Propuesta Tecnica

Villa Valentin

Introduccion

En este documento se desarrollara las caracteristicas de el sistema que se diseno para el cliente, Hospital Austra.

Comenzaremos presentando brevemente las problematicas, luego las rpropuestas soluciones, para continuar con los modulos del sistema.

Problematicas

Historial Medico

Por el momento, el cliente dispone de todos los documentos referentes al historial medico de sus clientes de forma fisica, no digital. Esta propiedad, dificulta la distribucion, el almacenamiento, y busqueda de registros medicos, teniendo un gran impacto en el servicio que recibe el cliente.

Es una ocurrencia cotidiana que los clientes desean consultar sus previas visitas, y como no poseen aceso a ella, llamen a servicio de cliente, quienes no podran ayudarlos.

Inventario

El inventario se ve organizado y mantenido regularmente. Un equipo dedicado almacena, mueve, y cuenta todo el stock, pero la division dentro de la organizacion es clara, y la cadena de comunicacion tiene que pasar por muchos niveles para poder llegar a aquellos miembros de la organizacion que hacen analisis de contaduria tanto de stock, como mediaciones, como financiero.

Turnos

El cliente, recibe pedidos de turnos por telefono o en persona. El personal que recibe estos pedidos, organiza los turnos, y luego adminsitra los clientes dentro de el hospital. Este es de los departamentos con mas personal.

Por mas que el proseso ya se vio refinado con el tiempo, es claro que las diferentes herramientas informaticas involucradas en el proceso llevan a una fragmentacion de informacion de alto valor. Los turnos, los clientes, las observaciones medicas, sus medicinas, turnos cancelados, etc, todos distribuidos atraves de distintas bases de datos y sistemas.

Tambien, los clientes suelen no asistir a sus turnos por varias razon. Muchos llaman varias veces para consultar sus fechas. El servicio de estos clientes es muy intensivo en cuanto a tiempo y recursos.

Medicacion

Las medicaciones se ven asignadas por los doctores durante una visita. Estas, escritas en papel, son incontables debido a que son entregadas de manera fisica al cliente, y habia una sola copia, son irrecuperables. Por intereses financieros, el cliente desea de grabar esta informacion en una base de datos para compartir con sus socios, las aseguradoras, quienes ofrecen beneficios a aquellos que distribuyen un mayor numero de medicacion. Por otro lado, tambien es importante contar aquella medicacion que se le entrego al cliente durante la visita.

La medicacion forma parte del inventario. Aunque distinto en muchos aspectos, estos elementos deben ser almacenados, contandos, y registrados de manera apropiada.

Soluciones

Historial Medico

Se propone como solucion una plataforma digital web donde los clientes, una vez asociados al sistema, pueda visualizar su historial medico personal, adjuntando las notas del doctor responsable de la visita, y las medicacion prescriptas.

Esto le serviria al cliente y reduciria el consumo de recursos de atencial al cliente. Mas importantemente, permitira a el doctor responsable de la visita ver todo el historial medico del pasiente, para una mejor atencion.

Al mismo tiempo crear un registro que puede ser utilizado luego para el estudio de ciertos casos, observando como evoluciono el caso especifico de el cliente.

Inventario

El inventario siendo un componente critico de nuestro cliente debe de ser enfrentado con varias soluciones especificas que trabajen en conjunto.

Primero de todo, se propone una plataforma web donde empleados puedan cargar la informacion de el inventario que entra y el inventario que sale y su informacion clave, como precio por unidad y cantidades.

Otro componente hermano es el registro de moviminetos. Este se encarga de registrar cada moivimiento singular que se realize.

Muy similar al ultimo, el reporte mensual esta disenado para poder entender de manera rapida la trayectoria de estos elementos, y permitir a analistas ver reportes del pasado para aportar a su analisis.

Elemenetal, un area del sistema que disponenga de toda la informacion actual del mes y el inventario, como cuantas unidades hay, cuanto valor, cuanto gasto, etc. Este es clave para el analisis continuo, la creacion de reportes, la comunicacionas bilateral o jerarquica, y permite analisar el estado actual.

Turnos

Para los turnos se disenaron varias soluciones.

Primario esta la plaraforma de la que disponen los clientes para crear reservar, donde pueden especificar hora y fecha tanto como el doctor acargo de la visita. Ellos tambien podran ver sus turnos reservados y podran cancelardos si es lo que desean.

Tambien se crearan registros de estas visitas para poder generar analisis.

Acompanando estara el reporte de visitas que generara un analisis espontaneo de las visitas del ultimo mes y permitira al equipo de analsitas informarse de manera rapida.

Para aquellos gestores, habra una tabla mostrando los turnos del dia actual, para facilitar su trabajo.

Mediacion

Las soluciones propuestas para el problema de la mediacion son altamente similares a aquella soluciones expresadas para el inventario.

DIsponera de una plataforma para cargar la informacion d ela medicina entrante y la saliente. Por otro lado habra una seccion de reporte donde se podra visualizar los gatos, el valor del stock, el consumo, y mas informacion referente a la medicacion.

Se registraran los moviminetos de las mediaciones y su consumo.

Simientos del Sistema

Para el sistema que estoy construyendo, he seleccionado una arquitectura tecnológica moderna y escalable, teniendo en cuenta las necesidades específicas de la aplicación y el rendimiento necesario para una plataforma de uso intensivo en términos de datos y transacciones. A continuación, detallo las tecnologías elegidas para cada componente, explicando las razones detrás de cada elección.

Backend: Node.js con Express y Prisma

Node.js: Node.js es la elección para el backend por su capacidad de manejar aplicaciones de alto rendimiento y escalabilidad. Node.js ejecuta JavaScript en el servidor, lo cual es coherente con la elección de JavaScript/TypeScript en el frontend y permite un flujo de trabajo de pila completa en el mismo lenguaje. Además, Node.js se maneja muy bien con aplicaciones de I/O intensivo, lo cual es ideal para un sistema de gestión de turnos que constantemente está interactuando con una base de datos.

Express: Express es un framework minimalista que simplifica la construcción de APIs RESTful en Node.js. Permite definir rutas y controladores de forma clara y directa, haciendo que el código sea más mantenible. En este sistema, con endpoints como `insert`, `delete`, y `update` para gestionar usuarios, turnos, y relaciones, Express facilita la definición de esas rutas de forma organizada.

Prisma como ORM: Prisma es un ORM moderno que permite trabajar con bases de datos relacionales de manera declarativa. La elección de Prisma se debe a su excelente integración con TypeScript, su sistema de migraciones fácil de manejar y su capacidad para interactuar con modelos complejos de datos. Además, Prisma genera tipos de datos automáticamente a partir de los modelos de base de datos, lo cual asegura que las consultas estén correctamente tipadas, reduciendo la posibilidad de errores.

Frontend: React con TypeScript

Elección de React: React es una biblioteca de JavaScript ampliamente adoptada para construir interfaces de usuario dinámicas y de alto rendimiento. La razón principal para usar React es su capacidad para construir aplicaciones de una sola página (SPAs) que permiten una experiencia de usuario fluida sin necesidad de recargar la página completa. En este caso, al crear un sistema con formularios interactivos, múltiples vistas y una navegación constante entre componentes, React permite organizar la interfaz de forma modular y reutilizar componentes.

TypeScript: La elección de TypeScript en el frontend es clave, dado que el sistema maneja varios tipos de datos que deben ser correctamente tipados. TypeScript ayuda a evitar errores comunes al proporcionar un sistema de tipos estáticos, lo cual aumenta la robustez del código y facilita el mantenimiento a largo plazo. Además, dado que el sistema maneja datos sensibles de usuarios, como sus turnos y roles, TypeScript asegura que esos datos se manipulen correctamente en cada componente.

Justificación de JSX: JSX facilita la creación de componentes React con una sintaxis que mezcla HTML y JavaScript. Esto es particularmente útil, ya que permite diseñar rápidamente la interfaz de usuario y ver el flujo de datos entre componentes de forma clara y concisa.

