Propuesta página web:
Facturación de clientes Billing Software Automation

Héctor Daniel Parra Martínez

Estudiante de Ingeniería de

Software

Asignatura.
Desarrollo de aplicaciones web II

Presentado a.
Ing. Joaquin Sanchez

Corporación Universitaria Iberoamericana
2025

Tabla de contenido.

Objetivos.

Objetivos generales.

Modelar la aplicación expuesta en la web demostrando la aplicabilidad de los conceptos realizados en clase los cuales a su vez demuestran el comportamiento Back End & Front End.

Objetivos Específicos.

- Demostrar los modelos de conectividad bajo arquitecturas de envió & respuesta.
- Ilustrar líneas de código resumidas donde se demuestra la aplicabilidad de los conceptos.

Introducción

En el contexto de las aplicaciones web se procede a generar nuevos modelos día a día por eso es inmediato conocer los protocolos de conectividad basándonos en modelos de respuesta appi los cuales nos permite experimentar diferentes alternativas para diseñar la página web.

Pagina web billing factoring

REST (Representational State Transfer):

- Se utiliza para diseñar **APIs web** que permiten a diferentes aplicaciones comunicarse entre sí mediante HTTP.
- Cada recurso (ejemplo: facturas, clientes, productos) se expone con una URL y se manipula con métodos estándar:

- **GET** → obtener datos
- **POST** → crear
- **PUT/PATCH** → actualizar
- **DELETE** → eliminar

Swagger (OpenAPI):

- Es una herramienta que se usa para **documentar y probar APIs REST**.

Ejemplo de código resumido:

```
// Endpoint REST en la app de Billing
app.get('/invoices', (req, res) => res.json([{ id: 1, number: 'FAC-001', amount: 1500 }]));

// Documentación Swagger montada en /api-docs
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument));
```

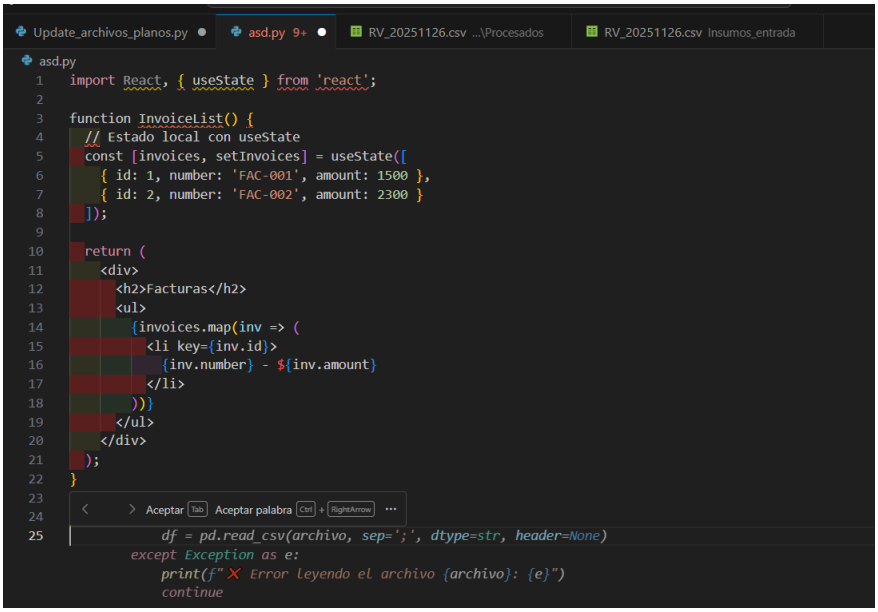
Consulta hecha para enviar una solicitud en forma de endpoint para comprobar si la factura FAC-001 existe, en este contexto nos tendría que dar una respuesta.

ReactJS es una **biblioteca de JavaScript** creada por Facebook para construir **interfaces de usuario** basadas en componentes. Su idea principal es que la interfaz se divide en piezas llamadas **componentes**, cada uno con su propio estado y lógica.

Características clave:

- **Declarativo:** describes cómo quieres que se vea la UI y React se encarga de actualizarla.
- **Basado en componentes:** cada parte de la interfaz es un bloque reutilizable.
- **Virtual DOM:** optimiza el rendimiento al actualizar solo lo necesario.
- **Unidireccional:** los datos fluyen en una sola dirección, lo que facilita el control del estado.

Ejemplo sencillo de ReactJS



Qué hace este ejemplo?

