Steps to Building an API

A Camariana - MDI'a National Diploma in Computer Science

Once you identify your project name and have your schema in place, follow the followings steps to get up and running with your project.

Step 1

Go to github.com and create a new repository.

- The name of the repository should be the name of your project.
- add a description if you want
- add a README
- add a .gitignore but choose **node**

For me, my project name is **bintaface**

Once, you create the repository clone it to your local computer.

Next, open the project with VS Code

Step 2

Once you open the project, if all is good, you should see the following two files

- .gitignore
- README.md

Nice progress, let's move on.

Within VS Code go to Terminal on the menu on top and click on New Terminal

Within the terminal type the following to initialise the project

```
npm init -y
```

Note: this will automatically create a package.json file

Step 3

Now lets install the packages we need to for our API. Here's is a list of packages we need for now

- express
- mongoose
- nodemon

```
npm install express mongoose
```

and we install nodemon as dev dependency

```
npm install --save-dev nodemon
```

Note: You need to be connected to the internet for this work

You also need to make sure you have installed mongoDB locally in your system.

Step 4

Project struture,

Within the root of the project, create a folder called **src**

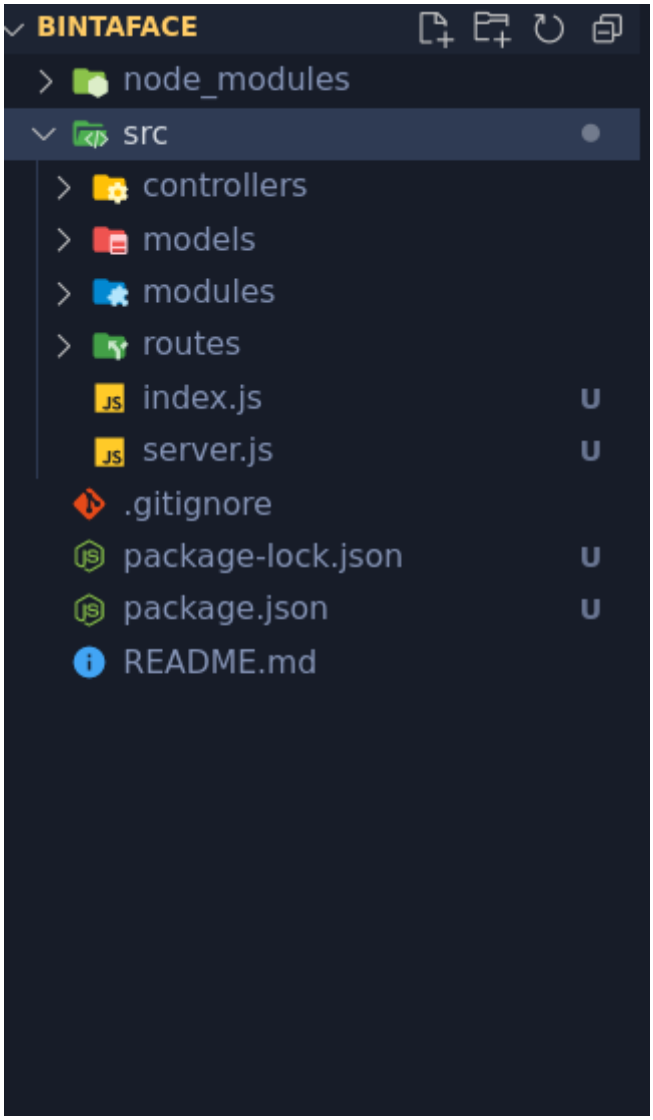
Inside the /src folder create the following folders:

- routes
- controllers
- models
- modules or utilities (the same idea, stick to one)

Inside the /src folder create the following files:

- index.js
- server.js

Here's is an example of how you project should look like for now.



Step 5

The initial server code for our express API

Within the **server.js** do the following code

```
const express = require('express');
const app = express();



app.use(express.json());
app.use(express.urlencoded({ extended: true }));


app.get('/', (req, res) => {
  res.status(200).json({ message: 'Peace' });
});


module.exports = app;
```

And within the **index.js** do the following code

```
const app = require('./server');

const PORT = 3001;

app.listen(PORT, () => {
    console.log(`Rest API listening on port ${PORT}`);
});
```

So alot is happening here with few lines of code, thanks to the power of express.

