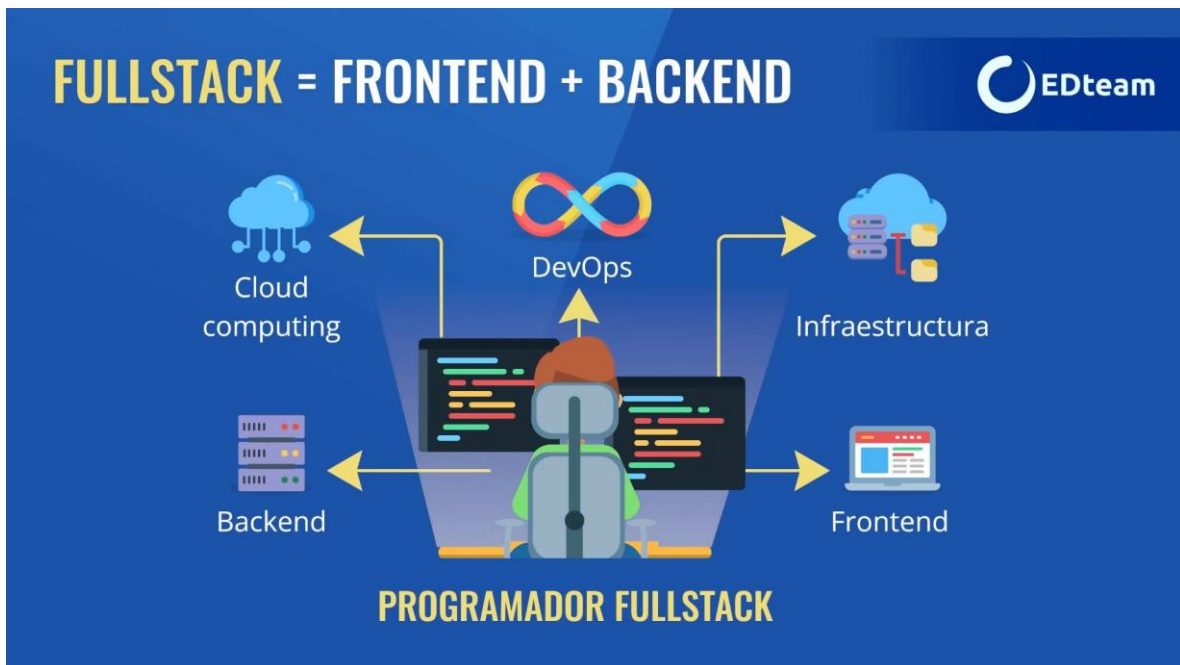


MANUAL: EJEMPLO APP FULL STACK: JAVA-SPRING-MySQL- REACT



Contenido

Contenido	2
Requisitos	4
1) Vamos a https://start.spring.io/	4
2) Abrimos el proyecto en IntelliJ IDEA.....	6
3) Creando la clase User y conectando con MySQL	7
3.1) Creamos interfaz.....	8
3.2) Creamos base de datos en MySQL	10
3.3) Regresamos a colocar el nombre de la base de datos en application.properties	10
3.4) Probamos conexión y revisamos creación de la tabla	11
4) @PostMapping Creamos clase controlador: Enviar datos a la base de datos	12
4.1) Revisar funcionamiento con postman.....	13
4.2) Damos de alta nuestro primer registro	15
5) @GetMapping Recibir datos de la base de datos.....	17
6) Creamos app en react.....	18
6.1) React snippet (extensión para programar mas rápido)	20
6.2) Creamos Navbar Usando Bootstrap	21
6.3) Importamos Navbar a App.js	27
6.4) Editar Navbar.....	29
6.5) Creamos Home con Bootstrap	30
7) AXIOS GET: Mostrar información	32
7.1) ¿Qué es axios?	33
7.2) Creamos hooks	33
7.3) ¿Qué es un hook?	34
7.4) ¿Qué es un prop?	35
7.5) Función cargar usuarios	35
7.6) @CrossOrigin.....	36
7.7) Result.data	37
7.8) ¿Por qué aun no sale la información de la base de datos?	37
7.9) Editando Home.js.....	38
8) Enrutamiento REAC-ROUTER-DOM	42

8.1)	Creando botones	42
8.2)	AddUser.js	43
9)	Creamos registro de usuario con React	49
10)	Almacenar información del usuario dentro del estado	52
11)	AXIOS POST: Publicar datos en la base de datos	57
11.1)	useNavigate	59
11.2)	Botón Cancel.....	61
12)	Manejo de excepciones de spring boot.....	62
12.1)	Probando las excepciones con Postman	64
13)	PUTMapping: editar usuario	66
13.1)	Probando con Postman	66
14)	@DeleteMapping: borrar usuario con id específico.....	69
14.1)	Probando con Postman	70
15)	AXIOS PUT: editar usuario.....	72
16)	AXIOS DELETE : borrar usuario	78
17)	Ver un usuario específico	80
17.1)	Método 1 ViewUser.js	¡Error! Marcador no definido.

Requisitos

- Conocimientos de Java y Spring
- Conocimientos de JavaScript y React
- Editor de texto como IntelliJ IDEA
- MySql database
- Editor de texto como VS Code
- Postman

1) Vamos a <https://start.spring.io/>

Project: maven, Language: java, Spring boot: el que este por defecto o el que sea el caso, colocamos un nombre al grupo, colocamos nombre y una descripción Packing: jar y la versión de java debe ser menor o igual a la que tenemos.

The screenshot shows the Spring Boot Start configuration page. The following elements are highlighted with red boxes:

- Project:** The **Maven** radio button is selected and highlighted.
- Language:** The **Java** radio button is selected.
- Spring Boot:** The **3.0.6** radio button is selected.
- Project Metadata:** A large red box encompasses the input fields for **Group** (com.ejemploFull), **Artifact** (fullstack-backend), **Name** (fullstack-backend), **Description** (Backend application), and **Package name** (com.ejemploFull.fullstack-backend).
- Packaging:** The **Jar** radio button is selected and highlighted.
- Java:** The **17** radio button is selected and highlighted.

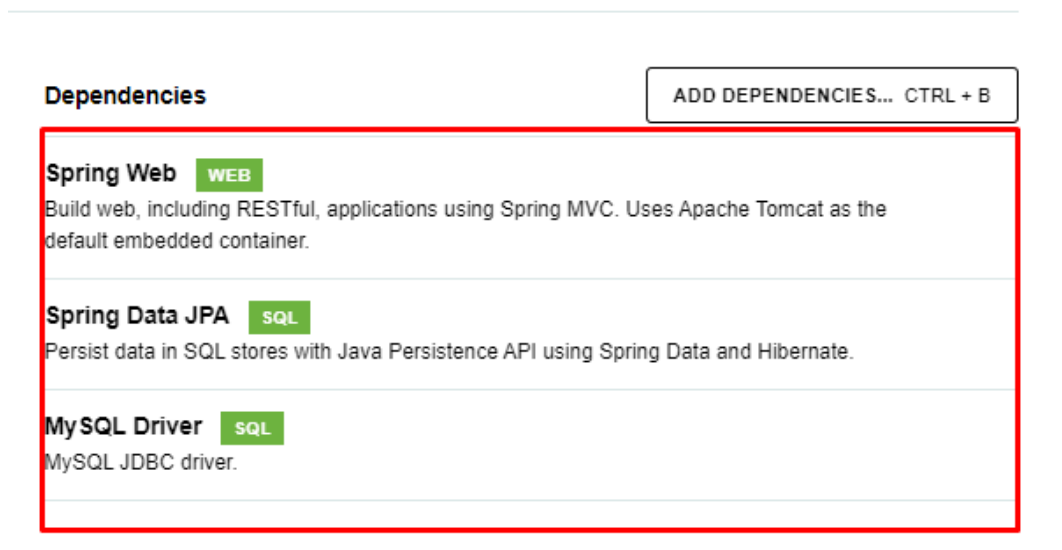
Explicación del empaquetado:

JAR (Java ARchive): Es un archivo empaquetado que contiene todos los archivos necesarios para ejecutar una aplicación de forma autónoma, incluido el código fuente, las bibliotecas, los recursos y los metadatos. El archivo JAR se puede ejecutar en cualquier plataforma que tenga instalado Java y solo se necesita un comando para ejecutarlo.

WAR (Web ARchive): Es un archivo empaquetado que contiene todos los archivos necesarios para ejecutar una aplicación web. Esto incluye el código fuente, las bibliotecas, los recursos y los metadatos, así como los archivos HTML, CSS y JavaScript que se utilizan para crear la interfaz de usuario. El archivo WAR se despliega en un servidor web que se encarga de ejecutar la aplicación y proporcionar una interfaz web para los usuarios.

En resumen, si estás construyendo una aplicación web con Spring Boot, generalmente se utiliza un archivo WAR para empaquetar y desplegar la aplicación en un servidor web. Si estás construyendo una aplicación independiente que no requiere un servidor web, entonces un archivo JAR puede ser suficiente

Descargamos las dependencias

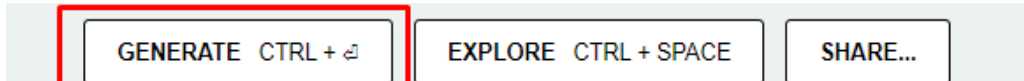


Spring Web se utiliza para desarrollar aplicaciones web con Spring. Proporciona herramientas para manejar solicitudes HTTP y respuestas, y es compatible con diferentes plantillas de vista, como Thymeleaf y JSP.

Spring Data JPA es una biblioteca de Spring que simplifica el acceso a bases de datos relacionales a través del uso de JPA (Java Persistence API). Proporciona una capa de abstracción sobre JPA, lo que significa que los desarrolladores no necesitan preocuparse por la implementación subyacente de JPA. Spring Data JPA simplifica el trabajo con JPA al proporcionar una API fácil de usar y al reducir el código repetitivo.

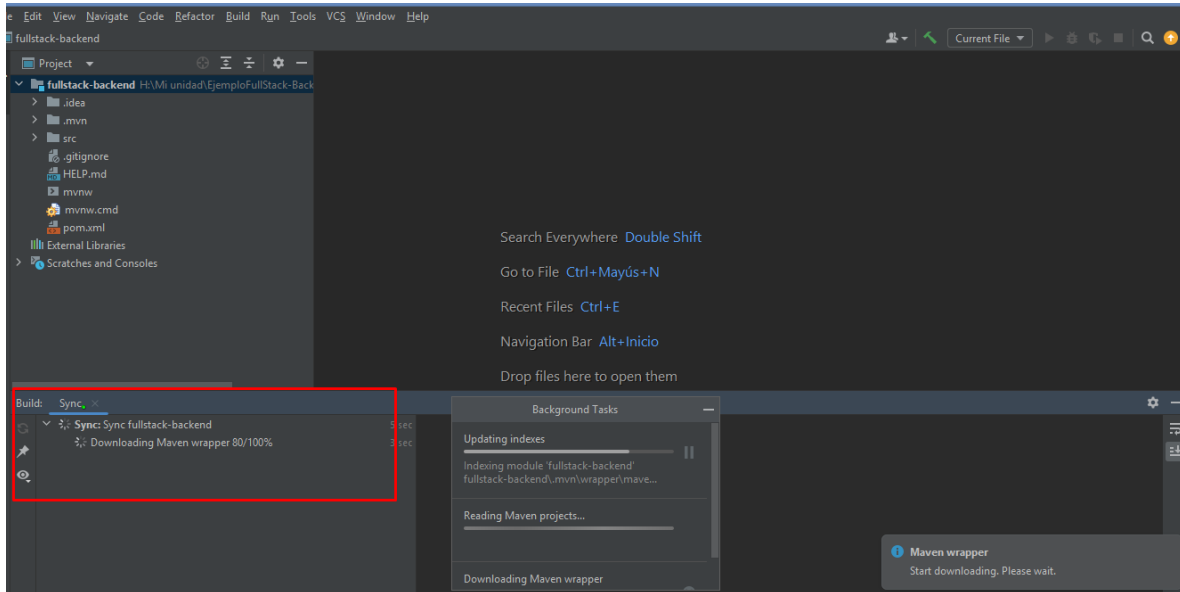
MySQL Driver es un controlador JDBC que permite a las aplicaciones Java interactuar con una base de datos MySQL. Spring Data JPA utiliza el controlador JDBC para conectarse a la base de datos MySQL y realizar operaciones de CRUD (Crear, Leer, Actualizar y Borrar) en la base de datos.

Y generamos el archivo

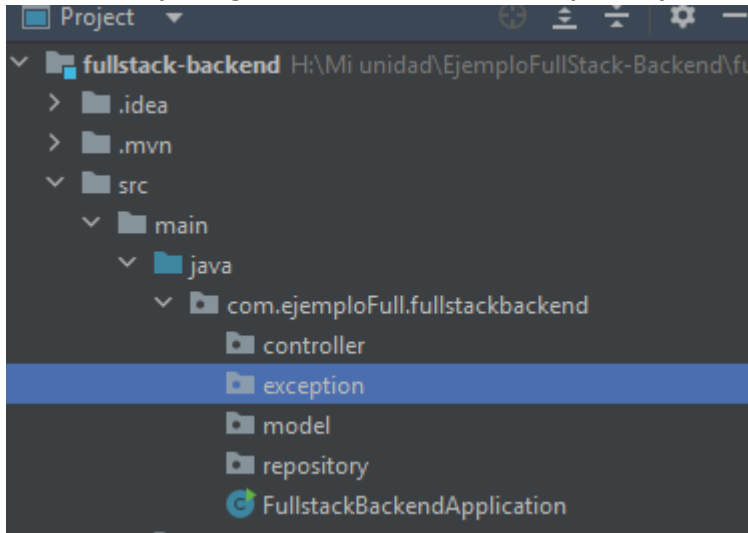


2) Abrimos el proyecto en IntelliJ IDEA

En automático se actualizarán las dependencias

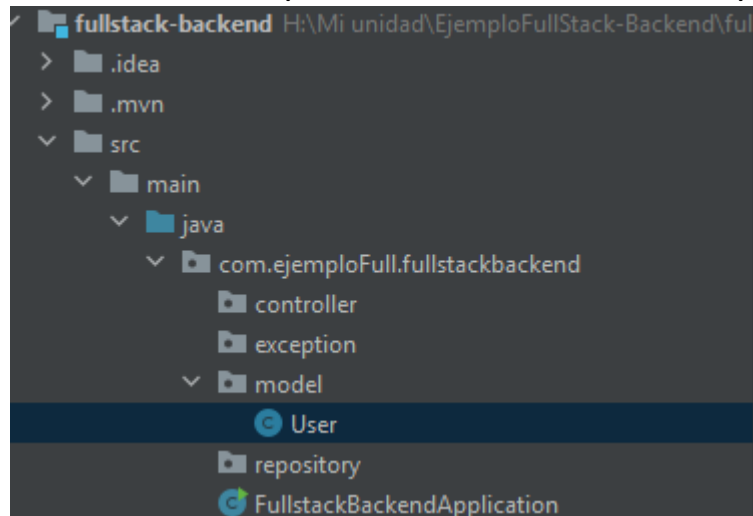


Creamos un package con el nombre model , repository, controller y exception

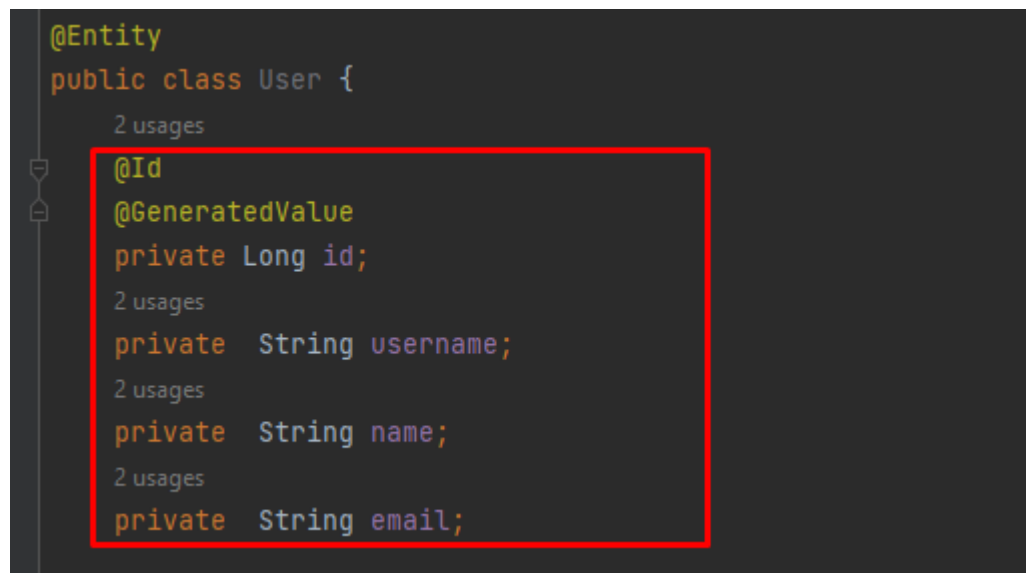


3) Creando la clase User y conectando con MySQL

Creamos la clase User (este será el nombre de nuestra tabla)



Definimos sus parámetros y generamos Getter y Setter de cada uno



```
package com.ejemploFull.fullstackbackend.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
```

```

@Entity
public class User {
    @Id
    @GeneratedValue
    private Long id;
    private String username;
    private String name;
    private String email;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

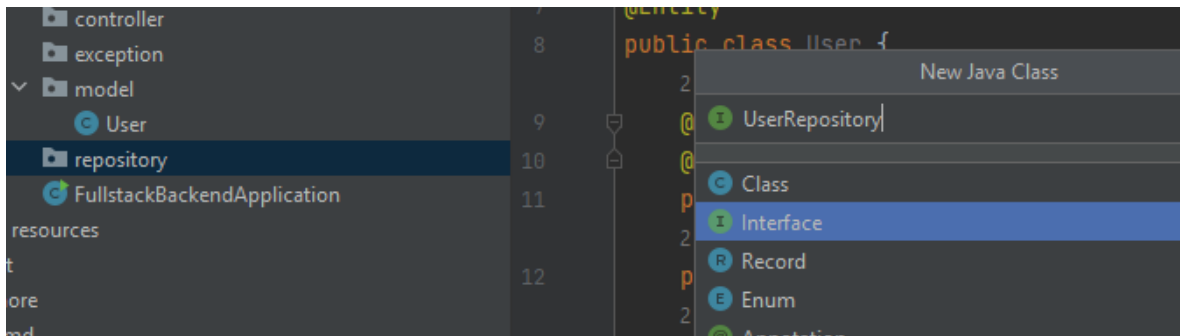
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}

```

3.1) Creamos interfaz

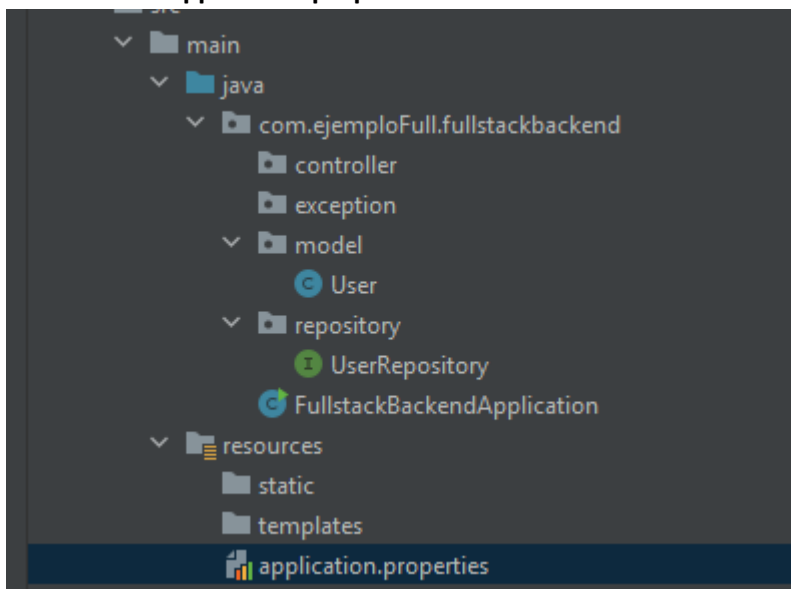
Creamos una interfaz en repository con el nombre UserRepository



