

# EXAMEN APERTURA DE RESTAURANTES

La empresa ha decidido ofrecer a los propietarios la posibilidad de cambiar manualmente el estado de sus restaurantes abiertos (online u offline).

En caso de que un restaurante tenga un estado online u offline, la nueva funcionalidad proporcionaría un botón para alternar entre ambos estados. De lo contrario, dicho botón no debe estar disponible.

Tenga en cuenta que cualquier restaurante nuevo se almacena inicialmente como offline de forma predeterminada. Solo los restaurantes offline pueden estar en línea y viceversa.

Por otro lado, si un restaurante alcanza un estado de cerrado o cerrado temporalmente, la posibilidad de cambio manual no estará disponible y su uso estará prohibido por el sistema.

Además, tenga en cuenta que un restaurante tampoco podrá cambiar su estado si tiene pedidos con un valor nulo en deliveredAt.


La nueva funcionalidad también implicará proporcionar la lista de restaurantes propiedad del usuario ordenados por estado (ascendentemente) y, para el mismo estado, por nombre.

Finalmente, el estado de cada restaurante debe ser visible.

El sistema debe mostrar estos requisitos como se muestran en las siguientes capturas de pantalla:

My Restaurants

Create restaurant




Hola

This restaurant is **online**

Shipping: 1.00€

EditDeleteoffline




1 producto

1 producto

This restaurant is **offline**

Shipping: 1.50€

EditDeleteonline



100 montaditos


Una forma divertida y variada de disfrutar de la comida. Un lugar para compartir experiencias y dejarse llevar por el momento.

This restaurant is **offline**

Shipping: 1.50€

EditDeleteonline

My Restaurants




30 productos

1000 productos

This restaurant is **offline**

Shipping: 1.50€

EditDeleteonline




50 productos

1000 productos

This restaurant is **offline**

Shipping: 1.50€

EditDeleteonline




0 productos

0 productos

This restaurant is **closed**

Shipping: 1.50€

EditDelete



Casa Félix

Cocina Tradicional

This restaurant is **closed**

Shipping: 2.50€

EditDelete


My Restaurants

Control Panel

Profile

There was an error while trying to toggle status. Error: Forbidden

Create restaurant



100 montaditos

Una forma divertida y variada de disfrutar de la comida. Un lugar para compartir experiencias y dejarse llevar por el momento.

This restaurant is **online**

Shipping: 1.50€

EditDeleteoffline

# BACKEND

## ¿ Sobre qué modelo vamos a trabajar ?

- Estamos trabajando sobre el modelo Restaurant, el cual ya tiene definida la propiedad status, luego no tendremos que añadir una nueva propiedad al modelo.

```
status: {  
  type: DataTypes.ENUM,  
  values: [  
    'online',  
    'offline',  
    'closed',  
    'temporarily closed'  
  ]  
},
```

- Además, trabajaremos con el modelo de Order, pues dice que si tiene la propiedad deliveredAt nula, no podremos implementar la nueva función.

```
deliveredAt: DataTypes.DATE,
```

## ¿ Qué tenemos que hacer ?

- Si leemos el enunciado:

*“La empresa ha decidido ofrecer a los propietarios la posibilidad de cambiar manualmente el estado de sus restaurantes abiertos (online u offline).”*

- Nos están indicando que la nueva funcionalidad es solo para los propietarios, luego en la nueva ruta que hagamos tenemos que verificar que el usuario sea owner.

*“En caso de que un restaurante tenga un estado online u offline, la nueva funcionalidad proporcionaría un botón para alternar entre ambos estados. De lo contrario, dicho botón no debe estar disponible”*

- Tenemos que crear la lógica que altere el estado del restaurante al pulsar el botón, tendremos que hacerlo en el RestaurantController, pues ahí es donde esta toda la lógica de ese modelo.

*“Tenga en cuenta que cualquier restaurante nuevo se almacena inicialmente como offline de forma predeterminada. Solo los restaurantes offline pueden estar en línea y viceversa. Por otro lado, si un restaurante alcanza un estado de cerrado o cerrado temporalmente, la posibilidad de cambio manual no estará disponible y su uso estará prohibido por el sistema. Además, tenga en cuenta que un restaurante tampoco podrá cambiar su estado si tiene pedidos con un valor nulo en deliveredAt.”*

- Tenemos que crear la restricción que impida cambiar el estado de aquellos que no se encuentran en online u offline.
- Además de la restricción que impide que si deliveredAt es nula tampoco se puede cambiar el estado.
- Ambas restricciones las añadimos al Middleware de Restaurant.

