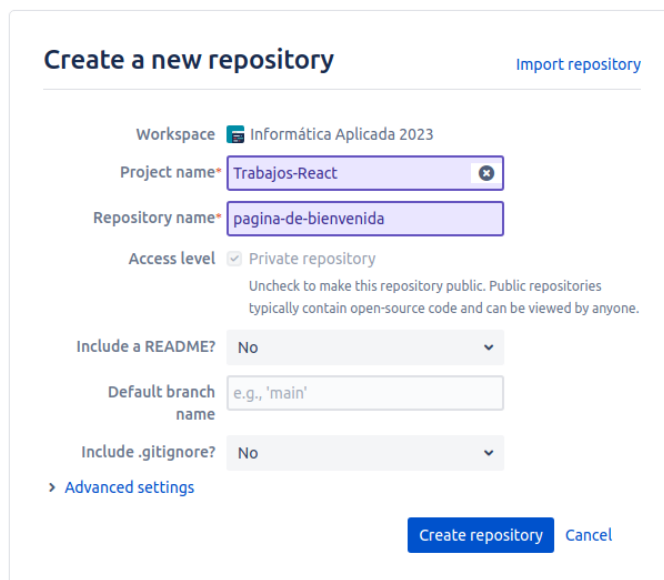


Práctico React

1. Nuevo proyecto y repositorio

Crear un nuevo proyecto y repositorio en Bitbucket. Pueden nombrar al proyecto "Trabajos-React" y al repositorio "pagina-de-bienvenida".

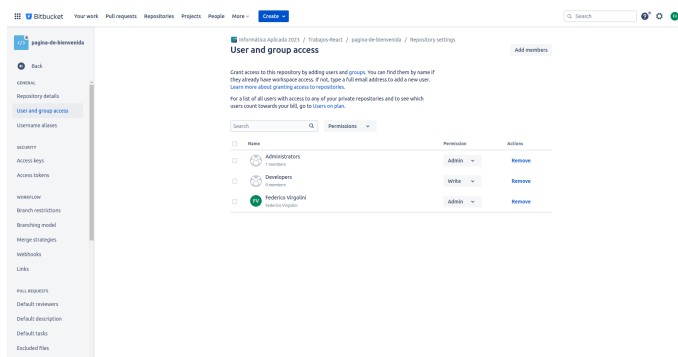
Nota: en la configuración que se les presenta al crear un nuevo repositorio, seleccionen las opciones de NO incluir el archivo README ni .gitignore; como se muestra en la siguiente imagen:



The screenshot shows the 'Create a new repository' form in Bitbucket. The workspace is 'Informática Aplicada 2023'. The project name is 'Trabajos-React' and the repository name is 'pagina-de-bienvenida'. The access level is set to 'Private repository'. The 'Include a README?' dropdown is set to 'No'. The 'Default branch name' is 'e.g., 'main''. The 'Include .gitignore?' dropdown is set to 'No'. There are links for 'Advanced settings', 'Create repository', and 'Cancel'.

2. Compartir el repositorio

Ahora vamos a agregar a más gente para que participe de este repositorio. Para ello, se debe ir a la configuración del repositorio (la opción de configuración se encuentra en la columna derecha de la página). Allí buscaremos la pestaña de "User and group access", lo que significa acceso de usuario y grupos. Esta pestaña de configuración debería tener la siguiente pinta:




Seleccionaremos la opción de agregar usuarios (arriba a la derecha), y allí ingresaremos las cuentas de mail de nuestro compañero/s y profesor. Recordar asignar los permisos de lectura y escritura.

