Cuenta de Microsoft:

Email: [Jonatanesprellalopez@gmail.com](mailto:Jonatanesprellalopez@gmail.com)

Contrasena:

Bases de datos:

Usuario: Cec-fgi

Contrasena: C-F-2020

BASE DE DATOS

Link de MySQL 🡪 <https://dev.mysql.com/downloads/mysql/5.5.html?os=3&version=5>  
  
MySQL 🡪 Credentials

User Default: Root

Password: 12345678

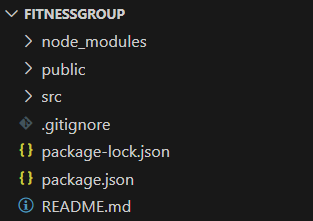
Creación de la Base de datos



Front-End

Comando Principal para la generación del esqueleto Front – End

**npx create-react-app FITNESSGROUP**



Para el uso del **react-router-dom** de debe instalar con este comando:

**npm install react-router-dom**

backend

Para la instalación de backend:

Instalamos Axios en la carpeta FRONTEND:

**npm install axios**

**Se realiza la Instalación de PHP**

<https://windows.php.net/download#php-8.2>

**A momento de instalar PHP se debe ir a propiedades de “Mi Equipo”, Luego ir a ”Propiedades avanzadas del equipo” luego a “Variables de entorno”, Por ultimo a “path” y agregas la ruta del PHP “C:\Program Files\php-8.2.14’**

**Se realiza la instalación de Composer**

<https://getcomposer.org/download/>

**INSTALACIÓN DE FRONT**

npm init -y

Esto creará el archivo package.json que permitirá dar seguimiento a todos los scripts de nuestra app y manejar cualquiera de las dependencias que necesite nuestra app.

El código de nuestro servidor vivirá en una carpeta con el nombre server. Vayamos a crearlo.

Ahí dentro pondremos un solo archivo, desde el cual correremos nuestro servidor: index.js.

Usaremos Express para crear un servidor web simple para que corra en el puerto 3001 por defecto, es decir, al no darle un valor distinto a la variable de entorno, PORT Heroku le dará este valor (3001) cuando despleguemos nuestra aplicación.

// server/index.js

const express = require("express");

const PORT = process.env.PORT || 3001;

const app = express();

app.listen(PORT, () => {

console.log(`Server listening on ${PORT}`);

});

Luego en nuestra terminal instalaremos Express como una dependencia para utilizar en nuestro proyecto:

npm i express

Luego de esto, crearemos un script en el archivo package.json que iniciará la ejecución de nuestro servidor web con el comando npm start:

// server/package.json

...

"scripts": {

"start": "node index.js"

},

...

Finalmente, podemos correr nuestra aplicación usando este script con el comando npm start en nuestra terminal y deberíamos verlo ejecutándose en el puerto 3001:

npm start

> node index.js

Server listening on 3001 (servidor escuchando en el puerto 3001)

En nuestro caso, podemos simplemente enviar un mensaje a nuestro app React que diga "Hola desde el servidor!", en un objeto JSON.

El código de abajo crea un endpoint para la ruta /api.

Si nuestra app realiza una petición GET a esa ruta, respondemos (usando res, que significa respuesta) con nuestros datos JSON:

// server/index.js

...

app.get("/api", (req, res) => {

res.json({ message: "Hola desde el servidor!" });

});

app.listen(PORT, () => {

console.log(`Servidor escuchando en el puerto: ${PORT}`);

});

Nota: Asegúrate de posicionar esto previo a la función app.listen.

Ya que hemos realizado cambios a nuestro código Node, necesitaremos reiniciar nuestro servidor.

Para este fin, termina el "start script" presionando Command/Ctrl + C. Luego lo reiniciaremos usando npm start, nuevamente, desde el terminal.

Y para probar esto podemos simplemente visitar http://localhost:3001/api en nuestro navegador web y ver en la pantalla el mensaje:

**Paso 3: Crear el cliente React**

Luego de haber creado el backend (servidor), vayamos al frontend (cliente).

Abre otra pestaña en la terminal y utiliza el comando create-react-app para crear un nuevo proyecto React con el nombre client:

npx create-react-app client

Luego de esto tendremos una aplicación React con todas sus dependencias instaladas.

El único cambio que tenemos que hacer es agregar una propiedad llamada proxy a nuestro archivo package.json.

Esto nos permitirá realizar peticiones a nuestro servidor Node sin tener que proporcionar el origen en el que se encuentra corriendo (http://localhost:3001) cada vez que le hagamos una petición de red:

// client/package.json

...

"proxy": "http://localhost:3001",

...

Luego podemos iniciar nuestra app React al ejecutar el script "start", que es el mismo para nuestro servidor Node. Primero asegúrate de entrar en la carpeta /client recién creada usando cd client en el terminal - luego ya podrás ejecutar el comando npm start correctamente.

Luego de esto la app iniciará en localhost:3000:

cd client

npm start

Compiled successfully!

You can now view client in the browser.

Local: http://localhost:3000

## Paso 4: Realiza peticiones HTTP desde React a Node

Ahora que tenemos una aplicación React en funcionamiento, queremos usarla para interactuar con nuestra API.

