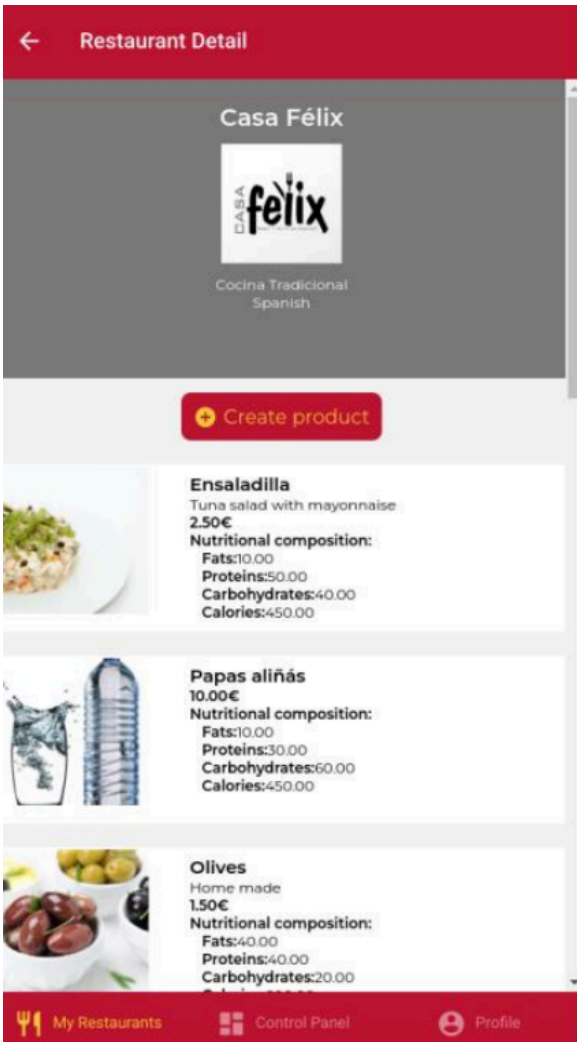
EXAMEN PRODUCTOS SALUDABLES - JUNIO 2022

Realice las modificaciones que considere necesarias, tanto en backend como en frontend, para satisfacer los nuevos requisitos que a continuación se describen.
Se desea una nueva funcionalidad que consista en informar al cliente de los gramos de macronutrientes que contienen cada uno de los productos.

Tras la reunión con el cliente, se han establecido los siguientes requisitos:

1. **Debido a la normativa reguladora de alimentación, se pide que se informe al cliente de la cantidad de carbohidratos, proteínas y grasas por cada 100 gramos de cada uno de los productos que están a la venta en DeliverUS.**
2. **Dada la existencia de platos hipercalóricos que no están recomendados en una dieta saludable, se pide que un plato no pueda contener más de 1000 calorías por 100g de producto. Para ello, se usará la siguiente formula aproximada de cálculo energético:**

   Calorías producto = (grasas * 9) + (proteínas * 4) + (carbohidratos * 4)



BACKEND

Añadir las nuevas propiedades:

Debido a la normativa reguladora de alimentación, se pide que se informe al cliente de la cantidad de **carbohidratos, proteínas y grasas por cada 100 gramos** de cada uno de los productos que están a la venta en DeliverUS.

Tendremos que añadir 3 nuevas propiedades al modelo de Product:

```
    fats: DataTypes.DOUBLE,
    proteins: DataTypes.DOUBLE,
    carbs: DataTypes.DOUBLE,
    calories: DataTypes.DOUBLE
```

Y en el create Product:

```
      fats: Sequelize.DOUBLE,
      proteins: Sequelize.DOUBLE,
      carbs: Sequelize.DOUBLE,
      calories: Sequelize.DOUBLE
```

Vamos a ver que nos piden validar:
Dada la existencia de platos hipercalóricos que no están recomendados en una dieta saludable, se pide que un plato no pueda contener más de 1000 calorías por 100g de producto. Para ello, se usará la siguiente fórmula aproximada de cálculo energético:

Calorías producto = (grasas * 9) + (proteínas * 4) + (carbohidratos * 4)

Tenemos que:

- Validar que las calorías totales sean menor que 1000:

```
const noMoreThan1000Calories = async (grasas, proteinas, carbohidratos)
=> {
  if (parseFloat(grasas) * 9 + parseFloat(proteinas) * 4 +
parseFloat(carbohidratos) * 4 > 1000.0) {
    return Promise.reject(new Error('The sum of 1000 calories cannot
exceed 1000.'))
  } else {
    return Promise.resolve()
  }
}
```

- Validar que la suma de los gramos sea 100:

```
const check100grams = async (grasas, proteinas, carbohidratos) => {
  if (parseFloat(grasas) + parseFloat(proteinas) +
parseFloat(carbohidratos) !== 100.0) {
    return Promise.reject(new Error('The sum of 100 grams cannot exceed
100.'))
  } else {
    return Promise.resolve()
  }
}
```

Y añadir al create y update:

```
   check('fats').custom((value, { req }) => {
    return check100grams(value, req.body.proteins, req.body.carbs)
  }).withMessage('The sum of 100 grams cannot exceed 100.'),
  check('fats').custom((value, { req }) => {
    return noMoreThan1000Calories(value, req.body.proteins,
req.body.carbs)
  }).withMessage('The sum of 100 grams cannot exceed 100.')
```

O así:

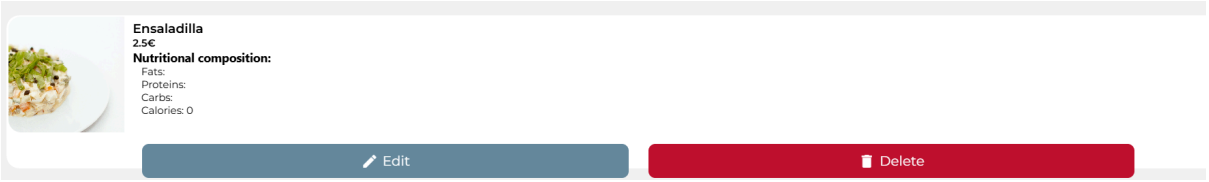
```
  check('fats').custom((values, { req }) => {
    const { fats, proteins, carbohydrates } = req.body
    return check100Grams(fats, proteins, carbohydrates)
  }).withMessage('The values of fat, protein and carbohydrates must be
in the range [0, 100] and the sum must be 100.'),
  check('fats').custom((values, { req }) => {
    const { fats, proteins, carbs } = req.body
    return checkCalories(fats, proteins, carbs)
  }).withMessage('The number of calories must not be greater than
1000.')
```

Después añadimos en create y update del controlador las calorias totales:

```
const create = async function (req, res) {
  let newProduct = Product.build(req.body)
  // SOLUCION
  newProduct.fats = req.body.fats
  newProduct.proteins = req.body.proteins
  newProduct.carbs = req.body.carbs
  newProduct.calories = req.body.fats * 9 + req.body.proteins * 4 +
req.body.carbs * 4

  try {
    newProduct = await newProduct.save()
    res.json(newProduct)
  } catch (err) {
    res.status(500).send(err)
  }
}

const update = async function (req, res) {
  try {
    // SOLUCION
    const caloriesTotal = req.body.fats * 9 + req.body.proteins * 4 +
req.body.carbs * 4
    await Product.update(
      { ...req.body, fats: req.body.fats, proteins: req.body.proteins,
carbs: req.body.carbs, calories: caloriesTotal },
      { where: { id: req.params.productId } })
    const updatedProduct = await Product.findByPk(req.params.productId)
    res.json(updatedProduct)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

FRONTEND

Tenemos que añadir en el Restaurant Detail Screen un texto con las nuevas propiedades:



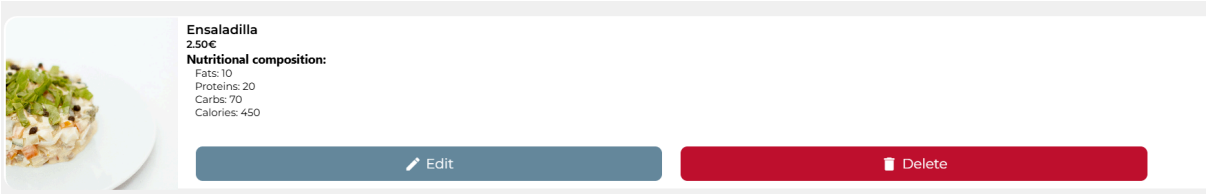

```
const renderProduct = ({ item }) => {
    const Fats = item.fats
    const Proteins = item.proteins
    const Carbs = item.carbs
    const calories = Fats * 9 + Proteins * 4 + Carbs * 4

    return (
      <ImageCard
        imageUri={item.image ? { uri: process.env.API_BASE_URL + '/' +
item.image } : defaultProductImage}
        title={item.name}
      >
        <TextRegular numberOfLines={2}>{item.description}</TextRegular>
        <TextSemiBold
textStyle={styles.price}>{item.price.toFixed(1)}€</TextSemiBold>
        {<TextSemiBold style={{ fontWeight: 'bold' }}>Nutritional
composition:</TextSemiBold>}
        <View style={{ flexDirection: 'column', alignSelf: 'left',
paddingLeft: 10 }}>
          <TextRegular>Fats: {item.fats}</TextRegular>
          <TextRegular>Proteins: {item.proteins}</TextRegular>
          <TextRegular>Carbs: {item.carbs}</TextRegular>
          <TextRegular>Calories: {calories}</TextRegular>
        </View>

        {!item.availability &&
          <TextRegular textStyle={styles.availability }>Not
available</TextRegular>
        }
```

He bajado:

```
actionButtonsContainer: {
    flexDirection: 'row',
    bottom: -20,
    position: 'relative',
    width: '90%'
```

Y la imageCard.

Saldra:



Ahora el Edit Product Screen y Create Product Screen tendremos que añadir inputs para fats, carbs y proteins.

En el const y en el validation esquema de ambos:

```
  const [initialProductValues, setInitialProductValues] = useState({
name: null, description: null, price: null, order: null,
productCategoryId: null, availability: null, image: null,
carbs: 0, fats: 0, proteins: 0
 })
```

Y en el validation esquema:

```
fats: yup
      .number()
      .min(0, 'Fats must be between 0 and 100')
      .max(100, 'Fats must be between 0 and 100')
      .required('Fats is required'),
    proteins: yup
      .number()
      .min(0, 'Proteins must be between 0 and 100')
      .max(100, 'Proteins must be between 0 and 100')
      .required('Proteins is required'),
    carbs: yup
      .number()
      .min(0, 'Carbs must be between 0 and 100')
      .max(100, 'Carbs must be between 0 and 100')
      .required('Carbs is required')
```

Y añadimos al return:

```
{/* SOLUCION */ }
              <InputItem
                name='fats'
                label='Fats:'
              />
              <InputItem
                name='proteins'
                label='Proteins:'
              />
              <InputItem
                name='carbs'
                label='Carbs'
              />
```