EXAMEN DESCUENTOS- JUNIO 2024

Ante eventos televisivos masivos como la Fórmula 1 o los partidos de Liga de Campeones, durante los cuales se produce un incremento de pedidos en los restaurantes, hemos decidido incluir una nueva funcionalidad para los propietarios de los establecimientos. Mediante un cambio en la interfaz de la edición de restaurantes, los propietarios podrán modificar el precio de todos los productos de un restaurante en cuestión de segundos.

- **El acceso rápido consistirá en dos botones (flecha arriba y flecha abajo) en la vista de edición de restaurantes, que permiten modificar (incrementar o decrementar) el porcentaje que desea aplicarse al precio base de todos los productos del restaurante en cuestión. Este porcentaje puede ser positivo, lo que implica que se incrementa el precio base, o negativo, que implica un descuento sobre el precio base del producto. Por defecto, el valor de este campo será 0.**

- **Al pulsar sobre cada uno de los botones, el porcentaje se incrementará o decrementará en más/menos 0.5% con respecto al valor anterior de esta propiedad llamada 'percentage'.**

- **Una vez finalizados los cambios, el propietario pulsará sobre el botón guardar del formulario de edición de restaurante. Al pulsar este botón, el backend deberá actualizar no sólo la propiedad porcentaje del restaurante, sino también el precio de todos los productos de este, incrementando o decrementando el precio base en el porcentaje indicado. En caso de que el valor del campo sea distinto a 0, la aplicación debe pedir confirmación al propietario antes de guardar los cambios; utilice para ello el componente suministrado ConfirmationModal, similar al componente DeleteModal utilizado en clase.**

- **Debe crear una nueva propiedad llamada 'basePrice' en la entidad Product. Esta propiedad siempre contendrá el precio original del producto y tan sólo debe modificarse en los métodos create y update de ProductController. Tenga en cuenta que estos cambios ya se encuentran implementados en el proyecto base.**

- **Al modificar el porcentaje, los productos pasarán a tener un precio igual al precio base más el porcentaje indicado:**

    - **precio_final = precio_base * (1 + (porcentaje_actual / 100))**

- **De ese modo un producto que valga 4 euros podrá pasar por los siguientes estados:**

    - **Cambio +3%: precio_base=4; precio_final=4*(1+(3/100))=4.12**
    - **Cambio -2%: precio_base=4; precio_final=4*(1+(-2/100))=3.92**
    - **Cambio 0%: precio_base=4; precio_final=4*(1+(0/100))=4**

- **Por último, en el listado de restaurantes, aquellos restaurantes que tengan un porcentaje modificador del precio de los productos distinto a 0, deberán mostrar una etiqueta que lo indique: ¡Incremento de precio aplicado! o ¡Descuento aplicado! en función de si el porcentaje es mayor o menor que 0.**

Tenga en cuenta que este cambio DEBE persistir el precio del producto en la base de datos, aplicando el nuevo precio al listado de productos de dicho restaurante individualmente. Esto hará que todas las consultas relacionadas con dichos productos reciban la información actualizada. Se valorará el uso de transacciones.

