



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
2do. Semestre 2022

Manual técnico

Grupo 4

Practica 1

Practica 1 del curso de Análisis y diseño de Sistemas, 2do Semestre 2022.

Prerrequisitos:

Se necesita tener instalado y poseer conocimientos básicos de las siguientes herramientas y/o lenguajes de programación:

- Git
- Git Flow – WorkFlow
- JavaScript
- NodeJs
- React
- Draw.io

Repositorio - Github

El repositorio de la práctica en github es el siguiente:
https://github.com/jorgeisa/practica1_grupo2.git

Contenido

Backend

Nodejs

Es un entorno de tiempo de ejecución de JavaScript, en el cual creamos o iniciamos nuestro servidor con el nombre de index.js. Que encontraremos en la dirección Backend/Src.

Este lo iniciamos con el comando **npm init -y**

En el cual importamos las librerías que utilizaremos, en este caso será el express, cors. También encontraremos el import de la clase o función indexRoutes.

Luego iniciaremos nuestro servidor express con la constante app y lo iniciaremos en el puerto 3000 y para correr el programa usaremos el comando:

Node index.js

```
Backend > src > JS index.js > ...
You, hace 5 horas | 2 authors (Isaac Xicol and others)
1  import express from 'express'
2  import indexRoutes from './routes/index.js'
3  import cors from 'cors'      You, hace 5 horas • [ADD]bugfix ...
4  const app = express()
5  app.use(cors());
6  // Settings
7  app.set('port', 3000)
8
9  // Routes
10 app.use(indexRoutes)
11
12 app.listen(app.get('port'), ()=>{
13   console.log(`Aplicación corriendo en el puerto ${app.get('port')} :^`)
14 })
15
16
```

En la función o clase indexRoutes agregaremos los diferentes endpoints pedidos, como sería el verificar si el numero es par o impar, la función Fibonacci, invertir una palabra o funciones básicas como multiplicar o dividir, entre otras cosas.

```
// Funcion Raiz
router.get('/raiz/:numero', function(req, res) {
  if (!isNaN(req.params.numero) ){
    let result_raiz = raiz(req.params.numero);

    res.json({
      mensaje: result_raiz
    }) }else{
      res.json({mensaje: 'No es un numero'})
    }
  })
})
```

```
router.get('/alreves/:palabra', function(req, res) {
  if (isNaN(req.params.palabra) ){
    let cadenaAlreves = alreves(req.params.palabra);

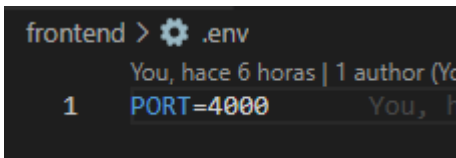
    res.json({
      mensaje: cadenaAlreves
    }) }else{
      res.json({mensaje: 'No es una palabra'})
    }
  })
})
```

React

React es una herramienta que nos ayuda a crear interfaces de usuario interactivas, el cual utilizamos para crear nuestro frontend. Iniciamos esta herramienta con el siguiente comando:

Npx créate-react-app frontend

Luego creamos un archivo .env para especificarle el puerto en el 4000



```
frontend > .env
You, hace 6 horas | 1 author (Yo)
1 PORT=4000 You, h
```

Instalamos las dependencias que usaremos con el comando:

```
npm install "dependencias"
```

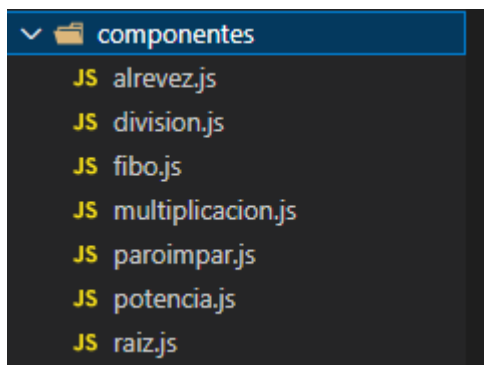
las dependencias instaladas son las siguientes:

- Axios
- Bootstrap
- Dotenv
- Html-react-parser
- Reactstrap
- Wouter
- Web-vitals

Configuramos nuestra función App del archivo src/App.js

```
function App() {
  return (
    <>
      <Navbar collapseOnSelect expand="lg" bg="success" variant="dark" >
        <Navbar.Brand > &nbsp; &nbsp; Practica 1 </Navbar.Brand>
        <Navbar.Toggle aria-controls="responsive-navbar-nav" />
        <Navbar.Collapse id="responsive-navbar-nav" >
          </Navbar.Collapse>
        </Navbar>
      <br />
      <Container>
        <Route exact path="/" component={Alreves}></Route>
        <Route exact path="/" component={Division}></Route>
        <Route exact path="/" component={Multi}></Route>
        <Route exact path="/" component={Fibo}></Route>
        <Route exact path="/" component={Parimpar}></Route>
        <Route exact path="/" component={Potencia}></Route>
        <Route exact path="/" component={Raiz}></Route>
      </Container>
    </>
  );
}
```

Creamos una carpeta llamada componentes en la cual estarán nuestras diversas funciones o rutas de la siguiente manera:



En cada una de los componentes se vera una estructura similar a la siguiente:

```
import React, { Component, useEffect, useState } from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import Form from 'react-bootstrap/Form';
import Container from 'react-bootstrap/esm/Container';
import Row from 'react-bootstrap/Row';
import Col from 'react-bootstrap/Col';
import Button from 'react-bootstrap/Button';
import axios from 'axios';
import { Route } from "wouter"
```

Encontraremos los imports de las dependencias usadas ya sea de estilos o del mismo react.

```
const url = "http://localhost:3000/"

export default function Alreves() {
  const [val, setVal] = useState("");
  const [resultado, setResultado] = useState("");

  async function enviar() {
    const URL = url+"alreves/"+val
    const { data } = await axios.get(URL)
    setResultado(data.mensaje)
  }
}
```

Encontraremos la URL base del servidor en nodejs al principio, luego declararemos una función, con las variables que usaremos en este caso son val y resultado. Y la función que ira a traer la información a través la URL con los valores a enviar. Al mismo tiempo devolverá el valor que se mostrará en la interfaz de la siguiente manera.



Funcion Alreves

hola

Enviar Palabra

Resultado

aloh,false

Donde muestra la palabra hola al revés y también nos especifica si la palabra es un palíndromo.

Git – Git flow

Se utilizó la herramienta git-flow para trabajar la práctica y crear las ramas release, hotfix, features y develop. Los comandos usados serán descritos a continuación.

- Antes de comenzar a trabajar con git-flow en el repositorio se creó la rama develop.

```
$ git branch develop
```

```
$ git push -u origin develop
```

- Creada la rama develop se procedió a iniciar el proyecto con git-flow.

```
isaacxicol@isaac-xicol:~/Documents/Gitkraken/Gitlab/practica1_grupo2$ git flow init
```

- **Inicio de Ramas Features:** Para la creación de las ramas features se utilizó el siguiente comando gitflow. Este comando crea la rama feature especificada a partir de la rama develop. En esta rama se trabajaron las distintas tareas especificadas como la división y potencia. El nombre especificado de las ramas features fueron colocados conforme a su desarrollador y su respectivo carnet.

Ejemplo con el desarrollador 4:

```
isaacxicol@isaac-xicol:~/Documents/Gitkraken/Gitlab/practica1_grupo2$ git flow feature start Desarrollador4-201801210
```

- **Fin de Ramas Features:** Para terminar de trabajar en las ramas features de los desarrolladores se utilizó el siguiente comando. Este comando realiza el respectivo merge en la rama develop del trabajo realizado en la rama, además se elimina la rama feature trabajada, puesto que el trabajo realizado en la rama se guarda en develop.

