TP 2 Desarrollo de sistemas:
Lo de Miguel

Integrantes:

- Bruno Enzo Malvasio
- Santino García Corsaro

Profesor:

- Manuel Corales

Fecha de entrega: 20/6

Índice:

Requerimientos:

- Menú completo de platos disponibles para el cliente
- Que el cliente pueda consultar el menú, reservar mesa, revisar su estado y realizar un pedido.
- Que el cliente/administrador se pueda registrar
- Que el administrador puede agregar/eliminar platos
- Que el administrador pueda manejar el estado de los pedidos/mesas
- Que el cliente tenga descuentos de acuerdo a la cantidad de pedidos realizados
- Que las mesas tengan estados que cambian de “pendiente” a “en cocina” y luego a “enviado”.

Endpoints de la API:

Autentificación:

Servicio:

El servicio de autentificación está compuesto por la clase “userVerificationService”, con el fin de verificar los datos ingresados por el usuario y generar el token de sesión para otorgarle al usuario que está ingresando en el sistema. Y además aquí se implementa la función “getUserBySessionToken” para hallar usuarios a partir de la session token guardad en sesiones.

La función “verifyUser” toma como parámetros el mail del usuario y la contraseña, este se fija en la base de datos, buscando el perfil de usuario de acuerdo al email ingresado y compara la contraseña almacenada con la que está ingresando el usuario. Si se verifica correctamente devuelve el usuario. Por otro lado está “generateUserSession” el cual genera un token a partir de un random uuid y lo inserta en la tabla de sesiones, asociado al id de usuario que la misma función recibe como parámetro.

Router:

El elemento router de la autentificación se encarga de realizar las acciones de registro y logueo de usuario. El registro se utiliza para registrar todos los datos personales del usuario, como puede ser la dirección, teléfono, o la contraseña. Luego

procede a crear la cuenta e ingresarla en la base de datos y logear automáticamente al usuario.
Mientras que la función de “loggear” solamente toma el email y contraseña del usuario para buscarlo en la base de datos y verificar los datos ingresados antes de otorgarle acceso a la cuenta.

Método POST/register:

Request body:

```
{
  "nombre": "Juan Pérez",
  "telefono": "1122334455",
  "email": "juanperez@example.com",
  "contrasena": "12345678",
  "direccion": "Calle Falsa lolo"
}
```

Respuestas:

- Mensaje 201 creado:

```
{
  "data": "id_del_usuario"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Mensaje de error"
```

```
}
```

Método POST/login:

Request body:

```
{
  "email": "juanperez@example.com",
  "contrasena": "123456"
}
```

Respuestas:

- Mensaje 200 ok:

```
{
  "data": "token_de_sesion"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Mensaje de error"
}
```

Mesa:

Servicio:

Implementa la clase “mesaService”, con las funciones de “getMesas”, “getMesasByUser”, “getMesaById”, así como “createMesa”. “getMesas” simplemente devuelve todas las

mesas en la base de datos. “getMesasByUser” devuelve, si está asociada, una mesa a un usuario, tomando como parámetro el id del usuario el cual se quiere buscar. La función “getMesaById” extrae una mesa de acuerdo a su identificador, tomando como parametro en número de la mesa. Y finalmente “createMesa” crea un nuevo objeto mesa y lo almacena en la base de datos.

Así como las funciones “updateEstadoMesa” y “deleteMesa” para actualizar el estado de la mesa y borrarla del sistema si es necesario, pasando como parámetro el Id de la mesa.

Router:

El router de las mesas maneja el acceso a ellas y su estado de acuerdo a la información que está pidiendo el cliente o administrador. Cada cliente puede ver la disponibilidad de todas las mesas en el restaurante para reservar solamente aquellas que estén disponibles. Mientras que el administrador puede agregar nuevas mesas al sistema y modificar su estado de “disponible” a “reservada”. Así como poder ver las mesas reservadas al nombre de un determinado cliente, gracias a que cada mesa guarda el id del cliente que la reservó.