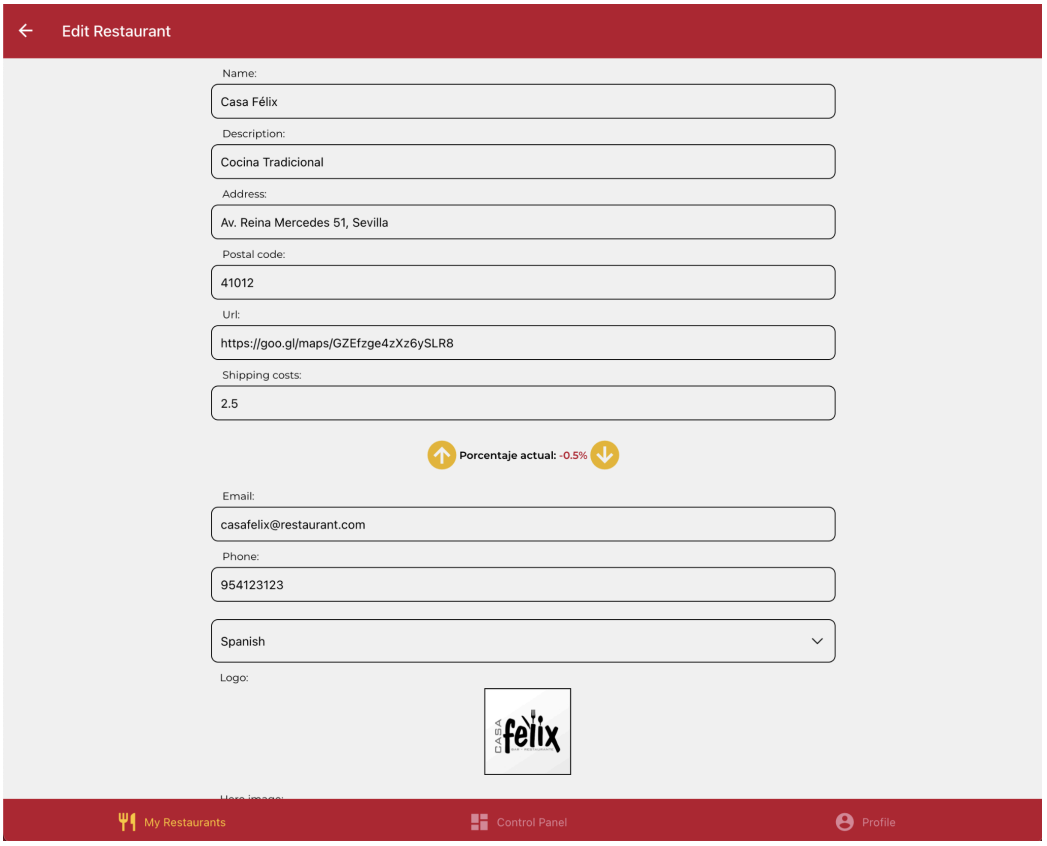
Existe una limitación en el porcentaje para evitar incrementos u ofertas muy agresivas, de manera que el nuevo campo sólo podrá contener valores decimales entre -5 y 5 (excluidos ambos valores). Deberá asegurarse de que esta restricción se cumple tanto en el formulario de edición como en el método adecuado en el backend.

Ejercicio 1
Realice todos los cambios necesarios en el proyecto de backend para implementar el nuevo requisito. Los test de backend esperan que la ruta sea: PUT /restaurants/:restaurantId y GET /restaurants/:restaurantId/products

Ejercicio 2
Realice todos los cambios necesarios en el proyecto de frontend para implementar el nuevo requisito.

Puede renderizar los iconos de las flechas hacia arriba y hacia abajo del siguiente modo:

```
<MaterialCommunityIcons
  name={'arrow-up-circle'}
  color={GlobalStyles.brandSecondaryTap}
  size={40}
/>
<MaterialCommunityIcons
  name={'arrow-down-circle'}
  color={GlobalStyles.brandSecondaryTap}
  size={40}
/>
```


Edit Restaurant
Name:
Casa Félix
Description:
Cocina Tradicional
Address:
Av. Reina Mercedes 51, Sevilla
Postal code:
41012
Url:
https://goo.gl/maps/GZEfzge4zXz6ySLR8
Shipping costs:
2.5
Porcentaje actual: -0.5%
Email:
casafelix@restaurant.com
Phone:
954123123
Spanish
Logo:
My Restaurants
Control Panel
Profile

Edit Restaurant

Postal code:

41012

Url:

https://goo.gl/maps/GZEfzge4zXz6ySLR8

Shipping costs:

2.5

Porcentaje actual: -0.5%

Email:

Va a aplicar un porcentaje correctivo sobre el precio de los productos de este restaurante

Cancel

Confirm

Hero image:

Save

My Restaurants | Control Panel | Profile

My Restaurants

Create restaurant

Casa Félix
Cocina Tradicional
Shipping: 2.50€
¡Descuentos aplicados!

Edit | Delete

100 montaditos
A fun and varied way to enjoy food. A place to share experiences and get carried away by the moment.
Shipping: 1.50€
¡Incremento de precios aplicados!