Ejemplo con el feature creado para el desarrollador 4:

```
isaacxicol@isaac-xicol:~/Documents/Gitkraken/Gitlab/practica1_grupo2$ git flow feature finish Desarrollador4-201801210
```

- **Inicio de Ramas Release:** Comando usado para crear las ramas release, se especifica la versión del programa y la rama es creada a partir de la rama develop.

Ejemplo con la versión 4.0.0

```
isaacxicol@isaac-xicol:~/Documents/Gitkraken/Gitlab/practica1_grupo2$ git flow release start 4.0.0
```

- **Fin de Ramas Release:** Comando que se ejecuta cuando el desarrollador termina de trabajar en la rama. Se realiza el merge en la rama “main” de la rama release trabajada, también crea un tag que es creado en el repositorio y se actualiza la rama main y la develop.

Ejemplo con la versión 4.0.0

```
isaacxicol@isaac-xicol:~/Documents/Gitkraken/Gitlab/practica1_grupo2$ git flow release finish 4.0.0
```

- **Inicio de Ramas Hotfix:** Comando que realiza una rama hotfix para realizar conexiones en la versión de la rama main. Estas ramas se realizan a partir de la rama main.

Ejemplo con corrección en función de división.

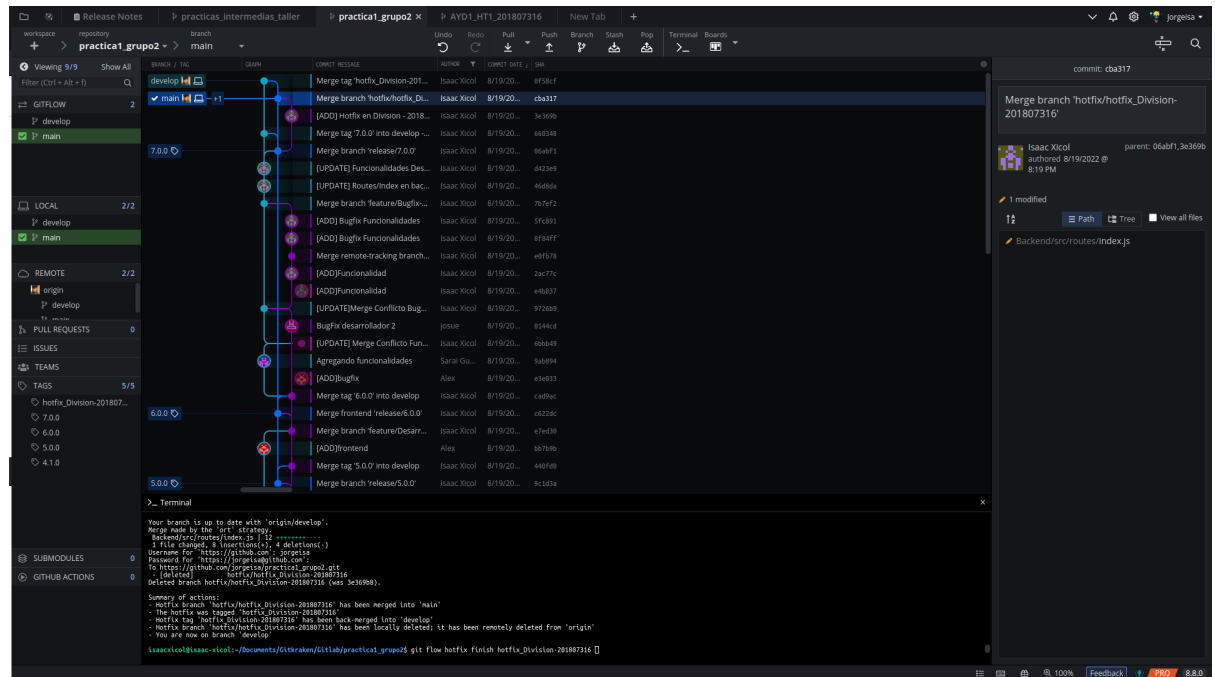
```
isaacxicol@isaac-xicol:~/Documents/Gitkraken/Gitlab/practica1_grupo2$ git flow hotfix start hotfix_Division-201807316
```

- **Fin de Ramas Hotfix:** Comando que finaliza el trabajo realizado en la rama. Primero realiza el merge en la rama “main” de la rama hotfix trabajada y luego realiza el merge en la rama “develop” de la rama hotfix trabajada. La rama hotfix es eliminada, ya que este trabajo es guardado tanto en la rama main como en la rama develop.