Heredamos de la clase `JpaRepository`, entre `<>` colocamos el nombre de la clase (tabla) que vamos a mandar a MySQL seguido de una coma (,) y el tipo de dato en este caso `Long` que es el tipo de dato del Id, puede marcar error en el nombre de la clase solo basta con importar la clase.

```
public interface UserRepository extends JpaRepository<User, Long> {  
}
```

Nos vamos a `application.properties`

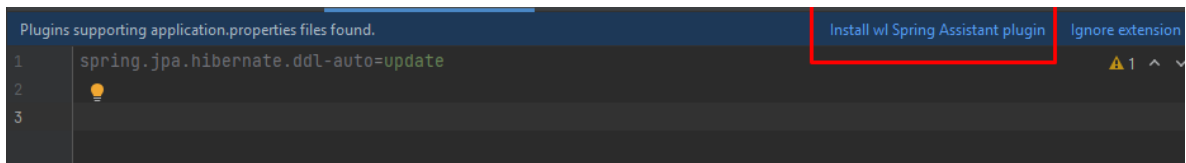


Y escribimos las siguientes líneas de código:

```
spring.datasource.url= jdbc:mysql://localhost:3306  
spring.jpa.hibernate.ddl-auto=update
```

Es la dirección de nuestra base de datos local

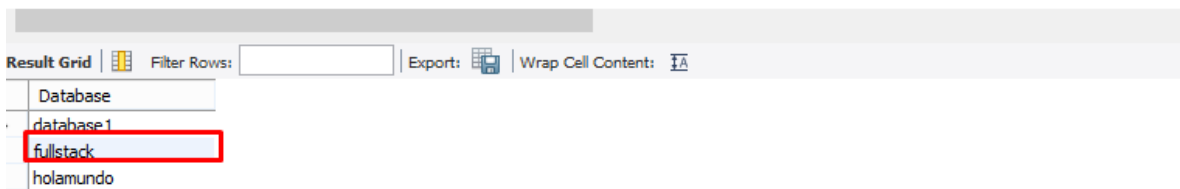
Tal vez sea necesario descargar el plugin



3.2) Creamos base de datos en MySQL

Vamos a MySQL y creamos nuestra base le pondremos fullstack

```
1 create database fullstack;  
2 show databases;  
3
```



3.3) Regresamos a colocar el nombre de la base de datos en application.properties

spring.datasource.url= jdbc:mysql://localhost:3306/fullstack

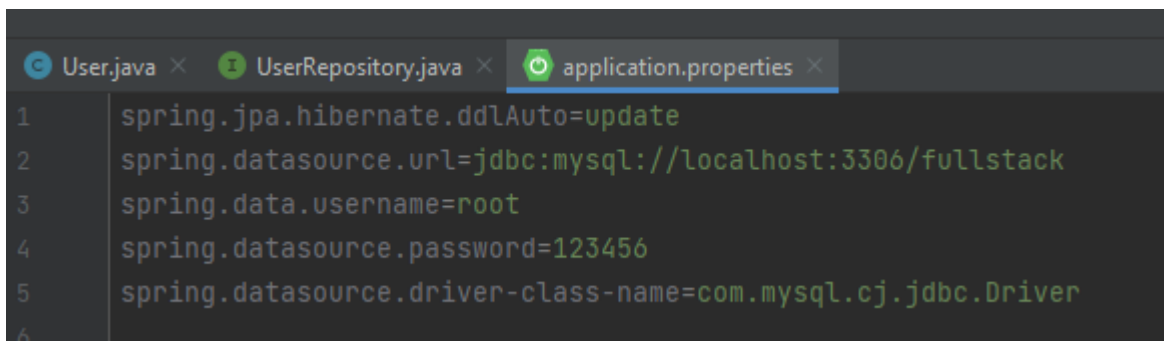
spring.datasource.username =root

spring.datasource.password=123456

spring.jpa.hibernate.ddl-auto=update

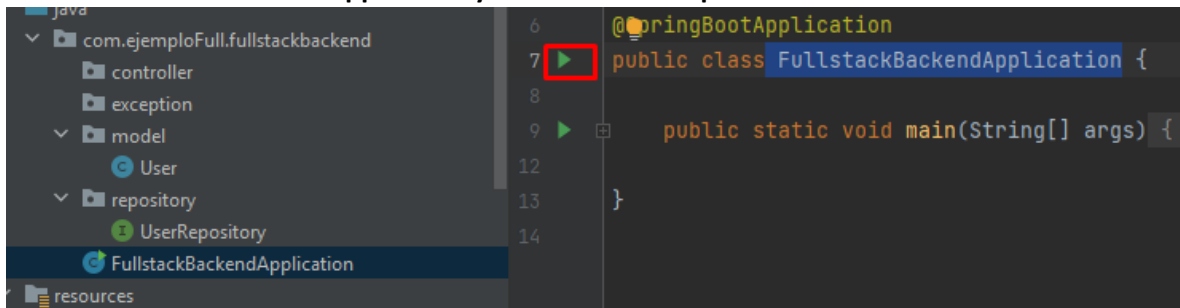
Nombre base de datos

Nombre de usuario y contraseña de nuestra base de datos



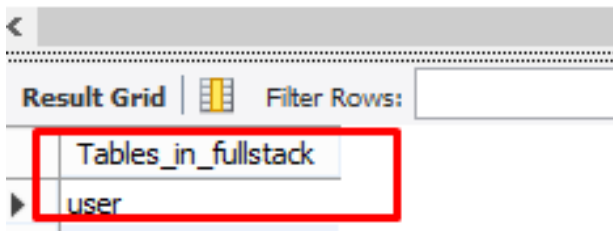
3.4) Probamos conexión y revisamos creación de la tabla

Vamos a FullstackBackendApplication y damos en correr para revisar la conexión



Vamos a MySQL y podemos revisar que se creó la tabla

```
1 • use fullstack;  
2 • show tables;  
3
```



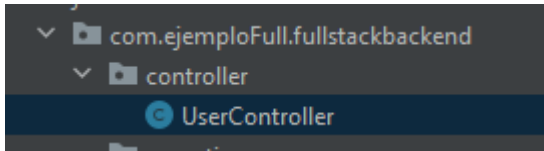
Y su contenido

```
2 • desc user;
```

The screenshot shows the MySQL command line interface with the 'Result Grid' tab selected. The output of the 'desc user;' query is displayed in a table with columns: Field, Type, Null, Key, Default, and Extra.

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	
email	varchar(255)	YES		NULL	
name	varchar(255)	YES		NULL	
username	varchar(255)	YES		NULL	

4) @PostMapping Creamos clase controlador: Enviar datos a la base de datos



```
package com.ejemploFull.fullstackbackend.controller;

public class UserController {
}
```

@RestController se usa en Spring para crear controladores que manejan solicitudes HTTP y devuelven objetos JSON como respuesta. Es una abreviatura conveniente para @Controller y @ResponseBody juntos. Los métodos anotados con @RequestMapping en un controlador RestController devuelven objetos de dominio directamente en lugar de una vista renderizada. Los datos se convierten automáticamente en JSON utilizando Jackson o cualquier otra biblioteca de serialización. En resumen, RestController se utiliza para crear servicios web RESTful en Spring.

@Autowired: esta anotación se utiliza para inyectar automáticamente una dependencia en un componente de Spring. Se puede aplicar a variables, constructores y métodos setter.

@PostMapping: esta anotación se utiliza para mapear solicitudes HTTP POST a métodos de controlador específicos en una aplicación Spring MVC. Permite manejar solicitudes POST en un método de controlador específico y devolver una respuesta.

@RequestBody: esta anotación se utiliza para vincular el cuerpo de la solicitud HTTP con un objeto en un método de controlador en Spring MVC. Es útil cuando se espera que el cuerpo de la solicitud contenga un objeto serializado en formato JSON, XML, entre otros.

Implementamos estas anotaciones

```
package com.ejemploFull.fullstackbackend.controller;

import com.ejemploFull.fullstackbackend.model.User;
import com.ejemploFull.fullstackbackend.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

    @Autowired
    private UserRepository userRepository;
    //private Interfaz Nombre;

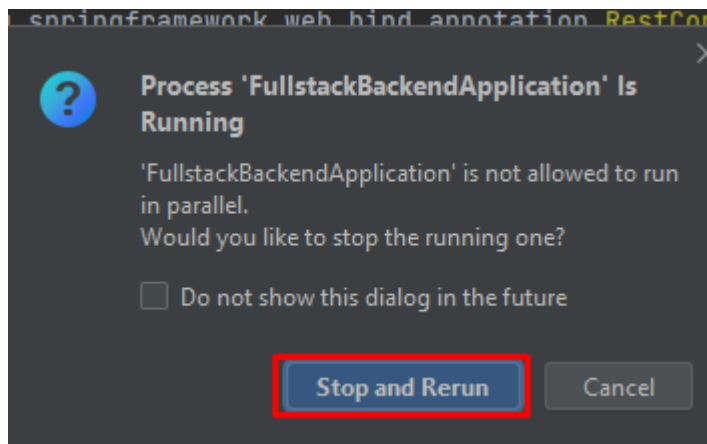
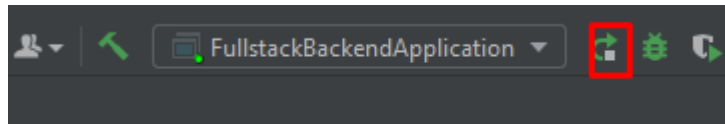
    @PostMapping("/user")
```

```

    User newUser(@RequestBody User newUser) {
        return userRepository.save(newUser);
    }
}

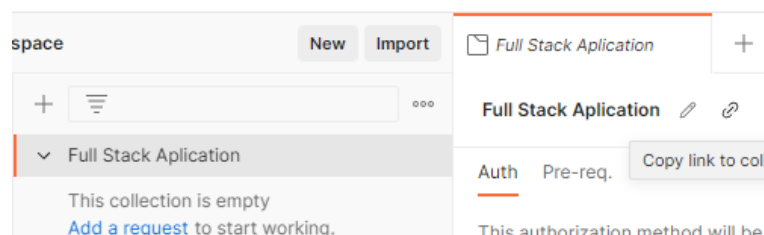
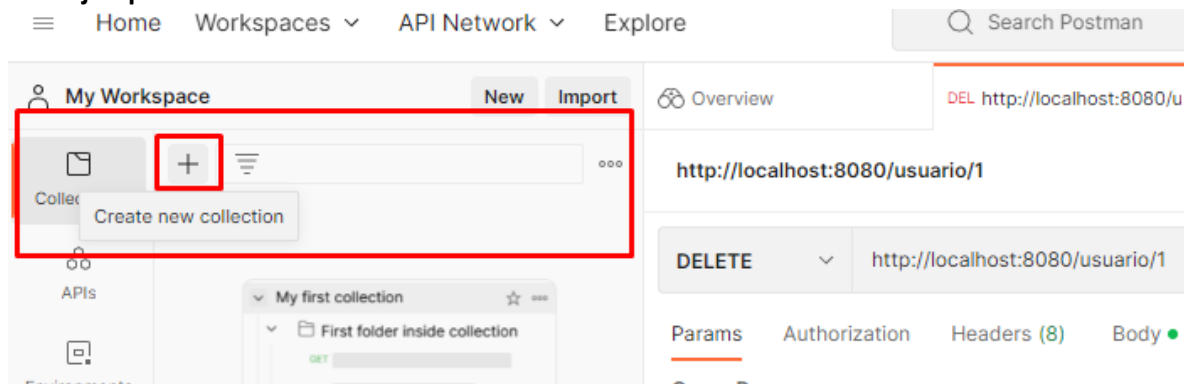
```

Corremos de nuevo y abrimos Postman

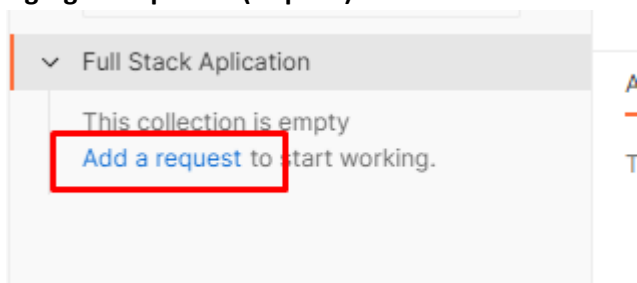


4.1) Revisar funcionamiento con postman

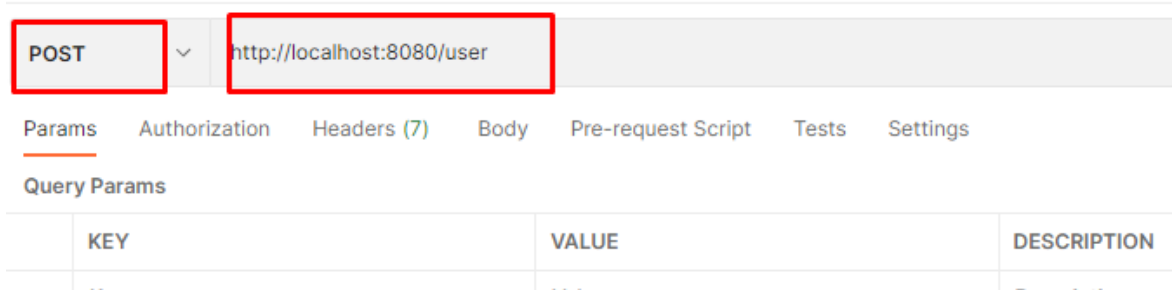
Creamos una nueva colección y le ponemos Full Attack Application para no perder el nombre de este ejemplo



Agregamos pedido (request)



Colocamos tipo POST y el localhost

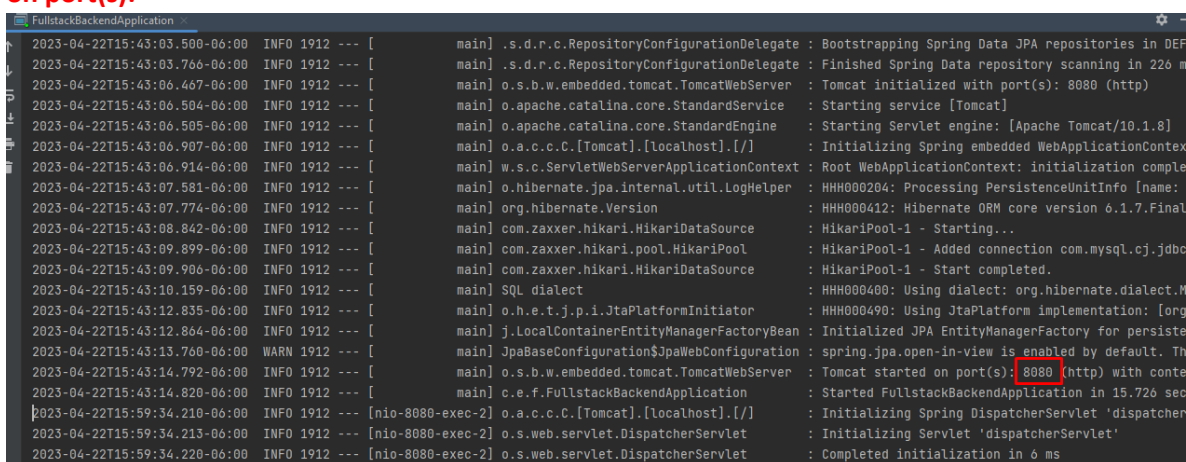


<http://localhost:8080/user>

```
@PostMapping("/user")
User newUser(@RequestBody User newUser){
    return userRepository.save(newUser);
}
```

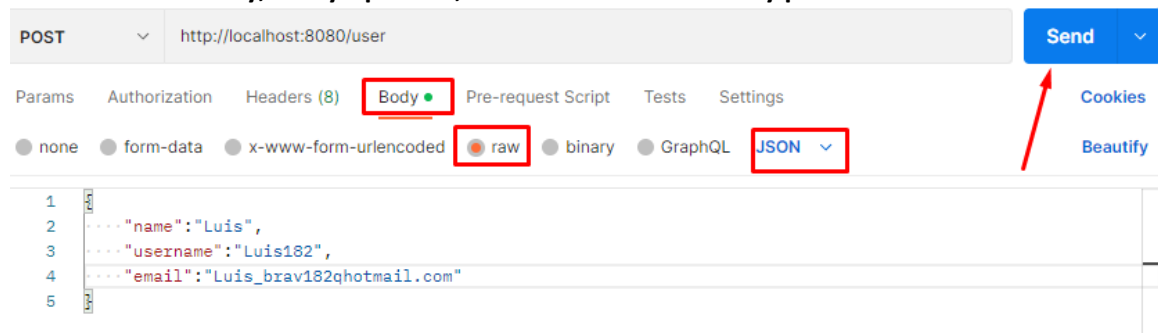
Es el campo que definimos en la anotación Postmappin en la clase UserController

El localhost lo podemos ver a la hora de correr el programa en la pantalla de run en Tomcat started on port(s):



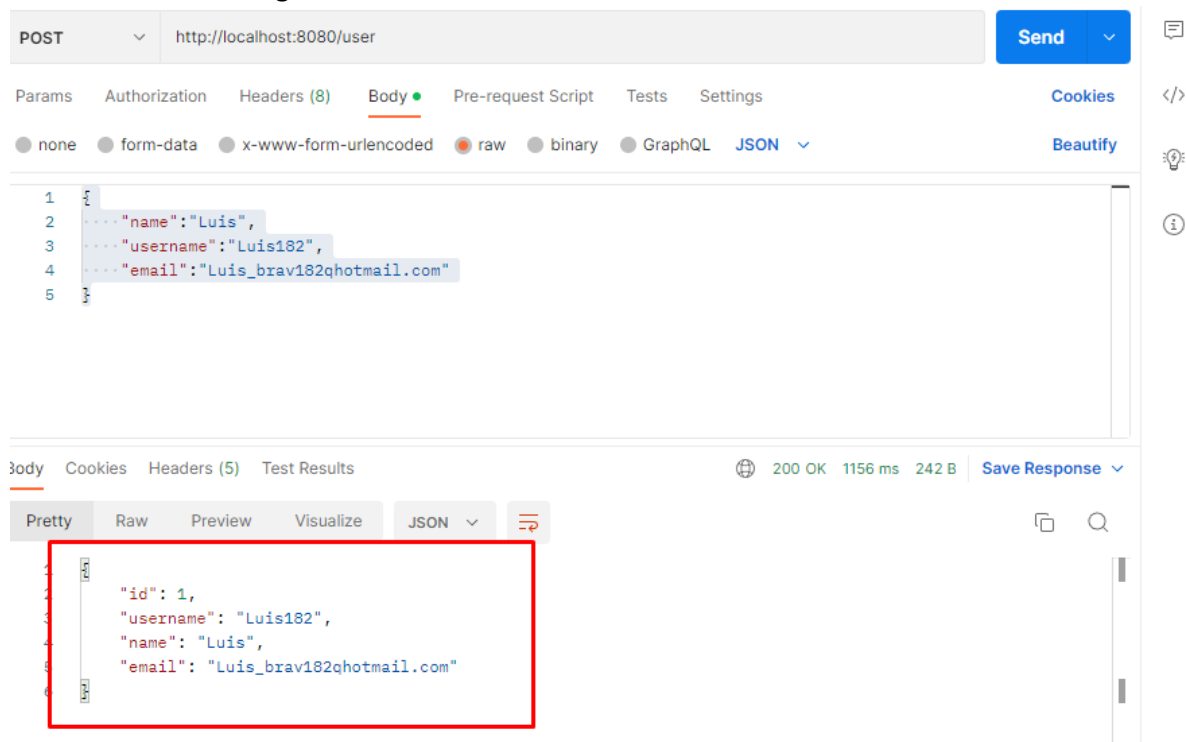
4.2) Damos de alta nuestro primer registro

Seleccionamos Body, raw y tipo Json, damos de alta los datos y presionamos enviar



```
{
  "name": "Luis",
  "username": "Luis182",
  "email": "Luis_brav182@hotmail.com"
}
```

Ahora nos saldrá el registro con su número de Id



```

1
2     "id": 1,
3     "username": "Luis182",
4     "name": "Luis",
5     "email": "Luis_brav182@hotmail.com"
6

```

Revisamos en MySQL



Result Grid				
Filter Rows: <input type="text"/>				
Edit:				
id	email	name	username	
1	Luis_brav182@hotmail.com	Luis	Luis182	
	NULL	NULL	NULL	NULL

Creamos un segundo usuario

```

1
2     .... "name": "Paola",
3     .... "username": "Paola182",
4     .... "email": "Paola@hotmail.com"
5

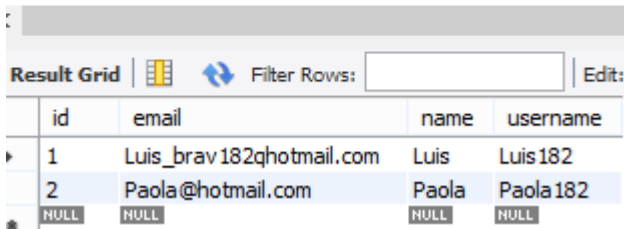
```

body Cookies Headers (5) Test Results



Y del mismo modo revisamos en MySQL

```
1 • select * from user;
```



	id	email	name	username
▶	1	Luis_brav182@hotmail.com	Luis	Luis182
	2	Paola@hotmail.com	Paola	Paola182
*	NULL	NULL	NULL	NULL

5) @GetMapping Recibir datos de la base de datos

Regresamos a la clase control y escribimos

```
@GetMapping("/users")
List<User>
    return
}

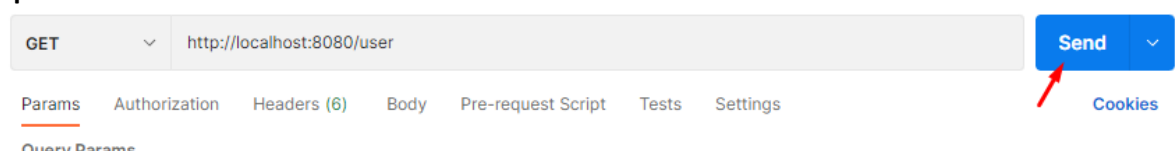
getAllUsers() {
    userRepository.findAll();
}
```

[@GetMapping](#) es una anotación de Spring que se utiliza para mapear solicitudes HTTP GET a métodos de controlador específicos en una aplicación web. Al colocar la anotación @GetMapping encima de un método de controlador en una clase de controlador, Spring mapeará automáticamente las solicitudes GET a ese método. Por ejemplo, si un usuario hace una solicitud GET a la URL <http://ejemplo.com/mi-recurso>, Spring buscará un método de controlador anotado con @GetMapping y mapeado a la ruta /mi-recurso, y ejecutará ese método para manejar la solicitud.

