Hands on Exercises

Information about the Hands-On Sessions

The spirit of this document lends the individual quite a lot of room for creativity. To that end, a very serious attempt has been made to avoid creating questions that simply lead the reader through the construction process. With each question, we hope that the reader is encouraged to think about the solution and the process through which that solution is determined. We hope you agree that this approach makes for a much richer learning experience.

There are often many correct solutions to these problems. We encourage you to interact with the instructor as much as possible during the hands-on sessions.

1) Node.JS Quick Start

1.1. Executing Node.js Scripts

Create a new folder named lab1.1 into your workspace. In this folder create a file named helloworld.js and edit file content as shown below:

helloworld.js

```
console.log("hello world");
```

Open a command prompt window and switch to test folder. Type

```
>node helloworld.js
```

You will see "hello world message" on the screen

1.2. Node Package Manager

1) Open https://www.npmjs.com/ site and search/find some packages. Example: express, mongoose, connect, …
2) Open a command prompt window and switch to lab1.1 folder. Type
   ```
   >npm help
   ```
3) You will see command list. If you want to get help about specific term, type "npm help <term>". To try this type following commands:
   ```
   >npm help install
   >npm help package.json
   ```
4) Let install a package. Type
   ```
   >npm install express
   ```
5) Examine lab1.1 folder. You will see newly added "node_modules" folder. Examine inside of that folder.
6) Type "npm ls" to list installed packages
7) To remove a package uninstall command can be used. Type
   ```
   >npm uninstall express
   ```
   After that check node_modules folder again. What is the difference?
8) To globally install packages, use –g option. Type following command:
   ```
   >npm install -g express
   ```
   Go to global file location and test existence of the express module

   Sample url: "C:\Users\Esma\AppData\Roaming\npm\node_modules"
9) Type "npm ls -g" to list globally installed packages

10) Uninstall globally installed express module with:

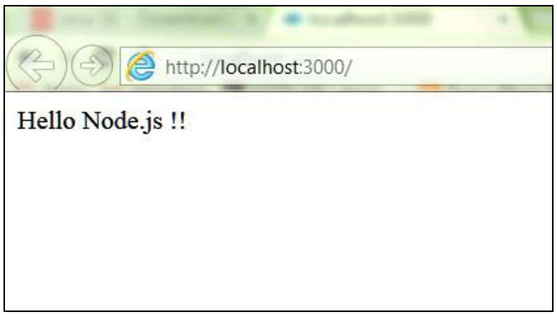
```
>npm uninstall -g express
```

1.3. Create a simple http server

1) In the lab1.1 folder create a new js file named sample.js
2) Edit file content as shown below:

```
var http = require("http");
var server = http.createServer(function(request, response) {
response.write("Hello Node.js !!");
response.end();
});
server.listen(3000);
console.log('Server started');
```

3) Type "node sample.js" to run file.
4) Open a browser and type "localhost:3000". You should see the welcome message on the screen



Hello Node.js !!

2) Node.JS Middleware

- In your workspace create a new folder named lab2
- Open a command prompt and switch to lab2 folder
- Type "npm init" to initialize project. You can use default answers for questions
- After that you will see that the file **package.json** has been created. Let create a new file named **index.js**
- Install connect module

```
>npm install connect -- save
```

- Let build our own http middleware. A middleware component is not much more than a function that receives the request and response objects. Our first middleware component responds with the string "Hello World!"and ends the response. For that purpose create a new file named **hello_world.js** which has following content:

```
function helloWorld(req, res) {
res.end('Hello World!!');
}
module.exports = helloWorld;
```

- Edit index.js : import hello_world and connect modules, add helloWorld middleware and create a server

```
var connect = require('connect');
// import middlewares
var helloWorld = require('./hello_world');
var app= connect();
app.use(helloWorld);


app.listen(3000);
```

switch to command prompt and run index.js with typing "node index"

Open a web browser and type " http://localhost:3000/". Check your result. You will see the "'Hello World!!" message on the screen

3) Creating Express Application with express-generator

1) Create a folder named lab3 . Open a command prompt and switch to lab3 folder
2) We will create a sample express application with express-generator. To do this we should install express-generator module globally.

