CONTROLLERS

Los controladores son los componentes que se encargan de implementar toda la lógica de negocio, es decir operaciones sobre la BD. Cada método de controlador recibe una solicitud `req` y un `res` objeto de respuesta. El objeto de solicitud `req` representa la **solicitud HTTP** y tiene las siguientes propiedades:

- `req.body` representa los datos que provienen del cliente
- `req.params` representa los parámetros de ruta. . Por ejemplo, si en la ruta de una solicitud tenemos `/:restaurantId`, entonces cualquier valor que se ponga en lugar de restaurantId será accesible a través de `req.params.restaurantId`
- `req.query` representa parámetros de consulta. . Por ejemplo, si la URL de una solicitud incluye `?status=activo`, entonces podrías acceder al valor `activo` usando `req.query.status`.
- `req.user` representa el usuario que inició sesión y realizó la solicitud. Por lo tanto, si necesitas información sobre el usuario que está realizando la solicitud, puedes encontrarla aquí

Por otro lado, el objeto de respuesta `res` representa la **respuesta HTTP** que envía una aplicación Express cuando recibe una solicitud HTTP. Tiene los siguientes métodos:

- `res.json(entityObject)` Este método envía al cliente un objeto `entityObject` como un documento JSON con un código de estado HTTP 200. Por ejemplo, si queremos enviar información sobre un restaurante al cliente, usaremos `res.json(restaurante)` y el cliente recibirá los datos del restaurante en formato JSON.
- `res.json(message)` devuelve una cadena `message` al cliente como un documento JSON con código de estado HTTP 200.
- `res.status(500).send(err)` devuelve el `err` objeto (que normalmente incluye algún tipo de mensaje de error) y un código de estado HTTP 500 al cliente

CREAR UNA ENTIDAD:

```
const create = async function (req, res) {
  //SINCRONO:: se crea directamente
  const newRestaurant = Restaurant.build(req.body)
// creamos un nuevo restaurante con la informacion del cliente
  newRestaurant.userId = req.user.id // usuario actualmente autenticado
  try{
    //ASINCRONO: se necesita algo de tiempo en realizarse
    const restaurant = await newRestaurant.save()
// guardamos el restaurante nuevo en la bd
    res.json(restaurant) // devolvemos en formato json el restaurante
  }
  catch(err){
    res.status(500).send(err)
  }
}
```

LEER UNA ENTIDAD-LISTADO

Así seria incluyendo todo y sin ninguna pecularidad:

```
const index = async function (req, res) {
  try{
    const restaurants = Restaurant.findAll()
    res.json(restaurants)
  }
  catch(err){
    res.status(500).send(err)
  }
}
```

```
const index = async function (req, res) {
  try {
    const restaurants = await Restaurant.findAll(
      {
        attributes: { exclude: ['userId'] },
        include:{
        model: RestaurantCategory, as: 'restaurantCategory' },
        order:
[[{ model: RestaurantCategory, as: 'restaurantCategory'}, 'name', 'ASC']]
      }
    )
    res.json(restaurants)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

```
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'
        }]
      })
    res.json(restaurants)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

LEER UNA ENTIDAD ESPECIFICA + DETALLES

Así seria incluyendo todo y sin ninguna pecularidad:

```
const show = async function (req, res) {
  try{
    const restaurant = await Restaurant.findByPk(req.params.restaurantId)
    res.json(restaurant)
}
catch(err){
  res.status(500).send(err)
}
}
```

```
  const show = async function (req, res) {
    try{
      const restaurant = await Restaurant.findByPk(req.params.restaurantId,
        //el id del restaurante a consultar esta en la ruta, yse pasa como
  parametro
        {
          attributes: { exclude: ['userId'] },
          include: [
            {
            model: Product,
            as: 'products',
            include : { model: ProductCategory, as: 'productCategory' },
          },
          {
            model: RestaurantCategory,
            as: 'restaurantCategory'
          }
        ],
        order: [[{model:Product, as: 'products'}, 'order', 'ASC']],
        }
      )
      res.json(restaurant)
  }
  catch(err){
    res.status(500).send(err)
  }
  }
```

Product a su vez tiene una asociación luego tenemos q incluirla tb!!