Everything will be explain soon

Let's update our **package.json** to include the following start script in the "scripts" section:

The script section looks like this

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
},
```

Now change it to this

```
"scripts": {
   "start": "node src/index.js",
   "dev": "nodemon src/index.js"
},
```

We can check to see if things are working as expected by running our newly created start script.

Within the terminal in VS Code, run the following command

```
npm run dev
```

You should see nodemon doing its thing and

a message saying Rest API listening on port 3001

In your browser, if you go to the address, http://localhost:3001/ you should see

```
{
  "message": "Peace"
}
```

Congrats, you have build a simple API.

Step 6

Connecting to our database

Before we actually connect to our database, let's create a connection module to use to connect to our database.

Within the **/modules** folder, create a file called **connect.js** and do the following code

```
const mongoose = require('mongoose');

const URI = 'mongodb://localhost:27017/bintaface';

const connect = () => {
    return mongoose.connect(URI, {
        useNewUrlParser: true,
        useUnifiedTopology: true,
        autoIndex: false,
    })
};

module.exports = connect;
```

Note: Don't forget to update the database name at the end of the URI to your own database name

this line

```
const URI = 'mongodb://localhost:27017/bintaface';
```

Next up let's require our connect module in the **server.js** file and connect to our database

Here's the code to that,

First we require the connect module like this:

```
const connect = require('./modules/connect');
```

And here is the connection function to include in the **server.js** after requiring it

```
// Connect to Database
void (async () => {
  try {
    await connect();
    console.log('connected to database');
  } catch (error) {
    console.log('error connecting to database:', error.message);
  }
})();
```

This will automatically create your database in mongoDB, you can use compass to make sure the database is created by simply looking at the list of databases.

Here's the rest of the updated **server.js** file

```
const express = require('express');
const app = express();

const connect = require('./modules/connect');


app.use(express.json());
app.use(express.urlencoded({ extended: true }));


// Connect to Database
void (async () => {
  try {
    await connect();
    console.log('connected to database');
  } catch (error) {
    console.log('error connecting to database:', error.message);
  }
})();


app.get('/', (req, res) => {
  res.status(200).json({ message: 'Peace' });
});



module.exports = app;
```

Nice progress, if you get here.

Step 7

Translating our schemas into models

This process is the same for all your models, the difference will be the name of the model/entity and it's attributes

1. **The model**

Let's start with the user model

Inside the **/models** folder, create a file called **user.js** and do the followig code

```
const mongoose = require('mongoose')

const userSchema = new mongoose.Schema(
  {
    email: {
      type: String,
      required: true,
      unique: true,
      trim: true
    },
    password: {
      type: String,
      required: true
    },
    firstname: {
      type: String,
      trim: true,
      maxlength: 25
    },
    minit: {
      type: String,
      trim: true,
      maxlength: 25
    },
    lastname: {
      type: String,
      trim: true,
      maxlength: 25
    },
    role: {
      type: String,
      enum: ['Admin', 'User'],
      default: 'User'
    }
  },
  { timestamps: true }
)
```

```
38
39
40  userSchema.set('toJSON', {
41    transform: (document, returnedObject) => {
42      returnedObject.id = returnedObject._id.toString()
43      delete returnedObject._id
44      delete returnedObject.__v
45      // the password should not be revealed
46      delete returnedObject.password
47    }
48  })
49
50
51  const User  = mongoose.model('user', userSchema)
52
53  module.exports = User
```

2. **The controller**

Once we create the user model and export the model, we can happily require the model in our controller and create a new user.

Inside the **/controllers** folder, create a file called **user.js** and do the followig code

```
1   const User = require('../models/user');
2
3   const createUser = async (req, res) => {
4     const content = req.body;
5
6     console.log(content);
7     try {
8       const user = await User.create({ ...content });
9
10      return res.status(201).json({ data: user });
11    } catch (error) {
12      console.log(error);
13      return res.status(500).end();
14    }
15  };
16
17
18
19  module.exports = {
20    createUser
21  };
```

3. **The router**

Then we can create a new file called **router.js** inside the **/routes** to handle all of routes of the entire application. But for now will do only the user router.

So, within the **router.js** file

let's do the following code

