

2020-  
2021



RehabTime

E89835

Cloud Computing: Fund. & Infra.  
Máster Ingeniería Informática

## Contenido

0. Motivación .....	4
1. Pasos previos.....	5
2. Creación de la aplicación.....	7
3. Historias de usuario.....	9
4. Estructura y componentes .....	10
4.1. Fuentes.....	10
4.2. Pruebas.....	12
5. Visualización.....	15
6. Lighthouse .....	22
7. Docker .....	23
7.1. Pasos previos.....	23
7.2. Creación ficheros e imagen.....	24
7.3. Ejecución de imagen .....	26
7.4. Docker Hub.....	27
8. CI/CD .....	29
8.1. Netlify .....	29
8.2. Travis .....	34
8.3. Lighthouse Netlify .....	38
8.4. Lighthouse Surge .....	38
9. API .....	39
9.1. Postman .....	39

## Ilustraciones

Ilustración 1: Simple React Snippets (e.p.) .....	5
Ilustración 2: Incluir lenguaje JavaScript (e.p.) .....	5
Ilustración 3: Instalación de react-router-dom (e.p.) .....	6
Ilustración 4: Instalación de styled-components (e.p.) .....	6
Ilustración 5: Creación de la aplicación rehabtime con npx (e.p.) .....	7
Ilustración 6: Finalización y apertura de la aplicación rehabtime (e.p.) .....	7
Ilustración 7: Hola mundo de React (e.p.) .....	8
Ilustración 8: Servidor json funcionando en puerto 8000 (e.p.) .....	11
Ilustración 9: Comprobación dependencias para testear (e.p.) .....	13
Ilustración 10: Ejecución de test de la aplicación (e.p.) .....	14
Ilustración 11: Página Home de la aplicación (e.p.) .....	15
Ilustración 12: Detalles de una entrada de la aplicación (e.p.) .....	15
Ilustración 13: Detalles de la base de datos (e.p.) .....	16
Ilustración 14: Detalles de contacto de la aplicación (e.p.) .....	16
Ilustración 15: Sección Buscar en la aplicación (e.p.) .....	17
Ilustración 16: Ejemplo de búsqueda en la aplicación (e.p.) .....	17
Ilustración 17: Sección Añadir de la aplicación (e.p.) .....	18
Ilustración 18: Requisitos para la creación de una entrada (e.p.) .....	18
Ilustración 19: Error mostrado al introducir una URL no válida (e.p.) .....	19
Ilustración 20: Mensaje de carga de la aplicación (e.p.) .....	19
Ilustración 21: Ejemplo de error al traer los datos de la base de datos (e.p.) .....	20
Ilustración 22: Ejemplo modo noche - Home (e.p.) .....	20
Ilustración 23: Ejemplo modo noche - Crear (e.p.) .....	21
Ilustración 24: Evaluación Lighthouse de la aplicación (e.p.) .....	22
Ilustración 25: Configuración Docker (e.p.) .....	23
Ilustración 26: Extensión Docker en VS (e.p.) .....	23
Ilustración 27: Añadiendo Ficheros Docker (e.p.) .....	24
Ilustración 28: Edición del fichero Dockerfile (e.p.) .....	24
Ilustración 29: Creación Imagen Docker (e.p.) .....	25
Ilustración 30: Imagen de Docker rehabtime creada (e.p.) .....	25
Ilustración 31: Contenedor ejecutando la imagen Docker rehabtime:latest (e.p.) .....	26
Ilustración 32: Ejecución aplicación en contenedor Docker (e.p.) .....	26
Ilustración 33: Etiqueta y push de la imagen Docker (e.p.) .....	27
Ilustración 34: Push terminado de la imagen Docker (e.p.) .....	27
Ilustración 35: Imagen en Docker Hub (e.p.) .....	28
Ilustración 36: Detalles imagen en Docker Hub (e.p.) .....	28
Ilustración 37: Instalación Netlify (e.p.) .....	29
Ilustración 38: Construcción del proyecto (e.p.) .....	30
Ilustración 39: Vinculación VS-Github (e.p.) .....	30
Ilustración 40: Pop-up confirmación uso Github (e.p.) .....	31
Ilustración 41: Autorizar VS a acceder a GitHub (e.p.) .....	31
Ilustración 42: Detalles de la autorización VS-GitHub (e.p.) .....	31
Ilustración 43: Contraseña GitHub (e.p.) .....	31
Ilustración 44: Mensaje de confirmación de autenticación (e.p.) .....	32
Ilustración 45: Creación de nuevo sitio Netlify (e.p.) .....	32
Ilustración 46: Selección de repositorio Netlify (e.p.) .....	32

Ilustración 47: Despliegue Netlify (e.p.) .....	33
Ilustración 48: Ejecución de la aplicación en Netlify (e.p.) .....	33
Ilustración 49: Cambio configuración Netlify (e.p.) .....	34
Ilustración 50: Instalación surge.sh (e.p.) .....	35
Ilustración 51: Build de la aplicación (e.p.) .....	35
Ilustración 52: Configuración surge de la aplicación (e.p.) .....	35
Ilustración 53: Aplicación ejecutada en Surge (e.p.) .....	36
Ilustración 54: Acceso a GitHub por Travis (e.p.) .....	36
Ilustración 55: Variables de entorno Travis (e.p.) .....	36
Ilustración 56: Construcción Travis (e.p.) .....	37
Ilustración 57: Construcción Travis completada (e.p.) .....	37
Ilustración 58: Notificación construcción Travis completada (e.p.) .....	37
Ilustración 59: Lighthouse Netlify (e.p.) .....	38
Ilustración 60: Lighthouse Surge (e.p.) .....	38
Ilustración 61: Método GET de la API (e.p.) .....	39
Ilustración 62: Método POST de la API (e.p.) .....	40
Ilustración 63: Método PUT de la API (e.p.) .....	40
Ilustración 64: Resultado método PUT de la API (e.p.) .....	41
Ilustración 65: Método DELETE de la API (e.p.) .....	41

## 0. Motivación

Este documento recoge la aplicación RehabTime en React de la asignatura Cloud Computing: Fundamentos e Infraestructuras, del Máster de Ingeniería Informática en la Universidad de Granada, para el curso 2020-2021.

Inicialmente consideraremos los pasos previos necesarios, tales como configuración e instalación de extensiones.

En el apartado segundo, comprenderemos como crear la aplicación y como ejecutar el “¡Hola mundo!” de React.

En el apartado tercero, analizaremos las historias de usuario.

En el apartado cuarto, estudiaremos la estructura y los componentes de la aplicación, con una breve descripción sobre la funcionalidad de cada uno.

En el apartado quinto, observaremos diferentes capturas de las funcionalidades de la aplicación.

En el apartado sexto, observaremos el rendimiento usando la herramienta Lighthouse.

En el apartado séptimo, veremos la configuración y pasos necesarios para crear una imagen de Docker y subirla a Docker Hub.

En el apartado octavo, examinaremos dos métodos de integración continua, Netlify y Travis CI.

En el apartado noveno, comprobaremos las llamadas a la API usando Postman.

## 1. Pasos previos

En este apartado veremos las herramientas necesarias para la realización de la aplicación.

En nuestro caso, se ha elegido el programa Visual Studio Code (VS), en el que añadimos la extensión mostrada en la Ilustración 1: Simple React Snippets v1.2.3, publicada por Burke Holland.

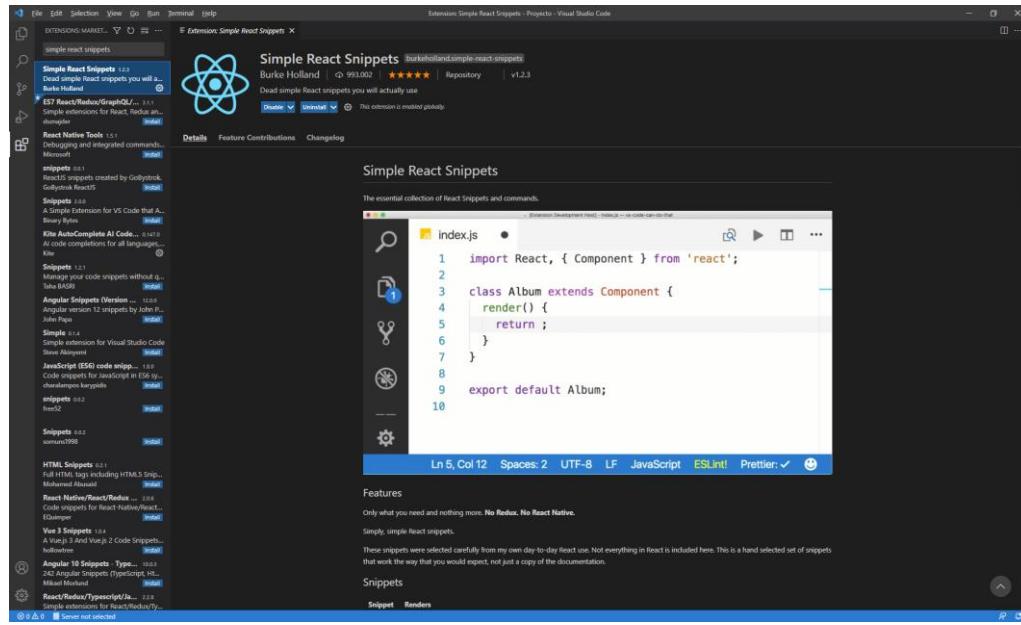


Ilustración 1: Simple React Snippets (e.p.)

A continuación, incluimos las abreviaciones de JavaScript, dado que no están soportadas por defecto. Esto lo hacemos como muestra la Ilustración 2.

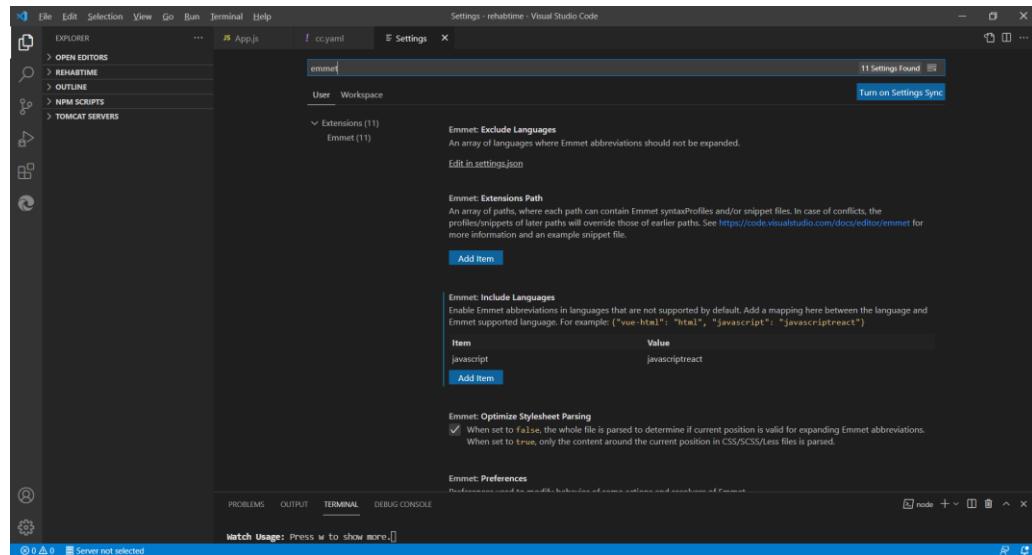
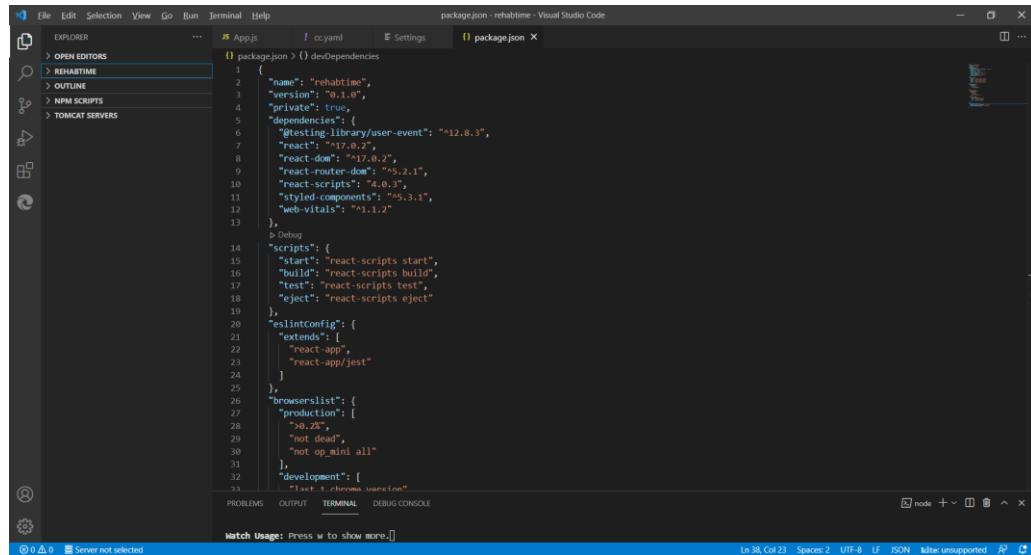


Ilustración 2: Incluir lenguaje JavaScript (e.p.)

Por último, instalamos la extensión *react-router-dom* para poder redirigir las distintas páginas de las entradas en el blog. Podemos comprobar la correcta instalación al incluirse automáticamente en el fichero *package.json* como muestra la Ilustración 3. Nótese que hay que importarlo en el fichero *App.js*.



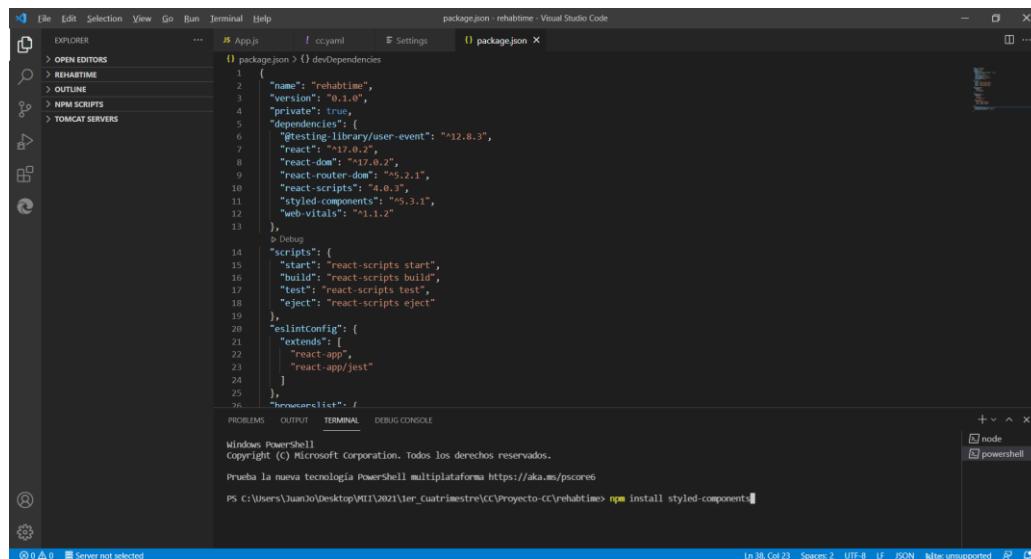
```

{
  "name": "rehabtime",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "testing-library/user-event": "^12.8.3",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-router-dom": "^5.2.1",
    "react-scripts": "4.0.3",
    "styled-components": "5.3.1",
    "web-vitals": "^1.1.2"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version"
    ]
  }
}

```

Ilustración 3: Instalación de *react-router-dom* (e.p.)

Usaremos también el paquete *styled components*, instalándolo desde la terminal con npm, que utilizaremos para modo noche:



```

{
  "name": "rehabtime",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "testing-library/user-event": "^12.8.3",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-router-dom": "^5.2.1",
    "react-scripts": "4.0.3",
    "styled-components": "5.3.1",
    "web-vitals": "^1.1.2"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": []
}

```

Windows PowerShell  
Copyright (C) Microsoft Corporation. Todos los derechos reservados.  
Prueba la nueva tecnología PowerShell multiplatforma <https://aka.ms/powershell>  
PS C:\Users\Juanjo\Desktop\MII\2021\ter\_Cuarto trimestre\CC\Proyecto CC\rehabtime> npm install styled-components

Ilustración 4: Instalación de *styled-components* (e.p.)

## 2. Creación de la aplicación

En este apartado veremos los comandos para crear la aplicación.

Abrimos una terminal, y creamos la aplicación con *npx* como muestra la Ilustración 5. Una vez finalizado, veremos el mensaje de finalizado con éxito y, a continuación, abrimos VS mediante la terminal, tal y como muestra la Ilustración 6.

```
npm install react react-dom react-scripts cra-template
Microsoft Windows [Versión 10.0.19043.1165]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\JuanJo>cd C:\Users\JuanJo\Desktop\MII\2021\1er_Cuatrimestre\CC\Proyecto-CC

C:\Users\JuanJo\Desktop\MII\2021\1er_Cuatrimestre\CC\Proyecto-CC>npx create-react-app rehabtime
npm notice
npm notice New minor version of npm available! 7.15.1 -> 7.21.1
npm notice Changelog: https://github.com/npm/cli/releases/tag/v7.21.1
npm notice Run npm install -g npm@7.21.1 to update!
npm notice

Creating a new React app in C:\Users\JuanJo\Desktop\MII\2021\1er_Cuatrimestre\CC\Proyecto-CC\rehabtime.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
[██████████] - idealTreetypescript: sill fetch manifest tslib@^1.8.1
```

Ilustración 5: Creación de la aplicación *rehabtime* con *npx* (e.p.)

```
Símbolo del sistema
Success! Created rehabtime at C:\Users\JuanJo\Desktop\MII\2021\1er_Cuatrimestre\CC\Proyecto-CC\rehabtime
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd rehabtime
  npm start

Happy hacking!

C:\Users\JuanJo\Desktop\MII\2021\1er_Cuatrimestre\CC\Proyecto-CC>code .

C:\Users\JuanJo\Desktop\MII\2021\1er_Cuatrimestre\CC\Proyecto-CC>cd rehabtime
C:\Users\JuanJo\Desktop\MII\2021\1er_Cuatrimestre\CC\Proyecto-CC\rehabtime>code .
```

Ilustración 6: Finalización y apertura de la aplicación *rehabtime* (e.p.)

