Un dibujo animado con letras

Descripción generada automáticamente con confianza media

## Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación.

Tarea grupal #3

Diseño de Software Gr. 50.

## Estudiantes:

Raschell Marianny Jarquín Quesada

Carlos Eduardo Solís Mora

Fabricio Alexander Porras Morera

## Sede San Carlos Fecha de entrega:

Martes, 14 de noviembre,

# Introducción

La empresa TECSoftware se encuentra en la búsqueda de expertos en diseño de software que puedan ayudarlos a llevar a cabo la creación de una aplicación de facturación integral que será una herramienta crucial en la modernización de las operaciones comerciales actuales. La compañía se encuentra en una fase de expansión y modernización, y necesitamos una aplicación que no solo nos permita gestionar nuestras facturas de manera eficiente, sino que también integre funcionalidades que faciliten la administración de productos y la generación de informes detallados de ventas. La seguridad, la eficiencia y la facilidad de uso son aspectos cruciales que deseamos incorporar en esta nueva herramienta.

# Teoría

**Requerimientos funcionales**

1. **Gestión de facturas:** La aplicación debe de ser capaz de añadir, editar, y eliminar facturas de manera sencilla y eficaz lo cual es indispensable para el usuario.
2. **Gestión de productos:** La aplicación debe de ser capaz de añadir, editar, y eliminar productos, también de poder asignar categorías para los productos para un mejor manejo de inventario.
3. **Gestión de clientes y vendedores:** La aplicación debe ser capaz de añadir y gestionar clientes y vendedores, con la capacidad de ver su historial de compras/ventas para un mejor manejo de creación, eliminación y edición de facturas.
4. **Generación de informes detallados de ventas:** El sistema debe ser capaz de generar un informe detallado de ventas cuando el usuario lo solicite para una mejor administración para el usuario.
5. **Integración de impuestos:** La aplicación deberá ser capaz de integrar impuestos a las facturas durante su creación de forma automática para facilidad del usuario.
6. **Categorización de facturas:** El sistema deberá de ser capaz de categorizar facturas, en distintas categorías cómo cancelado, pendiente, vencida. Para una mejor administración de estas.
7. **Gestión de promociones:** El sistema deberá ser capaz de gestionar promociones para poder aplicarlas a facturas cuando el usuario lo considere necesario.
8. **Categorización de productos:** El sistema deberá ser capaz de categorizar productos, de manera que sea más accesible para el usuario encontrar alguno.
9. **Facturación electrónica:** El sistema debe ser capaz de generar facturas electrónicas cuando el cliente lo solicite, siguiendo las regulaciones fiscales necesarias.
10. **Recordatorios de falta de inventario:** El sistema deberá mandar un recordatorio al usuario cuándo en su inventario de productos existan productos que necesiten stock.

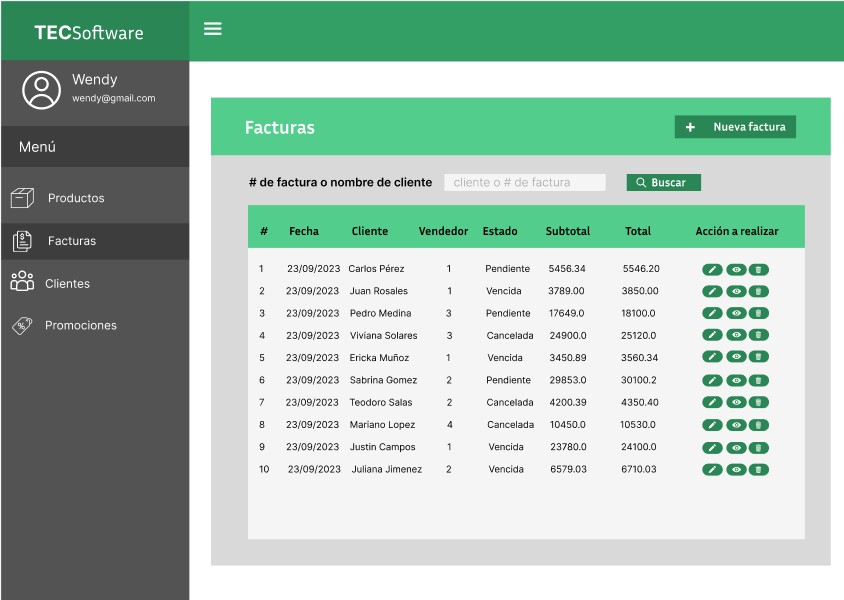
**Requerimientos no funcionales**

1. **Rendimiento:** El sistema debe ser rápido y eficaz, teniendo un tiempo de respuesta de menos de 3 segundos.
2. **Seguridad:** Los datos de la empresa y de los clientes estarán protegidos mediante estándares se seguridad de datos y cifrado.
3. **Usabilidad:** La interfaz de usuario debe ser intuitiva y factible de usar para que los usuarios puedan acoplarse a ella rápidamente sin necesitar capacitación.
4. **Compatibilidad:** La aplicación debe ser compatible con los navegadores web principales y dispositivos IOS y Android.
5. **Escalabilidad:** La aplicación debe ser escalable para adaptarse al crecimiento que pueda existir en la empresa.

