

ISSI2: Evaluación individual laboratorio 2022-2023

Preparación del entorno

Windows:

- Abra un terminal y ejecute el comando `npm run install:all:win`.

Linux/MacOS:

- Abra un terminal y ejecute el comando `npm run install:all:bash`.

Ejecución y depuración

- Para **ejecutar el backend**, abra un terminal y ejecute el comando `npm run start:backend`.
- Para **ejecutar el frontend**, abra un nuevo terminal y ejecute el comando `npm run start:frontend`.
- Para **depurar el backend**, asegúrese de que **NO** existe una instancia en ejecución, pulse en el botón **Run and Debug** de la barra lateral, seleccione **Debug Backend** en la lista desplegable, y pulse el botón de *Play*.
- Para **depurar el frontend**, asegúrese de que **EXISTE** una instancia en ejecución, pulse en el botón **Run and Debug** de la barra lateral, seleccione **Debug Frontend** en la lista desplegable, y pulse el botón de *Play*.
- Para **rehacer las migraciones y seeders del backend**, abra un terminal y ejecute el comando `npm run migrate:backend`.

Enunciado - Creación de categorías de restaurantes personalizadas

Realice las modificaciones que considere necesarias, tanto en backend como en frontend, para satisfacer los nuevos requisitos que a continuación se describen.

Se desea permitir a los dueños de restaurantes crear sus propias categorías de restaurantes. Para ello, en la pantalla de creación de restaurantes se incluirá un botón para acceder a una nueva pantalla que permitirá introducir un nuevo nombre de categoría de restaurante (ver capturas). Puede usar este icono para el botón:

```
<MaterialCommunityIcons name='folder-plus-outline' color={'white'} size={20} />
```

Al volver a la pantalla de creación de restaurantes tras introducir la nueva categoría, dicha categoría debe estar disponible en la lista desplegable de categorías de restaurantes.

No se debe permitir la creación de una categoría que ya existiera. En dicho caso, el Backend debe responder con un error que será visualizado en la pantalla de creación de categorías de restaurantes al pulsar el botón de submit. Además, el tamaño máximo para los nombres de las categorías de restaurante será de 50 caracteres. Esta restricción debe comprobarse tanto a nivel de formulario en el Frontend como a nivel de Backend.

← Create Restaurant

Name:

Rollito's

Description:

Comida mejicana

Address:

Calle Pajaritos, 8

Postal code:

41008

Url:

http://www.rollitosrestaurant.es

Shipping costs:

5

Email:

rollitos@rollitosrestaurant.es

Phone:


675562456

Select the restaurant category

▼

New category

Logo:



My Restaurants

Control Panel

Profile

← Create Restaurant Category

Name:

Mexican food

Save

My Restaurants

Control Panel

Profile

← Create Restaurant

Name:

Rollito's

Description:

Comida mejicana

Address:

Calle Pajaritos, 8

Postal code:

41008

Url:

http://www.rollitosrestaurant.es

Shipping costs:

5

Email:

rollitos@rollitosrestaurant.es

Phone:

675562456

Select the restaurant category


^

Spanish

Fast food

Mexican food

Logo:



My Restaurants

Control Panel

Profile

← Create Restaurant Category

Name:

Fast food

name-The category Fast food already exists.

Save

My Restaurants

Control Panel

Profile

Procedimiento para la entrega

1. Elimine las carpetas node_modules de los proyectos de backend y frontend y la carpeta .expo del proyecto frontend.
2. Cree un zip que incluya ambos proyectos. **OJO: Compruebe que el zip resultante no es el mismo que descargó y que por lo tanto incluye sus soluciones**
3. Avise a su profesor antes de realizar el envío.
4. Cuando tenga el visto bueno de su profesor, envíe el fichero resultante a través de enseñanza virtual.

