



Universitat
Oberta
de Catalunya

M4.258 - Herramientas HTML y CSS II

PEC 2

Aníbal Santos Gómez

ÍNDICE: DOCUMENTACIÓN.

- 1. Objetivos.**
- 2. Herramientas utilizadas.**
- 3. Metodologías aplicadas.**
- 4. Desarrollo.**
- 5. Publicación.**

1. Objetivos.

- Diseñar y ejecutar un pequeño sitio web responsive. Partiendo de un boilerplate.
- Escoger criterios de desarrollo y arquitectura reutilizable CSS. Utilizar el lenguaje de preprocesado de estilos Sass.
- Utilizar y configurar Stylelint como linter CSS para mejorar nuestra escritura de código CSS.
- Utilizar las técnicas de maquetado CSS flex y grid. Utilizar la librería de componentes Bootstrap.
- Incorporar React.js como librería JS para agilizar la construcción de una UI atómica y aligerar la escritura de Sass.
- Utilizar y configurar herramientas para la mejora y estilo de código, como ESLint y Prettier.
- Publicar el repositorio en GitHub y realizar un deployment en Netlify.
- Generar una documentación práctica que siga la elección de criterios, proceso de desarrollo y el proceso de despliegue.

2. Herramientas utilizadas.

Para el desarrollo de una aplicación web moderna, necesitamos herramientas modernas que nos permitan fácilmente el desarrollo de la misma. El principal objetivo de este desarrollo es centrarnos en la construcción de una que ofrecerá una Landing Page o página de producto (en este caso un grupo de música).

Para ello se eligen un conglomerado de herramientas que nos facilitarán, el desarrollo en local, el pase a producción, el control de versiones, la maquetación, la generación de código con un control de calidad y otra serie de herramientas que se justifican continuación:

- Editor de código: **Visual Studio Code**.

Se utilizará **VS Code** como editor de código, ya que nos permite instalar plugins que nos permiten utilizar snippets para agilizar nuestra forma de programar. Además es totalmente personalizable en función del proyecto que se vaya a realizar, en cuestión de herramientas. Actualmente se han instalado plugins como auto close tag, rename tag, css formatter, highlight matching tag, JS snippets, Prettier, Styling plugin, ESLint, Prettier, Babel Javascript, Quokka.js.

- Control de versiones y repositorio: **Git y Github**.

Para el control de versiones utilizaremos **Git** y para almacenar nuestro repositorio en la nube utilizaremos el servicio que nos brinde **Github**.

- Gestor de paquetes: **Npm**.

Como gestor de paquetes de Node.js utilizaremos **Npm**, que contiene un sin fin de dependencias desarrolladas por otros programadores que nos facilitarán la configuración de los diversos paquetes que utilizaremos.

- Module bundler: **Parcel**.

Para empaquetar todo nuestro proyecto utilizaremos **Parcel** ya que es por defecto el module bundler que se encuentra definido en el package.json de UOC boilerplate. Además crearemos un archivo de configuración (.parcelrc) para poder utilizar los plugins de Parcel.

- Preprocesadores de código: **PostCSS y Sass**.

PostCSS es una herramienta de desarrollo de software que utiliza complementos basados en JavaScript para automatizar las operaciones de rutina de CSS.

Sass que nos permitirá extender funcionalidades CSS que no están contenidas de forma nativa, como funciones, variables... y que nos permitirán programar nuestras hojas de estilos de una manera mucho más ágil.

- Dependencias: **Fontawesome Free**, **Stylelint**, **Babel**, **React.js**, **ESLint**, **Prettier**, **Bootstrap 5**, **Popper.js**

Fontawesome, es un conjunto de herramientas de fuentes e iconos basado en CSS.

Stylelint, es un linter que nos ayudará a escribir mejor código CSS en base a unas reglas predefinidas según una configuración que utilicemos.

Babel, es un transcompilador de JavaScript gratuito y de código abierto que se utiliza principalmente para convertir el código ECMAScript 2015+ (ES6+) en una versión de JavaScript compatible con versiones anteriores que puede ser ejecutada por motores de JavaScript más antiguos.

React.js, es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Mantenido por Facebook y la comunidad de software libre.

ESLint, es una herramienta de análisis de código estático para identificar patrones problemáticos encontrados en el código JavaScript.

Prettier, es un formateador de código con opinión. Impone un estilo consistente analizando su código y reimprimiéndolo con sus propias reglas que tienen en cuenta la longitud máxima de la línea, envolviendo el código cuando es necesario.

Bootstrap 5, es un framework CSS gratuito y de código abierto dirigido al desarrollo web front-end responsive y orientado a los dispositivos móviles.

Popper.js, es una librería ligera utilizada por Bootstrap para posicionar popovers y tooltips.