```
const express = require('express');
const router = express.Router();

const { createUser } = require('../controllers/user');


// User route
router.post('/user', createUser);


module.exports = router;
```

Finally and finally, we can add our routes to the **server.js** for routing our users to the various endpoints.

Let's update our **server.js** to include our route and put it to use.

First we are going to require the routes like this:

```
const routes = require('./routes/router');
```

then we can put it to use like this:

```
app.use('/api', routes);
```

Here is the updated **server.js** file with the above code included

```
const express = require('express');
const app = express();

const connect = require('./modules/connect');
const routes = require('./routes/router');


app.use(express.json());
app.use(express.urlencoded({ extended: true }));


// Connect to Database
void (async () => {
  try {
    await connect();
```

```
    console.log('connected to database');
  } catch (error) {
    console.log('error connecting to database:', error.message);
  }
})();


app.get('/', (req, res) => {
  res.status(200).json({ message: 'Peace' });
});


app.use('/api', routes);


module.exports = app;
```

Step 8

Let's test to see, if wee would be able to create a new user using the VS CODE rest client

Within the **root folder** of the project creates a new folder called **requests**

Inside requests create a file called **create-one.rest**

In that file, let's send a POST request to create a new user, see the code below for that.

```
1  POST http://localhost:3001/api/user HTTP/1.1
2  content-type: application/json
3
4  {
5    "email": "ac@camariana.gm",
6    "password": "secret",
7    "firstname": "A",
8    "minit": "",
9    "lastname": "Camariana"
10 }
```

Take note of line 1

```
1  POST http://localhost:3001/api/user HTTP/1.1
```

if you lcarefuly study the HTTP protocol, you will notice the following

- 3001 - that's the port
- /api/user - that our user enpoint

Within the VS CODE rest client, you will see on top of line 1 **Send Request**, click it.

If all goes well you should see a response like this:

```
HTTP/1.1 201 Created
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 210
ETag: W/"d2-/I0yK/fE80rXZdmXASBEqN4fWZs"
Date: Sat, 15 Oct 2022 07:06:03 GMT
Connection: close

{
  "data": {
    "email": "ac@camariana.gm",
    "firstname": "A",
    "minit": "",
    "lastname": "Camariana",
    "role": "User",
    "createdAt": "2022-10-15T07:06:03.201Z",
    "updatedAt": "2022-10-15T07:06:03.201Z",
    "id": "634a5bdb441c0826a1b5e678"
  }
}
```

This is good, this means, we have successfully created one user.

Please add another different 2 - 5 users by updating the content of the POST request in the create-one.rest file and send a request.

Step 9

In this step we will learn how to retrieve data for **all users**. This method should work for getting data from any table.

So let's go to the **user controller**, in the **/controllers/user.js** let's do the following code to get all the users from the database

```
const getAllUsers = async (req, res) => {
  try {
    const users = await User.find({});

    return res.status(201).json({ data: users });
  } catch (error) {
    console.log(error);
    return res.status(500).end();
  }
};
```

Once you add that handle, let's export it so that our router can see it. To do that update the module.exports to add `getAllUsers` to the export list.

```
module.exports = {
  createUser,
  getAllUsers
};
```

So far our **user.js** controller should look like this

```
const User = require('../models/user');

const createUser = async (req, res) => {
  const content = req.body;

  console.log(content);
  try {
    const user = await User.create({ ...content });

    return res.status(201).json({ data: user });
  } catch (error) {
    console.log(error);
    return res.status(500).end();
  }
};

const getAllUsers = async (req, res) => {
  try {
    const users = await User.find({});

    return res.status(201).json({ data: users });
  } catch (error) {
    console.log(error);
    return res.status(500).end();
  }
};



module.exports = {
  createUser,
  getAllUsers
};
```

Once you do that let's require it in the **/routers/route.js** file.

currently our require for the user controller looks like this

```
const { createUser } = require('../controllers/user');
```

let's add the new `getAllUsers` handler to that line. it should now look like this:

```
const { createUser, getAllUsers } = require('../controllers/user');
```

Finally, let's add the route to the user route

```
router.get('/user', getAllUsers);
```

The final code for the /route.js should like this

```
const express = require('express');
const router = express.Router();

const { createUser, getAllUsers } = require('../controllers/user');


// User route
router.post('/user', createUser);
router.get('/user', getAllUsers);


module.exports = router;
```

Note the new changes in the router.js file, especially the verb is now **get**, not **post** like when we were creating.

Let's test and see of we will get all the users using out rest client

Inside **requests** create a file called **get-all.rest**

In that file, let's send a get request to get all the users, see the code below for that.

