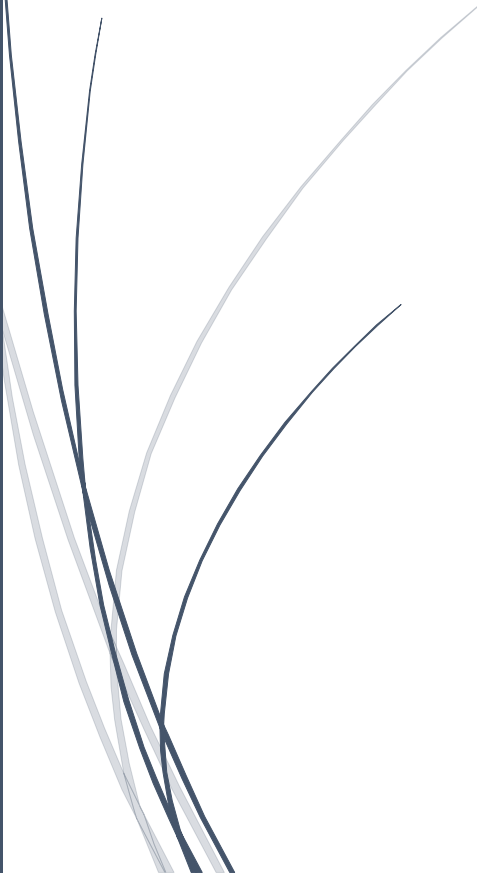


A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

21-3-2024

Manual Técnico

Aplicación Web

Several thin, curved lines in dark blue and light grey originate from the bottom left and curve upwards and to the right.

Juan Itzep 202202161
Angely García 202210483
Practicas Iniciales C

Manual Técnico

Base de Datos

La API funciona con una base de datos desarrollada en Mysql, para cumplir con los requerimientos se diseñaron 5 tablas: usuarios, cursos, publicaciones, catedráticos y comentarios.

Tabla Usuarios

En esta tabla se guarda toda la información que identifica a un usuario, esta tiene como campos el nombre del usuario de tipo char de 30 espacios, el apellido del usuario con 30 espacios, registro académico como un entero de 9 dígitos y que es la llave primaria, password para la contraseña del usuario de 20 espacios y correo de 50 espacios, todos estos campos no pueden ser nulos.

```
CREATE TABLE usuarios (  
    nombre VARCHAR(30) NOT NULL,  
    apellido VARCHAR(30) NOT NULL,  
    registroAcademico INT(9) NOT NULL PRIMARY KEY,  
    password VARCHAR(20) NOT NULL,  
    correo VARCHAR(50) NOT NULL UNIQUE  
);
```

Tabla Cursos

La tabla de cursos cuenta con 4 campos, el primero guarda el código del curso, el segundo el nombre del curso, el tercero los créditos que proporciona ganar el curso y el ultimo el catedrático que actualmente imparte el curso, todos los campos no pueden ser nulos y la llave primaria de la tabla es el código del curso(primer campo).

```
CREATE TABLE cursos (
    codigoCurso INT NOT NULL PRIMARY KEY,
    nombreCurso VARCHAR(70) NOT NULL,
    credits INT NOT NULL,
    catedratico VARCHAR(100) NOT NULL
);
```

Tabla Publicaciones

La tabla de las publicaciones guarda los datos necesarios de una publicación en 5 campos: el primer campo es el id de la publicación para tener un mejor orden en el manejo del orden de muestra de las publicaciones además es la llave primaria, el usuario que refiere al registro académico del usuario que esta realizando la publicación, la fecha que se toma automáticamente, el código del curso que refiere a alguno de los códigos de cursos que se encuentran en la tabla de cursos y el comentario que guarda el mensaje de la publicacion.

```
CREATE TABLE publicaciones (
    idPublicacion INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    usuario INT NOT NULL,
    fecha TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    codigoCurso INT NOT NULL,
    comentario TEXT,
    FOREIGN KEY (usuario) REFERENCES usuarios(registroAcademico),
    FOREIGN KEY (codigoCurso) REFERENCES cursos(codigoCurso)
);
```