Ahora vamos a postman y creamos una nueva request, ahora con GET y el siguiente local host

<http://localhost:8080/users>

Ahora colocamos user por que en el método @GetMapping se definio como users presionamos send



Y nos deben salir los registros que ya tenemos

```
body  cookies  headers (5)  test results
Pretty  Raw  Preview  Visualize  JSON  v  ⋮

1  [
2    {
3      "id": 1,
4      "username": "Luis182",
5      "name": "Luis",
6      "email": "Luis_brav182@hotmail.com"
7    },
8    {
9      "id": 2,
10     "username": "Paola182",
11     "name": "Paola",
12     "email": "Paola@hotmail.com"
13   }
14 ]
```

6) Creamos app en react

Vamos a Visual Code, para esto ya tenemos que tener node js en nuestro equipo, abrimos la carpeta donde está la carpeta que descomprimos, ahora abrimos la terminal con ctrl+shift+ñ y escribimos npx create-react-app nombre-en-minusculas

```
PS C:\Users\Luis Bravo\Desktop> npx create-react-app fullstack-front

Creating a new React app in C:\Users\Luis Bravo\Desktop\fullstack-front.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[██████████] - reify:eslint-plugin-jsx-a11y: timing reifyNode:node_modules/fork-ts-checker-web
```

En este caso el nombre de la carpeta es fullstack-front por lo que en código ponemos npx create-react-app fullstack-front

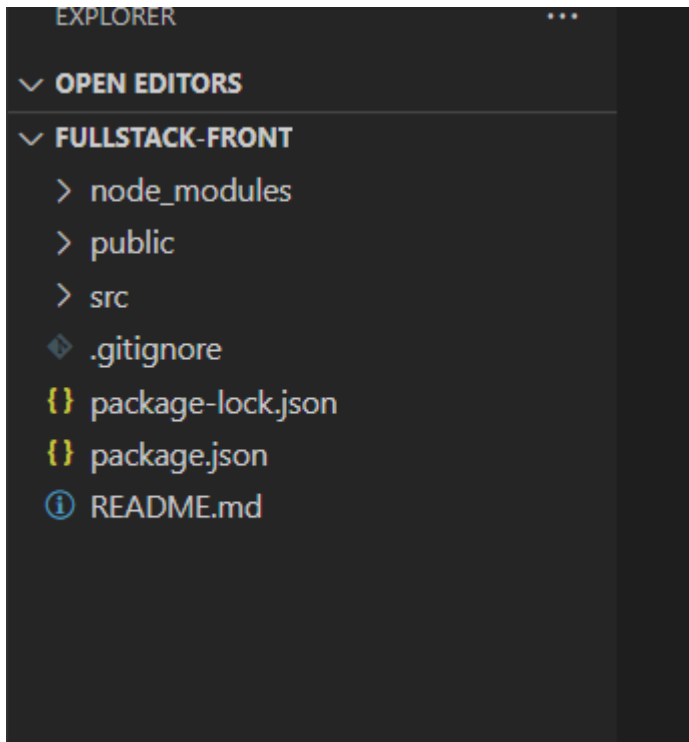
```
npm run eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

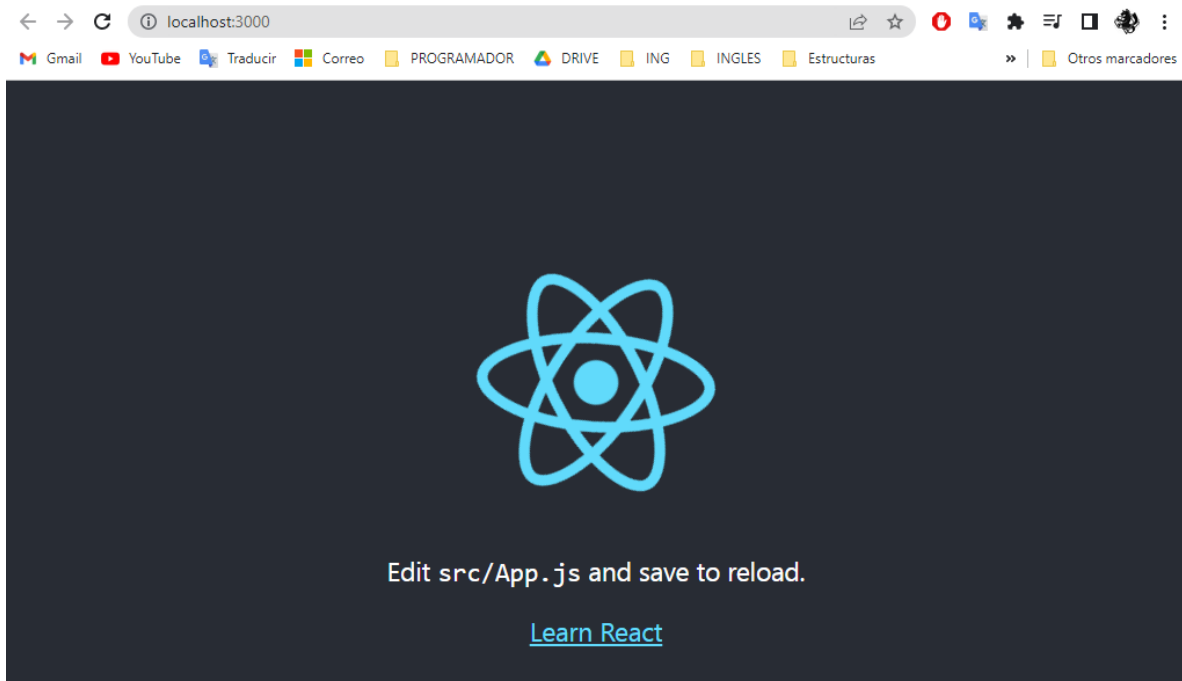
  cd fullstack-front
  npm start

Happy hacking!
PS C:\Users\Luis Bravo\Desktop>
```

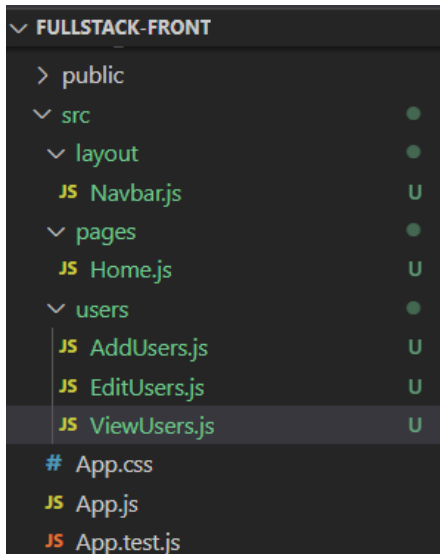
Una vez termine abrimos la carpeta



Abrimos terminal (Ctrl+ñ) y ejecutamos `npm start` para iniciar nuestra app, este abrirá en nuestro navegador

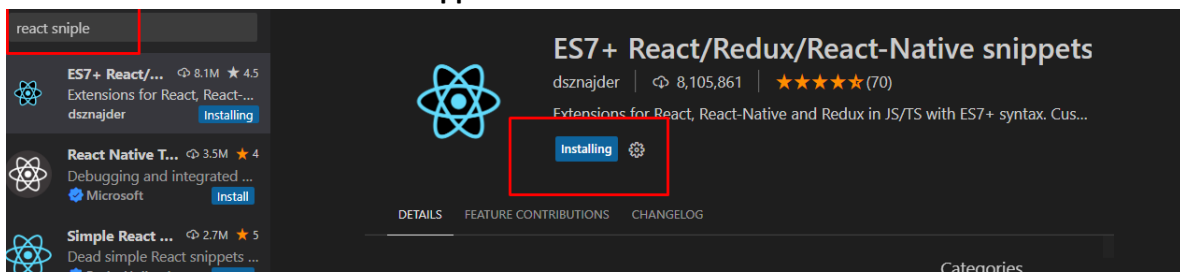


Creamos 3 carpetas layout, pages y users, en layout creamos Navbar.js, en pages creamos Home.js y en users creamos AddUsers.js, EditUsers.js y ViewUsers.js

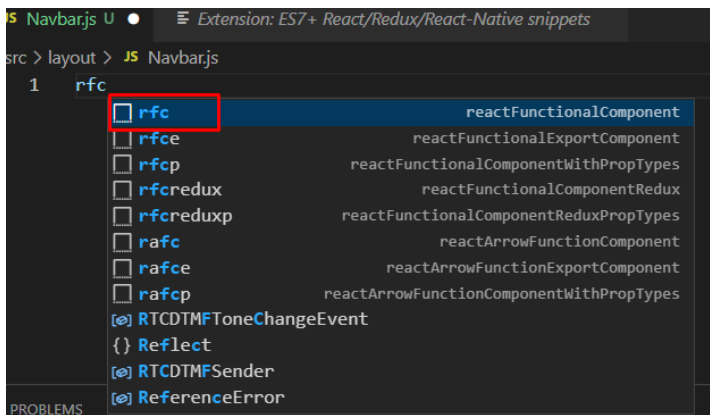


6.1) React snippet (extensión para programar mas rápido)

Instalamos en extensiones react snippet



Ahora al escribir rfc en un archivo JavaScript (en este caso en Navbar) en automático pondrá la estructura de react

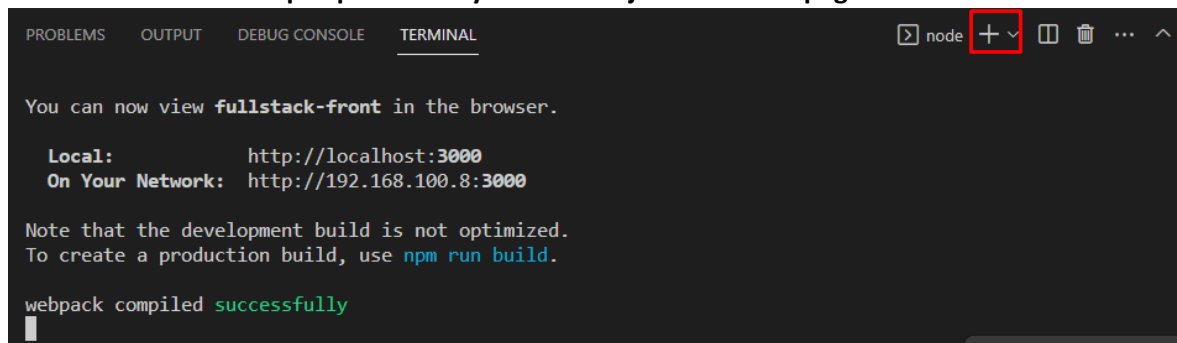


Damos clic y nos deberá salir como se muestra a continuación

```
src > layout > JS Navbar.js
1  import React from 'react'
2  ⚡
3  export default function Navbar() {
4    return (
5      <div>Navbar</div>
6    )
7  }
```

6.2) Creamos Navbar Usando Bootstrap

Abrimos otra terminal porque en una ya está trabajando nuestra página



The screenshot shows a VS Code terminal window with the 'TERMINAL' tab selected. The terminal output indicates that the development server is running successfully on localhost:3000. The message 'webpack compiled successfully' is shown in green. The terminal also displays the local and network URLs for viewing the application in a browser.

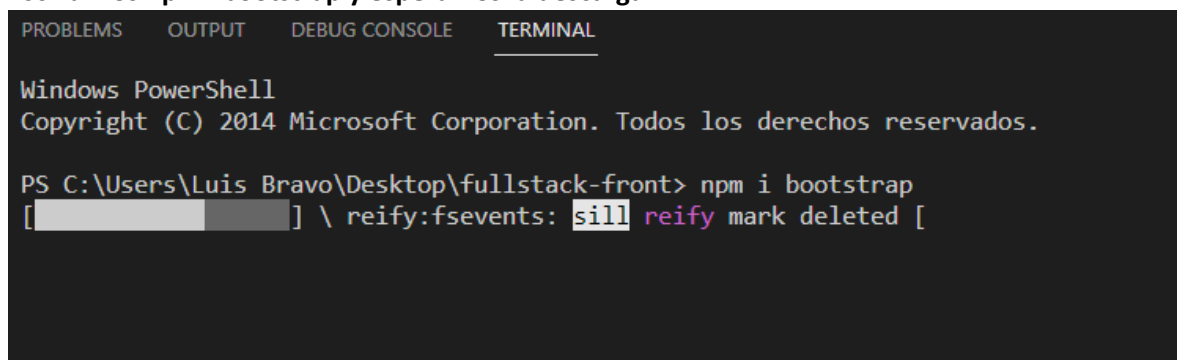
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
You can now view fullstack-front in the browser.

Local:      http://localhost:3000
On Your Network: http://192.168.100.8:3000

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

webpack compiled successfully
```

Escribimos npm i bootstrap y esperamos la descarga

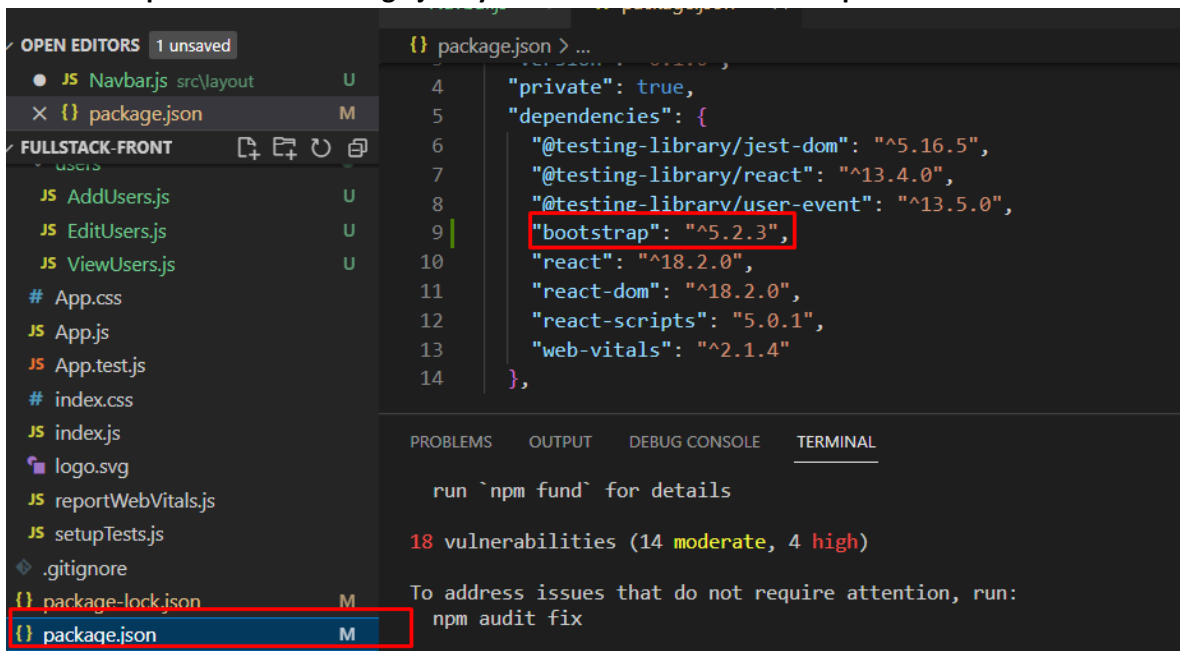


The screenshot shows a Windows PowerShell terminal window with the 'TERMINAL' tab selected. The command 'npm i bootstrap' has been executed, and the output shows the installation progress. The terminal also displays the Windows PowerShell prompt and the copyright notice for Microsoft Corporation.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\Luis Bravo\Desktop\fullstack-front> npm i bootstrap
[ ] \ reify:fsevents: sill reify mark deleted [
```

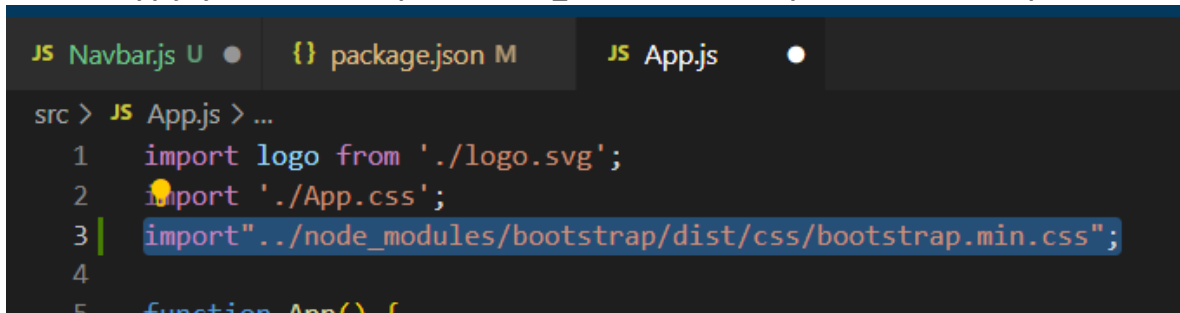
Terminado podemos ir a Package.json y revisar la versión de bootstrap