Base de Datos: PostgreSQL

PostgreSQL: Elegí PostgreSQL como la base de datos relacional por su estabilidad, robustez y soporte avanzado para tipos de datos complejos. PostgreSQL se adapta bien a las relaciones que existen en el sistema, tales como los modelos de `Turno`, `TurnoTime` y `User`, donde es crucial asegurar la integridad referencial. Además, PostgreSQL maneja de forma eficiente las consultas complejas y tiene una excelente integración con Prisma, lo cual facilita el acceso a los datos desde el backend.

Autenticación y Autorización: JSON Web Tokens (JWT) y Middleware en Express

JWT para Autenticación: Los JSON Web Tokens son ideales para gestionar la autenticación en una aplicación distribuida. JWT permite que cada solicitud enviada al servidor lleve consigo un token que contiene información de autenticación y permisos. Esto es particularmente útil en este sistema, donde distintos tipos de usuarios (como CLIENT, DOCTOR, ADMIN) tienen diferentes niveles de acceso y permisos. JWT permite una implementación de autenticación segura y escalable sin necesidad de almacenar sesiones en el servidor.

Middleware para Autorización en Express: Uso middleware en Express para interceptar las solicitudes y verificar que los usuarios tengan los permisos necesarios antes de acceder a ciertos endpoints. Esto asegura que solo los usuarios autorizados puedan realizar acciones como cancelar o modificar turnos. Además, el middleware mejora la legibilidad del código al separar la lógica de autorización de los controladores principales.

CSS y Diseño: Tailwind CSS

Tailwind CSS: Tailwind es un framework de diseño basado en utilidades que permite crear interfaces de usuario consistentes y responsivas de manera rápida. La elección de Tailwind se debe a su enfoque de "first-class utilities", que permite definir estilos directamente en los componentes sin tener que escribir CSS personalizado. Esto es útil para un sistema como este, que requiere una interfaz limpia y eficiente, con formularios y tablas que se adapten a distintos tamaños de pantalla y mantengan una experiencia de usuario fluida.

Beneficios para el Desarrollo Rápido: Tailwind permite un desarrollo de frontend más rápido y flexible, ideal para prototipos y ajustes rápidos. En este caso, ayuda a definir estilos claros y consistentes para elementos como formularios, botones y tarjetas de turnos sin necesidad de escribir CSS personalizado desde cero.

Gestión de Estado en Frontend: useState y useEffect en React

useState y useEffect: React Hooks, como `useState` y `useEffect`, son fundamentales para manejar el estado en el sistema. `useState` permite almacenar valores dinámicos, como los datos del usuario y los turnos seleccionados. `useEffect`, por otro lado, facilita la sincronización de la interfaz de usuario con cambios en el backend, como la actualización de turnos cancelados o modificaciones en los datos del usuario.

Control de Versiones y Colaboración: Git y GitHub

Git para Control de Versiones: Git es fundamental en cualquier proyecto colaborativo, y permite realizar cambios en el código sin miedo a perder el trabajo o introducir errores. La posibilidad de crear ramas independientes facilita el desarrollo de nuevas características sin afectar la versión estable del sistema.

Propiedades del Sistema

5.1.Niveles de Acceso

Debido a la complejidad de la problemática el sistema tendrá diferentes tipos de usuarios que en si poseerán distintos tipos de acceso como ya lo hacen dentro de la organización. Estos tipos de acceso se ven necesarios debido a legislación como también la simplificación de las operación y responsabilidades de estos personajes.

Una muy importante diferencia el mencionado nivel de acceso del cual existirán 4:

- Nivel de acceso 1
- Nivel de acceso 2
- Nivel de acceso 3
- Nivel de acceso 4

Los distintos niveles de acceso diferenciaran entre el acceso de información y funcionalidades. Sus detalles se verán expresados a lo largo del documento.

5.2.Usuarios

En el sistema se presentará una distinción entre usuarios que representan diferentes roles dentro del Hospital Austral.

Todos los usuarios se verán conformados por un nombre de usuario y una contraseña.

Adelante se detallan sus características…

- **Cliente:** El usuario *“cliente”* será representativo del individual a el cual el Hospital Austral le presta servicios y/o le vende mercancía. Este tendrá un nivel de acceso de tipo 1.
- **Doctor:** El usuario *“doctor”* es para aquellos empleados cuyo propósito es puramente medico ya sea de consultoría o acción médica. Estos tendrán un nivel de acceso de tipo 2.
- **Empleado Administrativo:** El usuario denominado *“empleado administrativo”* es como indica el nombre para empleados cuyas responsabilidades es el manejo de información, turnos, y atención al cliente que muchas veces colaboran con el personal médico para la atención de los clientes. Este usuario posee un nivel de acceso 3.
- **Administrador:** El *“administrador”* es un usuario que se ve necesario para las correcta y continua funcionalidad del sistema como también para todas las tareas de debugging, asistencia, e vigilancia del sistema. Este poseerá permisos y funcionalidades mucho más extensas que las de los otros usuarios y conllevara una responsabilidad mayor, es por ello que su nivel de acceso es nivel de acceso 4.

Modulos

Módulo: Registro de Usuario

Este módulo permite a los nuevos usuarios registrarse en la aplicación proporcionando su correo electrónico, contraseña, nombre y apellido. Al completar el registro, el sistema crea un nuevo registro de usuario asignándole, por defecto, un rol de "cliente" que otorga los permisos necesarios para interactuar con las funcionalidades de la aplicación.

Campos y Requerimientos

Campo	Tipo	Requerimiento	Descripción
`email`	Alfanumérico	Requerido, Único	Correo del usuario, utilizado como ID de login.
`password`	Alfanumérico	Requerido	Contraseña para la seguridad de la cuenta.
`firstname`	Alfanumérico	Requerido	Nombre del usuario.
`lastname`	Alfanumérico	Requerido	Apellido del usuario.

Descripción del Componente en el Frontend

El formulario de registro en el componente frontend recopila la siguiente información:

- **Correo Electrónico**: Se requiere un formato válido de correo; el campo valida la estructura para evitar errores.
- **Contraseña**: Campo de entrada con opción de ocultar o mostrar la contraseña para mayor seguridad.
- **Nombre** y **Apellido**: Entradas de texto donde se introducen el nombre y apellido del usuario respectivamente.

Punto de Acceso (Backend)

Endpoint: `/signup`

Método: `POST`

Solicitud (Request)

El frontend envía una solicitud para crear un nuevo usuario con la siguiente estructura:

```
{
 "user": {
   "email": "string",       // Correo del usuario, debe ser único en la base de datos.
   "password": "string",     // Contraseña del usuario.
   "firstname": "string",    // Nombre del usuario.
   "lastname": "string"      // Apellido del usuario.
  }
}
```

Respuesta (Response)

Códigos de Estado:

- `204 No Content`: Indica que el registro del usuario fue exitoso, sin errores.
- `400 Bad Request`: El correo electrónico ya existe en el sistema, o se produce un error de validación.
- `500 Internal Server Error`: Error inesperado en el servidor al intentar registrar al usuario.

Manejo de Errores:

- Si el correo ya existe, el backend responde con un estado `400`, informando al frontend que el usuario debe elegir un correo diferente.
- Cualquier error inesperado se registra en el sistema y retorna un estado `500`.

Flujo de Datos

1. **Componente Frontend**: El componente `Register` recopila la información del usuario y la envía mediante la función `signup` del hook de autenticación.
2. **Procesamiento en Backend**:
   - La función `signup` verifica si el correo electrónico ya está registrado.
   - Si el correo es único, el backend crea un nuevo usuario con el tipo `CLIENT` y genera un registro vinculado en la tabla de clientes.
1. **Base de Datos**:
   - `User`: Se crea un nuevo registro de usuario asignándole un rol y tipo de acuerdo a la lógica del negocio.
   - `Client`: Se genera un registro de cliente asociado para establecer la relación de cliente en el sistema.