Tabla Catedráticos

Esta tabla guarda la información de los catedráticos tiene como campos un id auto incremental para el manejo de los datos y que también es la llave primaria de la tabla, un segundo campo para el nombre del catedrático, un tercer campo para el apellido del catedrático y el código del curso que imparten que refiere a la tabla de cursos y su campo de código de cursos.

```
CREATE TABLE catedraticos (
  idCatedratico INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(30) NOT NULL,
  apellido VARCHAR(30) NOT NULL,
  codigoCurso INT NOT NULL,
  FOREIGN KEY (codigoCurso) REFERENCES cursos(codigoCurso)
);
```

Tabla Comentarios

La tabla de comentarios de una publicación funciona con 5 campos: el id de cada comentario al igual los anteriores se usa para el manejo de los datos y es la llave primaria, el usuario que refiere al registro académico del mismo que realiza el comentario, publicación es un entero que refiere al id de la publicación a la que pertenece el comentario, comentario es el texto para guardar y la fecha que se toma automáticamente.

```
CREATE TABLE comentarios (
  idComentario INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  usuario INT NOT NULL,
  publicacion INT NOT NULL,
  comentario TEXT,
  fecha TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (usuario) REFERENCES usuarios(registroAcademico),
  FOREIGN KEY (publicacion) REFERENCES publicaciones(idPublicacion)
);
```

Rest API

Se encuentra en la carpeta “Backend” y todas las funciones estan en “Server.js”

Dependencias

Para el desarrollo y funcionamiento de la REST API en el proyecto se necesitan las siguientes dependencias instaladas: Node js, express, mysql y cors.

Conexión a Base de Datos

Es importante la conexión a la base de datos con el método mysql Creation que se almacena en una constante.

```
const db = mysql.createConnection({  
  host: "localhost",  
  user: "root",  
  password: "",  
  database: "appwebdb"  
});
```

MiddleWares

Express utilizado para facilitar el enrutamiento de los endpoints o Https, Cors para establecer desde donde se puede llegar a un end point, y el ultimo middleware para facilitar el manejo de los datos obtenidos o para guardar en las bases de datos.

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

End Points

El end point “/registro” es un método de tipo post, este recibe el nombre, apellido registro académico, contraseña y correo en formato json, lo traslada a variables y valida que los campos no estén vacíos, si no están vacíos se crea una constante con un string con el formato query para agregar un nuevo usuario a la tabla de usuarios y un arreglo con los datos de los campos de la tabla, posteriormente se ejecuta la instrucción de la base de datos “.query” con el string y el array para guardar los datos y finalmente se valida que no haya algún error al realizar la instrucción.

```

app.post('/registro', (req, res) => {
  const { nombre, apellido, registroAcademico, password, correo } = req.body;

  if (!nombre || !apellido || !registroAcademico || !password || !correo) {
    return res.status(400).json({ error: "Por favor, completa todos los campos requeridos" });
  }

  const sql = "INSERT INTO usuarios (nombre, apellido, registroAcademico, password, correo) VALUES (?, ?, ?, ?, ?)";
  const values = [nombre, apellido, registroAcademico, password, correo];

  db.query(sql, values, (err, result) => {
    if(err) {
      console.log("Error inserting data:", err);
      return res.status(500).json({ error: "Error al guardar en la base de datos" });
    }
    console.log("Data inserted successfully");
    return res.json({ message: "Datos guardados en la base de datos" });
  });
});