Una vez abierto, procedemos a eliminar ficheros no necesarios y a ejecutar la versión de prueba (el “hola mundo” de React), quedando como muestra la Ilustración 7:

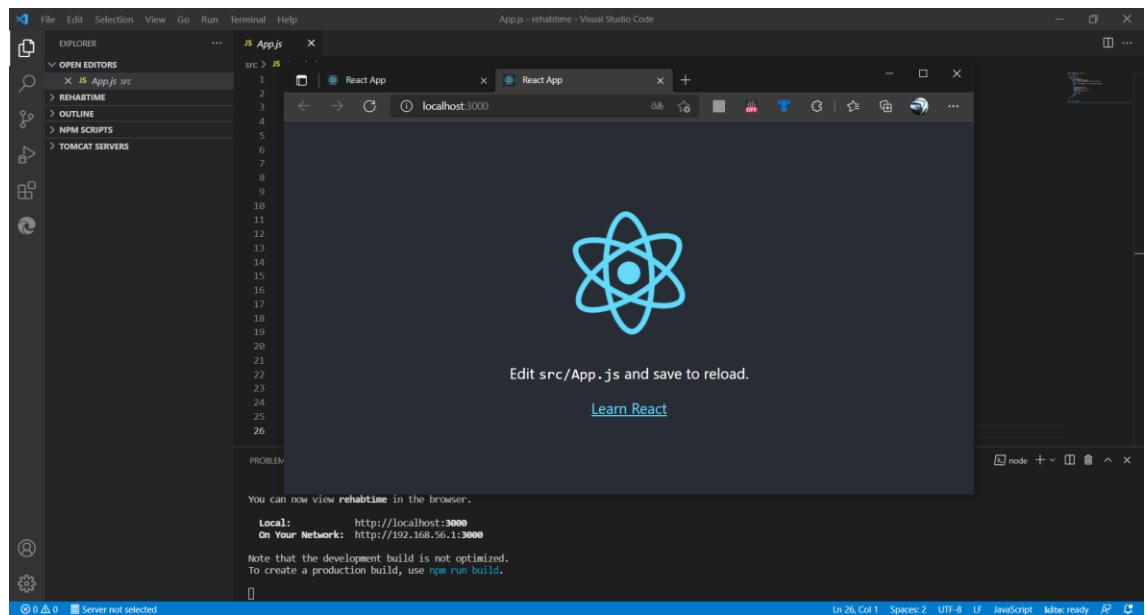


Ilustración 7: Hola mundo de React (e.p.)

### 3. Historias de usuario

En este apartado veremos las historias de usuario que se usan para la realización de la aplicación.

#### US1 - Alicia

Alicia trabaja a jornada completa en una oficina de secretaria, estando por la mañana trabajando además frente al público. Al cabo de unas semanas, empieza a tener dolores de espalda. Es por ello, que decide buscar en Google ejercicios para aliviar el dolor, llegando a RehabTime. En ella, encuentra varios ejercicios que le ayudan a descargar la tensión posicional como resultado de tantas horas frente al ordenador. Al estar tan satisfecha con los resultados, clica Me gusta en el post de la web.

#### US2 - Ernesto

Ernesto es fontanero en Granada. El martes, lo llaman para arreglar una fuga en una tubería en el baño de la familia García. Al ver el problema, sabe que tiene que cambiar el codo del desague, para lo que coge su llave grifa y, al desapretar, se hace un esguince de muñeca. Además de ir a fisioterapia, hace en su casa ejercicios de rehabilitación porque Luis, su fisio, le recomienda los ejercicios de RehabTime. Después de una semana, nota la mejoría, comentando su satisfacción con los resultados.

#### US3 - Luis

Luis se graduó de fisioterapeuta en la Universidad de Granada en el año 2020. Actualmente trabaja en MutuaGranada con pacientes del área metropolitana de distinta ocupación profesional. Un día encuentra una aplicación online que recomienda ejercicios de rehabilitación antidolor de todo el cuerpo. Se da cuenta que además de antidolor, sería bueno poner ejercicios de recuperación muscular, por lo que contacta con el propietario con sus ideas.

#### US4 - Julián

Julián es un estudiante de módulo de informática de Granada. En sus ratos libres, le encanta jugar al LOL y Fornite. Sus familiares le recuerdan que ha de hacer dieta y ejercicio, puesto que ha ganado unos kilos. Julián encuentra la aplicación, y aunque hay varias entradas para ejercitarse abdominales, el intenta acceder a una URL inválida, obteniendo el correspondiente mensaje de error.

## 4. Estructura y componentes

En este apartado veremos los distintos ficheros creados y sus utilidades.

### 4.1. Fuentes

En la carpeta fuentes (*src*) tenemos:

1. App.js

En este fichero es la raíz de la aplicación, en el controlamos el diseño y la navegación general. Todos los componentes tienen acceso.

2. Announcer.js

En este fichero pondremos lo necesario para poder buscar mientras escribimos (*search as you type*).

3. BlogDetails.js

En este fichero pondremos los detalles de cada entrada del blog, tendremos en especial consideración el identificador de la entrada. Esto permitirá usar la API para borrar y actualizar un elemento, por ejemplo.

4. BlogList.js

En este fichero está recogida la lista de todas las entradas disponibles en el blog. Para ello, usaremos props, dado que es más reusable, dado que los props permiten pasar datos de un parent a un hijo.

5. Contact.js

En este fichero vemos información sobre la aplicación.

6. Create.js

En este fichero crearemos las nuevas entradas al blog.

7. data/db.json

Este fichero se usará como base de datos, usando un servidor json en el puerto 8000 que tendremos que lanzar en una terminal desde VS como muestra la Ilustración 8. Este fichero incluirá todas las entradas del blog. Se usa esta forma para simplificar la instalación y configuración de una base de datos de un proveedor como MySQL o MongoDB.

```

{
  "blogs": [
    {
      "title": "¿Qué es Lorem Ipsum?",
      "body": "Lorem Ipsum es simplemente texto de relleno de la industria de la impresión y la composición tipográfica. Lorem Ipsum ha sido el texto de relleno estándar de la industria desde la década de 1500, cuando un impresor se puso en la galera y decidió hacer un libro de muestras tipográficas. Ha sobrevivido no solo a cinco siglos, sino también al salto a la composición tipográfica electrónica, permaneciendo esencialmente sin cambios. Se popularizó en la década de 1960 con el lanzamiento de hojas de Letraset que contenían pasajes de Lorem Ipsum.", "author": "Alasdair PageMaker que incluía versiones de Lorem Ipsum.",
      "likes": 1,
      "comments": [
        {
          "title": "Ejemplo de comentario 1",
          "body": "Juanjo",
          "author": "Juanjo",
          "likes": 0,
          "image": "default"
        }
      ],
      "id": 1
    },
    {
      "title": "De dónde viene?",
      "body": "Contemporáneamente a la creencia popular, Lorem Ipsum no es simplemente un texto aleatorio. Tiene sus raíces en una pieza de la literatura latina clásica del 45 a. C., por lo que tiene más de 2000 años. Richard McClintock, profesor de latín en Hampden-Sydney College en Virginia, buscó una de las palabras latinas más oscuras, 'consecetur', de un pasaje de Lorem Ipsum, y revisando las citas de la palabra en la literatura clásica, descubrió la fuente indudable. Lorem Ipsum proviene de las secciones 1.10.32 y 1.10.33 de 'de Finibus Bonorum et Malorum' (los extremos del bien y del mal) de Cicerón, escrito en el año 45 a. C. Este libro es un tratado de filosofía de la ética, muy popular durante el Renacimiento. La primera línea de Lorem Ipsum, 'Lorem ipsum dolor sit amet . . .', proviene de una línea en la sección 1.10.32. El fragmento entero de 'de Finibus Bonorum et Malorum' de Cicerón también se reproduce y continúa para los interesados. Las secciones 1.10.32 y 1.10.33 de 'de Finibus Bonorum et Malorum' de Cicerón también se reproducen en su forma original exacta, acompañadas de versiones en inglés de la traducción de",
      "author": "H. Rackham",
      "likes": 0,
      "comments": [
        {
          "title": "Juanjo",
          "body": "Juanjo",
          "author": "Juanjo",
          "likes": 0,
          "image": "default"
        }
      ],
      "id": 2
    }
  ]
}

```

Ilustración 8: Servidor json funcionando en puerto 8000 (e.p.)

## 8. GlobalStyles.js

En este fichero definimos un componente y le asignamos un fondo y un color, con lo que, cuando cliquemos el botón noche, se hará la transición en el tiempo deseado (0.5 s).

## 9. Home.js

Pantalla de bienvenida a la página.

## 10. Index.css

En este fichero están todos los estilos de la aplicación.

## 11. NavBar.js

Barra superior de la página, con título y los botones:

- Home.
- Contacto.
- Buscar.
- Crear blog.
- Modo noche.

En este fichero usaremos “*Link to*” en vez de “*a href*” para enlazar las páginas, mejorando así la eficiencia de la aplicación.

## 12. NotFound.js

En este fichero pondremos una página para cuando accedamos a cualquier URL no contemplada en la aplicación, es decir, el error 404.

## 13. Search.js

En este fichero haremos las búsquedas de la aplicación.

#### 14. Theme.js

En este fichero definimos los objetos claro y oscuro, con los colores que usará el fondo de la aplicación.

#### 15. useFetch.js

En este fichero tenemos un *custom hook*, es decir, hacemos la lógica externa para recoger los datos, haciendo el código más reusable, poniendo el componente en un fichero aparte para poder reutilizarlo exportándolo, en vez de tener que reescribirlo cada vez.

Además, usaremos una regla de limpiado para que no se produzcan alertas en caso de cambiar muy rápido dentro de la aplicación de una página a otra.

### 4.2. Pruebas

Para las pruebas de la aplicación, usaremos *React Testing Library* y Jest. Usaremos los comandos siguientes para instalarlos, respectivamente:

- npm install --save-dev @testing-library/react
- npm install --save-dev @testing-library/jest-dom

La biblioteca de pruebas de React fue creada por Kent Dodds y cuenta con un amplio apoyo en la comunidad de desarrolladores. Esta librería es un conjunto de utilidades de React DOM y permite probar de manera sencilla los componentes y simular el comportamiento de usuario. Entre sus ventajas está la consulta de elementos dentro de textos, etiquetas, etc., el disparo de cualquier evento y la espera de que aparezca un elemento.

Jest es el marco de trabajo más popular para pruebas con más de 16 millones de descargas a la semana. Fue creado y es mantenido por Facebook. Además, también ha sido adoptado por Airbnb, Uber y otras empresas. Jest viene con funciones de ejecución de pruebas y aserción. Entre sus mayores ventajas está su rapidez, realización de pruebas de métodos instantáneos, paralelización y asíncronos, simulación de funciones, sintaxis estándar y gran compatibilidad.

Podemos confirmar la instalación de los mismos mirando las nuevas dependencias añadidas en en *package.json* como muestra la Ilustración 9.

```

{
  "name": "rehabtime",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.14.1",
    "@testing-library/react": "^11.2.7",
    "@testing-library/user-event": "12.8.3"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  }
}

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/powershell
PS C:\Users\Juanjo\Desktop\MTI\2021\ter_cuatrimestre\CC\Proyecto-CC\rehabtime> npm install --save-dev @testing-library/react
npm WARN idealTree Removing dependencies @testing-library/react in favor of devDependencies @testing-library/react
[...]
V\idealTree:rehabtime: sill idealTree buildDeps

```

Ilustración 9: Comprobación dependencias para testear (e.p.)

Para ejecutar los test, usaremos el comando *npm run test* en la terminal.

En la carpeta pruebas (*test*) tenemos:

1. App.test.js

En este fichero comprobaremos que están presentes los elementos de la barra de navegación.

2. Contact.test.js

En este fichero comprobaremos que están presentes los elementos de contacto.

3. Create.test.js

En este fichero comprobaremos que están presentes los elementos de creación de entradas en la aplicación.

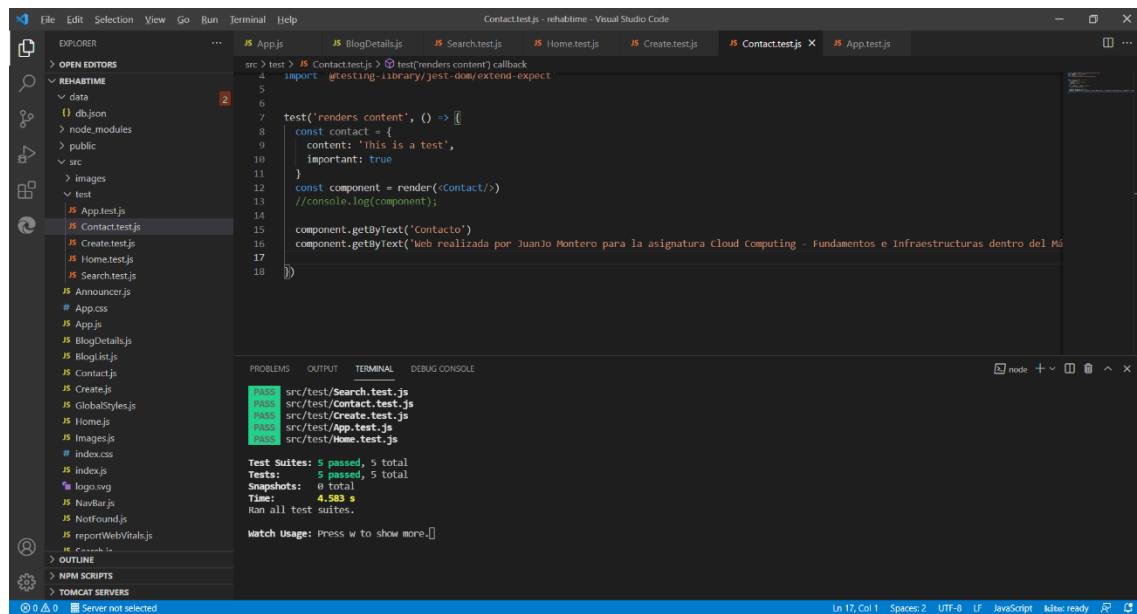
4. Home.test.js

En este fichero comprobaremos el mensaje de carga inicial de datos desde la base de datos de la aplicación.

5. Search.test.js

En este fichero comprobaremos que podemos realizar una búsqueda en la aplicación.

La Ilustración 10 muestra la ejecución de las pruebas anteriores.



*Ilustración 10: Ejecución de test de la aplicación (e.p.)*

## 5. Visualización

En este apartado veremos la visualización de la aplicación.

Comenzaremos por la página principal Home, como muestra la Ilustración 11.

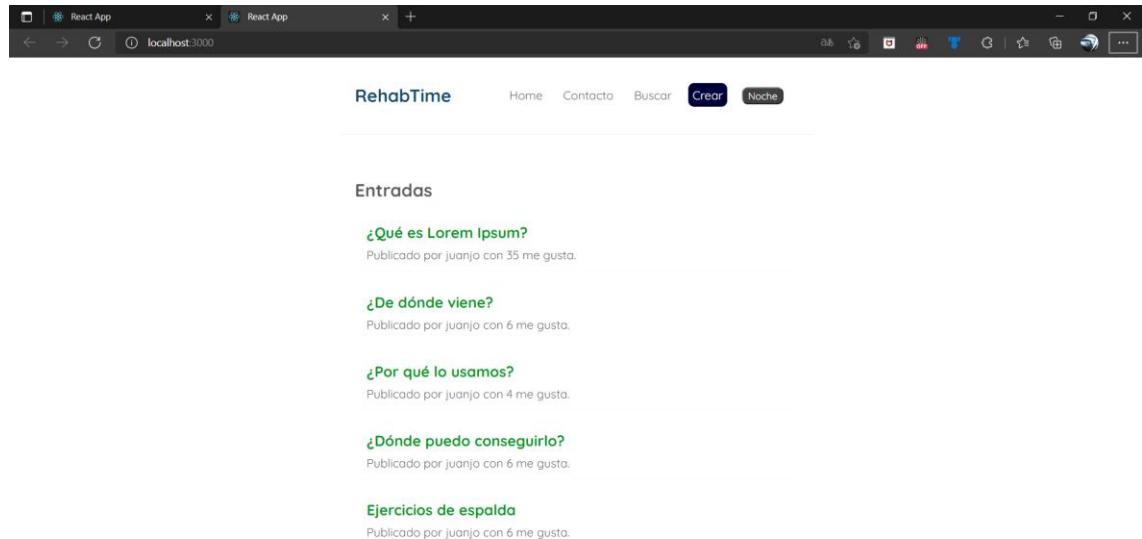


Ilustración 11: Página Home de la aplicación (e.p.)

Si clicamos en cualquiera de las entradas, nos redirige a los detalles de la misma, teniendo el título, autor, imagen, descripción, me gusta y comentarios. Además, podemos clicar en Me gusta, añadir comentarios y/o Eliminar la entrada, como muestra la Ilustración 12.