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

To create a skeleton application use express command:

```
> express shopping-services-app -e
```

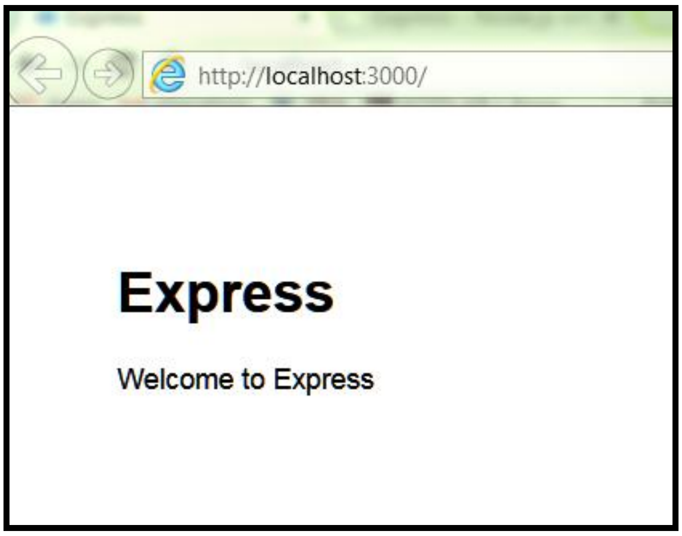
Thus an express application with ejs templation engine support will be created. Let continue to install dependencies:

```
> cd shopping-services-app
> npm install
```

Examine files and folders inside project. Especially app.js, package.json files and node-modules, public, bin folders

To start sample application type ">npm start"

After opening a web browser and typing "localhost:3000" then you will see the welcome page



Express

Welcome to Express

4) MongoDB

4.1. Run MongoDB & Mongo Shell

1) Open a command prompt window and change directory to \mongodb\bin folder. Run mongod.exe.

```
D:\>cd \mongodb\bin
D:\mongodb\bin>mongod
```

If your data directory is different from \data\db then use "--dbpath" parameter to specify mongo db data folder

```
d:\db\MongoDB\bin\mongod --dbpath d:\db\MongoDB-data
```

Your mongodb server should be started on port 27017.

2) Open another command prompt window and change directory to \mongodb\bin and run mongo.exe

4.2. Import Example DataSet

1) If dataset.json file is not provided then open following link and save content in to a file named dataset.json

https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/dataset.json

2) copy file into the mongodb\bin directory.
3) open a new command prompt window and change directory to "mongodb\bin"
4) type following command to insert sample dataset

```
mongodb\bin> mongoimport --db test --collection restaurants --drop --file dataset.json
```

this command will create a database named test and import data into restaurants collection.

4.3. MongoDB Shell Help Commands

1) Switch to mongo shell window
2) Type "help" and press enter button. You should see all commands and their description.
3) Type "show dbs" and press enter
4) To change current database to "test" type "use test"

5) To display collections in current database type "show collections". You should see "restaurants" as collection name
6) Type "db.collections.help()" and inspect result. You should see all commands related to collections

4.4. Using Mongo Shell

1. Insert a document

Insert a document into a collection named `restaurants`

```
db.restaurants.insert(
  {
    "address" : {
      "street" : "2 Avenue",
      "zipcode" : "10075",
      "building" : "1480",
      "coord" : [ -73.9557413, 40.7720266 ],
    },
    "borough" : "Manhattan",
    "cuisine" : "Italian",
    "grades" : [
      {
        "date" : ISODate("2014-10-01T00:00:00Z"),
        "grade" : "A",
        "score" : 11
      },
      {
        "date" : ISODate("2014-01-16T00:00:00Z"),
        "grade" : "B",
        "score" : 17
      }
    ],
    "name" : "Vella",
    "restaurant_id" : "41704620"
  }
)
```

You should see number of rows inserted as result:

```
WriteResult({ "nInserted" : 1 })
```