```

El end point “/” este método se encarga de validar un inicio de sesión, recibiendo el registro académico y la contraseña que guarda en un array, anteriormente crea un String con la instrucción query para buscar dentro de la tabla de usuarios que el registro académico y contraseña obtenidos correspondan, ejecuta el método “.query” y valida si hubo un error, también si los datos coinciden la base de datos, regresando una respuesta succes o failed.

```

app.post('/', (req, res) => {
  const sql = "SELECT * FROM usuarios WHERE registroAcademico = ? AND password = ?";
  const values = [
    req.body.registroAcademico,
    req.body.password
  ]
  console.log('Datos de la solicitud:', req.body);

  db.query(sql, [req.body.registroAcademico, req.body.password], (err, data) => {
    if(err) {
      console.log("Error al ejecutar la consulta SQL:", err);
      return res.json("Error");
    }
    console.log('Datos obtenidos de la base de datos:', data);

    if(data.length > 0) {
      console.log("Inicio de sesión exitoso");
      return res.json("Success");
    } else {
      console.log("Inicio de sesión fallido");
      return res.json("Failed");
    }
  });
});

```

El end point “/cursos” es un metodo de tipo get que se encarga de crear un String con la query para obtener el código y el nombre de los cursos guardados en la tabla cursos. Se realiza el método query de db y se regresan los datos obtenidos si no hubieron problemas.

```
app.get('/cursos', (req, res) => {  
  const sql = "SELECT codigoCurso, nombreCurso FROM cursos";  
  
  db.query(sql, (err, data) => {  
    if(err) {  
      console.log("Error al ejecutar la consulta SQL:", err);  
      return res.status(500).json({ error: "Error al recuperar los cursos de la base de datos" });  
    }  
    console.log('Cursos obtenidos de la base de datos:', data);  
    return res.json(data);  
  });  
});
```

El end point “/catedráticos/:codigoCurso”, este metodo de tipo get recibe el código de un curso, que se concatena al string de la query, la instrucción query obtiene el id del catedrático, la concatenación del nombre y apellidos del catedrático de la tabla catedráticos buscando con el código del curso recibido, posteriormente se realiza la instrucción query y si se obtienen datos se responde con el json de esos datos obtenidos.

```
app.get('/catedraticos/:codigoCurso', (req, res) => {  
  const codigoCurso = req.params.codigoCurso;  
  const sql = "SELECT idCatedratico, CONCAT(nombre, ' ', apellido) AS nombreCompleto FROM catedraticos WHERE codigoCurso = ?";  
  
  db.query(sql, [codigoCurso], (err, data) => {  
    if(err) {  
      console.log("Error al ejecutar la consulta SQL:", err);  
      return res.status(500).json({ error: "Error al recuperar los catedráticos de la base de datos" });  
    }  
    console.log('Catedráticos obtenidos de la base de datos:', data);  
    return res.json(data);  
  });  
});
```

El end point “/publicaciones” es un método de tipo Post que se encarga de guardar las publicaciones en la tabla de publicaciones de la base de datos, al igual que los métodos anteriores crea un String con una instrucción query para agregar los campos correspondientes a la tabla de

publicaciones, se crea un array con los datos de la publicación y se realiza el método .query para la base de datos.

```
app.post('/publicaciones', (req, res) => {
  console.log('Solicitud POST recibida en /publicaciones:', req.body);
  const { registroAcademico, codigoCurso, idCatedratico, comentario } = req.body;

  const sql = "INSERT INTO publicaciones (registroAcademico, codigoCurso, idCatedratico, comentario) VALUES (?, ?, ?, ?)";
  const values = [registroAcademico, codigoCurso, idCatedratico, comentario];

  db.query(sql, values, (err, result) => {
    if(err) {
      console.error("Error al insertar en la base de datos:", err);
      return res.status(500).json({ error: "Error al guardar en la base de datos" });
    }
    console.log("Publicación insertada correctamente en la base de datos");
    return res.json({ message: "Publicación guardada en la base de datos" });
  });
});
```

Frontend

Se desarrolló con React, css y axios, react-router-dom.