Mail del profesor: fvirgolini@itr.edu.ar

3. Fetchear el repositorio

Clonar el repositorio en la computadora utilizando el comando que nos brinda bitbucket (recordar hacerlo en una ubicación específica):

Let's put some bits in your bucket

HTTPS  git clone https://fedevirgolini-itr@bitbucket.org/informatica-aplicada-20 

e ingresar al repositorio recién clonado ejecutando:

```
$ cd pagina-de-bienvenida
```

4. Creación de nueva aplicación

Una vez parados en el repositorio, vamos a crear una nueva aplicación de react. Para eso vamos a ejecutar el siguiente comando:

```
$ npx create-react-app my-app
```

La ejecución de este comando puede tardar un poco; especialmente la primera vez que se ejecuta. Esto se debe a que cuando se ejecuta por primera vez no solo tiene que ejecutar los paquetes solicitados, sino que también debe descargarlos.

Analicemos el comando anterior:

- **npx** es la abreviación que hace referencia a Node Package Executer. Es decir, esta parte del comando le indica a la computadora que vamos a estar ejecutando algunos paquetes (programas) de node.
- **Create-react-app** hace referencia a el paquete que vamos a estar ejecutando. Este paquete en particular se utiliza para crear una nueva aplicación de react desde cero.
- **my-app** hace referencia al nombre que le vamos a dar a nuestra aplicación.

5. Checkeo de archivos y primer commit

Una vez que se termine de ejecutar el comando ya deberíamos tener nuestra aplicación creada. Ejecutando los siguientes comandos, los outputs deberán ser semejantes a los que se indican a continuación:

```
$ ls
my-app
$ cd my-app/
$ ls
node_modules  package.json  package-lock.json  public  README.md  src
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
./
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

El output del último comando ejecutado nos está diciendo que todavía no hemos comiteado todos los archivos que acabamos de crear. Vamos a aplicar estos cambios a continuación

```
$ git add .
$ git commit -m "Commit inicial"
[master (root-commit) 63c61d4] commit inicial
18 files changed, 17590 insertions(+)
create mode 100644 my-app/.gitignore
create mode 100644 my-app/README.md
create mode 100644 my-app/package-lock.json
create mode 100644 my-app/package.json
create mode 100644 my-app/public/favicon.ico
create mode 100644 my-app/public/index.html
create mode 100644 my-app/public/logo192.png
create mode 100644 my-app/public/logo512.png
create mode 100644 my-app/public/manifest.json
create mode 100644 my-app/public/robots.txt
create mode 100644 my-app/src/App.css
create mode 100644 my-app/src/App.js
create mode 100644 my-app/src/App.test.js
create mode 100644 my-app/src/index.css
create mode 100644 my-app/src/index.js
create mode 100644 my-app/src/logo.svg
create mode 100644 my-app/src/reportWebVitals.js
create mode 100644 my-app/src/setupTests.js
```

Puede que el output obtenido no contenga los códigos como los del ejemplo, pero debería verse algo parecido. Ahora nuestro repositorio local debería estar un commit adelantado al repositorio remoto:

```
$ git status
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
  (use "git branch --unset-upstream" to fixup)

nothing to commit, working tree clean
```

6. La primera puesta en marcha

Ahora lo que haremos será levantar nuestra aplicación recién creada. Para ello ejecutaremos:

Nota: Para ejecutar correctamente el siguiente comando deben estar parados sobre la carpeta de la aplicación (en nuestro caso my-app); en caso contrario les va a tirar error.

```
$ npm start
Compiled successfully!

