Express-Node Back-end

Building a simple back-end application with Express, Node, and MongoDB

Need for back-end

- Back-end will connect to a DB, get some results, and do some processing with those results
- Back-end will save data to the DB
- Back-end will expose REST API that front-end will use to interact with the DB
- Back-end will accept HTTP requests from front-end app and use CRUD operations to interact with the DB

Using Express for back-end app

- Express is a minimal and flexible web application framework for Node.js

- Express provides a robust set of features for web and mobile applications

- Express provides a myriad of HTTP utility methods and middleware for creating a robust API quickly and easily

Setting up back-end app

- Install express-generator globally

```
> npm install express-generator -generator
```

- Change directory to where you which to create your back-end application

```
> express ContactsAppBackend
```

- This will generate all the boilerplate code and directory structure that you need to create an express application
-

Install missing modules

- install express locally and save it to your manifest

```
> npm install express --save
```

- Install all your dependency modules

```
> npm install
```

- start your express application

```
> npm start
```

Structure of app

- Survey **app.js** and **./bin/www**. Fix a few linter errors
  - **app.js** is the brain of the application
  - Port number for the application is assigned in **./bin/www**

- Explore **package.json** manifest
  - npm start script runs ./bin/www as node application

- Folder structure
  - **views** – back-end views, ideal for documenting api
  - **routes** – route controllers for processing http requests. This is where most of the work will be done
  - **public** – public artifacts (images, css, etc) will be stored there

Setup mongoDB

- Since the application is going to interact with a DB, we need to setup the DB
- In a terminal start the mongo daemon

```
> mongod
```

- MongoDB will stat on port 27017.
- In another terminal start the mongo shell and create our contactsappdb

```
> mongo
> use contactsappdb
```

Setup app to connect to mongoDB

- We need to setup Mongoose to use mongoDB.
  - Mongoose provides mongodb object modeling for node.js
  - Mongoose provides a straight-forward, schema-based solution to model your application data.
  - It includes built-in type casting, validation, query building, business logic hooks and more.

- Setting up Mongoose:
  - create a folder called **model** in ContactsAppBackend
  - create a new file called **ContactsAppBackend/model/db.js**
  - place our db connection code in db.js

- Create a variable in app.js that points to our db.js file

Wiring up more pieces

- We need to install the modules that are missing

```
> npm install mongoose --save
> npm install method-override --save
```

- Our app should still work when we run **npm start**

- Now we need to create our model and schema
  - This will be done in a new file in **ContactsAppBackend/model/**
  - Let's call it **contact.js.**
  - Each contact is going to have a firstName, lastName, email, homePhone, cellPhone, birthDay, website, address.
  - In **app.js** create a variable at the top, below our db variable we added earlier, that points to **contact.js**

Onto route controllers

- Now we need to create the route controllers for our REST API
- Add these lines in app.js to reference and use the route controllers

```
var routes = require('./routes/index');
var contacts = require('./routes/contacts');

          // AND

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

Let the fun begin!

- We're going to build our entire controller with all the CRUD and REST pieces completely baked in
- We are going to take this **piece by piece**, but all of this will go into the **routes/contacts.js** file

Getting all contacts

- We are going to build the **GET API** for grabbing all the Contacts from the database
- We can choose to display all the contacts in a backend view or send a JSON response to the client
- We can also build the **POST API** for creating a new Contact
- We will write and test code to do these in **routes/contacts.js**

Inserting data in DB

- In the **mongo terminal** create a contacts collection and insert a contact in it

- Enter the following command in the terminal

```
> db.contacts.insert(
... {
... "firstName": ”John",
... "lastName": ”Doe",
... "email": "john.doe@email.com"
... });
```

Getting contact by id

- After testing getting all contacts API, we are now ready to get contacts by id
- First, we need route middleware to validate id

```
router.param('id', function(req, res, next, id) {
        …
});
```

- After validating the id, we need to GET an individual contact to display or return to the client

Setup to get contact by id

```
router.route('/:id')
        .get(function(req, res) {
                ...
});
```

Other CRUD operations

- We will use **PUT** to **update** a contact.
  - PUT and POST do similar things
  - The difference here is that PUT first finds the contact and edit instead of creating a new contact

- **DELETE** is a crucial operation that can also be supported by using the REST API

- We would add views here if we did not have a front-end application

- We need to update our front-end application to consume the newly created REST API

Resources

- http://expressjs.com/
- https://www.airpair.com/javascript/complete-expressjs-nodejs-mongodb-crud-skeleton
- http://mongoosejs.com/index.html
- https://docs.mongodb.org/getting-started/shell/insert/