2. Find or query data

1) Query for All Documents in a Collection:

   To return all documents in a collection, call the find() method without a criteria document.

```
db.restaurants.find()
```

2) Query by a Top Level Field

```
db.restaurants.find( { "borough": "Manhattan" } )
```

3) Query by a Field in an Embedded Document

```
db.restaurants.find( { "address.zipcode": "10075" } )
```

4) Query by a Field in an Array

```
db.restaurants.find( { "grades.grade": "B" } )
```

5) Specify Conditions with Operators

```
db.restaurants.find( { "grades.score": { $gt: 30 } } )
db.restaurants.find( { "grades.score": { $lt: 10 } }
```

6) Combine Conditions¶

Logical AND

You can specify a logical conjunction (AND) for a list of query conditions by separating the conditions with a comma in the conditions document.

```
db.restaurants.find( { "cuisine": "Italian", "address.zipcode": "10075" } )
```

Logical OR

You can specify a logical disjunction (OR) for a list of query conditions by using the $or query operator.

```
db.restaurants.find(
  { $or: [ { "cuisine": "Italian" }, { "address.zipcode": "10075" } ] }
)
```

7) Sort Query Results

```
db.restaurants.find().sort( { "borough": 1, "address.zipcode": 1 } )
```

3. Update data

1) Update Specific Fields

```
db.restaurants.update(
   { "name" : "Juni" },
   {
     $set: { "cuisine": "American (New)" },
     $currentDate: { "lastModified": true }
   }
)
```

2) Update an Embedded Field

```
db.restaurants.update(
  { "restaurant_id" : "41156888" },
  { $set: { "address.street": "istanbul" } }
)
```

3) Update Multiple Documents

```
db.restaurants.update(
  { "address.zipcode": "10016", cuisine: "Other" },
  {
    $set: { cuisine: "Category To Be Determined" },
    $currentDate: { "lastModified": true }
  },
  { multi: true}
)
```

4. Replace a Document

To replace the entire document except for the _id field, pass an entirely new document as the second argument to the update() method.

```
db.restaurants.update(
   { "restaurant_id" : "41704620" },
   {
     "name" : "Vella 2",
     "address" : {
           "coord" : [ -73.9557413, 40.7720266 ],
           "building" : "1480",
           "street" : "2 Avenue",
           "zipcode" : "10075"
     }
   }
)
```

5. Remove Data

1) Remove All Documents That Match a Condition

```
db.restaurants.remove({"restaurant_id" : "41156888"} )
```

2) Use the justOne Option

By default, the remove() method removes all documents that match the remove condition. Use the justOne option to limit the remove operation to only one of the matching documents.

```
db.restaurants.remove( { "borough": "Queens" }, { justOne: true }
```

4.5. Data Aggregation

1) Group Documents by a Field and Calculate Count

The following example groups the documents in the restaurants collection by the borough field and uses the $sum accumulator to count the documents for each group.

```
db.restaurants.aggregate(
  [
    { $group: { "_id": "$borough", "count": { $sum: 1 } } }
  ]
);
```

2) Filter and Group Documents

```
db.restaurants.aggregate(
  [
    { $match: { "borough": "Queens", "cuisine": "Brazilian" } },
    { $group: { "_id": "$address.zipcode" , "count": { $sum: 1 } } }
  ]
);


db.restaurants.aggregate(
[
 { $group: { _id: "$cuisine", total: { $sum: 1 } } },
 { $match: { total: { $gte: 700 } } }
 ] )
```

4.6. Indexes

1. Create a Single-Field Index

Run following query and check query execution plan. You should see that there is no index usage.

```
db.restaurants.find({cuisine:"Italian"}).explain()
```

Create an ascending index on the "cuisine" field of the restaurants collection.

```
db.restaurants.createIndex( { "cuisine": 1 } )
```

Run query again. This time you should see that database uses index

```
"winningPlan" : {
        "stage" : "FETCH",
        "inputStage" : {
                "stage" : "IXSCAN",
                "keyPattern" : {
                        "cuisine" : 1
                },
                "indexName" : "cuisine_1",
                "isMultiKey" : false,
                "direction" : "forward",
                "indexBounds" : {
                        "cuisine" : [
                                "[\"Italian\", \"Italian\"]"
                        ]
                }
        }
},
```