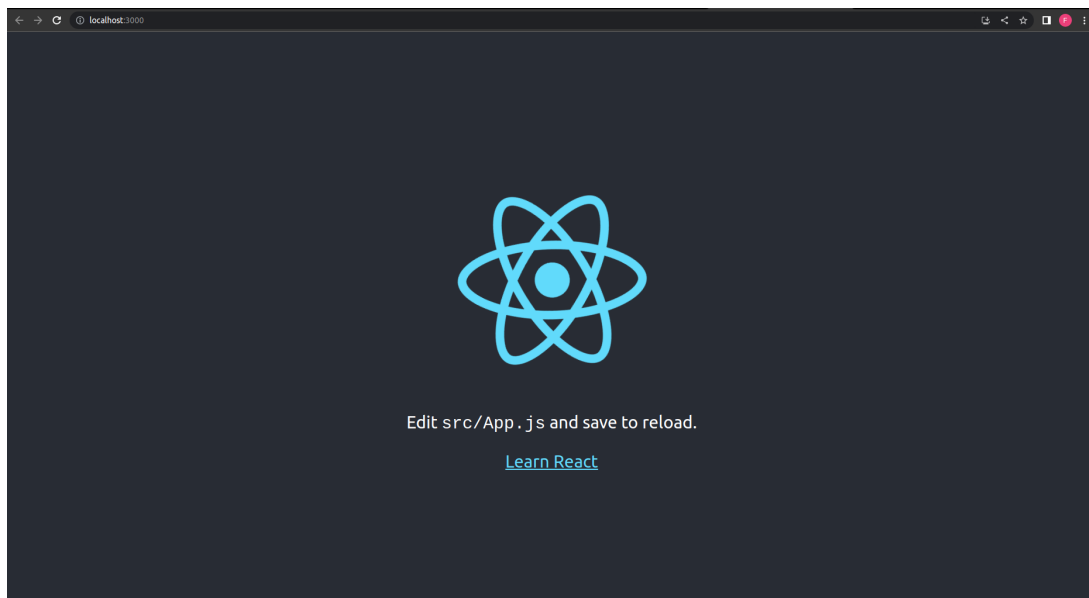
You can now view my-app in the browser.

Local:            http://localhost:3000
On Your Network:  http://192.168.0.148:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Si el comando se ejecutó correctamente, se debería abrir el navegador web mostrando una página similar a la siguiente:

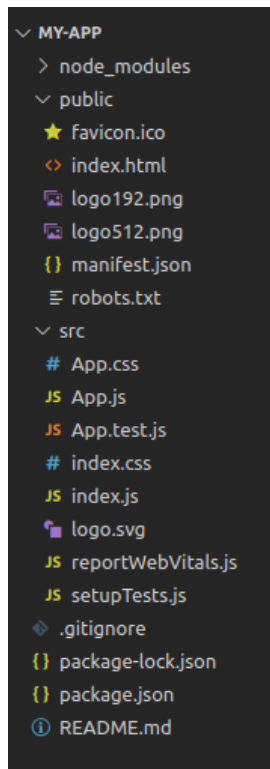


7. Empezando a leer código React

Ahora vamos a revisar los archivos creados. Empezaremos a tratar de entender como react nos organiza los archivos para trabajar. Para abrir el código ejecutaremos:

```
$ code .
```

A la derecha de la ventana de VS Code, podemos observar los archivos y su organización. Debería ser similar a la siguiente:



Vamos a abrir el archivo `./src/App.js` (es decir, el archivo llamado `App.js` que está en la carpeta `src`).

Ejercicio:

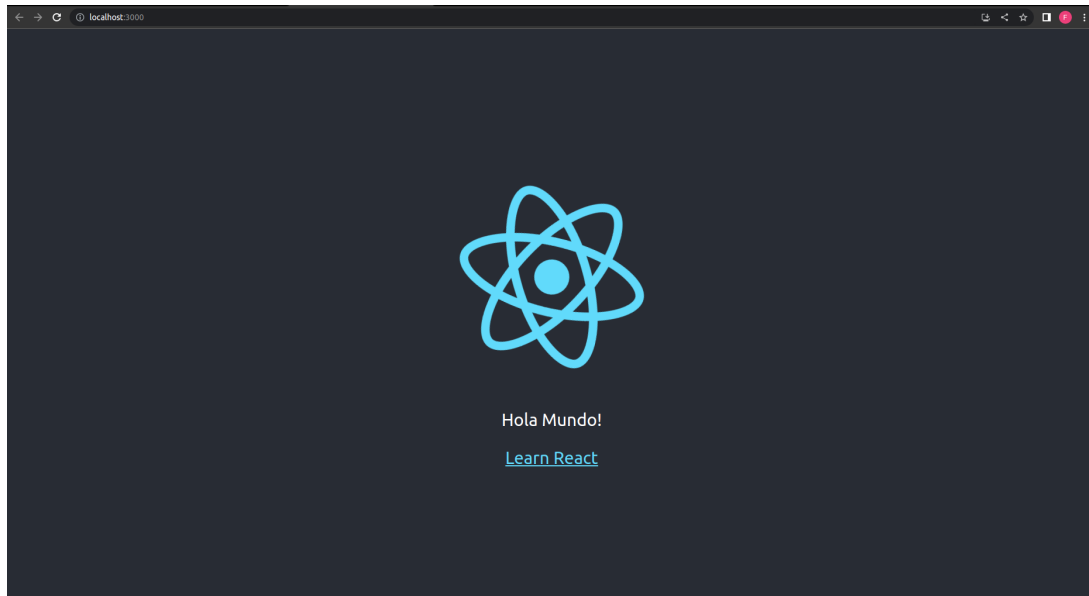
Leer detalladamente el contenido del archivo y tratar de entender el que hace y el como lo hace. Ayuda: pueden guiarse por la página que muestra el navegador al ejecutar la aplicación.

