LAB 1 REPORT

Enterprise App Development

- Created a simple HTTP endpoint in NodeJS
- Interfaced between Node and Postgres using Massive JS
- Executed simple Postgres queries using SQL and exposed those using an HTTP API
- Demonstrated how SQL injection can be performed on a badly implemented RDMBS backend interface
- Implement SQL-injection proofing in your implementation
- Implemented an API model layer using the Sequelize object relational mapper
- Implemented API in Express using an ORM-based model layer

Eric Strong
C15708709@mydit.ie
DT211C/4

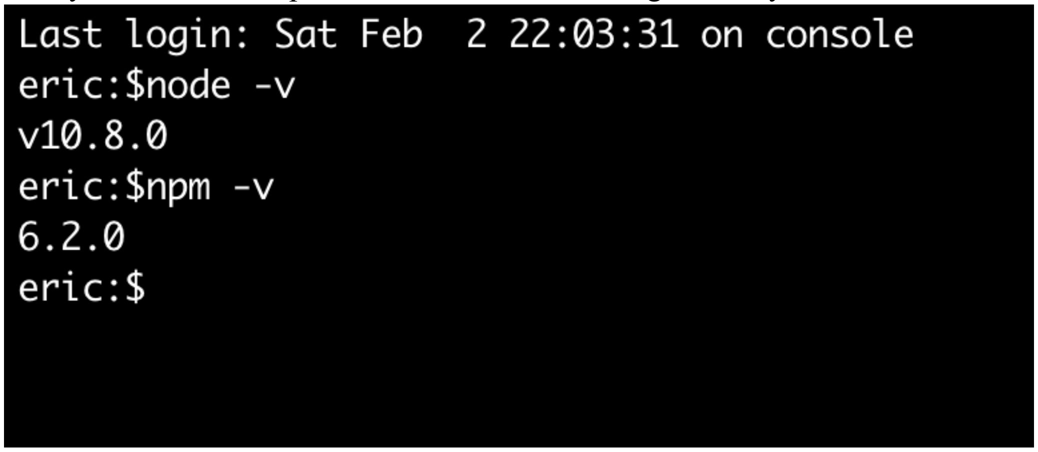
Contents

Video Demo of Lab

I made a video demonstration of the entire lab. It can be viewed here:
https://drive.google.com/open?id=1gCmAdQxZwS3z_EXZrZ3-hX2TcAGdL4an
All of the work below is included in a walk-through demonstration.

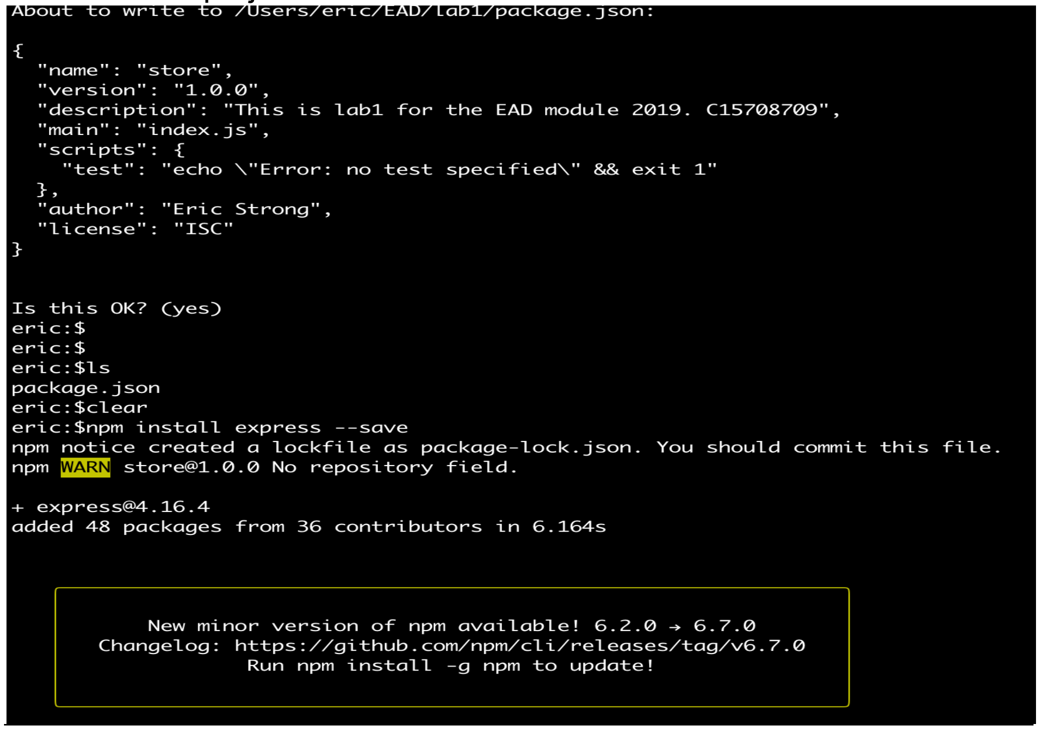
Setting Up

Install Node JS (*) on your laptop or sign up for a free cloud-based Node provider.
Verify that node and npm are installed and working correctly



```
Last login: Sat Feb  2 22:03:31 on console
eric:$node -v
v10.8.0
eric:$npm -v
6.2.0
eric:$
```

Create a new project folder

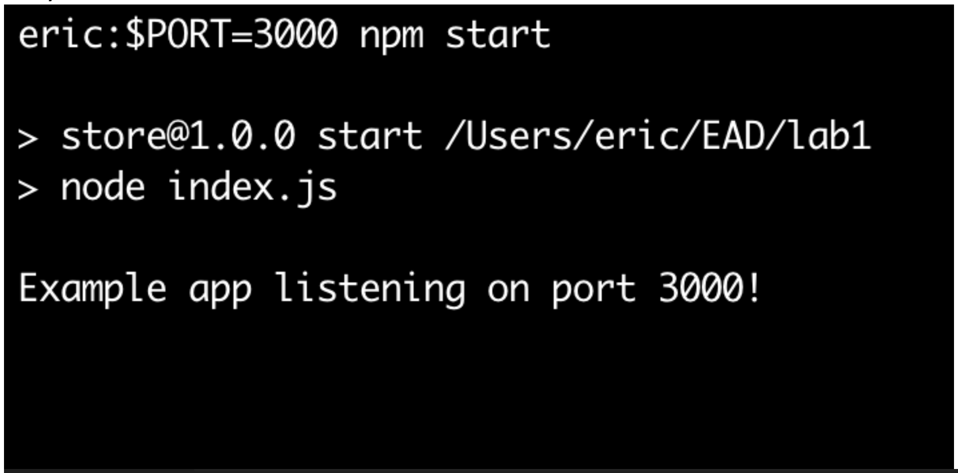


```
About to write to /Users/eric/EAD/lab1/package.json:

{
  "name": "store",
  "version": "1.0.0",
  "description": "This is lab1 for the EAD module 2019. C15708709",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Eric Strong",
  "license": "ISC"
}


Is this OK? (yes)
eric:$
eric:$
eric:$ls
package.json
eric:$clear
eric:$npm install express --save
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN store@1.0.0 No repository field.

+ express@4.16.4
added 48 packages from 36 contributors in 6.164s


        New minor version of npm available! 6.2.0 → 6.7.0
    Changelog: https://github.com/npm/cli/releases/tag/v6.7.0
               Run npm install -g npm to update!
```

