



Universidad Autónoma de Sinaloa
Facultad de Informática Culiacán
Licenciatura en Informática



Proyecto:

Conexión de Backend y Frontend

Materia:

Programación web del lado del servidor

Docente:

José Manuel Cazarez Alderete

Alumna:

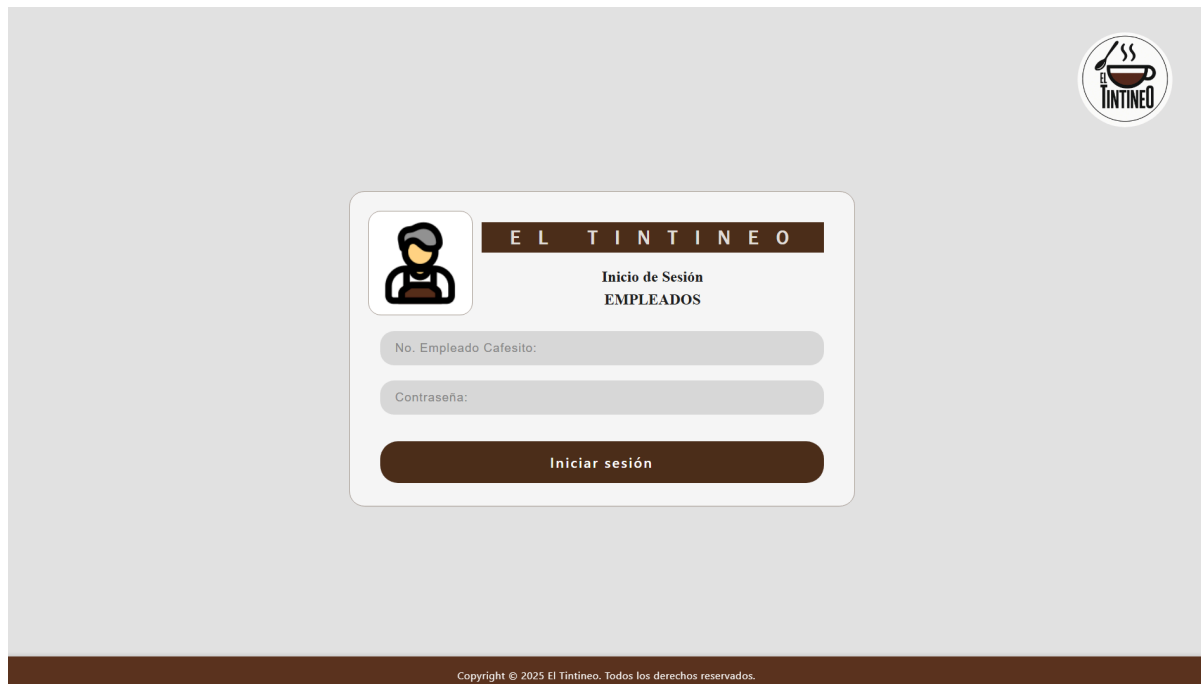
Calderón Beltrán Vanessa Noemí

Grado y Grupo:

2-1

Culiacán, Sinaloa, 25 Mayo del 2025

● App.jsx:



```
function App() {  
  const [empleado, setEmpleado] = useState('');  
  const [password, setPassword] = useState('');  
  const [error, setError] = useState('');  
  const navigate = useNavigate();  
}
```

◆ Hooks:

useState:

- **Empleado:** almacena el nombre de usuario o identificador de empleado, inicializado como una cadena vacía.
- **Password:** Almacena la contraseña, inicializado como una cadena vacía.
- **Error:** Almacena los mensajes de error generados, se inicializa como cadena vacía.

useNavigate:

- **Maps:** Proporcionado por react-router-dom, es una función que permite la navegación entre diferentes rutas o vistas dentro de la aplicación.