Ejemplo con corrección en función de división.

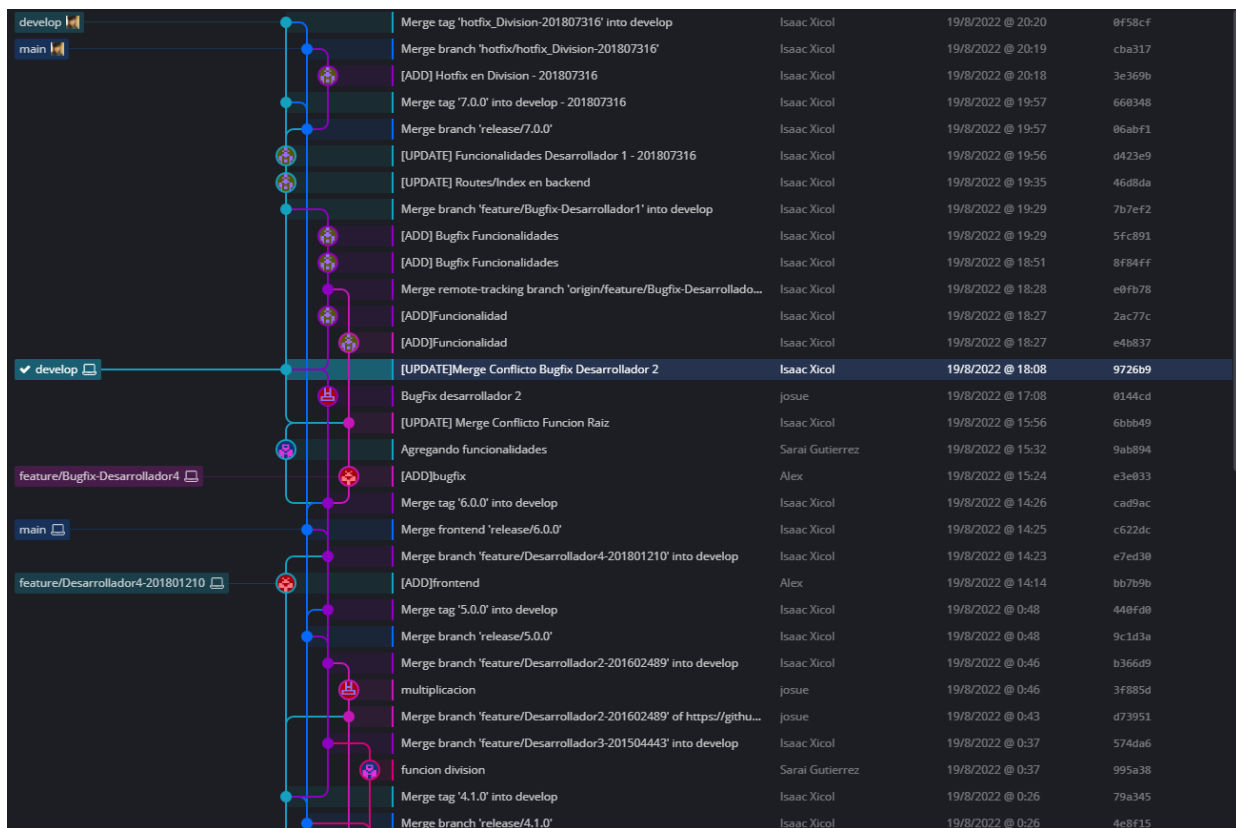
```
isaacxicol@isaac-xicol:~/Documents/Gitkraken/Gitlab/practica1_grupo2$ git flow hotfix finish hotfix_Division-201807316
```

- **Gitkraken:** El trabajo fue realizado en la consola de gitkraken, así mismo se puede ver el progreso de las ramas y commits.