ACTUALIZAR UNA ENTIDAD

```
const update = async function (req, res) {
  try {
    await Restaurant.update(req.body,
{ where: { id: req.params.restaurantId } })
    const updatedRestaurant = await
Restaurant.findByPk(req.params.restaurantId)
    res.json(updatedRestaurant)
  } catch (err) {
    res.status(500).send(err)
  }
}
```

ELIMINAR UNA ENTIDAD

```
const destroy = async function (req, res) {
  try {
    const result = await Restaurant.destroy({ where: { id:
req.params.restaurantId } })
    let message = ''
    if (result === 1) {
      message = 'Sucessfuly deleted restaurant id.' +
      req.params.restaurantId}
    else {
      message = 'Could not delete restaurant.'}
    res.json(message)
  } catch (err) {
    res.status(500).send(err) }}
```

ROUTES

```
const loadFileRoutes = function (app) {
  //FR1: Listado de restaurantes: los clientes podrán consultar todos los
restaurantes.
  app.route('/restaurants')
    .get(RestaurantController.index)
    .post(RestaurantController.create)

  //FR2: Detalles y menú de restaurantes: Los clientes podrán consultar
los detalles de los restaurantes y los productos que ofrecen + CRUD
  app.route('/restaurants/:restaurantId')
  .get(RestaurantController.show)
  .put(RestaurantController.update)
  .delete(RestaurantController.destroy)
```

```
app.route('/products')
  .post(
    isLoggedIn,
    hasRole('owner'),
    handleFilesUpload(['image'], process.env.PRODUCTS_FOLDER),
    ProductValidation.create,
    handleValidation,
    ProductMiddleware.checkProductRestaurantOwnership,
    ProductController.create)
```

Necesitamos implementar una ruta para poder obtenerlo. Siempre que queramos ver una entidad concreta vamos a preguntar si existe: `checkEntityExists`

```
app.route('/restaurants/:restaurantId')
  .get(
    checkEntityExists(Restaurant, 'restaurantId'),
    RestaurantController.show)
```

MIDDLEWARES

AUTENTICACION/IDENTIFICACION

```
const hasRole = (...roles) => (req, res, next) => {
  if (!req.user) {
    return res.status(403).send({ error: 'Not logged in' })
  }
  if (!roles.includes(req.user.userType)) {
    return res.status(403).send({ error: 'Not enough privileges' })
  }
  return next()}

const isLoggedIn = (req, res, next) => {
  passport.authenticate('bearer', { session: false })(req, res, next)
}
```

IDENTIFICACION EN BD

```
const checkEntityExists = (model, idPathParamName) => async (req, res,
next) => {
  try {
    const entity = await model.findByPk(req.params[idPathParamName])
    if (!entity) { return res.status(404).send('Not found') }
    return next()
  } catch (err) {
    return res.status(500).send(err)
  }
}
```

GENERAL

```
const handleValidation = async (req, res, next) => {
  const err = validationResult(req)
  if (err.errors.length > 0) {
    res.status(422).send(err)
  } else {
    next()
  }
}
```