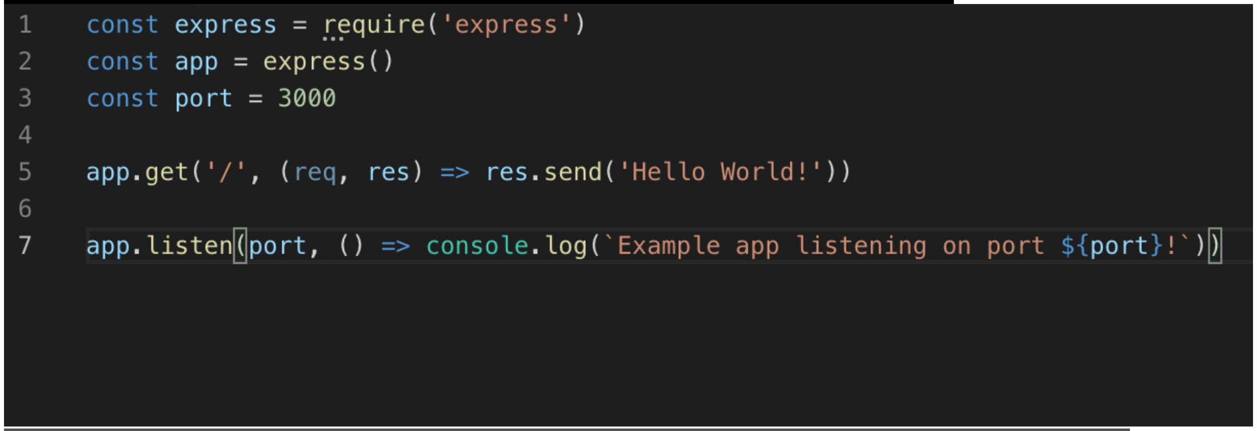
Created an index.js with the following boilerplate
Add in a start: command to the package.json

```
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js"
```

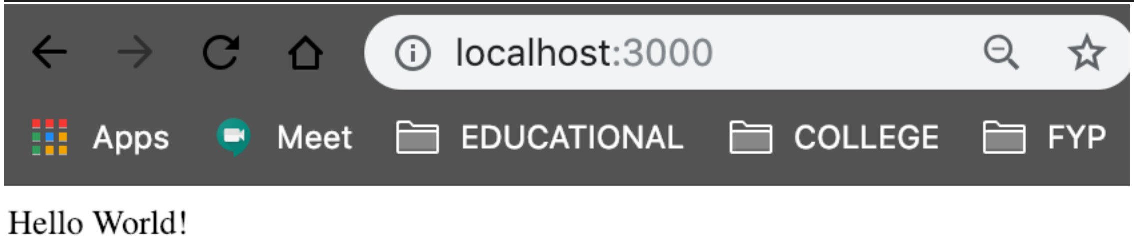
```
},
```



```
eric:$PORT=3000 npm start

> store@1.0.0 start /Users/eric/EAD/lab1
> node index.js

Example app listening on port 3000!
```



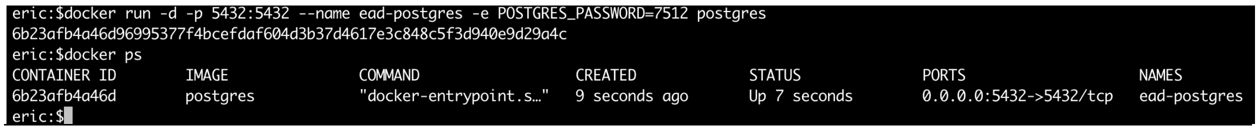
```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```



Hello World!

Install a recent of Postgres (*) on your laptop or sign up for a free cloud-based provider (**)
I have opted to use a docker image: This command gets a docker image from dockerhub, called postgres. It runs it in daemon mode and exposes the port 5432. I name the container ead-postgres so I can easily recognise it

docker run -d -p 5432:5432 --name ead-postgres -e POSTGRES_PASSWORD=7512 postgres

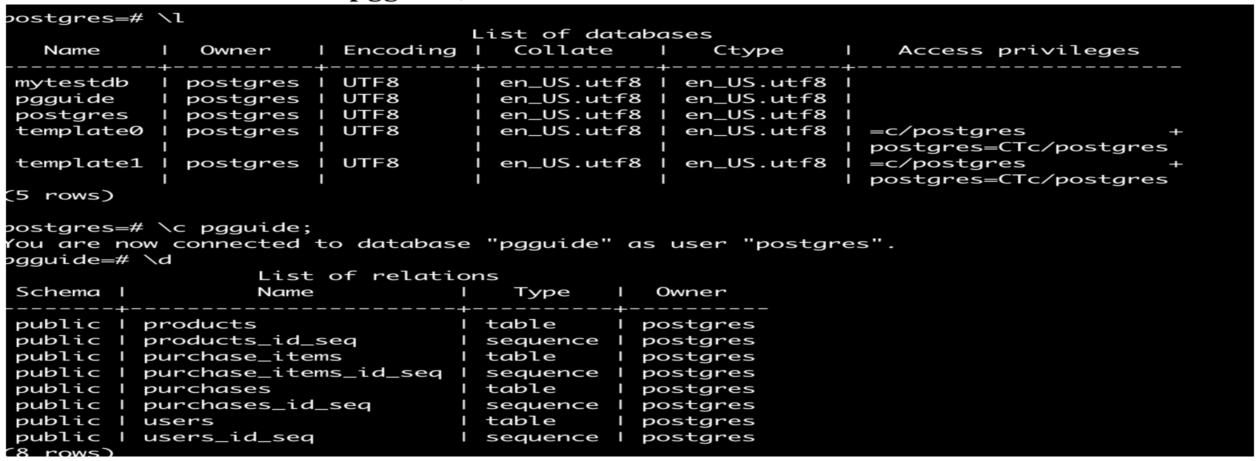


