() {
  return <h1>Hola Mundo</h1>
}
```

Revisemos **src/app/layout.jsx** (puede cambiar ligeramente, lo importante es {children})

```
import './globals.css'

import { Inter } from 'next/font/google'

const inter = Inter({ subsets: ['latin'] })

export const metadata = {
  title: 'My App',
  description: 'Aprendiendo NextJS',
}

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body className={inter.className}>{children}</body>
    </html>
  )
}
```

Creemos un directorio **src/app/about/** y dentro **src/app/about/page.jsx**

```
export default function AboutPage() {  
  return <h1>About</h1>  
}
```

Podríamos crear la estructura de directorios que deseemos bajo **src/app**, por ejemplo: **src/app/contact/book/page.js** y así sucesivamente.

```
export default function Book() {  
  return (  
    <div>  
      <h1>Reserva</h1>  
      <ul>  
        <li>Mesa</li>  
      </ul>  
    </div>  
  )  
}
```

De este modo podría acceder a mis páginas creadas tal que:

- <http://localhost:3000>
- <http://localhost:3000/about>
- <http://localhost:3000/contact/book>
-

Layout.js

El contenedor de toda la aplicación que recibe el componente hijo como prop con:
export default function.RootLayout({ children })

Podría añadir un navbar que aplique a todas mis páginas:

```
export default function RootLayout({ children }) {  
  return (  
    <html lang="en">  
      <body className={inter.className}>  
        <h1>Navbar</h1>  
        {children}  
      </body>  
    </html>  
  )  
}
```

2. Navegación

Para cambiar de ruta cuando estamos en una página, en layout.js usamos el componente Link (en vez de <a>). De modo que solo trae los datos que cambien (no vuelve a recargar toda la página).

```
import Link from 'next/link'
export default function RootLayout({ children }) {
  return (
    <html lang="es">
      <body>
        <nav>
          <h1>Navbar</h1>
          <ul>
            <li>
              <Link href="/">Home</Link>
            </li>
            <li>
              <Link href="/about">About</Link>
            </li>
            <li>
              <Link href="/contact/book">Book</Link>
            </li>
          </ul>
        </nav>
        {children}
      </body>
    </html>
  )
}
```

🔗 si añadimos dentro del directorio **src/app/** un fichero con distinto nombre a page.js o layout.js (nombres reservados), para Next sería transparente.

Me puedo llevar código a ficheros dentro de **src/components/Navbar.js**:

```
import Link from 'next/link'
export default function Navbar() {
  return (
    <nav className='bg-slate-400 mb-4 flex justify-between items-center px-20 p-3 font-bold text-black'>
      <Link href="/">
        Home
      </Link>
      <ul>
        <li>
          <Link href="/about" >
            About
          </Link>
        </li>
      </ul>
    </nav>
  )
}
```

```

        </li>
        <li>
          <Link href="/contact/book" >
            Book
          </Link>
        </li>
      </ul>
    </nav>
  )
}

```

E importarlo en `src/app/layout.js`

```

import './globals.css'
import Navbar from '@components/Navbar'
export const metadata = {
  title: 'My App',
  description: 'app',
}
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>
        <Navbar />
        {children}
      </body>
    </html>
  )
}

```

3. Layout

Podemos crear varios Layouts para cada componente que queramos englobar.

Por ejemplo, dentro del subdirectorio **contact/book/** puedo crear un `layout.js`.

Solo hay un `RootLayout` (de la de la raíz), pero podemos crear contenido solo para **book**, el cual devuelve la prop `children` dentro de un fragment.

En **`app/contact/book/info/page.js`**

```

export default function Info() {
  return (
    <div>
      <h1>Info</h1>
      <p>Lorem ipsum</p>
    </div>
  )
}

```

```
)  
}
```

En **app/contact/page.js**

```
function Contact() {  
  return (  
    <div>  
      <h1>Contact</h1>  
      <ul>  
        <li>E-mail</li>  
        <li>Phone number</li>  
        <li>Social Media</li>  
      </ul>  
    </div>  
  )  
}  
export default Contact
```

y en **app/contact/layout.js**

```
import Link from 'next/link'  
  
export const metadata = {  
  title: "Contacto",  
}  
  
export default function BookLayout({ children }) {  
  return (  
    <>  
      <nav>  
        <h3>Sección Contacto</h3>  
        <ul>  
          <li>  
            <Link href="/contact/book">Book</Link>  
          </li>  
          <li>  
            <Link href="/contact/book/info">Info</Link>  
          </li>  
        </ul>  
      </nav>  
      {children}  
    </>  
  )  
}
```

Hemos creado una sub-navegación y vemos que aparece en **book** y sus subdirectorios, como `app/contact/book/info`

Por lo que puedo crear sub-navegaciones o paneles laterales. Solo se renderiza la parte que cambia.

📌 un directorio se compone de un `page.js` y, opcionalmente, de un `layout.js`

SEO

Relacionado con SEO, aparición en buscadores (cómo indexan)

- Título y descripción correctos
 - `<head>`
 - `<title>`
 - `<meta name="description" content="">`
 - `<meta name="keywords" content="">`

Lo añadimos dentro del layout principal con (también en los otros layouts si se requiere):

```
export const metadata = {
  title: "Mi tienda",
  description: "Esta es la página principal de mi tienda",
  keywords: "tienda, online, ecommerce"
```

📌 Puedes verlo con inspeccionar del navegador, en `<html><head>`

Más información en: <https://nextjs.org/docs/app/building-your-application/optimizing/metadata>

Fonts

En el `layout.js` principal:

```
import {Roboto} from 'next/font/google'
```

Como Roboto es una clase, tengo que instanciarla, por ejemplo:

```
const roboto = Roboto({
  weight: ["300", "400", "500", "700"],
  styles: ["italic", "normal"],
  subsets: [latin]
})
```

Y lo llamo en