Enrutamiento

Enrutamiento general de las rutas a las que puede llegar.

```
return (
  <BrowserRouter>
    <Routes>
      <Route path="/" element={<Login />} />
      <Route path="/login" element={<Login />} />
      <Route path="/registro" element={<Registro />} />
      <Route path="/inicio" element={<Inicio />} />
      <Route path="/password_reset" element={<Password />} />
      <Route path="/perfil" element={<Perfil />} />
      <Route path="/publicaciones" element={<Publicaciones />} />
    </Routes>
  </BrowserRouter>
);
```


Registro

Creación de estado inicial con useState para el formato json que se pasará al backend al momento de querer realizar un registro de usuario, constante Navigate con useNavigate para redirigir a otra ruta.

Función Handlesubmit que obtiene el ultimo estado de values y por medio de axios consume el end point “/registro” y luego de registrar satisfactoriamente se dirige a “/” si todos los posibles errores de input ya están corregidos validación que se hace por medio de Signup Validation

```
const [values, setValues] = useState({
  nombre: '',
  apellido: '',
  registroAcademico: '',
  password: '',
  correo: ''
})

const navigate = useNavigate();
const [errors, setErrors] = useState({})

useEffect(() => {
  if (Object.values(errors).every(error => error === "")) {
    axios.post('http://localhost:8000/registro', values)
      .then(res => console.log(res))
      .catch(err => console.log(err));
  }
}, [errors, values]);

const handleInput = (event) => {
  setValues(prev => ({ ...prev, [event.target.name]: event.target.value }))
}

const handleSubmit = (event) => {
  event.preventDefault();
  setErrors(SignupValidation(values));
  if(errors.nombre === "" && errors.apellido === "" && errors.registroAcademico === "" && errors.password === "" && errors.correo === "") {
    axios.post('http://localhost:8000/registro', values)
      .then(res => {
        navigate('/');
      })
      .catch(err => console.log(err));
  }
}
```

Registro Validation

Validacion para que todos los campos estén llenos correctamente por medio de la función signup Validation

```

    if (values.nombre === "") {
        error.nombre = "Por favor, llena este campo"
    }
    else {
        error.nombre = ""
    }

    if (values.apellido === "") {
        error.apellido = "Por favor, llena este campo"
    }
    else if (!apellido_pattern.test(values.apellido)) {
        error.apellido = "Registro académico no encontrado"
    } else {
        error.apellido = ""
    }

    if (values.registroAcademico === "") {
        error.registroAcademico = "Por favor, llena este campo"
    }
    else if (!registroAcademico_pattern.test(values.registroAcademico)) {
        error.registroAcademico = "Registro académico no encontrado"
    } else {
        error.registroAcademico = ""
    }

    if(values.password === "") {
        error.password = "Por favor, llena este campo"
    }
    else {
        error.password = ""
    }

    if(values.correo === "") {
        error.correo = "Por favor, llena este campo"
    }
    else if (!correo_pattern.test(values.correo)) {
        error.correo = "Contraseña incorrecta"
    } else {
        error.correo = ""
    }

    return error;

```

Login

Dos estados useState para el registro académico y la contraseña, constante navigate igual a useNavigate.

La función `handleSubmit`, consume el endpoint `"/"`, enviando los datos de los últimos estados obtenidos, devolviendo una respuesta dependiendo si los datos del usuario son correctos, si son correctos se redirige a `"/inicio"` con `navigate`.

```
function Login() {
  const [registroAcademico, setRegistroAcademico] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  function handleSubmit(event) {
    event.preventDefault();
    axios.post('http://localhost:8000/', { registroAcademico, password })
      .then(res => {
        console.log('Respuesta del servidor:', res.data);
        if (res.data === "Success") {
          console.log("Inicio de sesión exitoso, redirigiendo al usuario...");
          localStorage.setItem('registroAcademico', registroAcademico);
          navigate('/inicio');
        } else {
          console.log("Inicio de sesión fallido, mostrando mensaje de alerta al usuario...");
          alert("No se encuentra registrado o los datos son incorrectos");
        }
      })
      .catch(error => {
        console.error('Error al enviar la solicitud:', error);
      });
  }
}
```