- Publicación: **Netlify**.

Por último para el despliegue se utilizará **Netlify**, que nos permite vincular nuestra cuenta de **Github**, y el repositorio en cuestión, y ejecutar la compilación en nuestro servidor en cuestión de un comando.

3. Metodologías utilizadas.

Para el desarrollo de esta aplicación web orientada a componentes e interfaces de usuario, hemos utilizado como metodologías CSS, dos metodologías que se complementan muy bien y nos permitirán reducir nuestro código css de una manera recursiva y sustancial.

Las metodologías elegidas son:

- **Mobile First:** es la forma que utilizaremos para construir el diseño, partiendo siempre desde pantallas pequeñas a pantallas más grandes. Por defecto construiremos orientado a dispositivos móviles y abordaremos el resto de comportamientos en función del ancho de la pantalla para definir comportamientos para tablet y desktop.
- **ITCSS:** como base para el desarrollo se escoge este tipo de metodología o arquitectura, que nos permite establecer un pequeño estándar a desarrollar para nuestra aplicación.

Esta metodología nos ayudará a definir determinados tipos de selectores, clases y configuraciones en función, en primer lugar, de la magnitud o alcance, la especificidad y según la claridad respecto a la abstracción.

- **OOCSS:** como arquitectura complementaria hemos elegido OOCSS, para poder realizar abstracción de ciertos elementos que se repiten a lo largo de nuestro proyecto y que tienen una base común, pero que de ninguna manera son ajustes, herramientas, configuraciones genéricas o elementos nativos HTML.

Esta selección junto con el uso de React.js nos permite atomizar los elementos que compondrán nuestras interfaces de usuario y el mantenimiento del mismo.

Como comentábamos comenzaremos construyendo primero en dispositivos móviles, ya que la reorganización de los elementos a otro tipo de pantallas será más sencilla si utilizamos el flow de bloques.

Utilizaremos ITCSS para crear capas jerárquicas que nos aislaran según la especificidad la reutilización de bloques css.

Pero sobre todo nos centraremos en este caso, en el desarrollo de la maquetación orientada a objetos y componentes. Quitando peso innecesario al resto de elementos arquitectónicos css.

Centrándonos en la maquetación orientada a componentes y apoyándonos en Bootstrap conseguiremos reducir gran parte de nuestro código en Sass y así poder plantear mejor una arquitectura que pueda ir creciendo en el tiempo, es decir, hacer el proyecto más mantible.

A continuación se detallan las dependencias que componen la arquitectura mencionada en estos los puntos anteriores. Para llevar a cabo la misma, hemos partido de la configuración de dependencias del boilerplate assets/styles. Los ficheros/dependencias a tener en cuenta por impacto de mayor a menor son:

- main.scss: este archivo carga todos los módulos de la arquitectura ITCSS. En él procedemos a importar cada módulo de dicha arquitectura.

```
/**  
 *! BASE CONTENTS FOR ITCSS  
 *  
 ** SETTINGS - 01_settings"  
 *  
 ** TOOLS - 02_tools  
 *  
 ** GENERIC - 03_generic  
 *  
 ** ELEMENTS - 04_elements  
 *  
 ** OBJECTS - 05_objects  
 *
```

```
** COMPONENTS - 06_components
*
** TRUMPS - 07_trumps
*/
```

- 01-settings: este módulo cargará variables de configuración reutilizables a lo largo de todo el proyecto, en este caso hemos definido, la **fuente** base, su tamaño base, modificando las variables que nos ofrece Bootstrap en Sass.
- 02-tools: este módulo define funciones y mixins globales reutilizables a lo largo de todo el proyecto. En este caso hemos creado mixines propios de media queries reutilizables para componentes y objetos. Además de una función utilizada para el hover del grid de la vista de la home, que es reutilizable en otros componentes.
- 03-generic: este módulo define el conjunto de normas normalizadoras que se ejecutará como primera capa para que podamos partir de elementos iguales sin tener en cuenta el navegador que utilicemos.
- 04-elements: en este módulo daremos estilo base a los diferentes elementos html por defecto. En este caso lo hemos utilizado únicamente para modificar los diferentes tipos de títulos html a nivel global.
- 05-objects: en este módulo realizaremos la arquitectura OOCSS, el patrón de objetos que utilizamos básicamente se compone de vista, contenedor título, contenedor notas, contenedor tarjetas y not found.
- 06-components: esta dependencia se compone de un índice que va importando todos los módulos css relativos a los componentes utilizados en React. Como los componentes en React son reutilizables, este código también.