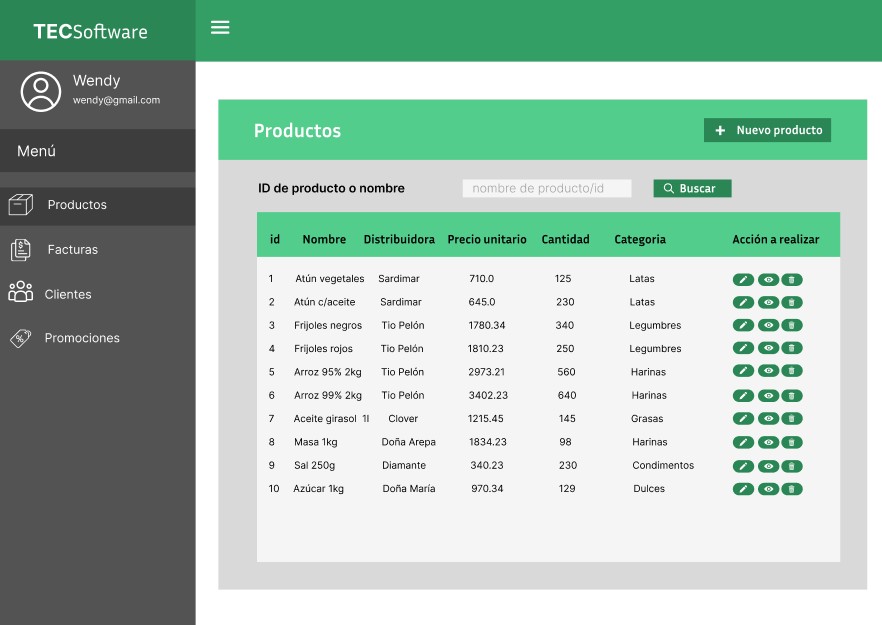
# Diseño

**Wireframes**

## Wireframe sobre gestión de facturas



## Wireframe sobre gestión de productos



# Elección de tecnologías

El desarrollo de una página web eficaz y robusta requiere la selección cuidadosa de tecnologías que se adapten a los objetivos y necesidades del proyecto. En este sentido, hemos optado por emplear React.js en el Front-End, Node.js en el Back-End, y PostgreSQL como base de datos. A continuación, se presenta una justificación detallada de estas elecciones:

**Front-End (React.js)**

React.js es una tecnología de desarrollo de interfaces de usuario muy reconocida y se ha destacado en los últimos años por sus grandes aplicaciones en desarrollo web. Se eligió esta tecnología principalmente para utilizar en el Front-End del proyecto una estructura basada en componentes que permita la creación de interfaces modulares y reutilizables, asegurando el desarrollo, pero, sobre todo, el mantenimiento de la aplicación. A su vez, la modificación, actualización y posterior expansión en sus funcionalidades será una tarea mucho más sencilla gracias a esta estructura basada en componentes.

A su vez, React.js cuenta con otros elementos para una posible expansión futura como lo puede ser la utilización de React Native para expandir la aplicación a plataformas móviles de ser requerido, lo que aumentaría mucho más el alcance de la aplicación. Sin embargo, este fue solo un elemento de incentivo más no la razón principal como la ya mencionada.

**Back-End (Node.js)**

Una de las razones principales por las que se decantó por utilizar Node.js es para mantener la coherencia en el lenguaje de programación, ya que es un beneficio fundamental. Utilizar JavaScript tanto en el lado del cliente como en el lado del servidor unifica la pila tecnológica, lo que simplifica el desarrollo y reduce la complejidad.

A su vez, Node.js es conocido por su capacidad para gestionar múltiples conexiones concurrentes de manera eficiente. Esto es esencial en aplicaciones web con alta concurrencia, garantizando un rendimiento estable y escalabilidad para afrontar un crecimiento futuro.