```
<body className={roboto.className}>
```

Ahora tengo las font-face importadas en mi app. Más info en **Google Fonts**.

Nos quedaría en **src/app/layout.js** algo como:

```
import Navbar from '@components/Navbar'
import { Roboto } from 'next/font/google'

import './globals.css'
export const metadata = {
  title: 'Bildy App',
  description: 'Gestiona tus albaranes',
  keywords: "tienda, online, ecommerce"
}

const roboto = Roboto({
  weight: ["300", "400", "500", "700"],
  styles: ["italic", "normal"],
  subsets: ["latin"]
})

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body className={roboto.className}>
        <Navbar />
        {children}
      </body>
    </html>
  )
}
```

Not Found (Personalizar error HTTP 404)

En **src/app/not-found.jsx**

```
import Link from "next/link"

export default function NotFound() {
  return (
    <section>
      <h1>404</h1>
      <p>Página no encontrada</p>
      <Link href="/">
        Volver
      </Link>
    </section>
  )
}
```



```
</section>
```

```
)
```

```
}
```

4. React Server Components

⚠ IMPORTANTE: Server Components vs Client Components

En Next.js (App Router), por defecto TODOS los componentes son Server Components, lo que significa:

- Se ejecutan en el servidor
- NO tienen acceso a hooks del cliente (useState, useEffect, etc.)
- NO pueden manejar eventos (onClick, onChange, etc.)
- Pueden acceder directamente a bases de datos o APIs del servidor
- Reducen el bundle JavaScript enviado al cliente

Usa "use client" solo cuando necesites:

- Interactividad (botones, formularios)
- Hooks de React (useState, useEffect, useContext)
- Acceso a APIs del navegador (localStorage, window, etc.)
- Librerías que dependen del navegador

Todo lo que tenemos en **app** es procesado en el servidor (no se renderiza en el cliente).

Si necesitas interactividad, como botones y formularios, hazlo en la parte cliente porque necesita renderizar ahí en vez de en el servidor ("use client").

Si no lo haces, verás el típico error de Next.js: *"If you need interactivity, consider converting part of this to a Client Component"*

Para ello, utiliza al principio de la página o componente:

```
"use client"
```

```
function aboutPage() {...}
```

📌 más info de cuándo usar "client" o "server" en:

<https://nextjs.org/docs/app/building-your-application/rendering/composition-patterns#when-to-use-server-and-client-components>

Simplemente por usar un Hook como `useState()`, necesitas el "use client". Por lo que deberías usarlo siempre que necesites algo del cliente (como acceder al localStorage).

Si esta función con "use client" tiene componentes hijos, ya no es necesario indicar el "use client" a estos aunque renderice en el cliente debido a que está dentro de un componente que sí lo usa.

📌 si solo lo necesito en el hijo, lo indico ahí solamente, así puedo poner *metadata* en el padre si es parte servidora, ya que solo puedo indicar *metadata* en el server.

5. Fetch data (desde el server)

src/app/posts/[page.js](#)

// Estrategias de Rendering en Next.js (App Router)

// SSR: Datos en tiempo real (peor performance, se actualiza en cada request)

// SSG Contenido estático (mejor performance, se actualiza en build time)

// ISR contenido semi-dinámico (performance medio, se actualiza cada X segundos)

// 1. SSR - Server-Side Rendering (sin caché)

```
async function loadPosts() {  
  const res = await fetch('https://jsonplaceholder.typicode.com/posts', {  
    cache: 'no-store' //Siempre actualizado, sin cache  
  })  
  return res.json()  
}
```

// 2. SSG - Static Site Generation (caché completo)

```
async function loadPosts() {  
  const res = await fetch('https://api.example.com/posts', {  
    cache: 'force-cache' // Por defecto, caché permanente })  
  return res.json()  
}
```

// 3. ISR - Incremental Static Regeneration (caché con revalidación)

```
async function loadPosts() {  
  const res = await fetch('https://api.example.com/posts', {  
    next: { revalidate: 3600 } // Revalidar cada hora })  
  return res.json()  
}
```

```
async function PostPages() {  
  const posts = await loadPosts()  
  return <div>  
    {
```

```

        posts.map(post => (
          <div key={post.id}>
            <h3>{post.id}. {post.title}</h3>
            <p>{post.body}</p>
          </div>
        ))
      }
    </div>
  }
}

export default PostPages

```

Si pudiéramos dentro del return un:

```

<button onClick={() => {}}>
  Click
</button>

```

fallaría. Así que...

¿Cómo podríamos tener un div con botones, pero donde los datos vinieran del server?

Podemos crear un componente para que lo haga:

src/components/PostCard.js (tal y como hemos visto en React, pasándole props.post) y, además, le añadimos el "use client" pero solo a **PostCard**.

```

"use client"
async function PostCard({ post }) {
  return <div>
    {
      <div>
        <h3>{post.id}. {post.title}</h3>
        <p>{post.body}</p>
        <button onClick={() => {
          alert('Click ok!')
        }}>
          Click
        </button>
      </div>
    }
  </div>
}

```

Y en **src/app/posts/page.js**

```
import PostCard from '@components/PostCard'

async function loadPosts() {
  const res = await fetch('https://jsonplaceholder.typicode.com/posts');
  const data = await res.json();
  return data;
}

async function PostPages() {
  const posts = await loadPosts()
  return (
    <div className='grid'>
      {
        posts.map(post => (
          <PostCard post={post} key={post.id} />
        ))
      }
    </div>
  )
}
```

export default PostPages

📌 en *import paquete from @/path/paquete*, @ hace referencia a la raíz (src/) de la app. Puedes configurarlo en jsconfig.json

Loading Page

¿Cómo podemos saber desde el front qué está ocurriendo mientras cargan los datos (fetch data desde el server)?

Vamos a forzar un setTimeout de unos segundos para verlo.