La nueva funcionalidad también implicará proporcionar la lista de restaurantes propiedad del usuario ordenados por estado (ascendentemente) y, para el mismo estado, por nombre.

- Tenemos que mostrar el listado de restaurantes en orden ascendente según el estado y después por el nombre, luego las funciones show del restaurantController deben tener ese orden específico.

Finalmente, el estado de cada restaurante debe ser visible.

- Esto es, en el frontend, debemos mostrar el estado de cada restaurante.

### Paso 1: Crear la lógica de la nueva función a implementar

Nueva función en un modelo → nueva función con la lógica en el controlador de ese modelo.

Cual es la lógica en este caso:

- Si un restaurante tiene estado Online poder ponerlo Offline, y viceversa. Esta lógica afecta exclusivamente a aquellos restaurantes con esos dos estados.
- Además estamos actualizando parcialmente el modelo de Restaurant, luego se trata de hacer una operación patch con una nueva ruta.

Una opción sin transacciones puede ser:

- Tenemos que:
  - Actualizar parcialmente: petición (request) para actualizar el restaurante con id dado en la ruta (params.restaurantId)

```
const alterStatus = async function (req, res) {
  const t = await sequelizeSession.transaction()
  try {
    // Buscamos el restaurante que vamos a actualizar con la petición
    const RestaurantToUpdate = await
Restaurant.findByPk(req.params.restaurantId)
    // si su estado es offline cambia a online, en otro caso a offline
    if (RestaurantToUpdate.status === 'online') {
      RestaurantToUpdate.status = 'offline'
    } else {
      RestaurantToUpdate.status = 'online'
    }
    // guardamos la nueva informacion del restaurante en la BD
    await RestaurantToUpdate.save({ transaction: t })
    await t.commit()
    res.json(RestaurantToUpdate)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

Tenemos que exportar la función, para luego añadirla a la ruta.

## Paso 2: Añadir las nuevas restricciones:

Tenemos que tener en cuenta que no se podrá alterar el estado si:

- 1) *Un restaurante alcanza un estado de cerrado o cerrado temporalmente, la posibilidad de cambio manual no estará disponible y su uso estará prohibido por el sistema.*
- 2) *Además, tenga en cuenta que un restaurante tampoco podrá cambiar su estado si tiene pedidos con un valor nulo en deliveredAt.*

Luego este tipo de restricciones que nos impiden alterar el modelo de la BD las añadimos a los middlewares:

(1) Condición de no estado closed ni temporarily closed:

```
const restaurantHasInvalidStatus = async (req, res, next) => {
  try {
    // Buscamos el restaurante que vamos a actualizar con la petición
    const restaurant = await
Restaurant.findByPk(req.params.restaurantId)
    // Si el estado de ese Restaurante es closed, retornamos un error
    if (restaurant.status === 'closed') {
      return res.status(409).send('This Restaurant is closed')
    }
    // Si el estado de ese Restaurante es temporarily closed,
retornamos un error
    if (restaurant.status === 'temporarily closed') {
      return res.status(409).send('This Restaurant is temporarily
closed')
    }
    return next()
  } catch (err) {
    return res.status(500).send(err)
  }
}
```

(2) Condición que le Restaurante no tenga Orders con deliveredAt nulos:

```
const restaurantHasNoPendingOrders = async (req, res, next) => {
  try {
    // Buscamos todos los orders del restaurante que vamos a actualizar
    con la petición
    const orders = await Order.findAll({
      where: { restaurantId: req.params.restaurantId }
    })
    // Para cada order, verificamos que no tiene un deliveredAt nulo
    for (const order of orders) {
      if (order.deliveredAt === null) {
        return res.status(409).send('There are pending orders')
      }
    }
    return next()
  } catch (err) {
    return res.status(500).send(err)
  }
}
```

### Paso 3: Añadir el orden en el que se muestran los restaurantes.

La nueva funcionalidad también implicará proporcionar la lista de restaurantes propiedad del usuario ordenados por estado (ascendentemente) y, para el mismo estado, por nombre.

Luego editamos la función `index` e `indexOwner` del `RestaurantController` que muestra los restaurantes:

```
const index = async function (req, res) {
  try {
    const restaurants = await Restaurant.findAll(
      {
        attributes: { exclude: ['userId'] },
        include:
        {
          model: RestaurantCategory,
          as: 'restaurantCategory'
        },
        order: [['status', 'ASC'], [{ model: RestaurantCategory, as:
'restaurantCategory' }, 'name', 'ASC']]
      }
    )
    res.json(restaurants)
  } catch (err) {
    res.status(500).send(err)
  }
}

const indexOwner = async function (req, res) {
  try {
    const restaurants = await Restaurant.findAll(
      {
        attributes: { exclude: ['userId'] },
        where: { userId: req.user.id },
        include: [{
          model: RestaurantCategory,
          as: 'restaurantCategory'
        }],
        order: [['status', 'ASC'], [{ model: RestaurantCategory, as:
'restaurantCategory' }, 'name', 'ASC']]
      }
    )
    res.json(restaurants)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

#### Paso 4: Añadir la nueva ruta:

Como vamos a crear una nueva función tendremos que crear una nueva ruta que permita hacer la actualización parcial, a la cual luego llamaremos en los endpoints del frontend para que funcione el cambio de status:

Luego en la ruta de los Restaurantes añadimos todo lo que hemos creado, siguiendo el orden de: validaciones de usuario, ya que especifica que sirve para los propietarios, luego los middlewares, si hay validation después de estos y por último la nueva función que permite ese cambio de status.

Quedando la ruta:

```
app.route('/restaurants/:restaurantId/alterStatus')
  .patch(
    isLoggedIn,
    hasRole('owner'),
    checkEntityExists(Restaurant, 'restaurantId'),
    RestaurantMiddleware.checkRestaurantOwnership,
    RestaurantMiddleware.restaurantHasNoPendingOrders,
    RestaurantMiddleware.restaurantHasInvalidStatus,
    RestaurantController.alterStatus
  )
```

Y con esto terminaremos con la parte de backend.



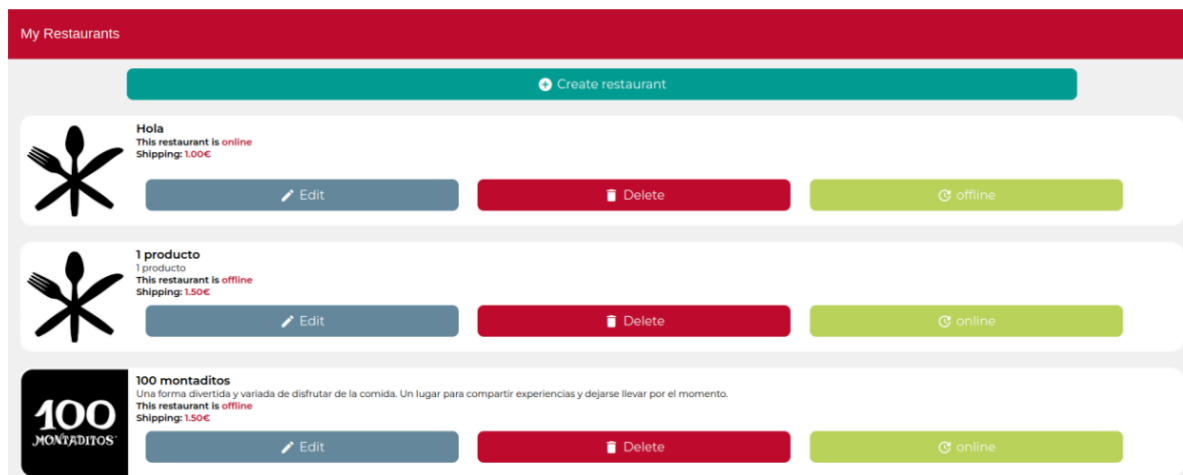
# FRONTEND

Una vez que hemos creado la nueva función, lo primero que debemos hacer es añadirla en el endpoint de Restaurant la llamada a esa función:

```
function alterStatus (id) {  
  return patch(`restaurants/${id}/alterStatus`)  
}
```

Tenemos que importar en la zona del import el método que vamos a utilizar (patch) y exportar la nueva función.

A partir de aquí, y con ayudas de las fotos podemos saber donde tenemos que poner cada cosa:



Estamos en la RestaurantScreen, tenemos que:

1. Añadir el botón:
  - a. Crear el useState que cambiara cuando pinchemos el botón.
2. Crear la función que asociaremos al botón para actualizar el estado del restaurante.