IMÁGENES/ARCHIVOS

```
const handleFilesUpload = (fieldNames, folder) => (req, res, next) => {
  const multerInstance = createMulter(fieldNames, folder)
  multerInstance(req, res, (err) => {
    if (err) {
      res.status(500).send({ error: err.message })
    } else {
      addFilenameToBody(req, fieldNames)
      next()
    }
  })
}
```

```
const checkRestaurantOwnership = async (req, res, next) => {
  try {
    const restaurant = await Restaurant.findByPk(req.params.restaurantId)
    if (req.user.id === restaurant.userId) {
      return next()
    }
    return res.status(403).send('Not enough privileges. This entity does not belong to you')
  } catch (err) {
    return res.status(500).send(err)
  }
}

const restaurantHasNoOrders = async (req, res, next) => {
  try {
    const numberOfRestaurantOrders = await Order.count({
      where: { restaurantId: req.params.restaurantId }
    })
    if (numberOfRestaurantOrders === 0) {
      return next()
    }
    return res.status(409).send('Some orders belong to this restaurant.')
  } catch (err) {
    return res.status(500).send(err.message)
  }
}
```

VALIDATIONS

VALIDATOR	DESCRIPCION
`contains(str, seed)`	Verifica si la cadena contiene la semilla.
`equals(str, comparison)`	Verifica si la cadena coincide con la comparación.
`isEmail(str)`	Verifica si la cadena es un correo electrónico válido
`isEmpty(str)`	Verifica si la cadena tiene una longitud de cero
`isLength(str, { min, max })`	Verifica la longitud de la cadena.
`isURL(str)`	Verifica si la cadena es una URL válida
`isInt(str)`	Verifica si la cadena es un entero
`isFloat(str)`	Verifica si la cadena es un número decimal
`isBoolean(str)`	Verifica si la cadena es un booleano
`isDate(str)`	Verifica si la cadena es una fecha válida.
`exists()`	

SANITIZADOR	DESCRIPCION
`toBoolean(input [, strict])`	Convierte el input a un booleano. Todo excepto '0', 'false' y retorna true. En modo estricto, solo '1' y 'true' retornan `true`.
`toDate(input)`	Convierte el input a una fecha, o `null` si el input no es una fecha.
`toFloat(input)`	Convierte el input a un número decimal, o `NaN` si el input no es un número decimal.
`toInt(input [, radix])`	Convierte el input a un número entero, o `NaN` si el input no es un número entero.
`trim(input [, chars])`	Recorta caracteres (espacios en blanco por defecto) de ambos lados del input.

```
const checkCreationOrder2 = async (value, { req }) => {
  try {
    // 1
    if (req.body.products.length < 1) {
      return Promise.reject(new Error(' El array de productos no puede estar vacio'))
    }
    // 2
    for (const product of value) {
      if (product.productId < 1) {
        return Promise.reject(new Error(' El id no puede ser 0'))
      }
      const productBD = await Product.findByPk(product.productId)
      // 3
      if (!productBD.availability) { return Promise.reject(new Error(' Todos los productos tienen que estar disponibles')) }

      // 4
      if (productBD.restaurantId !== req.body.restaurantId) { return Promise.reject(new Error(' Todos los productos tienen que estar disponibles')) }
    }
    return Promise.resolve()
  } catch (err) {
    return Promise.reject(new Error(err))
  }
}

const create = [
// 1. Check that restaurantId is present in the body and corresponds to an existing restaurant
  check('restaurantId').exists(),
  check('price').default(null).optional({ nullable: true }).isFloat().toFloat(),
  check('address').exists().isString().isLength({ min: 1, max: 255 }).trim(),
  check('products').custom(checkCreationOrder2),
  // 2- and quantity greater than 0
  check('products.*.quantity').isInt({ min: 1 }).toInt()
```

```
const checkUpdatedOrder = async (value, { req }) => {
  try {
    if (req.body.products.length < 1) {
      return Promise.reject(new Error('The array of products is empty'))
    }
    for (const p of value) {
      if (p.productId < 1) {
        return Promise.reject(new Error('The product has not valid Id'))
      }
      const product = await Product.findByPk(p.productId)
      if (!product.availability) {
        return Promise.reject(new Error('The product is not available'))
      }
      const original = await Order.findByPk(req.params.orderId)
      if (product.restaurantId !== original.restaurantId) {
        return Promise.reject(new Error('The product does not belong to the original restaurant'))
      }
    }
    return Promise.resolve()
  } catch (err) {
    return Promise.reject(new Error(err))
  }
}


// TODO: Include validation rules for update that should:
// 1. Check that restaurantId is NOT present in the body.
// 2. Check that products is a non-empty array composed of objects with productId and quantity greater than 0
// 3. Check that products are available
// 4. Check that all the products belong to the same restaurant of the originally saved order that is being edited.
// 5. Check that the order is in the 'pending' state.

const update = [
  check('userId').not().exists(),
  check('restaurantId').not().exists(),
  check('address').exists().isString().isLength({ min: 1, max: 255 }).trim(),
  check('products').custom(checkUpdatedOrder),
  check('products.*.quantity').isInt({ min: 1 }).toInt()
]
```

