Olimpiada Nacional de Educación Tecnico Profesional 2022

Instancia Institucional (Programación)

Escuela

Escuela de Educación Secundaria Nº2 “Educación y Trabajo”

Lugar

Mar del Plata, Buenos Aires, Argentina

Fecha

12/09/2022

Integrantes

- Agustin Simonte Del Russo
- Iñaki Emanuel Cabanes
- Federico Manuel Ali

Indice

Github:

- https://github.com/ElAgux/INET

Tareas Diarias

12.09.2022

Inicio: 16:00 p.m

Temas vistos:

- Se comenzó a analizar el problema.
- Se designó la división de trabajos.
- Se realizó el reconocimiento de las tablas principales en la base de datos, luego de esto se realizó un DER(Diagrama entidad relación).
- Se investigó sobre los lenguajes y las API a utilizar,

Fin: 22:00 p.m

13.09.2022

Inicio: 09:00 a.m (Fuera del horario escolar)

Temas vistos:

- Se perfeccionó la estructura de la base de datos.
- Se comenzo el backend.

Fin: 12:00 p.m

Inicio: 16:00 p.m

Temas vistos:

- Se continuo con el backend.
- Se comienza a programar el frontend.
- Se comienza a probar códigos con css.

Fin: 20:00 p.m

14.09.2022

Inicio: 09:00 a.m (Fuera del horario escolar)

Temas vistos:

- Se arreglaron algunas consultas del backend.
- Comenzamos a agregar los ABM (Alta, Baja y Modificación) correspondientes.

Fin: 12:00 p.m

Inicio: 16:00 p.m

Temas vistos:

- Continuamos con los ABM.
- Se le comienza a dar un estilo a la página.

Fin: 20:00 p.m

15.09.2022

Inicio: 09:00 a.m (Fuera del horario escolar)

Temas vistos:

- Decidimos modificar la estructura de la Base de Datos.
- Definimos el estilo a seguir en la página

Fin: 12:00 p.m

Inicio: 16:00 p.m

Temas vistos:

- Agregamos tablas a la Base de Datos.
- Comenzamos a gestionar el Sistema de Turnos

Fin: 20:00 p.m

15.09.2022

Inicio: 09:00 a.m (Fuera del horario escolar)

Temas vistos:

- Se incluyó un Inicio sesión para los administradores.
- Se añadieron los controladores de las tablas anteriormente agregadas relacionadas con el apartado de las visitas guiadas.
- Se realizó el componente para crear las visitas guiadas

Fin: 12:00 p.m

Inicio: 16:00 p.m

Temas vistos:

- Unificación de las partes del proyecto y corrección de errores

Fin: 20:00 p.m

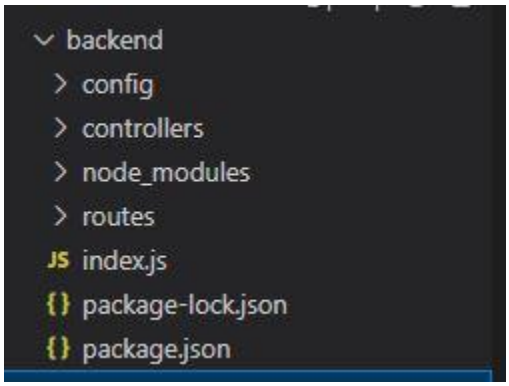
Tecnologias usadas

LENGUAJE USADO: JavaScript.
BACKEND: NODEJS
FRONTEND: REACTJS

BACKEND

Como servidor se uso Node junto a CORS como interprete de http.
Para hacer el pasaje de rutas entre el frontend y el backend se usó Express
Para la base de datos se usó MySql el cual corre sobre Apache, y para el desarrollo de las API de cada una de las tablas y la conexión a la base de datos se usó Sequelize.

Para organizar las carpetas se usó la siguiente configuración.



