

Guía de repaso de P2

El objetivo de este guión es hacer una estructura básica de lo que se esperaba en la P2 de esta asignatura. Repasaremos conceptos tales como:

- Crear proyecto NextJS
- Uso básico de css con los CSS Modules
- Uso de Layout para casos complicados
- Peticiones http gestionando el token de autenticación vía JWT

Se recomienda ENCARECIDAMENTE leer detenidamente esta guía mientras se va desarrollando. Si simplemente copiáis y pegáis comandos y código, no os va a servir para nada.

La idea de este guion es hacerlo después de haber repasado bien el uso de HTML, CSS y JS. Así mismo, ten a mano la documentación de NextJS por si quieres profundizar o repasar algunos conceptos como la estructura de carpetas.

Creación de un proyecto en NextJS para empezar a trabajar

Para crear un proyecto en NextJS, basta con ejecutar el siguiente comando en el directorio que más nos interese: **npx create-next-app@latest**

Desmenuzemos que hace cada parte:

- **npx** -> nos permite descargar y ejecutar una dependencia sin instalarla en nuestro equipo. Es temporal y se borrará.
- **create-next-app** -> Crea una aplicación de NextJS
- **@latest** -> el @ nos permite definir una versión de la aplicación a usar. Latest significa que vamos a usar la última.

Con lo visto, este comando nos va a permitir crear una aplicación NextJS utilizando la última versión.

El comando nos empezará a preguntar por las configuraciones del proyecto. Crealo así:

```
npx create-next-app@latest
✓ What is your project named? ... repaso_p2
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... 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 for `next dev`? ... No / Yes
✓ Would you like to customize the import alias ('@/*' by default)? ... No / Yes
✓ What import alias would you like configured? ... /*

Creating a new Next.js app in /home/cristian/Workspace/Comillas/DAS/sandbox/guia_repasso_p2/repasso_p2.
```

Básicamente es darle a todo a siguiente excepto por el nombre del proyecto y por el uso de alias.

Si quieres profundizar más en este paso, en la presentación de NextJS de la asignatura tienes una explicación.

Hay que mencionar aquí que el alias lo que nos permite es importar componentes siempre desde el mismo punto y con la misma sintaxis. Más cómodo que gestionar paths relativos. (lo veremos en uso más adelante).

Tras este paso, tendrás un proyecto creado en la carpeta con el nombre que hayas puesto:

```
~/Workspace/Comillas/DAS/sandbox/guia_repasso_p2 (0.013s)
ls
repaso_p2
```

Entra dentro de la carpeta y ábrela con tu editor de confianza.

El siguiente paso será ejecutar el proyecto para ser capaz de verlo en local (recuerda estar dentro de la carpeta):

npm run dev

Deberás ver algo tal como esto:

```
○ → repaso_p2 git:(main) npm run dev
> repaso_p2@0.1.0 dev
> next dev --turbopack

✖ next.js v15.2.4 is not yet supported in the Community edition of Console Ninja.
We are working hard on it for you https://tinyurl.com/3h9mtwra.

Estimated release dates:
- Community users: around 24th May, 2025 (subject to team availability)
- PRO users: priority access is available now

✖ next.js v15.2.4 is not yet supported in the Community edition of Console Ninja.
We are working hard on it for you https://tinyurl.com/3h9mtwra.

Estimated release dates:
- Community users: around 24th May, 2025 (subject to team availability)
- PRO users: priority access is available now

  ▲ Next.js 15.2.4 (Turbopack)
  - Local:      http://localhost:3000
  - Network:    http://192.168.1.209:3000

✓ Starting...
✓ Ready in 552ms
```

Vamos a desmenuzar el comando anterior:

- **npm** -> CLI de Node Package Management
- **run** -> Ejecuta un script definido en el fichero package.json
- **dev** -> Script del fichero package.json.

Si abrimos el fichero package.json, veremos un apartado `script` en el que se definen todos los comandos que se pueden ejecutar con `npm run`:

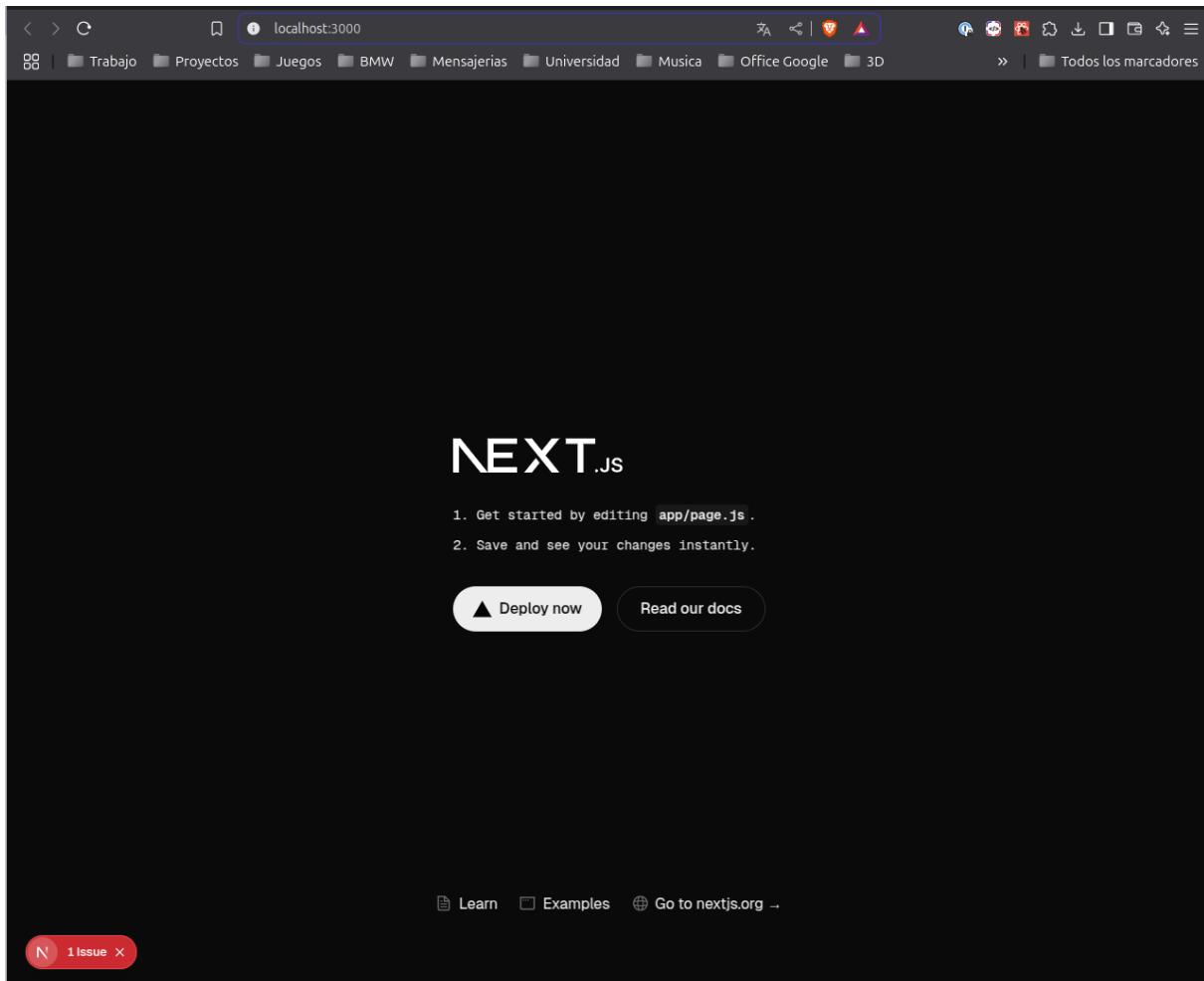
```
package.json > ...
You, 8 minutes ago | 1 author (You)
{
  "name": "repaso_p2",    "repaso": Unknown word.
  "version": "0.1.0",
  "private": true,
  ▷ Debug
  "scripts": {
    "dev": "next dev --turbopack",    "turbopack": Unknown word.
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "react": "^19.0.0",    -> 19.1.0
    "react-dom": "^19.0.0",    -> 19.1.0
    "next": "15.2.4"
  },
  "devDependencies": {
    "eslint": "^9",    -> 9.24.0
    "eslint-config-next": "15.2.4",
    "@eslint/eslintrc": "^3"    -> 3.3.1
  }
}
```

Si nos fijamos, tenemos 4 comandos:

- **dev** -> ejecuta next dev, que es el entorno de desarrollo de Nextjs
- **build** -> Compila la app para desplegar. Es lo que usa Vercel cuando se despliega
- **start** -> Ejecuta el compilado
- **lint** -> checkea estilo y errores.

En este punto, si nos fijamos en el resultado de npm run dev, veremos un enlace que suele ser <http://localhost:3000>. Ahí veremos en tiempo real nuestra app y los cambios que iremos haciendo.

De momento tendréis algo como esto:



El siguiente paso que vamos a hacer es limpiar todo lo que NextJS nos crea de primeras.

Dentro del directorio public, borra todo el contenido dejando la carpeta vacía:



Recordemos que lo que hay dentro de la carpeta de public, se moverá directamente al servidor y será accesible en <url>/<mi-recurso>.

Es decir, si tenemos un fichero img.png, este será accesible desde <http://localhost:3000/fichero.png> o desde <https://my-app.vercel.com/fichero.png>.

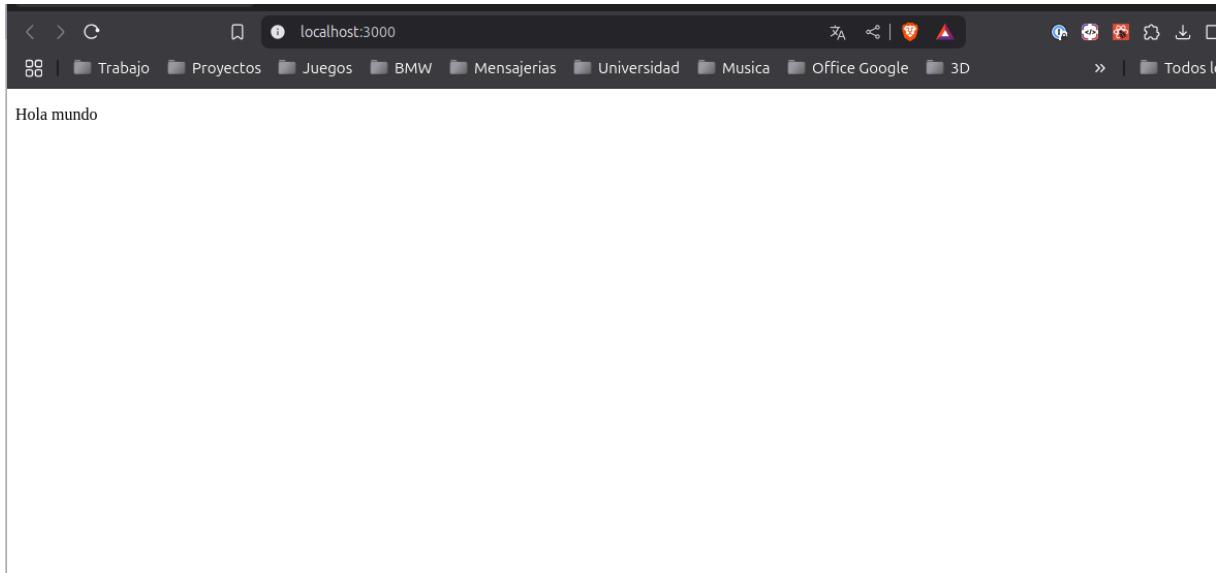
Dentro de la carpeta app,

- Borra todo el contenido del fichero global.css
- Borra todo el contenido del fichero page.module.css
- Deja el fichero page.js como se muestra:



```
JS page.js M X
app > JS page.js > ...
You, 45 seconds ago | 1 author (You)
1  export default function Index() {
2    return <p>Hola mundo</p>;
3  }
4
```

Deberías ver el contenido en tu web tal que así:



Este es el momento adecuado para crear un repositorio en github y subir el proyecto y empezar a trabajar desde git.

Repaso de la estructura de NextJS

Los proyectos de NextJS tienen 3 carpetas importantes:

- **Public:** como se explicó antes, lo que metamos aquí se publica en nuestro servidor directamente. Muy útil para compartir recursos que no van a cambiar como imágenes. Tened en cuenta que estos recursos son públicos. No se protegen.
- **App:** Aquí viven nuestras páginas web. Todas las carpetas que creemos aquí se transforman en una URL de nuestra app. Por tanto, SOLO debe haber carpetas que sean páginas. No es el sitio para crear componentes ni utilidades. SOLO pages.
- **Components:** Esta carpeta tenemos que crearla. No viene por defecto. Aquí vivirán todos aquellos componentes que vayamos a reusar. Regla para saber si se reusa un componente -> El componente se usa en dos o más páginas o componentes.

Crea una carpeta **components** en tu proyecto. A la altura de app:



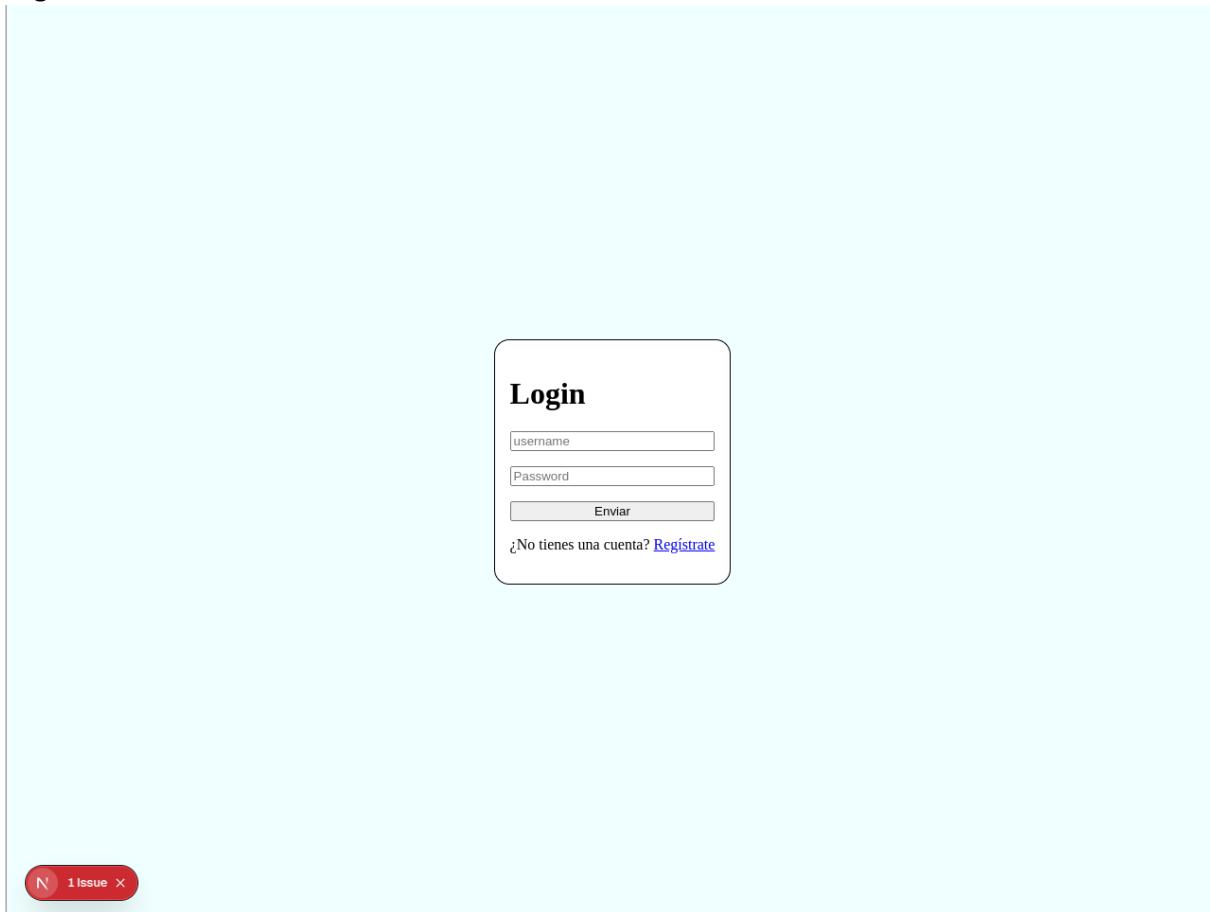
¿Qué vamos a hacer?