BACKEND

1. Nueva funcion en RestaurantCategory

```
const create = async function (req, res) {
  try {
    let newRestaurantCategorie = RestaurantCategory.build(req.body)
    newRestaurantCategorie = await newRestaurantCategorie.save()
    res.json(newRestaurantCategorie)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

2. Creacion de RestaurantCategoryValidation

“No se debe permitir la creación de una categoría que ya existiera”

```
import { check } from 'express-validator'
import { RestaurantCategory } from '../models/models.js'

const checkNoExists = async (value, { req }) => {
  try {
    const restaurantCategories = await RestaurantCategory.findAll()
    const names = restaurantCategories.map(restC => restC.name)
    if (names.includes(value)) {
      return Promise.reject(new Error(`The category ${value} already
exists`))
    }
    return Promise.resolve()
  } catch (err) {
    return Promise.reject(new Error(err))
  }
}

const create = [
  check('name').exists().isString().isLength({ min: 1, max: 250
}).trim(),
  check('name').custom(checkNoExists).withMessage('The category already
exists')
]

export { create }
```

3. Añadimos nueva ruta

```
const loadFileRoutes = function (app) {  
  app.route('/restaurantCategories')  
    .get(  
      RestaurantCategoryController.index)  
    .post(  
      RestaurantCategoryValidation.create,  
      handleValidation,  
      RestaurantCategoryController.create)  
}
```

FRONTEND

1. Nuevo endpoint en RestaurantEndPoint

```
function postRestaurantCategories (data) {  
  return post('restaurantCategories', data)  
}
```

2. Nueva Screen → RestaurantCategoryScreen + ACTUALIZAR EL RESTAURANT STACK MUY IMPORTANTE !!! (SI NO EN LA VIDA VA A APARECER LA SCREEN)

```
import React, { useEffect, useState } from 'react'  
import { Platform, Pressable, ScrollView, StyleSheet, View } from 'react-native'  
import * as ExpoImagePicker from 'expo-image-picker'  
import { MaterialCommunityIcons } from '@expo/vector-icons'  
import * as yup from 'yup'  
import { postRestaurantCategories } from '../../api/RestaurantEndpoints'  
import InputItem from '../../components/InputItem'  
import TextRegular from '../../components/TextRegular'  
import * as GlobalStyles from '../../styles/GlobalStyles'  
  
import { showMessage } from 'react-native-flash-message'  
import { Formik } from 'formik'  
import TextError from '../../components/TextError'  
export default function CreateRestaurantCategoryScreen ({ navigation }) {  
  const initialRestaurantCategoryValues = { name: null }  
  const [backendErrors, setBackendErrors] = useState()  
  
  const validationSchema = yup.object().shape({  
    name: yup  
      .string()  
      .max(50, 'Name too long')  
      .required('Restaurant Category is required')  
  })  
  
  useEffect(() => {  
    (async () => {  
      if (Platform.OS !== 'web') {  
        const { status } = await  
ExpoImagePicker.requestMediaLibraryPermissionsAsync()  
        if (status !== 'granted') {  
          alert('Sorry, we need camera roll permissions to make this  
work!')  
        }  
      }  
    })()  
  }, [])
```

```

const createRestaurantCategory = async (values) => {
  setBackendErrors([])
  try {
    const createdRestauranCategory = await
postRestaurantCategories(values)
    showMessage({
      message: `Restaurant Category ${createdRestauranCategory.name}
successfully created`,
      type: 'success',
      style: GlobalStyles.flashStyle,
      titleStyle: GlobalStyles.flashTextStyle
    })
    navigation.navigate('RestaurantsScreen', { dirty: true })
  } catch (error) {
    console.log(error)
    setBackendErrors(error.errors)
  }
}

```

```

return (
  <Formik
    validationSchema={validationSchema}
    initialValues={initialRestaurantCategoryValues}
    onSubmit={createRestaurantCategory}>
    ({({ handleSubmit, setFieldValue, values }) => (
      <ScrollView>
        <View style={{ alignItems: 'center' }}>
          <View style={{ width: '60%' }}>
            <InputItem
              name='name'
              label='Name:'
            />
          </View>
        </View>
        <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>
    {backendErrors &&
        backendErrors.map((error, index) => <TextError
key={index}>{error.param}-{error.msg}</TextError>)
    }
    </View>
</ScrollView>
    )}
</Formik>
)
}

```