2. Create a compound index

Create a compound index on the "cuisine" field and the "address.zipcode" field. The index orders its entries first by ascending "cuisine" values, and then, within each "cuisine", by descending"address.zipcode" values.

```
db.restaurants.createIndex( { "cuisine": 1, "address.zipcode": -1 } )
```

Run following command and check index usage:

```
db.restaurants.find({cuisine:"Italian","address.zipcode":"11209"}).explain()
```

3. Create multiple index and test usage of index

```
db.restaurants.createIndex( { "grades.score": 1 } )
db.restaurants.find({"cuisine": "Italian","grades.score" : {$gt:10} }).explain()
```

4. Geospatial Indexes (Optional)

1) Unzip dump.zip file into the mongodb\bin folder.
2) Open a new command prompt window and switch to \mongodb\bin folder. Type

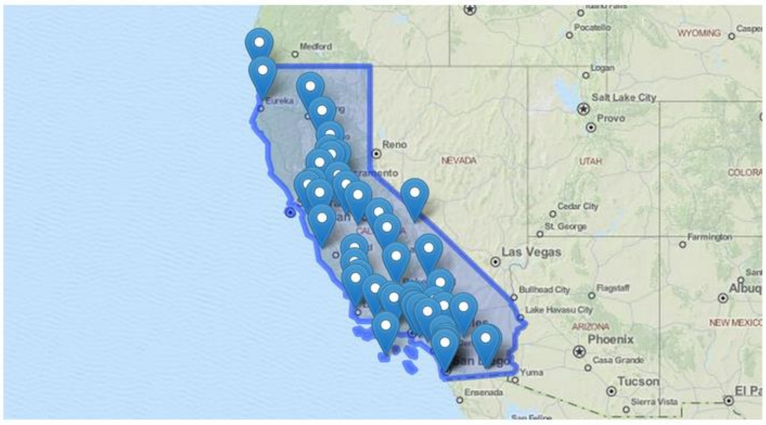
   ```
   >mongorestore dump
   ```

   If import is completed successfully delete dump.zip and dump folder from bin folder.

   Switch to mongodb shell window and type "show dbs". You should see newly added "geo" database. Type "use geo" to switch database
3) Type "show collections". List content of the collections: "db.states.find() db.airports.find()"
4) Find all the airports in California. For this you need to get the California location (Polygon) and use the command $geoWithin in the query. From the shell it will look like :

   ```
   var cal = db.states.findOne(  {code : "CA"}  );

   db.airports.find(
     {
       loc : { $geoWithin : { $geometry : cal.loc } }
     },
     { name : 1 , type : 1, code : 1, _id: 0 }
   );
   ```

   So the query is using the "California MultiPolygon" and looks in the airports collection to find all the airports that are in these polygons. This looks like the following image on a map:



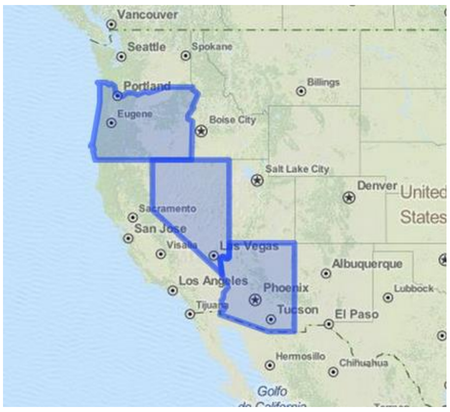
we are querying these documents with no index. You can run a query with the explain() to see what's going on. The $geoWithin operator does not need index but your queries will be more efficient with one so let's create the index:

```
db.airports.ensureIndex( { "loc" : "2dsphere" } );
```

Run the explain and you will se the difference.