Tenemos en este caso, componentes como, cards, footer, gallery, hero, modal, note, parallax, y toast. En el caso de modal y toast hemos utilizado las variables de Bootstrap en Sass para pre-configurar y modificar parcialmente estos componentes además de nuestra guía de estilos propia.

- 07-trumps: por último hemos utilizado este módulo como utilidades y helpers que sobreescribir clases y modifican algunos estilos que por base vienen definidos de otra manera genérica. Lo hemos utilizado para setear todos los títulos como fuente con serifa.

4. Desarrollo:

Para explicar el desarrollo completo de este proyecto, hemos dividido esta sección en las siguientes fases, que a continuación resumen el desarrollo del mismo:

1. Descargar boilerplate.

En primer lugar descargamos el boilerplate facilitado para esta práctica. En mi caso realicé previamente un fork del boilerplate de la UOC a mi cuenta de usuario en **Github**.

Podemos descargarlo haciendo git clone en cualquier dependencia de nuestro equipo que deseemos mediante terminal, introduciendo lo siguiente:

`git clone https://github.com/ansango/uoc-boilerplate`

o

`git clone https://github.com/uoc-advanced-html-css/uoc-boilerplate`

2. Instalación y configuración de dependencias.

En primer lugar instalaremos el proyecto mediante el siguiente comando:

`npm i`

Ya tendremos instaladas las dependencias base del boilerplate instaladas. A continuación instalaremos y configuraremos las dependencias **Fontawesome**, **Stylint**, **Babel**, **ESLint**, **Prettier**, **React.js**, **Bootstrap 5**, **Popper.js**.

Para instalar **Fontawesome** introduciremos en nuestro terminal en la raíz del proyecto lo siguiente:

```
npm install --save @fortawesome/fontawesome-free
```

Para utilizar **Fontawesome** en nuestro proyecto necesitaremos importar el módulo en src/assets/scripts/main.js escribiendo lo siguiente:

```
import "@fortawesome/fontawesome-free/css/all.css";
```

Verificaremos en nuestro package.json que se ha instalado correctamente nuestra dependencia:

```
"dependencies": {  
  "@fortawesome/fontawesome-free": "^5.15.2"  
}  
}
```

A continuación procederemos a instalar y configurar **Stylelint**, en primer lugar introduciremos en nuestro terminal, en la raíz del proyecto:

```
npm install --save-dev stylelint-scss stylelint-config-recommended-scss
```

En nuestro package.json deberíamos ver algo como esto:

```
"devDependencies": {  
  ...  
  "stylelint": "^13.12.0",  
  "stylelint-config-recommended-scss": "^4.2.0",  
  "stylelint-scss": "^3.19.0"  
},
```

Posteriormente crearemos un archivo de configuración para Stylelint, ".stylelintrc.json", que contendrá las reglas que utilizaremos para la validación. La base para este archivo es la siguiente:

```
{
  "extends": "stylelint-config-recommended-scss",
  "rules": {}
}
```

Para poder ejecutar y probar toda nuestra configuración debemos añadirlo a nuestro package.json como un nuevo script:

```
"scripts": {
  "build": "npm-run-all clean stylelint parcel:build",
  "stylelint": "stylelint src/**/*.{scss}"
},
```

Para ejecutarlo:

```
npm run stylelint
```

Para agilizar nuestras pruebas hemos configurado un plugin muy útil en nuestro IDE, que se comentará en el punto de **Entorno de desarrollo y configuración**.

A continuación vamos a instalar **React**, para ello, desde terminal introduciremos el siguiente comando:

```
npm i react react-dom react-router-dom
```

Con esto hemos instalado React, React DOM y React Router, para poder trabajar con rutas. Antes de instanciar React y empezar a montar la aplicación necesitaremos instalar **Babel** y su preset para poder trabajar con **JSX**, además, necesitaremos utilizar un plugin para **Parcel**, que nos transforme las urls de las imágenes que importamos como objetos al proyecto, de otra manera no podremos renderizar imágenes de forma dinámica ni importarlas en nuestros archivos de Javascript.

Primero instalaremos **Babel** y sus plugins:

```
npm install --save-dev @babel/core @babel/preset-env @babel/preset-react  
@parcel/transformer-sass
```

A continuación crearemos un archivo de configuración de **Babel**, **babelrc.json**:

```
{  
  "presets": ["@babel/preset-env", "@babel/preset-react"]  
}
```

Después necesitaremos añadir un plugin a Parcel para la importación de imágenes.

