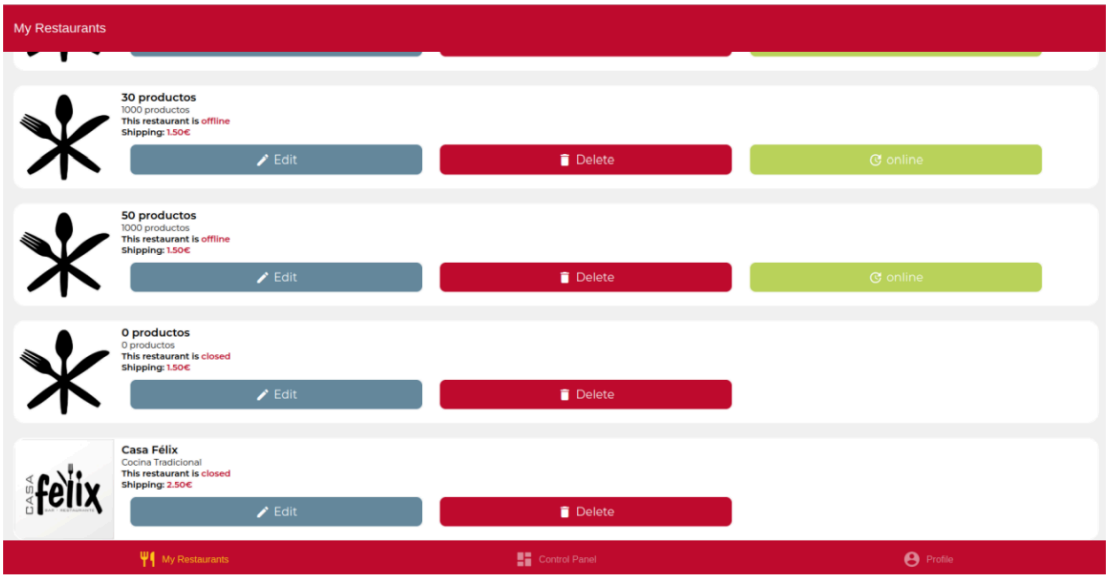
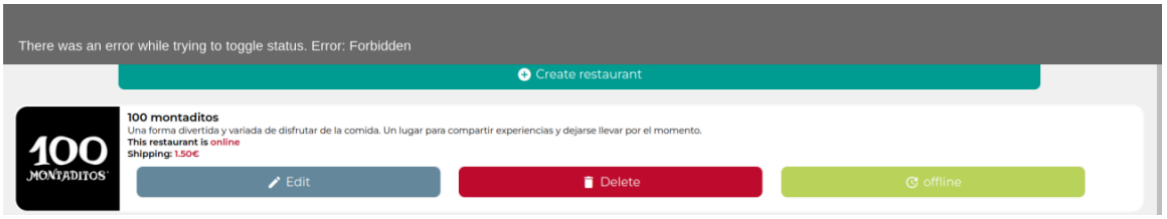
EXAMEN APERTURA DE RESTAURANTES - OCTUBRE 2023

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:





BACKEND

No hay que añadir nueva propiedad:

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

Ya tenemos una propiedad llamada status en el modelo de Restaurants:

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

Tenemos que implementar la lógica para poner online u offline:

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. ç

Creamos la lógica:

```
const alterStatus = async function (req, res) {
  try {
    const restaurantToBeChanged = await
Restaurant.findByPk(req.params.restaurantId)
    if (restaurantToBeChanged.status === 'offline') {
      restaurantToBeChanged.status = 'online'
    } else {
      restaurantToBeChanged.status = 'offline'
    }
    const restaurant = await restaurantToBeChanged.save()
    res.json(restaurant)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

Sin transacción podria ser:

```
const alterStatus = async function (req, res) {
  const t = await sequelizeSession.transaction()
  try {
    const restaurant = await
Restaurant.findByPk(req.params.restaurantId)
    if (restaurant.status === 'online') {
      restaurant.status = 'offline'
      await restaurant.save({ transaction: t })
    } else if (restaurant.status === 'offline') {
      restaurant.status = 'online'
      await restaurant.save({ transaction: t })
    }
    await t.commit()
    const updatedRestaurant = await
Restaurant.findByPk(req.params.restaurantId)
    res.json(updatedRestaurant)
  } catch (err) {
    await t.rollback()
    res.status(500).send(err)
  }
}
```

Y después en el middleWare añadimos las restricciones:
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.

Tenemos que añadir una restricción de que no esté ni cerrado ni cerrado temporalmente:

```
const restaurantHasNoClosedOrClosedTemporaryStatus = async (req, res,
next) => {
  try {
    const restaurant = await Order.findByPk(req.params.restaurantId)
    if (restaurant.status === 'closed') {
      return res.status(409).send('The restaurant is closed.')
    }
    if (restaurant.status === 'temporarily closed') {
      return res.status(409).send('The restaurant is temporarily
closed.')
    }
  } catch (err) {
    return res.status(500).send(err.message)
  }
}
```

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

Tenemos que añadir una propiedad que verifica que los order no sean nulos en deliveredAt:

```
const restaurantHasNoPendingOrders = async (req, res, next) => {
  try {
    const restaurantOrders = await Order.findAll({
      where: { restaurantId: req.params.restaurantId }
    })
    for (const order of restaurantOrders) {
      if (order.deliveredAt === null) {
        return res.status(409).send('Some orders belong to this
restaurant.')
      }
    }
    return next()
  } catch (err) {
    return res.status(500).send(err.message)
  }
}
```

Añadimos el orden:

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

```
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', 'DESC']]
      })
    res.json(restaurants)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

Y por último la ruta:

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

FRONTEND

Tenemos que añadir primero el endpoint que recibe la función que hemos creado:

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

Luego añadimos el botón:

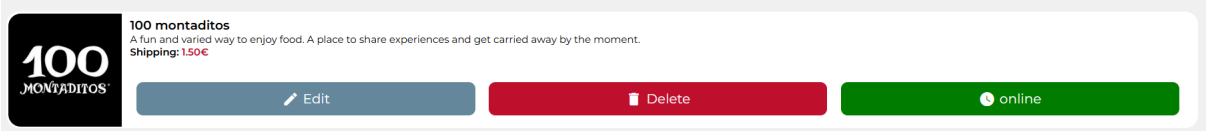
Añadimos el useState:

```
const [restaurantToBeChangedItsSatus, setRestaurantToBeChangedItsSatus]
= useState(null)
```

Y el botón:

```
 {(item.status === 'online' || item.status === 'offline') &&
        <Pressable
            onPress={() => { setRestaurantToBeChangedItsSatus(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>}
```

Ya tenemos el botón:



Ahora tenemos que crear la función que llama la endpoint para cambiar el estado:

```
const changeStatus = async (restaurant) => {
    try {
      await alterStatus(restaurant.id)
      await fetchRestaurants()
      setRestaurantToBeChangedItsSatus(null)
      showMessage({
        message: `Restaurant ${restaurant.name} succesfully cghanged
its status`,
        type: 'success',
        style: GlobalStyles.flashStyle,
        titleStyle: GlobalStyles.flashTextStyle
      })
    } catch (error) {
      console.log(error)
      setRestaurantToBeChangedItsSatus(null)
```

```
      showMessage({
        message: `Restaurant ${restaurant.name} could not be changed
its status.`,
        type: 'error',
        style: GlobalStyles.flashStyle,
        titleStyle: GlobalStyles.flashTextStyle
      })
    }
  }
```

Y tendremos que crear un Modal que permita cambiar y añadirlo al return:

```
   <AlterStatusModal
      isVisible={restaurantToBeChangedItsSatus !== null}
      onCancel={() => setRestaurantToBeChangedItsSatus(null)}
      onConfirm={() => changeStatus(restaurantToBeChangedItsSatus)}>
        <TextRegular>The status of the restauarant its about to
change</TextRegular>
    </AlterStatusModal>
```