```
/** En la función loadPosts() {
  ...
  await new Promise((resolve) => setTimeout(resolve, 3000))
}*/
```

📌 probadlo con la rueda: “vaciar la cache y volver a cargar”

Puedo crear un fichero **src/app/posts/loading.jsx**

📌 está en minúscula porque no son componentes de React.

```
function LoadingPage() {
  return (
    <div>
```

```

    <h1>Loading...</h1>
  </div>
)
}
export default LoadingPage

```

6. Params

Supongamos que quiero una página para ver publicaciones.

Podría crear un directorio `posts/1/` y dentro un `page.js`, pero si mis ids son del 1 al 100, sería mejor hacer un número dinámico.

Así que cambio el directorio tal que `[id]` (puedo darle el nombre que quiera).

Y para capturarlo, lo tomo de *props.params*, en este caso, *params.id*

Si lo queremos para consultar un back-end (API):: creo una función `loadPost()`

Creo una cabecera en el component `PostCard`, de modo que le añadimos un link que nos envíe el correspondiente *id*.

src/components/PostCard.js

```

"use client"
import Link from 'next/link'

async function PostCard({ post }) {

  return (
    <div className="bg-gray-950 p-10">
      {
        <div>
          <Link href={` /posts/${post.id}`}>

```

```

        <h3 className='text-xl font-bold mb-4'>{post.id}.
{post.title}</h3>
        </Link>
        <p className='text-slate-300'>{post.body}</p>
        <button onClick={() => {
            alert('Click ok!')
        }}>
            Click
        </button>
    </div>
}
</div>
)
}
export default PostCard

```

src/app/posts/[id]/page.js

```

async function loadPost(id) {
    const res = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`)
    const data = await res.json()
    return data
}

async function Page({ params }) {
    //console.log(props)
    const post = await loadPost(params.id)
    return (
        <div>
            <h1>Post Page {params.id}</h1>
            <h2>{post.id}. {post.title}</h2>
            <p>{post.body}</p>
            <hr />
            <h3>Otras publicaciones</h3>
        </div>
    )
}
export default Page

```

Suspense

¿Qué ocurre si tarda en cargar una parte de la página?

Volvemos a activar el `setTimeout` (en `posts/page.js`).

Carga una por id, pero espera a que cargue la parte completa.

¿Cómo lo cargo individualmente? Es decir, que la publicación n que es más rápido se muestre, y que la parte de otras publicaciones cargue individualmente.

Lo usamos en **src/app/posts/[id]/page.js** (que es quien lo usa):

```
import {Suspense} from 'react'
```

y ponemos `<Posts>` dentro de `<Suspense fallback={<div>Cargando...</div>}>`

Es muy útil si tenemos una página muy cargada de componentes (no tenemos que esperar al componente más pesado).

```
import Posts from '../page'
import { Suspense } from 'react'

async function loadPost(id) {
  const res = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`)
  const data = await res.json()
  return data
}

async function Page({ params }) {
  //console.log(props)
  const post = await loadPost(params.id)
  return (
    <div>
      <h1>Post Page {params.id}</h1>
      <h2>{post.id}. {post.title}</h2>
      <p>{post.body}</p>
      <hr />
      <h3>Otras publicaciones</h3>
      <Suspense fallback={<div>Cargando publicaciones...</div>}>
        <Posts />
      </Suspense>
    </div>
  )
}
```

Estilos (CSS)

Si vamos a añadir estilos globales, lo haremos en el layout principal.

En **src/app/globals.css** definimos los estilos globales, y lo importamos en **src/app/layout.js** (`import './globals.css'`)

📌 tailwind lo vemos en el siguiente apartado

```
@tailwind base;
@tailwind components;
@tailwind utilities;

body {
  background: #202020;
  color: white;
}
```

Ahora, por ejemplo, creamos **src/components/Navbar.css**

```
.navbar {
  background: #101010;
  display: flex;
  justify-content: space-between;
  padding: 0 4rem;
  align-items: center;
}

.navbar ul {
  display: flex;
  gap: 1rem;
}

.navbar ul li {
  list-style: none;
}
```

y lo importo en **src/components/Navbar.js** (`import './Navbar.css'`)


```

import Link from 'next/link'
import './Navbar.css'

export default function Navbar() {
  return (
    <nav className="navbar py-5">
      <Link href="/">
        <h1 className="text-3xl font-bold">Clase de Nextjs</h1>
      </Link>
      <ul>
        <li>
          <Link href="/">Home</Link>
        </li>
        <li>
          <Link href="/about">About</Link>
        </li>
        <li>
          <Link href="/contact">Contact</Link>
        </li>
        <li>
          <Link href="/contact/book">Booking</Link>
        </li>
        <li>
          <Link href="/contact/book/info">Info</Link>
        </li>
        <li>
          <Link href="/posts">Posts</Link>
        </li>
      </ul>
    </nav>
  )
}

```

Lo mismo para **src/app/posts/Posts.css**

```

.grid {
  display: grid;
}

```

```
grid-template-columns: repeat(3, 1fr);
grid-gap: 1rem;
margin: 1rem 0;
padding: 4rem;
}
```

e importar desde **src/app/posts/page.jsx** (`import './Posts.css'`)

1. CSS Framework: Tailwind

<https://tailwindcss.com/>

En `src/app/globals.css` teníamos:

```
@tailwind base;
```

```
@tailwind components;
```

```
@tailwind utilities;
```

Ahora, podemos ir modificando los ficheros `components/Navbar.jsx` y `components/PostCard.jsx` con las clases de Tailwind

Por ejemplo:

```
<h1 className="text 3xl font-bold">
```

Revisa la documentación: <https://tailwindcss.com/docs/utility-first>

📌 como ayuda, puedes instalar la extensión de VS Code: Tailwind CSS IntelliSense

Ver tema Tailwind CSS

Primer proyecto con Next.js

`npm create-next-app nextproj`

(selecciona los valores por defecto)

TypeScript? No

ESLint? Yes

Tailwind CSS? Yes

src/ directory? Yes

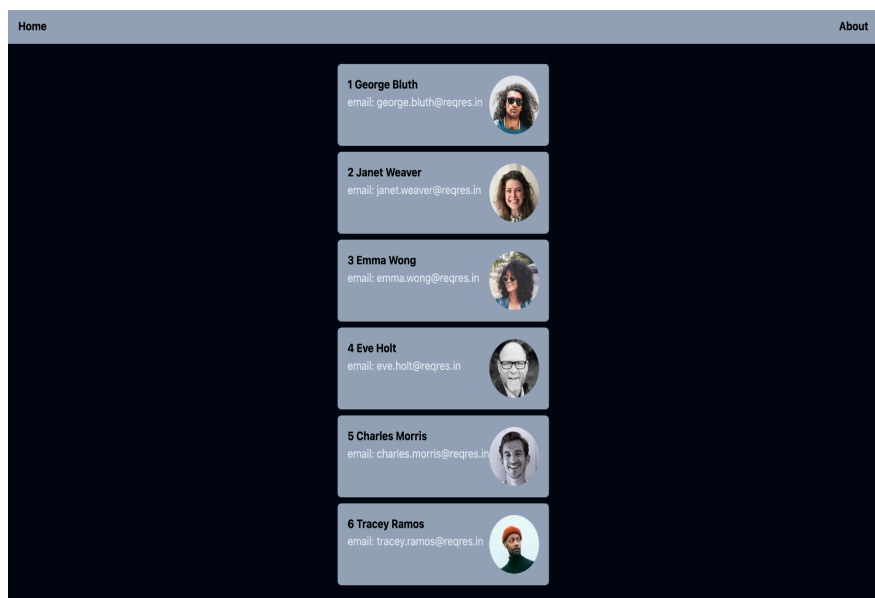
App Router? Yes

import alias? No

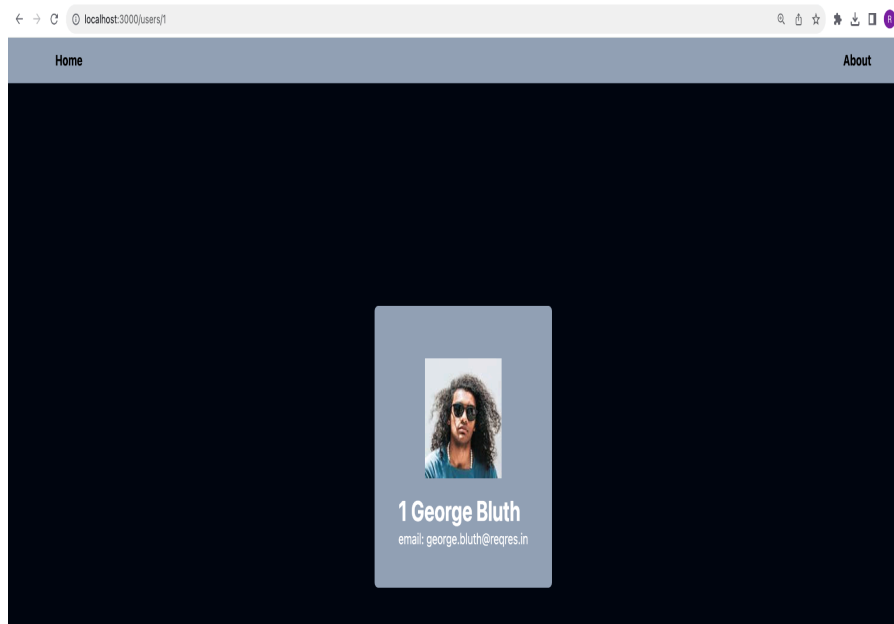
`cd nextproj`
`npm run dev`

Sigue los siguientes pasos:

- Borra el contenido de `src/app/page.js`, dejando un `<div>HomePage</div>`
- Borra el contenido de `src/app/globals.css` (excepto los `@tailwind`)
- Para descargar datos de usuarios podrás llamar al API: `reqres.in`
 - Get List Users: `https://reqres.in/api/users`
- En `src/app/page.js` crea una función `fetchUsers` (que llame a lo anterior)
- Aplicad css de tailwind (en `layout.js` con `className` de tailwind) o personalizadas.
- Cread un `src/components/Users.js` (con “use client”) y añadirle un Link a cada user.
- También cread un `src/components/Navbar.js` con un about
- Ejemplo de visualización para todos:



- y para uno con `app/users/[id]/page.js`



3. Use Router

Next Navigation (similar al componente Link)

Por ejemplo, cambiamos aboutPage:

Si quiero añadir un botón con un Link dentro de about para enviarlo a Home, podría hacerlo. Pero si quiero que antes de redireccionar haga alguna acción, como traer datos de un back-end, no puedo hacerlo con Link porque solo redirecciona, no ejecuta lógica.

Lo haré con useRouter que tendrá funciones de back, forward, push, ... Y antes podrá ejecutar lógica.

en **src/app/about/page.js**

```
"use client"
import {useRouter} from 'next/navigation'

function AboutPage() {
  const router = useRouter()
  return <section>
    <h1>About</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, ...</p>
    <button className="bg-sky-400 px-3 py-2 rounded-md"
      onClick={() => {
        alert("Executing...")
        router.push('/')
      }}
    >
      Click
  </section>
}
```

```
    </button>
  </section>
}
export default AboutPage;
```

4. Deploy de nuestra aplicación en la nube

Podemos usar [Vercel](#)

5. Extra: Autenticación y sesión con middleware.js