Crearemos un archivo .babelrc:

```
{  
  "extends": "@parcel/config-default",  
  "transformers": {  
    "*.{png,jpg,jpeg)": ["@parcel/transformer-raw"]  
  }  
}
```

Ahora ya podemos importar React en el main.js de nuestra aplicación:

```
import React from "react";  
import ReactDOM from "react-dom";  
import App from "./App";  
ReactDOM.render(<App />, document.getElementById("root"));
```

Nuestro package.json debería de contener además de las dependencias mencionadas anteriormente, las siguientes:

```
"devDependencies": {  
  "@babel/core": "^7.13.10",  
  "@babel/preset-env": "^7.13.15",  
  "@babel/preset-react": "^7.13.13",  
  "@parcel/transformer-sass": "^2.0.0-beta.2",  
},
```

```
"dependencies": {  
  "@fortawesome/fontawesome-free": "^5.15.2",  
  "react": "^17.0.2",  
  "react-dom": "^17.0.2",  
  "react-router-dom": "^5.2.0"  
}
```

A continuación vamos a instalar Bootstrap 5, Popper.js y lo importamos en nuestro proyecto:

```
npm install bootstrap@next @popperjs/core
```

Antes de hacer los imports de React en main.js haremos lo siguiente:

```
import * as bootstrap from "bootstrap";
```

Y ya tendremos metido bootstrap en nuestro proyecto.

Por último vamos a instalar y configurar ESLint y Prettier, dos dependencias que nos ayudaran a escribir mejor código en JS (ESLint) y formatearlo (Prettier) a un estándar por proyecto. Esto es muy útil para aprender buenas prácticas y tener una misma escritura cuando el proyecto es desarrollado por un equipo.

Además de que como es la primera vez que utilizamos React, nos ayudará a interiorizar conceptos clave y corregir a buenas prácticas el código que realicemos.

Para instalar **ESLint** y **Prettier**:

```
npm i --save-dev eslint eslint-config-prettier eslint-plugin-import  
eslint-plugin-jsx-a11y eslint-plugin-prettier eslint-plugin-react  
eslint-plugin-react-hooks prettier babel-eslint
```

Con esto instalamos todo las dependencias necesarias para lintear un proyecto moderno en React (Hooks, imports), con Babel y Prettier.

Ahora tenemos que configurar ESLint, creamos en la raíz del proyecto un archivo de configuración, .eslintrc.json:

```
{
  "env": {
    "browser": true,
    "es6": true,
    "node": true
  },
  "extends": [
    "plugin:react/recommended",
    "plugin:jsx-a11y/recommended",
    "plugin:import/recommended",
    "plugin:prettier/recommended"
  ],
  "plugins": ["react-hooks"],
  "globals": {
    "Atomics": "readonly",
    "SharedArrayBuffer": "readonly"
  },
  "parser": "babel-eslint",
  "parserOptions": {
    "ecmaVersion": 2018,
    "sourceType": "module"
  },
  "rules": {
    "react-hooks/rules-of-hooks": "error",
    "react-hooks/exhaustive-deps": "warn"
  },
  "settings": {
    "react": {
      "version": "detect"
    }
  }
}
```

A continuación hacemos lo mismo para Prettier, creamos otro archivo de configuración, prettierrc.json:

```
{  
  "semi": true,  
  "singleQuote": false  
}
```

Ya tenemos instaladas y configuradas todas las dependencias. Ahora nos queda obligar a la compilación a utilizar linter y e interrumpir el build cuando existan fallos y que no se despliegue la aplicación en producción. En el package.json los scripts deberían de quedar de la siguiente manera:

```
"scripts": {  
  "parcel:serve": "parcel serve src/index.html -p 8123 --open",  
  "parcel:build": "parcel build src/index.html --public-url ./ --dist-dir dist  
  --no-source-maps --no-cache",  
  "clean": "rimraf dist .cache .cache-loader .parcel-cache",  
  "dev": "npm-run-all clean parcel:serve",  
  "build": "npm-run-all clean lint stylelint parcel:build",  
  "test": "echo 'Everything is working as expected ✅\nStart working on your project  
by running \\033[1mnpm run dev\\033[0m'",  
  "stylelint": "stylelint src/**/*.{css,scss}",  
  "lint": "eslint src"  
},
```

Podemos utilizar el linter en desarrollo también, pero en mi caso he preferido utilizar los plugins de VSCode tanto para ESLint como para Prettier, que comentaré en el siguiente punto.

Así como stylelint ejecuta una validación de css, lint ejecuta lo mismo para js, por lo que si ambos tienen fallos el build fallará.

3. Entorno de desarrollo y configuración.

Una vez instaladas y configuradas las dependencias procederemos a abrir nuestro proyecto con el IDE que queramos, en este caso se utilizará VSCode.