The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows the project structure with 'package.json' selected. The main editor displays the contents of 'package.json'. The 'dependencies' section is visible, and the version for 'bootstrap' is '^5.2.3', which is highlighted with a red box. Below the editor, the Output panel shows the command 'run `npm fund` for details' and the result '18 vulnerabilities (14 moderate, 4 high)'. The Terminal panel is also visible at the bottom.

```
{
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "bootstrap": "^5.2.3",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  }
}
```

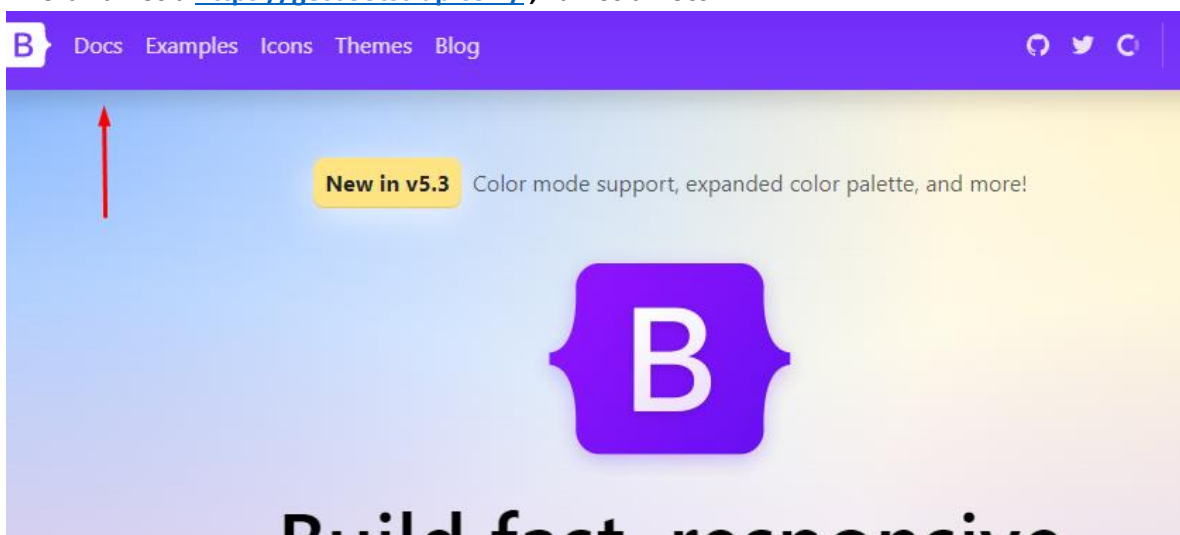
Vamos a App.js y escribimos: `import "../node_modules/bootstrap/dist/css/bootstrap.min.css";`



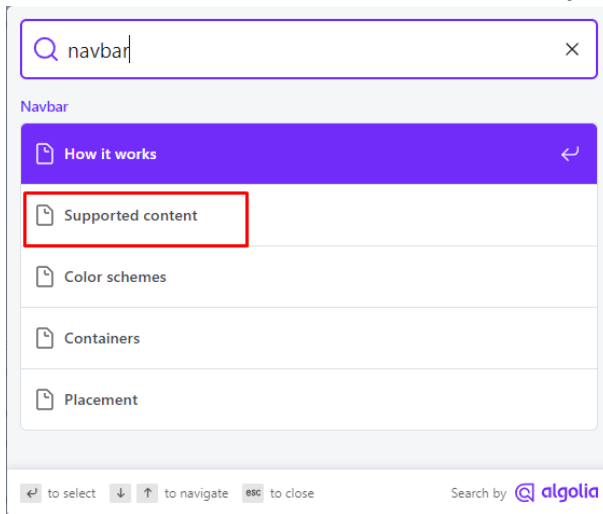
The screenshot shows the VS Code interface with 'App.js' selected in the Explorer sidebar. The main editor displays the contents of 'App.js'. The line `import "../node_modules/bootstrap/dist/css/bootstrap.min.css";` is highlighted with a blue selection bar. The code also includes imports for 'logo' and 'App.css', and a function definition for 'App'.

```
src > JS App.js > ...
1 import logo from './logo.svg';
2 import './App.css';
3 import "../node_modules/bootstrap/dist/css/bootstrap.min.css";
4
5 function App() {
```

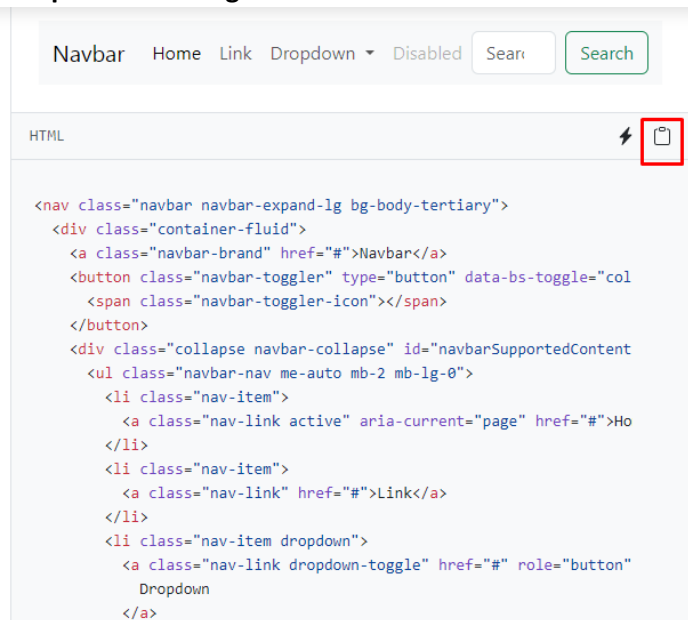
Ahora vamos a <https://getbootstrap.com/>, vamos a Docs



Una vez en docs damos clic en search o ctrl+k y buscamos navbar y damos clic



Y copiamos el código



Ahora vamos a Navbar.js y pegamos entre los <div>, dejaremos el código hasta el primer botón lo demás lo quitamos

```
Navbar.js 2, U  {} package.json M  JS App.js  ●
src > layout > JS Navbar.js > default
1  import React from 'react'
2
3  export default function
4  () {
5    return (
6      <div>
7        <nav class="navbar navbar-expand-lg bg-body-tertiary">
8        <div class="container-fluid">
9          <a class="navbar-brand" href="#">Navbar</a>
10         <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" data-bs-collapse="collapse">
11           <span class="navbar-toggler-icon"></span>
12         </button>
13         <div class="collapse navbar-collapse" id="navbarSupportedContent">
14           <ul class="navbar-nav me-auto mb-2 mb-lg-0">
15             <li class="nav-item">
16               <a class="nav-link active" aria-current="page" href="#">Home</a>
17             </li>
18             <li class="nav-item">
19               <a class="nav-link" href="#">Link</a>
20             </li>
21             <li class="nav-item dropdown">
22               <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" data-bs-collapse="collapse">
23                 Dropdown
24               </a>
```

Nos quedaría de la siguiente forma

```
JS Navbar.js U  {} package.json M  JS App.js  ●
src > layout > JS Navbar.js > default
1  import React from 'react'
2
3  export default function
4  () {
5    return (
6      <div>
7        <nav class="navbar navbar-expand-lg bg-body-tertiary">
8        <div class="container-fluid">
9          <a class="navbar-brand" href="#">Navbar</a>
10         <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" data-bs-collapse="collapse">
11           <span class="navbar-toggler-icon"></span>
12         </button>
13
14       </div>
15     </nav>
16   </div>
17 )
18 }
19
```


Ctrl+shif+f para acomodar el código

```
JS Navbar.js U {} package.json M JS App.js
src > layout > JS Navbar.js > default
1  import React from "react";
2
3  export default function () {
4    return (
5      <div>
6        <nav class="navbar navbar-expand-lg bg-body-tertiary">
7          <div class="container-fluid">
8            <a class="navbar-brand" href="#">
9              Navbar
10            </a>
11            <button
12              class="navbar-toggler"
13              type="button"
14              data-bs-toggle="collapse"
15              data-bs-target="#navbarSupportedContent"
16              aria-controls="navbarSupportedContent"
17              aria-expanded="false"
18              aria-label="Toggle navigation"
19            >
20              <span class="navbar-toggler-icon"></span>
21            </button>
22          </div>
23        </nav>
24      </div>
```

Vamos a cambiar la etiqueta class por className, usamos ctrl+f para hacer cambios, escribimos class y damos clic a la flecha

```
import React from "react";
> class
Aa ab * 5

export default function () {
  return (
    <div>
      <nav class="navbar navbar-expand-lg bg-body-tertiary">
        <div class="container-fluid">
          <a class="navbar-brand" href="#">
            Navbar
          </a>
          <button
            class="navbar-toggler"
            type="button"
            data-bs-toggle="collapse"
```

Escribimos className y damos clic en replace all

```
> layout > JS Navbar.js > default
1 import React from "react";
2
3 export default function () {
4   return (
5     <div>
6       <nav class="navbar navbar-expand-lg bg-body-tertiary">
7         <div class="container-fluid">
8           <a class="navbar-brand" href="#">
9             Navbar
10          </a>
11          <button
12            class="navbar-toggler"
13            type="button"
14            data-bs-toggle="collapse"
```

Y listo

```
<div>
  <nav className="navbar navbar-expand-lg bg-body-tertiary">
    <div className="container-fluid">
      <a className="navbar-brand" href="#">
        Navbar
      </a>
      <button
        className="navbar-toggler"
        type="button"
        data-bs-toggle="collapse"
        data-bs-target="#navbarSupportedContent"
        aria-controls="navbarSupportedContent"
        aria-expanded="false"
        aria-label="Toggle navigation"
      >
        <span className="navbar-toggler-icon"></span>
      </button>
    </div>
  </nav>
</div>
```

6.3) Importamos Navbar a App.js

Borramos el siguiente código y todo lo que no necesitemos

```
Navbar.js U • {} package.json M JS App.js •
> JS App.js > App
1 import logo from './logo.svg';
2 import './App.css';
3 import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <img src={logo} className="App-logo" alt="logo" />
10        <p>
11          Edit <code>src/App.js</code> and save to reload.
12        </p>
13        <a
14          className="App-link"
15          href="https://reactjs.org"
16          target="_blank"
17          rel="noopener noreferrer"
18        >
19          Learn React
20        </a>
21      </header>
22    </div>
23  );
24}
```

Escribimos el nombre de nuestro archivo y en automático nos sale el nombre

```
import './App.css';
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';

function App() {
  return (
    <div className="App">
      <Navbar
      </div>
    </div>
  );
}

export default App;
```

Al dar clic en automático se importa ahora solo cerramos la etiqueta

```

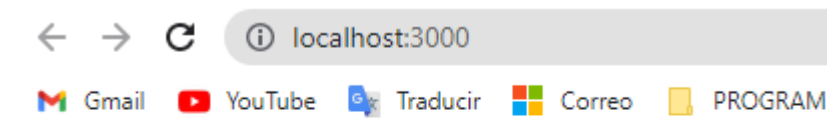
import './App.css';
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
import Navbar from './layout/Navbar';

function App() {
  return (
    <div className="App">
      <Navbar/>
    </div>
  );
}

export default App;

```

Y en la página ya nos debe de salir (si no se actualiza la página hay que guardar o ctrl+s en cada archivo trabajado)



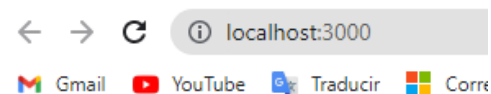
Navbar

Vamos a Navbar y cambiamos el titulo

```

export default function () {
  return (
    <div>
      <nav className="navbar navbar-expand-lg bg-body-tertiary">
        <div className="container-fluid">
          <a className="navbar-brand" href="#">
            FullStack App
          </a>
          <button
            className="navbar-toggler"
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent"
            aria-expanded="false"
            aria-label="Toggle navigation"
          >
            <span className="navbar-toggler-icon"></span>
          </button>
        </div>
      </nav>
    </div>
  );
}

```

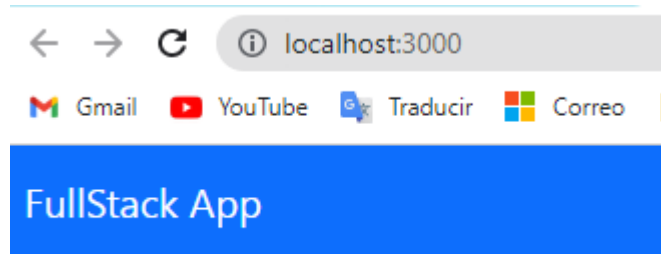


FullStack App

6.4) Editor Navbar

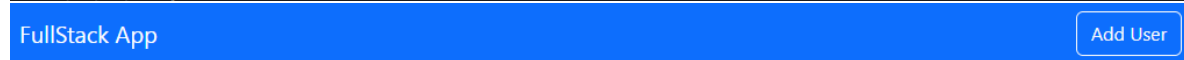
Escribimos: navbar-dark bg-primary"

```
export default function () {  
  return (  
    <div>  
      <nav className="navbar navbar-expand-lg navbar-dark bg-primary">  
        <div className="container-fluid">  
          <a className="navbar-brand" href="#">  
            FullStack App  
          </a>  
          <button  
            className="navbar-toggler"  
            type="button"
```



Agregamos un botón: <button className="btn btn-outline-light">Add User</button>

```
      data-bs-toggle="collapse"  
      data-bs-target="#navbarSupportedContent"  
      aria-controls="navbarSupportedContent"  
      aria-expanded="false"  
      aria-label="Toggle navigation"  
    >  
      <span className="navbar-toggler-icon"></span>  
    </button>  
    <button className="btn btn-outline-light">Add User</button>  
  </div>  
</nav>
```



6.5) Creamos Home con Bootstrap

Escribimos rfc y cargamos el código

```
1 import React from 'react'
2
3 export default function Home() {
4   return (
5     <div>Home</div>
6   )
7 }
8
```

Dejamos el código como se muestra a continuación

```
export default function Home() {
  return (
    <div className='container'>
      <div className='py-4'>
        </div>
      </div>
    </div>
  )
}
```

Vamos a bootstrap y buscamos table y copiamos el código

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
```



Y pegamos en el div con className py-4, cambiamos class por ClassName e importamos a App.js

```
1 import './App.css';
2 import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
3 import Navbar from './layout/Navbar';
4 import Home from './pages/Home';
5
6 function App() {
7   return (
8     <div className="App">
9       <Navbar/>
10      <Home/>
11    </div>
12  );
13 }
14
15 export default App;
```

Así se ve ahora nuestra aplicación

FullStack App Add User

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

Vamos a darle bordes a la tabla vamos a Home.js y escribimos border apra borde y shadow para sombra

```
2
3 export default function Home() {
4   return (
5     <div className='container'>
6       <div className='py-4'>
7         <table class="table border shadow">
8       <thead>
9         <tr>
10          <th scope="col">#</th>
11          <th scope="col">First</th>
```

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

7) AXIOS GET: Mostrar información

Vamos a npmjs.com y buscamos axios

The screenshot shows the npmjs.com website. At the top, there's a navigation bar with links like 'Traducir', 'Correo', 'PROGRAMADOR', 'DRIVE', 'ING', 'INGLES', and 'Estructuras'. Below this, the search bar contains 'axios' and a 'Search' button. The search results show 'aws-sdk' as the top result. Below the search bar, there's a section for 'Seleccionamos' (We select) with a red box around the 'axios' result, which is labeled 'exact match'. The 'axios' result shows it's a 'Promise based HTTP client for the browser and node.js' by 'jasonsaayman', published 1.3.6, 5 days ago. Below this, there's a section for 'superagent'. At the bottom, there's a section for 'Copiamos y pegamos en un terminal nuevo' (We copy and paste into a new terminal), which shows the command 'npm i axios' in a red box.

AXIOS

Promise based HTTP client for the browser and node.js

[Website](#) • [Documentation](#)

v1.3.6 cdnjs v1.3.6 CI passing Gitpod Ready-to-Code coverage 94%

Install

```
> npm i axios
```

Repository

github.com/axios/axios


```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\Luis Bravo\Desktop\fullstack-front> npm i axios
```

```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\Luis Bravo\Desktop\fullstack-front> npm i axios
[██████████] | idealTree: timing idealTree Completed in 8345ms
```

Una vez terminado podemos ir a package.json y revisar que axios está instalado, vamos a Home.js e importamos axios

```
1 import React from 'react'
2 import axios from 'axios'
3
4 export default function Home() {
5   return (
6     <div className='container'>
7       <div className='py-4'>
```

7.1) ¿Qué es axios?

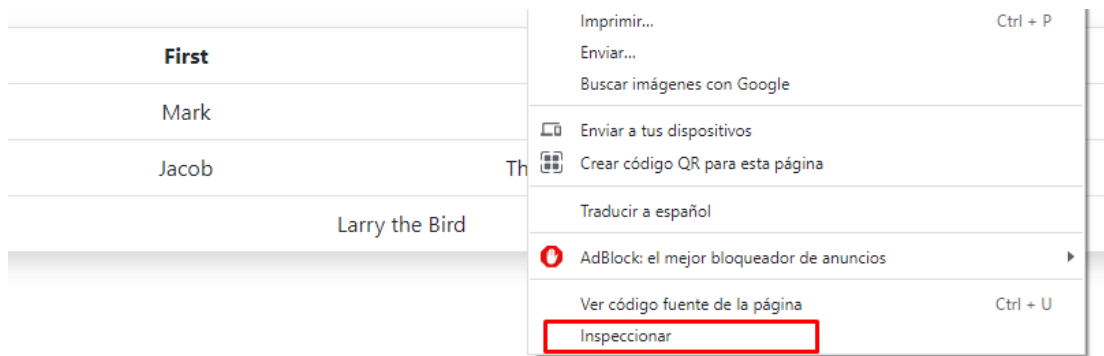
Axios es una librería de JavaScript utilizada para realizar solicitudes HTTP en el navegador o en Node.js. Se utiliza comúnmente en aplicaciones web para interactuar con APIs o para realizar operaciones CRUD (crear, leer, actualizar, eliminar) en una base de datos.

7.2) Creamos hooks

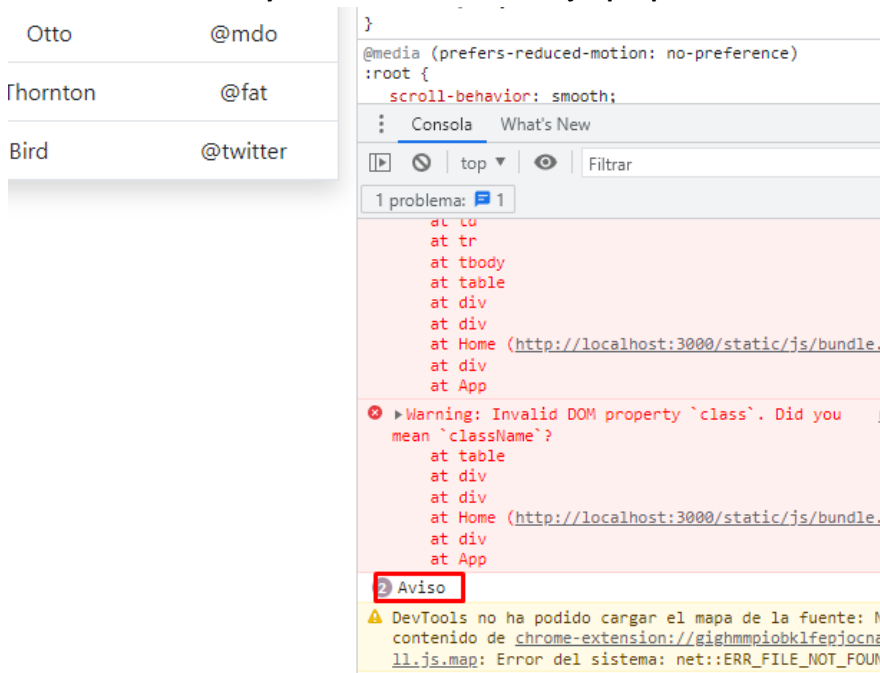
Ahora crearemos un hook para manejar los estados y uno de aviso