Módulo: Inicio de Sesión

El módulo de **Inicio de Sesión** permite a los usuarios registrados acceder a la aplicación introduciendo su correo electrónico y contraseña. Durante el inicio de sesión, el sistema valida las credenciales del usuario y, de ser correctas, genera y asigna dos cookies de autenticación: `access_token` y `refresh_token`. Estos tokens permiten la persistencia de la sesión del usuario en la aplicación y la renovación automática de la autenticación.

Campos y Requerimientos

Campo	Tipo	Requerimiento	Descripción
`email`	Alfanumérico	Requerido	Correo del usuario, utilizado como identificador.
`password`	Alfanumérico	Requerido	Contraseña para la autenticación del usuario.

Descripción del Componente en el Frontend

El formulario de inicio de sesión en el frontend permite a los usuarios introducir sus credenciales de acceso:

- **Correo Electrónico**: Campo de entrada que requiere un formato de correo válido.

- **Contraseña**: Campo de entrada de texto seguro que permite ocultar o mostrar el contenido según el estado del botón de visibilidad.

Punto de Acceso (Backend)

Endpoint: `/auth/login`

Método: `POST`

Solicitud (Request)

El frontend envía una solicitud `POST` a `/auth/login` para autenticar al usuario con el siguiente cuerpo JSON:

```
{
 "user": {
  "email": "string",      // Correo del usuario.
  "password": "string"    // Contraseña en texto plano.
 }
}
```

Respuesta (Response)

Códigos de Estado:

- `200 OK`: Inicio de sesión exitoso; se devuelven los datos del usuario.
- `400 Bad Request`: Credenciales inválidas o correo no registrado en el sistema.
- `500 Internal Server Error`: Error inesperado en el servidor.

Manejo de Errores:

- Si las credenciales no son válidas, el backend responde con un código `400`, informando al frontend de un error en el correo o contraseña.
- Si ocurre un error inesperado, el backend limpia las cookies de sesión y devuelve un código `500`.

Flujo de Datos

1. **Componente Frontend**: El componente `Login` envía la solicitud de autenticación al backend mediante la función `login` del hook de autenticación. Si el inicio de sesión es exitoso, se redirige al usuario a la página principal.
2. **Procesamiento en Backend**:
   - La función `login` verifica si el correo está registrado y si la contraseña proporcionada es correcta.
   - Si las credenciales son válidas, el backend asigna dos cookies de autenticación: `refresh_token` (para renovación de sesión) y `access_token` (para autenticación en solicitudes).
1. **Base de Datos**:
   - `User`: La función `readByEmail` busca el usuario en la base de datos según el correo proporcionado. Si existe, se valida la contraseña.

- `RefreshToken`: Si el usuario ya tiene un `refresh_token` asignado, este se reutiliza; de lo contrario, se genera uno nuevo.

Cookies de Sesión

Las siguientes cookies se asignan al iniciar sesión:

- `access_token`: Token de acceso utilizado en cada solicitud para mantener la autenticación del usuario. Expira en 1 hora.
- `refresh_token`: Token que permite la renovación de sesión automáticamente. Expira en 30 días si es un token nuevo o en 10 años si ya existía.

Datos Retornados en la Respuesta

Si el inicio de sesión es exitoso, el backend retorna la siguiente información del usuario:

```
{
  "user_id": 1,
  "email": "usuario@ejemplo.com",
  "firstname": "Nombre",
  "lastname": "Apellido",
  "role": 1,
  "type": "CLIENT",
  "doctor_id": null,
  "client_id": 123,
  "desk_id": null,
  "admin_id": null
}
```

Módulo: Inventario

El módulo de **Inventario** permite la gestión de los ítems en stock en el sistema. Este módulo facilita al usuario las siguientes funcionalidades: visualizar el inventario completo, crear nuevos ítems, filtrar ítems por nombre, eliminar múltiples ítems seleccionados y consumir unidades del inventario.

Campos y Requerimientos

Campo	Tipo	Requerimiento	Descripción
`item_id`	Numérico	Automático	Identificador único de cada ítem en el inventario.
`name`	Alfanumérico	Único	Nombre del ítem, debe ser único en el inventario.
`category`	Alfanumérico	Requerido	Categoría a la que pertenece el ítem.
`quantity`	Numérico	Requerido	Cantidad en stock del ítem.
`unit_price`	Numérico	Requerido	Precio unitario del ítem.
`description`	Alfanumérico	Opcional	Descripción del ítem en el inventario.

Descripción del Componente en el Frontend

En el frontend, el componente `Inventario` muestra una tabla con todos los ítems disponibles en el sistema y permite las siguientes funcionalidades:

- **Visualización y Filtrado de Ítems**: Se muestra una tabla con columnas que incluyen el `item_id`, `nombre`, `categoría`, `cantidad`, `precio unitario` y `descripción`. Los usuarios pueden filtrar los ítems ingresando texto en el campo de búsqueda.
- **Eliminación de Ítems**: Los usuarios pueden seleccionar múltiples ítems y eliminarlos en lote.
- **Consumo de Ítems**: Los usuarios pueden reducir la cantidad de ítems en stock mediante el botón de "Consumir", lo cual representa la salida de stock de una unidad.

Punto de Acceso (Backend)

Endpoint:

- `/inv/selectAll` para obtener todos los ítems.
- `/inv/insert` para crear un ítem.
- `/inv/deletemany` para eliminar varios ítems.
- `/inv/subtractmany` para consumir ítems seleccionados.

Solicitudes (Requests)

1. **Obtener todos los ítems**
   - **Método**: `POST` a `/inv/selectAll`.
   - **Request Body**: No se requiere.
   - **Funcionalidad**: Retorna todos los ítems en inventario, incluyendo su `item_id`, `nombre`, `categoría`, `cantidad`, `precio unitario` y `descripción`.
1. **Crear un nuevo ítem**
   - **Método**: `POST` a `/inv/insert`.
   - **Request Body**:

```
{
 "item": {
  "name": "string",         // Nombre único del ítem.
  "category": "string",      // Categoría del ítem.
  "unit_price": "number",    // Precio unitario.
  "description": "string",   // Descripción opcional.
  "stock": {
   "quantity": "number"     // Cantidad en stock.
  }
 }
}
```

   - Funcionalidad: Crea un nuevo ítem en el inventario. Si el nombre ya existe, el servidor responde con un error 400.
1. Eliminar múltiples ítems

- Método: POST a /inv/deletemany.
- Request Body:

```
{
 "items": [
  {
   "item_id": "number",      // ID del ítem a eliminar.
   "stock": { "quantity": "number" }  // Cantidad actual en stock.
  }
 ]
}
```

1. Consumir unidades de ítems

- Método: POST a /inv/subtractmany.
- Request Body:

```
{
 "items": [
  {
   "item_id": "number",         // ID del ítem a consumir.
   "stock": { "quantity": "number" }  // Cantidad actual en stock.
  }
 ]
}
```

- Funcionalidad: Disminuye en una unidad la cantidad de ítems seleccionados en el inventario.

Respuestas (Responses)

Códigos de Estado:

- `200 OK`: Operación completada exitosamente.
- `400 Bad Request`: Error en la solicitud, como cuando el nombre del ítem ya existe al intentar crear uno nuevo.
- `404 Not Found`: Ítem no encontrado al intentar eliminar o consumir una unidad de un ítem inexistente.
- `500 Internal Server Error`: Error inesperado en el servidor.

Datos Retornados en la Respuesta

- **Respuesta de Selección**: Una lista con los ítems del inventario, cada uno con su `item_id`, `nombre`, `categoría`, `cantidad`, `precio unitario`, y `descripción`.
- **Respuesta de Creación**: El ítem creado, incluyendo los detalles del stock.

- **Respuesta de Eliminación**: La lista de ítems eliminados.
- **Respuesta de Consumo**: El ítem consumido con el cambio de cantidad reflejado en la respuesta.

Manejo de Errores y Seguridad

- **Validación de Token**: Algunas operaciones en este módulo pueden requerir autenticación mediante un `access_token` en las cookies.
- **Mensajes de Error**: En caso de duplicación de nombre o ítems inexistentes, el servidor responde con un mensaje de error y el código HTTP correspondiente.