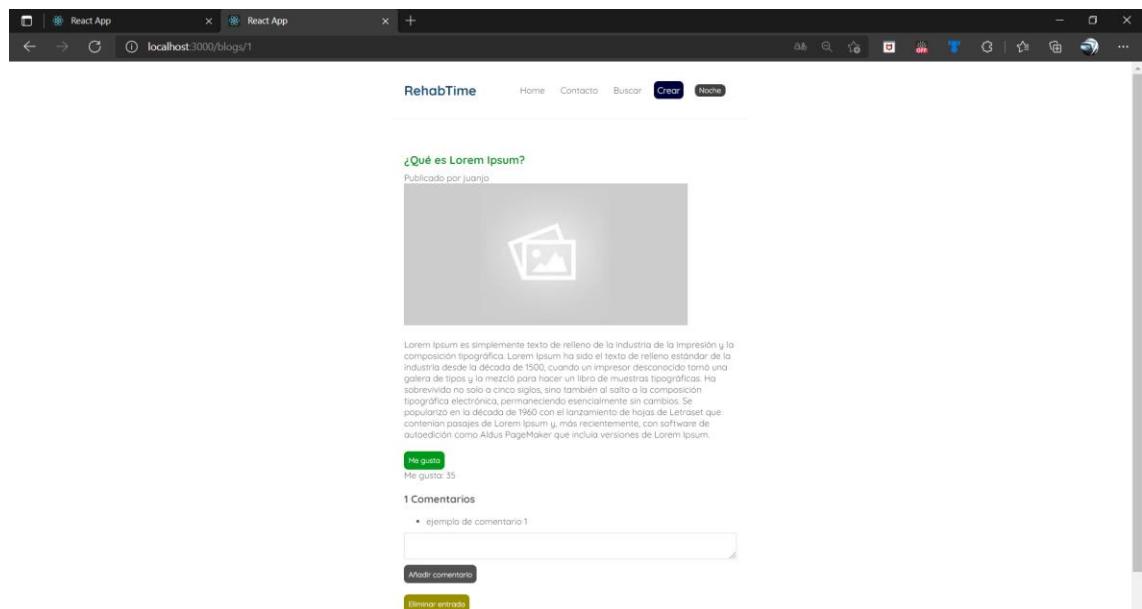
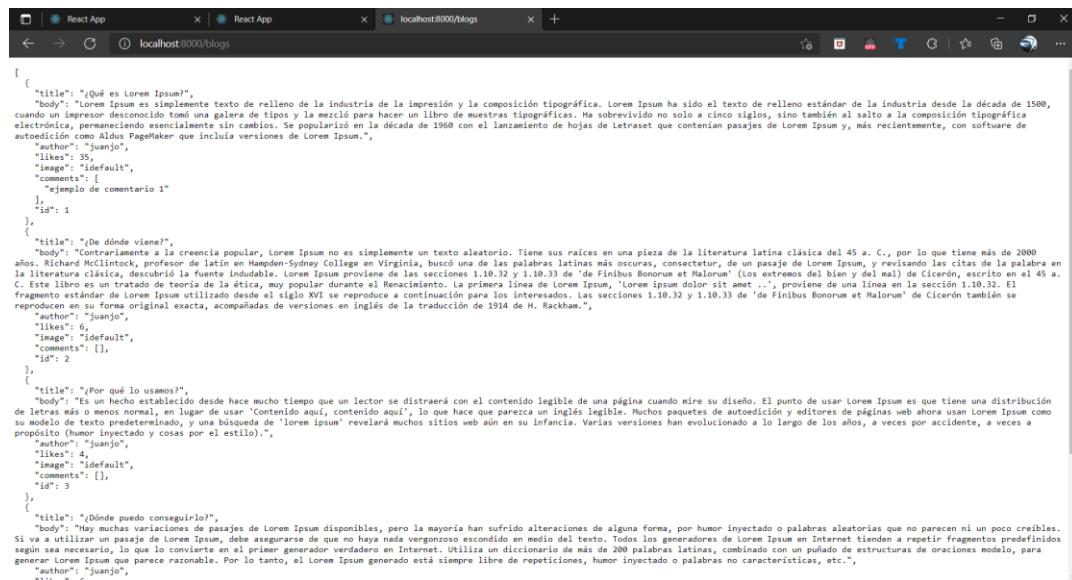


Ilustración 12: Detalles de una entrada de la aplicación (e.p.)

Usando la configuración mencionada en el apartado 4, podemos ver las entradas en la base de datos, con los campos título, cuerpo, autor, me gusta, imagen, comentarios e identificador, como muestra la Ilustración 13.



```
[{"id": 1, "title": "¿Qué es Lorem Ipsum?", "body": "Lorem Ipsum es simplemente texto de relleno de la industria de la impresión y la composición tipográfica. Lorem Ipsum ha sido el texto de relleno estándar de la industria desde la década de 1500, cuando un impresor desconocido tomó una galería de tipos y la mezcló para hacer un libro de muestras tipográficas. Ha sobrevivido no solo a cinco siglos, sino también al salto a la composición tipográfica electrónica, permaneciendo esencialmente sin cambios. Se popularizó en la década de 1960 con el lanzamiento de hojas de Letraset que contenían pasajes de Lorem Ipsum y, más recientemente, con software de autoedición como Aldus PageMaker que incluía versiones de Lorem Ipsum.", "author": "juanjo", "likes": 35, "image": "default", "comments": [{"text": "ejemplo de comentario 1"}]}, {"id": 2, "title": "De donde viene?", "body": "Contrariamente a la creencia popular, Lorem Ipsum no es simplemente un texto aleatorio. Tiene sus raíces en una pieza de la literatura latina clásica del 45 a. C., por lo que tiene más de 2000 años. Richard McClintock, profesor de latín en Hampden-Sydney College en Virginia, buscó una galería de tipos y la mezcló para hacer un libro de muestras tipográficas. Ha sobrevivido no solo a cinco siglos, sino también al salto a la composición tipográfica electrónica, permaneciendo esencialmente sin cambios. Se popularizó en la década de 1960 con el lanzamiento de hojas de Letraset que contenían pasajes de Lorem Ipsum y, más recientemente, con software de autoedición como Aldus PageMaker que incluía versiones de Lorem Ipsum.", "author": "juanjo", "likes": 12, "image": "default", "comments": []}, {"id": 3, "title": "Por que lo usamos?", "body": "Es un hecho establecido desde hace mucho tiempo que un lector se distraerá con el contenido legible de una página cuando mire su diseño. El punto de usar Lorem Ipsum es que tiene una distribución de letras más o menos normal, en lugar de usar 'Contenido aquí, contenido aquí', lo que hace que parezca un inglés legible. Muchos paquetes de autodiseño y editores de páginas web ahora usan Lorem Ipsum como modelo de texto predeterminado, y una búsqueda de 'Lorem ipsum' revelará muchos sitios web aún en su infancia. Varias versiones han evolucionado a lo largo de los años, a veces por accidente, a veces a propósito (humor inyectado y cosas por el estilo).", "author": "juanjo", "likes": 15, "image": "default", "comments": []}, {"id": 4, "title": "Dónde puedo conseguirlo?", "body": "Hoy en día, muchísimas páginas de Lorem Ipsum disponibles, pero la mayoría han sufrido alteraciones de alguna forma, por humor inyectado o palabras aleatorias que no parecen ni un poco creíbles. Si va a utilizar un pasaje de Lorem Ipsum, debe asegurarse de que no haya nada vergonzoso escondido en medio del texto. Todos los generadores de Lorem Ipsum en Internet tienden a repetir fragmentos predefinidos según sea necesario, lo que lo convierte en el primer generador verdadero en Internet. Utiliza un diccionario de más de 2000 palabras latinas, combinado con un puñado de estructuras de oraciones modelo, para generar Lorem Ipsum que parece razonable. Por lo tanto, el Lorem Ipsum generado está siempre libre de repeticiones, humor inyectado o palabras no características, etc.", "author": "juanjo", "likes": 10, "image": "default"}]
```

Ilustración 13: Detalles de la base de datos (e.p.)

En la sección Contacto, podemos ver información sobre nuestra aplicación como muestra la Ilustración 14.



Ilustración 14: Detalles de contacto de la aplicación (e.p.)

A continuación, tenemos la opción de buscar, pudiendo ver distintas cadenas sobre las que podemos buscar según escribimos (*search as you type*), viendo las totales y las filtradas, tal y como muestran las Ilustraciones 15 y 16, respectivamente.

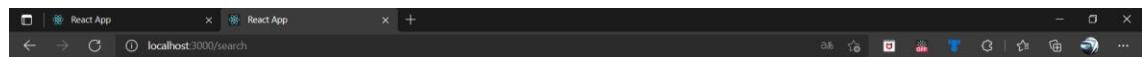
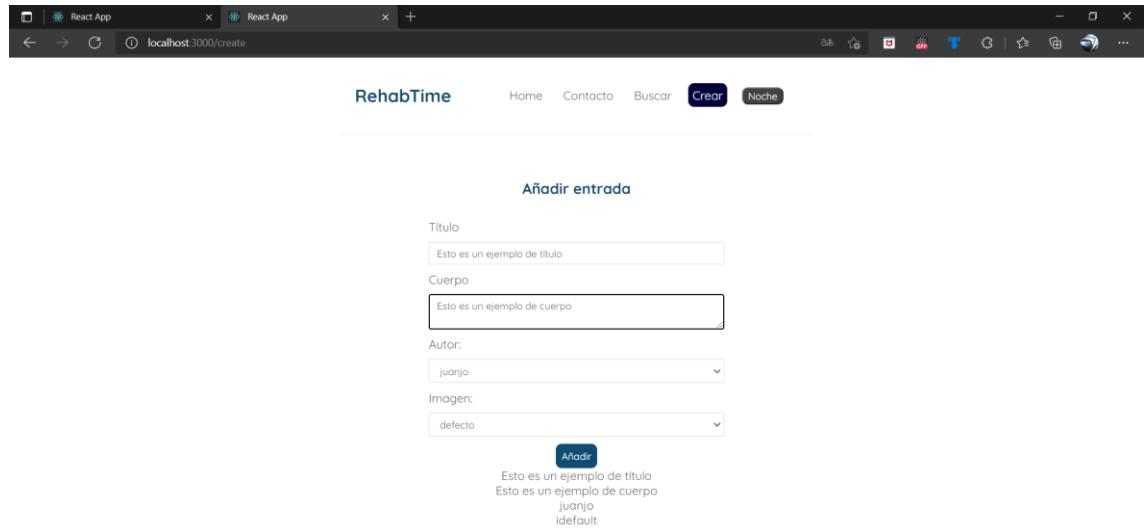


Ilustración 15: Sección Buscar en la aplicación (e.p.)



Ilustración 16: Ejemplo de búsqueda en la aplicación (e.p.)

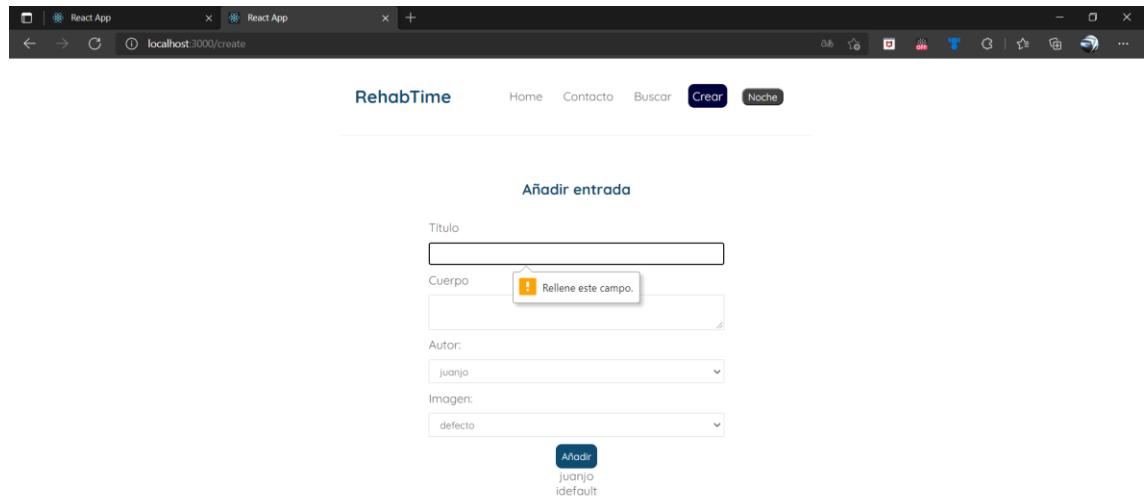
La siguiente sección en la navegación de la aplicación es la creación de una entrada, para la que necesitaremos un título, cuerpo y seleccionar un autor e imagen de las disponibles. Una vez hayamos acabado, se incluirá la nueva entrada en nuestra base de datos. Nótese que podemos ver una vista previa debajo del botón añadir.



The screenshot shows a browser window with two tabs open, both titled "React App". The active tab is "localhost:3000/create". The page title is "RehabTime". The main content is a "Añadir entrada" (Add entry) form. It has four input fields: "Título" (Title) containing "Esto es un ejemplo de título", "Cuerpo" (Body) containing "Esto es un ejemplo de cuerpo", "Autor:" (Author) containing "juanjo", and "Imagen:" (Image) containing "defecto". Below the form is a preview area showing the same data. At the bottom right is a dark blue "Añadir" (Add) button.

*Ilustración 17: Sección Añadir de la aplicación (e.p.)*

Cabe mencionar que los campos título y cuerpo son obligatorios, tal y como muestra la Ilustración 18.



The screenshot shows the same browser setup as Illustration 17. The "Cuerpo" (Body) field is empty. A red exclamation mark icon appears in the top-left corner of the input field, with the message "Rellene este campo." (This field is required). The other fields ("Título", "Autor:", "Imagen:") and the preview area below the form remain the same as in Illustration 17.

*Ilustración 18: Requisitos para la creación de una entrada (e.p.)*

Como mecanismo de seguridad, cuando accedamos a una URL no válida o incorrecta, veremos el error mostrado en la Ilustración 19.

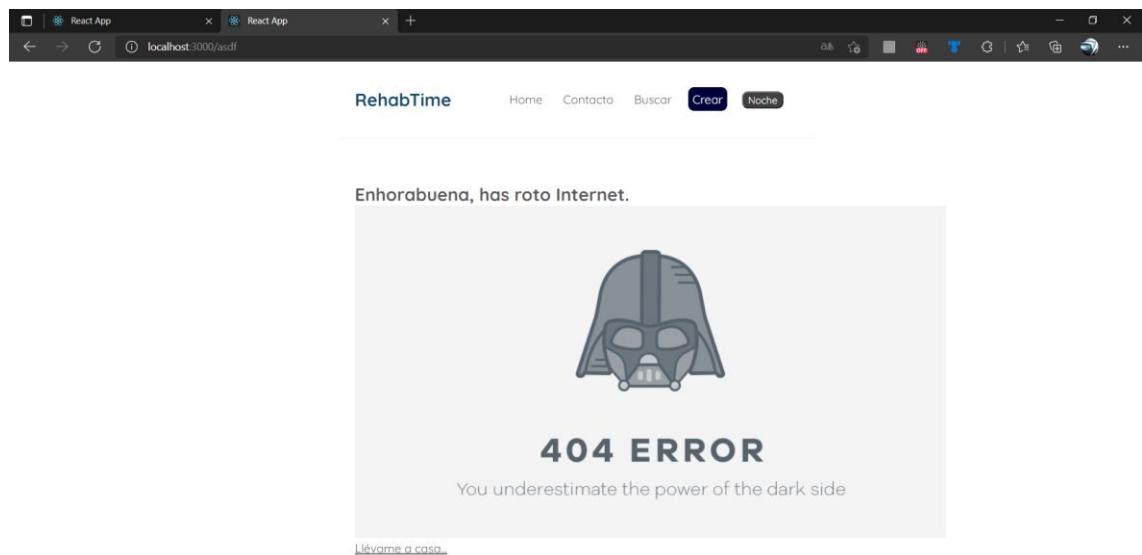


Ilustración 19: Error mostrado al introducir una URL no válida (e.p.)

Otro mecanismo utilizado en la aplicación es mostrar un mensaje de carga en la página principal de la aplicación, para mostrar al usuario que la aplicación no se ha colgado y está cargando, tal y como muestra la Ilustración 20.

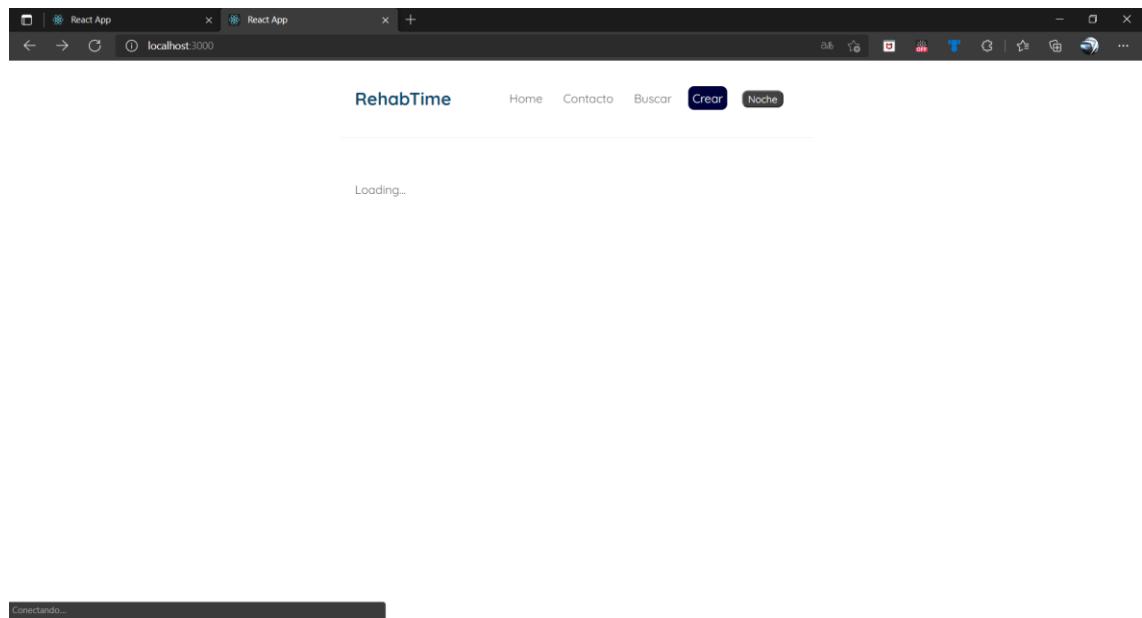


Ilustración 20: Mensaje de carga de la aplicación (e.p.)