Vamos a realizar una página web de subastas que va a compartir una estructura inicial con un header y footer **EXCEPTO** en la pantalla de login y registro.

Este caso no es el que se os pedía en la práctica, pero varios lo habéis implementado y añade un ligero toque de dificultad, así que lo vamos a hacer así. Recordad, que, si hacéis un header y un footer en todas vuestras páginas, os queda más sencillo ya que sólo tenéis que usar un layout.

La idea es diseñar algo como esto:

Login:



The screenshot shows a light blue background with a central login form. The form has a rounded rectangle border and contains the word "Login" at the top. Below it are three input fields: "username", "Password", and a "Enviar" button. At the bottom of the form is a link "¿No tienes una cuenta? [Regístrate](#)". In the bottom-left corner of the light blue area, there is a small red circular badge with white text that says "1 issue".

Auctions:



Recordad que el que sea más o menos bonita, no puntúa. No os compliquéis la vida.

Normas de estilos de la web:

- El header y el footer tiene que estar pegados al top y al bottom de la página respectivamente.

Implementación del login y registro

Como hemos visto en el apartado anterior, el login y el registro van a tener un layout diferente al resto. Para ello la idea va a ser NO USAR el componente layout.js porque este se hereda, sino crear un componente que envuelva a las páginas de login y registro con el layout que queremos.

Básicamente vamos a saltarnos las opciones de layout de NextJS en este punto. Pero primero, vamos a preparar el terreno.

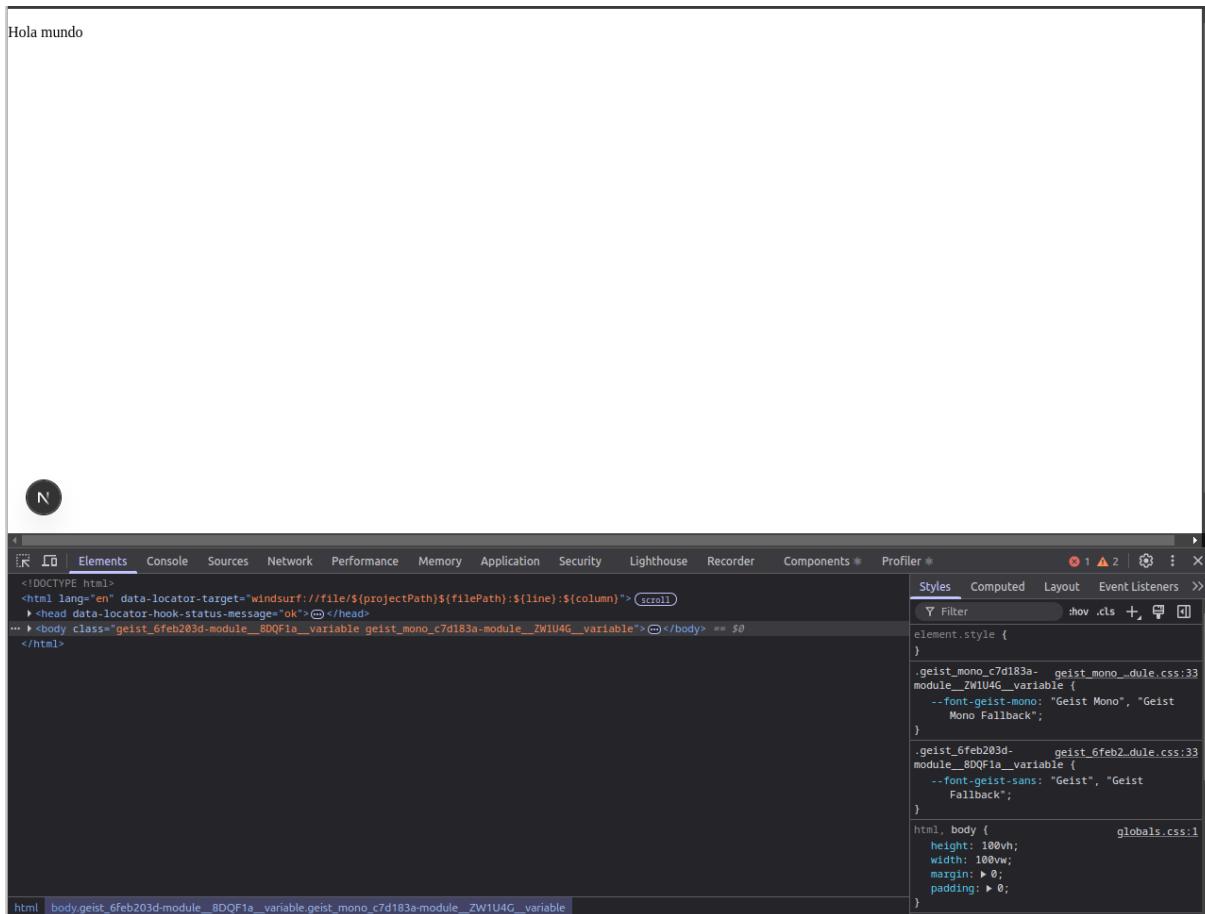
Ve al fichero globals.css y déjalo tal y como se muestra:



```
css globals.css M X
app > css globals.css > 📁 html > Explain Refactor Add Doc
You, 7 seconds ago | 1 author (You)
1 html,
2 body {
3   height: 100vh;
4   width: 100vw;
5   margin: 0;
6   padding: 0;
7 }
```

Con esto nos aseguramos de que nuestro html y nuestro body ocupen el 100% del viewport (parte visible de nuestra web)

Podemos comprobarlo yendo al **navegador > click derecho > Inspeccionar**:



En esta pantalla puedes ver los elementos HTML así como los estilos aplicados.

Verás que el **body** tiene, a parte del css que acabamos de aplicar, otro CSS que define unas fuentes. Esto se hace en el fichero `app/layout.js`. En este fichero verás, a parte de las definiciones de las fuentes, que se importa el `globals.css`.

```

  ts globals.css M      JS layout.js X
app > JS layout.js > ...
You, 44 minutes ago | 1 author (You)
1 import { Geist, Geist_Mono } from "next/font/google"; 665 (gzipped: 384)
2 import "./globals.css";
3
4 const geistSans = Geist({
5   variable: "--font-geist-sans",
6   subsets: ["latin"],
7 });
8
9 const geistMono = Geist_Mono({
10   variable: "--font-geist-mono",
11   subsets: ["latin"],
12 });
13
14 export const metadata = {
15   title: "Create Next App",
16   description: "Generated by create next app",
17 };
18
19 export default function RootLayout({ children }) {
20   return (
21     <html lang="en">
22       <body className={`${geistSans.variable} ${geistMono.variable}`}>
23         | {children}
24       </body>
25     </html>
26   );
27 }
28

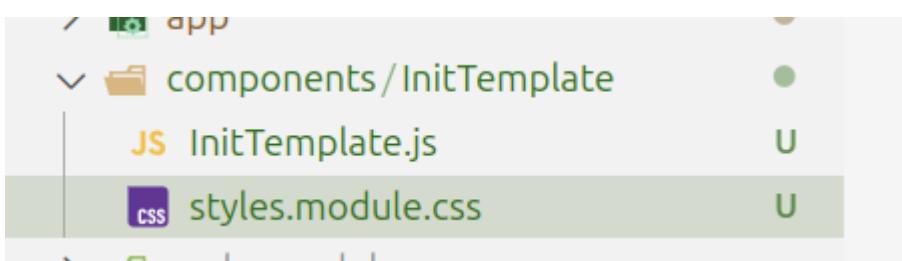
```

Lo vamos a dejar tal y como está.

¡¡NOTA: globals.css es el único fichero en el que debemos usar etiquetas HTML como selectores (html, body) y que no es un fichero module.css!!

Vamos ahora a crear, dentro de la carpeta **components**, un componente llamado **InitTemplate** que será el que nos defina el cómo se va a ver la pantalla de login y registro. Dentro de esa carpeta, crea lo siguiente:

- un fichero que se llame **InitTemplate.js** (ficheros de JS que tengan un componente de React se escriben en mayúscula): Aquí definiremos nuestro componente
- Un fichero llamado **styles.module.css**: Aquí definiremos los estilos de nuestro componente



La idea de este componente es que nos posicione todo el contenido en el centro y dentro de un componente Card que haremos más adelante.

Pon este código dentro de InitTemplate.js:

```
components > InitTemplate > JS InitTemplate.js > ...
1 import React from "react"; 7.9k (gzipped: 3.1k)
2 import styles from "./styles.module.css";
3
4 const InitTemplate = ({ children }) => {
5   return <div className={styles.container}>{children}</div>;
6 };
7
8 export default InitTemplate;
9
```

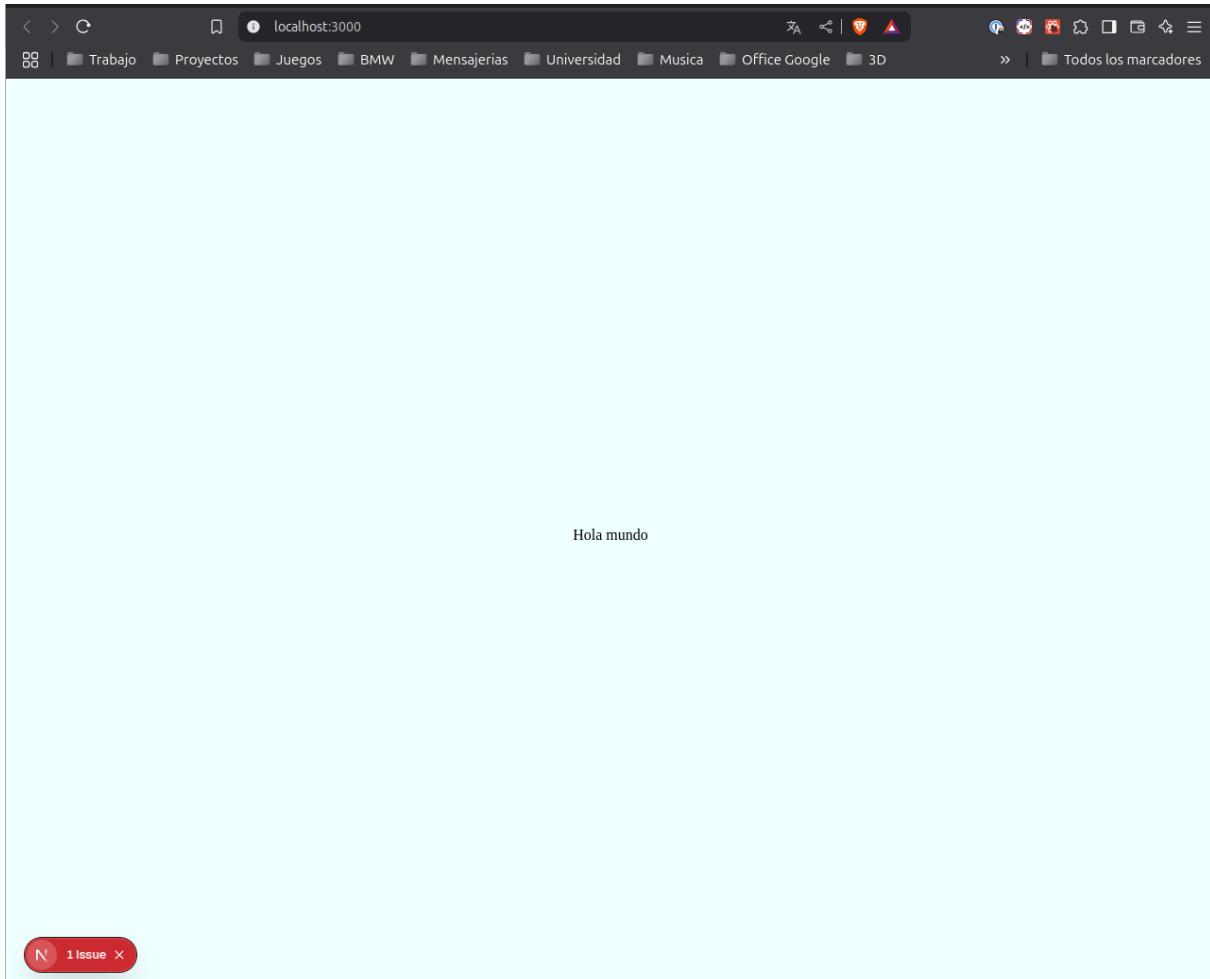
Pon este código dentro de styles.module.css

```
components > InitTemplate > css styles.module.css > .container > Explain Refactor Add Docstring
1 .container {
2   width: 100%;
3   height: 100%;
4
5   background-color: #azure;
6
7   display: flex;
8   justify-content: center;
9   align-items: center;
10 }
```

Ahora, vete a *app/page.js* y déjalo tal que así:

```
app > JS page.js > ...
You, 1 second ago | 1 author (You)
1 import InitTemplate from "@/components/InitTemplate/InitTemplate";
2
3 export default function Index() {
4   return (
5     <InitTemplate>
6       <p>Hola mundo</p>    "mundo": Unknown word.
7     </InitTemplate>
8   );
9 }
10
```

Deberías ver que tú web ha cambiado y ahora se muestra así:



¿Qué hemos hecho aquí?