```

const styles = StyleSheet.create({
  button: {
    borderRadius: 8,
    height: 40,
    padding: 10,
    width: '60%',
    marginTop: 20,
    marginBottom: 20
  },
  text: {
    fontSize: 16,
    color: 'white',
    textAlign: 'center',
    marginLeft: 5
  },
  imagePicker: {
    height: 40,
    paddingLeft: 10,
    marginTop: 20,
    marginBottom: 80
  },
  image: {
    width: 100,
    height: 100,
    borderWidth: 1,
    alignSelf: 'center',
    marginTop: 5
  }
})

```

```
}}
```

MUY IMPORTANTE ACTUALIZAR EL STACK

```
<Stack.Screen
  name='CreateRestaurantCategoryScreen'
  component={CreateRestaurantCategoryScreen}
  options={{
    title: 'Create Restaurant Category'
  }} />
```

3. Nuevo boton para ir a la pantalla de CreateCategoryScreen

```
<Pressable
  onPress={() =>
navigation.navigate('CreateRestaurantCategoryScreen')}
  style={({ pressed }) => [
    {
      backgroundColor: pressed
        ? GlobalStyles.brandGreen
        : GlobalStyles.brandGreen
    },
    styles.button
  ]>
  <View style={[{ flex: 1, flexDirection: 'row',
justifyContent: 'center' }]]>
    <MaterialCommunityIcons name='folder-plus-outline'
color={'white'} size={20} />
    <TextRegular textStyle={styles.text}>
      New category
    </TextRegular>
  </View>
</Pressable>
```


IISSI2: Evaluación individual laboratorio Octubre 2023

Preparación del entorno

Windows:

- Abra un terminal y ejecute el comando `npm run install:all:win`.

Linux/MacOS:

- Abra un terminal y ejecute el comando `npm run install:all:bash`.

Ejecución y depuración

- Para **ejecutar el backend**, abra un terminal y ejecute el comando `npm run start:backend`.
- Para **ejecutar el frontend**, abra un nuevo terminal y ejecute el comando `npm run start:frontend`.
