**Bonus Exercise**

*Creating a RESTful data service in Express.js, backed by MongoDB, with Docker Compose*

**Prior Knowledge**

Basic understanding HTTP verbs, REST architecture

Some docker-compose knowledge

Basic JavaScript coding

**Objectives**

Understand Express.js better

Understand MongoDB a bit

See another Docker Compose app in action

**Software Requirements**

(see separate document for installation of these)

* Docker
* Docker Compose
* Google Chrome/Chromium plus Chrome Advanced REST extension

**Overview**

*Express.js is a framework for building RESTful apps in node.js*

*Mongoose is a client for MongoDB that gives a high-level of abstraction for databases*

*We are going to use these together to create a nice RESTful data service.*

Steps:

1. Firstly, make a new directory for your code, and copy the initial code into the directory  
     
   mkdir ~/bonus  
   cd ~/bonus  
   cp -r ~/repos/ox-soa2/code/bonus .
2. Take a look at the code that is there.  
   Firstly, start with the **docker-compose.yml** and you will see we have two services – a node app and a mongo database.
3. The **Dockerfile-mongo** basically lets us take the existing Mongo and hack it to import some basic data. Mongo stores its data in a JSON-like manner and the format of imports and exports is JSON. You can see the data in the data/ directory.
4. The data is just the JSON we want to use with an internal id and an internal version number, e.g.:

{"\_id":{"$oid":"575461725531e805005a97be"},"name":"BigCo","id":"00002","email":"accounts@big.co","\_\_v":0}

{"\_id":{"$oid":"575461725531e805005a97bf"},"name":"Acme","id":"00001","email":"accounts@acme.com","\_\_v":0}

1. The **Dockerfile-node** is really straightforward. It installs the npm requirements:  
     
   express.js – the REST framework  
   uuid – I thought this might be useful, but didn’t use it!  
   mongoose – MongoDB library  
   body-parser – makes it easier to consume JSON messages  
     
   Then it copies the src/\*.js and runs server.js
2. To complete this exercise, there is no need to change the docker setup, only the src JavaScript.
3. **Customer.js**  
     
   This is the code that initializes a Customer object backed by Mongo/mongoose. We could have put this into the server.js, but this approach would allow you to re-use this data definition in several .js files.  
     
   You can see the nice way of defining schemas in Mongoose.  
   For more information on mongoose, see <http://mongoosejs.com/docs/index.html>

var mongoose = module.parent.exports.mongoose;

var customerSchema = mongoose.Schema({

name: String,

id: {type: String, unique: true},

email: String

});

var customer = mongoose.model('Customer', customerSchema);

module.exports = customer;

1. Here are some interesting parts of **server**.**js**
2. Firstly, notice how we refer to the mongo container as mongo-data which is defined in docker-compose.yml. The “link” means that the DNS will resolve properly inside the container.  
     
   mongoose.connect('mongodb://**mongo-data**/test');
3. We need to import the Customer object that was defined separately.  
     
   var Customer = require("./Customer.js");
4. Express has the concept of a Router:  
   var router = express.Router();  
   This is very useful as it allows us to define multiple targets for different URLs to route to different verbs.  
   You can see how it is used in the router.route(..) syntax below.  
   We finally bind it at the bottom using:  
   app.use('/',router);  
     
   You can think of this as comparable to how JAX-RS uses @Path assertions.
5. We can search all customers using this syntax:  
   Customer.find({}, function(err, data) {
6. Notice that this returns JS data which we can directly serialise into JSON with:  
    res.json(data);  
   The only problem is that this includes some Mongo stuff in it (e.g. \_id and \_\_v) so we have to remove that before we send it back to the client.  
     
   However, overall this is why node+mongo is so popular in the world of HTTP+JSON – we have almost no coding to do to serialise/deserialise and the rest of the code is almost declarative.
7. We can also use very simple fluent code to respond:  
   e.g.  
   res.set("Location","/customers/"+c.id);  
   res.status(201).json(data);
8. You can play with this service. Start it with:  
   docker-compose up --build  
   Use ARC to do GET and POST operations  
   The port is 8000
9. Your challenge is to add a new URL handler to handle our catalogue.  
     
   There is already some catalogue data in the database, in a collection called entries. You need to create:
   1. an Entry.js that captures this data in a schema
   2. router entries in server.js that add support for a new URL /cat
   3. Here is the JSON we want to support:

{

"itemname": "Widget"

"id": "1"

"cost": 5

}

1. If you want to see my version in action, you can:

mkdir ~/new && cd ~/new  
git clone <https://github.com/pzfreo/simple-rest-node-mongo.git>   
docker-compose up –build

That’s all folks