1 - hemos definido un componente llamado `InitTemplate` que lo que hace es centrar cualquier **children** que reciba. Recuerda, que en componentes React, la prop **children** es cualquier otro componente o HTML que esté dentro:

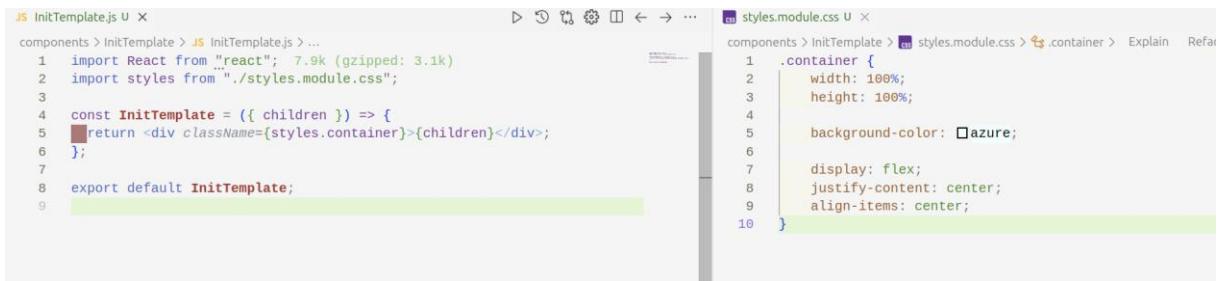
```

<InitTemplate>
  <p>Hola mundo</p>
</InitTemplate>
  
```

*El componente `<p>hola mundo</p>` es el **children** de `InitTemplate`*

2 – Hemos creado un componente (`InitTemplate`) que se va a reusar en varias págs. Por eso vive dentro de la carpeta `components`. Además, hemos creado un fichero de estilos para ese componente de tipo CSS Modules. Recuerda, ahora ya no hacemos ficheros css normales. Hacemos ficheros CSS Modules **POR COMPONENTE** que viven junto a él.

3- Hemos repasado como centrar un elemento con flex:



```

JS InitTemplate.js U X
components > InitTemplate > JS InitTemplate.js > ...
1 import React from "react"; 7.9k (gzipped: 3.1k)
2 import styles from "./styles.module.css";
3
4 const InitTemplate = ({ children }) => {
5   return <div className={styles.container}>{children}</div>;
6 };
7
8 export default InitTemplate;
9

css styles.module.css U X
components > InitTemplate > styles.module.css > .container > Explain Refactor
1 .container {
2   width: 100%;
3   height: 100%;
4
5   background-color: #azure;
6
7   display: flex;
8   justify-content: center;
9   align-items: center;
10 }
  
```

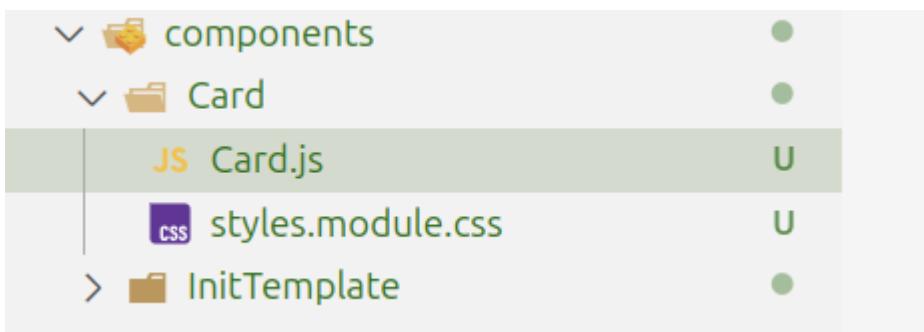
- HTML y BODY ocupan el 100% gracias a lo definido en global.css
- Hemos creado una clase css llamada container aplicado al div que envuelve {children}
- Con display flex y sus propiedades justify-content y align-items, centramos todo.

(Recuerda: Grid nos da estructura, Flex nos ayuda a posicionar dentro de Grid. En este caso, podemos omitir Grid dado que solo queremos centrar todo en una pantalla. Puedes verlo como que Grid aquí es una fila y una columna)

Vamos ahora a crear un componente Card (Tarjeta) para que quede más bonito:

1. Crea una nueva carpeta dentro de componentes y llámala Card.

 - a. Crea dentro un fichero Card.js
 - b. Crea dentro un fichero styles.module.css




```

JS Card.js U X
components > Card > JS Card.js > ...
1 import React From "react"; 7.9k (gzipped: 3.1k)
2 import styles from "./styles.module.css";
3
4 const Card = ({ children }) => {
5   return <div className={styles.container}>{children}</div>;
6 };
7
8 export default Card;
9

css styles.module.css U X
components > Card > styles.module.css > .container > Explain Refactor Add Docstring
1 .container {
2   display: flex;
3   flex-direction: column;
4
5   padding: 1rem;
6   background-color: white;
7
8   border: 1px solid black;
9   border-radius: 1rem;
10 }
  
```

Como verás, es más o menos lo mismo que hemos hecho con InitTemplate, pero simplemente creando un componente con bordes redondeados y, padding, y un bg blanco.

Ahora, vamos a meterlo dentro de InitTemplate.js:

JS InitTemplate.js U X

components > InitTemplate > JS InitTemplate.js > ...

```

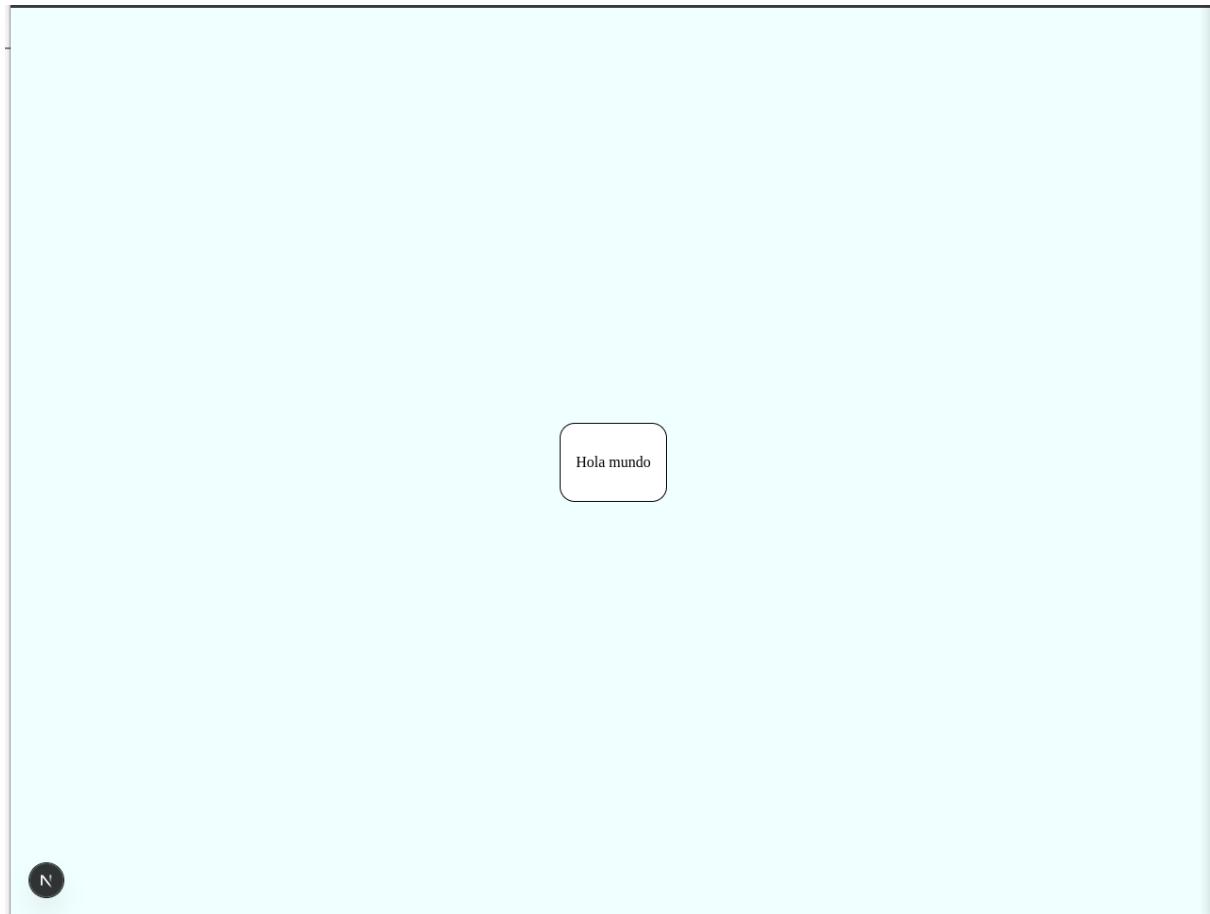
1 import React from "react"; 7.9k (gzipped: 3.1k)
2 import styles from "./styles.module.css";
3 import Card from "@/components/Card/Card";
4
5 const InitTemplate = ({ children }) => {
6   return (
7     <div className={styles.container}>
8       <Card>{children}</Card>
9     </div>
10 );
11
12
13 export default InitTemplate;
14

```

Fíjate que en la linea 3, se ha importado usando '@/components'. Esta es la ventaja de usar un alias. Si no lo tuviéramos, la importación sería algo del estilo a:

Import Card from "../Card/";

Ahora deberías ver tu web tal que así:



Ahora, vamos a desarrollar el formulario de login.

Vuelve al fichero app/page.js y déjalo tal que así:

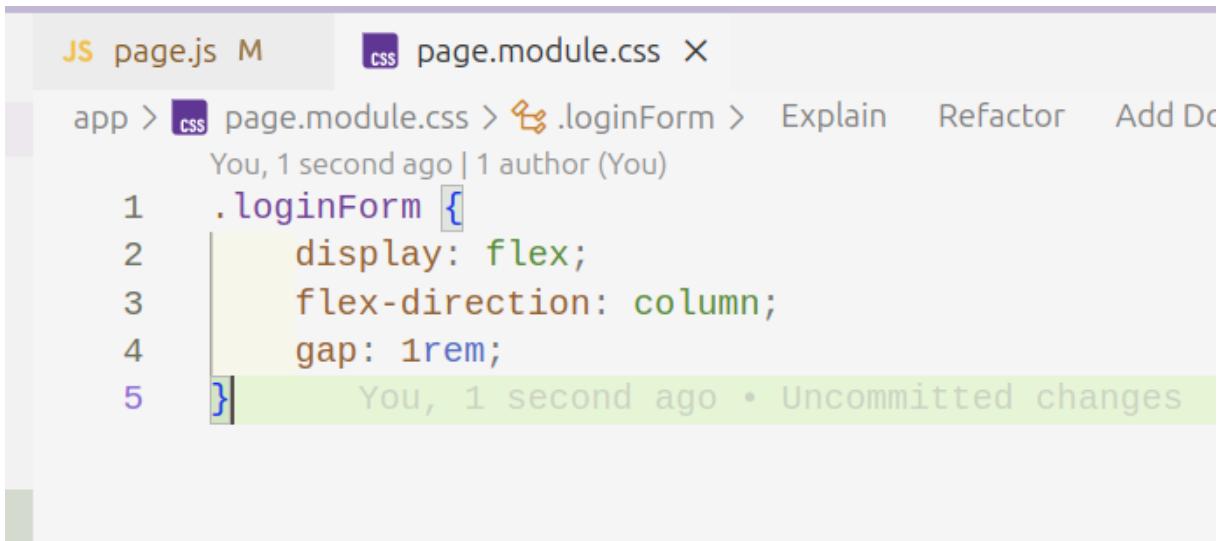
```
YOU, 8 seconds ago | 1 author (YOU)
import InitTemplate from "@/components/InitTemplate/InitTemplate";
import Link from "next/link"; 40k (gzipped: 11.8k)

export default function Index() {
  return (
    <InitTemplate>
      <h1>Bienvenid@! Inica sesión</h1>    "Bienvenid": Unknown word.
      <form>
        <input type="text" />
        <input type="password" />
        <button type="submit">Login</button>
      </form>
      <p>
        No tienes cuenta? <Link href="/register">Regístrate</Link>    "tienes": Unknown word.
      </p>
    </InitTemplate>
  );
}
```

Deberías ver esto en tu web:



Vamos a centrar el formulario. Abre tu fichero page.module.css que está dentro de app y déjalo tal que así:



```

JS page.js M      CSS page.module.css X
app > CSS page.module.css > .loginForm > Explain Refactor Add Doc
You, 1 second ago | 1 author (You)
1 .loginForm {
2   display: flex;
3   flex-direction: column;
4   gap: 1rem;
5 }
You, 1 second ago • Uncommitted changes

```

Vamos a aplicar esta clase a nuestro form:



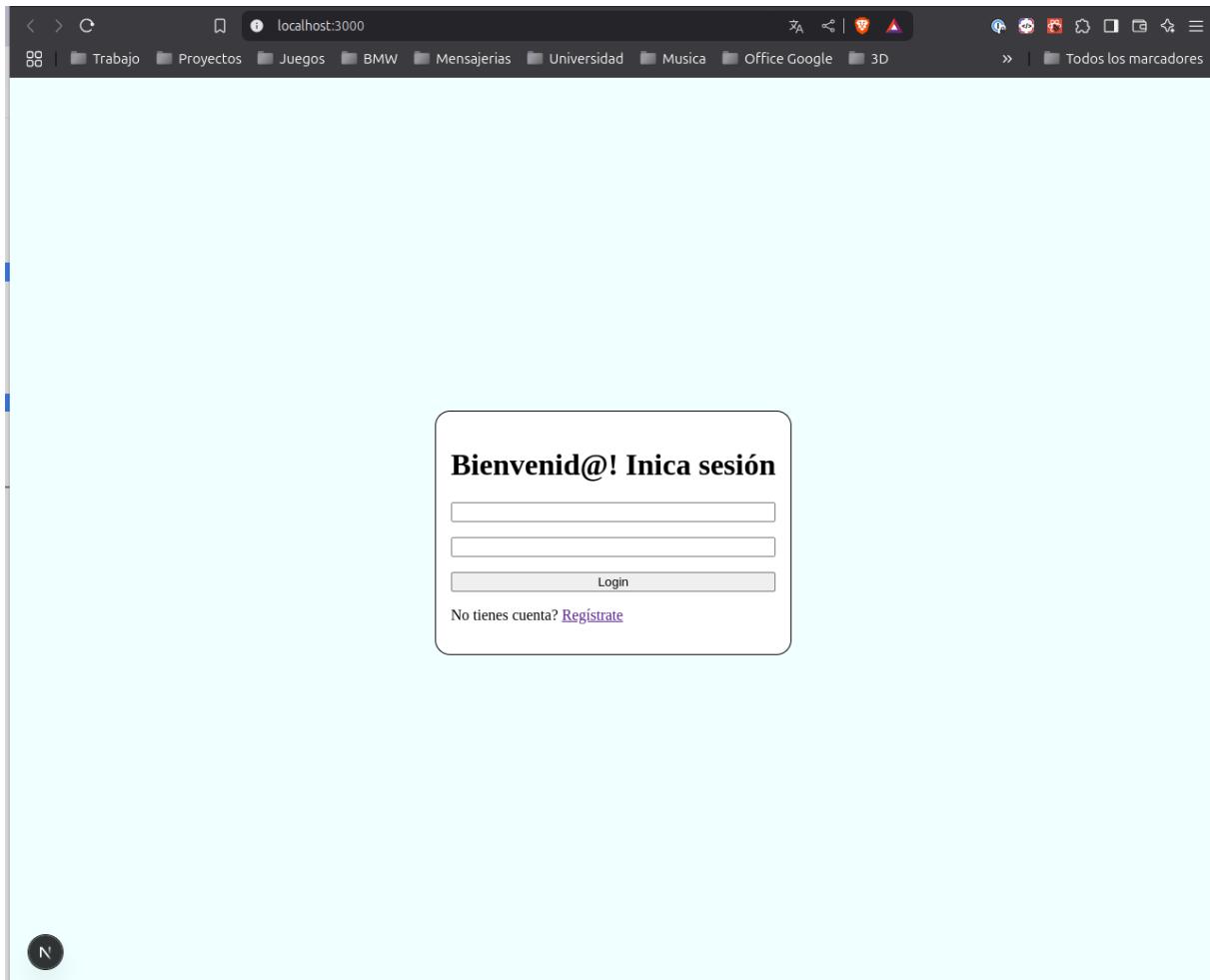
```

You, 1 second ago | 1 author (You)
1 import InitTemplate from "@/components/InitTemplate/InitTemplate";
2 import Link from "next/link"; 40k (gzipped: 11.8k)
3 import styles from "./page.module.css";
4
5 export default function Index() {
6   return (
7     <InitTemplate>
8       <h1>Bienvenid@! Inica sesión</h1>    "Bienvenid": Unknown word.
9       <form className={styles.loginForm}>
0         <input type="text" name="username" />
1         <input type="password" name="password" />
2         <button type="submit">Login</button>
3       </form>
4       <p>
5         No tienes cuenta? <Link href="/register">Regístrate</Link>    "tienes": Unk
6       </p>
7     </InitTemplate>
8   );
9 }
Ctrl+I for Command, Ctrl+L for Cascade

```

Fíjate que hemos importado el fichero page.module.css y lo hemos llamado styles. Hemos usado la clase en el form.

Ahora deberías verlo así:

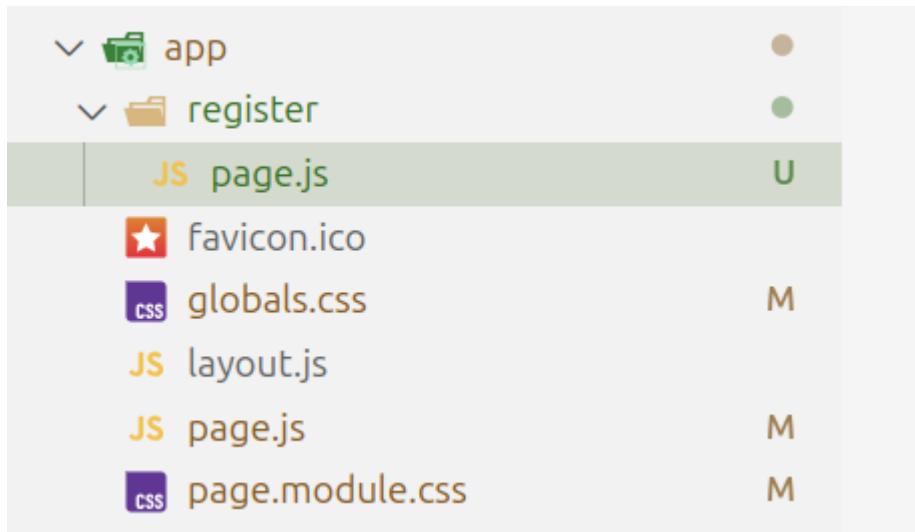


Revisa el código nuevo de page.js. Fijate en que no se ha usado el elemento `<a>` para la redirección, sino el elemento `<Link>` que proporciona NextJS.

Vamos ahora por el siguiente paso. Vamos a crear la pantalla de Registro. Verás como ahora que hemos hecho todo este trabajo, sale super sencillo.

Crea una carpeta dentro de app llamada **register** y, dentro un fichero llamado **page.js**. Observa que la carpeta se llama **register** para coincidir con la redirección del enlace que hemos puesto.

Crea también un fichero **page.module.css** para añadir estilo.



Dentro del fichero page.module.css, crea lo siguiente:

```
app > register > page.module.css > .loginForm
1   .loginForm {
2     display: flex;
3     flex-direction: column;
4     gap: 1rem;
5 }
```

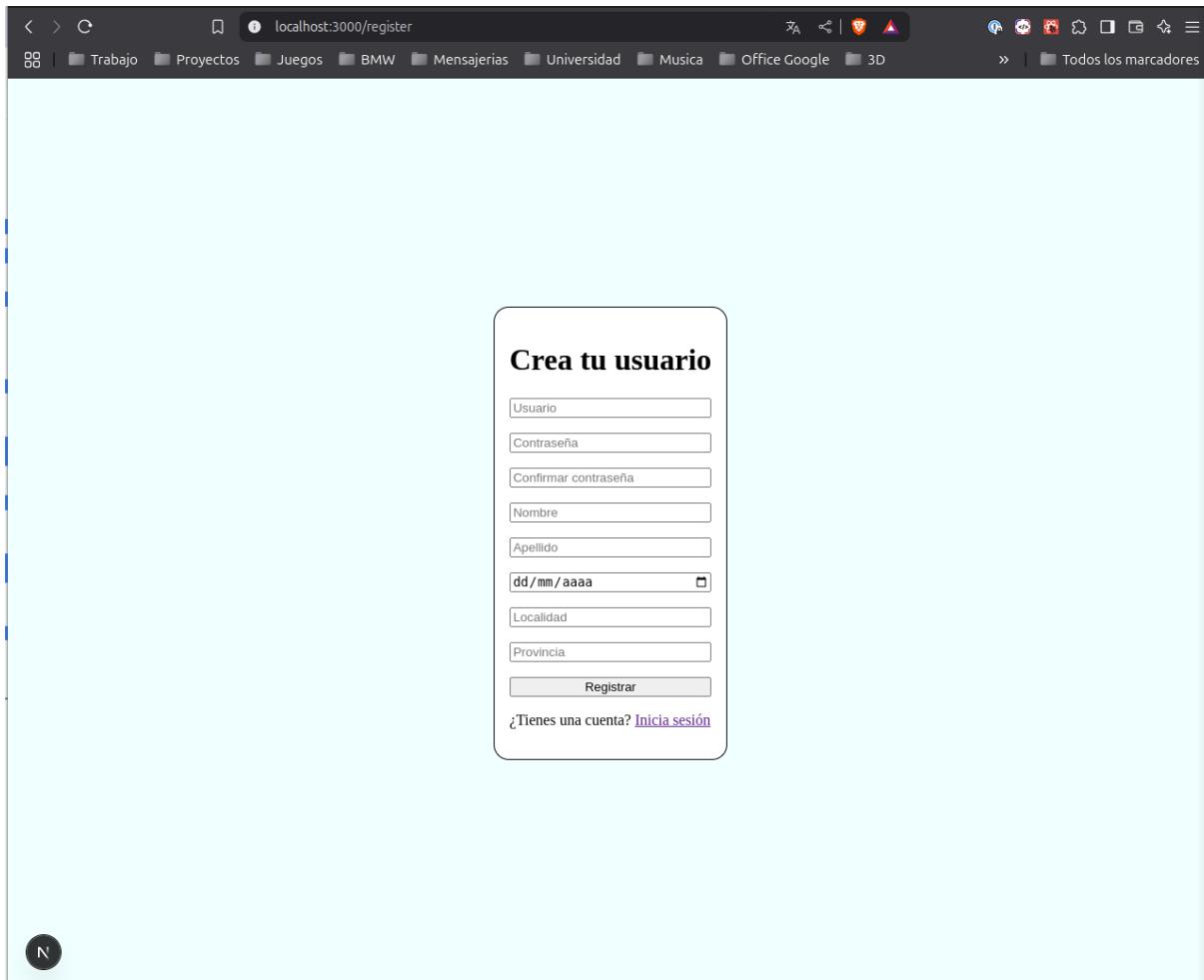
Dentro del fichero page.js, crea lo siguiente:

```

JS page.js app M    JS page.js .../register U ×  CSS page.module.css U
app > register > JS page.js > ...
1  import InitTemplate from "@/components/InitTemplate/InitTemplate";
2  import styles from "./page.module.css";
3  import Link from "next/link";  40k (gzipped: 11.8k)
4
5  export default function Register() {
6    return (
7      <InitTemplate>
8        <h1>Crea tu usuario</h1>    "Crea": Unknown word.
9        <form className={styles.loginForm}>
10          <input type="text" placeholder="Usuario" name="username" required />    "Usuario": Unknown word.
11          <input
12            type="password"
13            placeholder="Contraseña"    "Contraseña": Unknown word.
14            name="password"
15            required
16          />
17          <input
18            type="password"
19            placeholder="Confirmar contraseña"    "Confirmar": Unknown word.
20            name="confirmPassword"
21            required
22          />
23          <input type="text" placeholder="Nombre" name="name" required />    "Nombre": Unknown word.
24          <input type="text" placeholder="Apellido" name="lastName" required />    "Apellido": Unknown word.
25          <input
26            type="date"
27            placeholder="Fecha de nacimiento"    "Fecha": Unknown word.
28            name="birthDate"
29            required
30          />
31          <input type="text" placeholder="Localidad" name="locality" required />    "Localidad": Unknown word.
32          <input type="text" placeholder="Provincia" name="province" required />    "Provincia": Unknown word.
33          <button type="submit">Registrar</button>
34        </form>
35        <p>
36          ¿Tienes una cuenta? <Link href="/">Inicia sesión</Link>    "Tienes": Unknown word.
37        </p>
38      </InitTemplate>
39    );
40  }
A1

```

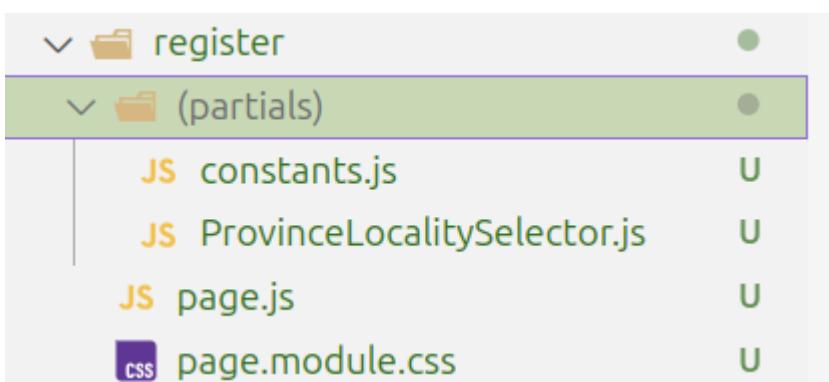
Si lo has hecho bien hasta ahora, deberías ver algo como lo siguiente:



Como verás, ahora, el selector de provincia y localidad no cumple lo requerido. Vamos a aprovechar para usar una de las ventajas de React: La componentización.

Dicho de otro modo, vamos a crear unos componentes que se encarguen de la lógica de mostrar las localidades según la provincia.

Crea una carpeta dentro de register y la llamas (*partials*). Recuerda que tiene que ir entre paréntesis para que NextJS no cree una URL basada en ese nombre. Dentro, crea un fichero llamado **constants.js** y otro llamado **ProvinceLocalitySelector.js**



Ahora, abre constants.js y escribe lo siguiente:

```
export const LocalitiesByProvince = {
  Valencia: ["Valencia", "Alcoy", "Castellón"],    "Alcoy"
  Madrid: ["Madrid", "Alcobendas", "Alcorcón"],    "Alcorcón"
};
```

Ahora, dentro del componente ProvinceLocalitySelector.js, escribe lo siguiente:

```
1 import React from "react"; 7.9k (gzipped: 3.1k)
2 import { LocalitiesByProvince } from "./constants";
3
4 const ProvinceLocalitySelector = ({{
5   province,
6   setProvince,
7   locality,
8   setLocality,
9 }) => {
10   return (
11     <>
12       <select
13         value={province}
14         onChange={(e) => setProvince(e.target.value)}
15         required
16       >
17         <option value="">Selecciona una provincia</option>    "Selecciona": Unknown word.
18         {Object.keys(LocalitiesByProvince).map((province) => (
19           <option key={province} value={province}>
20             {province}
21           </option>
22         )))
23       </select>
24       {province && (
25         <select
26           value={locality}
27           onChange={(e) => setLocality(e.target.value)}
28           required
29         >
30           <option value="">Selecciona una localidad</option>    "Selecciona": Unknown word.
31           {LocalitiesByProvince[province].map((locality) => (
32             <option key={locality} value={locality}>
33               {locality}
34             </option>
35           )))
36         </select>
37       )}
38     </>
39   );
40 };
41
42 export default ProvinceLocalitySelector;
43
```

Cosas importantes para tener en cuenta:

- Las provincias se cargan gracias a Object.keys. Esta función obtiene las keys del objeto LocalitiesByProvince (“Valencia” y “Madrid”)
- El selector de localidad se pintará solo cuando se ha elegido una provincia gracias al check {province && de la linea 24)

- El componente usa un fichero constants en el que están definidos constantes que no van a cambiar nunca.

Este uso es algo sencillo de React. Estúdialo, entiéndelo y práctica. Deberías ser capaz de hacer algo así sin problema.