- Para **depurar el backend**, asegúrese de que **NO** existe una instancia en ejecución, pulse en el botón Run and Debug de la barra lateral, seleccione Debug Backend en la lista desplegable, y pulse el botón de Play.
- Para **depurar el frontend**, asegúrese de que **EXISTE** una instancia en ejecución, pulse en el botón Run and Debug de la barra lateral, seleccione Debug Frontend en la lista desplegable, y pulse el botón de Play.
- Para **rehacer las migraciones y seeders del backend**, abra un terminal y ejecute el comando `npm run migrate:backend`.

Enunciado

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.

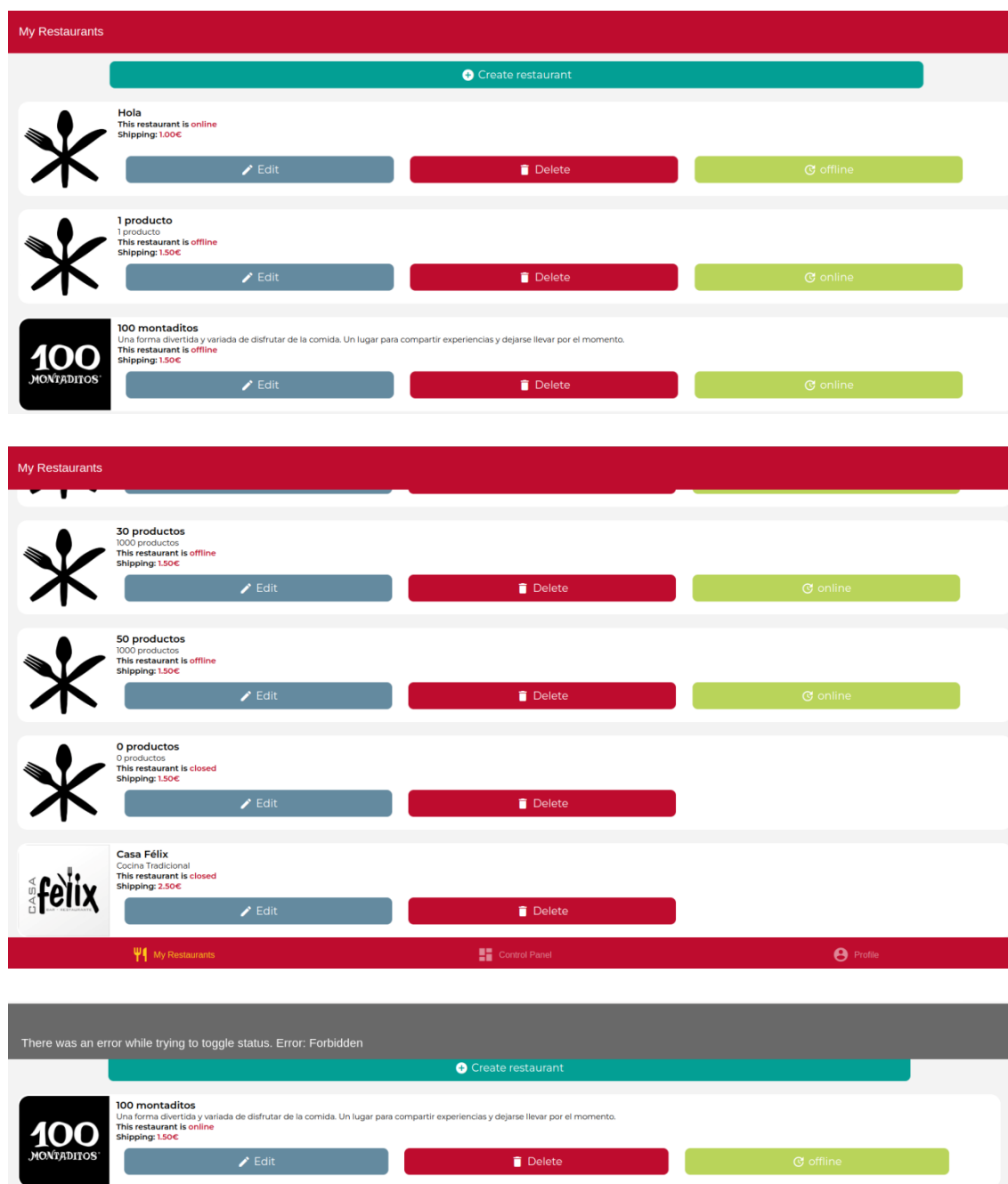
Tenga en cuenta que cualquier restaurante nuevo se almacena inicialmente como `offline` de forma predeterminada. Solo los restaurantes `offline` pueden estar en línea y viceversa. Por otro lado, si un restaurante alcanza un estado de cerrado o cerrado temporalmente, la posibilidad de cambio manual no 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:



Le pedimos que implemente los cambios necesarios en el backend y frontend para incluir la funcionalidad requerida.

Procedimiento para la entrega

- 1. Elimine las carpetas `node_modules` de los proyectos de backend y frontend y la carpeta `.expo` del proyecto frontend.
- 2. Cree un zip que incluya ambos proyectos. **OJO: Compruebe que el zip resultante no es el mismo que descargó y que por lo tanto incluye sus soluciones**
- 3. Avise a su profesor antes de realizar el envío.
- 4. Cuando tenga el visto bueno de su profesor, envíe el fichero resultante a través de enseñanza virtual.

BACKEND

1. Nueva funcion en RestaurantController

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

2. Nueva funcion en RestaurantMiddleware

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.