```
GET http://localhost:3001/api/user
```

You see a list of all the users in the you db like the one below. I only have one user


```
{
  "data": [
    {
      "email": "ac@camariana.gm",
      "firstname": "A",
      "minit": "",
      "lastname": "Camariana",
      "role": "User",
      "createdAt": "2022-10-15T07:06:03.201Z",
      "updatedAt": "2022-10-15T07:06:03.201Z",
      "id": "634a5bdb441c0826a1b5e678"
```

```
    }
  ]
}
```

Copy and paste your results to slack, let me see your users.

Step 10

We wil now create a handler to get only **one user**.

So let's go back to the **user controller**. in the **/controllers/user.js** let's do the following code to get one user from the database

```
const getOneUser = async (req, res) => {
  const id = req.params.id
  try {
    const user = await User.findOne({ _id: id });

    if (!user) {
      return res.status(400).json({ message: 'user not found' });
    }
    return res.status(201).json({ data: user });
  } catch (error) {
    console.log(error);
    return res.status(500).end();
  }
};
```

Once you add that handle, let's export it so that our router can see it. like this below

```

module.exports = {
  createUser,
  getAllUsers,
  getOneUser
};
```

Next, require it in the **/routers/route.js** file. and add the route

```
const { createUser, getAllUsers, getOneUser } =
require('../controllers/user');
```

and add it to the route

```
// User route
router.post('/user', createUser);
router.get('/user', getAllUsers);
router.get('/user/:id', getOneUser);
```

Note how we add the parameter to the path `'/user/:id'`

Let's test and see of we will get one user using out rest client

Inside **requests** create a file called **get-one.rest**

In that file, let's send a get request to get one user, see the code below for that.

```
GET http://localhost:3001/api/user/6306ae79da5f62731c434830
```

Note: look at what is after the forward slash after the /user/... <-- this is the id of the user we are looking for.

If you use this id it will say not found. so copy one of the id from your users response and paste it after the /user/here and send a request, you should get that single user data like mine here

```
{
  "data": {
    "email": "ac@camariana.gm",
    "firstname": "A",
    "minit": "",
    "lastname": "Camariana",
    "role": "User",
    "createdAt": "2022-10-15T07:06:03.201Z",
    "updatedAt": "2022-10-15T07:06:03.201Z",
    "id": "634a5bdb441c0826a1b5e678"
  }
}
```

Step 11

We wil now create a handler to update only **one user**.

So let's go back to the **user controller**. in the **/controllers/user.js** let's do the following code to update one user from the database

```
const updateOne = async (req, res) => {
  const id = req.params.id
  const content = req.body

  try {
    const user = await User.findOneAndUpdate(
      { _id: id },
      content,
      { new: true }
    );

    if (!user) {
```

```
      return res.status(400).json({ message: 'user not found' });
    }
    return res.status(201).json({ data: user });
  } catch (error) {
    console.log(error);
    return res.status(500).end();
  }
};
```

Once you add that handler, let's export it, so that our router can see it. like this below

```
module.exports = {
  createUser,
  getAllUsers,
  getOneUser,
  updateOne
};
```

Next, require it in the **/routers/route.js** file. and add the route

```
const { createUser, getAllUsers, getOneUser, updateOne } =
require('../controllers/user');

```

and add it to the route

```
// User route
router.post('/user', createUser);
router.get('/user', getAllUsers);
router.get('/user/:id', getOneUser);
router.put('/user/:id', updateOne);
```

Note how we add the parameter to the path `'/user/:id'` and note the verb here is **put** instead of **get**

Let's test and see if we will update one user using out rest client

Inside **requests** create a file called **update-one.rest**

In that file, let's send a get request to update one user, see the code below for that.