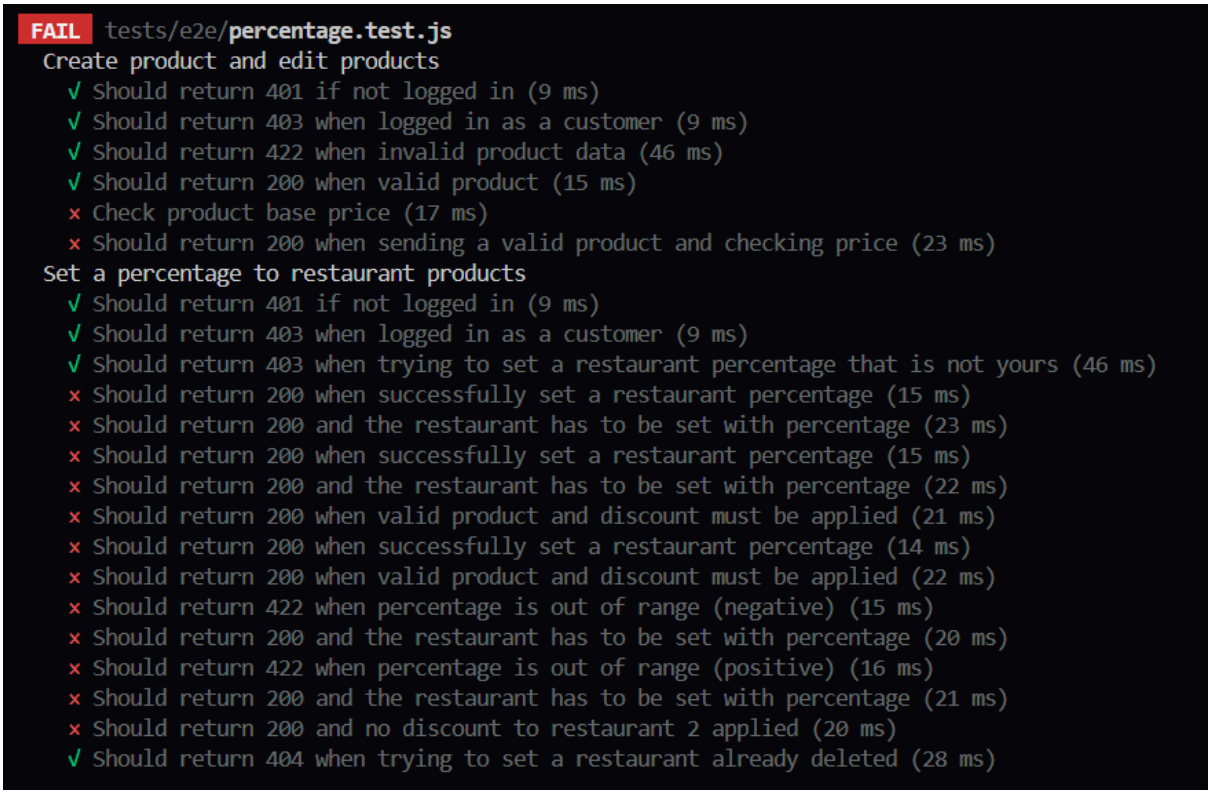
Edit | Delete

1000 products
1000 products
Shipping: 1.50€
¡Descuentos aplicados!

Edit | Delete

BACKEND

Nada más empezar ejecutar el test:



PASO 1 - ¿QUÉ NOS ESTÁN PIDIENDO?

ENUNCIADO:
…el backend deberá actualizar no sólo la propiedad porcentaje del restaurante, … Por defecto, el valor de este campo será 0.
… Debe crear una nueva propiedad llamada 'basePrice' en la entidad Product.

- Nueva propiedad: percentage
- De tipo double
- En el modelo de Restaurant
- Valor por defecto 0.0

Ya te lo dan:
- Nueva propiedad: basePrice
- De tipo double
- En el modelo de Product

PASO 2 - AÑADIR LA NUEVA PROPIEDAD

Vamos a poner en el modelo de Products la nueva propiedad (ya te la dan):

```
basePrice: DataTypes.DOUBLE
```

Y en el modelo de Restaurant la segunda nueva propiedad:

```
percentage: {
      type: DataTypes.DOUBLE,
      defaultValue: 0.0
    },
```

Y después en los migrations:
Create Product (te lo dan):

```
basePrice: {
        allowNull: false,
        type: Sequelize.DOUBLE
      },
```

Create Restaurant:

```
percentage: {
        type: Sequelize.DOUBLE,
        defaultValue: 0.0
      },
```

PASO 3 - LEER BIEN LOS TEST PARA BUSCAR SOLUCIÓN

De Restaurant:
VALIDATIONS:
Should return 422 when percentage is out of range (negative) (16 ms)
Should return 422 when percentage is out of range (positive) (16 ms)

ENUNCIADO:
Existe una limitación en el porcentaje para evitar incrementos u ofertas muy agresivas, de manera que el nuevo campo sólo podrá contener valores decimales entre -5 y 5 (excluidos ambos valores). Deberá asegurarse de que esta restricción se cumple tanto en el formulario de edición como en el método adecuado en el backend.

Si vemos el test:

```
it('Should return 422 when percentage is out of range (negative)',
async () => {
    const newRestaurantCopy = newRestaurant
    newRestaurantCopy.percentage = -6
    const response = await
request(app).put(`/restaurants/${newRestaurantCopy.id}`).set('Authoriza
tion', `Bearer ${owner.token}`).send(newRestaurantCopy)
    expect(response.status).toBe(422)
  })
```

Es cuando vamos a actualizar (put), luego solo lo hacemos en el update de Restaurant Validation.