```
eric:$docker run -d -p 5432:5432 --name ead-postgres -e POSTGRES_PASSWORD=7512 postgres
6b23afb4a46d96995377f4bcefdaf604d3b37d4617e3c848c5f3d940e9d29a4c
eric:$docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
6b23afb4a46d        postgres            "docker-entrypoint.s…"   9 seconds ago       Up 7 seconds        0.0.0.0:5432->5432/tcp   ead-postgres
eric:$
```

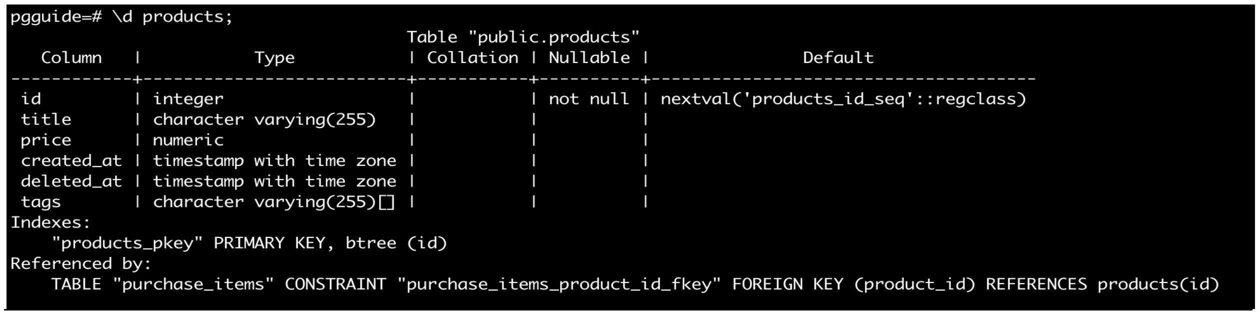
To access container:
docker exec -it ead-postgres bash
install CURL into my docker container
apt-get update; apt-get install curl
get the schema dump
curl -L -O http://cl.ly/173L141n3402/download/example.dump
to create the database from the dump file:

psql -U postgres
CREATE DATABASE pgguide
\q
psql -U postgres
pg_restore --verbose --clean --no-acl --no-owner -h localhost -U postgres -d pgguide example.dump

connect to the database: \c **pgguide;**

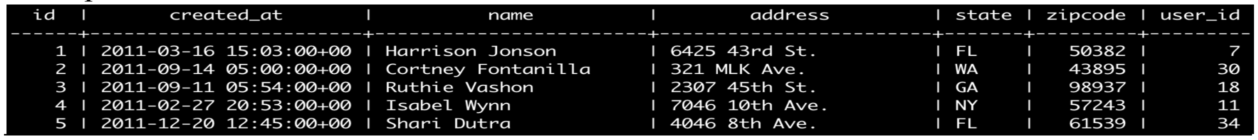


```
ostgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 mytestdb  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 pgguide   | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
(5 rows)

ostgres=# \c pgguide;
You are now connected to database "pgguide" as user "postgres".
pgguide=# \d
                  List of relations
 Schema |          Name           |   Type   |  Owner
--------+-------------------------+----------+----------
 public | products                | table    | postgres
 public | products_id_seq         | sequence | postgres
 public | purchase_items          | table    | postgres
 public | purchase_items_id_seq   | sequence | postgres
 public | purchases               | table    | postgres
 public | purchases_id_seq        | sequence | postgres
 public | users                   | table    | postgres
 public | users_id_seq            | sequence | postgres
(8 rows)
```



```
pgguide=# \d products;
                                 Table "public.products"
   Column   |           Type           | Collation | Nullable |               Default
------------+--------------------------+-----------+----------+--------------------------------------
 id         | integer                  |           | not null | nextval('products_id_seq'::regclass)
 title      | character varying(255)   |           |          |
 price      | numeric                  |           |          |
 created_at | timestamp with time zone |           |          |
 deleted_at | timestamp with time zone |           |          |
 tags       | character varying(255)[] |           |          |
Indexes:
    "products_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "purchase_items" CONSTRAINT "purchase_items_product_id_fkey" FOREIGN KEY (product_id) REFERENCES products(id)
```

Table purchases;



```
  id  |       created_at       |          name          |        address         | state | zipcode | user_id
------+------------------------+------------------------+------------------------+-------+---------+---------
    1 | 2011-03-16 15:03:00+00 | Harrison Jonson        | 6425 43rd St.          | FL    |   50382 |       7
    2 | 2011-09-14 05:00:00+00 | Cortney Fontanilla     | 321 MLK Ave.           | WA    |   43895 |      30
    3 | 2011-09-11 05:54:00+00 | Ruthie Vashon          | 2307 45th St.          | GA    |   98937 |      18
    4 | 2011-02-27 20:53:00+00 | Isabel Wynn            | 7046 10th Ave.         | NY    |   57243 |      11
    5 | 2011-12-20 12:45:00+00 | Shari Dutra            | 4046 8th Ave.          | FL    |   61539 |      34
```

Install massive JS and other libraries:

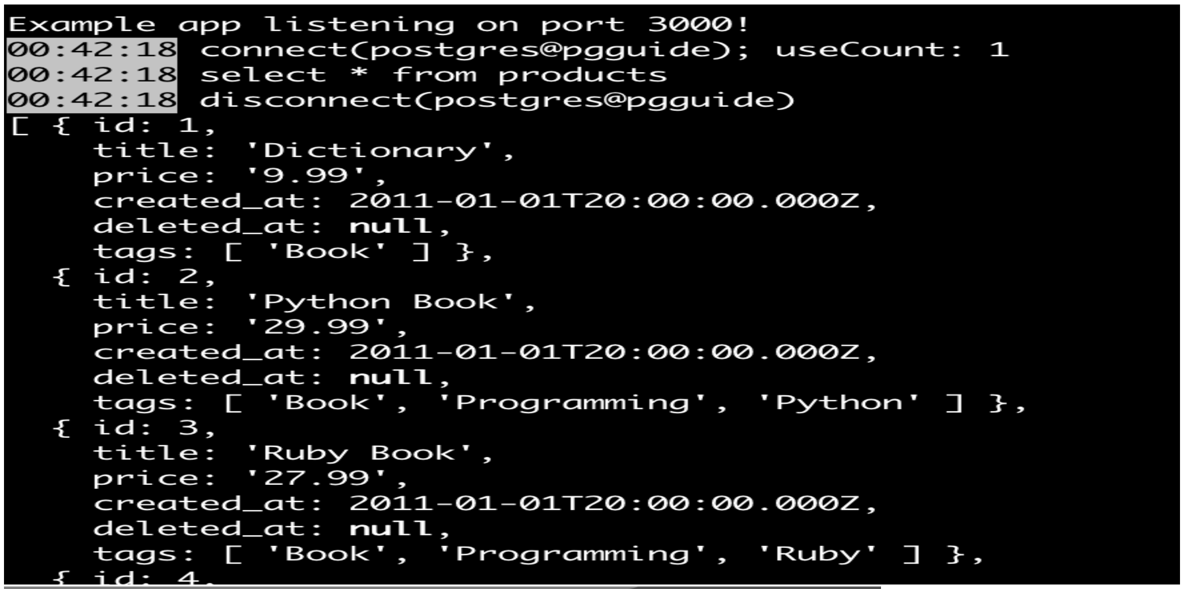
npm install massive *–save*
npm install bluebird *–save*
npm install pg *–save*
npm install pg-monitor *–save*

add code to the index.js

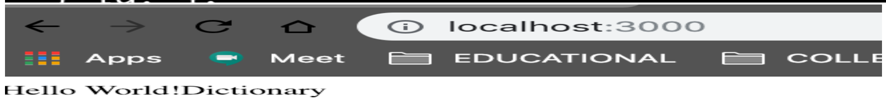
```
//db stuff
const massive = require('massive');
const monitor = require('pg-monitor');

var d = null;
const promise = require('bluebird');
var connectionInfo = 'postgres://postgres:7512@localhost:5432/pgguide';
massive(connectionInfo, {}, {
  promiseLib: promise
}).then(db => {
    monitor.attach(db.driverConfig);
    db.query('select * from products').then(data => {
      // monitor output appears in the console
      d = data;
      console.log(data);
    });
});
```

Add to the get response