```
PUT http://localhost:3001/api/user/634a5bdb441c0826a1b5e678
content-type: application/json

{
  "email": "ac@camariana.gm",
  "firstname": "A",
  "minit": "",
  "lastname": "Camara",
  "role": "Admin"
}
```

Note: look at what is after the forward slash after the /user/... <-- this is the id of the user we are looking for to update.

Look at what I am also updating, the **lastname** and the **role** to Admin

If you use this id it will say not found. so copy one of the id from your users response and paste it after the /user/here and send a request, you should get that single user data like mine here

```
{
  "data": {
    "email": "ac@camariana.gm",
    "firstname": "A",
    "minit": "",
    "lastname": "Camara",
    "role": "Admin",
    "createdAt": "2022-10-15T07:06:03.201Z",
    "updatedAt": "2022-10-21T12:26:10.448Z",
    "id": "634a5bdb441c0826a1b5e678"
  }
}
```

So what changes?

Before the changes the user data was like this:

```
{
  "data": {
    "email": "ac@camariana.gm",
    "firstname": "A",
    "minit": "",
    "lastname": "Camariana",
    "role": "User",
    "createdAt": "2022-10-15T07:06:03.201Z",
    "updatedAt": "2022-10-15T07:06:03.201Z",
    "id": "634a5bdb441c0826a1b5e678"
  }
}
```

So what changes?

I updated the **lastname** and the **role**. Compare the above file and the updated one above it.

Step 12

We wil now create a handler to **delete** only **one user**.

So let's go back to the **user controller**. in the **/controllers/user.js** let's do the following code to delete one user from the database

```
const deleteOne = async (req, res) => {
  const id = req.params.id

  try {
    const user = await User.findOneAndRemove({ _id: id });

    if (!user) {
      return res.status(400).json({ message: 'user not found' });
    }
    return res.status(201).json({ message: 'deleted successfully', data: user });
  } catch (error) {
    console.log(error);
    return res.status(500).end();
  }
};
```

Once you add that handler, let's export it, so that our router can see it. like this below

```
module.exports = {
  createUser,
  getAllUsers,
  getOneUser,
  updateOne,
  deleteOne
};
```

Next, require it in the **/routers/route.js** file. and add the route

```
const { createUser, getAllUsers, getOneUser, updateOne, deleteOne } = require('../controllers/user');
```

and add it to the route

```
// User route
router.post('/user', createUser);
router.get('/user', getAllUsers);
router.get('/user/:id', getOneUser);
router.put('/user/:id', updateOne);
router.delete('/user/:id', deleteOne);
```

Note how we add the parameter to the path `'/user/:id'` and note the verb here is **delete** instead of **get** or **put**

Let's test and see if we will update one user using out rest client

Inside **requests** create a file called **delete-one.rest**

In that file, let's send a get request to update one user, see the code below for that.

```
DELETE http://localhost:3001/api/user/634a5bdb441c0826a1b5e678
```

If all goes well, you should see a similar message like the one below

```
{
  "message": "deleted successfully",
  "data": {
    "email": "ac@camariana.gm",
    "firstname": "A",
    "minit": "",
    "lastname": "Camara",
    "role": "Admin",
    "createdAt": "2022-10-15T07:06:03.201Z",
    "updatedAt": "2022-10-21T12:26:10.448Z",
    "id": "634a5bdb441c0826a1b5e678"
  }
}
```

Note: in the above object, the first property in the object says this

```
"message": "deleted successfully",
```

and finally the data that was deleted

```
"data": {
    "email": "ac@camariana.gm",
    "firstname": "A",
    "minit": "",
    "lastname": "Camara",
    "role": "Admin",
    "createdAt": "2022-10-15T07:06:03.201Z",
    "updatedAt": "2022-10-21T12:26:10.448Z",
    "id": "634a5bdb441c0826a1b5e678"
  }
```

Okays folks, we have seen all the so called CRUD operations in as far as databases are concern. Their is more to it anyways, but we will see.

Step 13

With this step we want to establish a one-to-many relationship and also learn to do references across collections. With that in mind, let's look at the two entities user (User) and sales (Sale)

User -----> Sale

Let's assume that the *users* collection contains two users:

```
{
  "data": [
    {
      "email": "ac@camariana.gm",
      "firstname": "A",
      "minit": "",
      "lastname": "Camariana",
      "role": "User",
      "createdAt": "2022-11-05T10:37:43.369Z",
      "updatedAt": "2022-11-05T10:37:43.369Z",
      "id": "63663cf7061d3c204ad2747c"
    },
    {
      "email": "maria@camariana.gm",
      "firstname": "Maria",
      "minit": "",
      "lastname": "Qibtiyya",
      "role": "User",
      "createdAt": "2022-11-09T09:25:39.327Z",
      "updatedAt": "2022-11-09T09:25:39.327Z",
      "id": "636b72139daaabbcecf2143f"
    }
  ]
```

```
}
```

The *sales* collection contains three sales that all have a *user* field that references a user in the *users* collection:

```

```

Document databases do not demand the foreign key to be stored in the sale resources, it could *also* be stored in the users collection, or even both:

```

```

Since users can have many sales, the related ids are stored in an array in the *sales* field.

The Schema for sale

This could collection could be any of your collections, we are just learning how to do one-to-many relationship and also learn to do references across collections.

Now inside the **/models** folder create a new file caled **sale.js** and do the following code in it

Here's the code for our sale.js model (this might be a different model for you).

Note: Look at your model and adhere to the name and the attributes (fields) you have in your model and use those.

```
const mongoose = require('mongoose')

const saleSchema = new mongoose.Schema(
  {
    description: {
      type: String,
      required: true,
    },
    qty: {
      type: Number,
      required: true,
    },
    price: {
      type: Number,
      required: true,
    },
    Total: {
      type: Number,
      required: true,
    },
    user: {
```

```
      type: mongoose.Schema.Types.ObjectId,
      ref: 'user'
    }
  },
  { timestamps: true }
)


saleSchema.set('toJSON', {
  transform: (document, returnedObject) => {
    returnedObject.id = returnedObject._id.toString()
    delete returnedObject._id
    delete returnedObject.__v
  }
})


const Sale  = mongoose.model('sale', saleSchema)

module.exports = Sale
```

The sale references the user who created it, by this line

```
user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'user'
}
```

Let's go back and expand our user model to reference all of the sales created by a user.

Here's the expanded code of the user

```
const mongoose = require('mongoose')

const userSchema = new mongoose.Schema(
  {
    email: {
      type: String,
      required: true,
      unique: true,
      trim: true
    },
    password: {
      type: String,
      required: true
    },
```

```
    firstname: {
      type: String,
      trim: true,
      maxlength: 25
    },
    minit: {
      type: String,
      trim: true,
      maxlength: 25
    },
    lastname: {
      type: String,
      trim: true,
      maxlength: 25
    },
    role: {
      type: String,
      enum: ['Admin', 'User'],
      default: 'User'
    },
    sales: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'sale'
      }
    ]
  },
  { timestamps: true }
)


userSchema.set('toJSON', {
  transform: (document, returnedObject) => {
    returnedObject.id = returnedObject._id.toString()
    delete returnedObject._id
    delete returnedObject.__v
    // the password should not be revealed
    delete returnedObject.password
  }
})


const User  = mongoose.model('user', userSchema)

module.exports = User
```

The user has an array of references to all of the sales created by a user.

The ids of the sales are stored within the user document as an array of Mongo ids. The definition is as follows:

```
sales: [
    {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'sale'
    }
]
```

Now's let's test by creating new user and see what happens. To do that go back to your **create-one.rest** and a different user by editing the object in there.

Here's my new user

```
POST http://localhost:3001/api/user HTTP/1.1
content-type: application/json

{
  "email": "anna@datalu.gm",
  "password": "dibzy",
  "firstname": "Anna",
  "minit": "",
  "lastname": "Dibba"
}
```

If that happen successfully, you should now see a response like this

```
{
  "data": {
    "email": "anna@datalu.gm",
    "firstname": "Anna",
    "minit": "",
    "lastname": "Dibba",
    "role": "User",
    "sales": [],
    "createdAt": "2022-11-09T09:46:18.507Z",
    "updatedAt": "2022-11-09T09:46:18.507Z",
    "id": "636b76ea9daaabbcecf21442"
  }
}
```

Note: on line 8, we now have the sales array. It's empty at the moment because this user hasn't made any sales yet.