Respuestas:

Método GET:

- Mensaje 201 ok:

```
{
  "mesas": [
```

```
    {
      "id": "1",
      "estado": "disponible",
      "id_user": null
    },
    ...
  ]
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Internal server error"
}
```

Método GET/user/:id:

Respuestas:

- Mensaje 200 ok:

```
{
  "mesas": [
    {
      "id": "3",
      "estado": "reservada",
      "id_user": "abc123"
    },
    ...
  ]
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Internal server error"
}
```

Método GET/:id:

- Mensaje 201 ok:

```
{
  "mesas": [
    {
      "id": "3",
      "estado": "reservada",
      "id_user": "abc123"
    },
    ...
  ]
}
```

- Mensaje 404 no encontrado:

```
{
  "error": "Mesa not found"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Internal server error"
}
```

Método POST:

- Requiere pasar por la autorización teniendo el token de la sesión.
- Requiere que el usuario sea administrador.

Request Body:

```
{
  "estado": "disponible"
}
```

Respuestas:

- Mensaje 201 ok:

```
{
  "mesa": {
    "id": "10",
    "estado": "disponible",
    "id_user": "cliente123"
  }
}
```

- Mensaje 403 prohibido:

```
{
```

```
  "error": "No tienes permisos para crear una mesa"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Mensaje de error"
}
```

Método PATCH/:id:

- Requiere pasar por la autorización teniendo el token de la sesión.

Request body:

```
{
  "estado": "Disponible" // Opcional
}
```

Respuestas:

- Mensaje 200 ok:

```
{
  "mesa": {
    "id": "5",
    "estado": "Ocupada",
    "id_user": "user123"
  }
}
```

- Mensaje 400 mesa ocupada:

```
{
  "error": "La mesa ya está ocupada"
}
```

- Mensaje 404 no encontrado:

```
{
  "error": "Mesa no encontrada"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Mensaje de error"
}
```

Método DELETE/:id:

- Requiere pasar por la autorización teniendo el token de la sesión.
- Requiere que el usuario sea administrador.

Respuestas:

- Mensaje 200 ok:

```
{
  "mensaje": "Mesa eliminada correctamente",
  "mesa": {
    "id": "7",
    "estado": "Disponible",
    ...
```

```
  }
}
```

- Mensaje 403 prohibido:

```
{
  "error": "No tienes permisos para eliminar una mesa"
}
```

- Mensaje 404 no encontrado:

```
{
  "error": "Mesa no encontrada"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Mensaje de error"
}
```

Pedido:

Servicio:

Incluye funciones básicas como “getPedidos” y “createPedidos” las cuales devuelven todos los pedidos del sistema y suma un nuevo pedido a la tabla, al crear los pedidos también se calcula el monto total de acuerdo a los precios de los platos en el contenido del pedido y se le aplica el descuento si el cliente cumple con las condiciones.

Después las funciones “getPedidoByUser” y “getPedidoById” son para obtener los pedidos hechos por un cliente y buscar un pedido por un id específico respectivamente.

En este apartado también se declaran las funciones “createPlatoMenu” y “getPlatosMenu” para agregar opciones de platos al sistema y sacarlos para mostrarselos al cliente.

Finalmente las últimas tres funciones las cuales también son exclusivamente para administradores son “updateEstadoPedido”, “deletePedido” y “deletePlato” para cambiar el estado del pedido del cliente y borrar un pedido/plato del menú en caso que sea necesario, pidiendo como parámetro el id de lo que se desea borrar.

Router:

El router de los pedidos maneja todos los caminos y endpoints relacionados a los pedidos que puede realizar el usuario y como los administradores pueden procesarlos.

Método GET:

Respuestas:

- Mensaje 201 ok:

```
{
  "pedidos": [
    {
      "id": "123",
      "estado": "en cocina",
      "monto_total": 3500,
      ...
```

```
    },
    ...
  ]
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Internal server error"
}
```

Método GET /user/:id:

Respuestas:

- Mensaje 200 ok:

```
{
  "pedidos": [
    {
      "id": "001",
      "estado": "entregado",
      "monto_total": 4200,
      ...
    }
  ]
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Internal server error skibidi"
}
```

Método GET /:id:

Respuestas:

- Mensaje 200 ok:

```
{
  "id": "0001",
  "estado": "pendiente",
  "platos": [...],
  "monto_total": 2300,
  "porcentaje_descuento": 0,
  ...
}
```

- Mensaje 404 no encontrado:

```
{
  "error": "Pedido not found"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Internal server error sigma"
}
```

Método POST:

- Requiere pasar por la autorización teniendo el token de la sesión.
- Requiere que el usuario sea administrador.