```
// middleware.js
import { NextResponse } from 'next/server'

export function middleware(request) {
  const token = request.cookies.get('token')?.value
  const { pathname } = request.nextUrl

  // Redirects simples, NO seguridad crítica
  if (token && pathname === '/') {
    return NextResponse.redirect(new URL('/dashboard', request.url))
  }

  return NextResponse.next()
}

export const config = {
  matcher: ['/((?!api|_next/static|_next/image|.*\\.png$).*)'],
}
```

⚠ ACTUALIZACIÓN IMPORTANTE (2025):

El middleware **YA NO** debe ser la única línea de defensa para autenticación. Debido a vulnerabilidades de [seguridad](#) (CVE-2025-29927), Next.js recomienda un enfoque multi-capa.

ENFOQUE RECOMENDADO: Data Access [Layer](#) (DAL)

1. Crear lib/dal.js - Verificación centralizada de sesión:

```
// lib/dal.js
import { cache } from 'react'
import { cookies } from 'next/headers'
import { redirect } from 'next/navigation'
```

```

export const verifySession = cache(async () => {
  const cookieStore = await cookies()
  const token = cookieStore.get('token')?.value

  if (!token) {
    return null
  }

  // Validar token (JWT, backend, etc.)
  // const payload = await verifyJWT(token)
  // return payload

  return { token, isAuthenticated: true }
})

export const getSessionOrRedirect = cache(async () => {
  const session = await verifySession()
  if (!session) {
    redirect('/')
  }
  return session
})

```

Proteger páginas directamente:

```

// app/dashboard/page.js
import { getSessionOrRedirect } from '@lib/dal'

export default async function DashboardPage() {
  // Verificación DIRECTA en la página
  const session = await getSessionOrRedirect()
  return <div>Dashboard protegido</div>
}

```

PRÁCTICA: Digitalización de albaranes

Descripción:

Desarrolla una aplicación web con Next.js que permita la gestión de albaranes (partes de horas o materiales) entre clientes y proveedores. La aplicación debe permitir a los usuarios lo siguiente:

Cread las siguientes páginas realizando las peticiones necesarias al back-end, ya provisto, en <https://bildy-rpmaya.koyeb.app/api-docs/> (puedes usar fetch o axios para las llamadas)

- **OnBoarding**
 - Crear una cuenta de usuario: POST /api/user/register

Create your ZoSale ID

First name

Last name

Email

Password

☐ By proceeding, you agree to the [Terms and Conditions](#)

Sign up with email

NOTA: Al crear la cuenta (o al logarte más adelante), recibirás un token JWT desde el back-end, este deberás almacenarlo en, por ejemplo, la caché local del navegador:

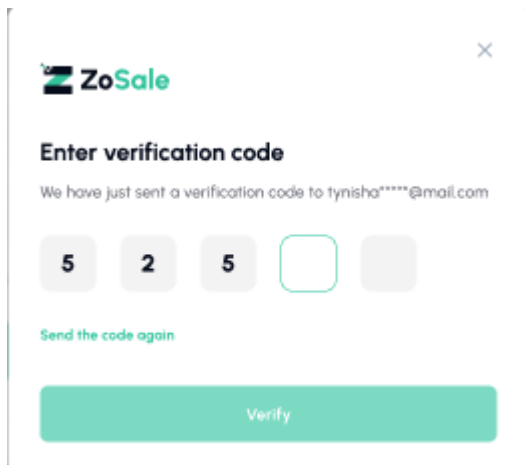
```
//Guardar datos en localStorage: localStorage.setItem('clave', 'valor')
```

```
localStorage.setItem('jwt', token)
```

```
//Leer datos de localStorage: const valor = localStorage('clave')
```

```
const token = localStorage('jwt')
```

- Validar el correo a través de un código recibido en el mismo:
PUT /api/user/validation

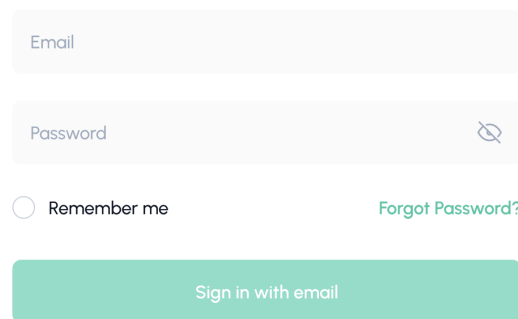
A screenshot of a web application window titled 'ZoSale' with a close button in the top right corner. The main heading is 'Enter verification code'. Below it, a message states: 'We have just sent a verification code to tynisha*****@mail.com'. There are five input fields for the code; the first three contain the digits '5', '2', and '5', while the last two are empty. Below the input fields is a link that says 'Send the code again'. At the bottom is a large green button labeled 'Verify'.

NOTA: Como la petición requiere token de sesión, guardado previamente en localStorage, enviaremos la petición con las cabeceras:

```
headers: { 'Content-Type': 'application/json', 'Authorization': `Bearer ${token}` }
```

- Login en la aplicación (devuelve un token de sesión): POST /api/user/login

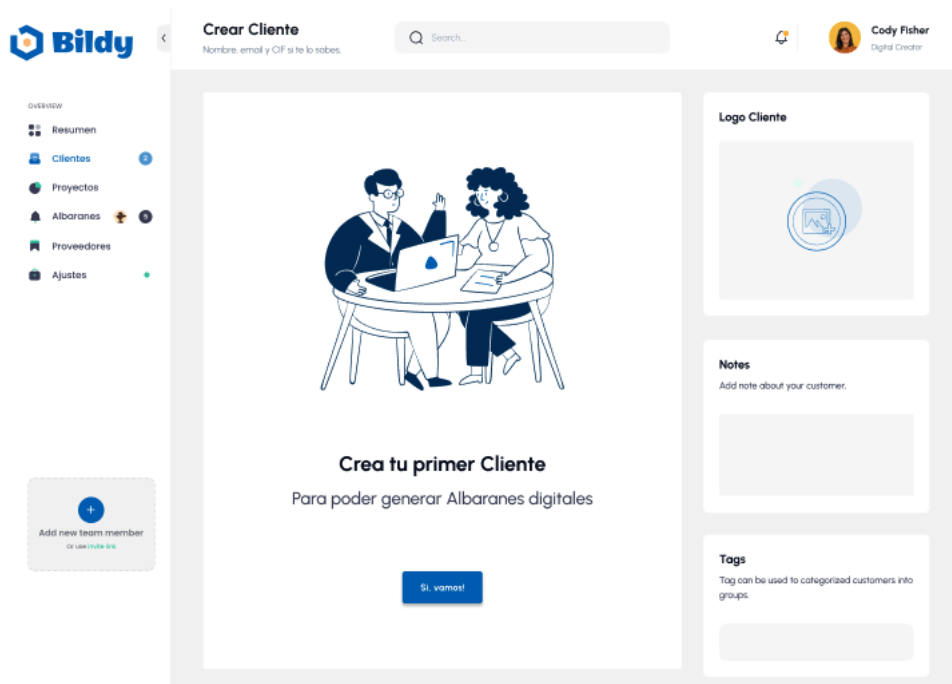
Login to your account

A login form with two input fields: 'Email' and 'Password'. The 'Password' field has a toggle icon on the right. Below the fields are two options: a radio button labeled 'Remember me' and a link labeled 'Forgot Password?'. At the bottom is a green button labeled 'Sign in with email'.

Don't have an account? [Get Started](#)

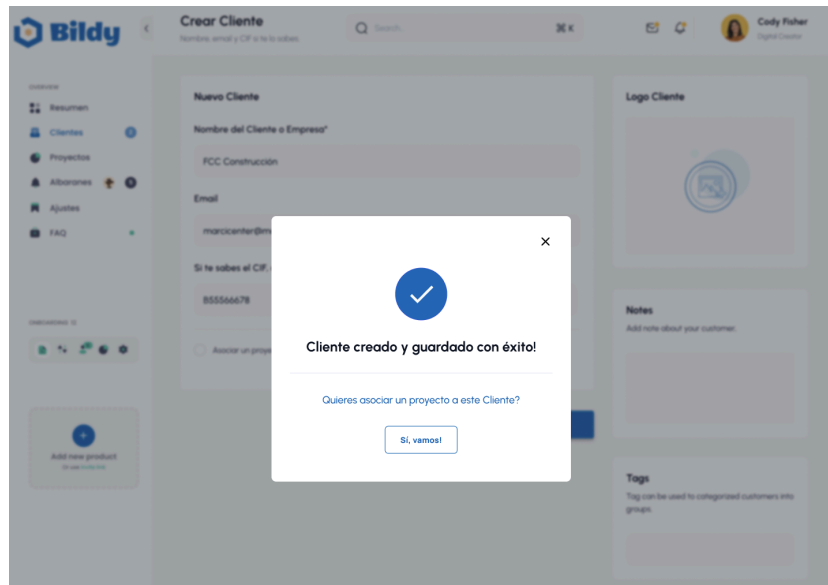
- **Crear un cliente:**

- Si todavía no hay ninguno, primero mostrar esta pantalla:

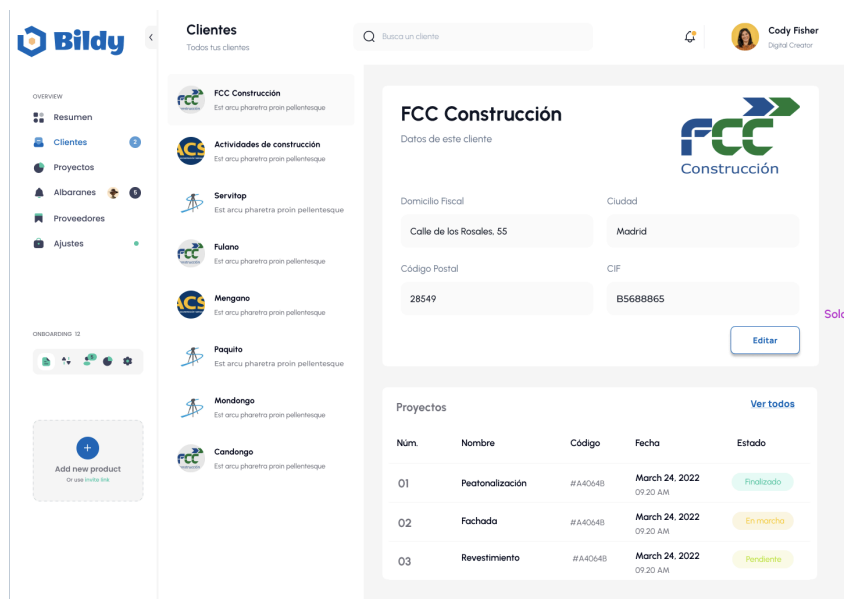


- Crear el cliente: POST /api/client

- Confirmar la creación en ventana emergente



- Si ya hay clientes, mostrarlos todos: GET /api/client
- y uno en concreto: GET /api/client/{id}



- **Proyectos:**

- Vista de proyectos en general de un cliente: GET /api/project

	Código	Fecha	Nombre	Cliente	Código Interno	Status
<input type="checkbox"/>	#12415346512	2/5/2022 06:24 AM	Marquezz	Samuel	-\$455	CANCELED
<input type="checkbox"/>	#12415346563	2/5/2020 06:24 AM	Marquezz	Cindy	+\$5,553	COMPLETED
<input type="checkbox"/>	#12415346563	2/5/2020 06:24 AM	Marquezz	Cindy	+\$5,553	COMPLETED
<input type="checkbox"/>	#124153465125	2/5/2020 06:24 AM	Marquezz	David	-\$12,768	PENDING
<input type="checkbox"/>	#124153465125	2/5/2020 06:24 AM	Marquezz	Samuel	-\$455	CANCELED
<input type="checkbox"/>	#124153465125	2/5/2020 06:24 AM	Marquezz	David	-\$12,768	PENDING
<input type="checkbox"/>	#12415346563	2/5/2020 06:24 AM	Marquezz	Luisana	+\$987	COMPLETED

- Crear un proyecto asociado a un cliente: POST /api/project
Además, si quiero actualizarlo: PUT /api/project/{projectId}

Nuevo Proyecto

Nombre proyecto

Datos de tu Cliente

Código

Dirección de proyecto

Email

Datos de tu Empresa

Código interno del proyecto

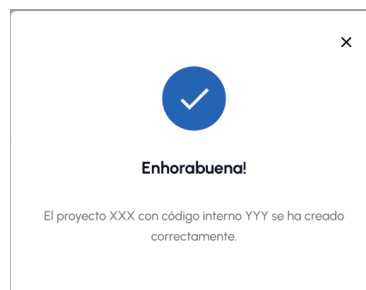
Cliente

Domicilio Fiscal
Calle de los Rosales, 55,
28054, Madrid

CIF
B55566678

Notas
Add note about your customer.

Descartar Guardar



- Vista de un proyecto específico: GET /api/project/one/{id}

Bıldı Proyectos Detalle de Proyecto

Peatonalización
Datos de este Proyecto

Código Interno: UCO23654-32 Ubicación: Calle de la Bomba, 68. Madrid. 28080. Editar

Albaranes Ver todos

Núm.	Nombre	Código	Fecha	Estado
01	Peatonalización	#A4064B	March 24, 2022 09:20 AM	Finalizado
02	Fachada	#A4064B	March 24, 2022 09:20 AM	En marcha
03	Revestimiento	#A4064B	March 24, 2022 09:20 AM	Pendiente
01	Peatonalización	#A4064B	March 24, 2022 09:20 AM	Finalizado
02	Fachada	#A4064B	March 24, 2022 09:20 AM	En marcha
03	Revestimiento	#A4064B	March 24, 2022 09:20 AM	Pendiente

Cliente