La restricción es que tiene que tener un valor entre -5 y 5, además de ser double:

```
check('percentage').exists().isFloat({ min: -5, max: 5 }).toFloat()
```

Sólo en la función de update!!

Si volvemos a ejecutar:

```
√ Should return 422 when percentage is out of range (negative) (14 ms)
√ Should return 422 when percentage is out of range (positive) (12 ms)
```

Todos los errores del 20X son del controlador, tenemos que actualizar según:
Al modificar el porcentaje, los productos pasarán a tener un precio igual al precio base más el porcentaje indicado:

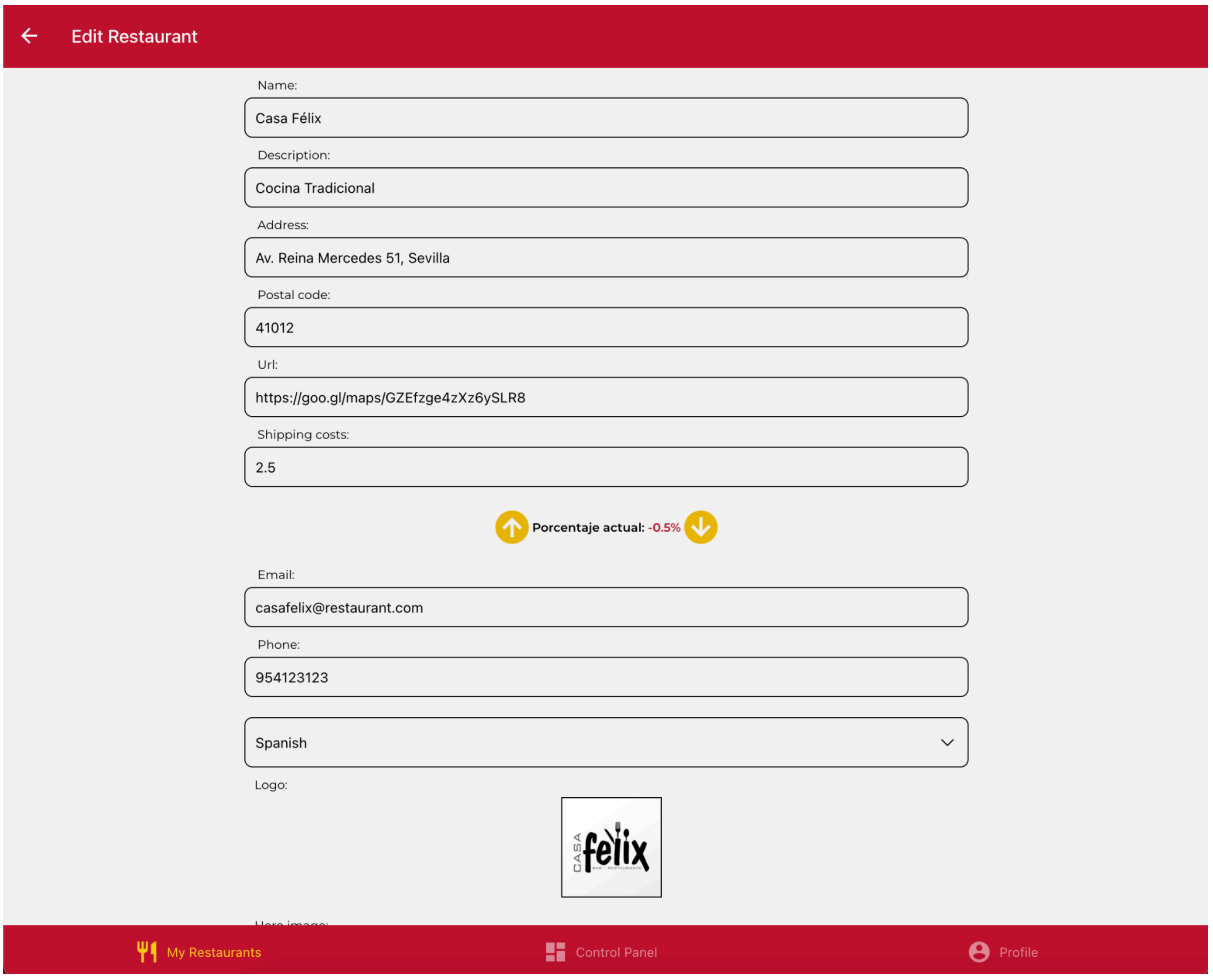
precio_final = precio_base * (1 + (porcentaje_actual / 100))