Inicio

Estado de `useState` para almacenar los datos de usuario que accedió a inicio, `UseEffect` obtiene el registro académico del usuario, la constante `Verperfil` es un metodo que con `nav`, otra constante igual a `useNavigate` redirige a `"/perfil"`

```

const nav = useNavigate();
const [registroAcademico, setRegistroAcademico] = useState('');

useEffect(() => {
  axios.get('http://localhost:8000/registro')
    .then(response => {
      console.log('Registro académico obtenido:', response.data);
      setRegistroAcademico(response.data.registroAcademico);
    })
    .catch(error => {
      console.error('Error al obtener el registro académico del usuario:', error);
    });
}, []);

const verPerfil = () => {
  nav('/perfil');
};

```

Publicaciones

Estados para el curso y catedráticos seleccionados de la lista, para los catedráticos y cursos que se obtengan de la base de datos para llenar las listas, para el contenido de la publicación(comentarios, curso, etc) y el registro académico del usuario que esta realizando la publicación.

useEffect consume el end point “/cursos” y recibe toda la información de los cursos de la tabla de datos para pasarlo a una lista desplegable.

handleCursoChange consume el endpoint “/catedraticos” por medio de axios concatenado con el código del curso seleccionado y así buscar obtener la información del catedrático.

```

function Publicaciones() {
  const [selectedCurso, setSelectedCurso] = useState('');
  const [selectedCatedratico, setSelectedCatedratico] = useState('');
  const [catedraticos, setCatedraticos] = useState([]);
  const [cursos, setCursos] = useState([]);
  const [contenidoPublicacion, setContenidoPublicacion] = useState('');
  const [registroAcademico, setRegistroAcademico] = useState(''); // Utilizamos registroAcademico en lugar de usuario

  useEffect(() => {
    console.log('Fetching cursos...');
    axios.get('http://localhost:8000/cursos')
      .then(response => {
        console.log('Cursos recibidos:', response.data);
        setCursos(response.data);
      })
      .catch(error => {
        console.error('Error al obtener los cursos:', error);
      });

    // obtener el regAca del usuario del almacenamiento local
    const storedRegistroAcademico = localStorage.getItem('registroAcademico');
    if (storedRegistroAcademico) {
      setRegistroAcademico(storedRegistroAcademico);
    }
  }, []);

  const handleCursoChange = (event) => {
    const codigoCurso = event.target.value;
    setSelectedCurso(codigoCurso);

    axios.get(`http://localhost:8000/catedraticos/${codigoCurso}`)
      .then(response => {
        console.log('Catedráticos recibidos:', response.data);
        setCatedraticos(response.data);
        setSelectedCatedratico('');
      })
      .catch(error => {
        console.error('Error al obtener los catedráticos:', error);
      });
  };
}

```

HandleSubmit obtiene los valores de los estados necesarios para la publicación y crea un objeto json para pasárselo al axios que consume el end point “/publicaciones” para agregar una nueva publicación a la base de datos.

```

const handleSubmit = (e) => {
  e.preventDefault();

  const nuevaPublicacion = {
    registroAcademico: registroAcademico,
    codigoCurso: selectedCurso,
    idCatedratico: selectedCatedratico,
    comentario: contenidoPublicacion
  };

  axios.post('http://localhost:8000/publicaciones', nuevaPublicacion)
    .then(response => {
      console.log('Publicación creada exitosamente:', response.data);
      setSelectedCurso('');
      setSelectedCatedratico('');
      setContenidoPublicacion('');
    })
    .catch(error => {
      console.error('Error al crear la publicación:', error);
    });
};

```

Perfil

Función exportada que recibe un usuario como estado de tipo json con la información de un usuario y la muestra en una división de HTML.

```
import styles from './Perfil.module.css';

function Perfil({ usuario }) {
  return (
    <div className={styles.profileContainer}>
      <h2>Perfil del Usuario</h2>
      <p>Nombre: {usuario.nombre}</p>
      <p>Apellido: {usuario.apellido}</p>
      <p>Registro Académico: {usuario.registroAcademico}</p>
      <p>Correo: {usuario.correo}</p>
    </div>
  );
}

export default Perfil;
```