Módulo: Medicación

El módulo de **Medicación** permite gestionar el inventario de medicamentos en el sistema. Este módulo le facilita al usuario visualizar la lista de medicaciones disponibles, filtrar por nombre, agregar nuevas medicaciones, eliminar varias medicaciones a la vez y actualizar la cantidad de unidades en el stock. A través de estas funcionalidades, se asegura una administración ordenada y eficiente del inventario de medicaciones.

Campos y Requerimientos

Campo	Tipo	Requerimiento	Descripción
`med_id`	Numérico	Automático	Identificador único de cada medicamento.
`name`	Alfanumérico	Único	Nombre del medicamento, debe ser único en el inventario.
`quantity`	Numérico	Requerido	Cantidad en stock del medicamento.
`unit_price`	Numérico	Requerido	Precio unitario del medicamento.
`description`	Alfanumérico	Opcional	Descripción del medicamento en el inventario.

Descripción del Componente en el Frontend

En el frontend, el componente `Medicación` permite al usuario interactuar con la lista de medicamentos en stock. Las funcionalidades principales incluyen:

- **Visualización y Filtrado**: Una tabla con las columnas `med_id`, `nombre`, `cantidad`, `precio unitario` y `descripción`, que permite al usuario filtrar por nombre.
- **Eliminación de Medicamentos**: Selección y eliminación en lote de varias medicaciones del inventario.
- **Consumo de Medicamentos**: Permite reducir en una unidad la cantidad en stock de las medicaciones seleccionadas.

Punto de Acceso (Backend)

Endpoint:

- `/med/selectAll` para obtener todos los medicamentos.
- `/med/insert` para agregar un nuevo medicamento.
- `/med/deleteMany` para eliminar múltiples medicamentos.
- `/med/subtractQuantityManyByOne` para reducir en una unidad el stock de múltiples medicamentos.

- `/med/subtractQuantityMany` para reducir el stock en una cantidad específica indicada por el usuario.

Solicitudes (Requests)

1. **Obtener todos los medicamentos**
   - **Método**: `POST` a `/med/selectAll`.
   - **Request Body**: No se requiere.
   - **Funcionalidad**: Retorna la lista de todos los medicamentos, con sus detalles de `med_id`, `nombre`, `cantidad`, `precio unitario` y `descripción`.
1. **Agregar un nuevo medicamento**
   - **Método**: `POST` a `/med/insert`.
   - **Request Body**:

```
{
 "med": {
  "name": "string",          // Nombre único del medicamento.
  "unit_price": "number",    // Precio unitario.
  "description": "string",   // Descripción opcional.
  "stock": {
   "quantity": "number"     // Cantidad inicial en stock.
  }
```

   - Funcionalidad: Crea un nuevo registro de medicamento en el inventario. Si el nombre ya existe, el servidor responde con un error 400.

1. Eliminar múltiples medicamentos

- Método: POST a /med/deleteMany.
- Request Body:

```
{
 "meds": [
  {
   "med_id": "number",        // ID del medicamento a eliminar.
   "stock": { "quantity": "number" }  // Cantidad en stock.
  }
 ]
}
```

   - Funcionalidad: Permite eliminar varios medicamentos seleccionados del inventario.

1. Reducir la cantidad de medicamentos en stock en una unidad

- Método: POST a /med/subtractQuantityManyByOne.

Request Body:

```
{
 "meds": [
  {
   "med_id": "number"         // ID del medicamento.
  }
 ]
}
```

- Funcionalidad: Disminuye en una unidad la cantidad en stock de los medicamentos seleccionados.

Reducir la cantidad de medicamentos en stock por una cantidad específica

- Método: POST a /med/subtractQuantityMany.
- Request Body:

```
{
 "meds": [
  {
   "med_id": "number",         // ID del medicamento.
   "quantity": "number"        // Cantidad a reducir.
  }
 ]
}
```

- Funcionalidad: Disminuye en una cantidad especificada el stock de los medicamentos seleccionados. Si la cantidad especificada supera el stock disponible, devuelve un error.

Respuestas (Responses)

Códigos de Estado:

- `200 OK`: Operación completada exitosamente.
- `400 Bad Request`: Error en la solicitud, por ejemplo, si el nombre del medicamento ya existe al crear uno nuevo.
- `404 Not Found`: Medicamento no encontrado al intentar eliminar o consumir una unidad.
- `500 Internal Server Error`: Error inesperado en el servidor.

Datos Retornados en la Respuesta

- **Respuesta de Selección**: Lista de medicamentos en el inventario, con `med_id`, `nombre`, `cantidad`, `precio unitario`, y `descripción`.
- **Respuesta de Creación**: El medicamento creado con los detalles del stock.
- **Respuesta de Eliminación**: Lista de medicamentos eliminados.

- **Respuesta de Consumo**: Los medicamentos consumidos, con el cambio de cantidad reflejado en la respuesta.

Manejo de Errores y Seguridad

- **Validación de Token**: Algunas operaciones de este módulo requieren autenticación mediante un `access_token`.
- **Manejo de Errores**: Si se solicita una cantidad mayor a la disponible, el servidor retorna un error detallado.

Módulo: Reserva de Turnos Médicos

Funcionalidad Principal:
El módulo de **Reserva de Turnos Médicos** permite a los usuarios reservar turnos con un médico específico en fechas y horarios seleccionados. Los usuarios pueden seleccionar un médico, ver las fechas y horas disponibles, y confirmar la reserva de un turno. Además, el sistema permite la cancelación de turnos ya reservados.

Componentes del Frontend

1. Reserva (Componente Principal)

- **Descripción:**
  Este componente permite la reserva de turnos en una interfaz interactiva. Presenta una lista de nombres de médicos disponibles, un calendario para seleccionar la fecha, un selector de horarios y un botón para confirmar la reserva.
- **Campos y Requisitos:**
- `availableNames`: Almacena los nombres de médicos que se pueden seleccionar.
- `selectedName`: Almacena el nombre del médico seleccionado (alfa).
- `doctorId`: Almacena el ID del médico (numérico, requerido).
- `unavailableDates`: Contiene las fechas no disponibles para el médico seleccionado (array de fechas).
- `availableTimes`: Contiene las horas disponibles en la fecha seleccionada.
- `selectedDate`: Fecha seleccionada por el usuario (fecha, requerido).
- `selectedTime`: Hora seleccionada por el usuario (hora, requerido).
- **Flujo de Trabajo:**

1. Se cargan los nombres de los médicos al iniciar el componente (`fetchDoctorNames`).
2. El usuario selecciona un médico, fecha y hora.
3. Al confirmar la selección, se envía una solicitud para reservar el turno (`handleRequestTurno`).

Métodos Backend

1. `getAvailableDates` (Ruta: `/turn/getAvailableDates`)

- **Descripción:**
  Recupera las fechas disponibles para un médico específico.
- **Parámetros:**
- `doctor_id` (numérico, requerido): ID del médico.