- Usa **ReactJS** para renderizar una lista de facturas.
- useState guarda el estado de las facturas.
- map() recorre el arreglo y genera elementos <li> dinámicamente.
- Si se agregan nuevas facturas al estado, la interfaz se actualiza automáticamente.

Como vemos esto nos puede ayudar de manera dinamica para resolver problemas como facturas dinamicas.

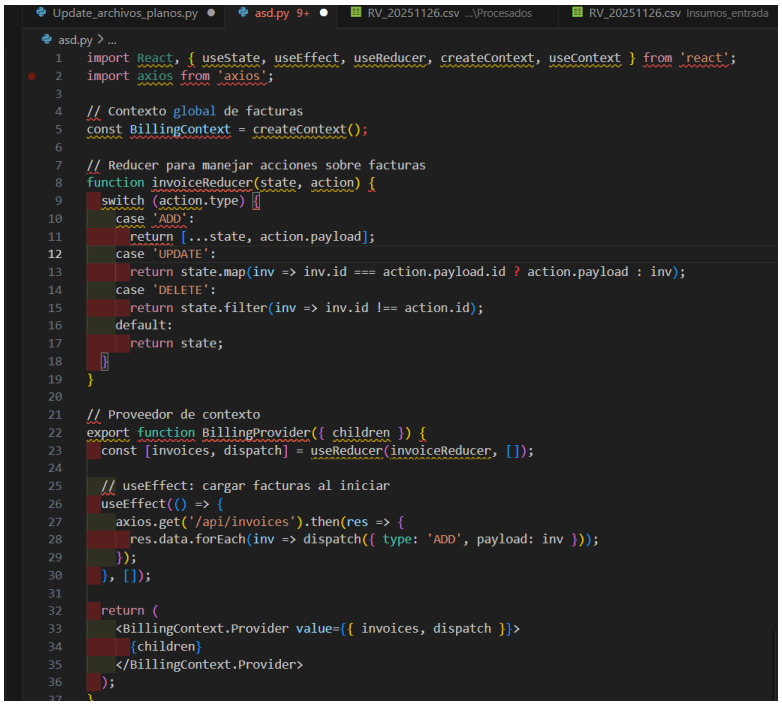
Ahora se dara el conexto de Hooks.

useState: Maneja el estado local de un componente. ☞ Ejemplo: guardar el valor de un campo de factura (cliente, monto).

useContext: Permite acceder a datos globales definidos en un **Context API** sin necesidad de pasar props manualmente. ☞ Ejemplo: acceder a la lista de facturas desde cualquier componente.

useEffect: Ejecuta efectos secundarios en el ciclo de vida del componente (al montar, actualizar o desmontar). ☞ Ejemplo: cargar facturas desde la API cuando se abre la pantalla.

useReducer: Maneja estados complejos con un reducer (similar a Redux pero integrado en React). ☞ Ejemplo: gestionar acciones de facturas (agregar, actualizar, eliminar).



- **useState:** controla los campos del formulario de nueva factura.
- **useContext:** accede al contexto global de facturas.
- **useEffect:** carga facturas desde la API al iniciar.
- **useReducer:** maneja las acciones de agregar, actualizar y eliminar facturas.

Axios es una librería de JavaScript que se usa para realizar **peticiones HTTP** desde el navegador o Node.js.. Se utiliza mucho en aplicaciones React porque:

- Simplifica las llamadas a APIs REST.
- Maneja automáticamente **promesas** (async/await).
- Permite configurar **headers**, interceptores y manejo de errores.
- Soporta **JSON** de forma nativa.

En una aplicación de **facturación (Billing)**, Axios sirve para comunicar el **frontend en React** con el **backend REST documentado con Swagger**.

```jsx
import React, { useEffect, useState } from 'react';
import axios from 'axios';

function InvoiceList() {
  const [invoices, setInvoices] = useState([]);

  // useEffect: cargar facturas al montar el componente
  useEffect(() => {
    axios.get('https://api.miempresa.com/invoices')
      .then(response => {
        setInvoices(response.data); // guardar facturas en el estado
      })
      .catch(error => {
        console.error('Error al cargar facturas:', error);
      });
  }, []);

  return (
    <div>
      <h2>Facturas</h2>
      <ul>
        {invoices.map(inv => (
          <li key={inv.id}>
            {inv.number} - ${inv.amount}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default InvoiceList;
```

Axios se usa para garantizar la transaccionalidad enviando peticiones en tiempo real y guarda las respuestas, si tenemos error en el servidor o conectividad lo captura en la variable .catch()

Rutas y navegación (React Router)