Además, si se agota el tiempo de carga y se produjese un error, este se mostraría por pantalla como el ejemplo de la Ilustración 21.

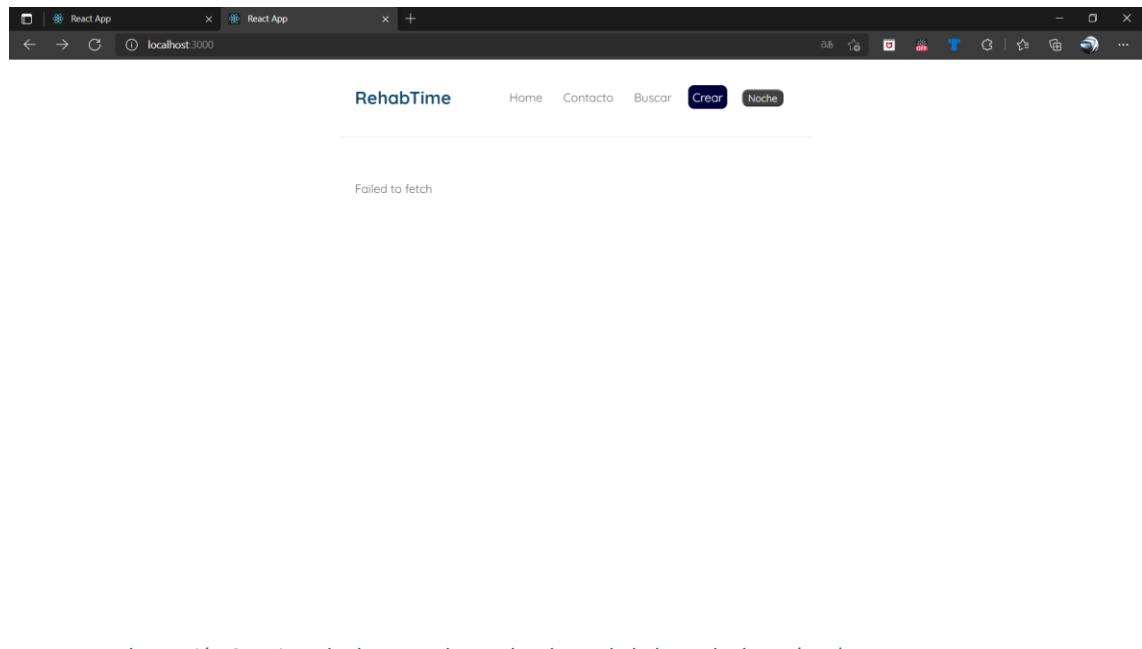


Ilustración 21: Ejemplo de error al traer los datos de la base de datos (e.p.)

Por último, se muestra en las Ilustraciones 22 y 23 un ejemplo del modo noche en la pantalla Home y Crear, respectivamente, siendo este modo usable en toda la aplicación.

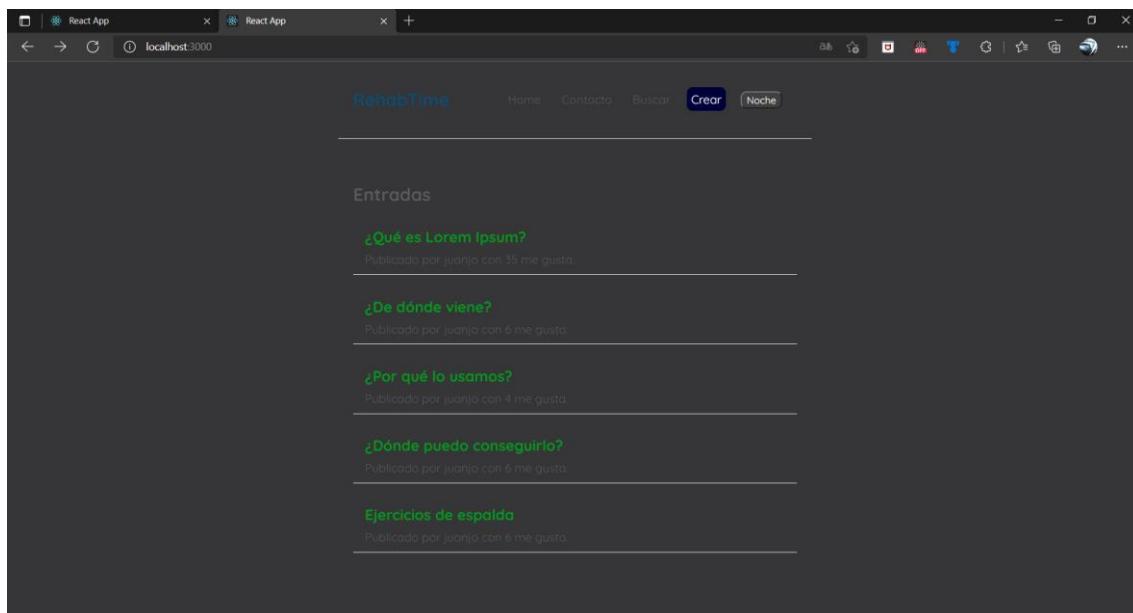


Ilustración 22: Ejemplo modo noche - Home (e.p.)

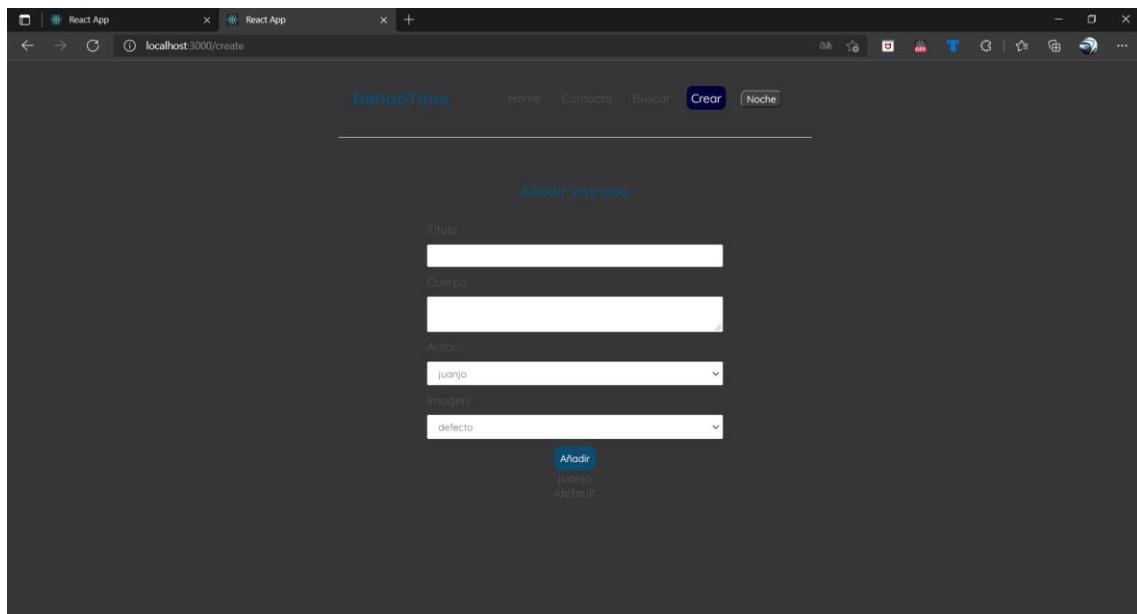


Ilustración 23: Ejemplo modo noche - Crear (e.p.)

## 6. Lighthouse

Lighthouse es una herramienta automatizada de código abierto para medir la calidad de las páginas web, pudiendo ejecutarse desde cualquier página web.

Para acceder a este informe, accedemos a nuestra web desde el navegador, vamos a opciones de desarrollador, y clicamos en Lighthouse. Aquí, evaluaremos el rendimiento, la accesibilidad, las prácticas y la optimización de motores de búsqueda de páginas web.

En la Ilustración 24 podemos ver el resultado obtenido usando la navegación privada del navegador Edge para la aplicación, obteniendo un 74 en rendimiento y un 100 en el resto de categorías evaluadas.

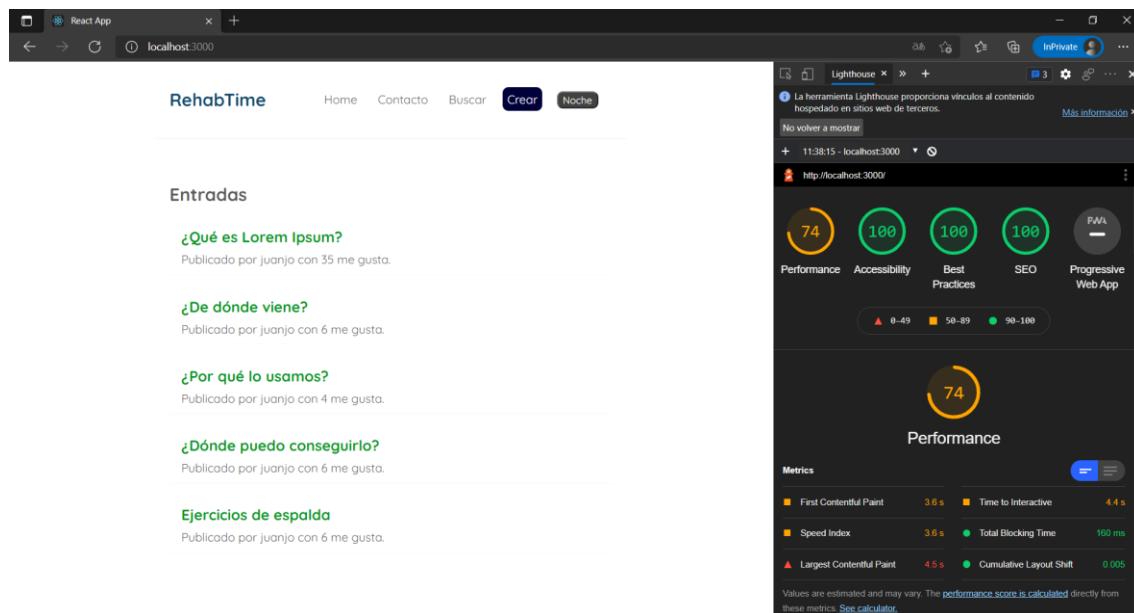


Ilustración 24: Evaluación Lighthouse de la aplicación (e.p.)

## 7. Docker

Docker es un conjunto de productos PaaS que utiliza la virtualización a nivel de sistema operativo para entregar software en paquetes llamados contenedores. Estos contenedores están aislados entre sí, agrupando su propio software, librerías y configuración; aunque pueden comunicarse entre sí a través de canales definidos. Tiene niveles gratuitos y de pago.

El principal motivo por el que elijo Docker es por su eficiencia y aceptación, además del encapsulamiento y la cantidad de imágenes disponibles.

### 7.1. Pasos previos

Comenzamos con la descarga de la versión de escritorio con la configuración mostrada en la Ilustración 25. Esperamos hasta que termine la instalación.

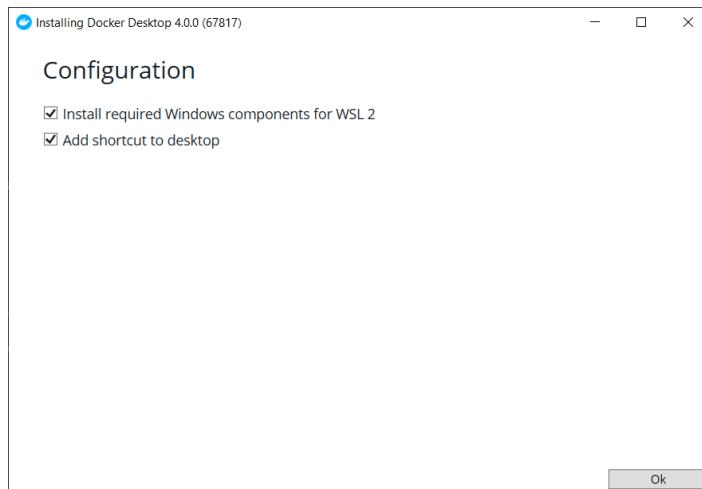


Ilustración 25: Configuración Docker (e.p.)

El siguiente paso es instalar la extensión de Docker en VS Code:

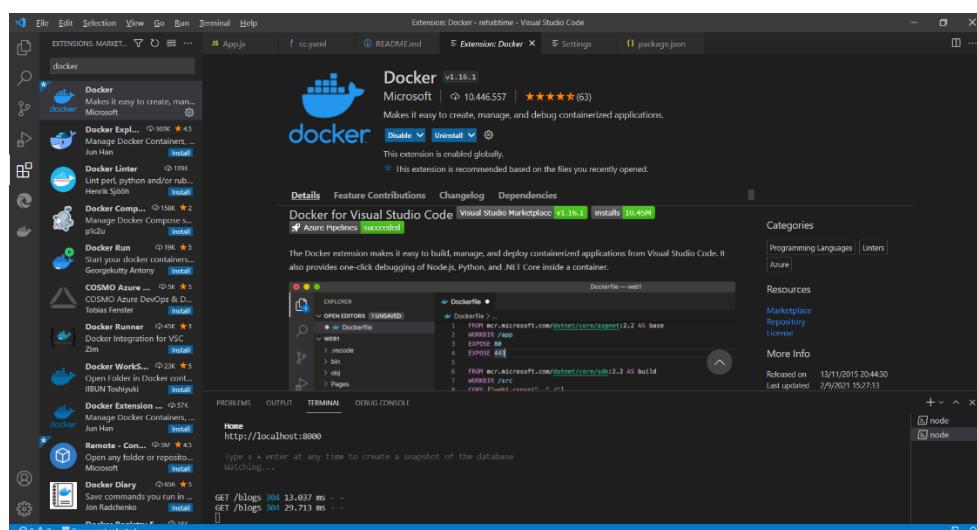


Ilustración 26: Extensión Docker en VS (e.p.)

## 7.2. Creación ficheros e imagen

A continuación, creamos los ficheros necesarios para Docker, para ello, clicamos Ctrl+Shift+P, abriéndose el menú mostrado en la Ilustración 27.

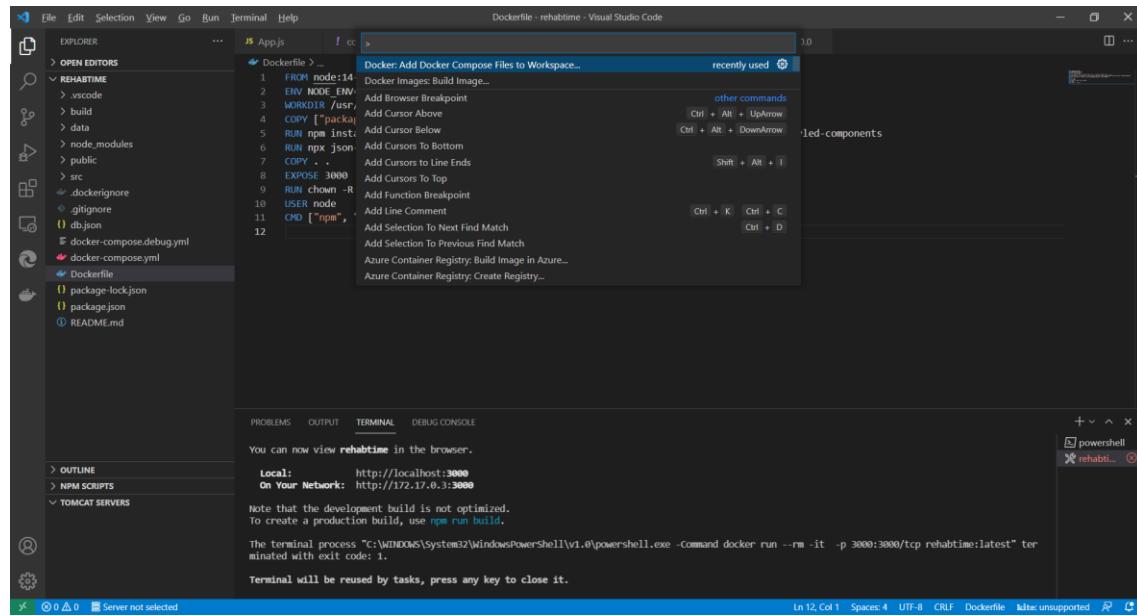


Ilustración 27: Añadiendo Ficheros Docker (e.p.)

Seleccionamos *Add Docker Compose files to Workspace* con la siguiente configuración:

- La plataforma será Node.js
- Elegimos nuestro paquete *package.json*.
- Seleccionamos el puerto 3000.

Ahora tendremos creados los ficheros *Dockerfile*, *docker-compose.yml* y *docker-compose-debug.yml*. A continuación, editamos el fichero *Dockerfile* quedando como muestra la Ilustración 28.

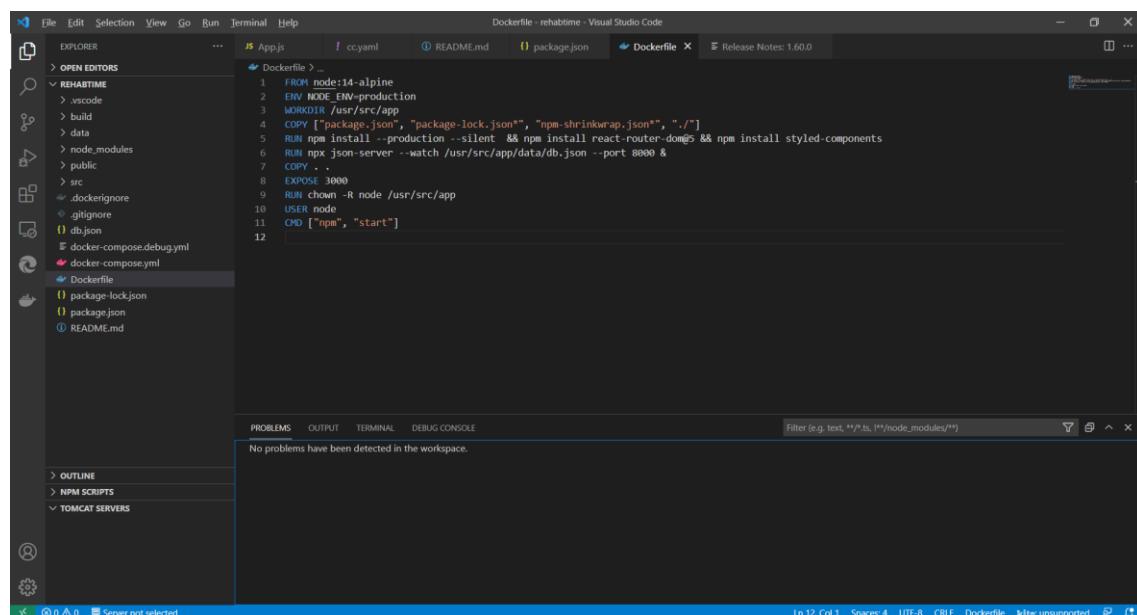


Ilustración 28: Edición del fichero Dockerfile (e.p.)