```
1 import React, { useEffect, useState } from 'react'
2 import axios from 'axios'
3
4 export default function Home() {
5
6   const [users, setUsers] = useState([]);
7
8   useEffect(() => {
9     console.log("Aviso")
10   });
11 }
```

En automático nos importa React, ahora vamos a nuestra página y podemos inspeccionar



No abrirá la ventana y nos debe salir el mensaje que pusimos



7.3) ¿Qué es un hook?

En el contexto de React, un hook es una función especial que permite a los componentes de React acceder a características de React, como el estado de un componente o su ciclo de vida, sin tener que escribir una clase. Los hooks fueron introducidos en React 16.8 y son funciones que se utilizan en los componentes de función para agregar ciertas funcionalidades.

Los hooks más comunes en React incluyen useState para manejar el estado de un componente, useEffect para manejar los efectos secundarios en un componente, useContext para manejar el contexto de la aplicación y useRef para referenciar elementos del DOM. Los hooks permiten un código más limpio, modular y fácil de entender en comparación con las clases y sus métodos de ciclo de vida en versiones anteriores de React.

[useState](#) es un hook de React que se utiliza para manejar estados en los componentes funcionales. Permite declarar variables de estado y actualizarlas. El hook toma como parámetro el valor inicial del estado y devuelve un array con dos elementos: el estado actual y una función para actualizar el estado.

Por ejemplo, si quisieras tener una variable de estado llamada users y actualizarla cada vez que un botón es presionado, podrías usar useUsers.

[useEffect](#) es un hook de React que permite agregar efectos secundarios a un componente funcional. Esto significa que permite ejecutar código en respuesta a ciertas acciones, como cambios en el estado de un componente, cambios en las props, o la carga inicial del componente.

El hook se utiliza para realizar tareas como la actualización del DOM, la carga de datos desde un servidor, la subscripción a eventos, y muchas otras cosas. useEffect toma dos argumentos: una función que contiene el código que se desea ejecutar, y una lista de dependencias que especifican cuando se debe volver a ejecutar el código.

7.4) ¿Qué es un prop?

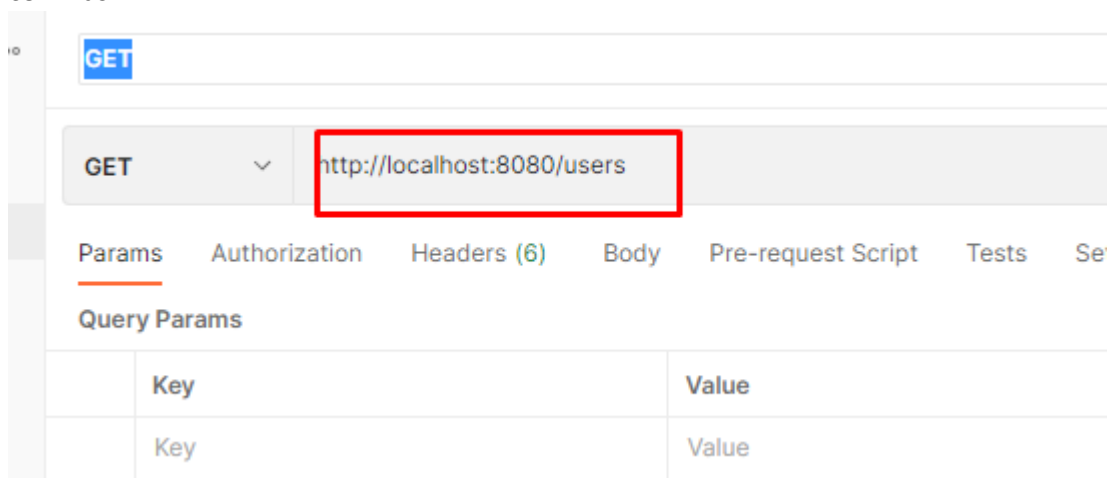
En React, los "props" (abreviatura de "properties", propiedades en español) son un mecanismo para pasar datos de un componente a otro. Los componentes de React pueden aceptar props como argumentos y usarlos para renderizar elementos diferentes o aplicar lógica adicional. Por ejemplo, si un componente necesita mostrar diferentes mensajes de bienvenida según el usuario que esté actualmente conectado, podría recibir el nombre de usuario como un prop y utilizarlo para personalizar el mensaje. Los props son pasados de un componente padre a un componente hijo, lo que permite la creación de componentes más flexibles y reutilizables.

7.5) Función cargar usuarios

En Home.js creamos la función flecha load users loadUsers

```
const loadUsers=()=>{  
  const result=axios.get("");  
};
```

Vamos a postman y copiamos el url que utilizamos en el método get y lo pegamos entre las comillas



```
const loadUsers=()=>{
  const result=axios.get("http://localhost:8080/users");
};
```

Ahora agregaremos las palabras `async` y `await`; y un mensaje en la consola y llamaremos nuestra función flecha en `useEffect`, y agregaremos una herencia vacía colocando `[]` después de la llave

```
useEffect(()=>{
  loadUsers();
},[]);

const loadUsers=async()=>{
  const result=await axios.get("http://localhost:8080/users");
};
```

[async y await son características de JavaScript que se utilizan para manejar promesas.](#)

[En el contexto de axios, async y await se pueden utilizar para realizar solicitudes HTTP asíncronas de manera más fácil y legible.](#)

[Por ejemplo, en lugar de utilizar la sintaxis de promesa convencional con .then\(\) y .catch\(\), podemos utilizar async y await para escribir código más limpio y legible.](#)

7.6) @CrossOrigin

Volvemos a IntelliJ IDEA y vamos al archivo `UserController` y escribimos

`@CrossOrigin`

```
@RestController
@CrossOrigin("")
public class UserController {

    2 usages
    @Autowired
    private UserRepository userRepository;
```

Ahora entre comillas colocamos el url de nuestra aplicación web

```
@RestController
@CrossOrigin("http://localhost:3000/")
public class UserController {
```

La anotación **@CrossOrigin** en Java se utiliza en una aplicación web para permitir solicitudes CORS (Cross-Origin Resource Sharing) en un controlador específico o en toda la aplicación. CORS es un mecanismo de seguridad que impide que los scripts de un sitio web accedan a los recursos de otro dominio. La anotación **@CrossOrigin** permite que los navegadores web superen esta política de seguridad y permitan que un recurso se solicite desde un dominio diferente al dominio del recurso.

La anotación **@CrossOrigin** se puede aplicar a un método de controlador específico o a una clase de controlador completa para permitir solicitudes CORS en todo el controlador. También se pueden especificar ciertas propiedades, como los orígenes permitidos, los métodos HTTP permitidos y los encabezados permitidos.

7.7) Result.data

Vovlemos a Visual code y agregamos el siguiente comando en la función loadUSers: console.log(result.data);

```
const loadUsers=async()=>{  
  const result=await axios.get("http://localhost:8080/users");  
  console.log(result.data);  
};
```

En el contexto de una llamada a una API usando Axios, **result.data** se utiliza para acceder a los datos que se obtienen de la respuesta de la llamada. Axios devuelve un objeto response que contiene información sobre la respuesta de la API, como el estado de la respuesta, las cabeceras y los datos devueltos. Los datos devueltos se almacenan en la propiedad data del objeto response. Para acceder a los datos devueltos, se utiliza **result.data**. Por ejemplo, si se hace una llamada a una API y se espera recibir un objeto JSON, se puede acceder a ese objeto utilizando **result.data**.

7.8) ¿Por qué aun no sale la información de la base de datos?

Simple, a pesar de que ya hemos seguido los pasos no hemos modificado Home.js, no hemos llamado la función ni asignado las props

#	First	Last	Handle
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry the Bird		@twitter

```
JS Home.js U X JS Navbar.js U JS App.js M {} package.json M
src > pages > JS Home.js > Home
27     </tr>
28   </thead>
29   <tbody>
30     <tr>
31       <th scope="row">1</th>
32       <td>Mark</td>
33       <td>Otto</td>
34       <td>@mdo</td>
35     </tr>
36     <tr>
37       <th scope="row">2</th>
38       <td>Jacob</td>
39       <td>Thornton</td>
40       <td>@fat</td>
41     </tr>
42     <tr>
43       <th scope="row">3</th>
44       <td colspan="2">Larry the Bird</td>
45       <td>@twitter</td>
46     </tr>
```

7.9) Editando Home.js

Borramos primero estos 2 usuarios y usaremos el primero para mantener el formato y lo encerraremos entre llaves

```
30     <tr>
31       <th scope="row">1</th>
32       <td>Mark</td>
33       <td>Otto</td>
34       <td>@mdo</td>
35     </tr>
36     <tr>
37       <th scope="row">2</th>
38       <td>Jacob</td>
39       <td>Thornton</td>
40       <td>@fat</td>
41     </tr>
42     <tr>
43       <th scope="row">3</th>
44       <td colspan="2">Larry the Bird</td>
45       <td>@twitter</td>
46     </tr>
```

Ahora llamamos al metodo .map, para buscar, antes de eso colocamos users que es el parámetro inicial del hook useState.

```
    </thead>
    <tbody>
      {
        users.map((user, index)=>(
          <tr>
            <th scope="row">1</th>
            <td>Mark</td>
            <td>Otto</td>
            <td>@mdo</td>
          </tr>
        ))
      }
    </tbody>
  </table>
```

La estructura `users.map((user, index)=>...)` es una función de orden superior en JavaScript que se utiliza para aplicar una función a cada elemento de un array `users` y crear un nuevo array con los resultados.

En este caso, `users` es un array de usuarios y la función que se está aplicando en cada elemento del array es una función anónima que toma dos argumentos: `user` (tabla de la base de datos) e `index` (id). Dentro de esta función se realiza alguna operación que utiliza los datos del usuario y el índice para generar algún resultado. El resultado de esta función se almacena en un nuevo array que se crea a partir de la ejecución del método `map()`.

Ahora vamos a vincular los campos a mostrar primero indicamos con el atributo key que deberá tomar los valores de manera única de la lista

```
{
  users.map((user, index)=>(
    <tr>
      <th scope="row" key={index}>{index+1}</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>@mdo</td>
    </tr>
  ))
}
```

En una estructura JSX, **key** es un atributo especial que se usa para identificar de manera única cada elemento en una lista. **index** es el índice del elemento actual en la lista.

En el código que muestras, **key={index}** establece la propiedad **key** de cada elemento de la lista con el valor del índice actual. **>{index+1}** muestra el número de orden correspondiente al elemento en la lista.

Por ejemplo, si **users** es un arreglo de tres elementos, el resultado de este código sería una lista con tres elementos, con **key** establecido como 0, 1 y 2 respectivamente, y el número de orden de cada elemento sería 1, 2 y 3

Ahora vinculamos cada campo faltante con el valor de la tabla de la app con el de la base de datos

```
{
  users.map((user, index)=>(
    <tr>
      <th scope="row" key={index}>{index+1}</th>
      <td>{user.name}</td>
      <td>{user.username}</td>
      <td>{user.email}</td>
    </tr>
  ))
}
```

En la línea **<td>{user.name}</td>**, **{user.name}** es una expresión de JavaScript que muestra el valor de la propiedad **"name"** del objeto **"user"**. Esta línea de código se utiliza para renderizar el nombre del usuario en una tabla. El objeto **"user"** es un elemento del array **"users"**, que es iterado por el método **map()** para generar la tabla. Por lo tanto, esta línea muestra el nombre del usuario correspondiente a la fila actual de la tabla que se está generando.

Guardamos pero en la app sigue sin mostrarnos nada

FullStack App				Add User
#	First	Last	Handle	

Vamos a Home.js, y podemos notar que no tenemos nada en la tabla porque result que es la función que usamos para conectar con la base de datos está siendo enviada a un mensaje de consola

```
const [users, setUsers] = useState([]);

useEffect(() => {
  loadUsers();
}, []);

const loadUsers = async () => {
  const result = await axios.get("http://localhost:8080/users");
  console.log(result.data);
};
```

Modificamos, en lugar de console.log, ponemos setUsers que es el array que regresa el hook useState

```
const [users, setUsers] = useState([]);

useEffect(() => {
  loadUsers();
}, []);

const loadUsers = async () => {
  const result = await axios.get("http://localhost:8080/users");
  setUsers(result.data);
};
```

Si vamos a la app ahora deben salir los usuarios

#	First	Last	Handle
1	Luis	Luis182	Luis_brav182@hotmail.com
2	Paola	Paola182	Paola@hotmail.com

8) Enrutamiento REAC-ROUTER-DOM

El enrutamiento en el desarrollo web es el proceso de determinar cómo responder a una solicitud de cliente a un recurso específico. En términos generales, el enrutamiento implica asociar una URL con una acción o contenido específico en una aplicación web. En otras palabras, cuando un cliente (por ejemplo, un navegador web) solicita una URL, el servidor web debe determinar qué acción o contenido responder a la solicitud y cómo hacerlo.

En el contexto de un framework como React, el enrutamiento se refiere a la forma en que la aplicación maneja las solicitudes de URL y decide qué componente de React debe renderizarse en respuesta a esa solicitud. En React, esto se puede lograr utilizando una biblioteca de enrutamiento como React Router, que permite a los desarrolladores definir rutas y componentes correspondientes.

8.1) Creando botones

Vamos a Home.js y editaremos los espacios de nuestra tabla.

```
<th scope="col">#</th>
<th scope="col">First</th>
<th scope="col">Last</th>
<th scope="col">Handle</th>
</tr>
```

Cambiaremos First por Name, Last por User name, Handle por Email y agregaremos otro espacio donde pondremos action

```
<thead>
<tr>
  <th scope="col">#</th>
  <th scope="col">Name</th>
  <th scope="col">User name</th>
  <th scope="col">Email</th>
  <th scope="col">Action</th>
</tr>
```

#	Name	User name	Email	Action
1	Luis	Luis182	Luis_brav182qhotmail.com	
2	Paola	Paola182	Paola@hotmail.com	

Ahora agregaremos 3 botones para que se muestren a un lado del registro

```
<tr>
  <th scope="row" key={index}>{index+1}</th>
  <td>{user.name}</td>
  <td>{user.username}</td>
  <td>{user.email}</td>
  <td>
    <button className='btn btn-primary mx-2'>View</button>
    <button className='btn btn-outline-primary mx-2'>Edit</button>
    <button className='btn btn-danger mx-2'>Delete</button>
  </td>
</tr>
```

8.2) AddUser.js

Vamos a AddUser.js en visual code e iniciamos con rfc nuestro archivo

```
import React from 'react'

export default function AddUsers() {
  return (
    <div>AddUsers</div>
  )
}
```

Vamos a <https://www.npmjs.com/> y buscamos react router dom, copiamos el código e iniciamos en una nueva terminal

React Router DOM

The `react-router-dom` package contains bindings for using **React Router** in web applications. Please see **the Getting Started guide** for more information on how to get started with React Router.

Install

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

Repository

github.com/remix-run/react-ro...

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\Luis Bravo\Desktop\fullstack-front> npm i react-router-dom
[redacted] | idealTree: sill logfile start cleaning logs, removing 2 files
```

Y como siempre podemos revisar en package.json que se haya instalado y que versión tenemos

```
{ } package.json M X  JS AddUsers.js U ●

{ } package.json > { } dependencies
6   "@testing-library/jest-dom": "^5.16.5",
7   "@testing-library/react": "^13.4.0",
8   "@testing-library/user-event": "^13.5.0",
9   "axios": "^1.3.6",
10  "bootstrap": "^5.2.3",
11  "react": "^18.2.0",
12  "react-dom": "^18.2.0",
13  "react-router-dom": "^6.10.0",
14  "react-scripts": "5.0.1",
15  "web-vitals": "^2.1.4"
```

Vamos a App.js y escribimos la siguiente línea de código: `import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';`

```
package.json M  JS App.js M ●  JS AddUsers.js U ●

c > JS App.js > ...
1   import './App.css';
2   import './node_modules/bootstrap/dist/css/bootstrap.min.css';
3   import Navbar from './layout/Navbar';
4   import Home from './pages/Home';
5   import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
```

La línea de código importa los componentes necesarios de la librería "react-router-dom" para habilitar el enrutamiento en una aplicación de React.

"BrowserRouter": Este componente envuelve toda la aplicación y permite utilizar la funcionalidad de enrutamiento.

"Routes": Este componente sirve para definir las diferentes rutas que la aplicación tendrá y qué componente se renderizará en cada una de ellas.

"Route": Este componente se utiliza para definir una ruta específica y qué componente se renderizará en ella.

En resumen, con esta línea de código se importan los componentes necesarios para habilitar el enrutamiento en una aplicación de React y se definen las rutas y componentes correspondientes.

Ahora agregaremos Router en el cuerpo de nuestra app

```
function App() {  
  return (  
    <div className="App">  
      <Router>  
      </Router>  
      <Navbar/>  
      <Home/>  
    </div>  
  );  
}  
  
export default App;
```

Colcoamos Navbar y Home dentro

```
function App() {  
  return (  
    <div className="App">  
      <Router>  
        <Navbar/>  
        <Home/>  
      </Router>  
    </div>  
  );  
}
```

Del mismo modo agregamos el componente Routes y agregamos el elemento Route con el siguiente código: <Route exact path="/" element={<Home/>}/>

```
function App() {  
  return (  
    <div className="App">  
      <Router>  
        <Navbar/>  
        <Routes>  
          <Route exact path="/" element={<Home/>}/>  
        </Routes>  
      </Router>  
    </div>  
  );  
}
```

La línea de código que agregamos es una definición de ruta específica que se utiliza en conjunto con los componentes importados previamente.

"exact": Este atributo indica que la ruta debe coincidir exactamente con el valor especificado en "path". Es decir, si se especifica la ruta "/" como en este caso, solo se coincidirá si la URL completa de la aplicación es exactamente "/".

"path": Este atributo define la ruta que se va a utilizar. En este caso, la ruta es "/" que se refiere a la página de inicio de la aplicación.

"element": Este atributo especifica el componente que se renderizará cuando la ruta coincida con la URL actual de la aplicación. En este caso, se está utilizando el componente "Home" que se importó previamente.