8. Empezando a modificar nuestra aplicación

Vamos a modificar la aplicación para que al ejecutar la página nos muestre el mensaje de 'Hola Mundo!'. Para ello, se debe modificar la línea 10 del archivo `./src/App.js`. Podemos escribir algo como:

```
<p>Hola mundo!</p>
```

Ahora al ejecutar `$ npm start` deberíamos tener un resultado como el siguiente:



Como realizamos un gran cambio a nuestro programa y todo funciona correctamente debemos comitearlo. Para ello ejecutaremos:

```
$ git add .  
$ git commit -m "Agregado de mensaje Hola Mundo!"
```

Ahora nuestro repositorio local debería estar dos commits más adelantado que el repositorio remoto.

9. Mensaje de bienvenida

Lo siguiente que haremos será agregar una nueva variable denominada `name`. Vamos a declarar esta variable dentro de la función `App()`, pero antes del `return`.

Por otro lado, vamos a escribir un mensaje de bienvenida para los usuarios que visiten nuestra página. Esto lo haremos utilizando algún tag de título (`h3`). Colocaremos el título por encima del logo. Se debe mostrar la leyenda "Bienvenidos a la página web de [nombre]".

El código resultante debería ser parecido a lo siguiente:

```
import logo from "./logo.svg";  
import "./App.css";  
  
function App() {  
  const name = "Juan Perez";  
  
  return (  
    <div className="App">  
      <header className="App-header">  
        <h3>Bienvenido a la página web de {name}</h3>  
        <img src={logo} className="App-logo" alt="logo" />  
        <p>Hola Mundo!</p>  
        <a  
          className="App-link"
```

```
        href="https://reactjs.org"
        target="_blank"
        rel="noopener noreferrer"
      >
        Learn React
      </a>
    </header>
  </div>
);
}

export default App;
```

Comitéa los cambios realizados utilizando un mensaje descriptivo.

10. Creando funciones flecha

Notemos que nuestra página puede dar un saludo de manera incorrecta a muchos usuarios. La programemos para que al menos nuestra página contemple el género masculino y femenino. Para ello vamos a agregar una variable `is_female` y la vamos a inicializar en `true`. Luego, haremos una "arrow function" llamada `welcomeGender` que nos devuelva el encabezado correspondiente en cada caso.

Esto se implementa de la siguiente manera:

```
import logo from "./logo.svg";
import "./App.css";

function App() {
  const name = "Juan Perez";
  const is_female = true;

  const welcomeGender = () => {
    return is_female ? "Bienvenida" : "Bienvenido";
  };

  return (
    <div className="App">
      <header className="App-header">
        <h3>
          {welcomeGender()} a la página web de {name}
        </h3>
        <img src={logo} className="App-logo" alt="logo" />
        <p>Hola Mundo!</p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}
```

```
        </header>
      </div>
    );
  }

  export default App;
```

Por el momento, solo podemos modificar el valor de `is_female` desde el código.