```
app.get('/', (req, res) => {
    res.send('Hello World!' + d[0].title);
}); // request response
```

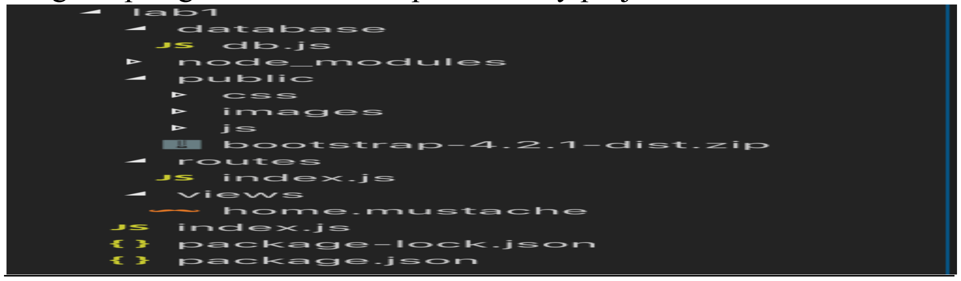


```
Example app listening on port 3000!
00:42:18 connect(postgres@pgguide); useCount: 1
00:42:18 select * from products
00:42:18 disconnect(postgres@pgguide)
[ { id: 1,
    title: 'Dictionary',
    price: '9.99',
    created_at: 2011-01-01T20:00:00.000Z,
    deleted_at: null,
    tags: [ 'Book' ] },
  { id: 2,
    title: 'Python Book',
    price: '29.99',
    created_at: 2011-01-01T20:00:00.000Z,
    deleted_at: null,
    tags: [ 'Book', 'Programming', 'Python' ] },
  { id: 3,
    title: 'Ruby Book',
    price: '27.99',
    created_at: 2011-01-01T20:00:00.000Z,
    deleted_at: null,
    tags: [ 'Book', 'Programming', 'Ruby' ] },
  { id: 4,
```



Hello World!Dictionary

For setting up part 5 to view your tables and see they exist run the endpoint http://localhost:3000/settingup
I went back and made some changes to the project to ensure that I could properly share a database object and call it from a separate file, instead of having all code lumped into on big file. I also set up a router to run endpoints better and make my code more modularized. I set up templating with Mustache to have some dynamic data render in the browser if needed. I also included static assets like bootstrap js and css for styling. I am still using MassiveJS and a docker image of postgres. Here is a snap shot of my project



```
◢ lab1
  ◢ database
    JS db.js
  ▸ node_modules
  ◢ public
    ▸ css
    ▸ images
    ▸ js
      bootstrap-4.2.1-dist.zip
  ◢ routes
    JS index.js
  ◢ views
    home.mustache
  JS index.js
  {} package-lock.json
  {} package.json
```

Problem Set 1

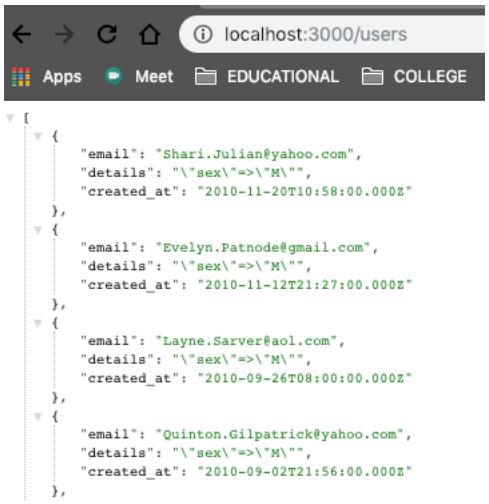
I created a universal function that would allow for all endpoints to easily and dynamically parse a query to the db object in order to execute a query.

```
// function to be used to query from the database
function getdata(table, res, q) {
    console.log('------------------ ' + table.toUpperCase() + ' -----------------');
    console.log(q);
    const db = dbObj.get('db');
    db.query(q).then(data => {
        // output to appear in browser
        res.json(data);
    })
}
```

1.1 Endpoint

```
// endpoint 1.1
router.get('/users', (req, res) => {
    //problem set: 1 part 1- users email and sex in order of most recently created.
    var q = 'select email, details, created_at from users ORDER BY created_at DESC;';
    getdata('users', res, q);
}); // request response
```

1.1 output

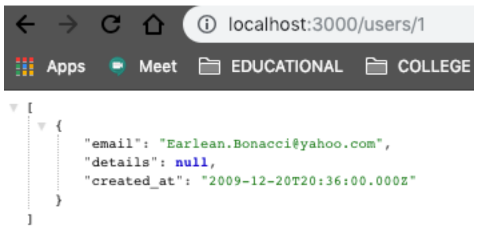


```
[
  {
    "email": "Shari.Julian@yahoo.com",
    "details": "\"sex\"=>\"M\"",
    "created_at": "2010-11-20T10:58:00.000Z"
  },
  {
    "email": "Evelyn.Patnode@gmail.com",
    "details": "\"sex\"=>\"M\"",
    "created_at": "2010-11-12T21:27:00.000Z"
  },
  {
    "email": "Layne.Sarver@aol.com",
    "details": "\"sex\"=>\"M\"",
    "created_at": "2010-09-26T08:00:00.000Z"
  },
  {
    "email": "Quinton.Gilpatrick@yahoo.com",
    "details": "\"sex\"=>\"M\"",
    "created_at": "2010-09-02T21:56:00.000Z"
  },
```

1.2 endpoint

```
//endpoint 1.2
router.get('/users/:id', (req, res) => {
    //problem set: 1 part 2 - users email and sex in order of most recently created where id = :id.
    var id = req.params.id;
    console.log('id:' + id)
    var q = 'select email, details, created_at from users where id = ' + id + ';';
    getdata('users', res, q);
}); // request response
```

1.2 output

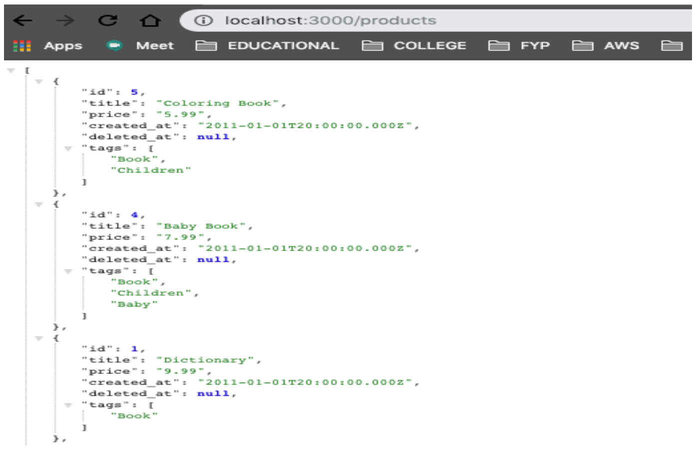


```
[
  {
    "email": "Earlean.Bonacci@yahoo.com",
    "details": null,
    "created_at": "2009-12-20T20:36:00.000Z"
  }
]
```

1.3 endpoint

```
//endpoint 1.3
router.get('/products', (req, res, next) => {

    if (!req.query.name) {
        console.log('no params')
        //problem set: 1 part 3- List all products in ascending order of price.
        var q = 'select * from products ORDER BY price ASC ;';
        getdata('products', res, q);
    } else {
        next();
    }