5) Suppose that you want to know what are all the adjacent states to California, for this we just need to search for all the states that have coordinates that "intersects" with California. This is done with the following query:

```
var cal = db.states.findOne(  {code : "CA"}  );
db.states.find(
  {
    loc : { $geoIntersects : { $geometry : cal.loc  }  } ,
    code : { $ne : "CA"  }
  },
  { name : 1, code : 1 , _id : 0 }
);
```



Same as before $geoIntersect operator does not need an index to work, but it will be more efficient with the following index:

```
db.states.ensureIndex( { loc : "2dsphere" } );
```

6) Last feature is related to query with proximity criteria. Let's find all the international airports that are located at less than 20km from the reservoir in NYC Central Park. For this you will be using the $near operator. So this query returns 2 airports, the closest being La Guardia, since the $near operator sorts the results by distance. Also it is important to raise here that the $near operator requires an index.

```
db.airports.ensureIndex( { loc : "2dsphere" } );
db.airports.find(
        {
          loc : {
            $near : {
              $geometry : {
                type : "Point" ,
                coordinates : [-73.965355,40.782865]
              },
              $maxDistance : 20000
            }
          },
          type : "International"
        },
        {
          name : 1,
          code : 1,
          _id : 0
        }
      );
```

5) Building Rest Services

We will develop following rest services during this lab session:

- **Getting all products**
  - ✓ **URL:** http://localhost:3000/api/products
  - ✓ **Method:** GET
  - ✓ **Path Variable:** NA
  - ✓ **Request Body:** NA
  - ✓ **Response Body:** Product list
  - ✓ **Exceptional Case:** The server will return 500 status code when request contains a query parameter named "code".

    Example: http://localhost:3000/api/products?code=111
- **Getting a specific product**
  - ✓ **URL:** http://localhost:3000/api/products/{id}
  - ✓ **Method:** GET
  - ✓ **Path Variable:** id of product to be read
  - ✓ **Request Body:** NA
  - ✓ **Response Body:** Product
- **Deleting a product**
  - ✓ **URL:** http://localhost:3000/api/products/{id}
  - ✓ **Method:** DELETE
  - ✓ **Path Variable:** id of product to be deleted
  - ✓ **Request Body:** NA
  - ✓ **Response Body:** deleted product
- **Saving a product**
  1) **URL:** http://localhost:3000/api/products
  2) **Method:** POST
  3) **Path Variable:** NA
  4) **Request Body:** product data
  5) **Response Body:** saved product

5.1. Setting up

1) Copy lab4 as lab6
2) Delete routes folder. We don’t need this folder.

3) Start mongodb

   Example:
```
d:\dev\db\MongoDB\bin\mongod --dbpath d:\dev\db\MongoDB-data
```

4) Open a command prompt and switch to lab6/shopping-services-app folder.
5) Type “npm install mongoose –save” to install mongoose.

5.2. Inserting initial data set

1) Create a new file named init.js under root of the project. This file will be used for inserting initial data set into the database
2) Edit file content as shown below:

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

var db = mongoose.connection;
db.on('error', console.error);
mongoose.connect('mongodb://localhost/shoppingdb');

var productSchema = mongoose.Schema({
    id: String,
    img: String,
    alt: String,
    name: String,
    price: String,
    description: String

});

var Product = mongoose.model('Product', productSchema);

var product1 = new Product({
    id:"0",
    img:"tablet",
    alt:"img-tablet",
    name:"10-Inch Tablet",
    price:"269",
    description: "Android 4.3 Jelly Bean, 10.1-inch Full HD (1920 x 1200) Display"
    });

    var product2 = new Product({
        id:"1",
        img:"shoe",
        alt:"img-shoe",
        name:"Running Shoe",
        price:"48",
        description: "Synthetic and Mesh, Imported, Rubber sole, Flex Film welded 
upper, HydraMAX moisture-wicking collar lining"
        });

    var product3 = new Product({
        id:"12",
        img:"watch",
        alt:"img-watch",
        name:"Slim Bracelet Watch",
        price:"48",
        description: "A narrow gold-
tone bracelet supports the round case of this  watch, which features three rhinest
ones marking each hour and a sparkling halo on the bezel"
        });

product1.save(function(err, product1) {
      if (err) {return console.error(err);}
    });
product2.save(function(err, product2) {
      if (err) {return console.error(err);}
    });
product3.save(function(err, product3) {
      if (err) {return console.error(err);}
    });

console.log("insert completed");
```

3) Switch to command prompt and run init.js