```
const restaurantHasNoOpen = async (req, res, next) => {
  try {
    const restaurant = await Restaurant.findById(req.params.restaurantId)
    const status = restaurant.status
    if (status === 'online' || status === 'offline') {
      return next()
    }
    return res.status(403).send('NO SE PUEDE')
  } 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.

```
const restaurantHasNoOrdersWithNull = async (req, res, next) => {
  try {
    const numberOfRestaurantOrdersWithDeliverNull = await Order.count({
      where: {
        restaurantId: req.params.restaurantId,
        deliveredAt: null
      }
    })
    if (numberOfRestaurantOrdersWithDeliverNull === 0) {
      return next()
    }
    return res.status(409).send('NO SE PUEDE ')
  } catch (err) {
    return res.status(500).send(err.message)
  }
}
```

3. Nueva ruta en RestaurantRoute

PATCH pq solo modificamos una propiedad

```
app.route('/restaurants/:restaurantId/status')
  .patch(
    isLoggedIn,
    hasRole('owner'),
    checkEntityExists(Restaurant, 'restaurantId'),
    RestaurantMiddleware.restaurantHasNoOrdersWithNull,
    RestaurantMiddleware.restaurantHasNoOpen,
    RestaurantController.changeStatus
  )
```

FRONTEND

1. Nuevo EndPoint en RestaurantEndPoint

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

2. Modificaciones en RestaurantScreen

```
export default function RestaurantsScreen ({ navigation, route }) {  
  const [restaurants, setRestaurants] = useState([])  
  const [restaurantStatus, setRestaurantStatus] = useState(null)  
  const [restaurantToBeDeleted, setRestaurantToBeDeleted] =  
  useState(null)  
  const { loggedInUser } = useContext(AuthorizationContext)
```

```
  const changeStatus2 = async (restaurant) => {  
    try {  
      await changeStatus(restaurant.id)  
      await fetchRestaurants()  
      setRestaurantStatus(restaurant.status)  
    } catch (error) {  
      setRestaurantStatus(null)  
      console.log(error)  
      showMessage({  
        message: 'There was an error while trying to toggle status.  
Error: Forbidden',  
        type: 'error',  
        style: GlobalStyles.flashStyle,  
        titleStyle: GlobalStyles.flashTextStyle  
      })  
    }  
  }  
}
```

```

const renderRestaurant = ({ item }) => {
  return (
    <ImageCard
      imageUri={item.logo ? { uri: process.env.API_BASE_URL + '/' +
item.logo } : restaurantLogo}
      title={item.name}
      onPress={() => {
        navigation.navigate('RestaurantDetailScreen', { id: item.id })
      }}
    >
      <TextRegular numberOfLines={2}>{item.description}</TextRegular>
      {item.averageServiceMinutes !== null &&
        <TextSemiBold>Avg. service time: <TextSemiBold textStyle={{
color: GlobalStyles.brandPrimary }}>{item.averageServiceMinutes}
min.</TextSemiBold></TextSemiBold>
      }
      <TextSemiBold>This restaurant is : <TextSemiBold textStyle={{
color: GlobalStyles.brandPrimary
}}>{item.status}</TextSemiBold></TextSemiBold>
      <TextSemiBold>Shipping: <TextSemiBold textStyle={{ color:
GlobalStyles.brandPrimary
}}>{item.shippingCosts.toFixed(2)}€</TextSemiBold></TextSemiBold>

      <View style={styles.actionButtonsContainer}>
        <Pressable
          onPress={() => navigation.navigate('EditRestaurantScreen', {
id: item.id })}
          style={({ pressed }) => [
            {
              backgroundColor: pressed
                ? GlobalStyles.brandBlueTap
                : GlobalStyles.brandBlue
            },
            styles.actionButton
          ]}
        >
          <View style={[{ flex: 1, flexDirection: 'row', justifyContent:
'center' }]}>
            <MaterialCommunityIcons name='pencil' color={'white'}
size={20}/>
            <TextRegular textStyle={styles.text}>
              Edit
            </TextRegular>
          </View>
        </Pressable>

        <Pressable
          onPress={() => { setRestaurantToBeDeleted(item) }}
          style={({ pressed }) => [

```

```

        {
          backgroundColor: pressed
            ? GlobalStyles.brandPrimaryTap
            : GlobalStyles.brandPrimary
        },
        styles.actionButton
      ]}]>
      <View style=[[{ flex: 1, flexDirection: 'row', justifyContent:
'center' }]]>
        <MaterialCommunityIcons name='delete' color={'white'}
size={20}/>
        <TextRegular textStyle={styles.text}>
          Delete
        </TextRegular>
      </View>
    </Pressable>

    <Pressable
      onPress={() => { changeStatus2(item) }}
      style={({ pressed }) => [
        {
          backgroundColor: pressed
            ? GlobalStyles.brandGreen
            : GlobalStyles.brandGreen
        },
        styles.actionButton
      ]}]>
      <View style=[[{ flex: 1, flexDirection: 'row', justifyContent:
'center' }]]>
        <MaterialCommunityIcons name='delete' color={'white'}
size={20}/>
        <TextRegular textStyle={styles.text}>
          {item.status === 'online' && item.status}
          {item.status === 'offline' && item.status}
        </TextRegular>
      </View>
    </Pressable>

  </View>
</ImageCard>
)
}

```

ISSI-2 IS: Examen de laboratorio Junio 2024.