Finalmente, el marco de trabajo y las posibilidades de bibliotecas con las que Node.js es compatible fue otra gran razón para su elección. El marco de trabajo Empress.js, por ejemplo, ofrece características y funcionalidades que simplifican la implementación de aplicaciones web, lo que acelera y facilita el desarrollo. A su vez, la implementación de prisma para la creación inmediata de la base de datos es una funcionalidad que el equipo de desarrollo encontró muy útil para trabajar en conjunto con una misma base que necesitaría constantes cambios. Por último, el ecosistema NPM al que se puede acceder con Node.js para enriquecer el proyecto de todo tipo de funcionalidades fue muy atractivo.

**Base de datos (PostgreSQL)**

Se eligió PostgreSQL principalmente por la familiaridad y comodidad que tiene el equipo de trabajo con esta base de datos, lo cual facilitaría enormemente la velocidad de desarrollo. Sin embargo, más allá de eso, se buscó utilizar PostgreSQL, ya que se destaca por su capacidad de extensión y flexibilidad, permitiendo la creación de funciones y tipos de datos personalizados, lo que facilita la adaptación a las necesidades específicas del proyecto.

Finalmente, PostgreSQL también ofrece características avanzadas de seguridad, como autenticación sólida, autorización precisa y cifrado de datos, que serían realmente útiles para aplicaciones web con información de carácter sensible, como lo es un sistema de facturación, como lo es el caso de este proyecto. Garantizar la protección de la información de los usuarios es una prioridad, y PostgreSQL cumple con los estándares más exigentes en este sentido.

# Patrones de diseño

En el desarrollo del proyecto de página web, se han aplicado diversos patrones de diseño para garantizar una estructura organizada y una comunicación eficiente entre los componentes clave. Estos patrones de diseño son esenciales para mantener la coherencia y la escalabilidad del proyecto. A continuación, describimos los patrones de diseño empleados:

**Repositorio Git-Hub**

Se ha utilizado el patrón de control de versiones a través de un repositorio Git-Hub como una parte fundamental de nuestra estrategia de desarrollo. Esto ha permitido gestionar el código fuente de manera colaborativa y mantener un historial de cambios detallado. La utilización de ramas, confirmaciones y fusiones en Git-Hub ha facilitado la colaboración efectiva entre los miembros del equipo. Este patrón garantiza la trazabilidad, la integridad y la continuidad en el desarrollo del proyecto.

**API Gateway**

La implementación de un API Gateway es una práctica común para gestionar, supervisar y proteger las API en una arquitectura de microservicios. Este patrón proporciona un punto de entrada centralizado para las solicitudes de la API y ha permitido simplificar la comunicación con los servicios Back-End. Además, el API Gateway también brinda funcionalidades de seguridad, escalabilidad y análisis de tráfico. Asimismo, se ha utilizado un diseño de rutas y puntos finales para enrutar solicitudes a los servicios correspondientes, lo que optimiza la estructura de la API.

**Patrón Modelo Vista-Controlador**

El patrón Modelo Vista-Controlador (MVC) se ha implementado para organizar y gestionar la interacción entre el Front-End y el Back-End de esta aplicación. Este patrón permite separar las preocupaciones y lograr una arquitectura modular.

El Modelo (Model), representa la estructura de datos y lógica de negocio en nuestra aplicación. En nuestro caso, el Modelo se encarga de la manipulación de datos en la base de datos PostgreSQL y proporciona la capa de abstracción de datos para la API.

La Vista (View), define la representación y la interfaz de usuario en el Front-End. La Vista se comunica con el Controlador para obtener datos y presentarlos de manera adecuada al usuario final.

El controlador (Controller), actúa como intermediario entre el Modelo y la Vista. Gestiona las solicitudes del usuario y coordina la interacción entre el Front-End y el Back-End. Además, el Controlador procesa las operaciones CRUD y consulta la base de datos a través del Modelo.

El uso del patrón MVC ha permitido una división clara de responsabilidades y ha facilitado la escalabilidad y el mantenimiento de la aplicación.

**Patrón Cliente-Servidor**

El patrón Cliente-Servidor se ha aplicado para establecer una comunicación eficiente y coherente entre el cliente, representado por el Front-End de nuestra aplicación, y el servidor, representado por el Back-End y la base de datos PostgreSQL. Este patrón define roles y responsabilidades claros:

El cliente (Client), representa la interfaz de usuario y la parte de la aplicación con la que interactúan los usuarios. El Cliente solicita y recibe datos y servicios del Servidor.

El servidor (Server), es responsable de procesar las solicitudes del Cliente, realizar operaciones en la base de datos y proporcionar respuestas a través de la API.

La implementación del patrón Cliente-Servidor garantiza una comunicación fluida y segura entre el Front-End y el Back-End, lo que mejora la experiencia del usuario y la eficiencia de la aplicación.