```
>node init
```

After completing insert operation, press Ctrl+C to stop process

5.3. Configure Mongoose

1) In the project create a new folder named app_api. This folder will contain all service codes.
2) Create "controllers", "models" and "routes" folders inside the app_api folder.

3) Lets begin by creating Mongoose model file in the app_api/models folder. Create a new file named **productModel.js** that contains following code snippet:

```
var mongoose = require('mongoose');
var productSchema = mongoose.Schema({
        id: String,
        img: String,
        alt: String,
        name: String,
        price: String,
        description: String
    });

mongoose.model('Product', productSchema);
```

4) Create a new file named **db.js** in the models folder that contains following code snippet:

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

var db = mongoose.connect("mongodb://localhost/shoppingdb");
var db = mongoose.connection;
db.on('error', console.error);
require('./productModel');
```

5.4. Creating Rest Services

1) Now we should define controller methods. Create a new file named productController.js in the /app_api/controllers folder. Add following code snippet:

```
var mongoose = require('mongoose');
var Product = mongoose.model('Product');

var sendJSONresponse = function(res, status, content) {
  res.status(status);
  res.json(content);
};
```

2) Add the following code snippet for getting all products:

```
module.exports.getProducts = function (req, res) {
  var code = req.query.code;
  if(code){
    sendJSONresponse(res, 500, "Internal Server Error");
  }else{
    Product.
        find({}).
        sort({name:"asc"}).
        exec(function(err, data) {
            if (err) {
              return console.error(err);
            };
            sendJSONresponse(res, 200, data);
          });
        }
  };
```

3) Add the following code snippet for getting a specific product:

```
module.exports.getProductById = function(req, res) {
  var id= req.params.id;
  Product.
    find({id: id}).
    sort({name:"asc"}).
    exec(function(err, data) {
        if (err) {
          return console.error(err);
        };
        sendJSONresponse(res, 200, data);
      });
};
```

4) Add the following code snippet for saving new product:

```
module.exports.addProduct = function(req, res, next) {
  var newId=Math.floor(Math.random() * 91 + 10);
  var newProduct= new Product(
  {   "id":newId,
      "img":req.body.img,
      "alt":req.body.alt,
      "name":req.body.name,
      "price":req.body.price,
      "description":req.body.description });

  newProduct.save(function(err){
    if (err) {
            return console.error(err);
    };
    sendJSONresponse(res, 200, newProduct);
  });
};
```

5) Add the following code snippet for deleting a product:

```
module.exports.deleteProductById = function(req, res) {
  var id= req.params.id;

  Product.
        find({id: id}).
        exec(function(err, data) {
            if (err) {
              return console.error(err);
            };
            Product.deleteOne({id: id}).
            exec(function(err, deleteResult) {
                if (err) {
                  return console.error(err);
                };
                sendJSONresponse(res, 200, data);
              });
          });
};
```

6) Now, we can continue with defining routes. Create a new file named index.js in the /app_api/routes folder. Edit file content as shown below:

```
var express = require('express');
var router = express.Router();
var productController = require('../controllers/productController');

router.get('/products', productController.getProducts);
router.get('/products/:id', productController.getProductById);
router.post('/products', productController.addProduct);
router.delete('/products/:id', productController.deleteProductById);
module.exports = router;
```

7) Open app.js and add routes configuration:

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

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

Delete following lines:

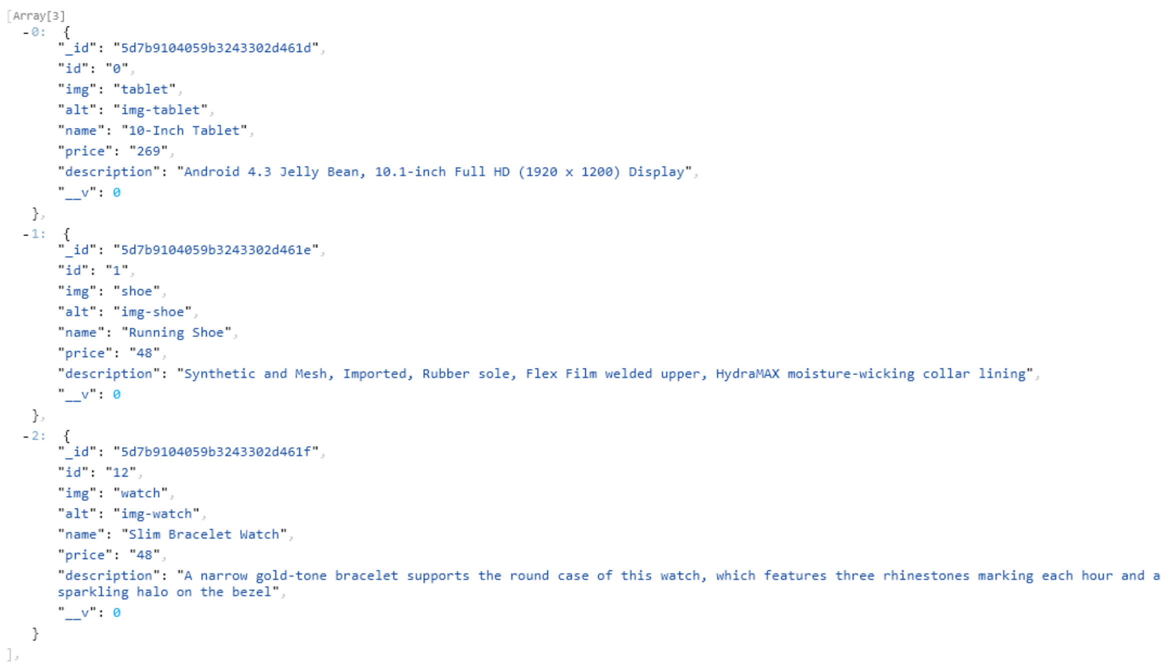
```
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

...
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

8) Add the top of the app.js add following dependency

```
    require('./app_api/models/db');
```

9) Start your application

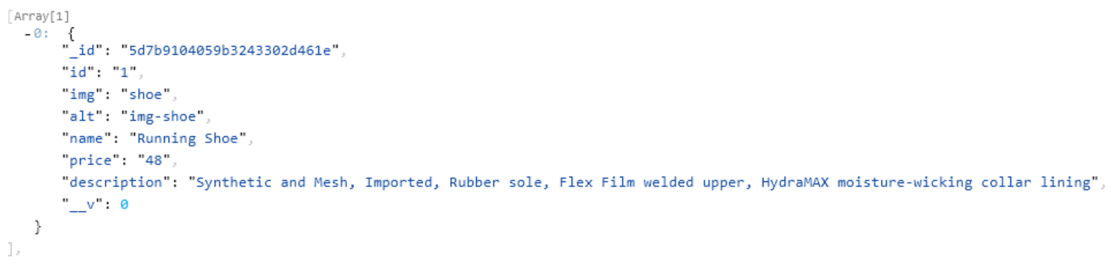
10) Open a rest client application (postman etc.) and type following url: http://localhost:3000/api/products/ change http method as GET. You should get all products as response