Ahora, clicamos de nuevo Ctrl+Shift+P y seleccionamos Docker Images: Build Image. Seleccionamos nuestro fichero *Dockerfile* de la Ilustración 28, y esperamos a que termine, accediendo a las imágenes de Docker creadas según la Ilustración 30.

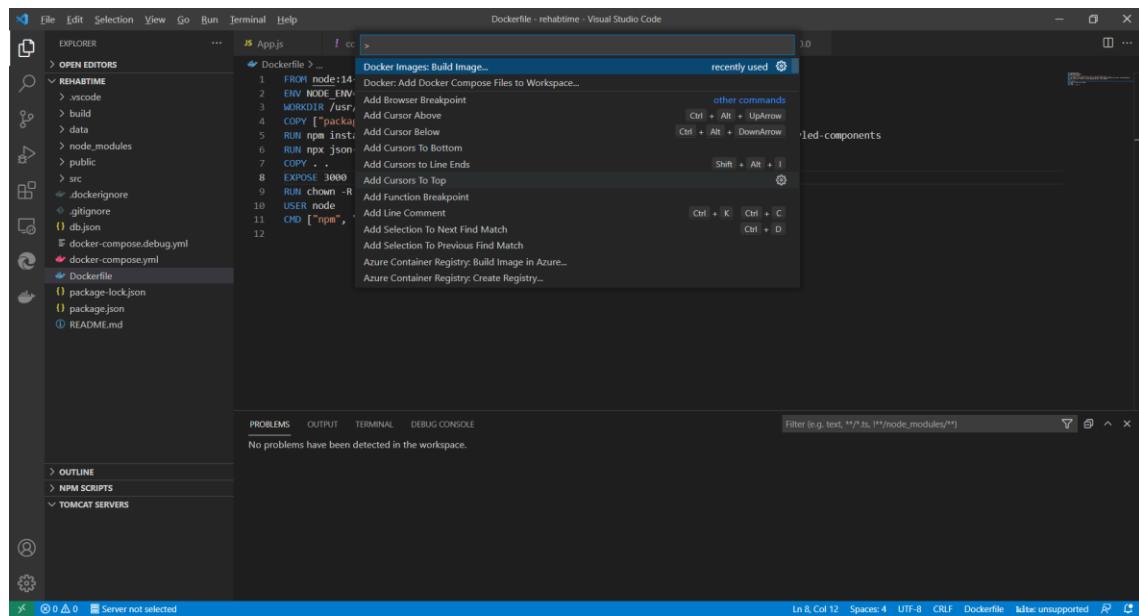


Ilustración 29: Creación Imagen Docker (e.p.)

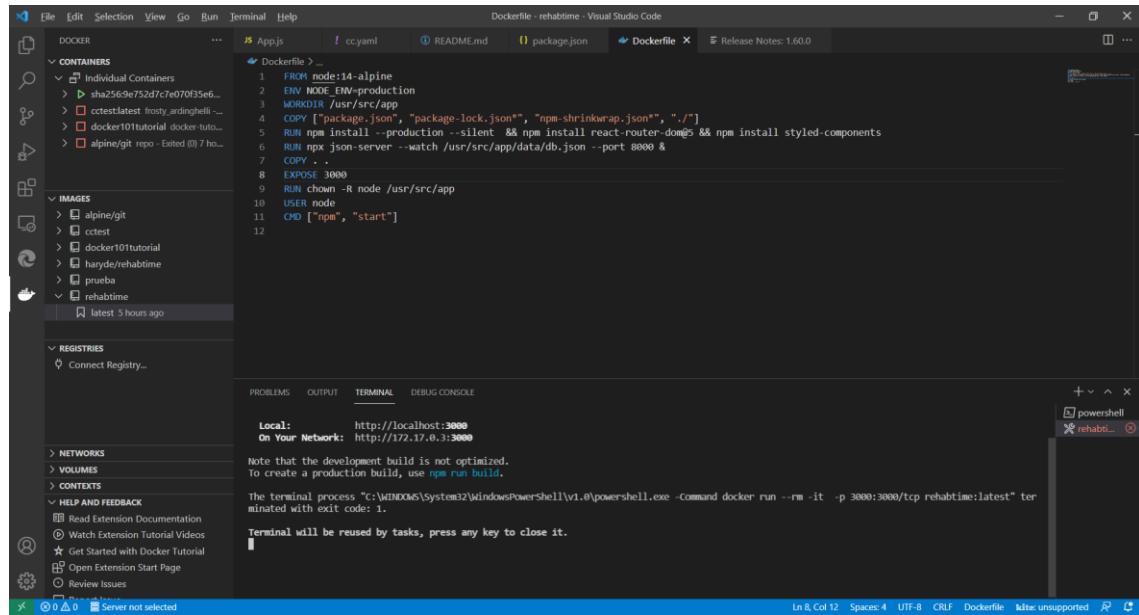


Ilustración 30: Imagen de Docker rehabtime creada (e.p.)

### 7.3. Ejecución de imagen

El siguiente paso es ir a la aplicación de Docker y ver creada la imagen `rehabtime:latest`, ejecutando el contenedor para comprobar que funciona correctamente según la Ilustración 31. Además, comprobamos la aplicación en el navegador como muestra la Ilustración 32.

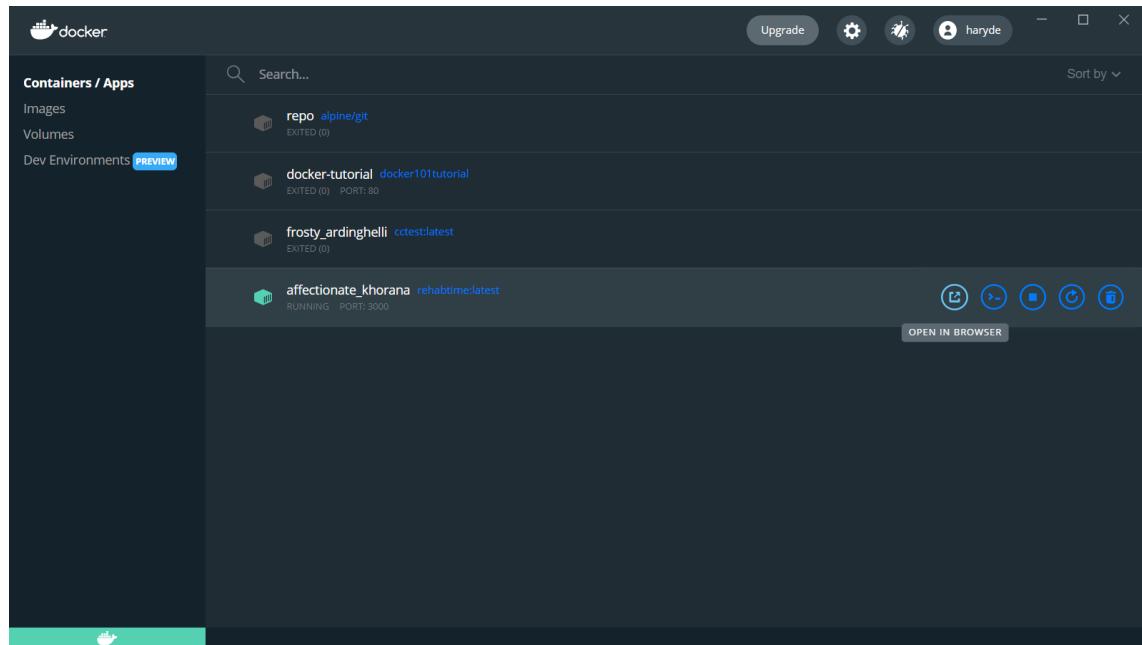


Ilustración 31: Contenedor ejecutando la imagen Docker `rehabtime:latest` (e.p.)

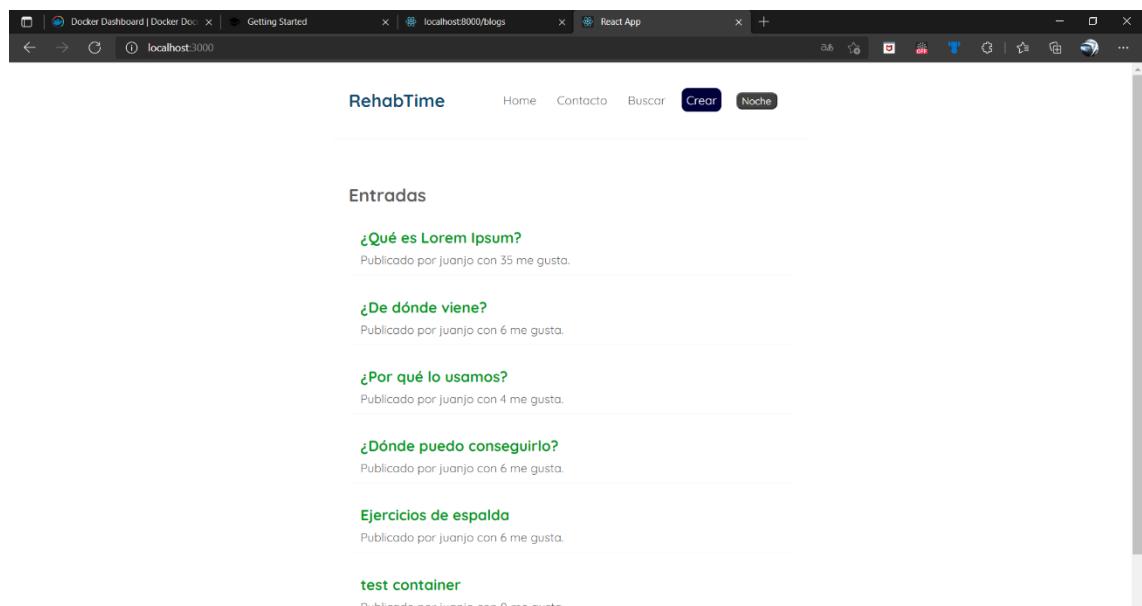
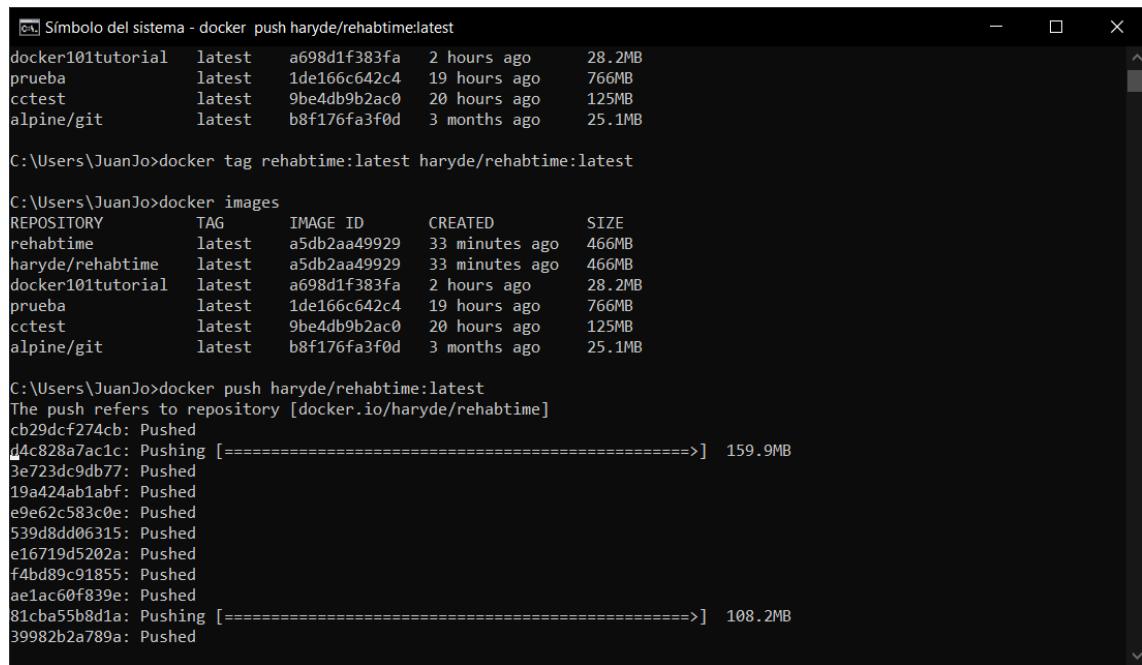


Ilustración 32: Ejecución aplicación en contenedor Docker (e.p.)

## 7.4. Docker Hub

El siguiente objetivo es publicar la imagen en Docker Hub, para lo cual, nos registramos y creamos una cuenta. A continuación, creamos un repositorio con la interfaz gráfica y, una vez creado, nos vamos a la terminal y etiquetamos la imagen para luego hacer *push* de la imagen Docker según la Ilustración 33, esperando a que esté terminado como muestra la Ilustración 34, comprobando finalmente en Docker Hub que la imagen Docker se subió correctamente como se ve en las Ilustraciones 35 y 36.



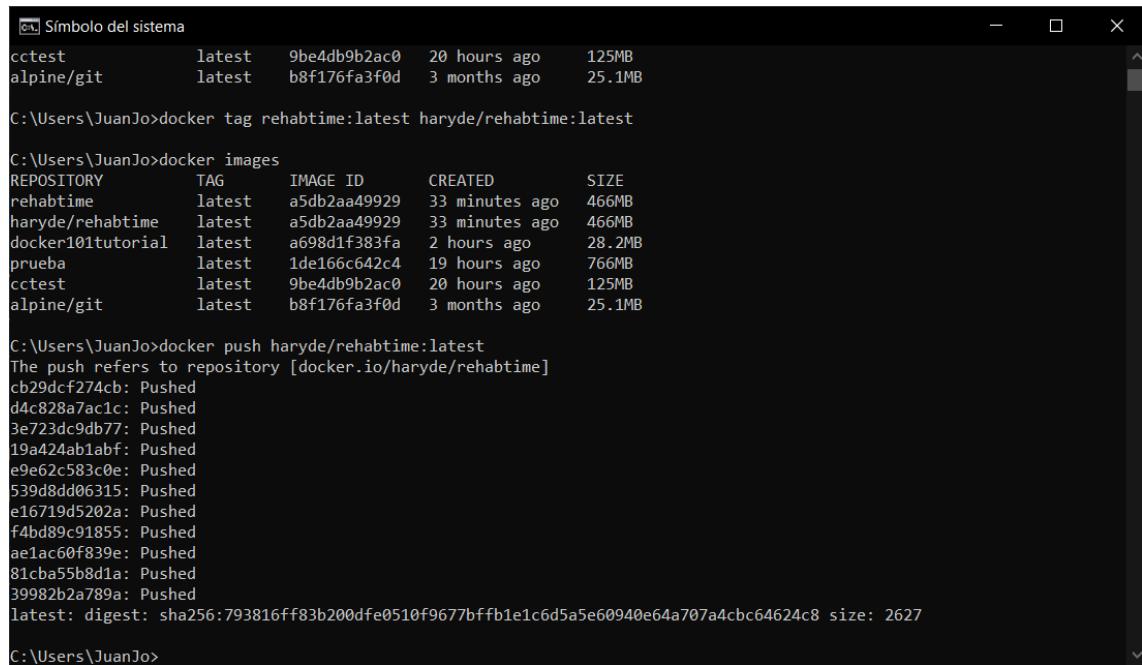
```
Símbolo del sistema - docker push haryde/rehabtime:latest
docker101tutorial    latest    a698d1f383fa  2 hours ago   28.2MB
prueba              latest    1de166c642c4  19 hours ago  766MB
cctest              latest    9be4db9b2ac0  20 hours ago  125MB
alpine/git          latest    b8f176fa3f0d  3 months ago  25.1MB

C:\Users\JuanJo>docker tag rehabtime:latest haryde/rehabtime:latest

C:\Users\JuanJo>docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
rehabtime       latest    a5db2aa49929  33 minutes ago  466MB
haryde/rehabtime latest    a5db2aa49929  33 minutes ago  466MB
docker101tutorial latest    a698d1f383fa  2 hours ago   28.2MB
prueba          latest    1de166c642c4  19 hours ago  766MB
cctest          latest    9be4db9b2ac0  20 hours ago  125MB
alpine/git      latest    b8f176fa3f0d  3 months ago  25.1MB

C:\Users\JuanJo>docker push haryde/rehabtime:latest
The push refers to repository [docker.io/haryde/rehabtime]
cb29dcf274cb: Pushed
d4c828a7ac1c: Pushing [=====>] 159.9MB
3e723dc9db77: Pushed
19a424ab1abf: Pushed
e9e62c583c0e: Pushed
539d8dd06315: Pushed
e16719d5202a: Pushed
f4bd89c91855: Pushed
ae1ac60f839e: Pushed
81cba55b8d1a: Pushing [=====>] 108.2MB
39982b2a789a: Pushed
```

Ilustración 33: Etiqueta y push de la imagen Docker (e.p.)



```
Símbolo del sistema
cctest       latest    9be4db9b2ac0  20 hours ago  125MB
alpine/git   latest    b8f176fa3f0d  3 months ago  25.1MB

C:\Users\JuanJo>docker tag rehabtime:latest haryde/rehabtime:latest

C:\Users\JuanJo>docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
rehabtime       latest    a5db2aa49929  33 minutes ago  466MB
haryde/rehabtime latest    a5db2aa49929  33 minutes ago  466MB
docker101tutorial latest    a698d1f383fa  2 hours ago   28.2MB
prueba          latest    1de166c642c4  19 hours ago  766MB
cctest          latest    9be4db9b2ac0  20 hours ago  125MB
alpine/git      latest    b8f176fa3f0d  3 months ago  25.1MB

C:\Users\JuanJo>docker push haryde/rehabtime:latest
The push refers to repository [docker.io/haryde/rehabtime]
cb29dcf274cb: Pushed
d4c828a7ac1c: Pushed
3e723dc9db77: Pushed
19a424ab1abf: Pushed
e9e62c583c0e: Pushed
539d8dd06315: Pushed
e16719d5202a: Pushed
f4bd89c91855: Pushed
ae1ac60f839e: Pushed
81cba55b8d1a: Pushed
39982b2a789a: Pushed
latest: digest: sha256:793816ff83b2000ffe0510f9677bffb1e1c6d5a5e60940e64a707a4cbc64624c8 size: 2627

C:\Users\JuanJo>
```

Ilustración 34: Push terminado de la imagen Docker (e.p.)