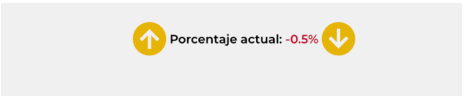
Luego tendremos que editar la función update, para que actualice el precio:

```
import Sequelize from 'sequelize'

const update = async function (req, res) {
  try {
    // Solution: not explicitly requested, but the use of a transaction
is valued
    const transaction = await sequelizeSession.transaction()
    await Restaurant.update(req.body, { where: { id:
req.params.restaurantId } }, transaction)

    const productsToBeUpdated = await Product.findAll({
      where: {
        restaurantId: req.params.restaurantId
      }
    })

    for (const product of productsToBeUpdated) {
      const newPrice = product.basePrice + product.basePrice *
(req.body.percentage / 100)
      await product.update({ price: newPrice }, transaction)
    }

    await transaction.commit()

    const updatedRestaurant = await
Restaurant.findByPk(req.params.restaurantId)

    res.json(updatedRestaurant)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

Y ya estaría.

FRONTEND



Tenemos que añadir: 

En EditRestaurantScreen:
PRIMERO:

- Añadir la propiedad y su valor por defecto:

```
  const [initialRestaurantValues, setInitialRestaurantValues] =
useState({ name: null, description: null, address: null, postalCode:
null, url: null, shippingCosts: null, percentage: 0, email: null,
phone: null, restaurantCategoryId: null, logo: null, heroImage: null })
```

SEGUNDO:

- Añadir la propiedad al validationEschema:

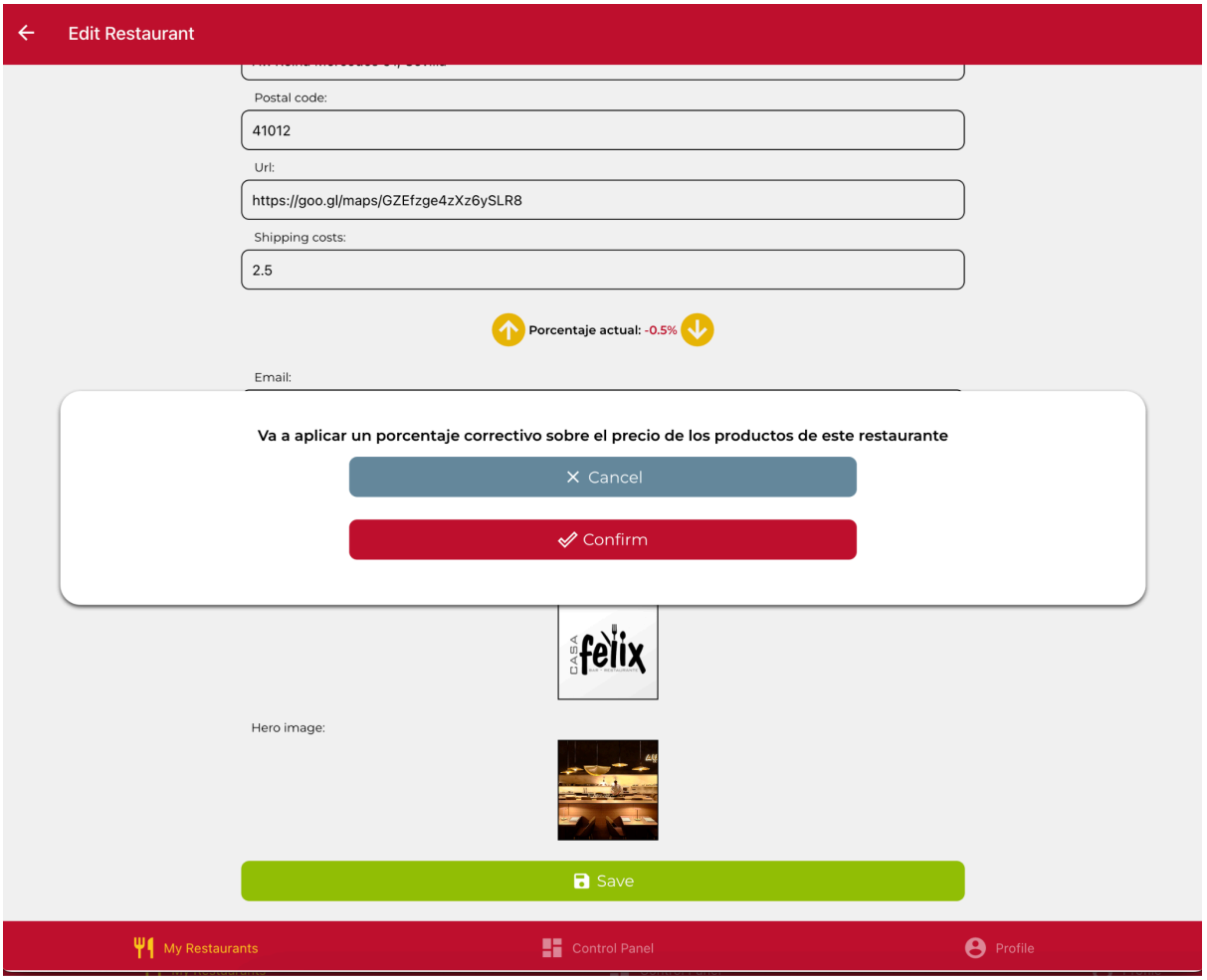
```
percentage: yup
      .number()
```