Dentro de la amalgama de plugins y extensiones que pueden añadirse a VSCode, hemos elegido los siguientes, que nos facilitarán la escritura de código:

- Better Comments: Comentarios coloreados de todo tipo. Personalmente he instalado este plugin para poder documentar de una manera mucho más llamativa toda la documentación y hacerla menos pesada.
- Colorize: para destacar los colores seleccionados en CSS. Es complicado memorizar colores hexadecimales, por lo que es útil tener un selector gráfico de los mismos.
- Mithril Emmet: son snippets inteligentes para crear código de manera rápida en HTML, CSS y Javascript.
- Prettier - Code formatter, nos permite formatear e indentar todo nuestro código, tanto html, js, css. Es instalable como dependencia, pero por el momento no hemos considerado esta opción y hemos preferido adaptarlo en el plugin de nuestro IDE.
- Stylelint, este plugin nos permite ejecutar Stylelint en tiempo real, por lo que en principio no necesitaremos ejecutar el comando de npm en desarrollo. Veremos directamente los fallos en nuestro código y las sugerencias de cambio.
- ES7 React/Redux/GraphQL/React-Native snippets, extensiones simples para React, Redux y Graphql en JS/TS con sintaxis ES7.
- ESLint: La extensión utiliza la biblioteca ESLint instalada en la carpeta abierta del espacio de trabajo. Si la carpeta no proporciona una, la extensión busca una versión de instalación global.

4. Desarrollo.

Para abordar el desarrollo propiamente dicho lo primero que haremos es iniciar git en nuestro proyecto con el siguiente comando:

```
git init
```

Después añadiremos todos los cambios y haremos commit de los mismos:

```
git add *
```

```
git commit -m "first commit"
```

Cambiaremos de rama:

```
git branch -M main
```

Accederemos a Github y crearemos un nuevo repositorio remoto e introduciremos por consola:

```
git remote add origin https://github.com/ansango/band-uoc
```

Por último haremos push del commit establecido:

```
git push -u origin main
```

Una vez creado nuestro repositorio remoto y configurado git podremos empezar a escribir código. Para abordar correctamente este proyecto hemos establecido las siguientes fases:

- HTML: en primer lugar vaciamos el contenido que tengamos en nuestra etiqueta <body> y lo sustituimos por lo siguiente:

```
<body>
  <div id="root"></div>
  <script src=".assets/scripts/main.js"></script>
  <!-- do not remove this line! -->
</body>
```

Esto importará en último lugar el script que carga react y cogerá el div con id root que renderizará nuestra aplicación React.

- Arquitectura CSS:
 - Módulos ITCSS: aquí llevaremos la arquitectura ITCSS explicada en el apartado metodología, que nos ayudará a ir creando las clases selectoras de más reutilizable a menos.
 - Objetos OOCSS: estableceremos la abstracción de objetos y la creación de componentes.
- Javascript: desde el main.js empezaremos a estructurar nuestra arquitectura en React. En nuestro caso, hemos creado una dependencia "App" que contiene un index.js que exporta por defecto el componente App que se renderiza en main.js:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./App";

ReactDOM.render(<App />, document.getElementById("root"));
```

A partir de aquí vamos haciendo code splitting y fragmentando en dependencias Components, Data y Views. La dependencia Components contendrá así mismo otras que albergarán todos los componentes reutilizables de nuestra UI.

La dependencia Views contendrá las vistas que se renderizan en el Router de nuestra aplicación, así podremos delegar el trabajo en vistas y componentes y estas podrán reutilizar dichos componentes con parámetros dinámicos. Esto nos ahorra escribir mucho código html.

Por último Data contiene todos los datos que se utilizarán en componentes y vistas, estos podrían venir de una base de datos, es simplemente un mockeo de datos.

5. Compilación para producción.

Después de haber realizado la fase de codificación, llevaremos a cabo nuestro primer despliegue de la aplicación. Para ello debemos verificar que todo ha ido bien compilando el proyecto. Ejecutaremos en la raíz del proyecto el siguiente comando:

npm run build

Este comando nos generará en la raíz del proyecto, una dependencia llamada **dist**, donde podremos ver toda la compilación. Podremos ver que todo funciona abriendo el index.html en cualquier navegador. Esta dependencia está lista para ser servida en un servidor.

Cabe añadir que utilizando React Router y la configuración que tenemos por defecto (BrowserRouter), si abrimos el index.html en local, el cambio entre vistas no funcionará, ya que BrowserRouter está preparado para un entorno de webservice. Si queremos ver que todo funciona correctamente incluyendo el Router, tendremos que cambiar el tipo de router por HashRouter o Memory Router. Esto se debería hacer en App/index.js de la siguiente manera:

cambiamos:

```
import { BrowserRouter as Router, Switch, Route } from "react-router-dom";
```

Por:

```
import { HashRouter as Router, Switch, Route } from "react-router-dom";
```

O:

```
import { MemoryRouter as Router, Switch, Route } from "react-router-dom";
```

Volvemos a ejecutar **npm run build**, y ahora si, en dist/index.html podremos navegar entre rutas sin problema.

6. Publicación.

Por último una vez compilado el proyecto y viendo que el proceso ha ido correctamente, vamos a desplegarlo en un servidor.

Para ello hemos utilizado **Netlify**, que nos ayudará a automatizar el sistema de despliegues de sitios estáticos.

Antes de pushear los cambios en el repositorio, añadiremos un pequeño archivo de configuración para **Netlify**, que nos corregirá el error por defecto al navegar en rutas no existentes en la aplicación (404 de Netlify por nuestra vista 404). Eso sucede en cualquier SPA. Por lo que haremos lo siguiente:

- Crearemos un archivo “netlify.toml” en la raíz del proyecto.
- Copiamos y pegamos lo siguiente:

```
[[redirects]]  
from = "/"  
to = "/index.html"  
status = 200
```

Por aclarar esta parte, comentar que en el componente App (App/index.js), donde llamamos al router, tenemos un componente para todas aquellas rutas que no sean las que están definidas (devuelve un 404 al uso):

```
const NotFound = React.lazy(() => import("./Views/404/index"));
```

```
...
```

```
<Route path="*" render={() => <NotFound />}></Route>
```

A continuación deberemos hacer push de los últimos cambios. Para configurar **Netlify**, seguiremos los siguientes pasos:

1. Crearemos un nuevo sitio seleccionando el botón: “**New site from git**”
2. En “**Continuous Deployment**” seleccionaremos **Github**.
3. De la lista de repositorios seleccionaremos en que creamos anteriormente.

4. Seleccionamos la rama **main**.
5. Escribimos como comando de build: **npm run build**.
6. Escribimos como directorio de publicación: **dist/**
7. Y por último seleccionamos el botón “**Deploy Site**”

5. Publicación.

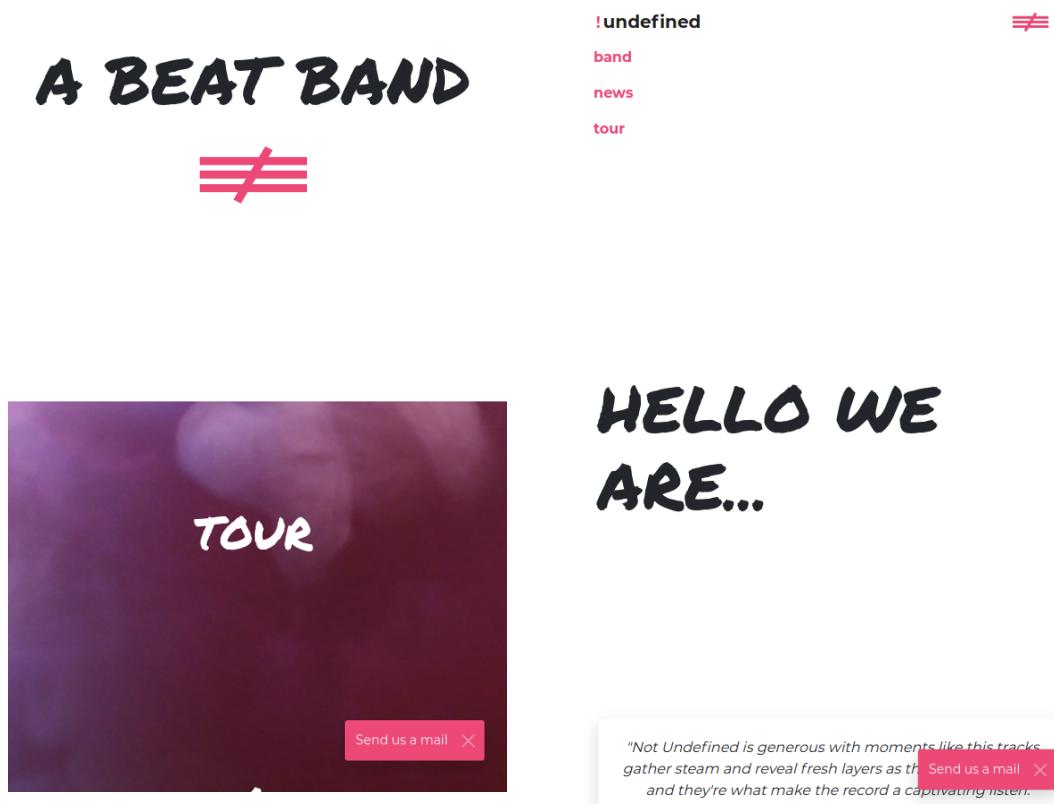
Como comentamos en el último subpunto del punto anterior, después del despliegue tendremos accesible nuestra aplicación en una url real.