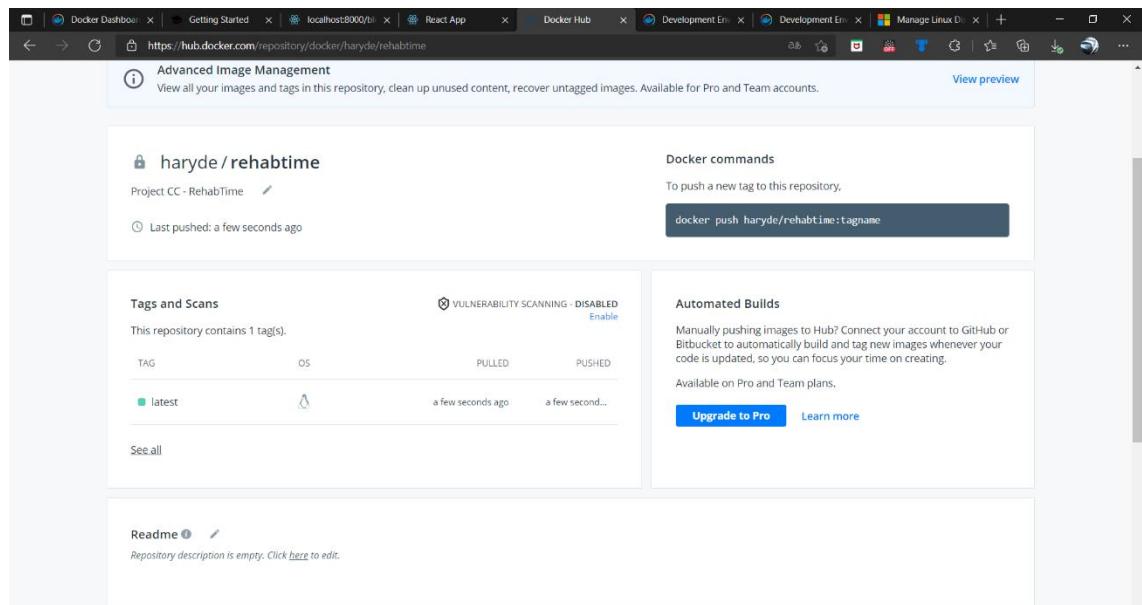


Ilustración 35: Imagen en Docker Hub (e.p.)

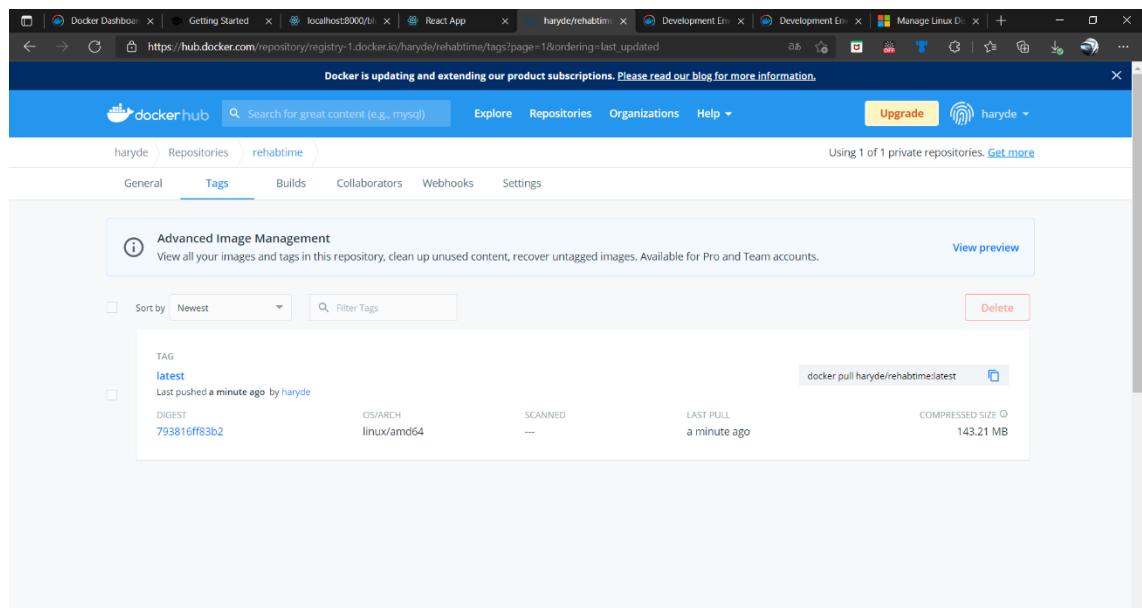


Ilustración 36: Detalles imagen en Docker Hub (e.p.)

## 8. CI/CD

En la ingeniería del software, la integración continua y entrega continua (por sus siglas en inglés, CICD) se refiere a las prácticas combinadas de ambas, es decir, el despliegue continuo. La integración continua es una práctica que consiste en hacer integraciones automáticas de un proyecto con la mayor regularidad posible con el objetivo de detectar errores lo antes posible. La entrega continua es una filosofía en la cual los equipos de desarrollo producen software en ciclos cortos, asegurando que el software pueda ser liberado en cualquier momento con seguridad y confianza.

En nuestro caso seleccionamos Netlify y Travis, que veremos a continuación.

### 8.1. Netlify

Netlify es una empresa de computación en la nube que ofrece servicios de alojamiento y *backend* sin servidor. La empresa proporciona alojamiento para sitios web cuyos archivos de origen se almacenan en Git y luego se generan en archivos de contenido web estático, servidos a través de una red de entrega de contenido. La empresa posteriormente amplió los servicios para incluir sistemas de gestión de contenido y funciones de computación sin servidor para manejar sitios web con funciones interactivas.

Lo primero que hacemos es incluir Netlify en VS y la instalamos por la terminal con el comando `npm install netlify-cli -g`.

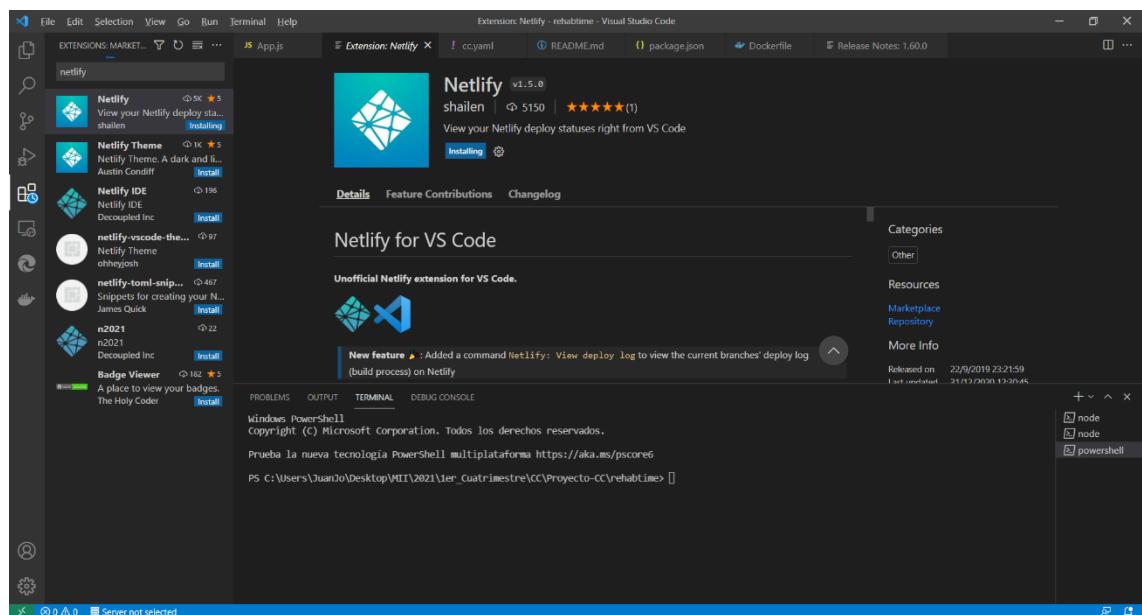


Ilustración 37: Instalación Netlify (e.p.)

Una vez instalado, construimos el proyecto:

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left lists files and folders including .vscode, build, data, node\_modules, public, src, images, test, Announcer.js, App.js, BlogDetails.js, BlogList.js, Contact.js, Create.js, GlobalStyles.js, Home.js, Images.js, index.css, index.js, logo.svg, NavBar.js, NotFound.js, and reportWebVitals.js. The Problems, Output, Terminal, and Debug Console tabs are visible at the bottom. The Terminal tab shows deployment instructions:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
688 B buildstatic\css\main.8545e9cf.chunk.css
The project was built assuming it is hosted at /.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.
You may serve it with a static server:
npm install -g serve
serve -s build

Find out more about deployment here:
https://cra.link/deployment
PS C:\Users\JuanJo\Desktop\MII\2021\1er Cuatrimestre\CC\Proyecto-CC\rehabtime>
```

The status bar at the bottom indicates: In 8, Col 7 Spaces: 4 UTF-8 CR/LF JavaScript file ready.

Ilustración 38: Construcción del proyecto (e.p.)

Ahora, vinculamos VS con Github.

The screenshot shows the Visual Studio Code interface with GitHub integration. The Explorer sidebar shows a message: "The folder currently open doesn't have a git repository. You can initialize a repository which will enable source control features powered by git." Below this are buttons for "Initialize Repository" and "Publish to GitHub". The code editor shows the same博Details.js file as in Illustration 38. The Terminal tab shows a PowerShell session:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnologia PowerShell multiplataforma https://aka.ms/pscore6
PS C:\Users\JuanJo\Desktop\MII\2021\1er Cuatrimestre\CC\Proyecto-CC\rehabtime>
```

The status bar at the bottom indicates: In 28, Col 37 Spaces: 4 UTF-8 CR/LF JavaScript file initializing.

Ilustración 39: Vinculación VS-Github (e.p.)

Los pasos son:

- Clicar en Continuar.
- Autorizar Github a VS a todos los repositorios y el flujo de trabajo.
- Introducimos nuestra contraseña.
- Aceptamos Git, esperando el mensaje de confirmación.

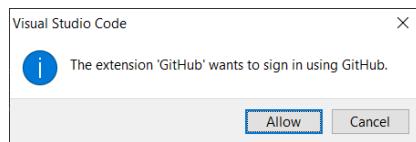


Ilustración 40: Pop-up confirmación uso Github (e.p.)



Authorize Visual Studio Code to access GitHub  
If you initiated this authorization from Visual Studio Code, click Continue to authorize access to GitHub.

[Continue](#)

[Do not authorize](#)

Ilustración 41: Autorizar VS a acceder a GitHub (e.p.)

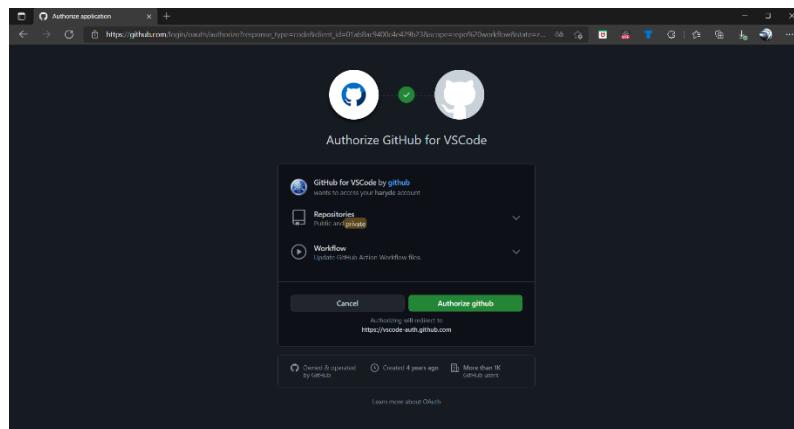


Ilustración 42: Detalles de la autorización VS-GitHub (e.p.)

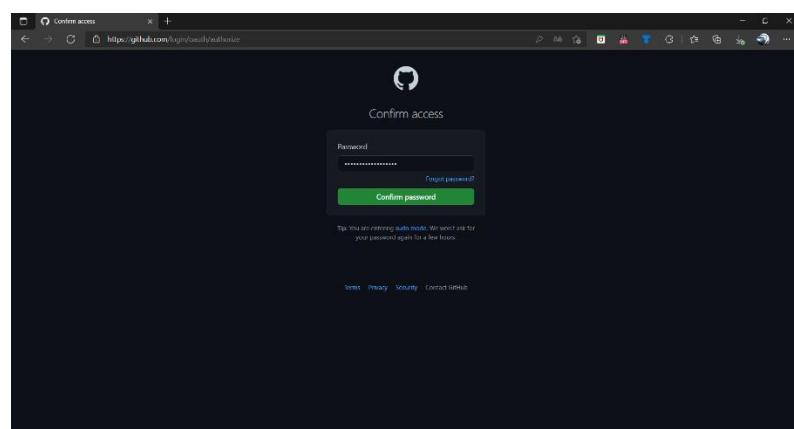


Ilustración 43: Contraseña GitHub (e.p.)

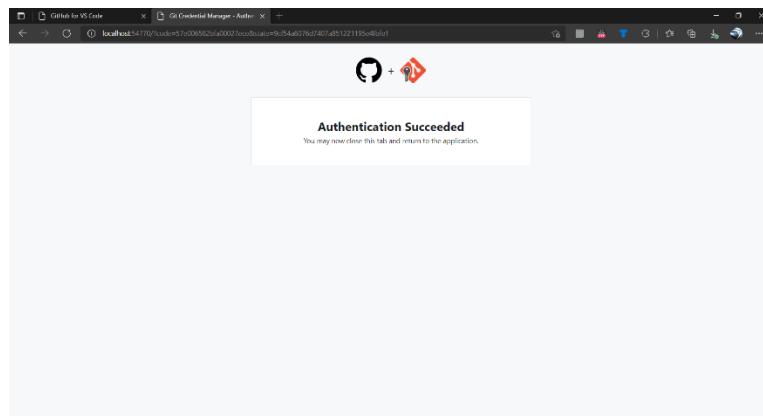


Ilustración 44: Mensaje de confirmación de autenticación (e.p.)

Una vez subido, comprobamos que se encuentra todo en GitHub. Ahora, podemos ir a la web de Netlify y crear nuestro nuevo sitio, registrándonos con GitHub.

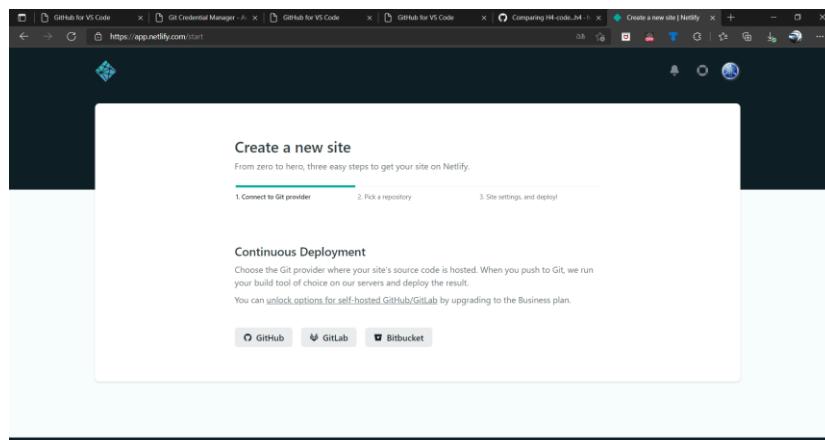


Ilustración 45: Creación de nuevo sitio Netlify (e.p.)