Diagramas

Árbol de gitKraken



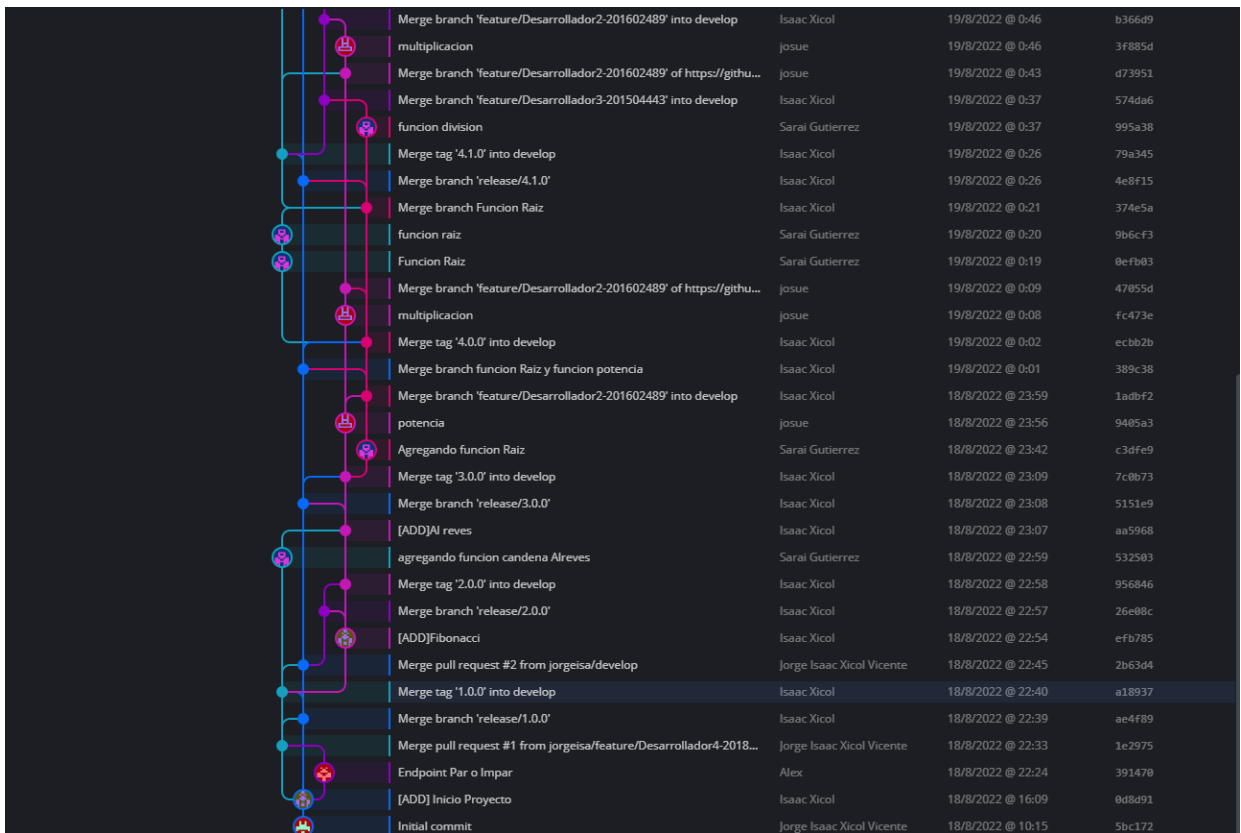


Diagrama hecho en Draw.io

19/8/22, 20:36

[AyD1]Workflow-Practica1