Productos con periodo de visibilidad. Enunciado

Una vez se ha puesto en marcha la primera versión de DeliverUS, los inversores han solicitado la inclusión de una nueva funcionalidad que consiste en ofrecer a los propietarios la posibilidad de establecer un momento en el que los productos dejarán de ser visibles (`visibleUntil`).

Un propietario podrá establecer este momento al crear o actualizar un producto con dos escenarios posibles:

- Por defecto, este momento será nulo, por lo que se considera que siempre estará visible.
- Si un propietario establece este momento, el producto solo estará visible hasta la fecha (inclusive).

Además se deben cumplir las siguientes reglas de negocio:

- Un propietario no podrá establecer un momento de fin anterior al momento actual.
- Un propietario no podrá establecer un producto como no disponible y, al mismo tiempo, un momento de fin.

Finalmente, los productos que estén a una semana o menos de desaparecer aparecerán en la interfaz marcados.

Ejercicio 1

Realice todos los cambios necesarios en el proyecto de backend para implementar el nuevo requisito asegurándose de que los test se ejecutan correctamente.

Recuerde que puede correr los tests con:

```
npm run test:backend
```



Céntrese en aquellos añadidos al archivo: `productsVisibility.test.js` .

Enlaces de ayuda:

- <https://sequelize.org/docs/v6/core-concepts/assocs/#special-methodsmixins-added-to-instances>
- <https://sequelize.org/docs/v7/querying/operators/>
- <https://express-validator.github.io/docs/api/validation-chain/#isdate>
- <https://www.jsdocs.io/package/yup#date>

Ejercicio 2

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

Puede renderizar el icono de fijado propuesto con



Restaurant Detail

Casa Félix



Cocina Tradicional
Spanish

+ Create product



Coca-cola

33 cc

1.50€



Edit



Delete



Promoción ensalada

fsdgfdsgfd

12.00€

Is about to dissapear!



Edit



Delete



Water

50 cc

1.00€



Edit



Delete



My Restaurants



Control Panel



Profile



Edit Product

Name:

Ensaladilla

Description:

Tuna salad with mayonnaise

Price:

2.5

Order/position to be rendered:

1

Starters



Visible until:

2024-05-23

Is it available?



Product image:



 Save



My Restaurants



Control Panel



Profile



Create Product

Name:

New product

Description:

new

Price:

12

Order/position to be rendered:

1

Sides



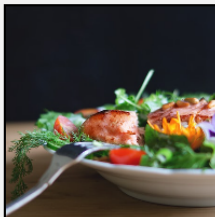
Visible until:

2024-03-01

Is it available?



Product image:



visibleUntil-The visibility must finish after the current date.
availability-Cannot set the availability and visibility at the same time.

 Save



My Restaurants



Control Panel



Profile



Create Product

Name:

Description:

Price:

Order/position to be rendered:

Select the product category

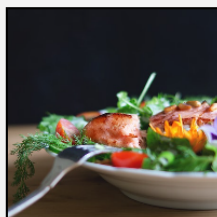


Visible until:

Is it available?



Product image:



 Save



My Restaurants



Control Panel



Profile

BACKEND

1. Añadimos propiedad en Model y Migration

```
Product.init({
  name: DataTypes.STRING,
  description: DataTypes.STRING,
  price: DataTypes.DOUBLE,
  image: DataTypes.STRING,
  order: DataTypes.INTEGER,
  availability: DataTypes.BOOLEAN,
  restaurantId: DataTypes.INTEGER,
  productCategoryId: DataTypes.INTEGER,
  visibleUntil: DataTypes.DATE
}, {
  sequelize,
  modelName: 'Product'
})
return Product

module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('Products', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      name: {
        allowNull: false,
        type: Sequelize.STRING
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE,
        defaultValue: new Date()
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE,
        defaultValue: new Date()
      },
      visibleUntil: {
        allowNull: true,
        type: Sequelize.DATE,
        defaultValue: null
      }
    })
  },
}
```