En este caso hemos generado la siguiente en **Netlify** desde el repositorio de **Github**:

<https://github.com/ansango/band-uoc>

<https://band-uoc.netlify.app/>

El resultado final es el siguiente:





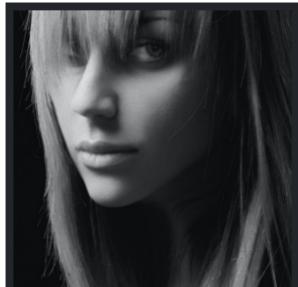
!undefined



band

news

tour



[Send us a mail](#)



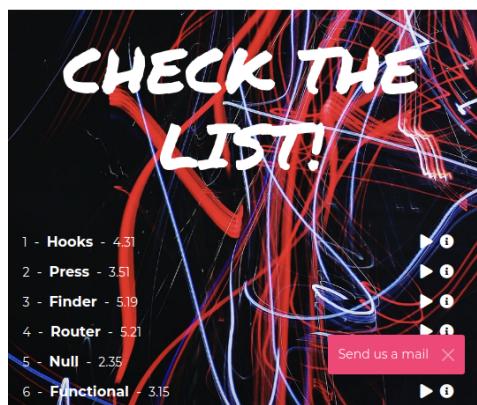
FAR FAR AWAY, BEHIND THE WORD MOUNTAINS, FAR FROM THE COUNTRIES VOKALIA AND CONSONANTIA, THERE L
[Send us a mail](#)

THE COUNTRIES VOKALIA AND CONSONANTIA

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean. Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean. Far far away, behind the word mountains

[Previous](#) [1](#) [2](#) [3](#) [Next](#)



!undefined

We are not undefined an electronic collective from Berlin,Germany. We make noise and beautiful things. For bookings or any other information you can send us an email to:

info@notundefined.com

Streaming

- Spotify
- Apple Music
- Youtube Music

Next date

Saturday 02 September 2021
Kindl-Bühne Wuhlheide,
Berlin, Germany

[Checkout all new dates!](#)

[Send us a mail](#)

!undefined

band news tour



!undefined

band

news

tour



Jun 2 Thu

Charles Krug Winery | 19:30

St. Helena, CA, United States

Tickets

Send us a mail

St. Helena, CA, United States

Tickets

Jun 2 Thu

Charles Krug Winery | 19:30

St. Helena, CA, United States

Tickets

Jun 2 Thu

Charles Krug Winery | 19:30

St. Helena, CA, United States

Tickets



Send us a mail

Tickets

Jun 2 Thu

Charles Krug Winery | 19:30

SAY WHAT YOU LIKE

Email address

Email

Name

Name

Say something!

I agree to the [privacy policy](#).

Send

Send us a mail

A BEAT BAND



[Send us a mail](#) X

!undefined

[band](#) [news](#) [tour](#)

HELLO WE ARE...

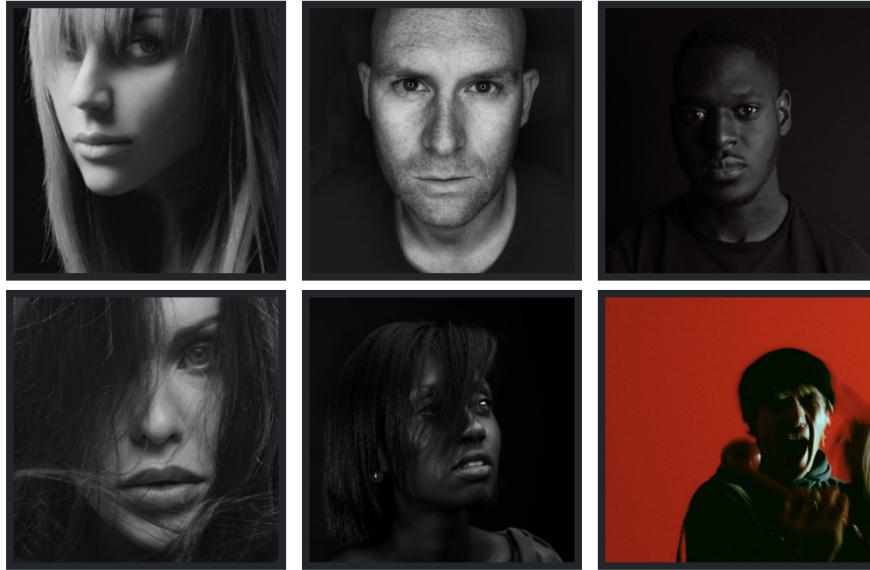
"Not Undefined is generous with moments like this tracks gather steam and reveal fresh layers as they move forward and they're what make the record a captivating listen."

—**Joe Colly**, *Pitchfork*