}); // request response
```

1.3 output



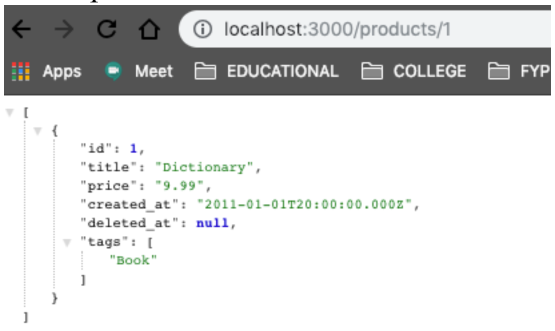
```
[
  {
    "id": 5,
    "title": "Coloring Book",
    "price": "5.99",
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null,
    "tags": [
      "Book",
      "Children"
    ]
  },
  {
    "id": 4,
    "title": "Baby Book",
    "price": "7.99",
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null,
    "tags": [
      "Book",
      "Children",
      "Baby"
    ]
  },
  {
    "id": 1,
    "title": "Dictionary",
    "price": "9.99",
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null,
    "tags": [
      "Book"
    ]
  },
```

1.4 endpoint

```
//endpoint 1.4
router.get('/products/:id', (req, res) => {
    //problem set: 1 part 4 – Show details of the specified products.
    var id = req.params.id;
    console.log('id:' + id)

```

```
      var q = 'select * from products  where id = ' + id + ' ORDER BY price ASC;';
      getdata('products', res, q);
 }); // request response
```

1.4 output

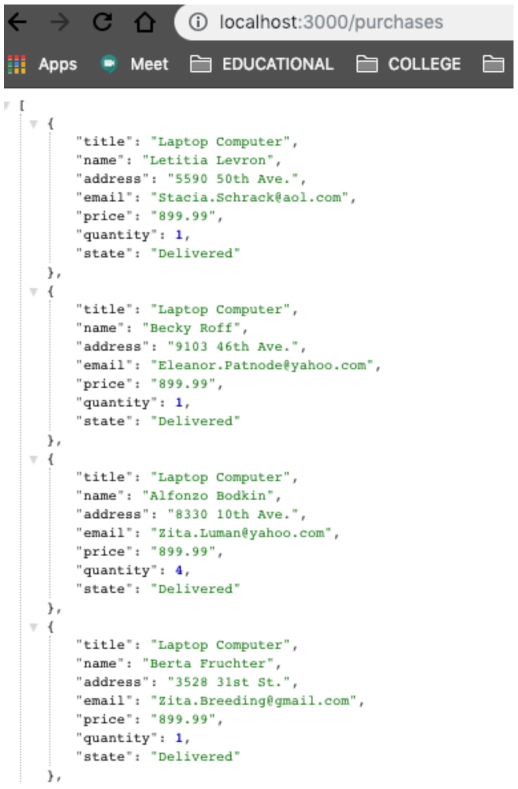


```
[
  {
    "id": 1,
    "title": "Dictionary",
    "price": "9.99",
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null,
    "tags": [
      "Book"
    ]
  }
]
```

1.5 endpoint

```

//endpoint 1.5
router.get('/purchases', (req, res) => {
    //problem set: 1 part 5– List purchase items to include the receiver’s name and,
address, the purchaser’s email address and the price, quantity and delivery status of
the purchased item. Order by price in descending order.
    var q = `
    SELECT
    Products.title,
    purchases.name,
    purchases.address,
    users.email,
    purchase_items.price,
    purchase_items.quantity,
    Purchase_items.state
    FROM purchases
    INNER JOIN users on purchases.user_id = users.id
    INNER JOIN purchase_items on purchase_items.purchase_id = purchases.id
    INNER JOIN products on purchase_items.product_id = products.id
    ORDER BY purchase_items.price DESC;`;

    getdata('users, purchases, purchaseitems and products', res, q);
}); // request response
```

1.5 output



```
[
  {
      "title": "Laptop Computer",
      "name": "Letitia Levron",
      "address": "5590 50th Ave.",
      "email": "Stacia.Schrack@aol.com",
      "price": "899.99",
      "quantity": 1,
      "state": "Delivered"
  },
  {
      "title": "Laptop Computer",
      "name": "Becky Roff",
      "address": "9103 46th Ave.",
      "email": "Eleanor.Patnode@yahoo.com",
      "price": "899.99",
      "quantity": 1,
      "state": "Delivered"
  },
  {
      "title": "Laptop Computer",
      "name": "Alfonzo Bodkin",
      "address": "8330 10th Ave.",
      "email": "Zita.Luman@yahoo.com",
      "price": "899.99",
      "quantity": 4,
      "state": "Delivered"
  },
  {
      "title": "Laptop Computer",
      "name": "Berta Fruchter",
      "address": "3528 31st St.",
      "email": "Zita.Breeding@gmail.com",
      "price": "899.99",
      "quantity": 1,
      "state": "Delivered"
  },
```

Problem Set 2

I first had to write some additional logic to handle the endpoint of /products?name=string
This endpoint was already used in part 1.3. So I needed to add some logic to check if the req.query.name exists. If it does then it executes a certain logic statement.
2.1 endpoint

```
//Problem set 2. filtering by name. badly.
router.get('/products', (req, res) => {
    if (req.query.name) {
        console.log('got params!');
        const name = req.query.name; // name is the actual variable name
        console.log(name);
        var q = "select * from products where title = '" + name + "'"; // NOTE no
semicolon so I can try SQLINJECTION
        getdata('products', res, q);

    }

}); // request response
```

Testing for SQLInjection

http://localhost:3000/products?name=Dictionary' or title = 'Python Book
http://localhost:3000/products?name=Ruby Book' or title = 'Baby Book
http://localhost:3000/products?name=Ruby Book' or title = 'Baby Book
http://localhost:3000/products?name=Ruby Book' ; select price, title from products where title = 'Python Book

more dangerous sqlinjection examples:

http://localhost:3000/products?name=Dictionary' ; select * from users where id > '0

http://localhost:3000/products?name=Dictionary' ; Delete from purchase_items ;select * from products where id > '0

Problem set 3

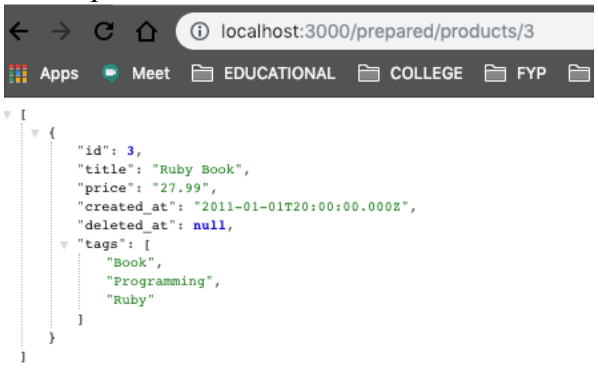
Solutions for SQL Injection. I used a parameterized query(Prepared Statement) in order to prevent any SQL injection attack. I created the following function:

```
//function to implement prepared statements for problem set 3
function preparedstatement(res, params) {
    console.log('----------------- Prepared Statement ----------------');

    //get database object
    const db = dbObj.get('db');
    console.log(params.id);
    db.query(
        'select * from products where id = ${id};', {
            id: params.id
        }
    ).then(data => {
        // returning the output
        res.json(data);
    });