- **Respuesta:**
  Devuelve un array de fechas no disponibles para el médico seleccionado.
- **Ejemplo de Respuesta:**

```
{
  "dates": ["2023-09-15", "2023-09-16", ...]
}
```

2. getAvailableTimes (Ruta: /turn/getAvailableTimes)

- Descripción:
  Obtiene los horarios disponibles en una fecha específica para el médico seleccionado.
- Parámetros:
- doctor_id (numérico, requerido): ID del médico.
- selectedDate (fecha, requerido): Fecha en la que se buscan horarios disponibles.
- Respuesta:
  Devuelve un array de horas disponibles en el día solicitado.
- Ejemplo de Respuesta:

```
{
  "times": ["09:00", "10:00", "14:00"]
}
```

3. createTurno (Ruta: /turn/createTurno)

- Descripción:
  Crea un turno reservando una fecha y hora específicas con un médico.
- Parámetros:
- client_id (numérico, requerido): ID del usuario que reserva el turno.
- doctor_id (numérico, requerido): ID del médico con el que se reserva el turno.
- turnoTime (objeto, requerido): Fecha y hora del turno.
- Respuesta:
  Devuelve el turno creado, confirmando la reserva.
- Ejemplo de Respuesta:

```
{
  "turno_id": 1,
  "client_id": 2,
  "doctor_id": 3,
  "turnoTime": {
     "date": "2023-09-20",
     "time": "14:00"
  },
  "status": "confirmed"
}
```

4. cancelById (Ruta: /turn/cancelById)

- Descripción:
  Cancela un turno existente en el sistema, indicando la fecha y hora de cancelación.
- Parámetros:
- turn_id (numérico, requerido): ID del turno a cancelar.
- Respuesta:
  Confirma la cancelación del turno con el turn_id correspondiente.
- Ejemplo de Respuesta:

```
{
  "turno_id": 1,
  "status": "cancelled"
}
```

Campos y Requisitos del Backend

- **Tabla `Turno`**
- `turno_id` (numérico, requerido): Identificador único del turno.
- `doctor_id` (numérico, requerido): Identificador del médico asignado al turno.
- `client_id` (numérico, requerido): Identificador del cliente que reservó el turno.
- `turnoTime` (fecha y hora, requerido): Fecha y hora del turno.
- `cancelation_date` (fecha, opcional): Fecha y hora de cancelación, si aplica.
- **Tabla `TurnoTime`**
- `date` (fecha, requerido): Fecha del turno.
- `time` (hora, requerido): Hora del turno.

Módulo: Consultar Turnos

Funcionalidad Principal:
El módulo de **Consultar Turnos** permite a los usuarios ver todos los turnos previamente reservados con sus respectivos médicos. Además, se incluye una opción para cancelar los turnos

en caso de que el usuario ya no pueda asistir o desee modificar su reserva. El sistema muestra la información detallada del turno, como el médico asignado, la fecha y la hora.

Componentes del Frontend

1. ConsultarTurnos

- **Descripción:**
  Este componente muestra todos los turnos reservados por el usuario autenticado. Presenta la información de cada turno en una tarjeta con detalles sobre el médico, la fecha y la hora de la cita. Además, permite al usuario cancelar cualquier turno directamente desde la interfaz.
- **Campos y Requisitos:**
- `turnos`: Almacena los turnos asociados al usuario logueado (array de objetos de turno).
- `user`: Información del usuario logueado, utilizada para filtrar los turnos.
- **Flujo de Trabajo:**

1. Al cargar el componente, se invoca la función `fetchTurnos` para obtener todos los turnos asociados al `user_id` del usuario logueado mediante el método `selectByUser_id`.
2. Los turnos se muestran en una lista de tarjetas, cada una con el nombre del médico, la fecha y la hora del turno.
3. Cada turno cuenta con un botón de "Cancelar" que invoca la función `cancelTurno`, que llama al método `cancelById` para cancelar el turno correspondiente y eliminarlo de la lista visible.

Métodos Backend

1. `selectByUser_id` (Ruta: `/turn/selectByUser_id`)

- **Descripción:**
  Recupera todos los turnos asociados a un usuario específico, filtrados por `user_id`.
- **Parámetros:**
- `user_id` (numérico, requerido): ID del usuario logueado.
- **Respuesta:**
  Devuelve un array de turnos, cada uno con detalles sobre el médico, la fecha y la hora del turno.
- **Ejemplo de Respuesta:**

```
[
  {
    "id": 1,
    "doctor": {
      "user": {
        "firstname": "Juan",
        "lastname": "Perez"
      }
    },
    "turnoTime": {
      "date": "2023-09-25",
      "time": "14:00"
    }
  },
  ...
]
```

cancelById (Ruta: /turn/cancelById)

- Descripción:
  Permite la cancelación de un turno especificado. El sistema marca el turno como cancelado y actualiza la base de datos.
- Parámetros:
- turn_id (numérico, requerido): ID del turno a cancelar.
- Respuesta:
  Devuelve el turno actualizado indicando que ha sido cancelado.
- Ejemplo de Respuesta:

```
{
  "turno_id": 1,
  "status": "cancelled"
}
```

Campos y Requisitos del Backend

- **Tabla `Turno`**
- `turno_id` (numérico, requerido): Identificador único del turno.
- `doctor_id` (numérico, requerido): Identificador del médico asignado al turno.
- `client_id` (numérico, requerido): Identificador del cliente que reservó el turno.
- `turnoTime` (objeto, requerido): Fecha y hora del turno.
- `cancelation_date` (fecha, opcional): Fecha y hora de cancelación del turno (campo opcional que solo se completa en caso de cancelación).

- **Tabla `TurnoTime`**
- `date` (fecha, requerido): Fecha en la que se llevará a cabo el turno.
- `time` (hora, requerido): Hora específica del turno.

Módulo: Gestión de Usuarios

Funcionalidad Principal:
El módulo de **Gestión de Usuarios** permite la administración completa de usuarios y departamentos dentro del sistema. Este módulo facilita la creación, modificación, eliminación y consulta de usuarios, y su asignación a departamentos. Los usuarios se categorizan en diferentes roles (Doctor, Administrador, Cliente y Escritorio), y el sistema ofrece filtros y acciones en masa para agilizar la administración de datos.

Componentes del Frontend

1. Usuarios

- **Descripción:**
  Este componente muestra una lista de usuarios con detalles específicos (como el ID, correo electrónico, nombre, apellido y rol) y permite realizar acciones de filtrado, eliminación en masa y actualización de usuarios.
- **Campos y Requisitos:**
- `userData` (array de objetos de usuario): Contiene todos los usuarios registrados.
- `userRows` (array de objetos de usuario): Contiene los usuarios a mostrar en la tabla.
- `departmentData` (array de objetos de departamento): Almacena los departamentos disponibles.
- `departmentRows` (array de objetos de departamento): Contiene los departamentos a mostrar en la tabla.
- `selectedUserRows` y `selectedDepartmentRows` (arrays de objetos seleccionados): Contiene los usuarios y departamentos seleccionados para realizar acciones en masa.
- **Flujo de Trabajo:**

1. Al cargar el componente, se invoca la función `fetchData` para obtener los datos de usuarios y departamentos utilizando el método `managerData`.
2. Los datos de usuarios y departamentos se presentan en tablas con opciones de filtrado y paginación.
3. Los usuarios y departamentos seleccionados pueden eliminarse mediante los botones de eliminación en masa (`bulkDeleteUsers` y `bulkDeleteDepartments`).
4. Un botón de "Crear" redirige a formularios de creación de nuevos usuarios y departamentos.

Métodos Backend

1. `managerData` (Ruta: `/user/manager`)

- **Descripción:**
  Recupera todos los datos de usuarios y departamentos para presentarlos en el módulo de gestión.
- **Parámetros:**
- Sin parámetros en el body; requiere token de autenticación.

- **Respuesta:**
  Devuelve dos arrays, `userData` y `departmentRows`, que contienen la información de todos los usuarios y departamentos.
- **Ejemplo de Respuesta:**

```
{
  "userData": [
    {
      "user_id": 1,
      "email": "juan.perez@ejemplo.com",
      "firstname": "Juan",
      "lastname": "Perez",
      "role": 2,
      "type": "DOCTOR"
    },
    ...
  ],
  "departmentRows": [
    {
      "id": 1,
      "name": "Cardiología"
    },
    ...
  ]
}
```

2. `insert` (Ruta: `/user/insert`)

- **Descripción:**
  Permite insertar un nuevo usuario en la base de datos.
- **Parámetros:**
- `user` (objeto, requerido): Datos del usuario a crear.
- **Respuesta:**
  Devuelve el objeto del usuario recién creado o un estado `400` si ocurre un error.

3. `deleteMany` (Ruta: `/user/deleteMany`)

- **Descripción:**
  Elimina múltiples usuarios seleccionados en la interfaz de gestión de usuarios.
- **Parámetros:**
- `users` (array de objetos de usuario, requerido): Usuarios a eliminar.
- **Respuesta:**
  Devuelve el array de usuarios eliminados o un estado `404` si no se encuentran.

