

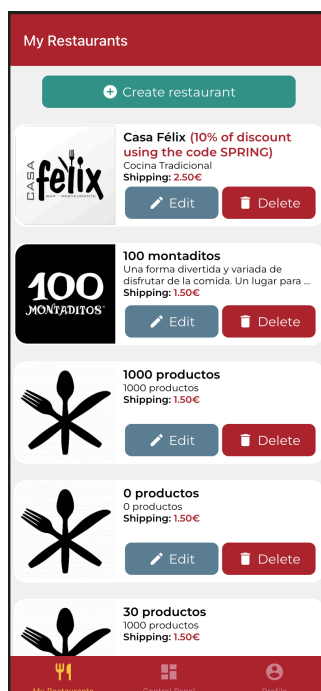
# EXAMEN DESCUENTOS Y CÓDIGOS DE DESCUENTO- JUNIO 2023

## Statement - Restaurant Discount codes

Make the modifications you consider necessary, both in the backend and in the frontend, to satisfy the new requirements described below.

The company has decided to offer owners the possibility of associating a discount code (e.g: SALES20) to their restaurants, so that the system can later display and apply the specified discount promotion code. As an illustrative example, a restaurant owner could apply the SPRING code associated with a 10% discount to Casa Felix, and another discount code, e.g. EXTREME, associated with a 30% discount to another restaurant.

The system should show the restaurants with the registered discount when both the discount code and the discount value have a value, as shown in the following screenshot:



## Remember that:

The maximum number of characters of a discount code is 10.

The discount is in the range [1, 99].

The discount code cannot be repeated for restaurants owned by the same owner.

# BACKEND

## Añadir la nueva propiedad

Tenemos dos nuevas propiedades, que serán:

- discountCode: String, puede ser null, con maximo de 10 caracateres
- discount: Float, puede ser null, entre 0 y 99

Las añadimos al modelo e Restaurant:

```
discountCode: {
  allowNull: true,
  type: DataTypes.STRING
},
discount: {
  allowNull: true,
  type: DataTypes.DOUBLE
}
```

Y después al create Restaurant:

```
discountCode: {
  allowNull: true,
  type: Sequelize.STRING,
  len: [1, 10]
},
discount: {
  allowNull: true,
  type: Sequelize.DOUBLE,
  min: 1,
  max: 99
}
```

## Añadir las validaciones de las nuevas propiedades

The maximum number of characters of a discount code is 10.

```
check('discountCode').optional({ nullable: true, checkFalsy: true })
.isString().isLength({ min: 1, max: 10 }).trim()
```

The discount is in the range [1, 99].

```
check('discount').optional({ nullable: true, checkFalsy: true
}).isFloat({ min: 0, max: 99}).toFloat()
```

**The discount code cannot be repeated for restaurants owned by the same owner.**

Tenemos que ver que los restaurantes, al contar cada código, solo aparezcan una sola vez.

Pero hay que diferenciar entre crear o actualizar los restaurantes:

PARA CREAR:

```
const checkDiscountCodeNotRepeatedCreate = async (discountCode,
ownerId) => {
  // Create a where clause object to filter the Restaurant model. The
discount code and the ownerId must match.
  const whereClause = {
    userId: ownerId, // The userId field of the Restaurant model must
match the ownerId parameter.
    discountCode: discountCode // The discountCode field of the
Restaurant model must match the discountCode parameter.
  }

  // Use the Restaurant model's count method to count the number of
restaurants that match the where clause.
  // The count method returns a Promise that resolves to the number of
matching restaurants.
  const numberRestaurantsWithSameDiscountCode = await
Restaurant.count({
    where: whereClause // Use the where clause to filter the
Restaurants.
  })

  // If the number of matching restaurants is greater than or equal to
1, throw an error indicating that the discount code is already in use.
  if (numberRestaurantsWithSameDiscountCode >= 1) {
    throw new Error('Restaurant discount codes cannot repeat among
restaurants of the same owner.');
```

## PARA ACTUALIZAR:

```
const checkDiscountCodeNotRepeatedUpdate = async (discountCode,
ownerId, restaurantId) => {
  // Create a where clause object to filter the Restaurant model. The
discount code and the ownerId must match.
  const whereClause = {
    userId: ownerId, // The userId field of the Restaurant model must
match the ownerId parameter.
    discountCode: discountCode, // The discountCode field of the
Restaurant model must match the discountCode parameter.
    // If a restaurantId is provided, it means we are updating a
restaurant, so we need to exclude the current restaurant.
    // We add a condition to the where clause to exclude the current
restaurant.
    id: { [Sequelize.Op.ne]: restaurantId }
  }
  // Use the Restaurant model's count method to count the number of
restaurants that match the where clause.
  // The count method returns a Promise that resolves to the number of
matching restaurants.
  const numberRestaurantsWithSameDiscountCode = await
Restaurant.count({
    where: whereClause // Use the where clause to filter the
Restaurants.
  })

  // If the number of matching restaurants is greater than or equal to
1, throw an error indicating that the discount code is already in use.
  if (numberRestaurantsWithSameDiscountCode >= 1) {
    throw new Error('Restaurant discount codes cannot repeat among
restaurants of the same owner.');
```