FORMIK

Los formularios del Fronted deben validarse antes de enviarlos al Frontend. Por ejemplo: un input para correo electrónico debe contener un correo electrónico válido, o la contraseña debe tener un tamaño mínimo.

Podemos escribir estas validaciones o usando Yup para crear un **schema validation** (required, email, strings, numbers, dates or default values):

STRING (`yup.string()`)	METODOS DE VALIDACIÓN
`required()`	Requiere que el campo no esté vacío.
`max(length)`	Requiere que el campo tenga una longitud máxima especificada.
`min(length)`	Requiere que el campo tenga al menos una longitud mínima especificada.
`email()`	Requiere que el campo tenga un formato de correo electrónico válido.
`url()`	Requiere que el campo tenga un formato de URL válido.
`matches(regex, message)`	Requiere que el campo coincida con una expresión regular.

NUMBER (`yup.number()`)	METODOS DE VALIDACIÓN
`required()`	Requiere que el campo no esté vacío.
`max(number)`	Requiere que el campo tenga una longitud máxima especificada.
`min(number)`	Requiere que el campo tenga al menos una longitud mínima especificada.
`positive()`	Requiere que el campo tenga un formato de correo electrónico válido.
`negative()`	Requiere que el valor del campo sea un número negativo.
`integer()`	Requiere que el valor del campo sea un número entero.

DATE (`yup.date()`)	METODOS DE VALIDACIÓN
`required()`	Requiere que el campo no esté vacío.
`max(date)`	Requiere que la fecha sea posterior o igual a una fecha mínima.
`min(date)`	Requiere que la fecha sea anterior o igual a una fecha máxima.

BOOLEAN (`yup.boolean()`)	METODOS DE VALIDACIÓN
`required()`	Requiere que el campo no esté vacío.

OBJECT (yup.object())	METODOS DE VALIDACIÓN
required()	Requiere que el campo no esté vacío.
shape(fields)	Define el esquema de validación para las propiedades del objeto.

ARRAY (yup.array())	METODOS DE VALIDACIÓN
required()	Requiere que el campo no esté vacío.
max(lenght)	Requiere que el array tenga una longitud máxima especificada.
min(lenght)	Requiere que el array tenga al menos una longitud mínima.
of(schema)	Define el esquema de validación para los elementos del array.

OPERATORS

OPERACIONES DE IGUALDAD

`Sequelize.Op.eq` →

```
SELECT * FROM "posts" WHERE "authorId" IS NULL
```

```
Post.findAll({
  where: {
    authorId: { [Op.is]: null },
  },
})
```

`Sequelize.Op.in` →

```
SELECT * FROM "posts" WHERE "authorId" IN (2, 3)
```

```
Post.findAll({
  where: {
    authorId: [2, 3],
  },
})
```

`Sequelize.Op.is` →

```
SELECT * FROM "posts" WHERE "authorId" = 12
```

```
Post.findAll({
  where: {
    authorId: { [Op.eq]: 12 },
  },
})
```

`Sequelize.Op.isNot:`

```
SELECT * FROM "posts" WHERE "authorId" IS NOT NULL
```

```
Post.findAll({
  where: {
    authorId: { [Op.isNot]: null },
  },
})
```