Veamos como podemos traer datos desde el endpoint /api que hemos creado anteriormente.

Para hacer esto podemos dirigirnos al componente App.js dentro de nuestra carpeta src y hacer una petición HTTP utilizando useEffect.

Haremos una petición GET simple a nuestro backend usando el Fetch API y luego tendremos nuestros datos de regreso en formato JSON.

Una vez que tengamos los datos, obtendremos la propiedad con el nombre "message" (para poder tomar el saludo que hemos enviado desde el servidor) y entonces ponerlo en una variable de "estado" ( state ) llamada data.

Esto nos permitirá mostrar ese mensaje en nuestra página (asumiendo que sí lo tenemos). Estamos usando un condicional en nuestro JSX para decir que si nuestros datos no están disponibles se muestre el texto "Loading...".

// client/src/App.js

import React from "react";

import logo from "./logo.svg";

import "./App.css";

function App() {

const [data, setData] = React.useState(null);

React.useEffect(() => {

fetch("/api")

.then((res) => res.json())

.then((data) => setData(data.message));

}, []);

return (

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<p>{!data ? "Loading..." : data}</p>

</header>

</div>

);

}

export default App;

## Paso 5: Despliega tu aplicación a la web con Heroku

Finalmente, desplegaremos nuestra aplicación a la web.

Primero, dentro de la carpeta /client de nuestro proyecto, asegúrate de remover el repositorio Git que se inicializa automáticamente al usar el comando create-react-app.

Esto es esencial para desplegar nuestra aplicación, ya que vamos a configurar un repositorio Git en el directorio principal de nuestro proyecto ( react-node-app), es decir, no en /client:

cd client

rm -rf .git

Cuando desplegamos nuestro backend Node (servidor) y nuestro frontend React (cliente) van a ser servidos desde el mismo dominio (i.e. mycoolapp.herokuapp.com).

Cómo nuestras peticiones son manejadas por nuestra API Node, necesitamos escribir algo de código que mostrará nuestro app React cuando sea visitado por nuestros usuarios (por ejemplo, cuando visiten la página de inicio).

Podemos implementar esto en server/index.js agregando el siguiente código:

// server/index.js

const path = require('path');

const express = require('express');

...

// Hacer que node sirva los archivos de nuestro app React

app.use(express.static(path.resolve(\_\_dirname, '../client/build')));

// Manejar las peticiones GET en la ruta /api

app.get("/api", (req, res) => {

res.json({ message: "Hello from server!" });

});

// Todas las peticiones GET que no hayamos manejado en las líneas anteriores retornaran nuestro app React

app.get('\*', (req, res) => {

res.sendFile(path.resolve(\_\_dirname, '../client/build', 'index.html'));

});

Lo que este código hará primero es darle acceso a Node a nuestro proyecto React compilado usando la función express.static para archivos estáticos.

Y si una petición GET llega que no esté contemplada para su manejo en nuestra ruta /api, nuestro servidor responderá con nuestro app React.

**Este código permite que nuestra app React y Node sea desplegada en el mismo dominio.**

Entonces podremos indicar a nuestro app (Node) como hacerlo, agregando un script build a nuestro archivo package.json con el propósito de que compile nuestra app para producción:

// server/package.json

...

"scripts": {

"start": "node server/index.js",

"build": "cd client && npm install && npm run build"

},

...

También recomendaría agregar un campo llamado "engines" (motores), dónde querrás especificar la versión de Node que estás usando para compilar tu proyecto. Esto será utilizado para el despliegue.

Puedes conocer la versión que estás usando de Node al ejecutar el comando node -v y puedes escribir el resultado en la propiedad "engines" (ejemplo v. 14.14.4):

// server/package.json

"engines": {

"node": "tu-número-de-versión-node"

}

Luego de esto estaremos listos para desplegar usando Heroku, así que asegúrate de tener una cuenta en [Heroku.com](https://heroku.com/).

Una vez que hayas iniciado sesión en Heroku y te encuentres viendo tu panel de control, seleccionarás: "New > Create New App" y proporcionarás un nombre único para tu app.

Luego de esto, querrás instalar el Heroku CLI (Interfaz de línea de comandos) para que puedas desplegar tu aplicación cuando sea que realices cambios usando Git. Podemos instalar el CLI ejecutando el siguiente comando:

sudo npm i -g heroku

Una vez instalado, podrás iniciar sesión en Heroku con el CLI usando el comando heroku login:

heroku login

Press any key to login to Heroku (Presiona cualquier tecla para iniciar sesión en Heroku.

Ya que has iniciado sesión en Heroku desde el CLI necesitas seguir las instrucciones de despliegue para nuestra recién creada aplicación desde la pestaña "Deploy".

Los siguientes comandos inicializarán un nuevo repositorio Git para nuestro proyecto, agregarán tus archivos ahí, realizarán un commit y agregarán un Git remote para Heroku.

git init

heroku git:remote -a inserta-el-nombre-de-tu-app-aqui

git add .

git commit -am "Desplegar app en Heroku"

Luego el último paso es publicar nuestra aplicación subiéndola ( git push )al remoto Git de Heroku que justo hemos agregado usando:

git push heroku master

¡Felicitaciones! Nuestra aplicación full-stack Node-React ya está en línea! ?