"Fusing elements of free jazz, breakbeat, acid house, dub, ambient, and more, the Not Undefined first full-length album is simultaneously comforting and destabilizing."

—**Nathan Smith**, *The Guardian*

[Send us a mail](#) X



WE WERE BORN IN BERLIN, WE MAKE NOISE, WE LIKE NOISE, WE CAN'T STOP MAKING NOISE. SOMETIMES WE DO QUIETER THINGS. BUT ONLY WHEN WE

[Send us a mail](#)

!undefined

[band](#) [news](#) [tour](#)



FAR FAR AWAY, BEHIND THE WORD MOUNTAINS, FAR FROM THE COUNTRIES VOKALIA AND CONSONANTIA, THERE LIVE THE BLIND TEXTS.

[Send us a mail](#)

Item Three

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean.

FAR FAR AWAY, BEHIND THE WORD MOUNTAINS, FAR FROM THE COUNTRIES VOKALIA AND CONSONANTIA

Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove right at the coast of the Semantics, a large language ocean. Far far away, behind the word mountains, far from the countries Vokalia and Consonantia, there live the blind texts. Separated they live in Bookmarksgrove



!undefined

band news tour

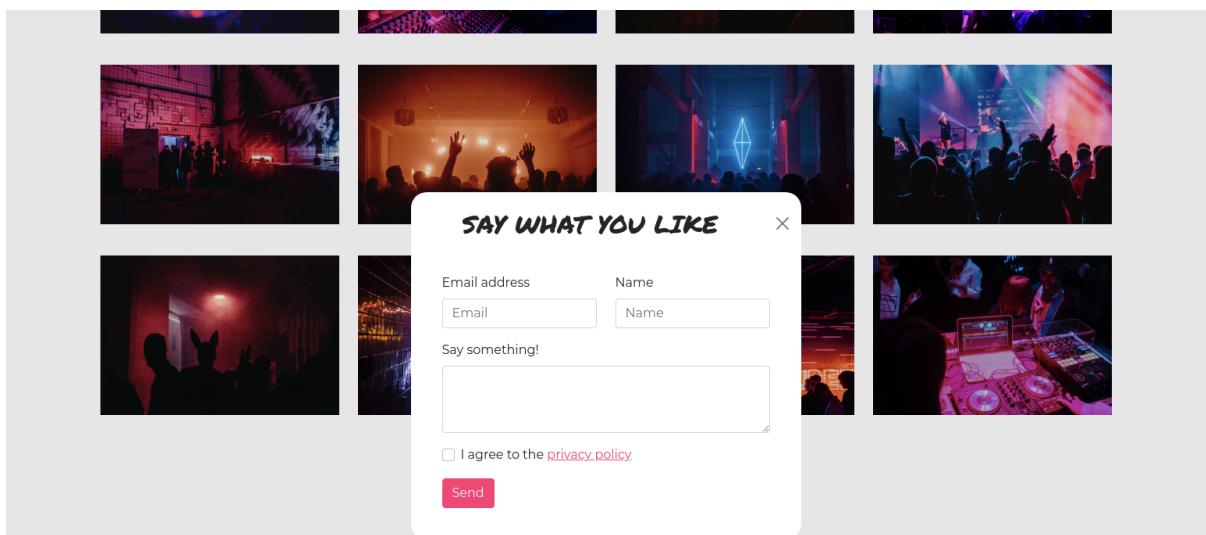


	Jun 2 Thu	Charles Krug Winery 19:30	St. Helena, CA, United States	Tickets
	Jun 2 Thu	Charles Krug Winery 19:30	St. Helena, CA, United States	Tickets
	Jun 2 Thu	Charles Krug Winery 19:30	St. Helena, CA, United States	Tickets
	Jun 2 Thu	Charles Krug Winery 19:30	St. Helena, CA, United States	Tickets
	Jun 2 Thu	Charles Krug Winery 19:30	St. Helena, CA, United States	Tickets Send us a mail X

Upcoming Events 19:30		View All Events	Next	
	Jun 2 Thu	Charles Krug Winery 19:30	St. Helena, CA, United States	Tickets
	Jun 2 Thu	Charles Krug Winery 19:30	St. Helena, CA, United States	Tickets



Send us a mail



Streaming

-

Next date

Saturday 02 September 2021
Kindl-Bühne Wuhlheide, Berlin
Germany

[Checkout all new dates!](#)

!undefined

We are not undefined an electronic collective from Berlin,Germany. We make noise and beautiful things. For bookings or any other information you can send us an email to: Send us

info@notundefined.com

[Send us a mail](#)