Y al final llamamos en cada método a la nueva función auxiliar quedando en total:

Create:

```
check('discountCode').optional({ nullable: true, checkFalsy: true
}).isString().isLength({ min: 1, max: 10 }).trim(),
  check('discount').optional({ nullable: true, checkFalsy: true
}).isFloat({ min: 0, max: 99 }).toFloat(),
  check('discountCode').custom((value, { req }) => {
    return checkDiscountCodeNotRepeatedCreate(value, req.user.id)
  }).withMessage('Restaurant discount codes cannot repeat among
restaurants of the same owner.')
```

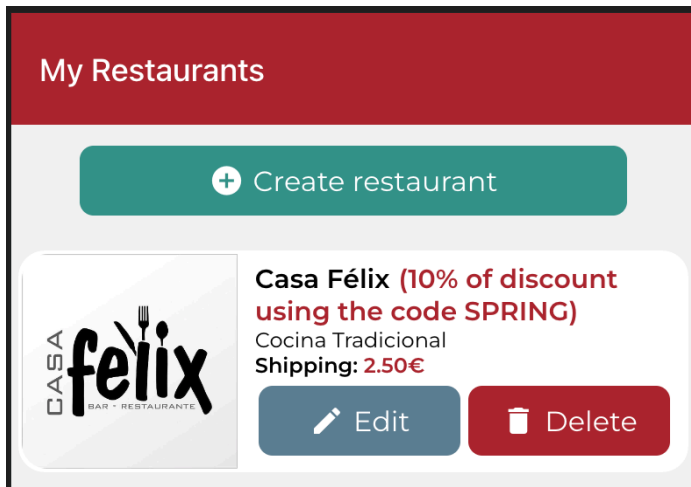
Update:

```
check('discountCode').optional({ nullable: true, checkFalsy: true
}).isString().isLength({ min: 1, max: 10 }).trim(),
  check('discount').optional({ nullable: true, checkFalsy: true
}).isFloat({ min: 0, max: 99 }).toFloat(),
  check('discountCode').custom((value, { req }) => {
    return checkDiscountCodeNotRepeatedUpdate(value, req.user.id,
req.params.restaurantId)
  }).withMessage('Restaurant discount codes cannot repeat among
restaurants of the same owner.')
```

Y por último ordenamos por descuento:

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

# FRONTEND



En RestaurantScreen tenemos que añadir un nuevo render:

```
const renderCardTitle = (item) => {
  return (<>
    <TextSemiBold style={{ fontSize: 15
  }}>{item.name}</TextSemiBold>
    { (item.discountCode && item.discount) && <TextSemiBold style={{
marginLeft: 5, fontSize: 15, color: GlobalStyles.brandPrimary
  }}>({item.discount}% of discount using the code
{item.discountCode})</TextSemiBold> }
    </>)
}
```

Y lo añadimos a:

```
const renderRestaurant = ({ item }) => {
  return (
    <ImageCard
      imageUrl={item.logo ? { uri: process.env.API_BASE_URL + '/' +
item.logo } : restaurantLogo}
      title={renderCardTitle(item)} //SOL
      onPress={() => {
        navigation.navigate('RestaurantDetailScreen', { id: item.id
      })
    }}
    />
  )
}
```

En el edit restaurant screen, añadimos:

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

Y en el validation Esquema:

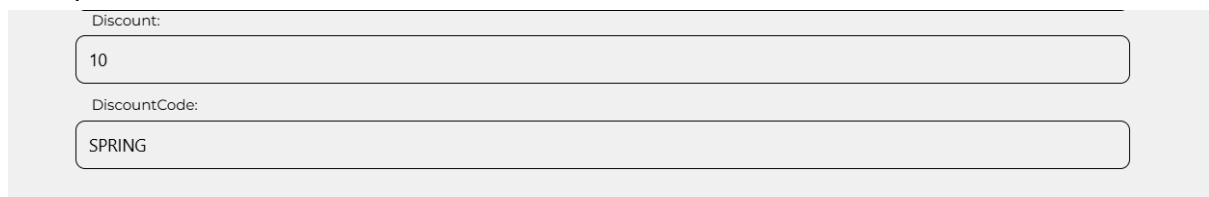
```
discount: yup
  .number()
  .max(99)
  .min(1)
  .nullable(),
discountCode: yup
  .string()
  .max(10, 'Discount code too long')
  .min(1, 'Discount code too short')
  .nullable()
```

Y añadimos el hueco donde irán:

```
<InputItem
  name='discount'
  label='Discount:'
/>
<InputItem
  name='discountCode'
  label='DiscountCode:'
/>
```

Para el create Restaurant es igual:

Nos queda:



Discount:

10

DiscountCode:

SPRING

Y si intentamos meter un codigo que no es:

Discount:

DiscountCode:


Discount code too long

Si es el mismo:


Discount:

DiscountCode:

Logo:



Hero image:



discountCode-Restaurant discount codes cannot repeat among restaurants of the same owner.