```
      .max(5)
      .min(-5),
```

Añadir ambos botones:

```
// PRIMER BOTÓN
<View style={{ flexDirection: 'row', justifyContent: 'center',
      alignItems: 'center', marginTop: 20, marginBottom: 10 }} >
  <Pressable onPress={() => {
    const newPercentage = values.percentage + 0.5
    if (newPercentage < 5) {setFieldValue('percentage',newPercentage) }
    }}
  >
     <MaterialCommunityIcons
        name={'arrow-up-circle'}
        color={GlobalStyles.brandSecondaryTap}
        size={40}
     />
   </Pressable>
//TEXTO DEL MEDIO
   <TextSemiBold>Porcentaje actual:
      <TextSemiBold textStyle={{ color: GlobalStyles.brandPrimary }}>
      {values.percentage.toFixed(1)}%</TextSemiBold>
   </TextSemiBold>
//SEGUNDO BOTÓN
   <Pressable onPress={() => {
     const newPercentage = values.percentage - 0.5
     if(newPercentage> -5) {setFieldValue('percentage',newPercentage) }
      }}
   >
       <MaterialCommunityIcons
          name={'arrow-down-circle'}
          color={GlobalStyles.brandSecondaryTap}
          size={40}
        />
   </Pressable>
</View>
```

Después tenemos que añadir el Modal de Confirmación:



Primero creamos el ConfirmationModal copiando el de delete que ya no los dan hecho, quedaría asi:

```
// This file has been created for solution

import React from 'react'
import { Modal, Pressable, StyleSheet, View } from 'react-native'
import { MaterialCommunityIcons } from '@expo/vector-icons'
import TextSemiBold from './TextSemibold'
import * as GlobalStyles from '../styles/GlobalStyles'
import TextRegular from './TextRegular'
export default function ConfirmationModal (props) {
  return (
    <Modal
      presentationStyle='overFullScreen'
      animationType='slide'
      transparent={true}
```

```
      visible={props.isVisible}
      onRequestClose={props.onCancel}>
      <View style={styles.centeredView}>
        <View style={styles.modalView}>
          <TextSemiBold textStyle={{ fontSize: 15 }}>A corrective
percentage will be applied to the price of this restaurants
products</TextSemiBold>
          {props.children}
          <Pressable
            onPress={props.onCancel}
            style={({ pressed }) => [
              {
                backgroundColor: pressed
                  ? GlobalStyles.brandBlueTap
                  : GlobalStyles.brandBlue
              },
              styles.actionButton
            ]}>
            <View style={[{ flex: 1, flexDirection: 'row',
justifyContent: 'center' }]}>
              <MaterialCommunityIcons name='close' color={'white'}
size={20} />
              <TextRegular textStyle={styles.text}>
                Cancel
              </TextRegular>
            </View>
          </Pressable>
          <Pressable
            onPress={props.onConfirm}
            style={({ pressed }) => [
              {
                backgroundColor: pressed
                  ? GlobalStyles.brandPrimaryTap
                  : GlobalStyles.brandPrimary
              },
              styles.actionButton
            ]}>
            <View style={[{ flex: 1, flexDirection: 'row',
justifyContent: 'center' }]}>
              <MaterialCommunityIcons name='check-outline'
color={'white'} size={20} />
              <TextRegular textStyle={styles.text}>
                Confirm
```