 Domicilio Fiscal: Calle de los Rosales, 55. 28054. Madrid
 CIF: B55566678
 Notas: Add note about your customer.

- **Albarán:**
 - Crear albarán de un proyecto en concreto: POST /api/deliverynote

Bıldı Proyectos Detalle de Proyecto

Peatonalización
Datos de este Proyecto

Código Interno: UCO23654-32 Ubicación: Calle de la Bomba, 68. Madrid. 28080.

Borrador Albarán
Salesline and your customers will use this information to contact you.

Store name: Makostore Industry: Clothing

Store currency: US Dollars (USD) Timezone: US & Canada Weight unit: Kilogram (kg)

Store Address
This address will appear on your invoices.

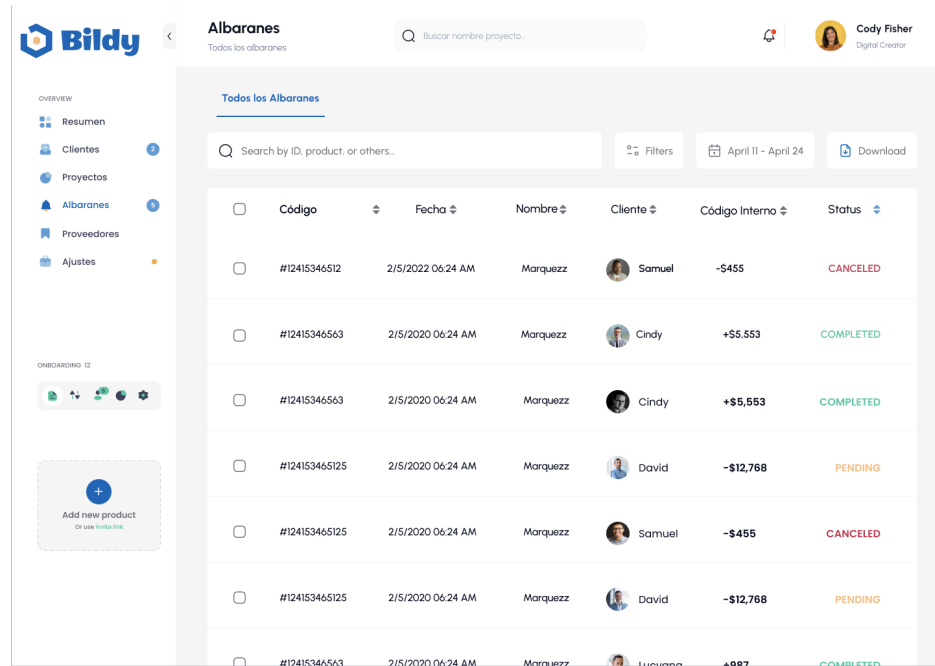
Legal name of company: Makostore

Apartment, suite, or etc.: Address: 4517 Washington Ave. Manchester

Cliente

 Domicilio Fiscal: Calle de los Rosales, 55. 28054. Madrid
 CIF: B55566678
 Notas: Add note about your customer.

- Listar albaranes: GET /api/deliverynote



- Descargar albarán en pdf: GET /api/deliverynote/pdf/{id}

Objetivos:

- Practicar todo lo aprendido realizando una aplicación con Next.js profesional..
- Implementar los conceptos básicos de React:
 - Creación de componentes
 - Props
 - useState hook
 - Manejo de eventos
 - Comunicación hijo-padre
 - Condicionales y listas
 - Aplicación de estilos
 - Asignación de estilos y clases dinámicos
 - Styled components y CSS modules
- Integrar librerías:
 - Formik o React Hook Form (opcional)
 - Peticiones HTTP (axios o fetch) a un API real.

Guía de resolución:

1. Planificación:

- Diseñar la estructura de la aplicación:
 - Páginas/rutas principales
 - Componentes
 - Flujo de navegación
- Definir el estado y las props de los componentes.
- Elegir las librerías a utilizar (opcional).

2. Implementación:

2.1 Routing:

- Configurar las rutas de la aplicación (árbol de directorios) con NextJS.
- Crear page.js y sus componentes para cada página.

2.2 Componentes:

- Desarrollar los componentes de la interfaz cuando aplique:
 - Header
 - Footer
 - Barra lateral
 - Menú de navegación
 - Página de inicio
 - Página de clientes
 - Página de proyectos
 - Página de albaranes
 - Formularios
 - Descarga de albaranes

2.3 Funcionalidades:

- Implementar la lógica de cada componente:
 - Filtrado de clientes, proyectos y albaranes
 - Manejo de formularios
 - Envío de datos (POST y PUT)

2.4 Estilos:

- Aplicar estilos a los componentes con CSS.
- Usar tailwind (opcional).

3. Integración de librerías (opcional):

- Implementar Formik o React Hook Form para los formularios (opcional).
- Realizar peticiones HTTP para obtener datos (axios o fetch).

4. Pruebas y depuración:

- Probar la aplicación en diferentes navegadores y dispositivos.
- Corregir errores y optimizar el rendimiento.

Navegación:

- Usar el componente Link para navegar a otras páginas:

```
<Link to="/clients">Clientes</Link>
```

- Usar useRouter para navegar ejecutando acciones cuando aplique.

Consideraciones adicionales:

- Redireccionar a los usuarios a la página 404 creada por nosotros si intentan acceder a una ruta inexistente.
- Mostrar mensajes de error personalizados si hay problemas.

CUESTIONARIO

1. ¿Qué es Next.js?

- a) Un framework de Node.js para construir APIs.
- b) Un framework basado en React para la renderización del lado del servidor.
- c) Una biblioteca de estilos CSS.
- d) Un lenguaje de programación.

Respuesta correcta: b) Un framework basado en React para la renderización del lado del servidor.

2. ¿Cómo se inicia un nuevo proyecto de Next.js?

- a) `npm init next-app`
- b) `npx create-next-app my-next-app`
- c) `npm start next-app`
- d) `npx init-next-app my-next-app`

Respuesta correcta: b) `npx create-next-app my-next-app`

3. ¿En qué directorio se colocan las páginas de un proyecto Next.js?

- a) `/src`
- b) `/public`
- c) `/src/app`
- d) `/src/components`

Respuesta correcta: c) `/src/app`

4. ¿Cuál de las siguientes es una característica clave de Next.js?

- a) Soporte para CSS-in-JS
- b) Generación de páginas estáticas en tiempo de compilación
- c) Renderización del lado del servidor
- d) Todas las anteriores son correctas

Respuesta correcta: d) Todas las anteriores son correctas

5. ¿Cuál es el propósito del archivo `next.config.js` en un proyecto Next.js?

- a) Configurar rutas y componentes
- b) Configurar el comportamiento y las opciones de compilación de Next.js
- c) Configurar los estilos globales de la aplicación
- d) Configurar los scripts del lado del servidor

Respuesta correcta: b) Configurar el comportamiento y las opciones de compilación de Next.js

6. ¿Cómo se define una ruta dinámica en Next.js?

- a) Creando un archivo con un nombre estático en la carpeta `/src/app`
- b) Creando un directorio con corchetes, como `[id]/page.js`, en la carpeta `/src/app`
- c) Configurando el archivo `next.config.js`
- d) Usando el hook `useRouter`

Respuesta correcta: a) Creando un directorio con corchetes, como [id].page.js, en la carpeta /src/app

7. ¿Cuál es el propósito del archivo `src/app/layout.js` en una aplicación Next.js?

- a) Definir rutas dinámicas
- b) Configurar el comportamiento de los componentes
- c) Proveer una estructura de diseño común para todas las páginas
- d) Gestionar las variables de entorno

Respuesta correcta: c) Proveer una estructura de diseño común para todas las páginas

8. ¿Cómo se integra Tailwind CSS en un proyecto Next.js?

- a) Añadiendo el archivo CSS de Tailwind directamente en cada componente
- b) Instalando Tailwind CSS al crear el proyecto NextJS. Configurándose en los archivos `src/app/globals.css`, `tailwind.config.js` y `postcss.config.js`
- c) Usando la etiqueta `<style>` en el componente principal
- d) Incluyendo Tailwind CSS en el archivo `package.json`

Respuesta correcta: b) Instalando Tailwind CSS al crear el proyecto NextJS

9. ¿Cómo se navega entre páginas en una aplicación Next.js?

- a) Usando etiquetas `<a>` estándar de HTML
- b) Usando el componente `Link` de `next/link`
- c) Usando el hook `useNavigate` de React Router
- d) Usando el método `navigateTo` de Next.js

Respuesta correcta: b) Usando el componente Link de next/link

10. ¿Para qué se utiliza el hook `useRouter` del módulo `next/navigation` en Next.js?

- a) Crear rutas del lado del servidor (back-end)
- b) Navegar entre diferentes páginas del cliente (front-end)
- c) Obtener datos estáticos
- d) Configurar estilos CSS

Respuesta correcta: b) Navegar entre diferentes páginas del cliente (front-end)

PARA SABER MÁS

[The React Framework for the Web](#)