```
[Array[3]
  -0:  {
        "_id": "5d7b9104059b3243302d461d",
        "id": "0",
        "img": "tablet",
        "alt": "img-tablet",
        "name": "10-Inch Tablet",
        "price": "269",
        "description": "Android 4.3 Jelly Bean, 10.1-inch Full HD (1920 x 1200) Display",
        "__v": 0
    },
  -1:  {
        "_id": "5d7b9104059b3243302d461e",
        "id": "1",
        "img": "shoe",
        "alt": "img-shoe",
        "name": "Running Shoe",
        "price": "48",
        "description": "Synthetic and Mesh, Imported, Rubber sole, Flex Film welded upper, HydraMAX moisture-wicking collar lining",
        "__v": 0
    },
  -2:  {
        "_id": "5d7b9104059b3243302d461f",
        "id": "12",
        "img": "watch",
        "alt": "img-watch",
        "name": "Slim Bracelet Watch",
        "price": "48",
        "description": "A narrow gold-tone bracelet supports the round case of this watch, which features three rhinestones marking each hour and a
        sparkling halo on the bezel",
        "__v": 0
    }
],
```

11) Make GET request to the URL: http://localhost:3000/api/products/1

You should only see the product with ID value of 1


```
[Array[1]
  -0:  {
       "_id": "5d7b9104059b3243302d461e",
       "id": "1",
       "img": "shoe",
       "alt": "img-shoe",
       "name": "Running Shoe",
       "price": "48",
       "description": "Synthetic and Mesh, Imported, Rubber sole, Flex Film welded upper, HydraMAX moisture-wicking collar lining",
       "__v": 0
    }
 ],
```

12) Make DELETE request to the URL: http://localhost:3000/api/products/1

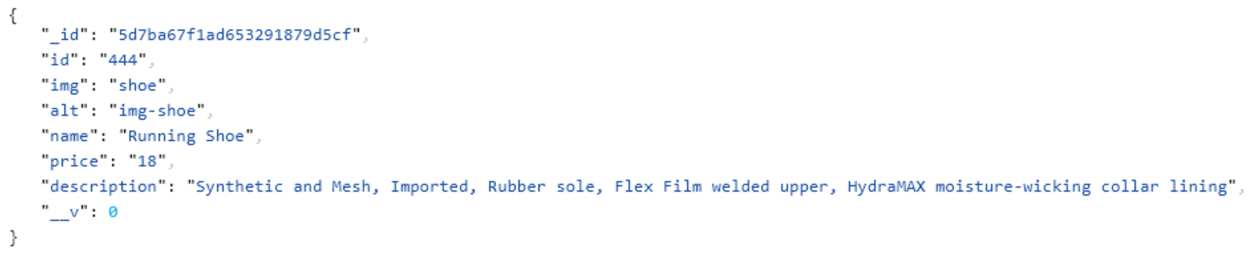
    You should get deleted product data

13) Make POST request to the URL: http://localhost:3000/api/products

    You can use following json data as request body

```
{
"img": "shoe",
"alt": "img-shoe",
"name": "Running Shoe",
"price": "18",
"description": "Synthetic and Mesh, Imported, Rubber sole, Flex Film welded upper, HydraMAX moisture-wicking collar lining"
}
```

Then, you should get a result similar to:


```
{
    "_id": "5d7ba67f1ad653291879d5cf",
    "id": "444",
    "img": "shoe",
    "alt": "img-shoe",
    "name": "Running Shoe",
    "price": "18",
    "description": "Synthetic and Mesh, Imported, Rubber sole, Flex Film welded upper, HydraMAX moisture-wicking collar lining",
    "__v": 0
}
```

5.5. How to enable CORS?

A request for a resource (like an image or a font) outside of the origin is known as a cross-origin request. CORS (cross-origin resource sharing) manages cross-origin requests. Cross-origin resource sharing (CORS) allows AJAX requests to skip the Same-origin policy and access resources from remote hosts. CORS exists for security reasons and to limit which resources a browser can gain access to, from another website. Let's say our site exists at http://someexampledomain.com and we want the JavaScript files on that site to access http://anotherdomain.com, we can't do that unless the server at http://anotherdomain.com allows it.

If we try an AJAX/xmlhttprequest to a file like that, we get an error message from the console in our browser.

```
XMLHttpRequest cannot load http://localhost:3000. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'null' is therefore not allowed access.
```

To allow exactly that to happend and have our client side code separate from our server and just make it load data, like with frameworks like Angular, Ember, Backbone or the like, we can use the following middleware function in express before we define our routes(app.js):

```
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
  next();
});
```