Routes and Controllers

NEXT TESTING WITH POSTMAN

Routes are the essence of Express. How does the client communicate with the server? How does the server act on request and send responses? These are the questions that your routes answer.

Routes are defined with the signature:

```
app.method('<path>', callbackFunction)

// Examples
app.get('/', callbackFunction)
app.post('/create', callbackFunction)
```

The `callbackFunction` is what I like to extract from the route definition into a self-contained file and refer to as **Controllers**.

Table of Contents

For more info on using Node routing and controllers, view our other course: https://scotch.io/courses/routing-node-applications

Let's create the routes needed. The root route (`/`) just tells us that our app is running and nothing more:

```
// ./routes/index.js
const express = require('express');
const router  = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'App running' });
});

module.exports = router;
```

The `router` object is a utility method that is built on top of express to make RESTful APIs even easier.

When sending templates (like ejs), rather than using `res.send` or `res.json` which are the most common, we use `res.render`.

API Routes

```
// routes/api/index.js
const express                = require('express');
const router                 = express.Router();
const productController      = require('../../controllers/product')
const manufacturerController = require('../../controllers/manufacturer')

router.get('/manufacturers', manufacturerController.all);

router.get('/products', productController.all);
router.get('/products/:id', productController.byId);
router.post('/products', productController.create);
router.put('/products/:id', productController.update);
router.delete('/products/:id', productController.remove);

module.exports = router;
```

The only difference is, as discussed, we are passing in a reference to a function rather than defining it here. These functions are extracted and called controllers

Note that the `router` object provides:

- `get`: Reading resources
- `post`: Creating new resources
- `put`: Updating existing resources and
- `delete`: Removing existing resources

Just as we saw in Vue's routing, it can accept parameters (`:id`) for dynamic operations. `GET /products/:id` does not always return a particular product, rather, it returns a product with the ID provided in the path when making a request.

Controllers

Controllers are the callback functions we passed to the `router` methods. We are already referring to these controllers in the routes above, I think it's time we create them:

```
// ./controllers/manufacturer
const Model = require('../model');
const {Product, Manufacturer} = Model;

const manufacturerController = {
    all (req, res) {
      // Returns all manufacturers
        Manufacturer.find({})
            .exec((err, manufacturers) => res.json(manufacturers))
    }
};

module.exports = manufacturerController;
```

```
// ./controllers/product
const Model = require('../model');
const {Product, Manufacturer} = Model;

const productController = {
    all (req, res) {
        // Returns all products
        Product.find({})
            // alongside it's manufacturer
            // information
            .populate('manufacturer')
            .exec((err, products) => res.json(products))
    },
    byId (req, res) {
        const idParam = req.params.id;
        // Returns a single product
        // based on the passed in ID parameter
        Product
            .findOne({_id: idParam})
            // as well as it's manufacturer
            .populate('manufacturer')
            .exec( (err, product) => res.json(product) );
    },
    create (req, res) {
        const requestBody = req.body;
        // Creates a new record from a submitted form
        const newProduct = new Product(requestBody);
        // and saves the record to
        // the data base
        newProduct.save( (err, saved) => {
            // Returns the saved product
            // after a successful save
            Product
                .findOne({_id: saved._id})
                .populate('manufacturer')
                .exec((err, product) => res.json(product));
        } )
    },
    update (req, res) {
        const idParam = req.params.id;
        let product = req.body;
        // Finds a product to be updated
        Product.findOne({_id: idParam}, (err, data) => {
            // Updates the product payload
            data.name = product.name;
            data.description = product.description;
            data.image = product.image;
            data.price = product.price;
            data.manufacturer = product.manufacturer;
            // Saves the product
            data.save((err, updated) => res.json(updated));
        })
    },
    remove (req, res) {
        const idParam = req.params.id;
        // Removes a product
        Product.findOne({_id: idParam}).remove( (err, removed) => res.json({idParam} )
    }
};

module.exports = productController;
```

The function signature follows exactly the signature of a callback for a route argument.

Now that we have our API defined, let's test the APIs with a tool called POSTman in the next lesson. Hang in there, we're almost done with our backend API and we can get back to the Vue goodness soon.