Ejercicio:

Lee detenidamente el código para entender lo que está sucediendo al ejecutarlo.

Comiteá los cambios realizados utilizando un mensaje descriptivo.

11. Introducción a Hooks

Los hooks son herramientas que nos brinda React para poder manejar de forma correcta los estados y comportamientos de nuestra página. Existen hooks por defecto, pero también podemos crear los nuestros. Es un concepto que se entiende mejor con la práctica, así que empecemos a utilizarlos.

En el paso 10 creamos una nueva variable declarándola como `const`. Esto no es correcto, ya que queremos modificar su valor. Sin embargo, tampoco es correcto declararla como una variable `let` (de hecho la página no funcionaría). Pero, por qué es esto?

Cuando hablamos de variables en nuestra página web más bien nos estamos refiriendo a estados. Por ejemplo en nuestro caso: Nuestra página podría estar en 2 estados:

- Mostrando el mensaje con la palabra "Bienvenido".
- Mostrando el mensaje con la palabra "Bienvenida".

Para ello, se utiliza el hook (o la función) `useState()`; que como su nombre lo indica, nos está denotando que usaremos estados en nuestra página. Este hook se utiliza de la siguiente manera:

```
const [is_female, setIs_female] = useState(true);
```

Reemplazá la declaración de la variable `is_female` por la línea superior.

La función `useState()` devuelve una lista de dos cosas:

- La primera es la variable que contiene el estado. En nuestro caso: `is_female`.
- La segunda es una función que utilizaremos para cambiar dicho estado. En nuestro caso: `setIs_female()`.

Notemos que la función toma un argumento al ser utilizada. Este argumento es el estado inicial que tendrá nuestra variable. Para conservar el valor inicial propuesto en el paso 10, inicializamos el estado en `true`.

Comiteá los cambios realizados utilizando un mensaje descriptivo.

12. Nuestra primera interacción

Hasta ahora no podemos interactuar con nuestra página. Tenemos un mensaje de bienvenida que es capaz adaptarse a algunos usuarios, y nuestra página cuenta con los estados para poder adaptarse al cambio; pero el usuario no puede indicarle a la página que cambie.

La primera solución que propondremos será un botón. Este botón lo que hará será modificar el estado de nuestra página web.

Reemplazá el link "learn React" que aparece en nuestra página por el tag del botón:

```
<button>Modificar mensaje</button>
```

Si querés podés agregarle algún estilo al botón. Para ello debés modificar el archivo `App.css`. A su vez debés agregarle un nuevo atributo al tag del boton denominado `className`, que le indicará cual estilo debe utilizar. Por ejemplo:

```
<button className=".button">Modificar mensaje</button>
```

Una vez creado el botón ya podemos verlo en nuestra página. Sin embargo, no realiza ningún comportamiento. Esto es porque no tiene el atributo `onClick`, que es que determina el comportamiento del botón al ser presionado.

Agregá el siguiente atributo al tag del botón:

```
onClick={/*Completar aqui*/}
```

Donde lo indica el comentario, agregá una función flecha que cambie el estado de la página según el siguiente comportamiento:

- Si el estado `is_female` es verdadero: asignarle el valor falso.
- Si el estado `is_female` es falso: asignarle el valor verdadero.

Una vez implementado correctamente este paso tendríamos nuestra página funcionando correctamente, con la capacidad de alternar el mensaje de bienvenida.

Comiteá los cambios realizados utilizando un mensaje descriptivo.

13. Manejando correctamente el click del botón

Lo que hicimos en el paso anterior no está del todo correcto. La página funciona de acuerdo a lo que queríamos, pero la implementación puede mejorarse. En este paso programaremos de manera más adecuada el comportamiento del botón.

El problema con el paso anterior es que estamos implementando las acciones que realiza el botón "dentro" del documento html. Esto puede ser una mala práctica, ya que todo el comportamiento lógico de cada componente debe realizarse en el cuerpo de la misma.

para ello reemplaremos lo que toma el atributo `onClick` del tag de nuestro botón como se muestra a continuación:

```
<button onClick={handleClick}>Modificar mensaje</button>
```

Ahora, implementá la función `handleClick()` en el cuerpo de la componente `App`

Ayuda: Esta función también será una función flecha. Podés guiarte por la función creada en el paso 10.