4. `getFullNames` (Ruta: `/user/getFullNames`)

- **Descripción:**
  Obtiene una lista de nombres completos de todos los usuarios para mostrar en opciones de selección o listas.
- **Parámetros:**
- Sin parámetros en el body; requiere token de autenticación.
- **Respuesta:**
  Devuelve un array de nombres completos de usuarios o un estado `400` si no se encuentran.

Campos y Requisitos del Backend

- **Tabla `User`**
- `user_id` (numérico, requerido): Identificador único del usuario.
- `email` (alfa-numérico, requerido): Correo electrónico único del usuario.
- `password` (alfa-numérico, requerido): Contraseña del usuario.
- `firstname` (alfabético, requerido): Nombre del usuario.
- `lastname` (alfabético, requerido): Apellido del usuario.
- `role` (numérico, requerido): Rol del usuario (ej. 1 para Cliente, 2 para Doctor, etc.).
- `type` (enum, requerido): Tipo de usuario (`CLIENT`, `DOCTOR`, `DESK`, `ADMIN`).
- **Tabla `Department`**
- `id` (numérico, requerido): Identificador único del departamento.
- `name` (alfabético, requerido): Nombre del departamento.

Módulo: Reporte de Visitas

Funcionalidad Principal:
El módulo de **Reporte de Visitas** permite a los usuarios visualizar estadísticas clave relacionadas con las visitas realizadas a los médicos, proporcionando información sobre la cantidad total de visitas, tasa de cancelación, cantidad de visitas completadas, demografía de pacientes (nuevos y recurrentes), y visitas distribuidas por departamento. La visualización de estos datos se realiza mediante gráficos, permitiendo a los usuarios interpretar la información de manera intuitiva y eficiente.

Componentes del Frontend

1. ReporteVisitas

- **Descripción:**
  Este componente muestra la información de visitas en gráficos de pastel y barras para representar la cantidad total de visitas, el desglose por pacientes nuevos y recurrentes, y las visitas por departamento.
- **Campos y Requisitos:**
- `canceledVisits` (numérico, requerido): Número de visitas canceladas.
- `cancellationRate` (numérico, opcional): Tasa de cancelación calculada en porcentaje.
- `completedVisits` (numérico, requerido): Número de visitas completadas.
- `newPatientsCount` (numérico, requerido): Número de nuevos pacientes.
- `returningPatientsCount` (numérico, requerido): Número de pacientes recurrentes.

- `totalVisits` (numérico, requerido): Número total de visitas.
- `visitCounts` (array de objetos, requerido): Contiene las visitas por departamento.
- **Visualización de Datos:**

1. **Visitas Totales y Cancelaciones:**
   - Representado con un gráfico de pastel que muestra la proporción entre visitas completadas y canceladas.
   - Muestra el total de visitas y el porcentaje de cancelaciones.
1. **Demografía de Pacientes:**
   - Gráfico de pastel que muestra la proporción entre nuevos pacientes y recurrentes.
1. **Visitas por Departamento:**
   - Gráfico de barras que ilustra el número de visitas de cada departamento.

Métodos Backend

1. `report` (Ruta: `/turn/report`)

- **Descripción:**
  Proporciona datos para la generación del reporte de visitas, incluyendo el número total de visitas, visitas canceladas, tasa de cancelación, visitas completadas, cantidad de nuevos y recurrentes pacientes, y visitas por departamento.
- **Parámetros:**
- Sin parámetros en el body; requiere token de autenticación.
- **Respuesta:**
  Devuelve un objeto con la información solicitada para cada sección del reporte:
- `visitCounts`: Array de objetos con el conteo de visitas por departamento.
- `newPatientsCount`: Número de nuevos pacientes.
- `returningPatientsCount`: Número de pacientes recurrentes.
- `totalVisits`: Número total de visitas.
- `canceledVisits`: Número de visitas canceladas.
- `cancellationRate`: Tasa de cancelación en porcentaje.
- `completedVisits`: Número de visitas completadas.
- **Ejemplo de Respuesta:**

```
{
  "visitCounts": [
    {
      "doctorName": "Dr. Juan Perez",
      "department": "Cardiología",
      "visitCount": 45
    },
    ...
  ],
  "newPatientsCount": 20,
  "returningPatientsCount": 75,
  "totalVisits": 100,
  "canceledVisits": 5,
  "cancellationRate": 5.0,
  "completedVisits": 95
}
```

2. `getAvailableDates` (Ruta: `/turn/getAvailableDates`)

- **Descripción:**
  Este método se utiliza para obtener las fechas en las cuales hay visitas programadas con un doctor específico.
- **Parámetros:**
- `doctor_id` (numérico, requerido): ID del doctor del cual se desean obtener las fechas disponibles.
- **Respuesta:**
  Devuelve un array con las fechas de visitas disponibles.

3. `getAvailableTimes` (Ruta: `/turn/getAvailableTimes`)

- **Descripción:**
  Recupera los horarios disponibles para un doctor en una fecha específica.
- **Parámetros:**
- `doctor_id` (numérico, requerido): ID del doctor.
- `selectedDate` (fecha, requerida): Fecha seleccionada para la consulta de horarios.
- **Respuesta:**
  Devuelve un array con los horarios disponibles en el día indicado.

Campos y Requisitos del Backend

- **Tabla `Turno`**
- `id` (numérico, requerido): Identificador único de cada turno.
- `turnoTime` (objeto, requerido): Incluye la fecha (`date`) y hora (`time`) de la visita.
- `cancelation_date` (fecha, opcional): Fecha en que se canceló la visita, si aplica.

- `doctor_id` (numérico, requerido): Identificador del doctor asignado a la visita.
- `client_id` (numérico, requerido): Identificador del cliente asignado a la visita.
- **Tabla `User`**
- `user_id` (numérico, requerido): Identificador único del usuario.
- `firstname` (alfabético, requerido): Nombre del usuario (para reportes con nombres de doctores).
- `lastname` (alfabético, requerido): Apellido del usuario.

Módulo: Reporte de Medicación

Funcionalidad Principal:
El módulo **Reporte de Medicación** permite a los usuarios visualizar datos financieros y operativos clave relacionados con el consumo y el valor de las medicinas en el inventario. Muestra el consumo total de cada medicamento, el costo total derivado de dicho consumo, y el valor actual del inventario, detallado por cada medicamento. La información se presenta mediante gráficos de pastel y de barras que ayudan a interpretar y analizar el uso y la disponibilidad de las medicinas de manera efectiva.

Componentes del Frontend

1. ReporteMedicacion

- **Descripción:**
  Este componente muestra los costos y consumos asociados a las medicinas en el inventario, detallados en gráficos de pastel y barras. Además, se proporciona una tabla con el historial de cambios en el stock de cada medicamento.
- **Campos y Requisitos:**
- `total_cost` (numérico, requerido): Costo total acumulado de los medicamentos consumidos.
- `total_consumption` (numérico, requerido): Consumo total de unidades de medicamento en el mes actual.
- `meds_consumed` (array de objetos, requerido): Lista de medicamentos con su nombre, consumo, y costo.
- `meds_values` (array de objetos, requerido): Lista de medicamentos con su valor total en el inventario.
- `total_value` (numérico, requerido): Valor total del inventario actual.
- `tableData` (array de objetos, requerido): Datos de cambios en el stock de medicamentos.
- **Visualización de Datos:**

1. **Costo Total de Medicinas Consumidas:**
   - Gráfico de pastel que muestra el costo derivado del consumo de cada medicamento en el mes actual.
1. **Consumo Total de Medicamentos:**
   - Gráfico de barras que muestra la cantidad de unidades consumidas de cada medicamento.
1. **Valor Total del Inventario:**
   - Gráfico de pastel que muestra el valor de cada medicamento en el inventario actual.
1. **Historial de Cambios de Stock:**

- Tabla que lista los cambios de stock en cada medicamento, detallando las cantidades previas y nuevas junto con la fecha de modificación.