Vamos a usar este componente en nuestra página de registro. Vete a page.js de register y añade las siguientes modificaciones:

```

JS ProvinceLocalitySelector.js U      JS page.js U X
app > register > JS page.js > Register > Explain Refactor Add Docstring
1 "use client";
2
3 import InitTemplate from "@/components/InitTemplate/InitTemplate";
4 import styles from "./page.module.css";
5 import Link from "next/link"; 40k (gzipped: 11.8k)
6 import { useState } from "react"; 4.6k (gzipped: 1.9k)
7 import ProvinceLocalitySelector from "./(partials)/ProvinceLocalitySelector";
8
9 export default function Register() {
10   const [province, setProvince] = useState("");
11   const [locality, setLocality] = useState("");
12   Ctrl+I for Command, Ctrl+L for Cascade
13   return (
14     <InitTemplate>
15       <h1>Crea tu usuario</h1>    "Crea": Unknown word.
16       <form className={styles.loginForm}>
17         <input type="text" placeholder="usuario" name="username" required />    "Usuario": Unknown
18         <input
19           type="password"
20           placeholder="Contraseña"    "Contraseña": Unknown word.
21           name="password"
22           required
23         />
24         <input
25           type="password"
26           placeholder="Confirmar contraseña"    "Confirmar": Unknown word.
27           name="confirmPassword"
28           required
29         />
30         <input type="text" placeholder="Nombre" name="name" required />    "Nombre": Unknown word.
31         <input type="text" placeholder="Apellido" name="lastName" required />    "Apellido": Unkn
32         <input
33           type="date"
34           placeholder="Fecha de nacimiento"    "Fecha": Unknown word.
35           name="birthDate"
36           required
37         />
38         <ProvinceLocalitySelector
39           province={province}
40           setProvince={setProvince}
41           locality={locality}
42           setLocality={setLocality}
43         />
44         <button type="submit">Registrar</button>
45       </form>
46     <p>
47       ¿Tienes una cuenta? <Link href="/">Inicia sesión</Link>    "Tienes": Unknown word.
48     </p>
49   </InitTemplate>
50 );
51 }

```

Modificaciones:

- Usamos “use client” porque vamos a usar estados. (línea 1)
- Añadimos dos state para la provincia y localidad (líneas 10 y 11)

- Importamos el nuevo componente (linea 7)
- Sustituimos los inputs de localidad y provincia por el nuevo componente que importamos (linea 38)

Ahora deberías ver algo así:



The screenshot shows a user registration form titled "Crea tu usuario". The form consists of several input fields: "Usuario", "Contraseña", "Confirmar contraseña", "Nombre", "Apellido", and "dd/mm/aaaa". There is also a dropdown menu labeled "Selecciona una provincia" and a "Registrar" button. Below the form, a link says "¿Tienes una cuenta? Inicia sesión".

Y cuando seleccionas una provincia:

Crea tu usuario

Usuario

Contraseña

Confirmar contraseña

Nombre

Apellido

dd/mm/aaaa

Valencia

Selecciona una localidad

¿Tienes una cuenta? [Inicia sesión](#)

Integración de Backend para Login

En este punto, vamos a darle funcionalidad a la pantalla de login y registro para crear y logearnos con un usuario.

Si observamos con postman la colección que os dimos, veréis que para hacer login necesitamos hacer un post a <https://das-p2-backend.onrender.com/api/users/login/> con un body como el que sigue:

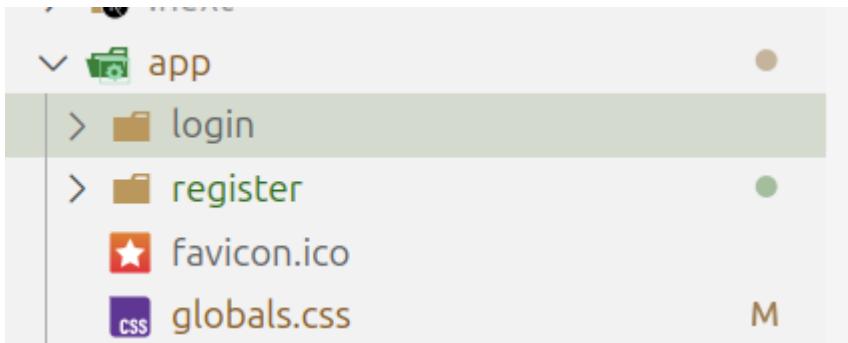
```
1▼ {  
2   "username": "testuser",  
3   "password": "securepassword"  
4 }  
5
```

Vamos a implementarlo.

Primero, vamos a hacer una reestructuración del proyecto.

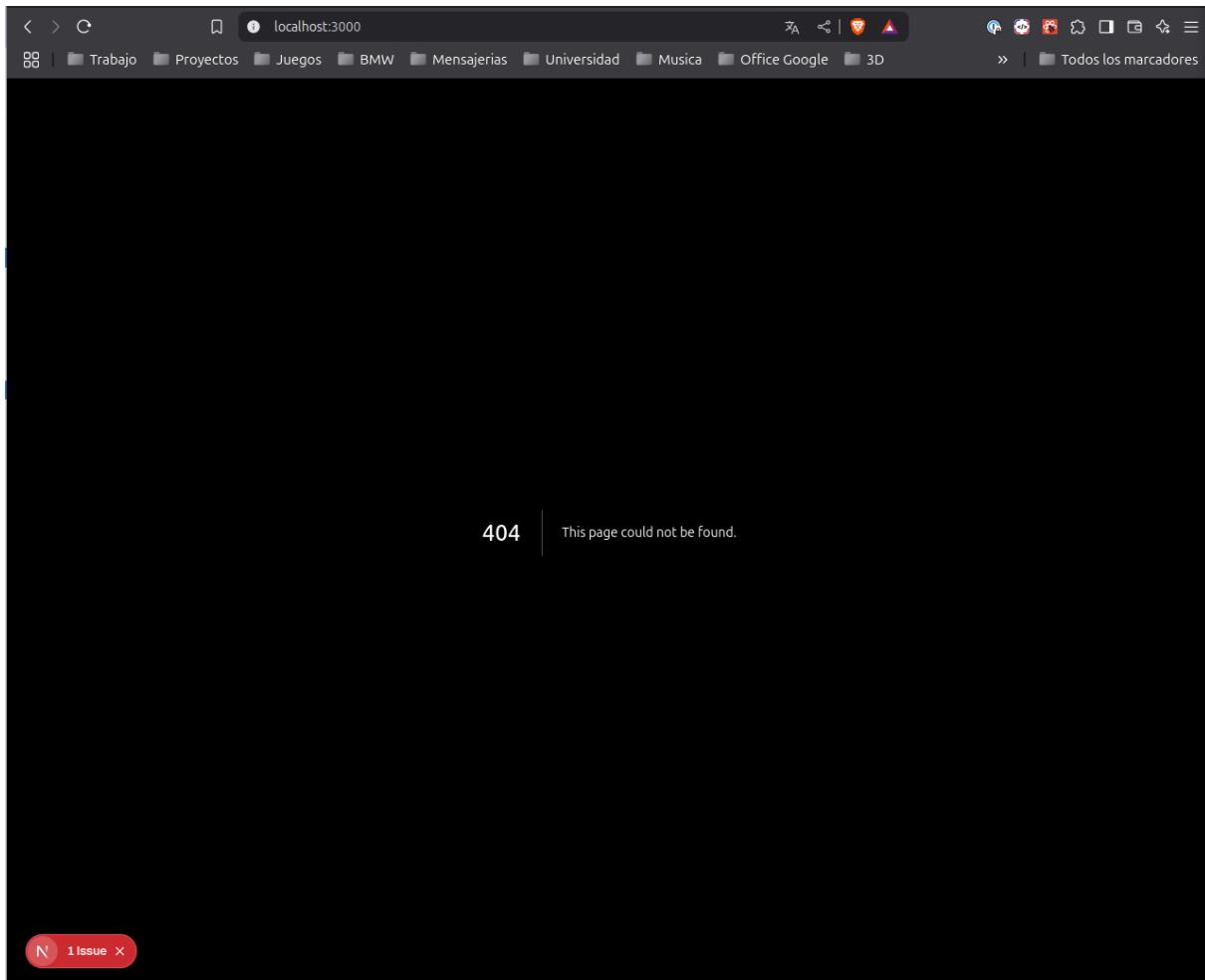
Si te das cuenta, hasta ahora, la pantalla de login no está en `http://localhost:3000/login` sino en <http://localhost:3000>. Vamos a solventar esto:

Crea dentro de app una carpeta llamada login.

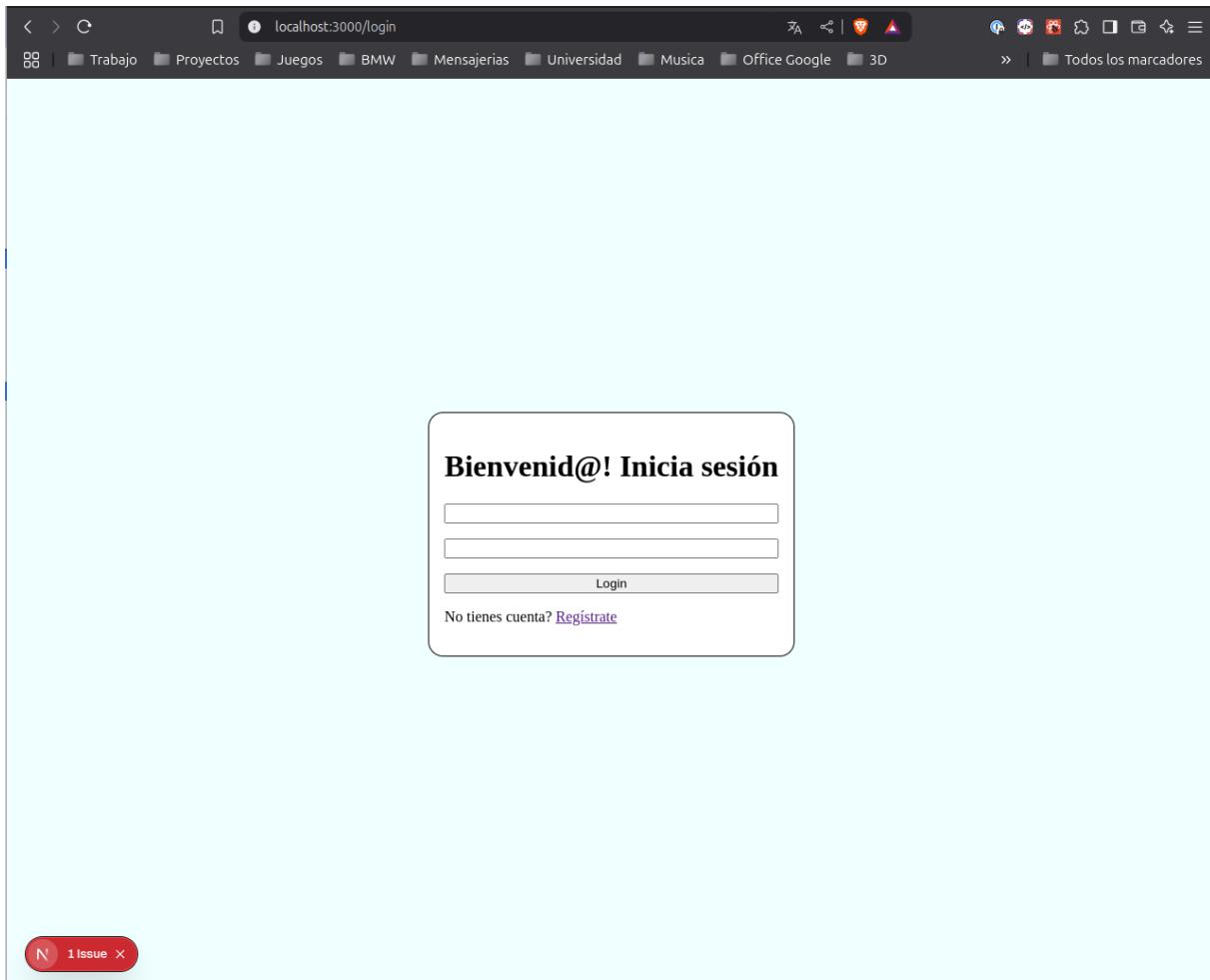


Ahora, arrastra dentro el fichero `page.js` y `page.module.css` que existen al mismo nivel. En los que has programado el login.

Si lo has hecho bien. Ahora cuando entres en <http://localhost:3000/> deberías ver un error 404:



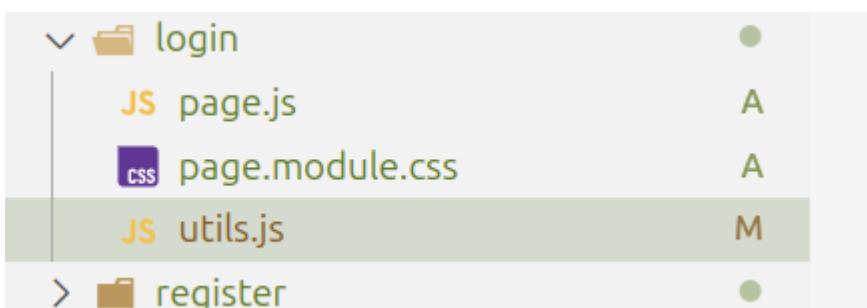
Pero si entras en <http://localhost:3000/login> ...



Ejercicio: Edita la página de registro para redireccionar al login en vez de al index.

No te preocupes ahora por ese error 404. Ya lo gestionaremos más tarde.

Vamos a implementar la petición de login. Para ello, ve a la carpeta que acabas de crear (login) y crea dentro un fichero llamado utils.js



Ahora, dentro de utils, implementa lo siguiente:

```

1 You, 1 minute ago | 1 author (You)
2 export const doLogin = async (username, password) => {
3   const response = await fetch(
4     {
5       method: "POST",
6       headers: {
7         "Content-Type": "application/json",
8       },
9       body: JSON.stringify({ username, password }),
10    }
11  );
12
13  const data = await response.json();
14
15  return data;
16}
17

```

Esta es una función llamada doLogin que, con un usuario y contraseña, hace una petición POST al backend <https://das-p2-backend.onrender.com/api/users/login/>.

Fíjate en la línea 9 como se hace un JSON.stringify del body.