OPERACIONES DE COMPORATORS

`Sequelize.Op.gt:` Mas grande que (>)

```
SELECT * FROM "posts" WHERE "commentCount" > 10
```

```
Post.findAll({
  where: {
    commentCount: { [Op.gt]: 10 },
  },
})
```

`Sequelize.Op.gte:` Mayor qué o igual a (>=)

```
SELECT * FROM "posts" WHERE "commentCount" >= 10
```

```
Post.findAll({
  where: {
    commentCount: { [Op.gte]: 10 },
  },
})
```

`Sequelize.Op.lt:` Menos que (<)

```
SELECT * FROM "posts" WHERE "commentCount" < 10
```

```
Post.findAll({
  where: {
    commentCount: { [Op.lt]: 10 },
  },
})
```

`Sequelize.Op.lte:` Menos que o igual a (<=)

```
SELECT * FROM "posts" WHERE "commentCount" <= 10
```

```
Post.findAll({
  where: {
    commentCount: { [Op.lte]: 10 },
  },
})
```

OPERACIONES BETWEEN

`Sequelize.Op.between`→

```
SELECT * FROM "posts" WHERE "commentCount" BETWEEN 1 AND 10
```

```
Post.findAll({
  where: {
    commentCount: { [Op.between]: [1, 10] },
  },
})
```

`Sequelize.Op.notBetween`→

```
SELECT * FROM "posts" WHERE "commentCount" NOT BETWEEN 1 AND 10
```

```
Post.findAll({
  where: {
    commentCount: { [Op.notBetween]: [1, 10] },
  },
})
```

OPERACIONES IN

`Sequelize.Op.in` →

```
SELECT * FROM "posts" WHERE "authorId" IN (2, 3)
```

```
Post.findAll({
  where: {
    authorId: { [Op.in]: [2, 3] },
  },
})
```

`Sequelize.Op.notIn` →

```
SELECT * FROM "posts" WHERE "authorId" NOT IN (2, 3)
```

```
Post.findAll({
  where: {
    authorId: { [Op.notIn]: [2, 3] },
  },
})
```

LOGICAL OPERATOR

`Sequelize.Op.Op.and`→

```
SELECT * FROM "posts" WHERE "authorId" = 12 AND "status" = 'active'
```

`Sequelize.Op.Op.or`→

```
SELECT * FROM "posts" WHERE "authorId" = 12 OR "status" = 'active'
```

```
Post.findAll({
  where: {
    [Op.or]: [{ authorId: 12 }, { status: 'active' }],
  },
})
```

```
SELECT * FROM "posts" WHERE "authorId" = 12 OR "authorId" = 24;
```

```
Post.findAll({
  where: {
    [Op.or]: [{ authorId: 12 }, { authorId: 24 }],
  },
});
```

OPERACIONES STRING

`Sequelize.Op.like`→

```
SELECT * FROM "posts" WHERE "title" LIKE '%The Fox & The Hound%'
```

```
Post.findAll({
  where: {
    title: { [Op.like]: '%The Fox & The Hound%' },
  },
})
```

`Sequelize.Op.notLike`→

```
SELECT * FROM "posts" WHERE "title" NOT LIKE '%The Fox & The Hound%
```

```
Post.findAll({
  where: {
    title: { [Op.notLike]: '%The Fox & The Hound%' },
  },
})
```

ARRAY OPERATOR

`Sequelize.Op.contains`→

```
SELECT * FROM "posts" WHERE "tags" @> ARRAY['popular', 'trending']
```

```
Post.findAll({
  where: {
    tags: { [Op.contains]: ['popular', 'trending'] },
  },
})
```

`Sequelize.Op.contained`→

```
SELECT * FROM "posts" WHERE "tags" <@ ARRAY['popular', 'trending']
```

```
Post.findAll({
  where: {
    tags: { [Op.contained]: ['popular', 'trending'] },
  },
})
```