Métodos Backend

1. `report` (Ruta: `/med/report`)

- **Descripción:**
  Este método genera el reporte de inventario de medicación, calculando el consumo y el costo de cada medicamento en el mes actual, junto con el valor actual del inventario.
- **Parámetros:**
- Sin parámetros en el body; requiere token de autenticación.
- **Respuesta:**
  Devuelve un objeto con los datos calculados para el reporte:
- `total_cost`: Costo total de los medicamentos consumidos.
- `total_consumption`: Cantidad total de medicinas consumidas.
- `meds_consumed`: Lista de medicamentos con consumo y costo detallados.
- `tableData`: Historial de cambios en el stock de cada medicamento.
- `med_value_array`: Lista de valores totales de cada medicamento en el inventario.
- `total_value`: Valor total de todos los medicamentos en inventario.
- **Ejemplo de Respuesta:**

```
{
  "total_cost": 3200,
  "total_consumption": 75,
  "items_consumed": [
    { "name": "Lápiz", "consumption": 20, "cost": 400 },
    { "name": "Cuaderno", "consumption": 55, "cost": 2800 }
  ],
  "stockChangeRows": [
    { "id": 1, "item_id": 102, "previous_quantity": 30, "new_quantity": 10,
"change_date": "2023-09-15T10:00:00Z" }
  ],
  "item_value_array": [
    { "item": { "name": "Lápiz" }, "total_value": 500 },
    { "item": { "name": "Cuaderno" }, "total_value": 1500 }
  ],
  "total_value": 2000
}
```

Módulo: Reporte de Inventario

Funcionalidad Principal:
El módulo **Reporte de Inventario** proporciona un análisis detallado de los artículos en

inventario. Permite visualizar el consumo total de cada artículo, el costo asociado a dicho consumo, y el valor actual de cada artículo en inventario. Los datos se presentan mediante gráficos de pastel y de barras para facilitar el análisis y la toma de decisiones en la gestión de inventario.

Componentes del Frontend

1. ReporteInventario

- **Descripción:**
  Este componente muestra el costo total derivado del consumo de artículos, la cantidad total de unidades consumidas, y el valor actual del inventario. Además, se presenta un historial de cambios en el stock de cada artículo en una tabla interactiva.
- **Campos y Requisitos:**
- `total_cost` (numérico, requerido): Costo total acumulado de los artículos consumidos en el período analizado.
- `total_consumption` (numérico, requerido): Total de unidades consumidas.
- `items_consumed` (array de objetos, requerido): Lista de artículos, detallando nombre, consumo y costo.
- `items_values` (array de objetos, requerido): Lista de artículos con su valor actual en inventario.
- `total_value` (numérico, requerido): Valor total de los artículos disponibles en inventario.
- `tableData` (array de objetos, requerido): Datos históricos de cambios en el stock de cada artículo.
- **Visualización de Datos:**

1. **Costo Total de Artículos Consumidos:**
   - Gráfico de pastel que representa el costo de consumo de cada artículo.
1. **Consumo Total de Artículos:**
   - Gráfico de barras que muestra la cantidad total consumida de cada artículo.
1. **Valor Total del Inventario:**
   - Gráfico de pastel que representa el valor actual de cada artículo en inventario.
1. **Historial de Cambios de Stock:**
   - Tabla que muestra los cambios en el stock de cada artículo, incluyendo cantidades antes y después del cambio, y la fecha de modificación.

Métodos Backend

1. `report` (Ruta: `/inv/report`)

- **Descripción:**
  Este método genera el reporte de inventario, calculando el consumo, costo, y valor actual de cada artículo en el período actual.
- **Parámetros:**
- No requiere parámetros en el cuerpo de la solicitud; se requiere autenticación.
- **Respuesta:**
  Devuelve un objeto con los datos calculados para el reporte de inventario:
- `total_cost`: Costo total de artículos consumidos.
- `total_consumption`: Cantidad total de artículos consumidos.

- `items_consumed`: Lista de artículos con detalles de consumo y costo.
- `stockChangeRows`: Historial de cambios en el stock de artículos.
- `item_value_array`: Lista de artículos con el valor total de cada uno.
- `total_value`: Valor total actual del inventario.
- **Ejemplo de Respuesta:**

```
{
  "total_cost": 3200,
  "total_consumption": 75,
  "items_consumed": [
    { "name": "Lápiz", "consumption": 20, "cost": 400 },
    { "name": "Cuaderno", "consumption": 55, "cost": 2800 }
  ],
  "stockChangeRows": [
    {
      "id": 1,
      "item_id": 102,
      "previous_quantity": 30,
      "new_quantity": 10,
      "change_date": "2023-09-15T10:00:00Z"
    }
  ],
  "item_value_array": [
```

Campos y Requisitos del Backend

- **Tabla `Item`**
- `item_id` (numérico, requerido): Identificador único de cada artículo.
- `name` (texto, requerido): Nombre del artículo.
- `unit_price` (numérico, requerido): Precio por unidad del artículo.
- `stock` (objeto, requerido): Relación con `ItemStock` que incluye la cantidad disponible.
- **Tabla `ItemStock`**
- `quantity` (numérico, requerido): Cantidad actual de unidades en inventario.
- **Tabla `ItemStockChanges`**
- `id` (numérico, requerido): Identificador único de cada cambio en el stock.
- `previous_quantity` (numérico, requerido): Cantidad en stock antes del cambio.
- `new_quantity` (numérico, requerido): Nueva cantidad después del cambio.
- `change_date` (fecha, requerida): Fecha del cambio.

Protocolo de Seguridad en el Componente AuthProvider: Funcionalidad, Beneficios y Significado

El componente `AuthProvider` implementa un protocolo de seguridad robusto y en múltiples capas, diseñado para gestionar y proteger la autenticación de usuarios en toda la aplicación. Centraliza funcionalidades relacionadas con autenticación, tales como registro, inicio de sesión, cierre de sesión y administración de tokens de actualización, mejorando la seguridad y la

eficiencia. Esta configuración no solo simplifica la experiencia del usuario, sino que también refuerza significativamente el sistema contra accesos no autorizados y el uso indebido de tokens.

Principales Funcionalidades:

1. **Gestión de Autenticación Basada en Contexto**
   Utilizando `AuthContext`, el componente `AuthProvider` proporciona una gestión centralizada de autenticación para toda la aplicación, permitiendo que todos los componentes accedan y actualicen el estado de autenticación de forma consistente. Esto garantiza que todas las solicitudes y respuestas manejen la autenticación de forma segura y sin código redundante, incrementando tanto la seguridad como la mantenibilidad.
2. **Autenticación Basada en Tokens**
   El sistema usa cookies `access_token` y `refresh_token` para gestionar la persistencia de sesiones de forma segura. El `access_token` es un token de corta duración, que se renueva automáticamente para mantener la continuidad de la sesión minimizando el riesgo de robo o mal uso. El `refresh_token` tiene una vida útil más prolongada, permitiendo una reautenticación segura sin necesidad de que los usuarios ingresen sus credenciales con frecuencia.
3. **Cookies HTTP-Only para Almacenamiento de Tokens**
   Los tokens se almacenan en cookies HTTP-only, lo que impide que los scripts del lado del cliente puedan acceder a ellos. Esto reduce drásticamente la superficie de ataque contra posibles ataques de Cross-Site Scripting (XSS), ya que los tokens no pueden ser accedidos o manipulados por JavaScript malicioso.
4. **Renovación Automática de Tokens**
   La función `initializeUser` renueva el `access_token` cada vez que se inicializa la aplicación, brindando una experiencia fluida a la vez que asegura que los usuarios permanezcan autenticados de manera segura. El endpoint del backend `/auth/refresh` valida el `refresh_token` existente, emite un nuevo `access_token` y renueva el `refresh_token` si es necesario.
5. **Cierre de Sesión Seguro y Gestión de Cookies**
   La función `logout` llama a un endpoint del backend para limpiar los tokens en el servidor y elimina la información de usuario local. Esto asegura que las sesiones se terminen completamente y de manera segura, minimizando el riesgo de acceso no autorizado. Adicionalmente, la función `clearAuthCookies` del backend borra todas las cookies, asegurando que no quede ningún dato de autenticación en el lado del cliente después del cierre de sesión.