Request body (body del pedido):

```
{
  "title": "Almuerzo completo",
  "contenido": [
    { "id": "plato1" },
    { "id": "plato2" },
    { "id": "plato3" }
  ]
}
```

Respuestas:

- Mensaje 200 ok:

```
{
  "pedido": {
    "id": "xyz",
    "estado": "pendiente",
    "monto_total": 2000,
    "porcentaje_descuento": 0,
    ...
  }
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Mensaje de error"
}
```

Método POST /plato:

- Requiere pasar por la autorización teniendo el token de la sesión.
- Requiere que el usuario sea administrador.

Request body:

```
{
  "nombre": "Milanesa con papas",
  "descripcion": "Milanesa de carne con papas fritas",
  "precio": 2500,
  "categoria": "plato principal"
}
```

Respuestas:

- Mensaje 200 ok:

```
{
  "plato": {
    "id": "abc",
    "nombre": "Milanesa con papas",
    ...
  }
}
```

- Mensaje 404 prohibido:

```
{
  "error": "No tienes permisos para crear un plato"
}
```

- Mensaje 500 error interno del servidor:

```
{
```

```
  "error": "Mensaje de error"
}
```

Método PATCH/:id/estado:

- Requiere pasar por la autorización teniendo el token de la sesión.
- Requiere que el usuario sea administrador.

Request body:

```
{
  "estado": "en cocina"
}
```

Respuestas:

- Mensaje 200 ok:

```
{
  "pedido": {
    "id": "001",
    "estado": "en cocina",
    ...
  }
}
```

- Mensaje 400 error de request:

```
{
  "error": "El campo 'estado' es requerido"
}
```

- Mensaje 403 prohibido:

```
{
  "error": "No tienes permisos para modificar el
estado del pedido"
}
```

- Mensaje 404 no encontrado:

```
{
  "error": "Pedido no encontrado"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Mensaje de error"
}
```

Método DELETE/:id:

- Requiere pasar por la autorización teniendo el token de la sesión.
- Requiere que el usuario sea administrador.

Respuestas:

- Mensaje 200 ok:

```
{
  "mensaje": "Pedido eliminado correctamente",
  "pedido": {
    "id": "001",
    ...
  }
}
```

- Mensaje 403 prohibido:

```
{
  "error": "No tienes permisos para eliminar un
pedido"
}
```

- Mensaje 404 no encontrado:

```
{
  "error": "Pedido no encontrado"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Mensaje de error"
}
```

Usuario:

Servicio:

Implementa funciones para sacar datos relacionados a los usuarios con “getUsuarios”, “getUsuarioByEmail” y “getUsuarioById”, con el propósito de registrar nuevos usuarios o buscarlos en la base de datos para registrarlos, tomando como parámetro ya sea el email o Id del usuario el cual se quiere buscar.

Además la función “createUsuario” se declara acá y valida las credenciales del usuario para poder registrarlo en la base de datos.

Router:

Genera un listado de todos los usuarios guardados en la base de datos, ya sean clientes del restaurante o administradores. Así como también declara funciones utilizadas para logear a los usuarios al sistema.

Método GET:

Respuestas:

- Mensaje 200 ok:

```
{
  "users": [
    {
      "id": "abc123",
      "nombre": "Juan Pérez",
      "email": "juan@example.com",
      "telefono": "1122334455",
      "direccion": "Calle Falsa lolo",
      "es_admin": false,
      "cant_pedidos": 4
    },
    ...
  ]
}
```

- Mensaje 500 error interno del servidor:

```
{
```

```
  "error": "Internal server error toilet"
}
```

Método GET/:id:

Respuestas:

- Mensaje 200 ok:

```
{
  "id": "abc123",
  "nombre": "Juan Pérez",
  "email": "juan@example.com",
  "telefono": "1122334455",
  "direccion": "Calle Falsa lolo",
  "es_admin": false,
  "cant_pedidos": 4
}
```

- Mensaje 404 no encontrado:

```
{
  "error": "User not found"
}
```

- Mensaje 500 error interno del servidor:

```
{
  "error": "Internal server error"
}
```