En resumen, esta línea de código define una ruta específica para la página de inicio de la aplicación ("/") y especifica que cuando la URL coincida exactamente con esta ruta, se renderizará el componente "Home"

Ahora nuestra página se debe ver así



#	Name	User name	Email	Action		
1	Luis	Luis182	Luis_brav182qhotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
2	Paola	Paola182	Paola@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>

#	Name	User name	Email	Action		
1	Luis	Luis182	Luis_brav182qhotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
2	Paola	Paola182	Paola@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>

Esto porque tenemos 2 veces el elemento Home, así que borramos el que esta solo

```
function App() {  
  return (  
    <div className="App">  
      <Router>  
        <Navbar/>  
  
        <Routes>  
          <Route exact path="/" element={<Home/>}/>  
        </Routes>  
        <Home/>  
      </Router>  
    </div>  
  );  
}
```

Y Ahora nos sale un solo elemento Home



#	Name	User name	Email	Action		
1	Luis	Luis182	Luis_brav182qhotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
2	Paola	Paola182	Paola@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>

Ahora en el elementos Routes agregamos la siguiente línea de código: <Route exact path="/adduser" element={<AddUsers/>}/>

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import AddUsers from './users/AddUsers';

function App() {
  return (
    <div className="App">
      <Router>
        <Navbar/>

        <Routes>
          <Route exact path="/" element={<Home/>}/>
          <Route exact path="/adduser" element={<AddUsers/>}/>
        </Routes>
      </Router>
    </div>
  );
}
```

En automático se importara: `import AddUsers from './users/AddUsers';`

Ahora vamos a `Navbar.js` y cambiamos de elemento el botón Add User, ahora será tipo Link

```
import React from "react";
import { Link } from "react-router-dom";

export default function () {
  return (
    <div>
      <nav className="navbar navbar-expand-lg navbar-dark bg-primary">
        <div className="container-fluid">
          <a className="navbar-brand" href="#">
            FullStack App
          </a>
          <button
            className="navbar-toggler"
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent"
            aria-expanded="false"
            aria-label="Toggle navigation"
          >
            <span className="navbar-toggler-icon"></span>
          </button>
          <Link className="btn btn-outline-light">Add User</Link>
        </div>
      </nav>
    </div>
  );
}
```


E importamos: `import { Link } from "react-router-dom"`; ahora en Link agregamos la ruta que mandara: `to="/adduser"`

```
    </button>
    <Link className="btn btn-outline-light" to="/adduser">Add User</Link>
  </div>
</nav>
</div>
```

La misma que tenemos en App.js

```
<Route exact path="/" element={<Home/>}/>
<Route exact path="/adduser" element={<AddUsers/>}/>
</Routes>
```

Si vamos a la app y presionamos el botón (Link) Add User debe salir lo siguiente, es decir hace el cambio entre app

FullStack App Add User

AddUsers

9) Creamos registro de usuario con React

Regresamos a AddUser.js y hacemos las siguientes modificaciones

```
src > users > JS AddUsers.js > AddUsers
1  import React from 'react'
2
3  export default function AddUsers() {
4    return (
5      <div className='container'>
6        <div className='row'>
7          <div className='col-md-6 offset-md-3 border rounded p-4 mt-2 shadow'>
8            <h2 className='text-center m-4'>Registrer user</h2>
9            <div className='mb-3'>
10             <label htmlFor='Name' className='form-label'>
11               Name
12             </label>
13             <input
14               type={"text"}
15               className='form-control'
16               placeholder='Enter your name'
17               name='name'
18             />
19           </div>
20         </div>
21       </div>
22     </div>
23   </div>
```

Nuestra app se vera de la siguiente manera:

FullStack App Add User

Registrar user

Name

Hasta ahora tenemos el campo para recibir el nombre del usuario, podemos copiar el siguiente div 2 veces para crear los espacios restantes

```
src > users > JS AddUsers.js > AddUsers
1  import React from 'react'
2
3  export default function AddUsers() {
4    return (
5      <div className='container'>
6        <div className='row'>
7          <div className='col-md-6 offset-md-3 border rounded p-4 mt-2 shadow'>
8            <h2 className='text-center m-4'>Registrar user</h2>
9            <div className='mb-3'>
10             <label htmlFor='Name' className='form-label'>
11               Name
12             </label>
13             <input
14               type='text'
15               className='form-control'
16               placeholder='Enter your name'
17               name='name'
18             />
19           </div>
20         </div>
21       </div>
22     </div>
23   )
24 }
```

Y modificamos como se muestra a continuación

```
19  </div>
20  <div className='mb-3'>
21    <label htmlFor='UserName' className='form-label'>
22      User name
23    </label>
24    <input
25      type='text'
26      className='form-control'
27      placeholder='Enter your user name'
28      name='username'
29    />
30  </div>
31  <div className='mb-3'>
32    <label htmlFor='Email' className='form-label'>
33      Email
34    </label>
35    <input
36      type='text'
37      className='form-control'
38      placeholder='Enter your Email'
39      name='email'
40    />
41  </div>
```

Nuestra app debe verse así:

FullStack App Add User

Registrer user

Name

User name

Email

Ahora agregaremos un botón, colocamos el código inmediatamente después del div de Email

```
35 </div>
36 <input
37   type={"text"}
38   className='form-control'
39   placeholder='Enter your Email'
40   name='email'
41 />
42 </div>
43 </div>
44 </div>
45 </div>
46 )
```

FullStack App Add User

Registrer user

Name

User name

Email

Submit

Creamos el botono cancelar

```
39     name='email'
40     />
41   </div>
42   <button type='submit' className='btn btn-outline-primary'>Submit</button>
43   <button type='submit' className='btn btn-outline-danger mx-2'>Cancel</button>
44 </div>
45 </div>
46 </div>
```

FullStack App Add User

Registrer user

Name

Enter your name

User name

Enter your user name

Email

Enter your Email

Submit

Cancel

10) Almacenar información del usuario dentro del estado

Vamos a AddUser.js y crearemos una función donde se inicialice el objeto user y reciba las propiedades de name, username y email, además de crear setUser el array que permita actualizar los valores.

```
import React, { useState } from 'react';

export default function AddUsers() {

  const[user, setUser]=useState({
    name:"",
    username:"",
    email:""
  });
```

Recordemos que en automático se importa react, ahora vamos a desestructurar el objeto user. Es decir que se van a crear 3 variables nuevas (name, username, y email) y se asignaran las que se guardaron en el objeto user del paso anterior

```
export default function AddUsers() {  
  
  const[user, setUser]=useState({  
    name:"",  
    username:"",  
    email:""  
  });  
  
  const{name,username,email}=user;
```

Ahora agregamos el atributo value en cada elemento y ligándolo a las variables del objeto desestructurando

```
<div className='mb-3'>  
  <label htmlFor='Name' className='form-label'>  
    Name  
  </label>  
  <input  
    type={"text"}  
    className='form-control'  
    placeholder='Enter your name'  
    name='name'  
    value={name}/>  
/>  
</div>  
<div className='mb-3'>  
  <label htmlFor='UserName' className='form-label'>  
    User name  
  </label>  
  <input  
    type={"text"}  
    className='form-control'  
    placeholder='Enter your user name'  
    name='username'  
    value={username}/>  
/>  
</div>  
<div className='mb-3'>  
  <label htmlFor='Email' className='form-label'>  
    Email  
  </label>  
  <input  
    type={"text"}  
    className='form-control'  
    placeholder='Enter your Email'  
    name='email'  
    value={email}/>  
/>  
</div>  
<button type='submit' className='btn btn-outline-primary'>Submit</button>  
<button type='submit' className='btn btn-outline-danger mx-2'>Cancel</button>  
</div>  
</div>
```

En React, el atributo `value` se utiliza en elementos de formulario (`<input>`, `<select>` y `<textarea>`) para especificar su valor inicial y permitir que el usuario interactúe con el elemento para actualizar su valor.

Cuando se utiliza el patrón de control de formulario en React (es decir, se utiliza el estado para mantener el valor de los elementos del formulario), es necesario asignar el valor de la variable de estado al atributo `value` del elemento del formulario para que refleje el valor actual del estado. De esta manera, cuando el usuario ingresa o selecciona un valor en el formulario, se actualiza la variable de estado correspondiente.

Ahora vamos a definir una función llamada `OnInputChange` que toma un evento (`e`) como argumento. Esta función se utiliza generalmente para manejar eventos de cambio (por ejemplo, cuando se escribe algo en un campo de entrada).

```
0
1  const {name, username, email} = user
2
3  const onInputChange = (e) => {}
4
5  return (
6    <div className='container'>
7      <div className='row'>
8        <div className='col-md-6 offset-md-3'>
```

Ahora agregaremos el siguiente código a cada elemento `onChange={(e)=>onInputChange(e)}`

```
<div className='mb-3'>
  <label htmlFor='Name' className='form-label'>
    Name
  </label>
  <input
    type='text'
    className='form-control'
    placeholder='Enter your name'
    name='name'
    value={name}
    onChange={(e)=>onInputChange(e)}
  />
</div>
<div className='mb-3'>
  <label htmlFor='UserName' className='form-label'>
    User name
  </label>
  <input
    type='text'
    className='form-control'
    placeholder='Enter your user name'
    name='username'
    value={username}
    onChange={(e)=>onInputChange(e)}
  />
</div>
</div>
</div>
</div>
</div>
```

```
</div>
<div className='mb-3'>
  <label htmlFor='Email' className='form-label'>
    Email
  </label>
  <input
    type='text'
    className='form-control'
    placeholder='Enter your Email'
    name='email'
    value={email}
    onChange={(e)=>onInputChange(e)}
  />
</div>
<button type='submit' className='btn btn-outline-primary'>Submit</button>
<button type='submit' className='btn btn-outline-danger mx-2'>Cancel</button>
</div>
</div>
</div>
```

Este código se utiliza en un elemento de entrada de formulario (como un input) y establece una función llamada `onInputChange` como el manejador de eventos para el evento `onChange`.

La función `onInputChange` es una función personalizada que se define para manejar los cambios en el elemento de entrada. Recibe el objeto de evento `e` como argumento y puede acceder al valor del elemento de entrada utilizando `e.target.value`.

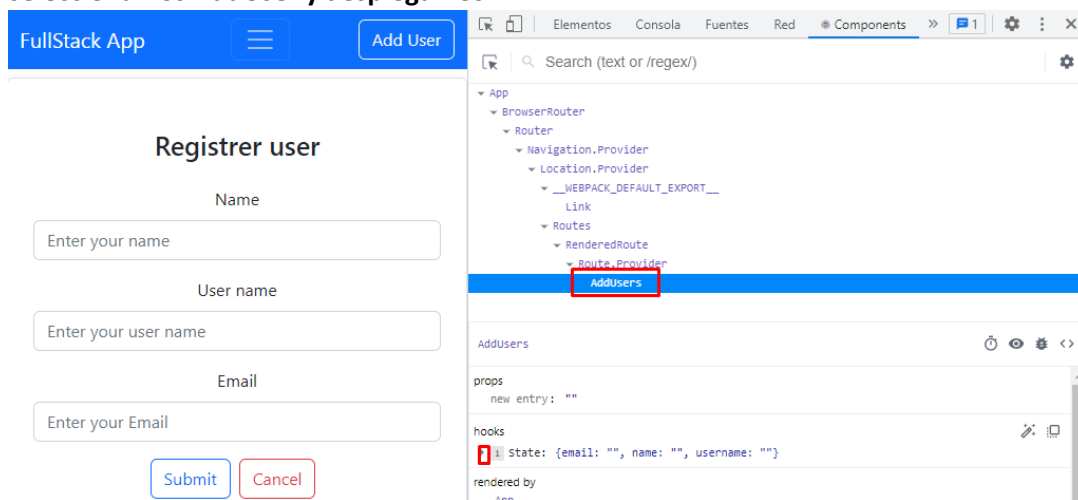
Al pasar `(e) => onInputChange(e)` como el manejador de eventos `onChange`, estamos asegurando que la función `onInputChange` se llame cada vez que el usuario cambie el valor del elemento de entrada.

Ahora modificamos la función `onInputChange`

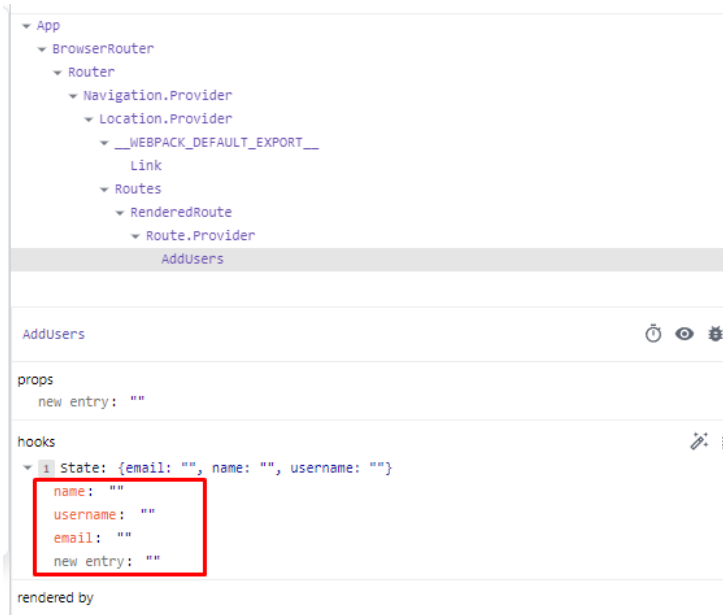
```
const onInputChange=(e)=>{  
  setUser({...user,[e.target.name]:e.target.value});  
};
```

En este código se utiliza destructuring assignment para obtener las propiedades `name`, `username` y `email` del estado `user`. Luego, se define una función `onInputChange` que será llamada cada vez que se detecte un cambio en los campos de un formulario, y que actualiza el estado `user` con los nuevos valores ingresados por el usuario. Para ello, se utiliza el método `setUser` y se crea un nuevo objeto usando el operador spread `...user`, para mantener las propiedades del objeto `user` que no se están modificando, y se actualiza la propiedad que se corresponde con el campo que cambió, utilizando el nombre del campo que se obtiene a través de `e.target.name` y el valor ingresado que se obtiene a través de `e.target.value`. De esta forma, se logra mantener el estado actualizado con los cambios que el usuario realiza en los campos del formulario.

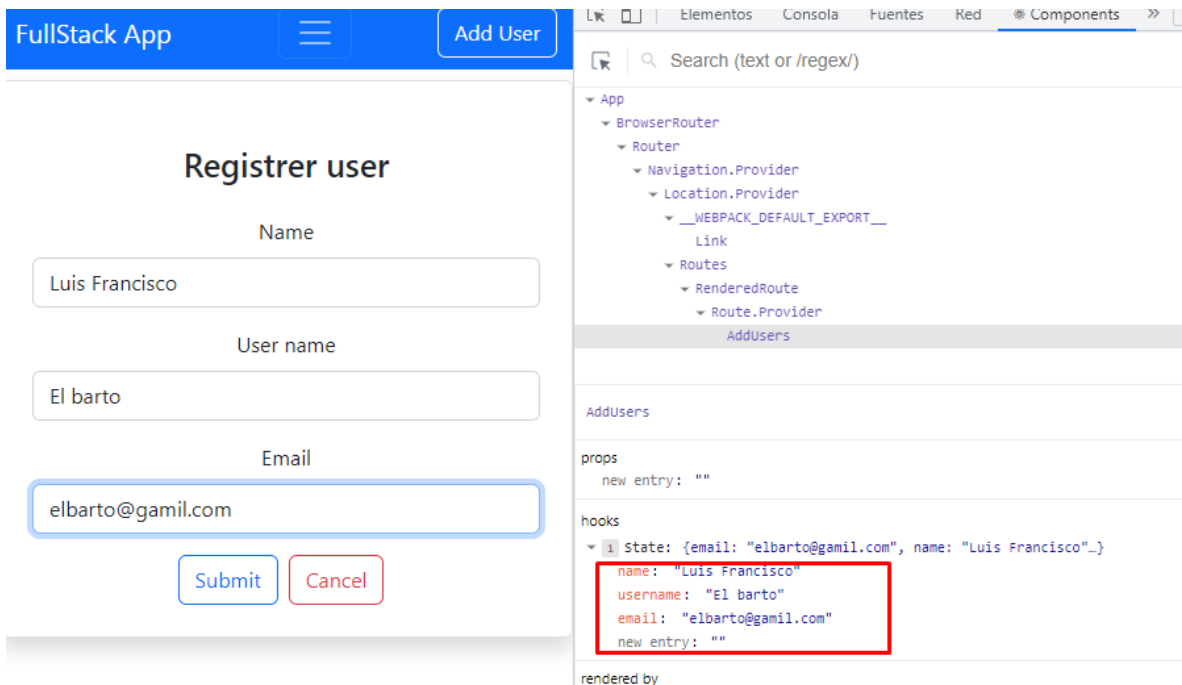
Podemos ir a nuestra app, dar clic derecho inspeccionar y revisar los componentes (para esto necesitamos tener instalado en nuestro navegador la extensión React Developer Tool), seleccionamos `AddUser` y desplegamos



Podemos observar que están vacíos los campos



Comenzamos a llenar los campos en la app y observamos cómo se van llenando los campos que antes estaban vacíos



11) AXIOS POST: Publicar datos en la base de datos

Le daremos función al botón submit para ello crearemos una nueva función en AddUser.js

```
const onSubmit=(e)=>{  
  e.preventDefault();  
};
```

El código `const onSubmit = (e) => { e.preventDefault(); };` es una función que se utiliza comúnmente en formularios de React para prevenir que la página se actualice o recargue cuando se envía el formulario.

Cuando se hace clic en el botón "submit" de un formulario, el navegador intenta enviar los datos a un servidor web. Sin embargo, en una aplicación de React, es común procesar los datos del formulario en la misma página, sin tener que enviarlos a un servidor.

Por lo tanto, la función `onSubmit` se utiliza para evitar que la página se actualice o se recargue cuando se envía el formulario. Al llamar a `e.preventDefault()`, se evita que el evento de envío del formulario se propague, lo que permite que el desarrollador procese los datos del formulario y actualice la página según sea necesario, sin tener que recargar la página completa.

Ahora agregamos axios y en automático se debe importar axios

```
const onSubmit= async(e)=>{  
  e.preventDefault();  
  await axios.post("http://localhost:8080/users",user);  
};  
  
import React, { useState } from 'react';  
import axios from 'axios'  
  
export default function AddUsers() {
```

Recordemos que la URL es la que utilizamos en nuestra base de datos

`await axios.post("http://localhost:8080/users",user)` es una llamada a la API utilizando el método HTTP POST a la URL "http://localhost:8080/users" con un objeto de usuario como datos en el cuerpo de la solicitud. El uso de `await` indica que la solicitud debe esperar la respuesta antes de continuar con la ejecución del código. En este caso, la respuesta puede contener información sobre el estado de la operación (por ejemplo, si se ha creado un nuevo usuario en la base de datos).

Ahora crearemos un form donde encerraremos toda la estructura del formulario

```

1  <div className='col-md-6 offset-md-3 border rounded p-4 mt-2 shadow'>
2    <h2 className='text-center m-4'>Registrer user</h2>
3
4    <form>
5      <div className='mb-3'>
6        <label htmlFor='Name' className='form-label'>
7          Name
8        </label>
9        <input
10         type='text'
11         className='form-control'
12         placeholder='Enter your name'
13         name='name'
14         value={name}
15         onChange={(e)=>onInputChange(e)}
16       />
17     </div>
18     <div className='mb-3'>
19       <label htmlFor='UserName' className='form-label'>
20         User name
21       </label>
22       <input
23         type='text'
24         className='form-control'
25         placeholder='Enter your user name'
26         className='form-control'
27         placeholder='Enter your user name'
28         value={username}
29         onChange={(e)=>onInputChange(e)}
30       />
31     </div>
32     <div className='mb-3'>
33       <label htmlFor='Email' className='form-label'>
34         Email
35       </label>
36       <input
37         type='text'
38         className='form-control'
39         placeholder='Enter your Email'
40         name='email'
41         value={email}
42         onChange={(e)=>onInputChange(e)}
43       />
44     </div>
45     <button type='submit' className='btn btn-outline-primary'>Submit</button>
46     <button type='submit' className='btn btn-outline-danger mx-2'>Cancel</button>
47   </form>
48 </div>

```

Ahora en el form agregaremos lo siguiente

```
<form onSubmit={(e)=>onSubmit(e)}>
<div className='mb-3'>
```

En React, el elemento `<form>` se usa para crear formularios y manejar su envío de datos. El atributo `onSubmit` es un controlador de eventos que se activa cuando el formulario se envía mediante el botón "Enviar" o cuando se presiona la tecla "Enter" en uno de los campos del formulario.

En este caso, `<form onSubmit={(e)=>onSubmit(e)}>` establece que el controlador de eventos `onSubmit` se llamará cuando se envíe el formulario. El evento `e` se pasa como argumento al controlador de eventos, lo que permite que el controlador acceda a los datos del formulario y realice alguna acción antes de enviar los datos al servidor. El `preventDefault` evita que la página se recargue cuando se envía el formulario.

11.1) useNavigate

Crearemos la siguiente función y en automatico se debe importar la biblioteca react-router-dom para su uso

```
import { useNavigate } from 'react-router-dom';

export default function AddUsers() {

  let navigate=useNavigate()

  const[user, setUser]=useState({
    name:"",
    username:"",
    email:""
  });
```

`let navigate=useNavigate()` se utiliza en una aplicación React que utiliza la biblioteca `react-router-dom` para permitir la navegación entre diferentes rutas en una aplicación de página única. `useNavigate` es un hook que proporciona `react-router-dom` para permitir la navegación programática a través del código JavaScript. La función `navigate` se utiliza para cambiar de ruta a través del código en lugar de hacer clic en un enlace o un botón. Por ejemplo, después de enviar un formulario o de realizar una acción específica, se puede utilizar la función `navigate` para redirigir al usuario a una nueva página o ruta.

Agregamos la función `navigate` a la función `onSubmit`