Vamos a usarla. Para ello, ve al fichero page.js de la carpeta login e implementa los siguientes cambios:

```

1 "use client";
2
3 import InitTemplate from "@/components/InitTemplate/InitTemplate";
4 import Link from "next/link"; 40k (gzipped: 11.8k)
5 import styles from "./page.module.css";
6 import { doLogin } from "./utils";
7 import { useRouter } from "next/navigation"; 34.8k (gzipped: 7.5k)
8 import { useState } from "react"; 4.6k (gzipped: 1.9k)
9

```

```

10  export default function Login() {
11    const [loading, setLoading] = useState(false);
12    const router = useRouter();
13
14    const handleOnSubmit = async (event) => {
15      event.preventDefault();
16      const formData = new FormData(event.target);
17
18      const username = formData.get("username");
19      const password = formData.get("password");
20
21      setLoading(true);
22      try {
23        const userLogged = await doLogin(username, password);
24
25        if (userLogged.error) {
26          alert(userLogged.error);
27          return;
28        }
29
30        localStorage.setItem("token-jwt", userLogged.access);
31        localStorage.setItem("userName", userLogged.username);
32
33        router.push("/auctions");
34      } catch {
35        alert("Algo salió mal!");   "salio": Unknown word.
36      } finally {
37        setLoading(false);
38      }
39    };
40    return (
41      <InitTemplate>
42        <h1>Bienvenid@! Inicia sesión</h1>   "Bienvenid": Unknown word.
43        <form className={styles.loginForm} onSubmit={handleOnSubmit}>
44          <input type="text" name="username" required />
45          <input type="password" name="password" required />
46          {loading ? <p>Espere, por favor...</p> : <button type="submit">Login</button>}
47        </form>
48      </p>
49      No tienes cuenta? <Link href="/register">Regístrate</Link>   "tienes": Unknown word.
50    </p>
51  </InitTemplate>
52 );
53
54

```

- Renombra el componente de Index a Login (línea 10)
- Activa “use client” dado que vamos a usar un estado (línea 1)
- Añade un estado para gestionar si está en estado de loading o no. (línea 11 y 46)
- Crearemos una función para gestionar el onSubmit del form (línea 14)
- Añade required y name en los inputs del form (línea 44 y 45)
- Añade evento onSubmit (línea 44)

Vamos a analizar la función **handleOnSubmit**:

- Prevenimos la burbuja para que no haga un refresco de página por el comportamiento por defecto del form
- Usamos un objeto FormData para parsear los valores del form. Ten en cuenta que el name del input debe coincidir con el parámetro de formData.get.
- Usamos la función doLogin teniendo en cuenta que es una promesa:
 - Ponemos a true el estado loading
 - Try -> Hacemos la llamada, obtenemos el token y lo guardamos en JWT.
 - Catch -> caso de error y mostramos un alert
 - Finally -> Se ejecuta tanto si ha ido bien como mal. Ponemos a false el estado de loading

Fíjate en la línea 33 en la que, si todo ha ido bien, hacemos una redirección a auctions (donde mostraremos el listado de subastas).

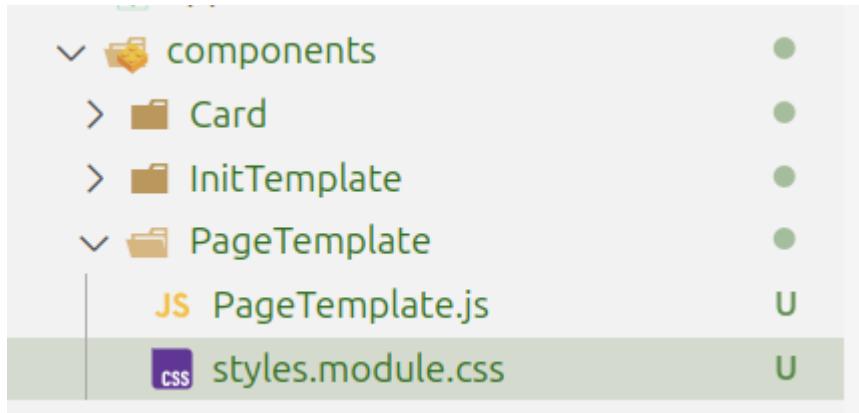
Con esto ya tenemos el login realizado haciendo una petición. El mismo concepto se aplicaría a la pantalla de registro.

Creación de pantalla de subastas

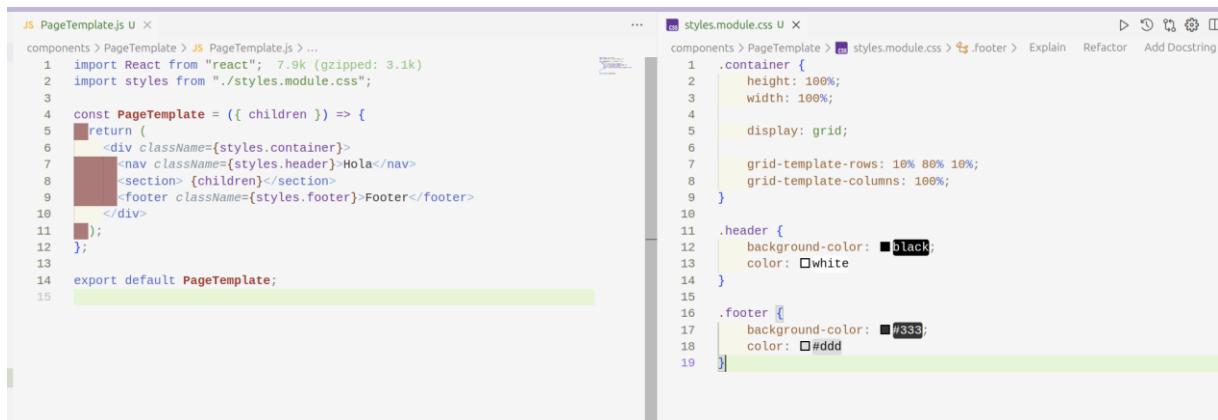
En este punto, ya tenemos nuestras pantallas de login y registro con cierta funcionalidad compleja y llamadas al backend. Vamos ahora a crear una pantalla de subastas. Además, tened en cuenta que a partir de este punto, todas las pantallas deben tener cabecera y footer.

Vamos a implementar el homónimo a **InitTemplate** pero que tenga cabecera, footer y, además, tenga botones de login.

Vamos a ir a la carpeta de **components** y vamos a crear una nueva estructura para un componente llamado **PageTemplate**:



Ahora, edita los ficheros tal que así:



```

JS PageTemplate.js U X
components > PageTemplate > JS PageTemplate.js > ...
1 import React from "react"; 7.9k (gzipped: 3.1k)
2 import styles from "./styles.module.css";
3
4 const PageTemplate = ({ children }) => {
5   return (
6     <div className={styles.container}>
7       <nav className={styles.header}>Hola</nav>
8       <section>{children}</section>
9       <footer className={styles.footer}>Footer</footer>
10    </div>
11  );
12}
13
14 export default PageTemplate;
15

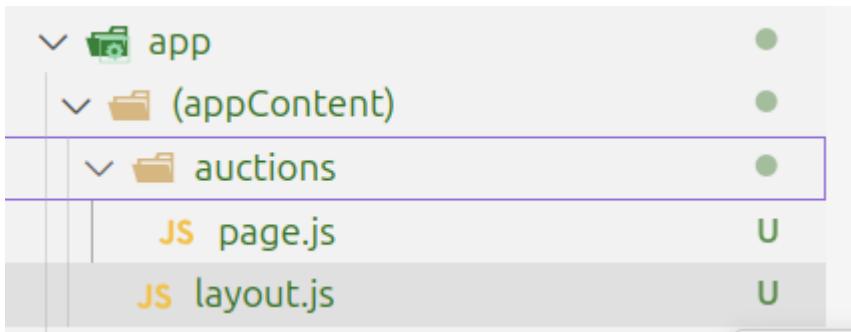
css styles.module.css U
components > PageTemplate > JS PageTemplate.js > styles.module.css > footer > Explain Refactor Add Docstring
1 .container {
2   height: 100%;
3   width: 100%;
4
5   display: grid;
6
7   grid-template-rows: 10% 80% 10%;
8   grid-template-columns: 100%;
9 }
10
11 .header {
12   background-color: black;
13   color: white
14 }
15
16 .footer {
17   background-color: #333;
18   color: #ddd
19 }

```

Ahora, vamos a crear la página de auctions. Pero, para ello, vamos a tener ciertas consideraciones:

- Queremos que ahora todas las páginas cargen un layout principal usando PageTemplate

Para ello, crea la siguiente estructura en app:



Observa que:

- La carpeta appContent está entre paréntesis
 - NextJS la ignorará, pero nos permite definir un layout que se compartirá
 - Layout.js está al mismo nivel que la carpeta auctions. Auctions y más carpetas las carpetas que creemos dentro de appContent lo usarán.

Ahora, dentro de layout.js, crea lo siguiente:

```

JS layout.js U ×
app > (appContent) > JS layout.js > ...
1 import PageTemplate from "@/components/PageTemplate/PageTemplate";
2
3 export default function AuctionsLayout({ children }) {
4   return <PageTemplate>{children}</PageTemplate>;
5 }
6
  
```

Y vamos a inicializar la página de auctions:

```

JS page.js U ×
app > (appContent) > auctions > JS page.js > ...
1 const Auctions = () => {
2   return <div>Auctions</div>;
3 };
4
5 export default Auctions;
6
  
```

Ahora, si visitas <http://localhost:3000/login> verás esto:

Bienvenid@! Inicia sesión

Login

No tienes cuenta? [Regístrate](#)

N 1 issue X

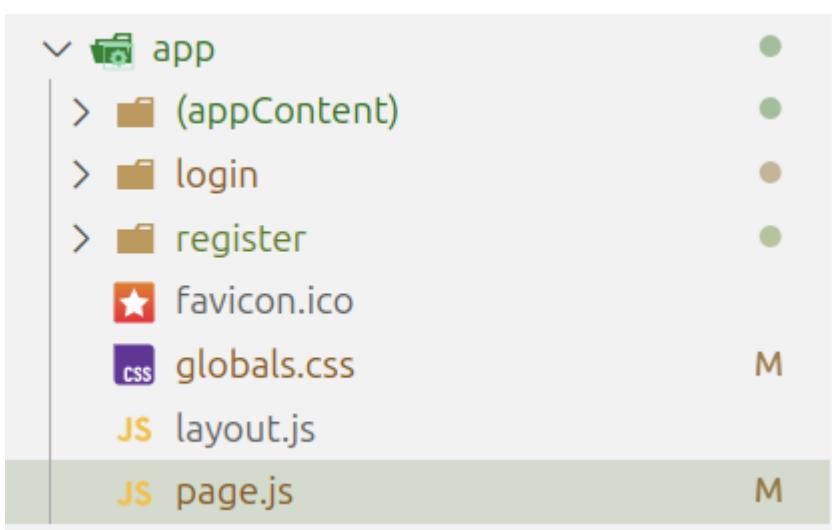
Pero si visitas <http://localhost:3000/auctions> verás:



En este punto, vamos a hacer un inciso para quitar el error 400 que aparece en <http://localhost:3000/>

La estrategia va a ser simple: Vamos a ignorar esta URL y hacer que siempre redireccione a /auctions.

Para ello, crea un fichero page.js a la altura de app:



Y dentro programa el siguiente contenido:

```

JS page.js .../auctions U      JS page.js app M X
app > JS page.js > ...
You, 1 second ago | 1 author (You)
"use client";
import { useEffect } from "react";  4.7k (gzipped: 2k)
import { useRouter } from "next/navigation";  34.8k (gzipped: 7.5k)

5 const Index = () => {
6   const router = useRouter();
7   useEffect(() => {
8     router.push("/auctions");
9   }, [router]);
10
11   return <div>Cargando...</div>;    "Cargando": Unknown word.
12 };
13
14 export default Index;
15 | Ctrl+I for Command, Ctrl+L for Cascade
  
```

Lo que hacemos aquí es redireccionar con router.push a /auctions cada vez que entramos.

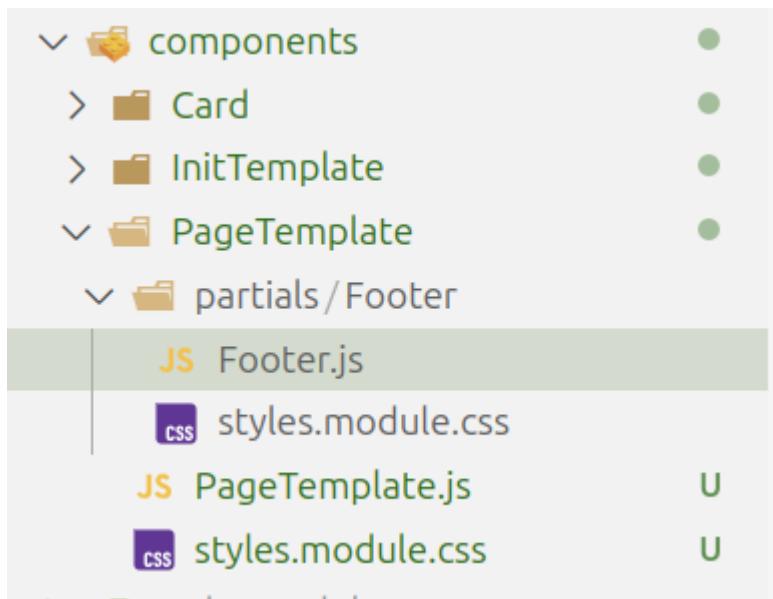
Si se viera algo, veríamos un texto que dice Cargando....

Habría mejores maneras de hacerlas, pero de momento esto nos salva.

Ahora, si observamos la pantalla de auctions, vemos que el header y el footer no tienen contenido. Simplemente tienen un texto sencillo. Vamos a cambiarlo.

Para ello, vamos a modificar el footer primero. Ahora, pensemos: El header y el footer se van a usar únicamente en el PageTemplate. Por lo tanto, no tiene sentido añadirlo en la carpeta Components. Es un partial de PageTemplate.