❖ API login (POST):

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setError('');
  try {
    const response = await fetch('http://localhost:3000/api/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        no_Empleado: empleado,
        contraseña: password
      })
    });
    const data = await response.json();
    if (response.ok && data.success) {
      navigate('/paginaInicio');
    } else {
      setError(data.message || 'Credenciales incorrectas');
    }
  } catch (err) {
    setError('Error de conexión con el servidor');
  }
}
```

- **e.preventDefault():** Evita que la página se recargue al enviar el formulario.
- **setError("")**: Limpia cualquier mensaje de error anterior.
- **fetch()**: Hace una petición POST al backend (<http://localhost:3000/api/login>), enviando el correo y la contraseña.
- **await response.json()**: Espera la respuesta del servidor y la convierte en un objeto JS.
- **(response.ok y data.success)**: Redirige al usuario a /paginaInicio.

◆HTML:

```
<div className="login-title">
  <span className="brand">E L &nbsp; T I N T I N E O</span>
  <div className="login-subtitle">
    <b>Inicio de Sesión</b>
    <br />
    <b>EMPLEADOS</b>
  </div>
</div>
```

Se muestra el título de la cafetería y los textos que aparecen en la parte superior del login.

```
<form className="login-form" onSubmit={handleSubmit}>
  <input
    type="text"
    placeholder="No. Empleado Cafesito:"
    className="login-input"
    value={empleado}
    onChange={e => setEmpleado(e.target.value)}
  />
  <input
    type="password"
    placeholder="Contraseña:"
    className="login-input"
    value={password}
    onChange={e => setPassword(e.target.value)}
  />
  <button type="submit" className="login-btn">
    Iniciar sesión
  </button>
  {error && <div style={{ color: 'red', marginTop: 10 }}>{error}</div>}
</form>
```

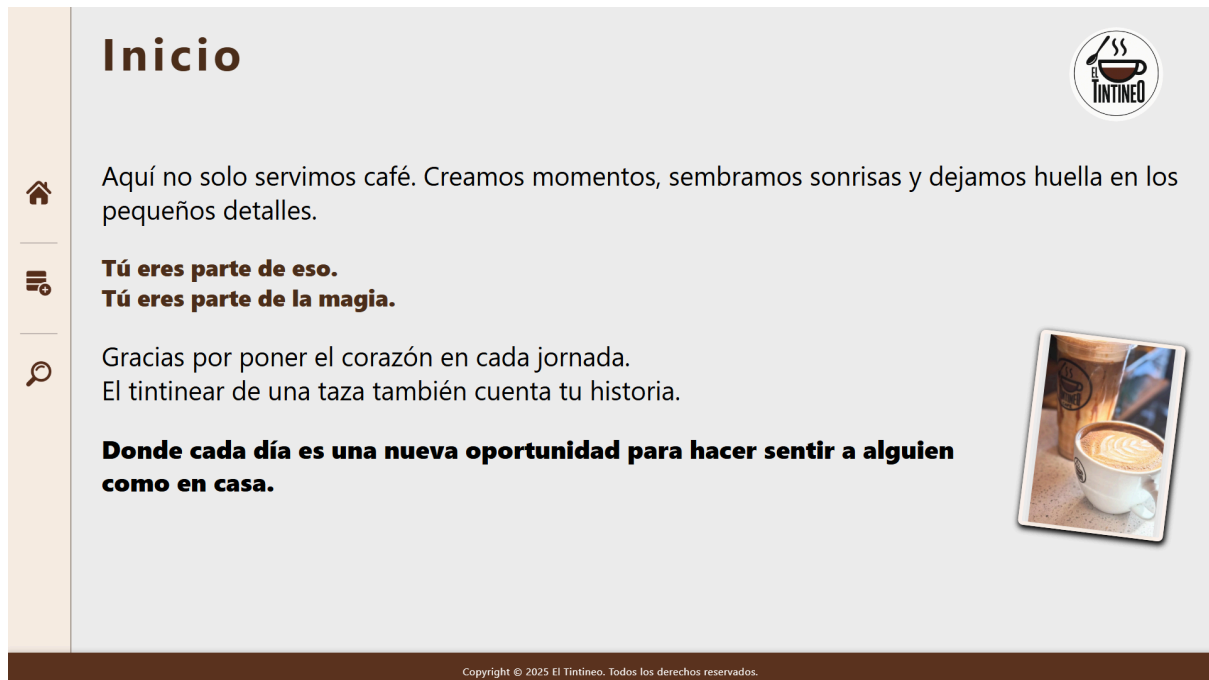
- **Form:** Engloba el formulario, dentro de éste se encuentran los inputs y el botón de iniciar.
- **Inputs:** Dentro de estos se escribirá el número de empleado y la contraseña.
- **e.target.value:** Almacena el valor escrito dentro de los inputs (lo que se envía a la api)
- **Button:** Es el botón con el que se envía la petición a la api

✦ Footer:

```
<footer className="login-footer">  
| Copyright © 2025 El Tintineo. Todos los derechos reservados.  
</footer>
```

- **Footer:** Es la parte inferior de la página, aquí se muestra el copyright.

● paginaInicio.jsx:



```
import './paginaInicio.css';
import logoCafe from './assets/logodeCafe.png';
import cafeFoto from './assets/cafeimagen.jpg';
import iconInicio from './assets/inicio.png';
import iconAgregar from './assets/agregar.png';
import iconBuscar from './assets/buscar.png';
import { useNavigate } from 'react-router-dom';
```

◆ Imports:

- import './paginaInicio.css': Importa los estilos para la página.
- logoCafe from './assets/logodeCafe.png': Importa la imagen del logo desde la carpeta "assets".
- cafeFoto from './assets/cafeimagen.jpg': Importa la imagen del café desde la carpeta "assets".
- iconInicio from './assets/inicio.png': Importa la imagen (casita/home) que se utilizará como botón para navegar a paginaInicio.
- iconAgregar from './assets/agregar.png': Importa la imagen (base de datos/+) que se utilizará como botón para navegar a paginaAgregar.
- iconBuscar from './assets/buscar.png': Importa la imagen (lupa) que se utilizará como botón para navegar a paginaBuscar.

◆HTML:

```
<aside className="sidebar">
  <div style={{ marginTop: "250px" }}></div>
  <div className="sidebar-icon">
    <img src={iconInicio} alt="Inicio" style={{ width: 40, height: 40 }} />
  </div>
  <div className="sidebar-divider"></div>
  <div
    className="sidebar-icon"
    onClick={() => navigate('/paginaAgregar')}
    style={{ cursor: 'pointer' }}
  >
    <img src={iconAgregar} alt="Agregar" style={{ width: 40, height: 40 }} />
  </div>
  <div className="sidebar-divider"></div>
  <div className="sidebar-icon"
    onClick={() => navigate('/paginaBuscar')}
    style={{ cursor: 'pointer' }}
  >
    <img src={iconBuscar} alt="Buscar" style={{ width: 40, height: 40 }} />
  </div>
</aside>
```

- **Aside:** Coloca todo en un lateral de la pantalla, en este caso la sidebar que contiene los botones de navegación.
- **onClick:** Lee el evento “click”, cuando se clickea, realiza la acción, en este caso es redireccionar a las páginas Agregar y Buscar