Una vez finalizado este paso, tendremos implementado de forma correcta nuestro primer botón.

Comiteá los cambios realizados utilizando un mensaje descriptivo.

14. Utilizando otros tipos de input

Investigá que otros tipos de inputs pueden existir en páginas web.

Particularmente investigá sobre el tag `<select>`. ¿Qué nos permite hacer ese tag? ¿Cómo se implementa?

Lo que haremos ahora será reemplazar el botón por una lista desplegable que nos permita seleccionar dos opciones, donde cada opción corresponda a un saludo de bienvenida.

Para mayor prolijidad podríamos hacer esta nueva versión en una nueva rama del repositorio. Por cuestiones de simplicidad, en este práctico continuaremos modificando este archivo, pero podés hacerlo en una nueva rama si lo deseás.

Vamos a borrar las cosas que no serán necesarias para esta implementación. Debés borrar: el logo, el botón (con la función que maneja el click) y la función flecha `welcomeGender`. También dejaremos de usar el estado `is_female`.

Ahora lo que haremos será crear un nuevo estado denominado `welcomeGender` y le asignaremos el nombre `setWelcomeGender` a su función de modificación. Iniciaremos este estado como `"Bienvenida"`

Ahora crearemos la etiqueta `<select>` para colocar nuestra lista de selección. recordá que toda etiqueta debe cerrarse, por lo que también debemos agregar `</select>`.

Dentro de esta nueva etiqueta estableceremos nuestras opciones. Las opciones tienen la siguiente estructura:

```
<option value="valor que toma esta opción">Texto de la opción</option>
```

Ahora agregá las opciones femenino y masculino a la etiqueta `<select>` que creamos anteriormente. Deben tener los valores `"Bienvenida"` y `"Bienvenido"` respectivamente.

Agregá el atributo `onChange={}` a la etiqueta `select` y creá una función flecha que cambie el estado al valor correspondiente. Ayuda: la función debe tomar un argumento `e`, y para obtener el valor de la opción seleccionada se utiliza `e.target.value`.

Comiteá los cambios realizados utilizando un mensaje descriptivo.

15. Rellenando nuestra página

Le agregaremos un poco de contenido a nuestra página, para que sea un poco más interesante. Para ello utilizaremos los tags ``, `<h1>...<h6>`, `<p>`, `<body>`, `<a>`, ``, etc. Pueden ver que otros tag existen para archivos html haciendo click [aquí](#).

Por el momento basta con agregarle un título, imagen y un pequeño párrafo. Para agregar una imagen, debemos utilizar su tag de la siguiente forma:

```
<img alt="" src="" />
```

Siendo `alt` el texto alternativo que mostrará la página en caso de no poder cargar la imagen (este atributo es obligatorio), y `src` la fuente de donde se obtendrá la imagen. Esta fuente puede ser tanto una ruta a la imagen (en caso que la tengamos guardada en la computadora), o también puede ser el link en el que se encuentre la imagen.

Podés probar de utilizar el siguiente link para su imagen:

```
"https://www.itr.edu.ar/images/LogoITR-60.jpg"
```

Agregá también:

- Un título o subtítulo a la página, utilizando `<h1>...<h6>`.
- Un párrafo conciso, utilizando `<p>` o cualquier otra.

Comiteá los cambios realizados utilizando un mensaje descriptivo.

16. Creando una nueva componente

Hasta ahora estamos siempre trabajando en la componente `App`. Para modularizar mejor nuestro código React nos permite crear diferentes componentes, con la idea de que cada componente cumpla una acción específica.

Para nuestra página crearemos una nueva componente llamada `Welcome`. En esta nueva componente incorporaremos todo lo relacionado a nuestro saludo de bienvenida, y de esta forma no se mezclará con el resto del contenido de nuestra página.