Dentro de la carpeta PageTemplate, crea una subcarpeta llamada partials. Dentro, crea una carpeta llamada Footer y dentro un fichero Footer.js y styles.module.css :



Los ficheros de Footer.js y styles.module.css deberían ir tal que así:



Left Editor (Footer.js):

```

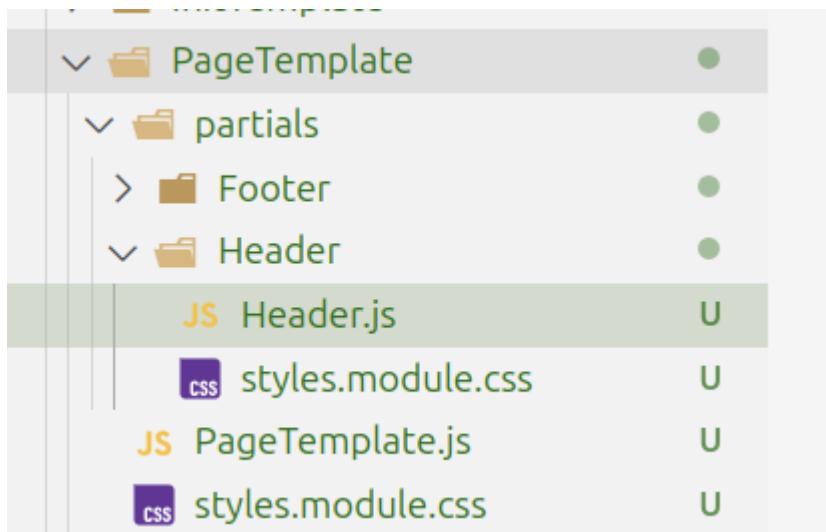
1 import React from "react"; 7.9k (zipped: 3.1k)
2 import styles from "./styles.module.css";
3
4 const Footer = () => {
5   return (
6     <footer className={styles.footer}>
7       Hecha en DAS para el curso {new Date().getFullYear()} "Hecha": l
8     </footer>
9   );
10 }
11
12 export default Footer;
13
  
```

Right Editor (styles.module.css):

```

1 .footer {
2   display: flex;
3   text-align: center;
4   justify-content: center;
5   align-items: center;
6
7   background-color: #333;
8   color: #ddd;
9 }
  
```

Ahora, vamos a hacer lo mismo para el header:



Y el contenido de sus ficheros....



```
1 .header {  
2     background-color: black;  
3     color: white;  
4     display: flex;  
5     justify-content: space-between;  
6     align-items: center;  
7     padding: 1rem;  
8 }  
9  
10  
11 .actionContainer {  
12     display: flex;  
13     gap: 1rem;  
14 }  
15  
16  
17 .linkStyle {  
18     text-decoration: none;  
19     color: white;  
20  
21     :hover {  
22         text-decoration: underline;  
23         color: #ccc;  
24     }  
25 }
```

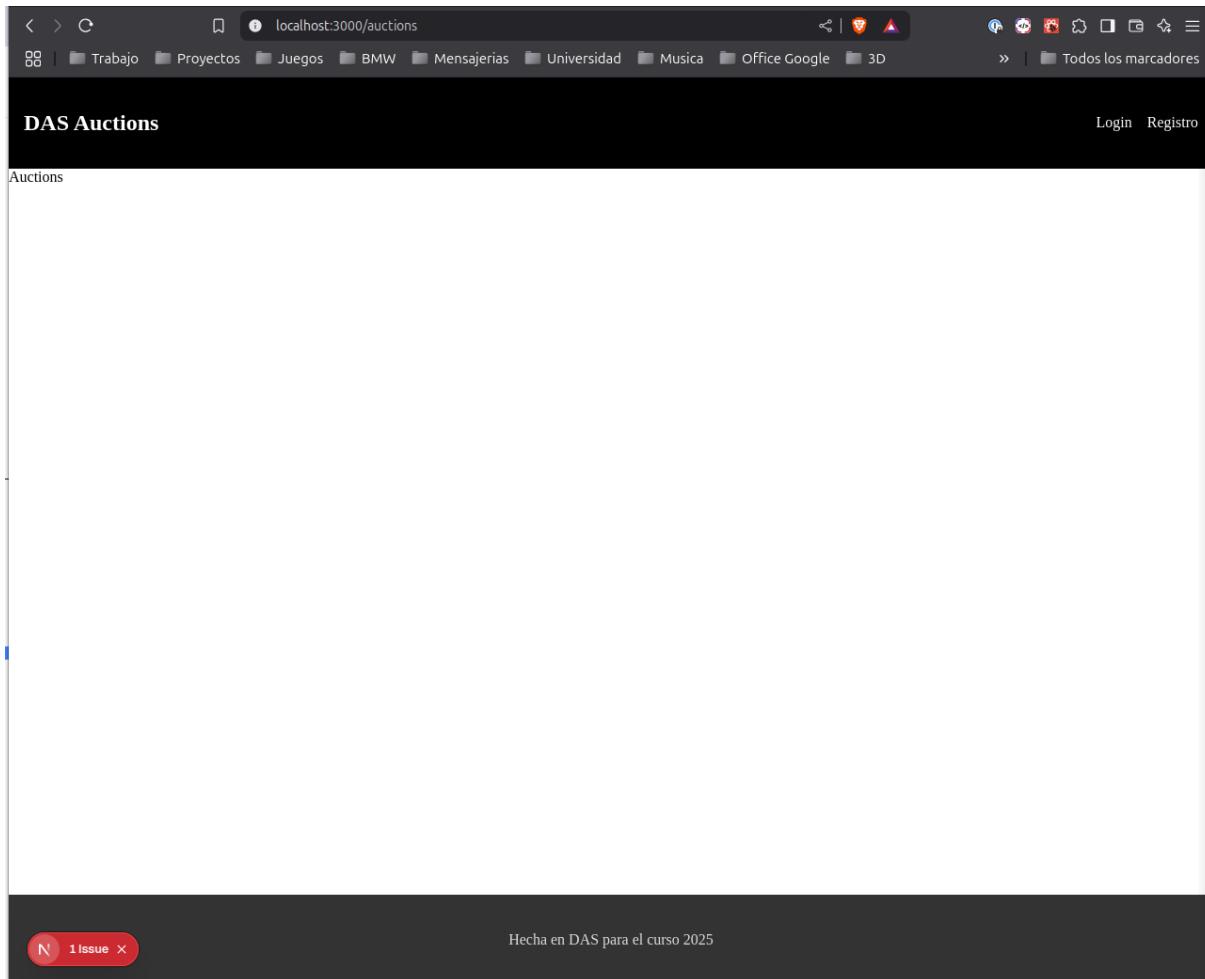
```

1  "use client";
2
3  import React from "react"; 7.9k (gzipped: 3.1k)
4  import styles from "./styles.module.css";
5  import Link from "next/link"; 40k (gzipped: 11.8k)
6
7  const Header = () => {
8    const token = localStorage.getItem("token-jwt");
9    const userName = localStorage.getItem("userName");
10
11   const handleLogout = () => {
12     localStorage.removeItem("token-jwt");
13     localStorage.removeItem("userName");
14   };
15
16   return (
17     <nav className={styles.header}>
18       <Link href="/auctions" className={styles.linkStyle}>
19         <h1>DAS Auctions</h1>{" "}
20       </Link>
21
22       <div className={styles.actionContainer}>
23         {token ? (
24           <>
25             <Link href="/me" className={styles.linkStyle}>
26               <p>{userName}</p>
27             </Link>
28             <Link
29               href="/auctions"
30               className={styles.linkStyle}
31               onClick={handleLogout}
32             >
33               <p>Log out</p>
34             </Link>
35           ) : (
36             <>
37               <Link href="/login" className={styles.linkStyle}>
38                 <p>Login</p>
39               </Link>
40               <Link href="/register" className={styles.linkStyle}>
41                 <p>Registro</p>    "Registro": Unknown word.
42               </Link>
43             </>
44           )
45         )
46       </div>
47     </nav>
48   );
49 }
50
51 export default Header;

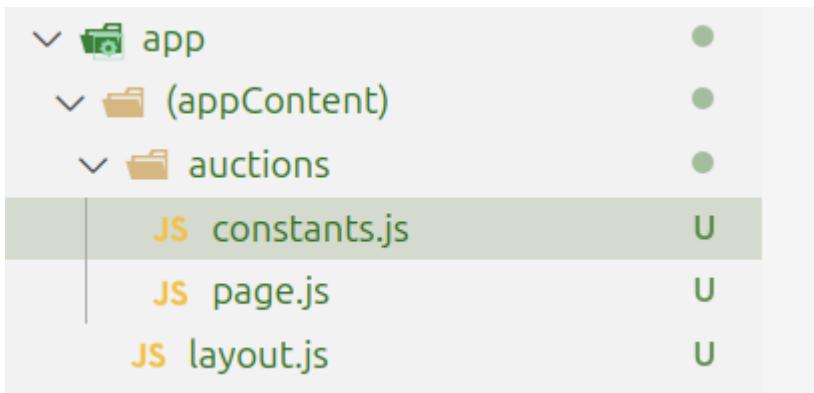
```

Observa que en el header gestionamos si hay o no token y username obteniendolos del local storage.

Deberías ver en tu web algo como esto:



Vamos a terminar la página de auctions. Crea en la carpeta auctions un fichero llamado constants.js:



Dentro, usa una IA para que te genere un array de subastas tal que así. Yo he pedido que me genere 50 subastas para gestionar un scroll:

```
$ constants.js ✘ ×
app > (appContent) > auctions > JS constants.js > [e] auctions > Explain Refactor Add Docstring
1  export const auctions = [
2  [
3    {
4      name: "Reloj de lujo",      "Reloj": Unknown word.
5      price: 100,
6      open: true,
7    },
8    {
9      name: "Teléfono último modelo",      "Teléfono": Unknown word.
10     price: 200,
11     open: false,
12   },
13   {
14     name: "Cartera de cuero",      "Cartera": Unknown word.
15     price: 300,
16     open: true,
17   },
18   {
19     name: "Ordenador portátil",      "Ordenador": Unknown word.
20     price: 400,
21   }
22 ]
```

Vamos ahora al page.js de auctions y vamos a añadir lo siguiente:

```
app > (appContent) > auctions > JS page.js > ...
1  import Card from "@/components/Card/Card";
2  import { auctions } from "./constants";
3
4  const Auctions = () => {
5    return (
6      <>
7        <h2>Subastas disponibles</h2>      "Subastas": Unknown word.
8        {auctions.map((auction) => (
9          <Card key={auction.name}>
10            <h3>{auction.name}</h3>
11            <p>{auction.price}€</p>
12            <p>{auction.open ? "Abierta" : "Cerrada"}</p>      "Abierta": Unk
13            {auction.open && <button>Participar</button>}      "Participar": "
14          </Card>
15        ))}
16      </>
17    );
18  };
19
20  export default Auctions;
21  Ctrl+I for Command, Ctrl+L for Cascade
```

Y deberías ver algo como esto:

DAS Auctions

Login Registro

Subastas disponibles

Reloj de lujo

100€

Abierta

Teléfono último modelo

200€

Cerrada

Cartera de cuero

300€

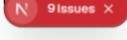
Abierta

Ordenador portátil

 9 issues 

Hecha en DAS para el curso 2025

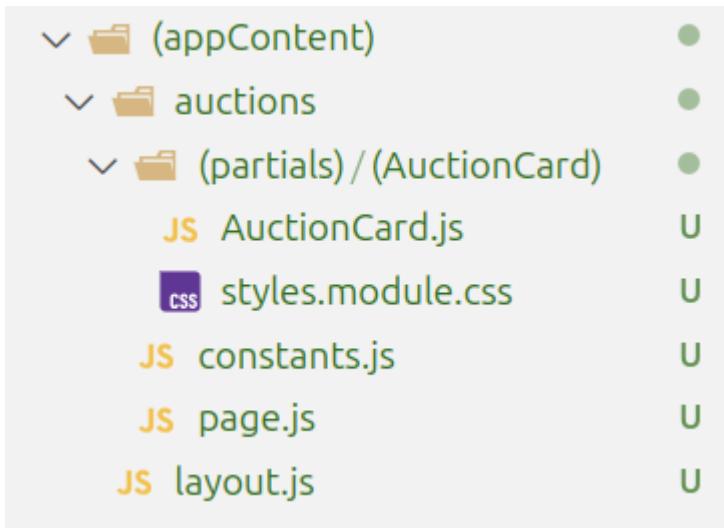
Además, verás que, si haces scroll para abajo, se rompe el layout de la web:

200€	Cerrada
Cartera de cuero	300€
Abierta	<input type="button" value="Participar"/>
Ordenador portátil	Hecha en DAS para el curso 2025
Bicicleta de montaña	500€
Abierta	<input type="button" value="Participar"/>
Televisor de 4K	600€
	 9 issues ×

Vamos a arreglar varias cosas por partes:

- Vamos a crear un componente llamado AuctionCard que usaremos para ver las subastas
- Vamos a crear un contenedor para las subastas
- Vamos a corregir el padding del PageTemplate para dejar un espacio dentro del contenedor
- Vamos a corregir el problema del scroll.

Crea un componente AuctionCard dentro de partials, en auctions, creando los ficheros para el componente y estilos:



El contenido de AuctionCard y styles es el siguiente:

```

JS AuctionCard.js U ×
app > (appContent) > auctions > (partials) > (AuctionCard) > JS AuctionCard.js > ...
1  "use client";
2  import Card from "@/components/Card/Card";
3  import styles from "./styles.module.css";
4  import Button from "@/components/Button/Button";
5
6  const AuctionCard = ({ name, price, open }) => {
7    return (
8      <Card className={styles.card}>
9        <h3>{name}</h3>
10       <p>{price}</p>
11       <p>{open ? "Abierta" : "Cerrada"}</p>    "Abierta": Unknown word.
12       {open && <Button label="Participar" onClick={() => {}} />}  "Part
13     </Card>
14   );
15 }
16
17 export default AuctionCard;
18

CSS styles.module.css U ×
app > (appContent) > auctions > (partials) > (AuctionCard) > CSS styles.module.css > .card > Explain Refactor ...
1  .card {
2    |   max-width: 10rem;
3  }
  
```

Ten en cuenta varias cosas:

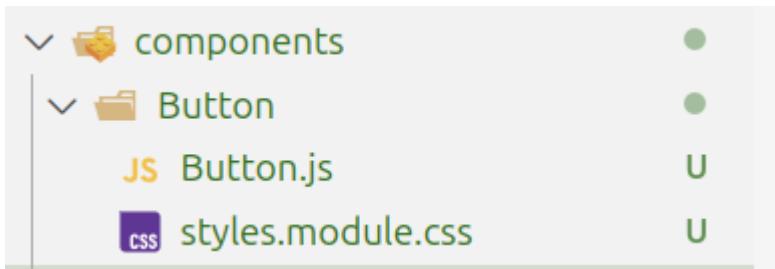
- Se usa un nuevo componente Button que definiremos ahora.
- Se modifica el componente Card para que acepte un className como propiedad. Así podemos extender con estilos y decirle que el max-width es de 10rem.