```
              </TextRegular>
            </View>
          </Pressable>
        </View>
      </View>
    </Modal>
  )
}

const styles = StyleSheet.create({
  centeredView: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    marginTop: 22
  },
  modalView: {
    margin: 20,
    backgroundColor: 'white',
    borderRadius: 20,
    padding: 35,
    alignItems: 'center',
    shadowOffset: {
      width: 0,
      height: 2
    },
    shadowOpacity: 0.75,
    shadowRadius: 4,
    elevation: 5,
    width: '90%'
  },
  actionButton: {
    borderRadius: 8,
    height: 40,
    marginTop: 12,
    margin: '1%',
    padding: 10,
    alignSelf: 'center',
    flexDirection: 'column',
    width: '50%'
  },
  text: {
    fontSize: 16,
```

```
    color: 'white',
    alignSelf: 'center',
    marginLeft: 5
  }
})
```

Y luego en EditRestaurantScreen añadimos el modal, siempre que haya un modal tenemos que crear un useState para que cuando vayamos a actualizar el restaurante cambie:

La función de update y el nuevo useState quedaría:

```
const [percentageShowDialog, setPercentageShowDialog] = useState(false)
```

```
const updateRestaurant = async (values) => {
    setBackendErrors([])
    // Solution
    if (values.percentage !== 0 && !percentageShowDialog) {
      setPercentageShowDialog(true)
    } else {
      // Solution
      setPercentageShowDialog(false)
      try {
        const updatedRestaurant = await update(restaurant.id, values)
        showMessage({
          message: `Restaurant ${updatedRestaurant.name} succesfully
            updated`,
          type: 'success',
          style: GlobalStyles.flashStyle,
          titleStyle: GlobalStyles.flashTextStyle
        })
        navigation.navigate('RestaurantsScreen', { dirty: true })
      } catch (error) {
        console.log(error)
        setBackendErrors(error.errors)
      }
    }
  }
```

Y por último añadir el Modal:

```
<ConfirmationModal
            isVisible={percentageShowDialog}
            onCancel={() => setPercentageShowDialog(false)}
```

```
            onConfirm={() => updateRestaurant(values)}>
          </ConfirmationModal>
```

Quedaría todo junto en el return asi:

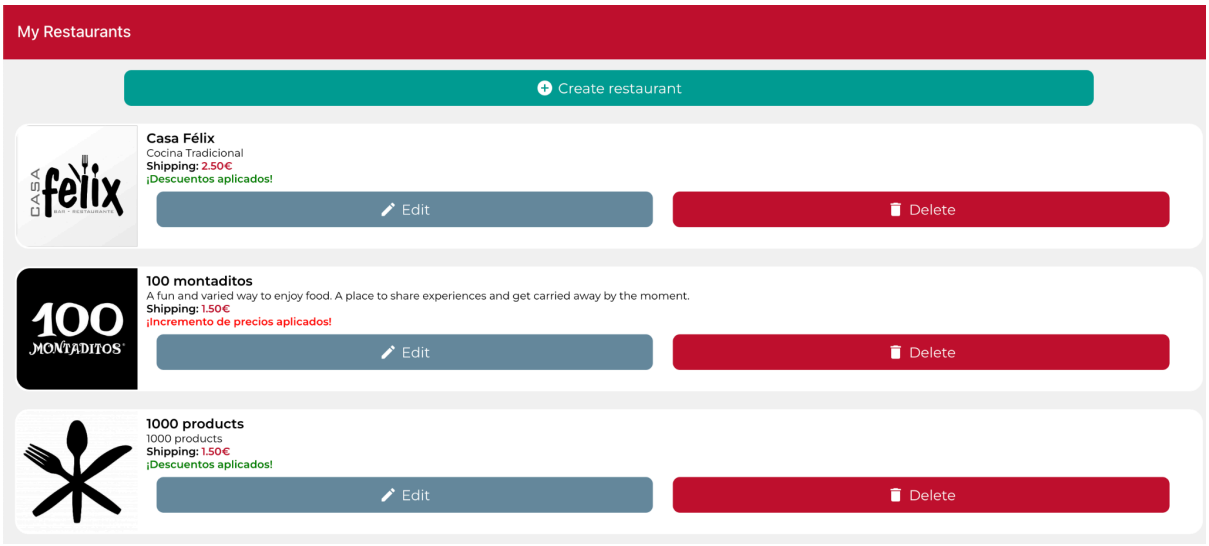
```
return (
    <Formik
      enableReinitialize
      validationSchema={validationSchema}
      initialValues={initialRestaurantValues}
      onSubmit={updateRestaurant}>
      {({ handleSubmit, setFieldValue, values }) => (
        <ScrollView>
          <View style={{ alignItems: 'center' }}>
            <View style={{ width: '60%' }}>
              <InputItem
                name='name'
                label='Name:'
              />
              <InputItem
                name='description'
                label='Description:'
              />
              <InputItem
                name='address'
                label='Address:'
              />
              <InputItem
                name='postalCode'
                label='Postal code:'
              />
              <InputItem
                name='url'
                label='Url:'
              />
              <InputItem
                name='shippingCosts'
                label='Shipping costs:'
              />
               <View style={{ flexDirection: 'row', justifyContent:
'center', alignItems: 'center', marginTop: 20, marginBottom: 10 }} >
                <Pressable onPress={() => {
                  const newPercentage = values.percentage + 0.5
                  if (newPercentage < 5) { setFieldValue('percentage',
newPercentage) }
```