Para ello crearemos un nuevo archivo en la carpeta `src`. Este nuevo archivo se llamará `Welcome.js` y contendrá el código de nuestra nueva componente. Lo primero que escribiremos en nuestro archivo recién creado será:

```
function Welcome() {  
  return (  

```

```
    <div>

    </div>
  );
}

export default Welcome;
```

Allí estamos creando nuestra nueva componente `Welcome`. La última línea, que dice `export default Welcome`, se utiliza para poder importar esta componente desde otros archivos, para poder utilizarla luego. Por ahora estamos devolviendo un `<div>` vacío; empecemos a agregarle cosas.

La idea es eliminar todo lo que tiene que ver con el mensaje de bienvenida de la componente `App` (ubicada en el archivo `App.js`). Por lo que antes de borrarlo en la componente `App` debemos copiarlo a esta nueva componente `Welcome`. Concretamente, debemos pasar la lista de selección y el título que da la bienvenida al archivo recién creado. Sin embargo, no solo basta con pasar los elementos html; ya que habíamos programado cierta lógica sobre ellos y es necesario también cambiar de ubicación los estados y constantes correspondientes para su correcto funcionamiento.

Una vez realizado esto, tenemos que utilizar esta componente. Para ello vamos a volver al archivo `App.js`. Allí vamos a importar la componente recién creada colocando la siguiente línea en la parte superior del archivo:

```
import './Welcome';
```

Luego, podremos llamar a nuestra componente utilizando el tag `<Welcome>` como si fuera un elemento más de html.

Si todos los pasos se realizaron correctamente, obtendremos una página idéntica a la del punto anterior. Si bien su apariencia y comportamiento no cambió, la estructura del código de nuestra página se encuentra más organizado.

Comitéa los cambios realizados utilizando un mensaje descriptivo.

17. Llamando a cada quién por su nombre

En este paso personalizaremos un poco más nuestra página. El objetivo final de este paso es que el usuario sea capaz de ingresar su nombre o apodo para luego dar un mensaje de bienvenida personalizado para cada persona que entre a la página.

para ello, en primer lugar vamos a tener que brindarle al usuario un lugar para que puede ingresar su nombre. Colocá la siguiente etiqueta en la componente `Welcome`, justo antes de la lista desplegable:

```
<input />
```

Notar que cerramos el tag de una forma particular. Esta es otra manera de indicar en un archivo html que cierre el tag. Esto solo lo podremos hacer cuando no exista contenido entre el tag de inicio y de cierre de la etiqueta.

Si ejecutamos la página web, veremos que ya nos aparece un cuadro de texto donde el usuario puede ingresar cualquier tipo de caracter. Sin embargo, nada sucede al ingresar algún texto; esto pasa porque todavía no le especificamos ningún comportamiento.

Para poder saludar al usuario por su nombre, debemos crear un nuevo estado. En este nuevo hook guardaremos el nombre del usuario. Creá este nuevo estado con nombre `username` y función de asignación `setUsername`, e inicializalo como un string vacío.

Una vez creado el estado, podremos agregar el atributo `onChange` a la etiqueta input. De manera semejante al paso 14, a este atributo le debemos asignar una función flecha que cambie el estado `username` con el contenido del cuadro de texto.

Ya tenemos nuestro cuadro de texto para que el usuario ingrese su nombre, y tenemos un estado que cambia dependiendo lo que se ingresa al cuadro de texto. Ahora solo queda mostrarle al usuario el mensaje personalizado. Eso lo dejo a criterio del desarrollador (es decir, de cada uno/a), pero una forma de hacerlo es la siguiente:

```
<h2>  
  Hola {username}!  
</h2>
```

Una vez terminado esto, nuestra página debería saludar al usuario por el nombre/apodo que ingresa en el cuadro de texto.

¿Qué problemas tiene la implemetación que acabamos de hacer? ¿Cómo se podría mejorar?

Comiteá los cambios realizados utilizando un mensaje descriptivo.