```
const onSubmit= async(e)=>{
  e.preventDefault();
  await axios.post("http://localhost:8080/users",user)
  navigate("/")
};
```

Al colocar "/" hacemos que nos mande a la página de inicio, ahora podemos probar la app
Llenamos los campos y damos clic en submit

Registrar user

Name

demoPost

User name

demoPostUser

Email

demoPost@gmail.com

Submit

Cancel

Nos regresa a la página inicial y aparece nuestro registro

#	Name	User name	Email	Action		
1	Luis	Luis182	Luis_brav182@hotmail.com	View	Edit	Delete
2	Paola	Paola182	Paola@hotmail.com	View	Edit	Delete
3	demoPost	demoPostUser	demoPost@gmail.com	View	Edit	Delete

Podemos ir MySQL y revisar que también se realizó el registro

```
1 • select * from user;
```

	id	email	name	username
1	Luis_brav182@hotmail.com	Luis	Luis182	
2	Paola@hotmail.com	Paola	Paola182	
52	demoPost@gmail.com	demoPost	demoPostUser	
	NULL	NULL	NULL	NULL

11.2) Botón Cancel

Hacemos lo mismo que hicimos en [Enrutamiento REAC-ROUTER-DOM](#), cambiamos el botón por el elemento Link, borramos `type='submit'` y asignamos la ruta a la que debe ir

```
</div>
<button type='submit' className='btn btn-outline-primary'>Submit</button>
<Link type='submit' className='btn btn-outline-danger mx-2' to="/">
  Cancel</Link>
</form>
```

Importamos Link en caso de que nos e haga en automatico

```
import React, { useState } from 'react';
import axios from 'axios'
import { useNavigate } from 'react-router-dom';
import { Link } from "react-router-dom";
```

Recordemos que en [AddUser.js](#) , enrutamos por lo cual cuando el Link se "/" nos mandara en home, como lo especificamos en App.js, como se muestra a continuación

```
function App() {  
  return (  
    <div className="App">  
      <Router>  
        <Navbar/>  
  
        <Routes>  
          <Route exact path="/" element={<Home/>}/>  
          <Route exact path="/adduser" element={<AddUsers/>}/>  
        </Routes>  
      </Router>  
    </div>  
  );  
}
```

y listo si probamos el botón cancel debería mandarnos a Home

12) Manejo de excepciones de spring boot

Regresamos a IntelliJ IDEA vamos a `UserControlle.java` y escribimos el siguiente texto

```
@GetMapping("/users")
List<User> getAllUsers() { return userRepository.findAll(); }

@GetMapping("/user/{id}")
User getUserById(@PathVariable Long id){
    return userRepository.findById(id).orElseThrow(()->new UserNotFoundException(id));
}
```

Este código define un **endpoint** de una API REST en Java utilizando el método **@GetMapping**. El **endpoint** tiene la ruta **/user/{id}** y espera recibir un parámetro de tipo **Long** que corresponde al id del usuario que se desea buscar.

El método **getUserById** es el controlador que manejará la petición HTTP y, una vez que reciba el id, buscará el usuario en una base de datos utilizando el repositorio **userRepository**. Si el usuario no es encontrado, lanzará una excepción personalizada **UserNotFoundException**. Si el usuario es encontrado, se devolverá como respuesta al cliente de la API.

Como podemos ver nos marca error en `UserNotFoundException` por que aún no está creada, la creamos en la carpeta de exceptions, al crearla nos aseguramos de heredar de la clase `RuntimeException`

```
public class UserNotFoundException extends RuntimeException{
}
```

Agregamos el siguiente código

```
package com.ejemploFull.fullstackbackend.exception;

public class UserNotFoundException extends RuntimeException{
    1 usage
    public UserNotFoundException(Long id){
        super("Could not found the user whit id "+ id);
    }
}
```

Este código define una excepción personalizada llamada **UserNotFoundException** que extiende la clase **RuntimeException**. Cuando se lanza esta excepción, se mostrará un mensaje que indica que

no se pudo encontrar un usuario con el ID proporcionado. Esta excepción se usa comúnmente en controladores de API REST para indicar que un recurso solicitado no existe.

Ahora importamos la clase en UserController.java para quitar el error

```
UserController.java x UserNotFoundException.java x FullstackBackendApplication.java x User.java x
1 package com.ejemploFull.fullstackbackend.controller;
2
3 import com.ejemploFull.fullstackbackend.exception.UserNotFoundException;
4 import com.ejemploFull.fullstackbackend.model.User;
5 import com.ejemploFull.fullstackbackend.repository.UserRepository;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.web.bind.annotation.*;
8
9 import java.util.List;
10
11 @RestController
12
13 @GetMapping("/user/{id}")
14 User getUserById(@PathVariable Long id){
15     return userRepository.findById(id).orElseThrow(()->new UserNotFoundException(id));
16 }
```

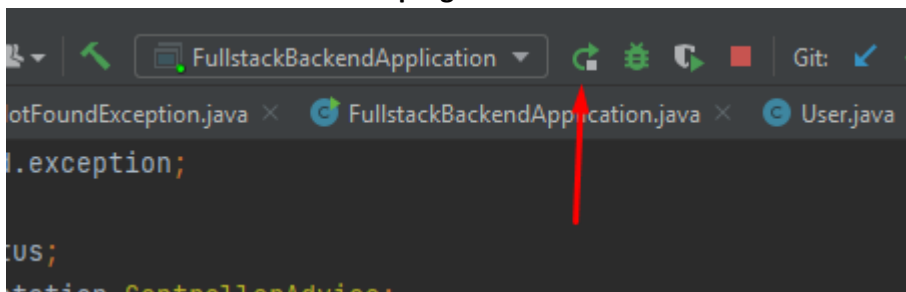
Ahora en la carpeta exception creamos UserNotFoundAdvice

```
package com.ejemploFull.fullstackbackend.exception;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import java.util.HashMap;
import java.util.Map;
@ControllerAdvice
public class UserNotFoundAdvice {
    @ResponseBody
    @ExceptionHandler(UserNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public Map<String,String> exceptionHandler(UserNotFoundException exception){
        Map<String,String> errorMap=new HashMap<>();
        errorMap.put("errorMessage",exception.getMessage());
        return errorMap;
    }
}
```

El código que muestras es un controlador de excepciones en Spring Boot. El objetivo de `@ControllerAdvice` es permitir que los controladores se compartan en varios controladores y de manejar excepciones de manera centralizada.

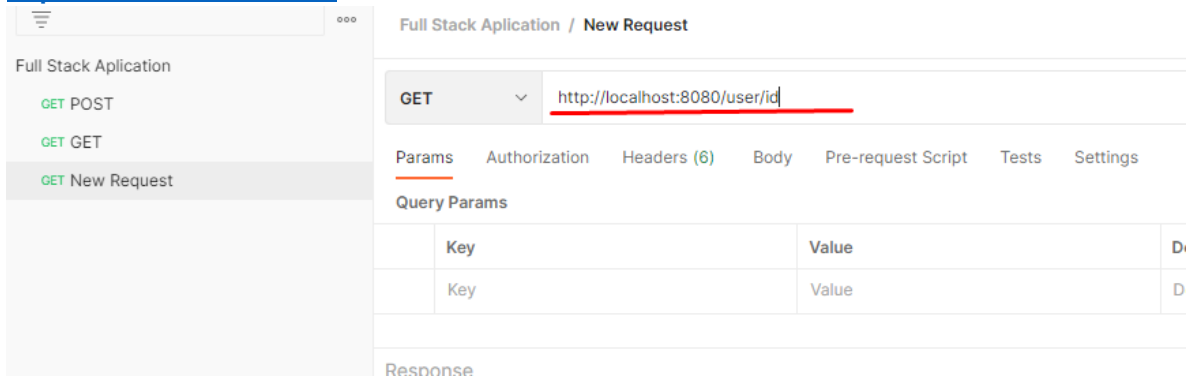
En este caso, `UserNotFoundAdvice` es una clase que maneja excepciones cuando se lanza `UserNotFoundException`, que es una excepción personalizada creada en el código anterior. Cuando se lanza esta excepción, el controlador de excepciones devuelve una respuesta HTTP con un código 404 (NOT FOUND) y un mapa JSON que contiene un mensaje de error personalizado. Esto se realiza utilizando las anotaciones `@ResponseBody` y `@ExceptionHandler`.

Ahora solo volvemos a correr el programa



12.1) Probando las excepciones con Postman

Creamos una nueva request, en la dirección nos aseguraremos que sea <http://localhost:8080/user/id>



En lugar de id en el url pondremos 1 y veremos cómo nos regresa el valor del usuario con id 1 o podemos usar cualquier número que ya tenemos en nuestra base de datos

	id	email	name	username
▶	1	Luis_brav182@hotmail.com	Luis	Luis182
	2	Paola@hotmail.com	Paola	Paola182
	52	demoPost@gmail.com	demoPost	demoPostUser
*	NULL	NULL	NULL	NULL

Por el momento usaremos 1

GET ▼ http://localhost:8080/user/1

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (8) Test Results 🌐 200 OK

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "id": 1,
3   "username": "Luis182",
4   "name": "Luis",
5   "email": "Luis_brav182@hotmail.com"
6 }
```

Ahora colocaremos un id inexistente en este caso 20

GET ▼ http://localhost:8080/user/20

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (8) Test Results 🌐 404 Not Found 67 ms

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "errorMessage": "Could not found the user whit id 20"
3 }
```

Observamos que nos manda el mensaje de error que no se encontró el usuario con id

13) PUTMapping: editar usuario

En UerController.java colocamos el siguiente código

```
}
@PutMapping("/user/{id}")
User updateUser(@RequestBody User newUser, @PathVariable Long id){
    return userRepository.findById(id)
        .map(user -> {
            user.setUsername(newUser.getUsername());
            user.setName(newUser.getName());
            user.setEmail(newUser.getEmail());
            return userRepository.save(user);
        }).orElseThrow(() -> new UserNotFoundException(id));
}
```

Este código se utiliza para actualizar un usuario existente en la base de datos.

- [@PutMapping\("/user/{id}"\)](#) indica que la solicitud HTTP debe ser manipulada por este método cuando se realiza una solicitud PUT en la URL ["/user/{id}"](#). La variable de ruta [{id}](#) se puede utilizar para identificar el usuario que se va a actualizar.
- [@RequestBody User newUser](#) indica que los datos del usuario a actualizar se pasan en el cuerpo de la solicitud HTTP y se deben mapear a un objeto User en Java.
- [@PathVariable Long id](#) indica que la variable de ruta [{id}](#) debe mapearse a una variable Long en Java.
- [userRepository.findById\(id\)](#) busca en la base de datos el usuario con el ID especificado.
- [.map\(user -> {...}\)](#) permite actualizar el objeto User recuperado si existe. Si no existe, se lanza una excepción [UserNotFoundException](#).
- [userRepository.save\(user\)](#) guarda el usuario actualizado en la base de datos y devuelve el objeto User actualizado.

Una vez colocado volvemos a correr el programa

13.1) Probando con Postman

Volvemos a postman y creamos otro request con la siguiente url <http://localhost:8080/user/id>, estaremos enBody, en raw y colocaremos el siguiente texto entre llaves

```
{
    "username": "",
    "name": "",
    "email": ""
}
```



Ahora en el url colocaremos 1, y escribiremos los siguiente en el texto

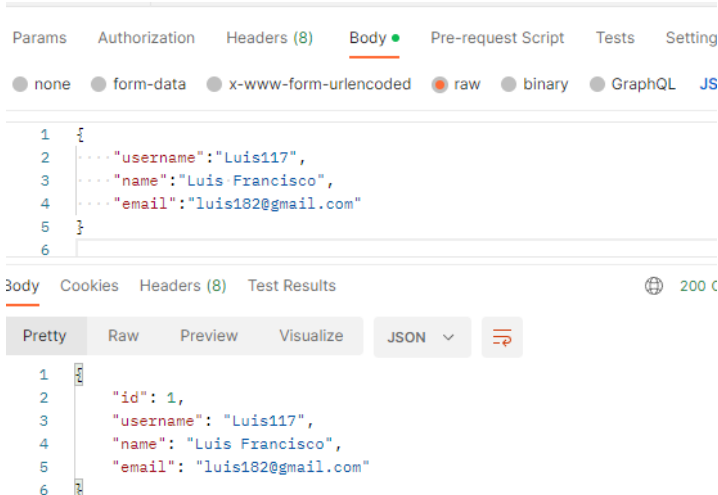
```
{
  "username": "Luis117",
  "name": "Luis Francisco",
  "email": "luis182@gmail.com"
}
```

Antes de ellos vamos a MySQL y revisamos el estado de nuestros registros

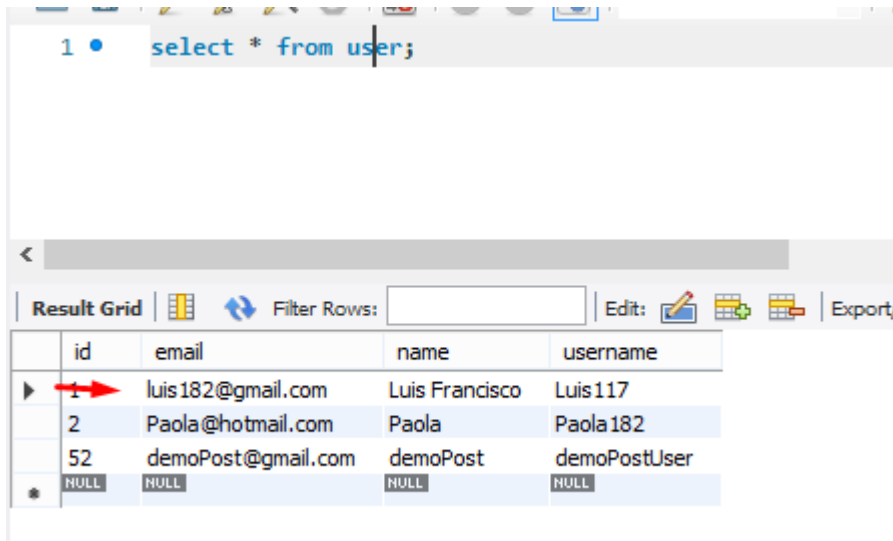
```
1 • select * from user;
```

id	email	name	username
1	Luis_brav182@hotmail.com	Luis	Luis182
2	Paola@hotmail.com	Paola	Paola182
52	demoPost@gmail.com	demoPost	demoPostUser
NULL	NULL	NULL	NULL

Presionamos send en postman



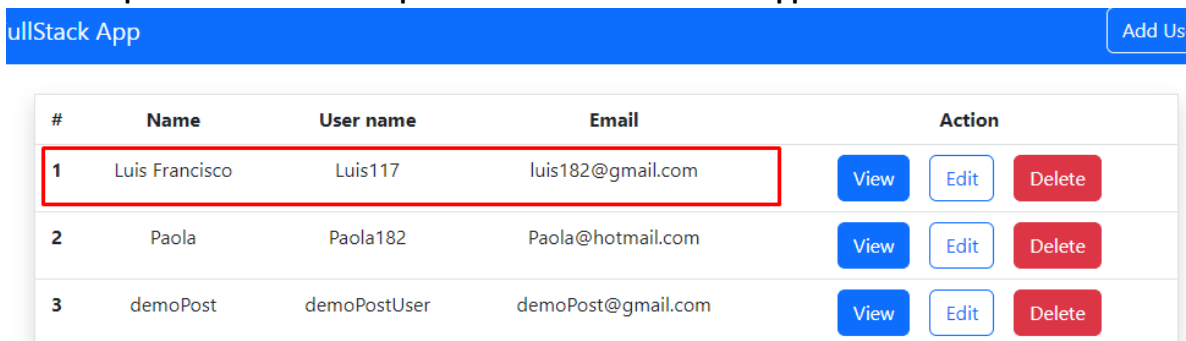
Podemos ir a MySQL y revisar otra vez nuestros registros



The screenshot shows a MySQL database client interface. At the top, a SQL query is entered: `select * from user;`. Below the query, the results are displayed in a table with the following columns: `id`, `email`, `name`, and `username`. The results are as follows:

	id	email	name	username
1	1	luis182@gmail.com	Luis Francisco	Luis117
2	2	Paola@hotmail.com	Paola	Paola182
52	52	demoPost@gmail.com	demoPost	demoPostUser
*	NULL	NULL	NULL	NULL

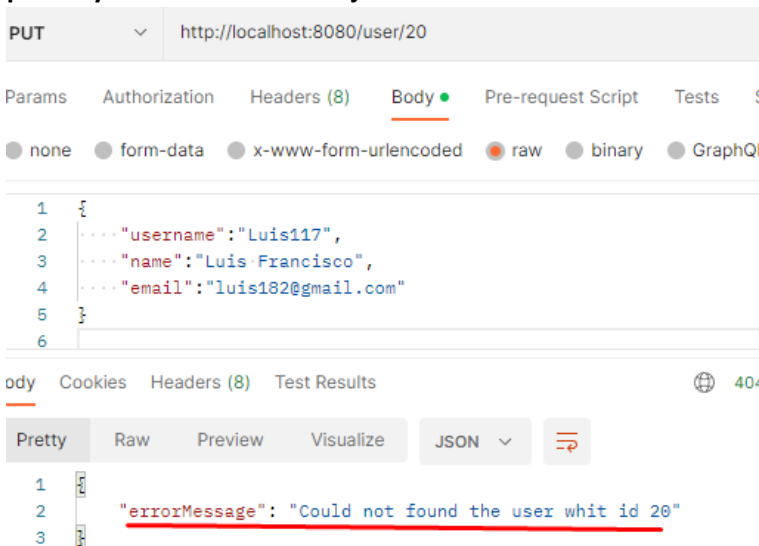
Y vemos que se han actualizado por ende también en nuestra app



The screenshot shows a web application header with the text "FullStack App" and a button labeled "Add User". Below the header, there is a table with the following columns: `#`, `Name`, `User name`, `Email`, and `Action`. The table contains three rows of user data:

#	Name	User name	Email	Action
1	Luis Francisco	Luis117	luis182@gmail.com	<button>View</button> <button>Edit</button> <button>Delete</button>
2	Paola	Paola182	Paola@hotmail.com	<button>View</button> <button>Edit</button> <button>Delete</button>
3	demoPost	demoPostUser	demoPost@gmail.com	<button>View</button> <button>Edit</button> <button>Delete</button>

También podemos ver qué pasa si intentamos editar un usuario que no existe cambiamos el id por 20 y nos saldrá el mensaje de error



The screenshot shows a REST client interface. At the top, a PUT request is shown with the URL `http://localhost:8080/user/20`. Below the URL, the request body is shown in JSON format:

```
{
  "username": "Luis117",
  "name": "Luis Francisco",
  "email": "luis182@gmail.com"
}
```

Below the request body, the response is shown. The response is a JSON object with the following structure:

```
{
  "errorMessage": "Could not found the user whit id 20"
}
```

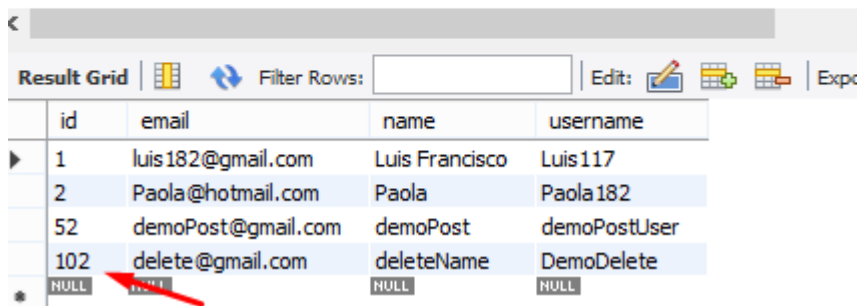
14) @DeleteMapping: borrar usuario con id específico

Agregamos un usuario más para pruebas de delete, ya sea que se agregue desde postman como lo hicimos en: [Damos de alta nuestro primer registro](#) o bien desde nuestra app

#	Name	User name	Email	Action		
1	Luis Francisco	Luis117	luis182@gmail.com	View	Edit	Delete
2	Paola	Paola182	Paola@hotmail.com	View	Edit	Delete
3	demoPost	demoPostUser	demoPost@gmail.com	View	Edit	Delete
4	deleteName	DemoDelete	delete@gmail.com	View	Edit	Delete

En MySQL podemos ver su id, en este caso 102