Beneficios:

Seguridad Mejorada
El uso de cookies HTTP-only y un enfoque de token en capas (access y refresh tokens) ofrece una protección robusta contra vulnerabilidades comunes como el XSS **Protocolo de Seguridad en el Componente `AuthProvider`: Funcionalidad, Beneficios y Significado**

El componente `AuthProvider` implementa un protocolo de seguridad robusto y en múltiples capas, diseñado para gestionar y proteger la autenticación de usuarios en toda la aplicación. Centraliza funcionalidades relacionadas con autenticación, tales como registro, inicio de sesión,

cierre de sesión y administración de tokens de actualización, mejorando la seguridad y la eficiencia. Esta configuración no solo simplifica la experiencia del usuario, sino que también refuerza significativamente el sistema contra accesos no autorizados y el uso indebido de tokens.

Principales Funcionalidades:

1. **Gestión de Autenticación Basada en Contexto**
   Utilizando `AuthContext`, el componente `AuthProvider` proporciona una gestión centralizada de autenticación para toda la aplicación, permitiendo que todos los componentes accedan y actualicen el estado de autenticación de forma consistente. Esto garantiza que todas las solicitudes y respuestas manejen la autenticación de forma segura y sin código redundante, incrementando tanto la seguridad como la mantenibilidad.
2. **Autenticación Basada en Tokens**
   El sistema usa cookies `access_token` y `refresh_token` para gestionar la persistencia de sesiones de forma segura. El `access_token` es un token de corta duración, que se renueva automáticamente para mantener la continuidad de la sesión minimizando el riesgo de robo o mal uso. El `refresh_token` tiene una vida útil más prolongada, permitiendo una reautenticación segura sin necesidad de que los usuarios ingresen sus credenciales con frecuencia.
3. **Cookies HTTP-Only para Almacenamiento de Tokens**
   Los tokens se almacenan en cookies HTTP-only, lo que impide que los scripts del lado del cliente puedan acceder a ellos. Esto reduce drásticamente la superficie de ataque contra posibles ataques de Cross-Site Scripting (XSS), ya que los tokens no pueden ser accedidos o manipulados por JavaScript malicioso.
4. **Renovación Automática de Tokens**
   La función `initializeUser` renueva el `access_token` cada vez que se inicializa la aplicación, brindando una experiencia fluida a la vez que asegura que los usuarios permanezcan autenticados de manera segura. El endpoint del backend `/auth/refresh` valida el `refresh_token` existente, emite un nuevo `access_token` y renueva el `refresh_token` si es necesario.
5. **Cierre de Sesión Seguro y Gestión de Cookies**
   La función `logout` llama a un endpoint del backend para limpiar los tokens en el servidor y elimina la información de usuario local. Esto asegura que las sesiones se terminen completamente y de manera segura, minimizando el riesgo de acceso no autorizado. Adicionalmente, la función `clearAuthCookies` del backend borra todas las cookies, asegurando que no quede ningún dato de autenticación en el lado del cliente después del cierre de sesión.

Beneficios:

1. **Seguridad Mejorada**
   El uso de cookies HTTP-only y un enfoque de token en capas (access y refresh tokens) ofrece una protección robusta contra vulnerabilidades comunes como el XSS y el robo de tokens. Además, permite una validación continua de las sesiones de usuario, asegurando que solo los usuarios autenticados puedan acceder a áreas seguras de la aplicación.
2. **Reducción de la Superficie de Ataque**
   Almacenar los tokens de forma segura en cookies HTTP-only y renovarlos automáticamente minimiza el riesgo de exposición de los tokens. Al aislar la gestión de

tokens del JavaScript del lado del cliente, este protocolo limita significativamente los vectores disponibles para ataques basados en tokens.

3. **Mejor Experiencia de Usuario**
   La gestión fluida de la renovación de tokens asegura que los usuarios permanezcan conectados sin interrupciones frecuentes de reautenticación. El mecanismo de renovación de tokens, integrado en `initializeUser`, permite que los usuarios continúen su sesión sin comprobaciones de autenticación visibles, mejorando la usabilidad.
4. **Lógica de Autenticación Centralizada**
   Centralizar la lógica de autenticación en `AuthProvider` permite un código más mantenible y menos propenso a errores, ya que todos los componentes dependen del mismo estado y métodos de autenticación. Esto reduce la posibilidad de errores de seguridad o inconsistencias en la gestión de sesiones a lo largo de la aplicación.
5. **Adaptabilidad y Escalabilidad**
   El diseño de `AuthProvider` permite realizar ajustes fácilmente en las políticas de autenticación, como los tiempos de expiración de tokens o los mecanismos de renovación, haciéndolo adaptable a estándares de seguridad en evolución. El uso de autenticación basada en contexto y funciones modulares también respalda el crecimiento escalable para aplicaciones de mayor tamaño.

Significado para el Sistema:

1. Implementar este protocolo de seguridad a través de `AuthProvider` agrega una capa esencial de protección y fiabilidad, convirtiéndolo en un activo valioso en cualquier aplicación donde la autenticación de usuarios y la privacidad de los datos son primordiales. Este protocolo no solo se alinea con las mejores prácticas de la industria en seguridad, sino que también mejora la experiencia de usuario, fomentando la confianza en el compromiso del sistema con la protección de datos. Este protocolo bien estructurado asegura que los usuarios puedan interactuar con la aplicación de manera segura, sabiendo que su sesión es gestionada y protegida cumpliendo con altos estándares de seguridad.y el robo de tokens. Además, permite una validación continua de las sesiones de usuario, asegurando que solo los usuarios autenticados puedan acceder a áreas seguras de la aplicación.
2. **Reducción de la Superficie de Ataque**
   Almacenar los tokens de forma segura en cookies HTTP-only y renovarlos automáticamente minimiza el riesgo de exposición de los tokens. Al aislar la gestión de tokens del JavaScript del lado del cliente, este protocolo limita significativamente los vectores disponibles para ataques basados en tokens.
3. **Mejor Experiencia de Usuario**
   La gestión fluida de la renovación de tokens asegura que los usuarios permanezcan conectados sin interrupciones frecuentes de reautenticación. El mecanismo de renovación de tokens, integrado en `initializeUser`, permite que los usuarios continúen su sesión sin comprobaciones de autenticación visibles, mejorando la usabilidad.
4. **Lógica de Autenticación Centralizada**
   Centralizar la lógica de autenticación en `AuthProvider` permite un código más mantenible y menos propenso a errores, ya que todos los componentes dependen del mismo estado y métodos de autenticación. Esto reduce la posibilidad de errores de seguridad o inconsistencias en la gestión de sesiones a lo largo de la aplicación.

5. **Adaptabilidad y Escalabilidad**
   El diseño de `AuthProvider` permite realizar ajustes fácilmente en las políticas de autenticación, como los tiempos de expiración de tokens o los mecanismos de renovación, haciéndolo adaptable a estándares de seguridad en evolución. El uso de autenticación basada en contexto y funciones modulares también respalda el crecimiento escalable para aplicaciones de mayor tamaño.

Significado para el Sistema:

Implementar este protocolo de seguridad a través de `AuthProvider` agrega una capa esencial de protección y fiabilidad, convirtiéndolo en un activo valioso en cualquier aplicación donde la autenticación de usuarios y la privacidad de los datos son primordiales. Este protocolo no solo se alinea con las mejores prácticas de la industria en seguridad, sino que también mejora la experiencia de usuario, fomentando la confianza en el compromiso del sistema con la protección de datos. Este protocolo bien estructurado asegura que los usuarios puedan interactuar con la aplicación de manera segura, sabiendo que su sesión es gestionada y protegida cumpliendo con altos estándares de seguridad.