Paso 1: Añadir el botón:

Primero, añadimos el useState para cambiar el status del restaurante:

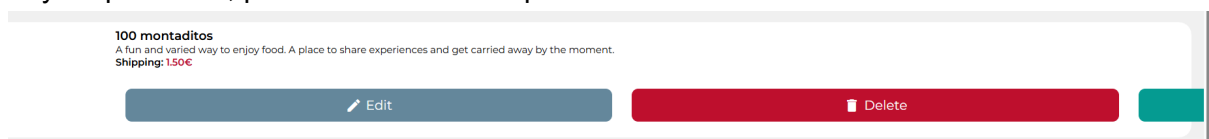
```
const [restaurantToBeChangedItsStatus, setRestaurantToBeChangedItsStatus]  
= useState(null)
```

Luego añadimos el botón:

- Debemos tener en cuenta que el botón nuevo solo debe aparecer cuando el estado de los restaurantes es online u offline, luego añadimos la condición al botón:

```
// CONDICIÓN
{ (item.status === 'online' || item.status === 'offline') &&
  <Pressable
    onPress={() => { setRestaurantToBeChangedItsStatus(item) }}
    style={({ pressed }) => [
      {
        backgroundColor: pressed
          ? GlobalStyles.brandGreenTap
          : GlobalStyles.brandGreen
      },
      styles.actionButton
    ]}>
    <View style={[{ flex: 1, flexDirection: 'row',
justifyContent: 'center' }]}>
      <MaterialCommunityIcons name='clock' color={'white'}
size={20}/>
      <TextRegular textStyle={styles.text}>
        {item.status}
      </TextRegular>
    </View>
  </Pressable>}
```

Hay un problema, pues ahora mismo la pantalla de nuestra web se ve así:



Tenemos que cambiar el estilo del botón para que no salga de la pantalla, para ello, si nos fijamos en el botón que hemos definido, estamos usando el `actionButton`, luego cambiamos el `width` de 90 a 62 y añadimos el `alignSelf = 'start'`

```
actionButtonsContainer: {
  flexDirection: 'row',
  bottom: 5,
  position: 'absolute',
  width: '62%', // SOLUCION
  alignSelf: 'start' // SOLUCION
},
```

Una vez creado el botón, tenemos que añadir la función que actualize, para ello importamos al restaurantScreen la nueva función que creamos antes en el endpoint y creamos la nueva función que llamará a la ruta creada en el backend:

```
const changeStatus = async (restaurant) => {
  try {
    await alterStatus(restaurant.id)
    await fetchRestaurants()
    setRestaurantToBeChangedItsStatus(null)
    showMessage({
      message: `Restaurant ${restaurant.name} successfully changed
its status`,
      type: 'success',
      style: GlobalStyles.flashStyle,
      titleStyle: GlobalStyles.flashTextStyle
    })
  } catch (error) {
    console.log(error)
    setRestaurantToBeChangedItsStatus(null)
    showMessage({
      message: `Restaurant ${restaurant.name} could not be changed
its status.`,
      type: 'error',
      style: GlobalStyles.flashStyle,
      titleStyle: GlobalStyles.flashTextStyle
    })
  }
}
```

Pero para que funcione tenemos que crear un Modal que añadimos a la carpeta de componentes, es copiar y pegar alguno que esté hecho y cambiar el nombre.

Añadimos el modal al return:

```
return (
  <>
    <FlatList
      style={styles.container}
      data={restaurants}
      renderItem={renderRestaurant}
      keyExtractor={item => item.id.toString()}
      ListHeaderComponent={renderHeader}
      ListEmptyComponent={renderEmptyRestaurantsList}
    />
    <DeleteModal
      isVisible={restaurantToBeDeleted !== null}
      onCancel={() => setRestaurantToBeDeleted(null)}
      onConfirm={() => removeRestaurant(restaurantToBeDeleted)}>
      <TextRegular>The products of this restaurant will be deleted as
well</TextRegular>
      <TextRegular>If the restaurant has orders, it cannot be
deleted.</TextRegular>
    </DeleteModal>
    {/* SOLUCION */}
    <AlterStatusModal
      isVisible={restaurantToBeChangedItsSatus !== null}
      onCancel={() => setRestaurantToBeChangedItsSatus(null)}
      onConfirm={() => changeStatus(restaurantToBeChangedItsSatus)}>
      <TextRegular>The status of the restauarant its about to
change</TextRegular>
    </AlterStatusModal>
    {/* SOLUCION */}
  </>
)
```