En una aplicación React, las **rutas** permiten organizar la interfaz en diferentes pantallas según la URL. Con **react-router-dom** puedes definir rutas como:

- /invoices → lista de facturas
- /invoices/new → formulario para crear una nueva factura
- /invoices/:id → detalle de una factura específica

Ejemplo de código:

```jsx
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import InvoiceList from './InvoiceList';
import NewInvoiceForm from './NewInvoiceForm';
import InvoiceDetail from './InvoiceDetail';

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/invoices">Facturas</Link>
        <Link to="/invoices/new">Nueva factura</Link>
      </nav>
      <Routes>
        <Route path="/invoices" element={<InvoiceList />} />
        <Route path="/invoices/new" element={<NewInvoiceForm />} />
        <Route path="/invoices/:id" element={<InvoiceDetail />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

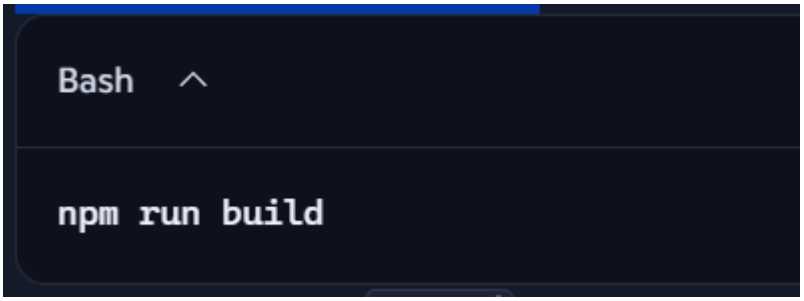
Esto permite que el usuario navegue entre pantallas sin recargar la página (SPA).

Despliegue

El **despliegue** es el proceso de publicar tu aplicación para que sea accesible en la web.
En React, normalmente se hace un **build optimizado** y se sube a un servicio de hosting.

Pasos típicos:

1. **Generar build de producción:**

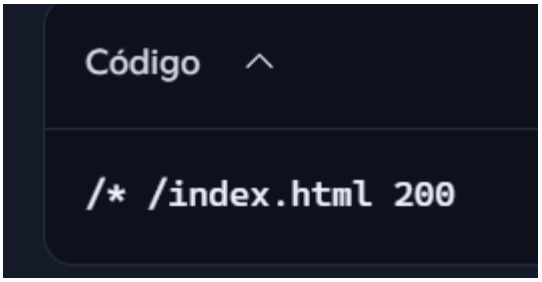


Elegir plataforma de despliegue:

- **Netlify** → arrastrar la carpeta build o conectar el repo de GitHub.
- **Vercel** → conectar el repo y desplegar automáticamente.
- **GitHub Pages** → publicar directamente desde el repositorio.

Configurar rutas SPA:

- En Netlify: archivo _redirects con



En Vercel: archivo vercel.json con rewrites hacia index.html.

Variables de entorno:

- Configurar API_URL para que Axios sepa dónde está el backend (ej. https://api.billing_factoring.com).

Conclusión.

Con esto podemos hacer un despliegue controlado en el cual evidenciamos cada uno de los procesos y proyectos que nos demuestran que no se necesita conocer mucho a profundidad de codigo, si no mas bien nos da un entendimiento el cual se compromente con modelos de software que empiezan con algoritmos de envio y respuesta de peticiones que se traducen en componente transaccionales lo que a su vez desmuestra que una pagina web debe actualizarce en tiempo real, pese a ser un modelo de respuesta es un espacio controlado para dar una experiencia al usuario mas conectiva y altamente active , sensible a cambios y conectividades de internet bajo commandos que reciben instruccion para ejecutar y facilitar procesos transaccionales.

Bibliografía & Referencias.

Regla, P. D. (2014). Diseño, contenidos y desarrollo del front-end del sitio web del proyecto auralizarte. [Trabajo de grado, Universidad Pública de Navarra].

Valdivia Caballero, J. J. (2021). Modelo de procesos para el desarrollo del front-end de aplicaciones web. [Tesis de maestría, Universidad Nacional Mayor de San Marcos]. (pp. 5-40).

Martínez Martínez, A. (2021). Proyecto feedback backend y frontend web. [Trabajo de grado, Universitat Jaume].Link de jira:

https://danielparra404.atlassian.net/jira/software/projects/SCRUM/boards/1/backlog

Repositorio GITHUB:

https://github.com/devdaniel404/analisis_sistemas.git

LUCID CHAR:

https://lucid.app/

Maze co:
https://app.maze.co/projects/469657915/mazes/469657931/results/share