}
```

3.1 endpoint

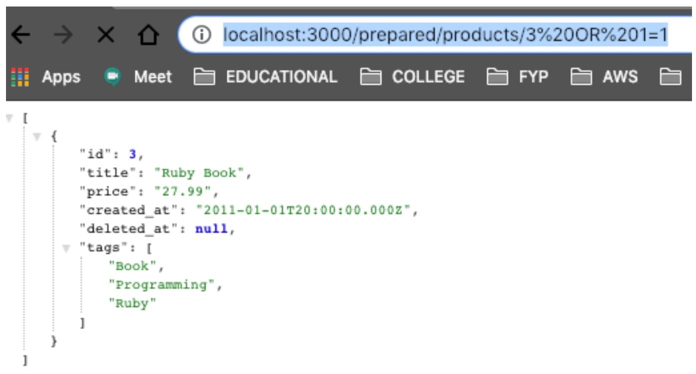
```
//Problem set 3. 3.1 Using prepared statements
router.get('/prepared/products/:id', (req, res) => {
    const params = req.params;
    preparedstatement(res, params);

    //test that it is imposible to use SQL injection
    //http://localhost:3000/prepared/products/1 OR 1=1
})
```

3.1 output



```
[
  {
    "id": 3,
    "title": "Ruby Book",
    "price": "27.99",
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null,
    "tags": [
      "Book",
      "Programming",
      "Ruby"
    ]
  }
]
```

3.1 output with attempt to inject – does not return anything it shouldn't



localhost:3000/prepared/products/3%20OR%201=1

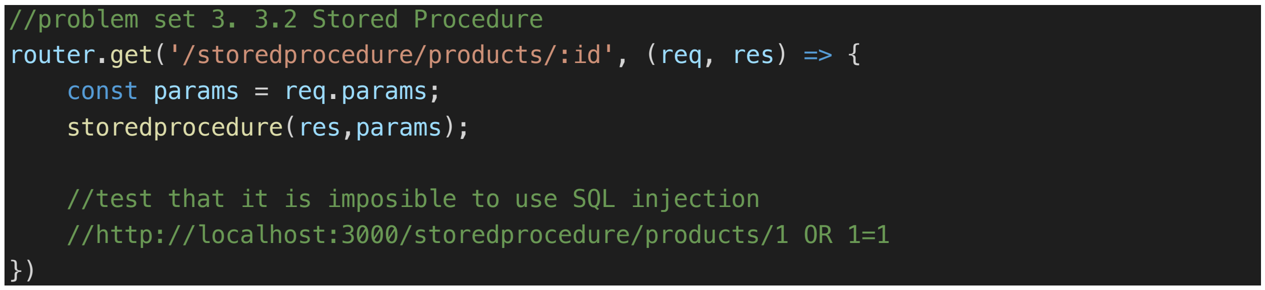
```
[
  {
    "id": 3,
    "title": "Ruby Book",
    "price": "27.99",
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null,
    "tags": [
      "Book",
      "Programming",
      "Ruby"
    ]
  }
]
```

3.2 Stored Procedure (function set). I created this via the postgres shell.

```
CREATE OR REPLACE FUNCTION erictest(id integer)
Returns setof products AS $func$
DECLARE
Sql text:='SELECT * FROM products WHERE id = $1';
BEGIN
RETURN QUERY EXECUTE sql
        USING id;
END;
$func$ LANGUAGE plpgsql;
```

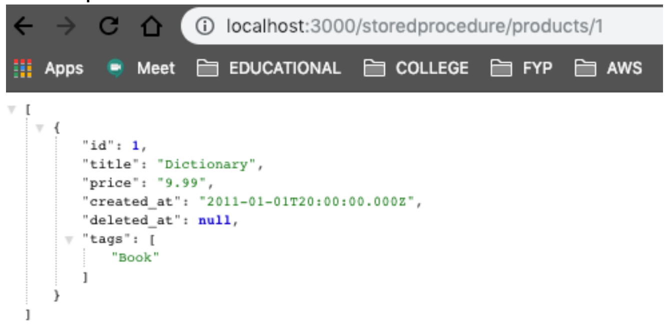
To call the stored procedure I simply use : select * from erictest(4);
I pass in the parameter I want into the erictest() function. This is used as an argument in the SELECT statement in the stored procedure.

3.2 endpoint



```
//problem set 3. 3.2 Stored Procedure
router.get('/storedprocedure/products/:id', (req, res) => {
    const params = req.params;
    storedprocedure(res,params);

    //test that it is imposible to use SQL injection
    //http://localhost:3000/storedprocedure/products/1 OR 1=1
})
```

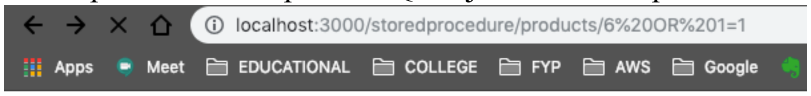
3.2 output



localhost:3000/storedprocedure/products/1

```
[
  {
    "id": 1,
    "title": "Dictionary",
    "price": "9.99",
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null,
    "tags": [
      "Book"
    ]
  }
]
```

3.2 output with atttemp with SQL Injection – as expected nothing is returned that should not be.



localhost:3000/storedprocedure/products/6%20OR%201=1

This project can be cloned on Github via:
https://github.com/ericstrongDIT/EnterpiseAppDev

Problem set 4

Install the dependencies
Npm install -g nodemon
Npm install sequelize --save
Npm install -g sequelize-cli
npm install --save pg pg-hstore

create the database connection and test it.

```
//////////// Database object creating ///////////////
const express = require('express');
var app = express();
const Sequelize = require('sequelize');
//const sequelize = new Sequelize('postgres://postgres:7512@localhost:5432/pgguide');
// simple way of connecting

//DB instance
const sequelize = new Sequelize('pgguide', 'postgres', 7512, {
  host: 'localhost',
  dialect: 'postgres',
  operatorsAliases: false,
  define: {
    timestamps: false
},

  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
});

//4.1 test connection - verify we have connection
sequelize
  .authenticate().then(() => {
    console.log('connected to database !');
  }).catch(err => {
    console.error('Unable to connect to the database:', err);
  });

app.set('db', sequelize);

module.exports = app;
```

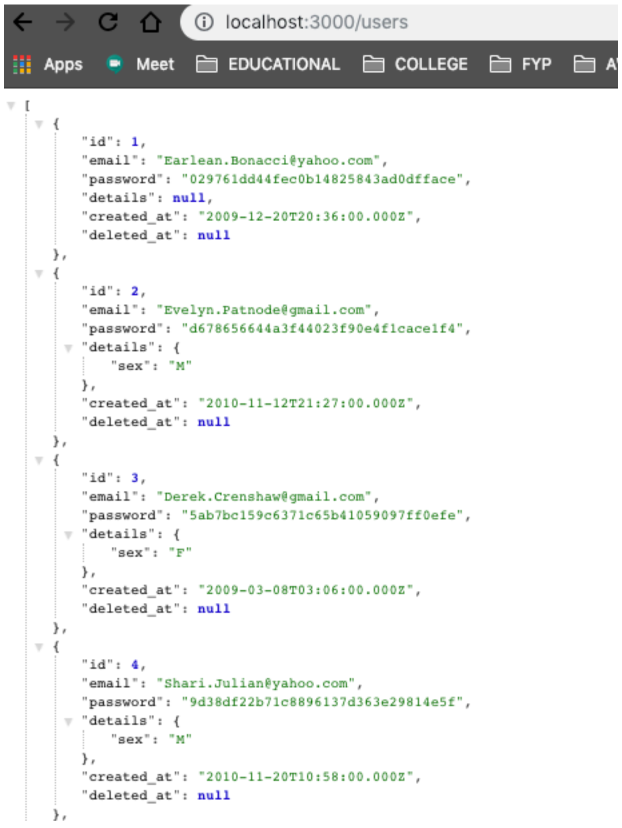
Now that we have the database connected we can do a quick test to get some data out of the database by creating a model