Seleccionamos nuestro repositorio:

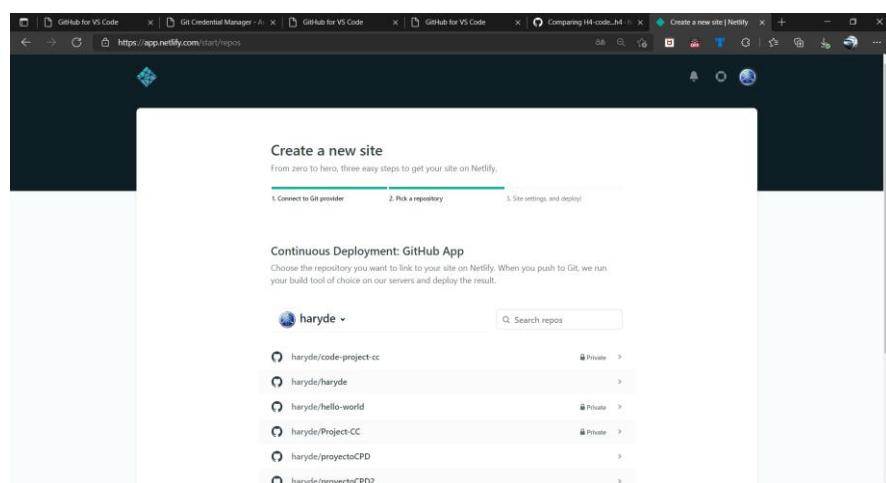
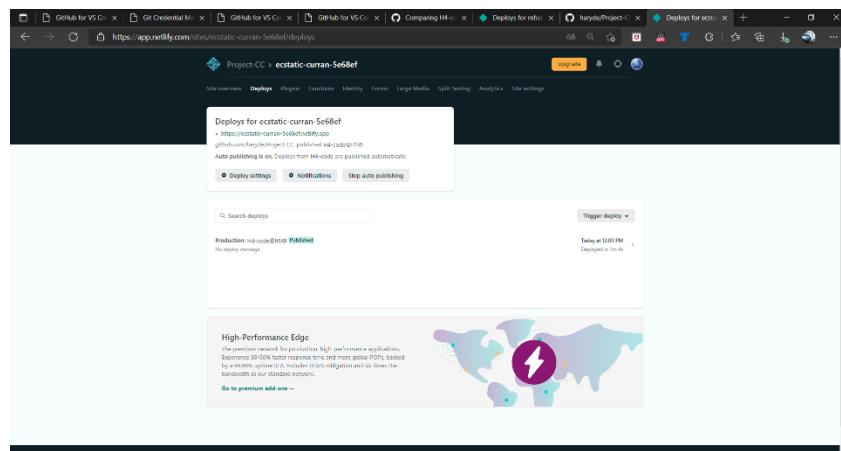


Ilustración 46: Selección de repositorio Netlify (e.p.)

Esperamos el despliegue y ejecutamos la aplicación:



*Ilustración 47: Despliegue Netlify (e.p.)*



*Ilustración 48: Ejecución de la aplicación en Netlify (e.p.)*

Por último, cambiamos el sitio del nombre a uno más legible:

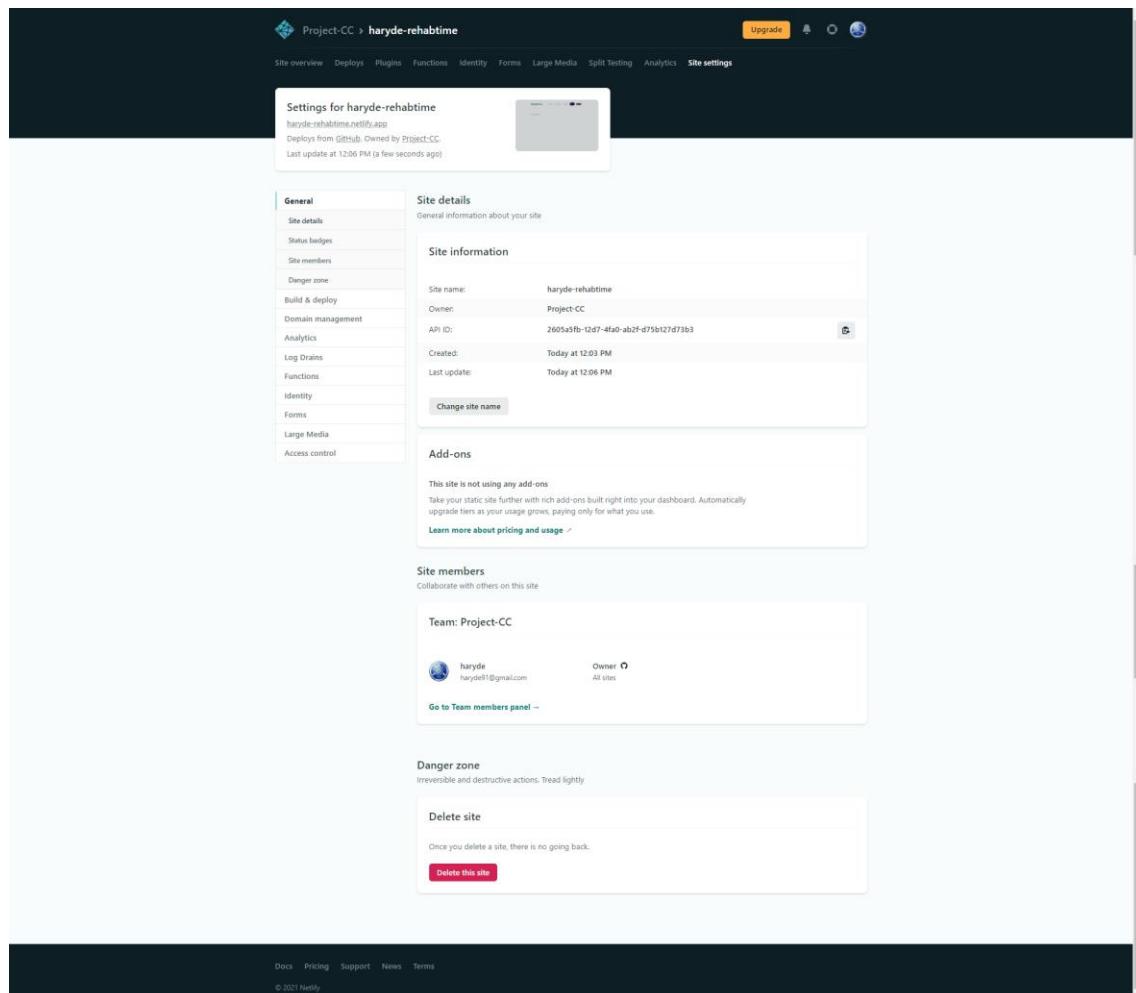


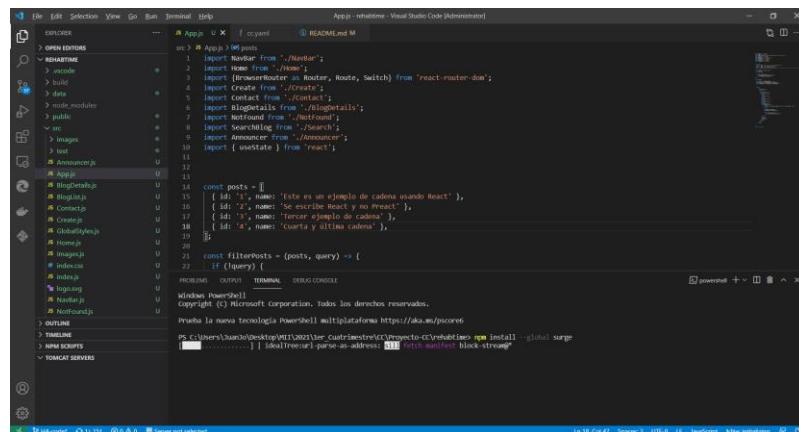
Ilustración 49: Cambio configuración Netlify (e.p.)

## 8.2. Travis

Travis CI es un servicio alojado de integración continua, que se usa para crear y comprobar proyectos de software alojados en GitHub y Bitbucket.

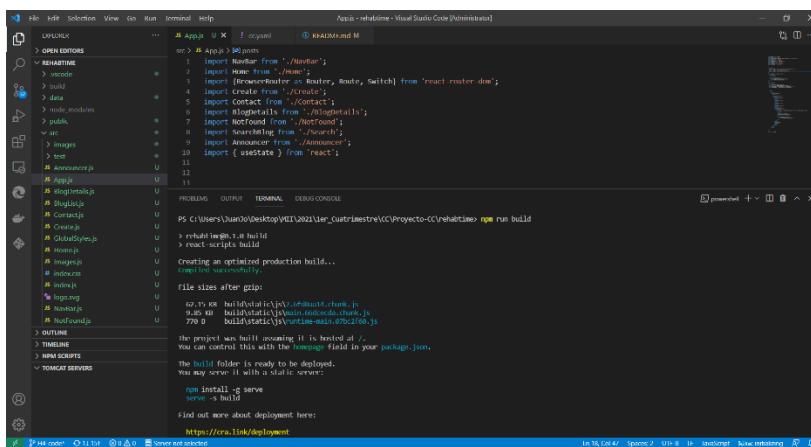
Travis CI fue el primer servicio de CI que brindó servicios a proyectos de código abierto de forma gratuita, siendo así en la actualidad. TravisPRO proporciona implementaciones personalizadas de una versión propietaria del cliente.

Para usar Travis, comenzamos instalando Surge, que es un publicador estático de web desde el terminal, según muestra la Ilustración 50.



*Ilustración 50: Instalación surge.sh (e.p.)*

Ahora, hacemos un *build* de la aplicación, y desde la terminal, creamos y configuraremos nuestro sitio web con surge como muestra la Ilustración 52.



*Ilustración 51: Build de la aplicación (e.p.)*

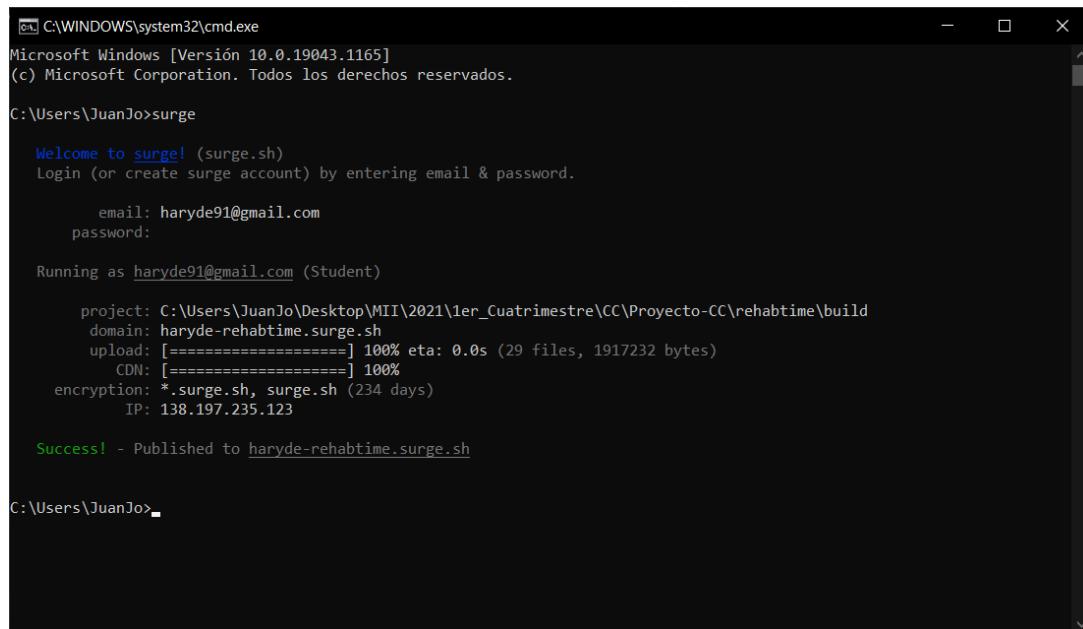


Ilustración 52: Configuración surge de la aplicación (e.p.)

A continuación, comprobamos que funciona la aplicación:

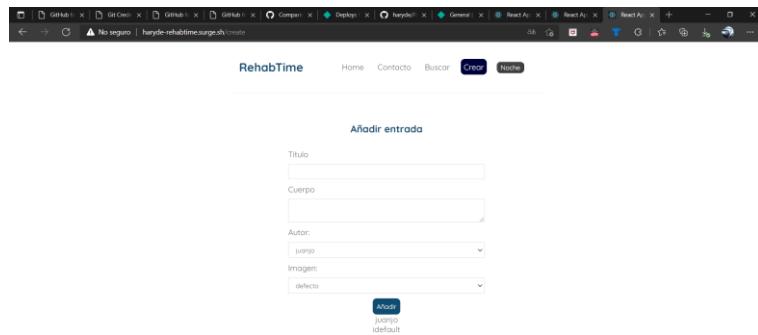


Ilustración 53: Aplicación ejecutada en Surge (e.p.)

El siguiente paso es autorizar acceso a GitHub por parte de Travis.

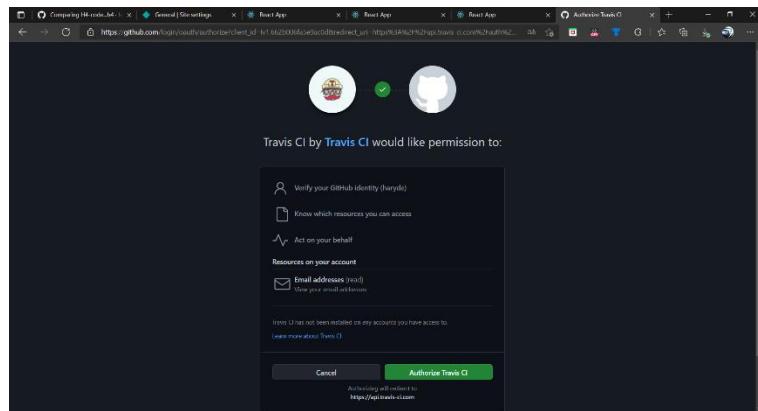


Ilustración 54: Acceso a GitHub por Travis (e.p.)

El siguiente paso es crearnos dos variables de entorno, siendo el login y el token de Surge. Para crear este token, sólo necesitamos poner *surge token* en la terminal.

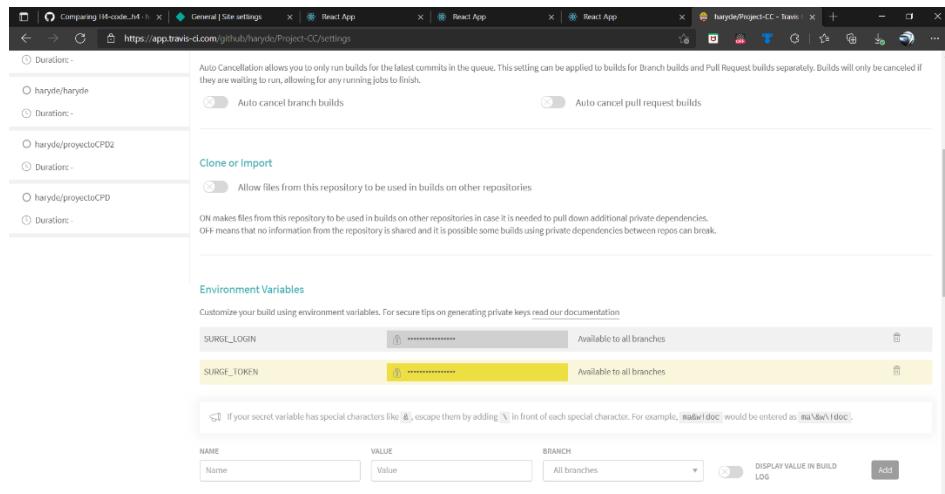


Ilustración 55: Variables de entorno Travis (e.p.)

El siguiente paso es seleccionar nuestro repositorio donde está alojado el código, le damos a construir (*build*) y esperamos a que termine. Además, recibiremos un correo con la terminación del mismo, como muestra la Ilustración 57.

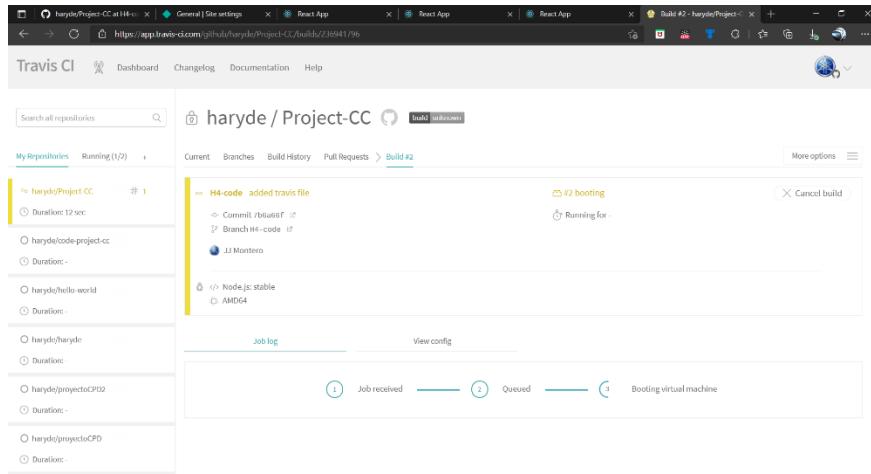


Ilustración 56: Construcción Travis (e.p.)

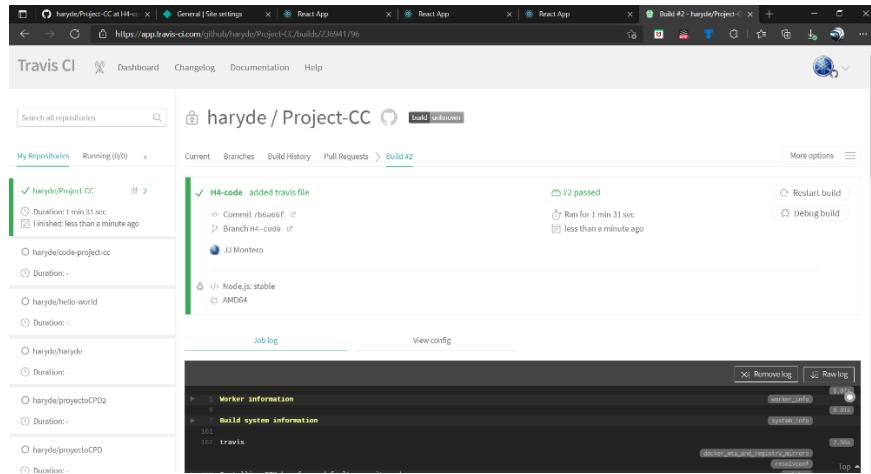


Ilustración 57: Construcción Travis completada (e.p.)