```
<main className="inicio-main">
  <div className="inicio-header">
    <h1 className="inicio-title">Inicio</h1>
    <img src={logoCafe} alt="El Tintineo" className="inicio-logo" />
  </div>
  <div className="inicio-content">
    <p>
      Aquí no solo servimos café. Creamos momentos, sembramos sonrisas y dejamos huella en los pequeños detalles.
    </p>
    <p className="inicio-destacado">
      <b>Tú eres parte de eso.</b><br />Tú eres parte de la magia.</p>
    <div className="inicio-foto">
      <img src={cafeFoto} alt="Café" />
    </div>
    <p>
      Gracias por poner el corazón en cada jornada.<br />
      El tintinear de una taza también cuenta tu historia.
    </p>
    <p className="inicio-final">
      <b>Donde cada día es una nueva oportunidad para hacer sentir a alguien como en casa.</b>
    </p>
  </div>
</main>
```

- **Main:** Parte principal de la página, dentro de ésta se encuentran los textos.

● paginaAgregar.jsx:

The screenshot shows a web form titled "Nueva Reservación" for "El Tintineo". The form is located on a light gray background with a dark brown sidebar on the left containing navigation icons (home, list, search). The form fields are as follows:

- Nombre del cliente: (single-line text input)
- Num. Personas: (single-line text input)
- No. Mesa: (single-line text input)
- Fecha de reservación: (single-line text input)
- Hora de reservación: (single-line text input)
- Información de contacto: (single-line text input)

Below the input fields is a dark brown button labeled "Agendar". At the bottom of the page, a small copyright notice reads: "Copyright © 2025 El Tintineo. Todos los derechos reservados."

```
export default function PaginaAgregar() {  
  const navigate = useNavigate();  
  const [formData, setFormData] = useState({  
    nombreCliente: '',  
    numeroPersonas: '',  
    numeroMesa: '',  
    fechaReservacion: '',  
    horaReservacion: '',  
    informacionContacto: '',  
  });  
}
```

◆ **PaginaAgregar()**: Inicializa un formulario con los datos que se tienen que guardar en la base de datos, pero con contenido vacío (").

❖ API CRUD (POST):

```
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const response = await fetch('http://localhost:3000/create/CRUD', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(formData),
    });

    if (response.ok) {
      alert('Reservación creada exitosamente');
      setFormData({
        nombreCliente: '',
        numeroPersonas: '',
        numeroMesa: '',
        fechaReservacion: '',
        horaReservacion: '',
        informacionContacto: '',
      });
    } else {
      alert('Error al crear la reservación');
    }
  } catch (error) {
    console.error('Error:', error);
    alert('Error al conectar con el servidor');
  }
};
```

- **Fetch:** Llama a la API y le envía lo que esté almacenado en formData

◆HTML:

```
<input
  className="agregar-input-reserva"
  type="text"
  name="nombreCliente"
  placeholder="Nombre del cliente:"
  value={formData.nombreCliente}
  onChange={handleChange}
/>
```

- Almacena el nombre del cliente.

```
<div className="agregar-form-row">
  <input
    className="agregar-input-reserva"
    type="text"
    name="numeroPersonas"
    placeholder="Num. Personas"
    value={formData.numeroPersonas}
    onChange={handleChange}
  />
```

- Almacena el número de clientes que asistirá a la reservación.

```
<input
  className="agregar-input-reserva"
  type="text"
  name="numeroMesa"
  placeholder="No. Mesa:"
  value={formData.numeroMesa}
  onChange={handleChange}
/>
```

- Almacena el número de la mesa en la que se hará la reservación.

```
<input
  className="agregar-input-reserva"
  type="text"
  name="fechaReservacion"
  placeholder="Fecha de reservación:"
  value={formData.fechaReservacion}
  onChange={handleChange}
/>
```

- Almacena la fecha en la que se realizó la reservación.

```
<input
  className="agregar-input-reserva"
  type="text"
  name="horaReservacion"
  placeholder="Hora de reservación:"
  value={formData.horaReservacion}
  onChange={handleChange}
/>
```




- Almacena la hora en la que se hizo la reservación.

