FORMULARIO LABORATORIO IISSI 2

CLASES DE BACKEND:

- MODELS:

Para añadir propiedades nuevas. SIEMPRE IGUAL.

Ejem:

```
Restaurant.init({
   name: DataTypes.STRING,
   description: DataTypes.TEXT,
promoted: {
     type: DataTypes.BOOLEAN,
     defaultValue: false
   },
   status: {
     type: DataTypes.ENUM,
     values: [
       'online',
       'offline',
       'closed',
       'temporarily closed'
     ]
   }
}
```

- ROUTES:

En el se establece una ruta para cada propiedad, se asignan restricciones de los middlewares y un controller para así poder acceder a ellos desde el Frontend.

En el deben añadirse todas las propiedades consultables de los models con sus respectivas restricciones y controllers.

Ejem:

```
module.exports = (options) => {
  const app = options.app
  const middlewares = options.middlewares

  app.route('/restaurants/:restaurantId')
    .get(RestaurantController.show)
    .put(
      middlewares.isLoggedIn,
      middlewares.hasRole('owner'),
      middlewares.checkEntityExists(Restaurant, 'restaurantId'),
      middlewares.checkRestaurantOwnership,
      upload,
      RestaurantValidation.update,
      middlewares.handleValidation,
      RestaurantController.update)
    .delete(
      middlewares.isLoggedIn,
      middlewares.hasRole('owner'),
      middlewares.checkEntityExists(Restaurant, 'restaurantId'),
      middlewares.checkRestaurantOwnership,
      RestaurantController.destroy)

app.route('/restaurants/:restaurantId/promoted')
     .patch(
       middlewares.isLoggedIn,
       middlewares.hasRole('owner'),
       middlewares.checkEntityExists(Restaurant, 'restaurantId'),
       middlewares.checkRestaurantOwnership,
       RestaurantController.promote
     )
}
```

- CONTROLLERS:

La clase controllers contiene todas las funciones necesarias:

- Index → Para mostrar los datos (SIEMPRE ESTA y SIEMPRE IGUAL):
  - Se añaden las nuevas propiedades.
  - Se establece el orden para mostrar los datos.
- Create → Para crear un datos nuevo.(SIEMPRE IGUAL, solo cambian restricciones)
- Update → Para actualizar un datos ya creado.(SIEMPRE IGUAL, solo cambian restricciones)
- Destroy → Para eliminar un dato existente. (SIEMPRE IGUAL)
- Show → Para mostrar un dato concreto.(Parecida al index pero solo para un dato)

Ejem:

```
exports.promote = async function (req, res) {
  const t = await models.sequelize.transaction()
  try {
    const existingPromotedRestaurant = await Restaurant.findOne({ where: { userId: req.user.id, promoted: true } })
    if (existingPromotedRestaurant) {
      await Restaurant.update(
        { promoted: false },
        { where: { id: existingPromotedRestaurant.id } },
        { transaction: t }
      )
    }
    await Restaurant.update(
      { promoted: true },
      { where: { id: req.params.restaurantId } },
      { 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)
  }
}
```

- VALIDATIONS:

Se establecen las restricciones para cada propiedad definida en el model y se crean funciones para cada restricción más elaborada. Restricciones para create y Update.

Ejem:

```
const checkPromoted =  async (owner, value) => {
if(value){
  try{
    const restaurant = await Restaurant.findAll({ where: {userId: owner, promoted:true }})
    if(restaurant.length !== 0){
      return Promise.reject(new Error('You can´t have more than one restaurant promoted'))
  }
}catch(err){
  return Promise.reject(new Error(err))
}
}
  return Promise.resolve('ok')
}

module.exports = {
  create: [
    check('name').exists().isString().isLength({ min: 1, max: 255 }).trim(),
    check('description').optional({ nullable: true, checkFalsy: true }).isString().trim(),
    check('address').exists().isString().isLength({ min: 1, max: 255 }).trim(),
    check('promoted').custom(async (value,{req}) => {
      return checkPromoted(req.user.id, value)
    })
  ]
```

- MIGRATIONS

En esta clase se definen las tablas de datos que aparecerán en la base de datos.

Hay que añadir las propiedades nuevas del model.

Ejem:

```
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('Restaurants', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      name: {
        allowNull: false,
        type: Sequelize.STRING
      } }
```

CLASES DE FRONTEND:

- SCREENS:

En la clase screen se diseña cada una de las pantallas que queremos que tenga nuestra aplicación con las funciones:

- renderHeader → Se diseña la cabecera de la página
- renderDato → Se diseña el cuerpo de la página, como aparecen los datos, los botones que se quieren añadir… (para cada botón se debe crear una función que lo maneje)
- renderEmptyDatoList
- fechtDato → Función para recopilar los datos y poder pintarlos en el cuerpo.
- Return:
  - FlatList → style, data, renderItem, keyExtractor, ListHeaderComponent,ListEmptyComponent
  - DeleteModal, ConfirmationModal → onConfirm, onCancel, isVisible
  - SI ES UN CUESTIONARIO:
    - Formik → en el se definen todos los campos del cuestionario
      - validationSchema, initialValues, onSubmit
      - ScrollView, View
      - InputItem → Para campos en los que se puede escribir.
      - DropDownPicker → para seleccionar propiedades prederterminadas (desplegable)
      - Pressable → Para campos en los que al presionar haya que subir algún tipo de dato o se realice alguna función.
      - Switch → 'Interruptor' para habilitar o no una función
      - Pressable Save → Siempre igual
- Styles → definimos estilos.

- ENDPOINTS :

En esta clase se definen las direcciones para ejecutar cada función y esta sea efectiva en la base de datos (para conectarse al Backend)

Importante añadir cada función al exports e importar todo.

Ejem:

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

- STACKS:

En esta clase se añaden todas las nuevas pantallas que se creen. MUY IMPORTANTE AÑADIRLAS!!!

Ejem:

```
export default function RestaurantsStack () {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name='RestaurantsScreen'
        component={RestaurantsScreen}
        options={{
          title: 'My Restaurants'
        }} />
      <Stack.Screen
        name='RestaurantDetailScreen'
        component={RestaurantDetailScreen}
        options={{
          title: 'Restaurant Detail'
        }} />
</Stack.Navigator>
  )
}
```