```
1 • select * from user;
```



	id	email	name	username	
1	1	luis182@gmail.com	Luis Francisco	Luis117	
2	2	Paola@hotmail.com	Paola	Paola182	
52	52	demoPost@gmail.com	demoPost	demoPostUser	
102	102	delete@gmail.com	deleteName	DemoDelete	
*	NULL	NULL	NULL	NULL	

Vamos a UserControlle.java y escribimos el siguiente código

```
@DeleteMapping("/user/{id}")
String deleteUser(@PathVariable Long id){
    if(!userRepository.existsById(id)){
        throw new UserNotFoundException(id);
    }
    userRepository.deleteById(id);
    return "User with id "+id+" has benn deleted success";
}
```

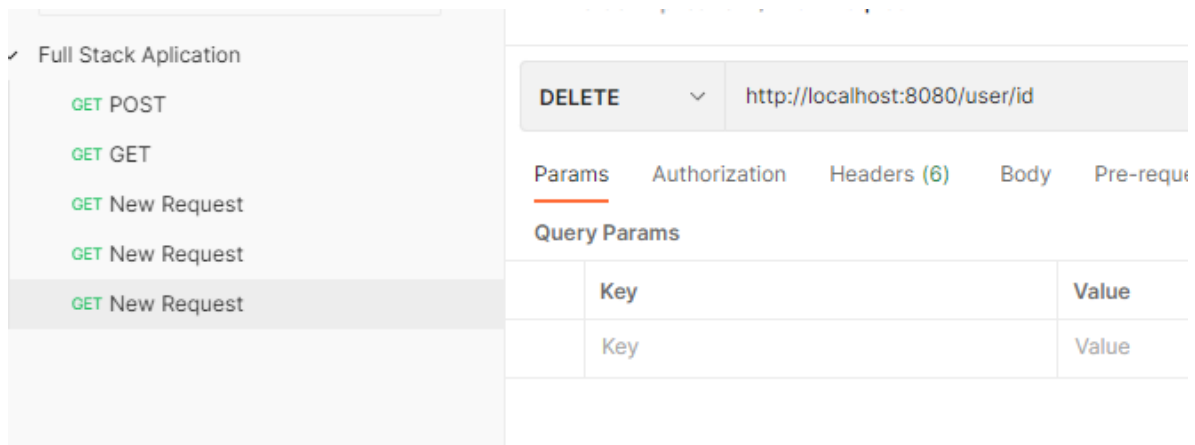
Este código define un método para eliminar un usuario existente en la base de datos mediante la anotación `@DeleteMapping`. El valor `"/user/{id}"` indica la ruta de acceso para el método y `{id}` indica que el valor del identificador del usuario se pasará en la URL.

El método acepta un parámetro de tipo Long con la anotación `@PathVariable` para obtener el valor del identificador de usuario de la URL. Primero verifica si el usuario existe en la base de datos mediante el método `existsById` del repositorio. Si el usuario no existe, se lanza una excepción `UserNotFoundException`. Si el usuario existe, se elimina utilizando el método `deleteById` del repositorio y se devuelve un mensaje de éxito con el identificador del usuario eliminado.

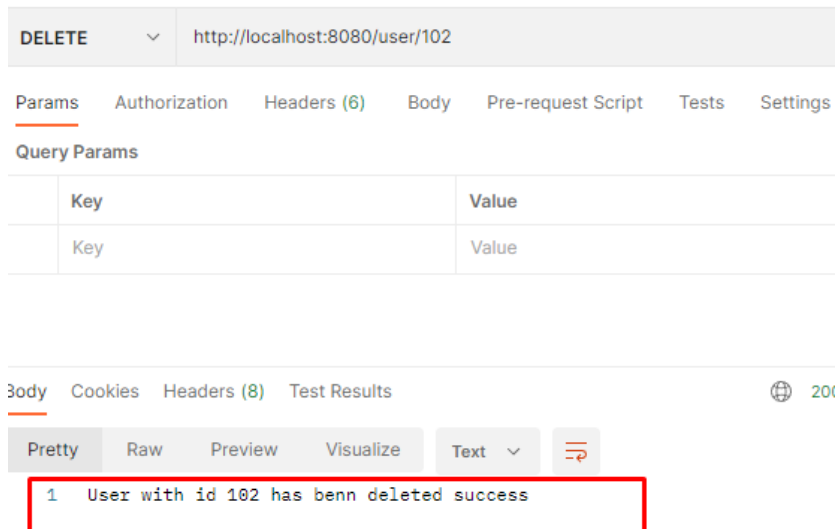
Corremos el programa

14.1) Probando con Postman

Volvemos a postman y creamos otro request con la siguiente url <http://localhost:8080/user/id>,



Ahora colocamos el id del perfil que creamos de prueba y presionamos send



Podemos revisar en MySQL y en nuestra app

```
1 • select * from user;
```

Result Grid | Filter Rows: | Edit:

	id	email	name	username
▶	1	luis182@gmail.com	Luis Francisco	Luis117
	2	Paola@hotmail.com	Paola	Paola182
	52	demoPost@gmail.com	demoPost	demoPostUser
*	NULL	NULL	NULL	NULL

FullStack App Add User

#	Name	User name	Email	Action
1	Luis Francisco	Luis117	luis182@gmail.com	View Edit Delete
2	Paola	Paola182	Paola@hotmail.com	View Edit Delete
3	demoPost	demoPostUser	demoPost@gmail.com	View Edit Delete

Si intento borrar un usuario que no existe como el 102 que fue el que borramos en los pasos anteriores sucede lo siguiente

DELETE ▼ http://localhost:8080/user/102 Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

body Cookies Headers (8) Test Results 404 Not Found 77 ms 315 B Save as Example ...

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "errorMessage": "Could not found the user whit id 102"
3 }
```

15) AXIOS PUT: editar usuario

Volvemos a Visual Code y ahora vamos a EditUser.js , iniciamos con rfc y ahora borramos el interior de la función

```
src > users > JS EditUsers.js > EditUsers
1  import React from 'react'
2
3  export default function EditUsers() {
4
5  }
6
```

Vamos a AddUser.js y copiamos el código dentro de la función nos servirá de base

```
src > users > JS AddUsers.js > AddUsers
import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { Link } from "react-router-dom";

export default function AddUsers() {
  let navigate=useNavigate();

  const[user, setUser]=useState({
    name: "",
    username: "",
    email: ""
  });

  const{name,username,email}=user;

  const onChange=(e)=>{
    setUser({...user,[e.target.name]:e.target.value});
  };

  const onSubmit= async(e)=>{
    e.preventDefault();
    await axios.post("http://localhost:8080/user",user);
    navigate("/")
  }
}
```

Lo pegamos dentro del cuerpo de EditUser.js, importamos todo lo que necesitamos


```

import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { Link } from "react-router-dom";

export default function EditUsers() {
  let navigate=useNavigate();

  const[user, setUser]=useState({
    name:"",
    username:"",
    email:""
  });

  const{name,username,email}=user;

  const onChange=(e)=>{
    setUser({ ...user,[e.target.name]:e.target.value});
  };

  const onSubmit= async(e)=>{
    e.preventDefault();
    await axios.post("http://localhost:8080/user",user)
  };

```

Cambiamos el título por Edit user+

```

return (
  <div className='container'>
    <div className='row'>
      <div className='col-md-6 offset-md-3 border rounded p-4 mt-2 shadow'>
        <h2 className='text-center m-4'>Edit user</h2>

        <form onSubmit={ (e)=>onSubmit(e)}>
          <div className='mb-3'>
            <label htmlFor='Name' className='form-label'>
              Name
            </label>

```

Vamos a Home.js y cambiamos el elemento button de edit por un elemento Link e importamos Link de react-router-dom

```

    <td>{user.username}</td>
    <td>{user.email}</td>
    <td>
      <button className='btn btn-primary mx-2'>View</button>
      <Link className='btn btn-outline-primary mx-2'>Edit</Link>
      <button className='btn btn-danger mx-2'>Delete</button>
    </td>
  </tr>
</tbody>
</table>

```

Agregamos la ruta

```

    <button className='btn btn-primary mx-2'>View</button>
    <Link className='btn btn-outline-primary mx-2'
      to={` /edituser/${user.id}`}>
      Edit</Link>
    <button className='btn btn-danger mx-2'>Delete</button>
  </td>

```

[to={`/edituser/\\${user.id}`} en un componente Link de react-router-dom especifica la ruta a la que se navegará al hacer clic en el enlace. En este caso, la ruta es `/edituser/\${user.id}`, lo que significa que se navegará a la página de edición de un usuario específico, donde `user.id` es el ID del usuario en cuestión.](#)

Vamos a App.js y agregamos la nueva ruta

```

import AddUsers from './users/AddUsers';
import EditUsers from './users/EditUsers';

function App() {
  return (
    <div className="App">
      <Router>
        <Navbar/>

        <Routes>
          <Route exact path="/" element={<Home/>}/>
          <Route exact path="/adduser" element={<AddUsers/>}/>
          <Route exact path="/edituser/:id" element={<EditUsers/>}/>
        </Routes>
      </div>
    )
}

```

Primero debemos observar que en esta ruta colocamos `/:id`, esto con el fin de que se muestre el id con el que estamos trabajando, en automático se debe importar la función desde `EditUsers`, si no lo importamos nosotros

Regresamos a `EditUser.js` y cambiamos la función `onSubmit`

```
const onSubmit= async(e)=>{
  e.preventDefault();
  await axios.put(`http://localhost:8080/user/${id}`,user)
  navigate("/")
};
```

Cambiamos la función `post` por `put`, cambiamos las comillas dobles por comillas invertidas para poder poner el código de javascript y colocamos `${id}` para indicar que se editara el usuario con el id especificado

El método `axios.put` es utilizado en aplicaciones web para enviar una solicitud HTTP PUT a un servidor para actualizar o modificar un recurso existente en el servidor. Este método envía los datos a través del cuerpo de la solicitud en formato JSON o en otros formatos soportados, y espera una respuesta del servidor en función del resultado de la operación de actualización o modificación del recurso en el servidor.

Por lo tanto, `axios.put` es útil cuando se necesita actualizar un recurso específico en un servidor, como por ejemplo un perfil de usuario, un registro en una base de datos, un artículo de una tienda en línea, entre otros casos de uso.

Definamos una función para recopilar la id e importamos `useParams` from `react-router-dom`

```
export default function EditUsers() {
  let navigate=useNavigate();

  const {id}=useParams();
```

```
import React, { useEffect, useState } from 'react';
import axios from 'axios'
import { useNavigate } from 'react-router-dom';
import { Link } from "react-router-dom";
import { useParams } from 'react-router-dom';
```

[useParams\(\)](#) es un hook proporcionado por React Router que permite acceder a los parámetros definidos en una ruta en una aplicación React.

En este caso, el código `const {id}=useParams()` está utilizando `useParams()` para obtener el valor del parámetro `id` en la ruta actual. Es decir, si la ruta actual es `/users/1`, entonces `id` será igual a 1. Luego, el valor de `id` se puede usar para hacer consultas específicas a la base de datos o para realizar otras acciones en la aplicación en función del ID del usuario.

Definimos una función para llamar los parámetros del usuario

```
const onSubmit= async(e)=>{
  e.preventDefault();
  await axios.put(`http://localhost:8080/user/${id}`,user)
  navigate("/")
};

const loadUsers = async()=>{
  const result=await axios.get(`http://localhost:8080/user/${id}`)
  setUser(result.data)
}
```

Este código define una función `loadUsers` que utiliza el método `axios.get` para hacer una solicitud a un servidor en la dirección `http://localhost:8080/user/${id}` y recuperar los datos del usuario con el `id` especificado. Luego, la función utiliza `setUser` para actualizar el estado del componente con los datos del usuario recuperado.

Ahora ejecutamos con ayuda de `useEffect`

```
const onChange=(e)=>{
  setUser({ ...user,[e.target.name]:e.target.value});
};

useEffect(()=>{
  loadUsers()
},[]);

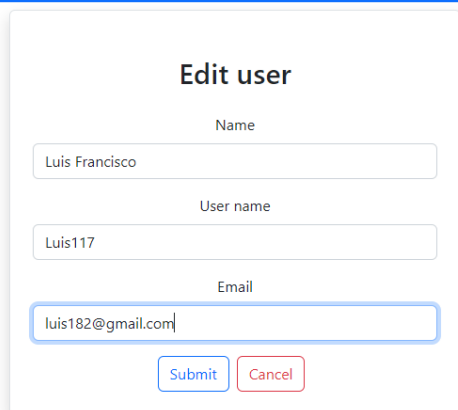
const onSubmit= async(e)=>{
  e.preventDefault();
  await axios.put(`http://localhost:8080/user/${id}`,user)
```

El hook `useEffect` es una función que se ejecuta automáticamente cada vez que se actualiza el componente. En este caso, el `useEffect` se utiliza para cargar los datos de usuario una vez que el componente es montado por primera vez.

El primer argumento que recibe `useEffect` es una función que se ejecutará cuando el componente se monte. En este caso, se llama a la función `loadUsers()` que realiza una petición HTTP utilizando `axios` para obtener los datos del usuario con el ID correspondiente. El segundo argumento es una lista de dependencias que se utiliza para controlar cuándo se debe ejecutar el `useEffect`. En este caso, se pasa un array vacío `[]` como segundo

argumento, lo que significa que **useEffect** sólo se ejecutará cuando el componente se monte por primera vez y no se volverá a ejecutar en futuras actualizaciones del componente.

Y listo podemos editar el perfil que queramos



Edit user

Name

Luis Francisco

User name

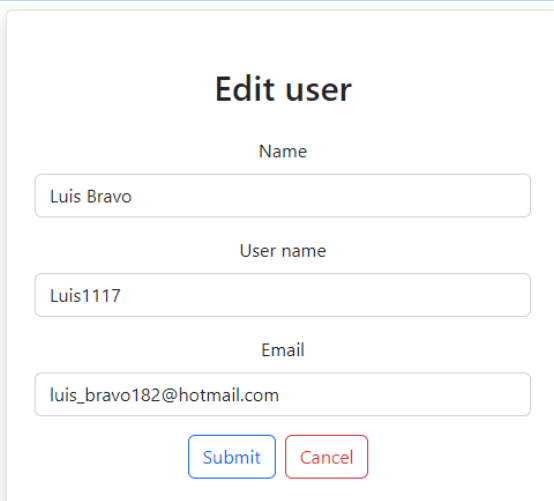
Luis117

Email

luis182@gmail.com

Submit Cancel

Cambiamos los contenidos y enviamos



Edit user

Name

Luis Bravo

User name

Luis1117

Email

luis_bravo182@hotmail.com

Submit Cancel

Y se debe mostrar el cambio en home, también podemos revisar en MySQL



#	Name	User name	Email	Action		
1	Luis Bravo	Luis1117	luis_bravo182@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
2	Paola	Paola182	Paola@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
3	demoPost	demoPostUser	demoPost@gmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>

```
1 • select * from user;
```

Result Grid					Filter Rows:		Edit:		Export/Import:	
	id	email	name	username						
▶	1	luis_bravo182@hotmail.com	Luis Bravo	Luis1117						
	2	Paola@hotmail.com	Paola	Paola182						
	52	demoPost@gmail.com	demoPost	demoPostUser						
*	NULL	NULL	NULL	NULL						

16) AXIOS DELETE : borrar usuario

Vamos a Home.js y agregamos la función para borrar similar a la que creamos para editar pero ahora con `axios.delete`, nos aseguramos que este user no users

```
};

const deleteUser=async(id)=>{
  await axios.delete(`http://localhost:8080/user/${id}`)
  loadUsers()
};

return (
  <div className='container'>
```

Nuestra función aún no está terminada tenemos que agregar use params para asignar el id, e importamos de react-router-dom

```

2 import axios from 'axios'
3 import { Link } from "react-router-dom";
4 import { useParams } from 'react-router-dom';
5
6 export default function Home() {
7
8     const [users, setUsers] = useState([]);
9
10    const {id} = useParams();
11
12    useEffect(() => {

```

Nuestra función esta lista falta asignarla al botón delete, agregamos la función con ayuda de `onClick`, indicando que vamos a mandar el `id` del user a la función `deleteUser`

```

<button className='btn btn-primary mx-2'>View</button>
<Link className='btn btn-outline-primary mx-2'
to={` /edituser/${user.id}`}>
  Edit</Link>
<button className='btn btn-danger mx-2'
onClick={() => deleteUser(user.id)}
>Delete</button>

```

Ahora podemos borrar un registro

#	Name	User name	Email	Action		
1	Luis Bravo	Luis1117	luis_bravo182@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
2	Paola	Paola182	Paola@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
3	demoPost	demoPostUser	demoPost@gmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
4	deleteName2	DemoDelete2	delete2@gmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>

#	Name	User name	Email	Action		
1	Luis Bravo	Luis1117	luis_bravo182@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
2	Paola	Paola182	Paola@hotmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>
3	demoPost	demoPostUser	demoPost@gmail.com	<button>View</button>	<button>Edit</button>	<button>Delete</button>

17) Ver un usuario específico

Primero vamos a Home.js y cambiamos el botón por Link y agregamos la ruta a seguir

```
<td>
  <Link className='btn btn-primary mx-2'
    to={` /viewuser/${user.id}`}
    >View</Link>
  <Link className='btn btn-outline-primary mx-2'
    to={` /edituser/${user.id}`}>
```

Agregamos la ruta en App.js

```
<Navbar/>

<Routes>
  <Route exact path="/" element={<Home/>}/>
  <Route exact path="/adduser" element={<AddUsers/>}/>
  <Route exact path="/edituser/:id" element={<EditUsers/>}/>
  <Route exact path="/viewuser/:id" element={<ViewUsers/>}/>
</Routes>
</Router>
</div>
```

Vamos a ViewUsers.js iniciamos con rfc, podemos copiar el cuerpo de EditUsers y editar

Borramos onSubmit

```
loadUsers();
},[]);

const onSubmit= async(e)=>{
  e.preventDefault();
  await axios.put(`http://localhost:8080/user/${id}`,user);
  navigate("/")
};

const loadUsers = async()=>{
  const result=await axios.get(`http://localhost:8080/user/${id}`)
```

Por lo tanto lo quitamos también de form


```

return (
  <div className='container'>
    <div className='row'>
      <div className='col-md-6 offset-md-3 border rounded p-4 mt-2 shadow'>
        <h2 className='text-center m-4'>Edit user</h2>

        <form onSubmit={ (e) => onSubmit(e)}>
          <div className='mb-3'>
            <label htmlFor='Name' className='form-label'>
              Name
            </label>

```

Borramos de igual manera el botón

```

      </div>
    </div>
    <button type='submit' className='btn btn-outline-primary'>Submit</button>
    <Link type='submit' className='btn btn-outline-danger mx-2' to="/">
      Cancel</Link>
    </form>
  </div>
</div>

```

Y finalmente agregamos disabled={true} a cada input para que no se pueda editar

```

    <form >
      <div className='mb-3'>
        <label htmlFor='Name' className='form-label'>
          Name
        </label>
        <input
          disabled={true}
          type={"text"}
          className='form-control'
          placeholder='Enter your name'
          name='name'
          value={name}
          onChange={ (e) => onChange(e)}
        />
      </div>
      <div className='mb-3'>
        <label htmlFor='UserName' className='form-label'>
          User name
        </label>
        <input
          disabled={true}
          type={"text"}

```

```
<div className='mb-3'>
  <label htmlFor='Email' className='form-label'>
    Email
  </label>
  <input
    disabled={true}
    type={"text"}
    className='form-control'
    placeholder='Enter your Email'
    name='email'
    value={email}
    onChange={(e)=>onInputChange(e)}
  />
</div>
  <Link type='submit' className='btn btn-outline-danger'
Cancel</Link>
```

Ahora podemos dar clic en cualquier usuario y ver su información