```
<input
  className="agregar-input-reserva"
  type="text"
  name="informacionContacto"
  placeholder="Información de contacto:"
  value={formData.informacionContacto}
  onChange={handleChange}
/>
```

- Almacena la información de contacto de la persona que realizó la reservación, puede ser número de celular o correo electrónico.


● [paginaBuscar.jsx](#):

Antes de la búsqueda:



Buscar Reservación por ID

Ingresar el folio...



Copyright © 2025 El Tintineo. Todos los derechos reservados.

Después de realizar la búsqueda:



Buscar Reservación por ID

9



Nombre del cliente:
vanessa

Num. Personas:
4

Fecha de reservación:
27/06/2025

Información de contacto:
6677889912

No. Mesa:
9

Hora de reservación:
4:30pm

 Editar

 Eliminar

Copyright © 2025 El Tintineo. Todos los derechos reservados.

```
export default function PaginaBuscar() {
  const navigate = useNavigate();
  const [folio, setFolio] = useState('');
  const [detalle, setDetalle] = useState(null);
  const [mensaje, setMensaje] = useState('');
  const [editando, setEditando] = useState(false);
  const [formEdit, setFormEdit] = useState({});

```

❖ **PaginaBuscar()**: se usa para buscar información por folio, mostrar detalles, y permitir editar esa información si es necesario.

```
const handleBuscar = async (e) => {
  e.preventDefault();
  setMensaje('');
  setDetalle(null);
  if (!folio) {
    setMensaje('Ingresa el folio');
    return;
  }
  try {
    const response = await fetch(`http://localhost:3000/read/CRUD/${folio}`);
    if (response.ok) {
      const data = await response.json();
      if (!data || Object.keys(data).length === 0) {
        setMensaje('No se encontró la reservación');
      } else {
        setDetalle(data);
      }
    } else {
      setMensaje('No se encontró la reservación');
    }
  } catch {
    setMensaje('Error de conexión');
  }
};

```

❖ **API CRUD (GET)**: Esta parte envía el folio a la base de datos y hace una búsqueda de las reservaciones, en caso de que se encuentre, la muestra en “data” y actualiza los valores de los inputs con los valores recibidos, si no la encuentra, en caso de que sea por no existir, responde un “No se encontró la reservación” y en caso de que suceda algún error de otro tipo, responde un “Error de conexión”

```
const handleEditar = () => {
  setEditando(true);
  setFormEdit(detalle);
};
```

✦ **handleEditar**: sirve para activar el modo edición en el componente.

```
const handleInputEdit = (e) => {
  const { name, value } = e.target;
  setFormEdit({ ...formEdit, [name]: value });
};
```

✦ **handleInputEdit**: maneja los cambios en los inputs del formulario.

```
const handleGuardar = async (e) => {
  e.preventDefault();
  try {
    const response = await fetch(`http://localhost:3000/update/CRUD/${folio}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(formEdit),
    });
    if (response.ok) {
      setDetalle(formEdit);
      setEditando(false);
      setMensaje('Reservación actualizada');
    } else {
      setMensaje('Error al actualizar');
    }
  } catch {
    setMensaje('Error de conexión');
  }
};
```

✦ **API CRUD (PUT)**: Envía los datos actualizados de los inputs hacia la API PUT que es el UPDATE, si el folio coincide, la actualización se realizará correctamente, en caso de que haya un error, éste se mostrará como “Error al actualizar”.

```
const handleEliminar = async () => {
  if (!window.confirm('¿Seguro que deseas eliminar esta reservación?')) return;
  try {
    const response = await fetch(`http://localhost:3000/delete/CRUD/${folio}`, {
      method: 'DELETE',
    });
    if (response.ok) {
      setDetalle(null);
      setFolio('');
      setMensaje('Reservación eliminada');
    } else {
      setMensaje('Error al eliminar');
    }
  } catch {
    setMensaje('Error de conexión');
  }
};
```

◆ **API CRUD (DELETE):** Primero se mostrará un aviso de confirmación, dónde se preguntará si se quiere eliminar esa reservación, en caso de aceptar, se enviará el folio a la API DELETE, la cual eliminará el registro de la base de datos y enviará un mensaje de “Reservación eliminada”, en caso de un error, se mostrará “Error al eliminar”.