2. Modificamos el show de RestaurantController

“Si un propietario establece este momento, el producto solo estará visible hasta la fecha (inclusive).”

```
const show = async function (req, res) {
  // Only returns PUBLIC information of restaurants
  try {
    const today = new Date()
    const restaurant = await Restaurant.findByPk(req.params.restaurantId,
    {
      attributes: { exclude: ['userId'] },
      include: [{
        model: Product,
        as: 'products',
        include: { model: ProductCategory, as: 'productCategory' },
        where: {
          visibleUntil: { [Sequelize.Op.or]:
            [{ [Sequelize.Op.eq]: null }, { [Sequelize.Op.gt]: today } ] }
        },
      },
      {
        model: RestaurantCategory,
        as: 'restaurantCategory'
      }
    ]],
    order: [[{ model: Product, as: 'products' }, 'order', 'ASC']]
  }

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

3. Añadimos los nuevos metodos necesarios para las reglas de negocio

“Un propietario no podrá establecer un momento de fin anterior al momento actual.”

```
const checkFechaCorrecta = async (value, { req }) => {
  const today = new Date()
  try {
    if (value < today) {
      return Promise.reject(new Error('Un propietario no podrá establecer un momento de fin anterior al momento actual.'))
    }
    return Promise.resolve()
  } catch (err) {
    return Promise.reject(new Error(err))
  }
}
```

“Un propietario no podrá establecer un producto como no disponible y, al mismo tiempo, un momento de fin”

```
const checkSiVisible = async (value, { req }) => {
  try {
    if (value && !req.body.availability) {
      return Promise.reject(new Error('Un propietario no podrá establecer un producto como no disponible y, al mismo tiempo, un momento de fin.'))
    }
    return Promise.resolve()
  } catch (err) {
    return Promise.reject(new Error(err))
  }
}

const create = [
  check('visibleUntil').isDate().toDate(),
  check('visibleUntil').custom(checkFechaCorrecta),
  check('visibleUntil').custom(checkSiVisible)
]
```

```
const update = [
  check('visibleUntil').isDate().toDate(),
  check('visibleUntil').custom(checkFechaCorrecta),
  check('visibleUntil').custom(checkSiVisible)
```

```
]
```

(el resto de propiedades del create y update se mantiene igual)

FRONTEND

1. Cambios en CreateProductScreen y EditProductScreen

```
const initialProductValues = { name: null, description: null, price: null, order: null, restaurantId: route.params.id, productCategoryId: null, availability: true, visibleUntil: null }
```

```
const validationSchema = yup.object().shape({
  name: yup
    .string()
    .max(255, 'Name too long')
    .required('Name is required'),
  price: yup
    .number()
    .positive('Please provide a positive price value')
    .required('Price is required'),
  order: yup
    .number()
    .nullable()
    .positive('Please provide a positive order value')
    .integer('Please provide an integer order value'),
  availability: yup
    .boolean(),
  productCategoryId: yup
    .number()
    .positive()
    .integer()
    .required('Product category is required'),
  visibleUntil: yup
    .date()
})
```

```
return (
  <Formik
    validationSchema={validationSchema}
    initialValues={initialProductValues}
    onSubmit={createProduct}>
    ({ handleSubmit, setFieldValue, values }) => (
      <ScrollView>
        <View style={{ alignItems: 'center' }}>
          <View style={{ width: '60%' }}>
            <InputItem
              name='name'
              label='Name:'
            />
          </View>
        </View>
      </ScrollView>
    )
  </Formik>
)
```



```

        <InputItem
            name='visibleUntil'
            label='Visible Until:'
        />
        <InputItem
            name='description'
            label='Description:'
        />

```

El resto igual

2. Cambios en RestaurantDetailScreen

“Finalmente, los productos que estén a una semana o menos de desaparecer aparecerán en la interfaz marcados.”

```

        <View style={{ flexDirection: 'row ', justifyContent: 'flex-end',
topLeft: 10 }}>
            {item.visibleUntil !== null && (new Date(item.visibleUntil) - new
Date()) / (1000 * 60 * 60 * 24) < 7 && (<TextRegular style={{ color:
'red' }}>is about to disappear!</TextRegular>
            )}
        </View>

```