El componente Card queda así:

```

JS AuctionCard.js U      JS Card.js U ×
components > Card > JS Card.js > ...
1  import React from "react"; 7.9k (gzipped: 3.1k)
2  import styles from "./styles.module.css";
3
4  const Card = ({ children, className }) => {
5    return <div className={`${styles.container} ${className}`}>{children}</div>;
6  };
7
8  export default Card;
9  Ctrl+I for Command, Ctrl+L for Cascade
  
```

El nuevo componente Button es así:



Y, su contenido:

```

1 import React from "react"; 7.9k (gzipped: 3.1k)
2 import styles from "./styles.module.css";
3
4 const Button = ({ className, label, onClick, type }) => {
5   return (
6     <button
7       className={`${styles.button} ${className}`}
8       onClick={onClick}
9       type={type}
10    >
11      {label}
12      </button>
13    );
14 }
15
16 export default Button;
17
  
```

```

1 .button {
2   padding: 0.5rem 1rem;
3   border: 1px solid #ccc;
4   border-radius: 1rem;
5   cursor: pointer;
6   background-color: #333;
7   color: #fff;
8 }
  
```

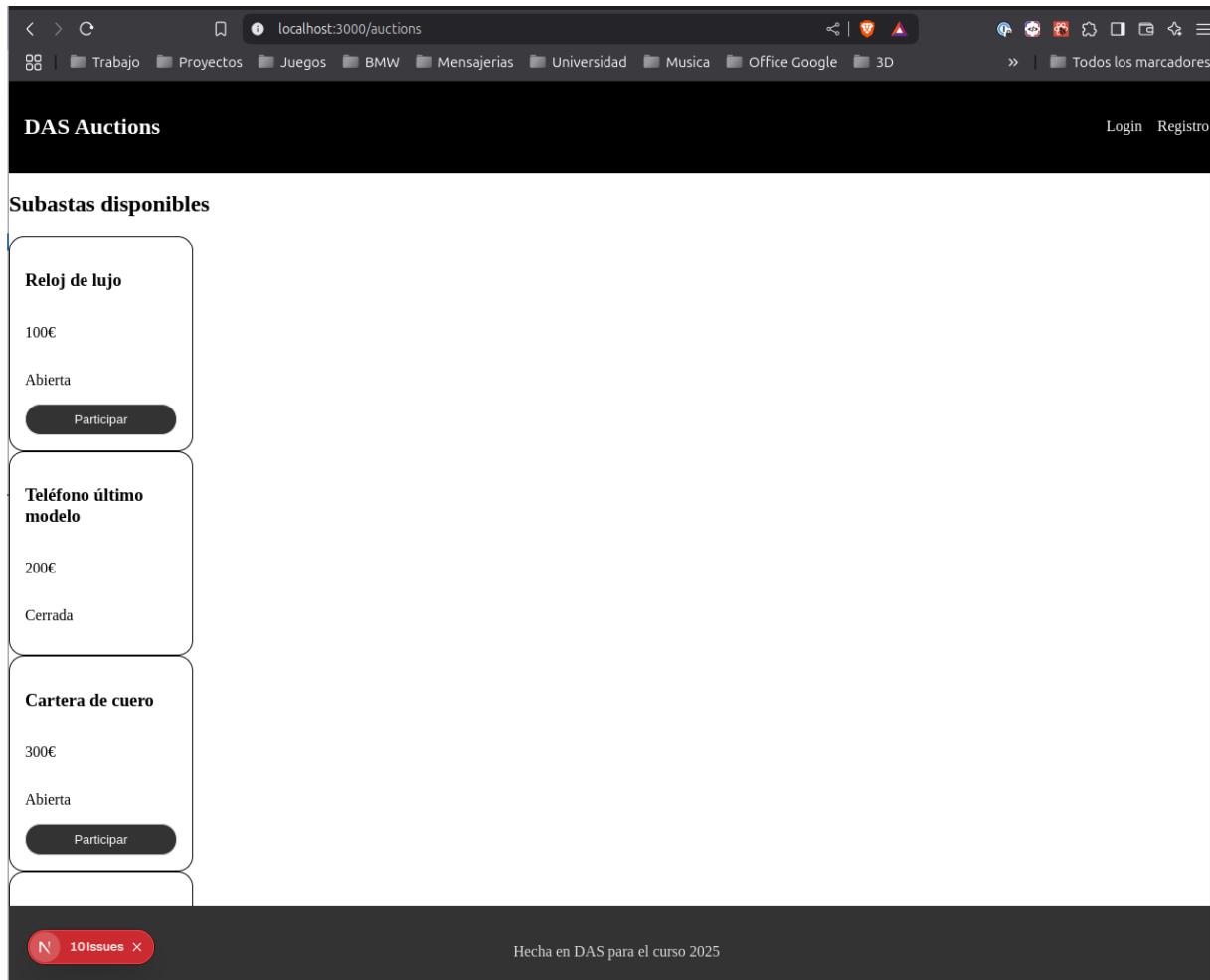
Fíjate en como className es una prop que se concatena.

Si usamos lo anterior en auctions....



```
css styles.module.css U JS page.js U X
app > (appContent) > auctions > JS page.js > ...
1 import { auctions } from "./constants";
2 import AuctionCard from "./(partials)/(AuctionCard)";
3
4 const Auctions = () => {
5   return (
6     <>
7       <h2>Subastas disponibles</h2>    "Subastas": [
8         {auctions.map((auction) => (
9           <AuctionCard
10             key={auction.name}
11             name={auction.name}
12             price={auction.price}
13             open={auction.open}
14           />
15         )));
16       </>
17     );
18   };
19
20 export default Auctions;
21
```

Deberíais ver algo tal que así:



DAS Auctions

Login Registro

Subastas disponibles

Reloj de lujo
100€
Abierta
Participar

Teléfono último modelo
200€
Cerrada

Cartera de cuero
300€
Abierta
Participar

N 10 Issues X Hecha en DAS para el curso 2025

Bien, ahora, para solventar el problema del scroll, vamos a realizar el siguiente cambio:

Id a components/PageTemplate/styles.module.css y dejadlo tal que así:

css styles.module.css U X

components > PageTemplate > **css** styles.module.css >  .section > Explain Re

```
1  .container {  
2      height: 100%;  
3      width: 100%;  
4      overflow: hidden;  
5  
6      display: grid;  
7  
8      grid-template-rows: 10% 80% 10%;  
9      grid-template-columns: 100%;  
10 }  
11  
12 .section {  
13     padding: 1rem;  
14     overflow: auto;  
15 }
```

Y components/PageTemplate/PageTemplate.js queda así:

JS PageTemplate.js U X

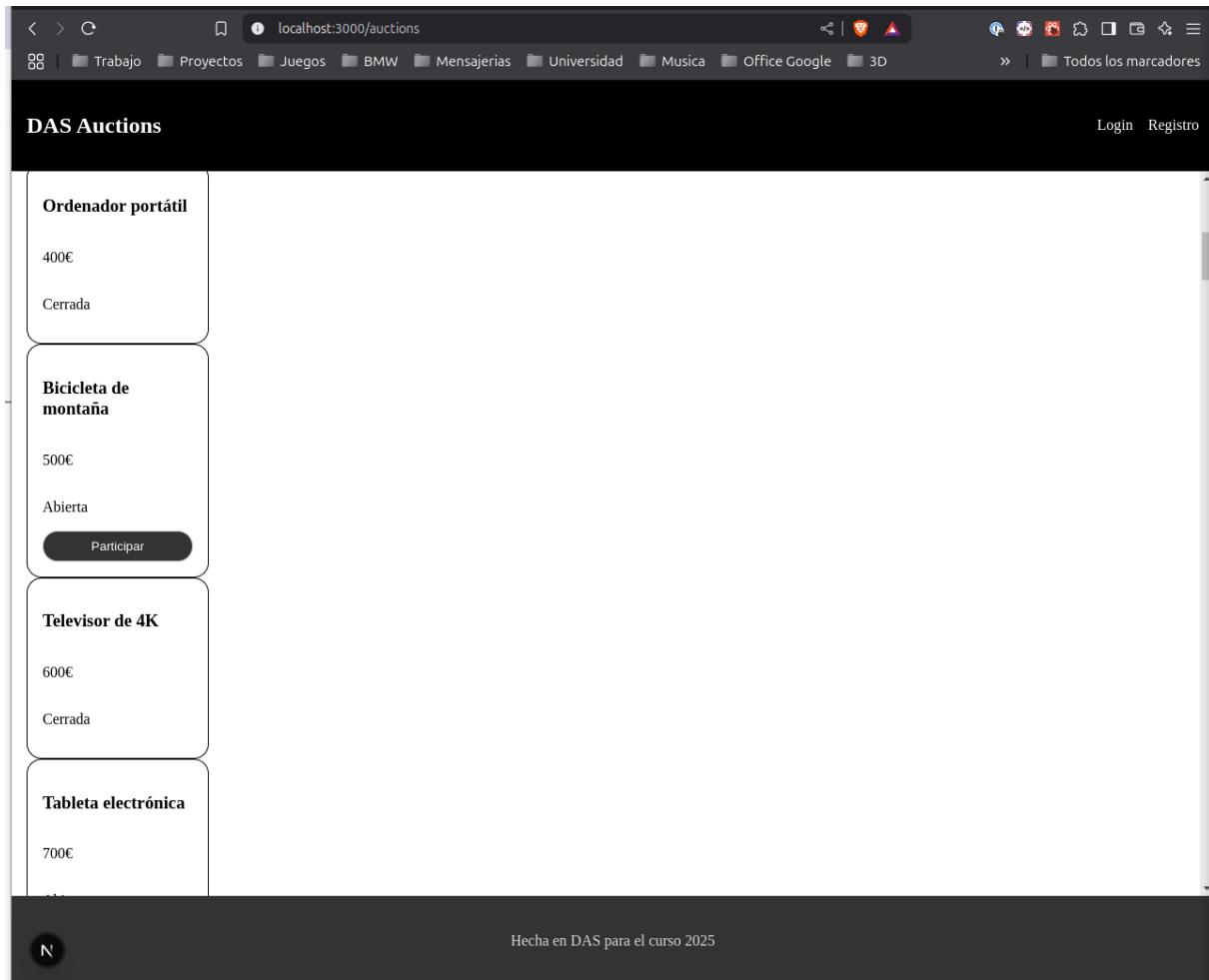
components > PageTemplate > JS PageTemplate.js > ...

```
1 import React from "react"; 7.9k (gzipped: 3.1k)
2 import styles from "./styles.module.css";
3 import Footer from "./partials/Footer/Footer";
4 import Header from "./partials/Header/Header";
5
6 const PageTemplate = ({ children }) => {
7   return (
8     <div className={styles.container}>
9       <Header />
10      <section className={styles.section}>{children}</section>
11      <Footer />
12    </div>
13  );
14}
15
16 export default PageTemplate;
17
```

Si os fijáis, lo que hemos hecho ha sido:

- Añadir un padding de 1 rem a section, que es donde se carga el contenido de la pág.
- Esconder el overflow del container (esto lo hace desaparecer) y añadirlo a section.

Ahora deberíais ver que el scroll queda justo entre el header y el footer 😊



DAS Auctions

Ordenador portátil
400€
Cerrada

Bicicleta de montaña
500€
Abierta
Participar

Televisor de 4K
600€
Cerrada

Tableta electrónica
700€

N Hecha en DAS para el curso 2025

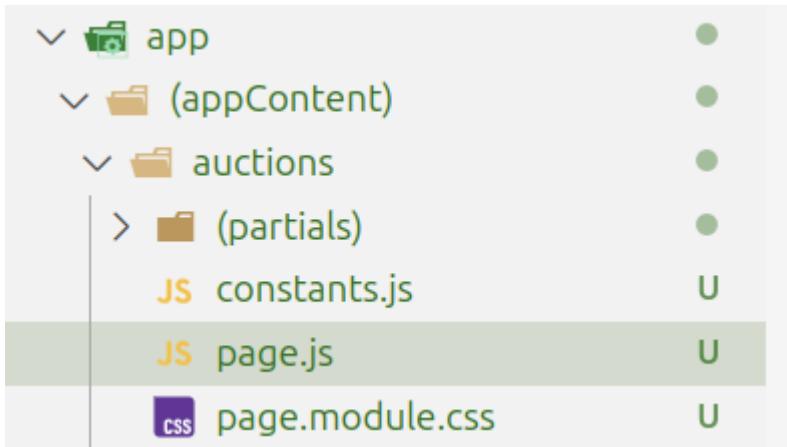
Vamos con el último paso: Crear un contenedor para las subastas.

Para ello vamos a envolver, en app/(appContent)/auctions/page.js, el auctions.map en un div que será el contenedor (línea 9):

```

JS page.js U X   CSS page.module.css U   CSS styles.module.css U
app > (appContent) > auctions > JS page.js > ...
1  import { auctions } from "./constants";
2  import AuctionCard from "./(partials)/(AuctionCard)/AuctionCard";
3  import styles from "./page.module.css";
4
5  const Auctions = () => {
6    return (
7      <>
8        <h2>Subastas disponibles</h2>    "Subastas": Unknown word.
9        <div className={styles.auctions}>
10       {auctions.map((auction) => (
11         <AuctionCard
12           key={auction.name}
13           name={auction.name}
14           price={auction.price}
15           open={auction.open}
16         />
17       ))}
18     </div>
19   </>
20 );
21
22
23 export default Auctions;
24
  
```

Añadimos el css pertinente, para ello creamos un fichero page.module.css para auctions:



Y asignamos el siguiente contenido:

css page.module.css U X css styles.module.css U

```
app > (appContent) > auctions > page.module.css > .auctions
1   .auctions {
2     display: flex;
3     flex-direction: row;
4     gap: 1rem;
5     flex-wrap: wrap;
6
7     justify-content: center;
8 }
```

Lo que hacemos con esto es usar Flex. Se podría hacer con Grid, pero para que tengáis otro ejemplo más del uso de Flex.

Lo que hacemos aquí es usar flex y la propiedad flex-wrap. Esta propiedad lo que hace es añadir componentes (AuctionCard) hasta el límite de la pantalla. Después, hace un salto.

Probad a redimensionar la pantalla y veréis como os entran distintas columnas de AuctionCards.

Por último, por estilo, en app/(appContent)/auctions/(partials)/(AuctionCard)/styles.module.css dejamos el estilo tal que así:

css page.module.css U css styles.module.css U

```
app > (appContent) > auctions > (partials) >
1   .card {
2     width: 10rem;
3 }
```

Esto lo hacemos para que sea homogéneo el tamaño de las cards.

Con esto, deberíais ver algo como esto:

DAS Auctions Login Registro

Subastas disponibles

Reloj de lujo 100€ Abierta Participar	Teléfono último modelo 200€ Cerrada Participar	Cartera de cuero 300€ Abierta Participar	Ordenador portátil 400€ Cerrada Participar	Bicicleta de montaña 500€ Abierta Participar	Televisor de 4K 600€ Cerrada Participar	Tableta electrónica 700€ Abierta Participar
Zapatos de lujo 800€ Cerrada Participar	Anillo de compromiso 900€ Abierta Participar	Cámara profesional 1000€ Cerrada Participar	Bolso de diseño 1100€ Abierta Participar	Smartphone de alta gama 1200€ Cerrada Participar	Reloj de pulsera 1300€ Abierta Participar	Cámara de seguridad 1400€ Cerrada Participar
Telescopio 1500€ Abierta Participar	Reproductor de música 1600€ Participar	Bicicleta de carretera 1700€ Participar	Cámara de fotos 1800€ Cerrada Participar	Reloj de pared 1900€ Abierta Participar	Altavoz inalámbrico 2000€ Participar	Teclado mecánico 2100€ Abierta Participar

N 9 Issues X

Hecha en DAS para el curso 2025

Con esto, tenéis más de la mitad de la P2 finalizada. Una estructura de NextJS sana y un ejemplo de peticiones.

Cosas importantes a tener en cuenta que hemos visto:

- No a ficheros css normales. Tenemos el css en modulos pequeñitos que acompañan al componente que aplican:
 - page.module.css para paginas
 - styles.module.css para componentes
- La carpeta de app se usa para las urls.
 - Uso de Layout segun necesidades
 - Uso de () para evitar crear URLs y crear partials
- Como usar ficheros constants para definir constantes.
- Ficheros utils para funciones puras de js
 - Si en una función usamos hooks, se transforma en un custom hook y viviría en un fichero hooks.js
- Uso variado de flex y grid para centrar y posicionar.