En la carpeta config esta la **conexión** a la base de datos:

```
import {Sequelize} from "sequelize";

const conexion = new Sequelize('museoinet','root','',{
    host: 'localhost',
    dialect: 'mysql'
});

export default conexion;
```

En la carpeta **controllers** se encuentran los controladores/API de cada tabla(de algunas tablas no se llegaron a hacer por falta de tiempo)
Un ejemplo con la tabla Áreas:

```
export const getAreas = async(req,res)=>{
    try {
        const [response]= await conexion.query("SELECT A.IdArea, A.Nombre, A.Descripcion, A.Estado, TA.Nombre as NombreTipoArea FROM areas A
        LEFT OUTER JOIN tiposareas TA ON A.IdTipoArea = TA.IdTipoArea");
        res.status(200).json(response);
        console.log(JSON.stringify(response, null,1))
    } catch (error) {
        console.log(error.message);
    }
}
```

Esta es una consulta de tipo GET donde se hace un LEFT OUTER JOIN con la tabla TiposAreas.

En la carpeta **node_modules** se encuentra todo el funcionamiento de Node y sus dependencias así como las instaladas.

En la carpeta **routes** se encuentran las rutas de cada API de cada tabla. Las rutas deberían estar separadas en varios archivos pero para ganar tiempo se compactaron todo en un mismo archivo.
Primero se importan las API:

```
import express from "express";
import {
    getObras,
    getObraById,
    crearObra,
    editarObra,
    cambiarEstadoObra
} from "../controllers/obrasController.js";
```

Se puede ver que también se importa la dependencia express para poder hacer el ruteo.

Luego se define la ruta para cada controlador:

```
//get
router.get('/obras/:IdObra', getObraById);
router.get('/obras', getObras);
router.get('/tiposObras/:IdTipoObra', getTipoObraById);
router.get('/tiposObras', getTiposObras);
router.get('/areas/:IdArea', getAreaById);
router.get('/areas', getAreas);
router.get('/tiposAreas/:IdTipoArea', getTipoAreaById);
router.get('/tiposAreas', getTiposAreas);
router.get('/turnos', getTurnos);
router.get('/idiomas', getIdiomas);
router.get('/visitasGuiadas', getVisitasGuiadas);


//post
router.post('/obras', crearObra)
//router.post('/tiposObras', crearTipoObra)
router.post('/areas', crearArea)
router.post('/tiposAreas', crearTipoArea)
router.post('/visitasGuiadas', crearVisitaGuiada)
router.post('/idiomasPorVisitasGuiadas', crearIdiomaPorVisitaGuiada)
router.post('/turnosPorVisitasGuiadas', crearTurnoPorVisitaGuiada)
router.post('/visitasGuiadasPorAreas', crearVisitaGuiadaPorArea)
router.post('/visitante', crearVisitante)



//patch
router.patch('/obras/:IdObra', editarObra)
router.patch('/tiposObras/:IdObra', editarTipoObra)
router.patch('/areas/:IdArea', editarArea)
router.patch('/tiposAreas/:IdTipoArea', editarTipoArea)
```

En el archivo **index.js** se hace la creación del servidor

```
import express from "express";
import cors from "cors";
import Routes from "./routes/Routes.js";

const app = express();
app.use(cors());
app.use(express.json());
app.use(Routes);

app.listen(5000, ()=> console.log('Server up and running...'));
```

FRONTEND

Para esta parte se usó React, en la cual se trabajó con la metodología de trabajo de componentes y se utilizaron las tecnologías de JSX, CSS, ReactStrap y SVG . Además se implementó la tecnología de react del manejo de variables mediante Estados/Hooks. Un ejemplo con

```
const [Nombre, setNombre] = useState("");
const [IdTipoArea, setIdTipoArea] = useState("");
const [Descripcion, setDescripcion] = useState("");
```

Para hacer el mapeado de las tablas se usó el framework DataTable al cual se le añadió la función de poder filtrar datos por búsqueda.