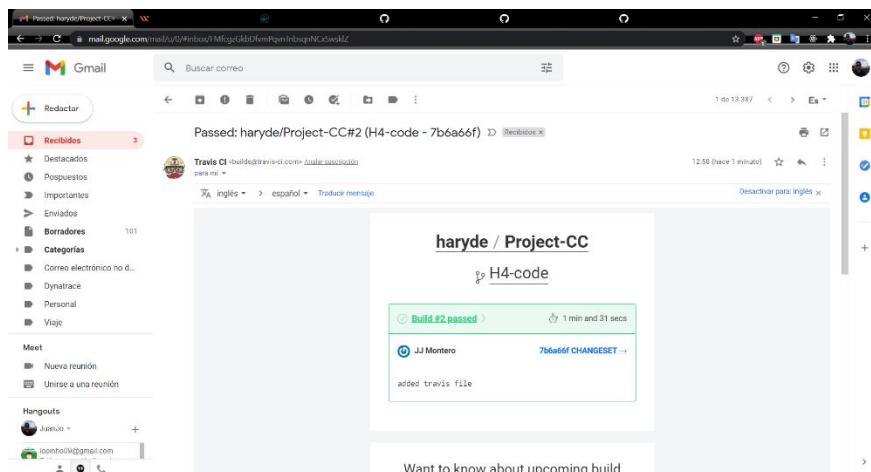


Ilustración 58: Notificación construcción Travis completada (e.p.)

### 8.3. Lighthouse Netlify

Para comprobar las prestaciones de la aplicación alojada en Netlify, procedemos a usar Lighthouse. Se obtiene una puntuación de 96 en rendimiento y 100 en disponibilidad, buenas prácticas y SEO.

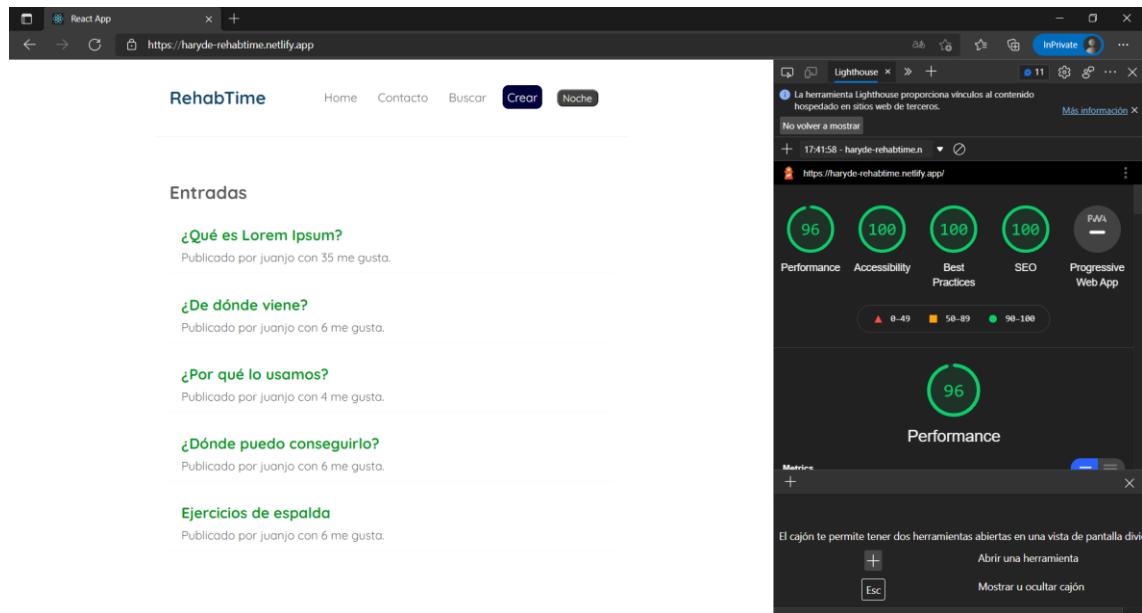


Ilustración 59: Lighthouse Netlify (e.p.)

### 8.4. Lighthouse Surge

Para comprobar las prestaciones de la aplicación alojada en Netlify, procedemos a usar Lighthouse. Se obtiene una puntuación de 96 en rendimiento y 100 en disponibilidad, 87 en buenas prácticas y 92 en SEO.

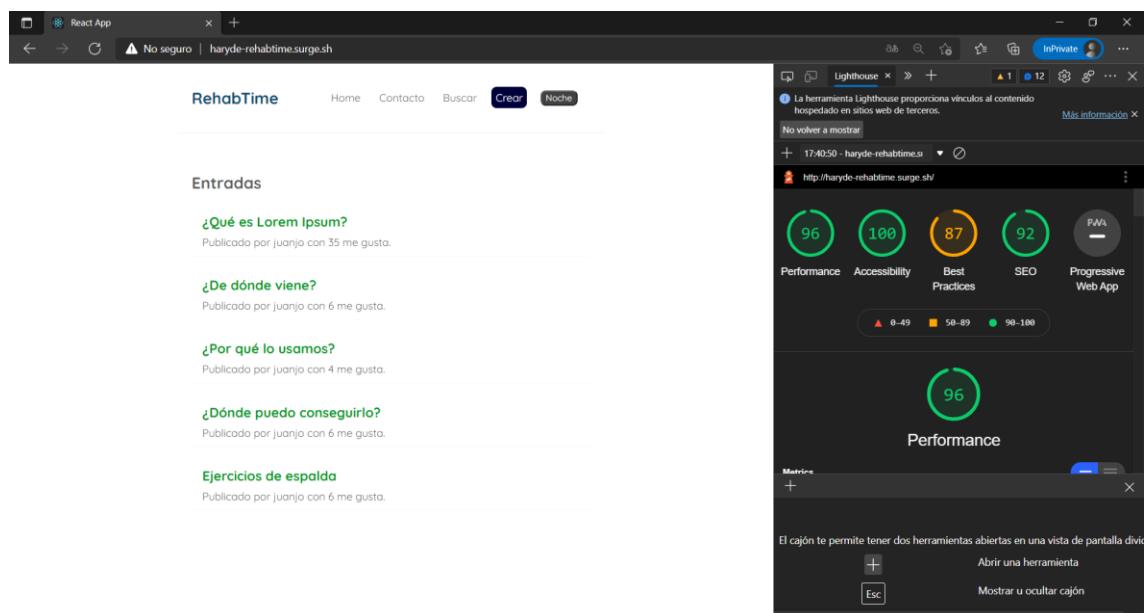


Ilustración 60: Lighthouse Surge (e.p.)

## 9. API

La interfaz de programación de aplicaciones (por sus siglas en inglés, API) es un conjunto de protocolos y definiciones que se usan para desarrollar e integrar el software de las aplicaciones. Las API permiten que sus servicios y productos se comuniquen con otros, sin necesidad de saber cómo están implementados.

Las llamadas a la API que usa la aplicación son:

- POST: cuando se crea una entrada en la aplicación (Create.js).
  - PUT: cuando se añade un comentario en una entrada, o se le da a me gusta (BlogDetails.js).
  - DELETE: cuando se elimina una entrada en la aplicación (BlogDetails.js).
  - GET: cuando se consultan las entradas de la aplicación.

Para comprobar que los datos se recogen correctamente, se hace uso del fichero `useFetch.js`, como se explicó en el punto 3.1 de este documento. En caso de no poder comunicarse con la base de datos, se lanzará el error `Failed to fetch`.

## 9.1. Postman

Para comprobar las llamadas a la API de forma externa, usaremos Postman, que es un programa para realizar llamadas a la API.

Comenzamos con la llamada GET. Vemos que se devuelve todas las entradas de la base de datos. Si quisieramos los detalles de una entrada, bastaría con añadir el identificador al final de la llamada a la API ([.../blogs/id](#)).

Postman

File Edit View Help

New Import Runner

My Workspace Invite

Filter

History Collections API Beta

Show Responses Clear all

Untitled Request

GET https://localhost:8000/blogs

Params Authorization Headers Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
8	Value	Description
Key	Value	Description

Body Cookies Headers (14) Test Results

Pretty Raw Preserve Visualization Beta JSON

1 

```
[{"id": 1, "title": "¿Qué es Lorem Ipsum?", "body": "Lorem Ipsum es simplemente texto de relleno de la industria de la impresión y la composición tipográfica. Lorem Ipsum ha sido el texto de relleno estándar de la industria desde la década de 1500, cuando un impresor desencadenó una galera de tipos y la mezcló para hacer un libro de muestras tipográficas. Ha sobrevivido no solo a cinco siglos, sino también al salto a la composición tipográfica electrónica, permaneciendo esencialmente intacto. Se popularizó en la década de 1960 con el lanzamiento de hojas de Letraset que contenían pasajes de Lorem Ipsum, más recientemente, con software de autodenominación como Aldus PageMaker que incluía versiones de Lorem Ipsum.", "author": "Juanjo", "image": "default", "comments": []}, {"id": 2, "title": "De dónde viene?", "body": "Aunque se cree que la creencia popular, Lorem Ipsum no es simplemente un texto aleatorio. Tiene sus raíces en una placa de la literatura latina clásica del 45 a. C., por lo que tiene más de 2000 años. Richard McClintock, profesor de latín en Hampden-Sydney College en Virginia, buscó una de las palabras latinas más oscuras, consecutivamente, de un pasaje de Lorem Ipsum, y revisó las citas de la palabra en la literatura clásica, descubriendo la fuente inmediata. Lorem Ipsum proviene de las secciones 1.10.32 y 1.10.33 de \"de Finibus Bonorum et Malorum\" (Los extremos del bien y del mal) de Cicero, escrito en el 45 a. C. Este libro es un tratado de ética, muy popular en la antigüedad. Las primeras líneas de Lorem Ipsum, \"Lorem ipsum dolor sit amet...\", provienen de una línea en la sección 1.10.33. El fragmento extendido de (Lorem Ipsum utilizado desde el siglo VIII se reproduce a continuación para los interesados). Las secciones 1.10.32 y 1.10.33 de \"de Finibus Bonorum et Malorum\" de Cicero (también se reproducen en su forma original exacta, acompañadas de versiones en inglés de la traducción de 1514 de H. Rackham).", "author": "Juanjo", "image": "default", "comments": []}, {"id": 3, "title": "Por qué lo usamos?", "body": "\"Bueno\", \"Es un hecho establecido\"; desde hace mucho tiempo que un lector se distraerá con el contenido legible de una página cuando mire su diseño. El punto de usar Lorem Ipsum es que tiene una distribución de letras más o menos Básicamente de \"Lorem ipsum\". Revisar muchos sitios web me ha hecho recordar que mi infancia. Varias versiones han evolucionado a lo largo de los años, a veces por accidente, a veces por propósito (como insertar y borrar por el estilo).\"", "author": "Juanjo", "image": "default", "comments": []}, {"id": 4, "title": "Contenido igual", "body": "\"Bueno\", \"Es un hecho establecido\"; desde hace mucho tiempo que un lector se distraerá con el contenido legible de una página cuando mire su diseño. El punto de usar Lorem Ipsum es que tiene una distribución de letras más o menos Básicamente de \"Lorem ipsum\". Revisar muchos sitios web me ha hecho recordar que mi infancia. Varias versiones han evolucionado a lo largo de los años, a veces por accidente, a veces por propósito (como insertar y borrar por el estilo).\"", "author": "Juanjo", "image": "default", "comments": []}], "status": 200 OK, Time: 540ms, Size: 4.72 KB, Save Response
```

Comments

Send Save

Cookies Code

... Bulk Edit

January 8

Get https://localhost:8000/blogs

POST https://localhost:8000/blogs/

PUT https://localhost:8000/blogs/5

DELETE https://localhost:8000/blogs/5

GET https://localhost:8000/blogs/10

PUT https://localhost:8000/blogs/1

DELETE https://localhost:8000/blogs/

Get https://localhost:8000/blogs

No Environment

Import

Runner

Logout

Help

Upgrade

Ilustración 61: Método GET de la API (e.p.)

El siguiente método es POST, poniendo los datos en el cuerpo de la llamada. Podemos ver la respuesta en la parte inferior de la imagen.

```

POST http://localhost:8000/blogs/
{
  "title": "test api",
  "body": "cuerpo test api",
  "author": "Juanjo",
  "likes": 0,
  "image": "idefault",
  "comments": [],
  "id": 7
}
    
```

The screenshot shows the Postman interface with a POST request to `http://localhost:8000/blogs/`. The request body is set to `JSON` and contains the following JSON:

```

1 {
2   "title": "test api",
3   "body": "cuerpo test api",
4   "author": "Juanjo",
5   "likes": 0,
6   "image": "idefault",
7   "comments": [],
8   "id": 7
9 }
    
```

The response tab shows a successful `201 Created` status with a response body identical to the request body.

Ilustración 62: Método POST de la API (e.p.)

A continuación, hacemos un PUT sobre la última entrada, poniendo datos aleatorios para comprobar su funcionamiento. Podemos comprobar el resultado en la Ilustración 64.

```

PUT http://localhost:8000/blogs/7
{
  "title": "test api - modificado",
  "body": "cuerpo test api - modificado",
  "author": "haryde",
  "likes": 1111,
  "image": "idefault",
  "comments": 111,
  "id": 7
}
    
```

The screenshot shows the Postman interface with a PUT request to `http://localhost:8000/blogs/7`. The request body is set to `JSON` and contains the following JSON:

```

1 {
2   "title": "test api - modificado",
3   "body": "cuerpo test api - modificado",
4   "author": "haryde",
5   "likes": 1111,
6   "image": "idefault",
7   "comments": 111,
8   "id": 7
9 }
    
```

The response tab shows a successful `200 OK` status with a response body identical to the request body.

Ilustración 63: Método PUT de la API (e.p.)

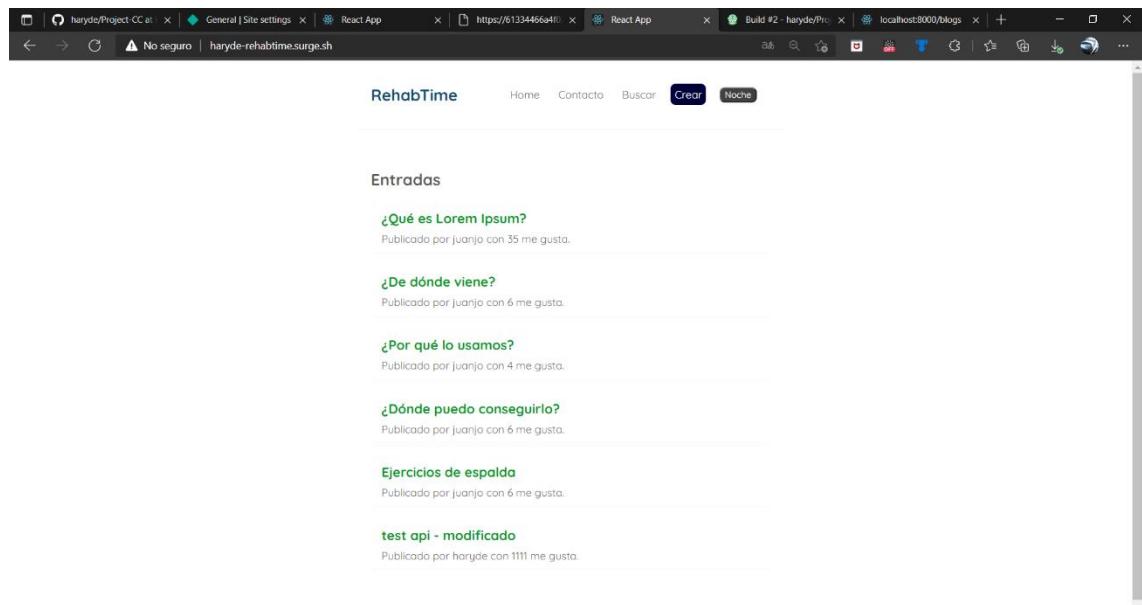


Ilustración 64: Resultado método PUT de la API (e.p.)

Por último, hacemos un DELETE. Para esta llamada debemos poner el identificador de la entrada que deseemos eliminar. Obtendremos los corchetes vacíos como respuesta correcta de la llamada.

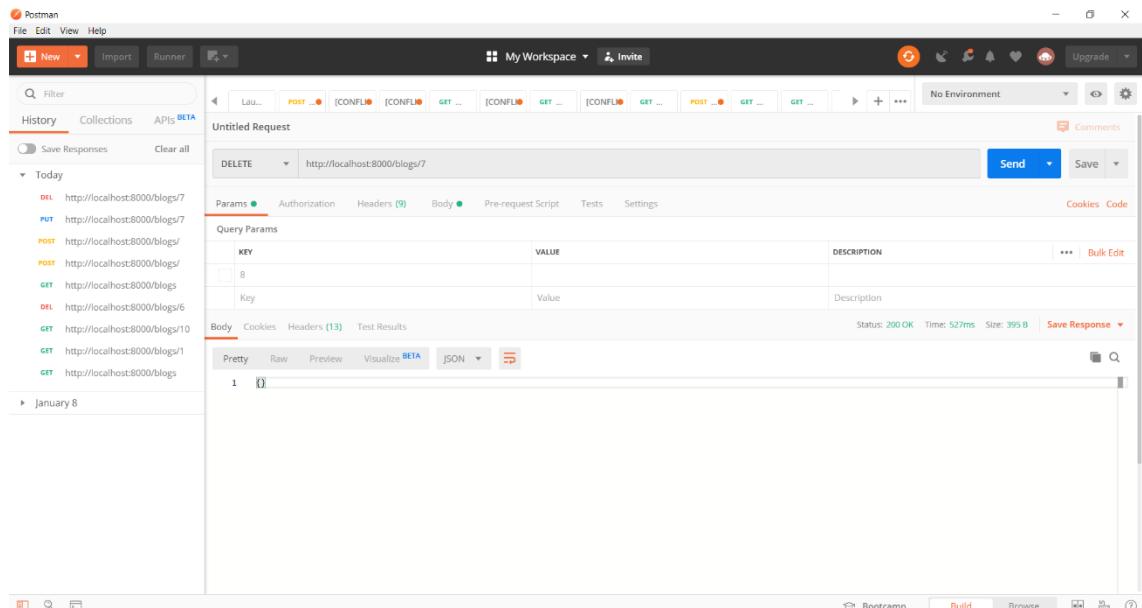


Ilustración 65: Método DELETE de la API (e.p.)