```
                }}>
                  <MaterialCommunityIcons
                    name={'arrow-up-circle'}
                    color={GlobalStyles.brandSecondaryTap}
                    size={40}
                  />
                </Pressable>

                <TextSemiBold>Porcentaje actual: <TextSemiBold
textStyle={{ color: GlobalStyles.brandPrimary
}}>{values.percentage.toFixed(1)}%</TextSemiBold></TextSemiBold>

                <Pressable onPress={() => {
                  const newPercentage = values.percentage - 0.5
                  if (newPercentage > -5) { setFieldValue('percentage',
newPercentage) }
                }}>
                  <MaterialCommunityIcons
                    name={'arrow-down-circle'}
                    color={GlobalStyles.brandSecondaryTap}
                    size={40}
                  />
                </Pressable>
              </View>
              <InputItem
                name='email'
                label='Email:'
              />
              <InputItem
                name='phone'
                label='Phone:'
              />

              <DropDownPicker
                open={open}
                value={values.restaurantCategoryId}
                items={restaurantCategories}
                setOpen={setOpen}
                onSelectItem={ item => {
                  setFieldValue('restaurantCategoryId', item.value)
                }}
                setItems={setRestaurantCategories}
                placeholder="Select the restaurant category"
```

```
                containerStyle={{ height: 40, marginTop: 20 }}
                style={{ backgroundColor: GlobalStyles.brandBackground
}}
                dropDownStyle={{ backgroundColor: '#fafafa' }}
              />
              <ErrorMessage name={'restaurantCategoryId'} render={msg
=> <TextError>{msg}</TextError> }/>

              <Pressable onPress={() =>
                pickImage(
                  async result => {
                    await setFieldValue('logo', result)
                  }
                )
              }
                style={styles.imagePicker}
              >
                <TextRegular>Logo: </TextRegular>
                <Image style={styles.image} source={values.logo ? {
uri: values.logo.assets[0].uri } : restaurantLogo} />
              </Pressable>

              <Pressable onPress={() =>
                pickImage(
                  async result => {
                    await setFieldValue('heroImage', result)
                  }
                )
              }
                style={styles.imagePicker}
              >
                <TextRegular>Hero image: </TextRegular>
                <Image style={styles.image} source={values.heroImage ?
{ uri: values.heroImage.assets[0].uri } : restaurantBackground} />
              </Pressable>

              {backendErrors &&
                backendErrors.map((error, index) => <TextError
key={index}>{error.param}-{error.msg}</TextError>)
              }

              <Pressable
                onPress={handleSubmit}
```

```
                style={({ pressed }) => [
                  {
                    backgroundColor: pressed
                      ? GlobalStyles.brandSuccessTap
                      : GlobalStyles.brandSuccess
                  },
                  styles.button
                ]}>
                <View style={[{ flex: 1, flexDirection: 'row',
justifyContent: 'center' }]}>
                  <MaterialCommunityIcons name='content-save'
color={'white'} size={20}/>
                  <TextRegular textStyle={styles.text}>
                    Save
                  </TextRegular>
                </View>
              </Pressable>
            </View>
          </View>
          <ConfirmationModal
            isVisible={percentageShowDialog}
            onCancel={() => setPercentageShowDialog(false)}
            onConfirm={() => updateRestaurant(values)}>
          </ConfirmationModal>
        </ScrollView>
      )}
    </Formik>
  )
```

Y por último añadir los textos:



Tenemos que añadir:

Según aumentemos o decrementemos el precio:
Y es en RestaurantScreen:

```
{item.percentage !== 0 &&
   <View style={{ flexDirection: 'row', justifyContent:
      'space-between', alignItems: 'flex-end' }}
    >
   <TextSemiBold textStyle={{ color: item.percentage > 0 ? 'red' :
      'green' }}
   >
      {item.percentage > 0 ?
         '¡Incremento de precios aplicados!' :
          '¡Descuentos aplicados!'}
   </TextSemiBold>
   </View>
 }
```