```
"sales": [],
```

Step 14

Now lets create a new sale (Again, for you this might be a different entity)

Inside the **/controllers** folder, create a file called **sale.js**

We will first require the following models into the sale controller:

1. The Sale model
2. The User model

```
const Sale = require('../models/sale');
const User = require('../models/user');


const createSale = async (req, res) => {
  const content = req.body;
  const user = await User.findById(content.userId);

  try {
    const sale = await Sale.create({ user: content.userId,  ...content
})

    user.sales = user.sales.concat(sale._id)
    await user.save();

    return res.status(201).json({ data: sale });
  } catch (error) {
    console.log(error);
    return res.status(500).end();
  }
};



module.exports = {
  createSale,
};
```

A lot is happening here, let's explain each one

First, we find the user who is creating the sale

```
const user = await User.findById(content.userId);
```

Note: we are currently manually putting this userId in the request for now. Under normal cercumstances this will be the logged in user.

Next, we create a sale by putting the userId to the user filed and the spread the rest of the content

```
const sale = await Sale.create({ user: content.userId,  ...content })
```

Next, we add the id of the current sale to the sales array in the user collection and then save it.

```
user.sales = user.sales.concat(sale._id)
await user.save();
```

finally reponse to the user with the current sales

```
return res.status(201).json({ data: sale });
```

Once you do that let's require it in the **/routers/route.js** file.

Remember we have also require the new sale controller first like this below

```
const { createSale,  } = require('../controllers/sale');
```

And, let's add the route to the sale route

```
router.get('/sale', createSale);
```

The code for the /route.js should now look like this:

```
const express = require('express');
const router = express.Router();

const { createUser, getAllUsers, getOneUser, updateOne, deleteOne } =
require('../controllers/user');
const { createSale,  } = require('../controllers/sale');


// User route
router.post('/user', createUser);
router.get('/user', getAllUsers);
router.get('/user/:id', getOneUser);
router.put('/user/:id', updateOne);
router.delete('/user/:id', deleteOne);

// Sale route
router.post('/sale', createSale);
```

```
17
18
19  module.exports = router;
```

Take note of the following:

1. Line 5 where we are requiring the sale controller
2. line 16, where we are adding the creation of a new sale route

Now let's test this endpoint and see if we can create a new sale.

To do that, go to the **create-one.rest** file inside the requests folder and add the following to create a new sale.

```
1  ###
2  POST http://localhost:3001/api/sale HTTP/1.1
3  content-type: application/json
4
5  {
6    "description": "Apple",
7    "qty": 2,
8    "price": 200,
9    "Total": 400,
10   "userId": "636b76ea9daaabbcecf21442"
11 }
```

Note on line 1, the three # tags, this help us create many requests in one file

And also note, I am hard coding the id of the user at line 10 by copying the id of the last user I created in step 13.

The code of **create-one.rest** should now look like this:

```
1  POST http://localhost:3001/api/user HTTP/1.1
2  content-type: application/json
3
4  {
5    "email": "anna@datalu.gm",
6    "password": "dibzy",
7    "firstname": "Anna",
8    "minit": "",
9    "lastname": "Dibba"
10 }
11
12
13 ###
14 POST http://localhost:3001/api/sale HTTP/1.1
15 content-type: application/json
16
17 {
```

```
  "description": "Apple",
  "qty": 2,
  "price": 200,
  "Total": 400,
  "userId": "636b76ea9daaabbcecf21442"
}
```

Okay now send the request to create a new sale.

If all goes well, you should see a similar response

```
{
  "data": {
    "description": "Apple",
    "qty": 2,
    "price": 200,
    "Total": 400,
    "user": "636b76ea9daaabbcecf21442",
    "createdAt": "2022-11-09T10:16:20.372Z",
    "updatedAt": "2022-11-09T10:16:20.372Z",
    "id": "636b7df4cd10931235c5976c"
  }
}
```

Note on line 7, where the user has the id of the user