ReactJs

Elegimos esta herramienta por el dinamismo que tiene gracias a su DOM virtual el cual aprovecha para compararlo con el DOM y realizar los cambios unicamente en los componentes que cambiaron en pantalla sin necesidad de actualizar toda la pagina y trabajando con un sistema de SPA (**S**ingle **P**age **A**pplication)

CSS

Elegimos esta herrramienta para desarrolar el sistema de cajas por el cual posicionamos los elementos en pantalla dentro de la aplicacion web sin seguir investigando otros frameworks o herramientas por falta de tiempo para el aprendizaje de las mismas.

```
    font-size: 40px;
}
.hidden{

    display: none;
    transition: all 2s linear;
}
.justbuttonbox{
    margin-bottom: 5%;
}
.buttonlogbox{
margin-left: 25%;
width:50%;

}
.mainadminbody{
    position: absolute;
    transition: all 2s linear;
    align-items: center;
    height: 100%;
    max-height:100%;
    width: 100%;
    background-color: white;
    background-size: cover;
    opacity: 100%;
```

SVG

Usamos los SVG ya que estos elementos son mejores para adaptarse teniendo una escalabilidad dinamica. Estos se definen en un archivo js y se exportan como componente asi:

```
export const AddIcon = () => {
    return (<svg xmlns="http://www.w3.org/2000/svg" class="ionicon" viewBox="0 0 450 450"><title>Add</title><path fill="none"
    stroke="currentColor" stroke-linecap="round" stroke-linejoin="round" stroke-width="10" d="M256 112v288M400 256H112"/></svg>
    )
    }
```

luego se importan desde otro componente:

```
d > src > components > JS ListarArea.js > ListarArea
import React, { useState, useEffect } from "rea
import axios from "axios";
import { Link } from "react-router-dom";
import DataTable from 'react-data-table-compone
import { CSVLink} from "react-csv";
import { AddIcon } from "./add-outline";
import  './src/css/museo.css';
import { BackIcon } from "./back-svgrepo-com";
```

y se llaman como componente al momento de usarlos en el codigo JSX:

```
<div className="headpressablebox">
        <div className="headbuttonpressablebox">
        <AddIcon></AddIcon>
        </div>
        <div className="headtitlepressablebox">
        <h1>Crear Area</h1>
        </div>
        <div className="headtextpressablebox"> <p>Crear area nueva de exposicion</p>
```

ReactStrap

Esta es la libreria de boostrap adaptada al sistema de componentes de React. se llaman los componentes para crear la interfaz de usuario y se agregan los atributos en el mismo componente:

```
import {
  Card,
  CardImg,
  CardBody,
  CardTitle,
  CardSubtitle,
  CardText,
  CardGroup,
  Button,
  Collapse,
  Navbar,
  NavbarToggler,
  NavbarBrand,
  Nav,
  NavItem,
  NavLink,
  UncontrolledDropdown,
  DropdownToggle,
  DropdownMenu,
  DropdownItem,
  NavbarText,
```

```
<NavbarBrand href="/">Museo Inteligente</NavbarBrand>
<NavbarToggler onClick={toggle} />
<Collapse isOpen={isOpen} navbar>
  <Nav className="me-auto" navbar>
    <NavItem>
      <NavLink href="/components/">Recorridos</NavLink>
    </NavItem>
    <NavItem>
      <Link to={'/login'}>
      <NavLink>
        login
      </NavLink>
      </Link>
    </NavItem>
    <UncontrolledDropdown nav inNavbar>
      <DropdownToggle nav caret>
        Options
      </DropdownToggle>
    <DropdownMenu right>
        <DropdownItem>Option 1</DropdownItem>
        <DropdownItem>Option 2</DropdownItem>
        <DropdownItem divider />
```

Inicializar proyecto

Para inicializar el back usamos el siguiente comando en la consola(previamente tiene que estar instalado el Node)

```
PS C:\xampp\htdocs\INET\backend> nodemon
```

Y nos deberia salir un resultado como este:

```
PS C:\xampp\htdocs\INET\backend> nodemon
[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server up and running...
```

Para inicializar el frontend usamos el siguiente comando:

```
PS C:\xampp\htdocs\INET\frontend> npm start
```

Y nos deberia salir el siguiente resultado:

```
You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.88.18:3000

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

webpack compiled successfully
```