```
const Sequelize = require('sequelize');
const dbObj = require('../database/db');
const db = dbObj.get('db');
const express = require('express');
var app = express();

const Datamodel = db.define('user',{
     id:{
         type: Sequelize.INTEGER,
         primaryKey: true
     },
     email:{
        type:Sequelize.STRING
     },
     password:{
         type:Sequelize.STRING
     },
     details: {
         type: Sequelize.HSTORE
     }
});

app.set('Datamodel', Datamodel);
module.exports = app;
```

Endpoint

```
const datamodelObj = require('../models/datamodels');
const Datamodel = datamodelObj.get('Datamodel');
// getting all of the users
router.get('/users',(req,res)=>{

    Datamodel.findAll().then(users =>{
        //console.log(users);
        res.send(users);
        }
     ).catch(err => console.log(err));

});
```

Display data



```
localhost:3000/users
```

```
[
  {
    "id": 1,
    "email": "Earlean.Bonacci@yahoo.com",
    "password": "029761dd44fec0b14825843ad0dfface",
    "details": null,
    "created_at": "2009-12-20T20:36:00.000Z",
    "deleted_at": null
  },
  {
    "id": 2,
    "email": "Evelyn.Patnode@gmail.com",
    "password": "d678656644a3f44023f90e4f1cace1f4",
    "details": {
      "sex": "M"
    },
    "created_at": "2010-11-12T21:27:00.000Z",
    "deleted_at": null
  },
  {
    "id": 3,
    "email": "Derek.Crenshaw@gmail.com",
    "password": "5ab7bc159c6371c65b41059097ff0efe",
    "details": {
      "sex": "F"
    },
    "created_at": "2009-03-08T03:06:00.000Z",
    "deleted_at": null
  },
  {
    "id": 4,
    "email": "Shari.Julian@yahoo.com",
    "password": "9d38df22b71c8896137d363e29814e5f",
    "details": {
      "sex": "M"
    },
    "created_at": "2010-11-20T10:58:00.000Z",
    "deleted_at": null
  },
```

Create **Sequalize migrations** for the `pgguide` sample database

Use the command:
Sequelize init
This creates the following:

    Created "config/config.json" // ensure the config.json file has you valid db info
    models folder at "/Users/eric/EAD/lab1part4/models" already exists.
    Successfully created migrations folder at "/Users/eric/EAD/lab1part4/migrations".
    Successfully created seeders folder at "/Users/eric/EAD/lab1part4/seeders".

sequelize migration:generate --name testingMigrations
This creates a 20190204210224-testingMigrations.js file
Where we can specify our migrations up and down

```
module.exports = {
  up: (queryInterface, Sequelize) => {

    Promise.all( [
      queryInterface.renameColumn('tablename1','oldname','newname'),
      queryInterface.renameColumn('tablename2','oldname','newname'),
      queryInterface.renameColumn('tablename3','oldname','newname'),

    ]);
  },

  down: (queryInterface, Sequelize) => {
    //reverting if something goes wrong
    Promise.all( [
      queryInterface.renameColumn('tablename1','newname','oldname'),
      queryInterface.renameColumn('tablename2','newname','oldname'),
      queryInterface.renameColumn('tablename3','newname','oldname'),


    ]);



  }
};
```

Once all tables have been identified and you run your migrations. Use the following command
sequelize db:migrate
This will migration our database. If anything goes wrong with the database. We can always use the down: migrations to revert back

Ensure that the appropriate associations and referential integrity checking are set up in your models

Users

```
/*
id          | integer                  |            | not null |
nextval('users_id_seq'::regclass)
 email      | character varying(255)   |            |          |
 password   | character varying(255)   |            |          |
 details    | hstore                   |            |          |
 created_at | timestamp with time zone |            |          |
 deleted_at | timestamp with time zone |            |          |
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "purchases" CONSTRAINT "purchases_user_id_fkey" FOREIGN KEY (user_id)
REFERENCES users(id)
*/
const User = db.define('user', {
    id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        allowNull: false,
        autoIncrement: true
    },
    email: {
        type: Sequelize.STRING
    },
    password: {
        type: Sequelize.STRING
    },
    details: {
        type: Sequelize.HSTORE
    }
});
```

Products

```
/*
                                      Table "public.products"
   Column   |            Type          | Collation | Nullable |                Default
------------+--------------------------+-----------+----------+-----------------------
---------------
 id         | integer                  |           | not null |
nextval('products_id_seq'::regclass)
 title      | character varying(255)   |           |          |
 price      | numeric                  |           |          |
 created_at | timestamp with time zone |           |          |
 deleted_at | timestamp with time zone |           |          |
 tags       | character varying(255)[] |           |          |

*/
const Products = db.define('products', {
    id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        allowNull: false,
        autoIncrement: true
    },
    title: {
        type: Sequelize.STRING
    },
    price: {
        type: Sequelize.NUMERIC
    },
    tags: {
        type: Sequelize.HSTORE
    }
});
```

Purchases

```
//Note that if you are using Sequelize migrations you will need to add the createdAt
and updatedAt fields to your migration definition:
/*
                                       Table "public.purchases"
   Column   |           Type           | Collation | Nullable |                Default
------------+--------------------------+-----------+----------+------------------------
----------------
 id         | integer                  |           | not null |
nextval('purchases_id_seq'::regclass)
 created_at | timestamp with time zone |           |          |
 name       | character varying(255)   |           |          |
 address    | character varying(255)   |           |          |
 state      | character varying(2)     |           |          |
 zipcode    | integer                  |           |          |
 user_id    | integer                  |           |          |
Indexes:
    "purchases_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "purchases_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(id)
Referenced by:
    TABLE "purchase_items" CONSTRAINT "purchase_items_purchase_id_fkey" FOREIGN KEY
(purchase_id) REFERENCES purchases(id)
*/

const Purchases = db.define('purchases', {
    id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        allowNull: false,
        autoIncrement: true
    },
    name: {
        type: Sequelize.STRING
    },
    address: {
        type: Sequelize.STRING
    },
    state: {
        type: Sequelize.STRING
    },
    zipcode: {
        type: Sequelize.INTEGER
    },
    user_id: {
        type: Sequelize.INTEGER,
        references: {
            // This is a reference to another model
            model: Users,

            // This is the column name of the referenced model
            key: 'id',

            // This declares when to check the foreign key constraint. PostgreSQL
only.
            deferrable: Sequelize.Deferrable.INITIALLY_IMMEDIATE
          }
    },

});
```

Purchase_items

```
/*
                                      Table "public.purchase_items"
   Column    |          Type          | Collation | Nullable |
Default
-------------+------------------------+-----------+----------+------------------------
--------------------
 id          | integer                |           | not null |
nextval('purchase_items_id_seq'::regclass)
 purchase_id | integer                |           |          |
 product_id  | integer                |           |          |
 price       | numeric                |           |          |
 quantity    | integer                |           |          |
 state       | character varying(255) |           |          |
Foreign-key constraints:
    "purchase_items_purchase_id_fkey" FOREIGN KEY (purchase_id) REFERENCES
purchases(id)
*/

const Purchase_items = db.define('purchase_items', {
    id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        allowNull: false,
        autoIncrement: true
    },
    purchase_id: {
        type: Sequelize.INTEGER,
        references: {
            // This is a reference to another model
            model: Purchases,
            // This is the column name of the referenced model
            key: 'id',

            // This declares when to check the foreign key constraint. PostgreSQL
only.
            deferrable: Sequelize.Deferrable.INITIALLY_IMMEDIATE
          }
    },
    product_id: {
        type: Sequelize.INTEGER,
    },
    price: {
        type: Sequelize.NUMERIC
    },
    quantity: {
        type: Sequelize.INTEGER
    },
    state: {
        type: Sequelize.STRING
    }


});
```

Problem set 5

Populate the database with some new data.
Running the endpoint will create the following entries to the database:
Endpoint: http://localhost:3000/createtestdata

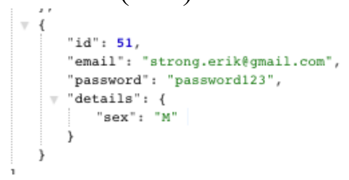
```
//Problem set 5 - creating test data
router.get('/createtestdata', (req, res) => {
    //Calling models to create data
    create_user('strong.erik@gmail.com', 'password123', {
        sex: 'M'
    });

    create_product("Drum Kit", 1500.00 );

    create_purchases("Eric Strong","19 Riversdale Palmerstown","DU",01,51); // Eric is
user 51

    create_purchase_items( 1001, 24, 1500.00 ,1 , "Pending" ); // purchase_id 1001 ,
product_id 24 price 1500, quant 1 this may need to be dynamic

    res.send('Data has been inserted!');
    //res.status(200);


});
```

Code Functions

```
//for problem set 5
// Functions to create new data - I will pass parameters of data into each function to
create a new data row
function create_user(email, password, details) {

    Users.create({
            email: email,
            password: password,
            details: details
        })
        //checking it doesnt already exist
        .then(() => Users.findOrCreate({
            where: {
                email: email,
                password: password,
                details: details
            }
        }))
        .spread((users, created) => {
            console.log(users.get({
                plain: true
            }))
            console.log(created);
        })

}
// function to create a new product
function create_product(title,price,tags){
    Products.create({ title:title,price:price,tags:tags })
    //checking it doesnt already exist
  .then(() => Products.findOrCreate({where: {title:title,price:price,tags:tags}}))
  .spread((products, created) => {
    console.log(products.get({
      plain: true
    }))
    console.log(created);
  })

}

//function to create a new purchase
function create_purchases(name,address,state,zipcode,user_id){
    Purchases.create({
name:name,address:address,state:state,zipcode:zipcode,user_id:user_id })
    //checking it doesnt already exist
  .then(() => Purchases.findOrCreate({where:
{name:name,address:address,state:state,zipcode:zipcode,user_id:user_id}}))
  .spread((purchases, created) => {
    console.log(purchases.get({
      plain: true
    }))
    console.log(created);
  })

}

//function to create a purchase items entry
function create_purchase_items(purchase_id,product_id,price,quantity,state){
    Purchase_items.create({
purchase_id:purchase_id,product_id:product_id,price:price,quantity:quantity,state:stat
e })
    //checking it doesnt already exist
  .then(() => Purchase_items.findOrCreate({where:
{purchase_id:purchase_id,product_id:product_id,price:price,quantity:quantity,state:sta
te}}))
  .spread((purchase_items, created) => {
    console.log(purchase_items.get({
      plain: true
    }))
    console.log(created);
  })
```

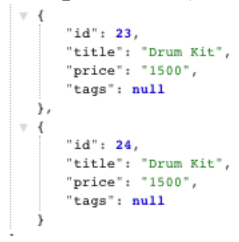
Outputs:

New users (Eric)



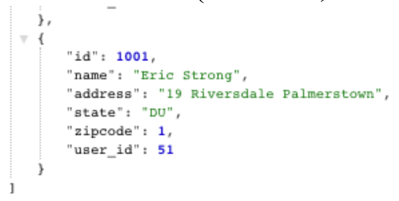
```
},
{
    "id": 51,
    "email": "strong.erik@gmail.com",
    "password": "password123",
    "details": {
        "sex": "M"
    }
}
]
```

New products (Drum Kit)



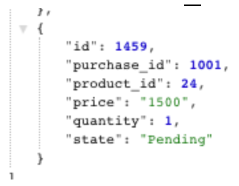
```
{
    "id": 23,
    "title": "Drum Kit",
    "price": "1500",
    "tags": null
},
{
    "id": 24,
    "title": "Drum Kit",
    "price": "1500",
    "tags": null
}
]
```

New Purchases (Eric Info, and reference to user)



```
},
{
    "id": 1001,
    "name": "Eric Strong",
    "address": "19 Riversdale Palmerstown",
    "state": "DU",
    "zipcode": 1,
    "user_id": 51
}
]
```

New Purchase_items



```
},
{
    "id": 1459,
    "purchase_id": 1001,
    "product_id": 24,
    "price": "1500",
    "quantity": 1,
    "state": "Pending"
}
]
```

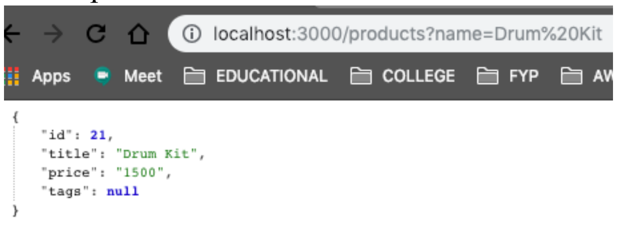
Problem set 6

RESTFul API using ORM.
Note**: npm install body-parser –save** was needed for post and put requests

6.1 endpoint

```
//6.1
router.get('/products', (req, res) => {
    if (req.query.name) {
        console.log('got params!');
        var name = req.query.name; // name is the actual variable name
        console.log(name);

        // search for attributes
        Products.findOne({
            where: {
                title: name
            }
        }).then(products => {
            res.send(products);
        })


    }
});
```

6.1 output



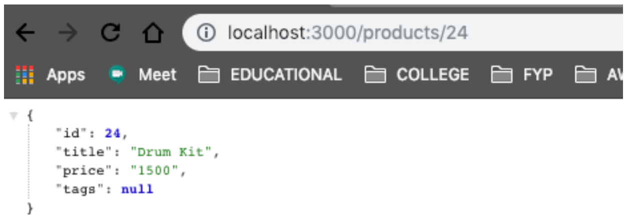
localhost:3000/products?name=Drum%20Kit

```
{
    "id": 21,
    "title": "Drum Kit",
    "price": "1500",
    "tags": null
}
```

6.2 endpoint

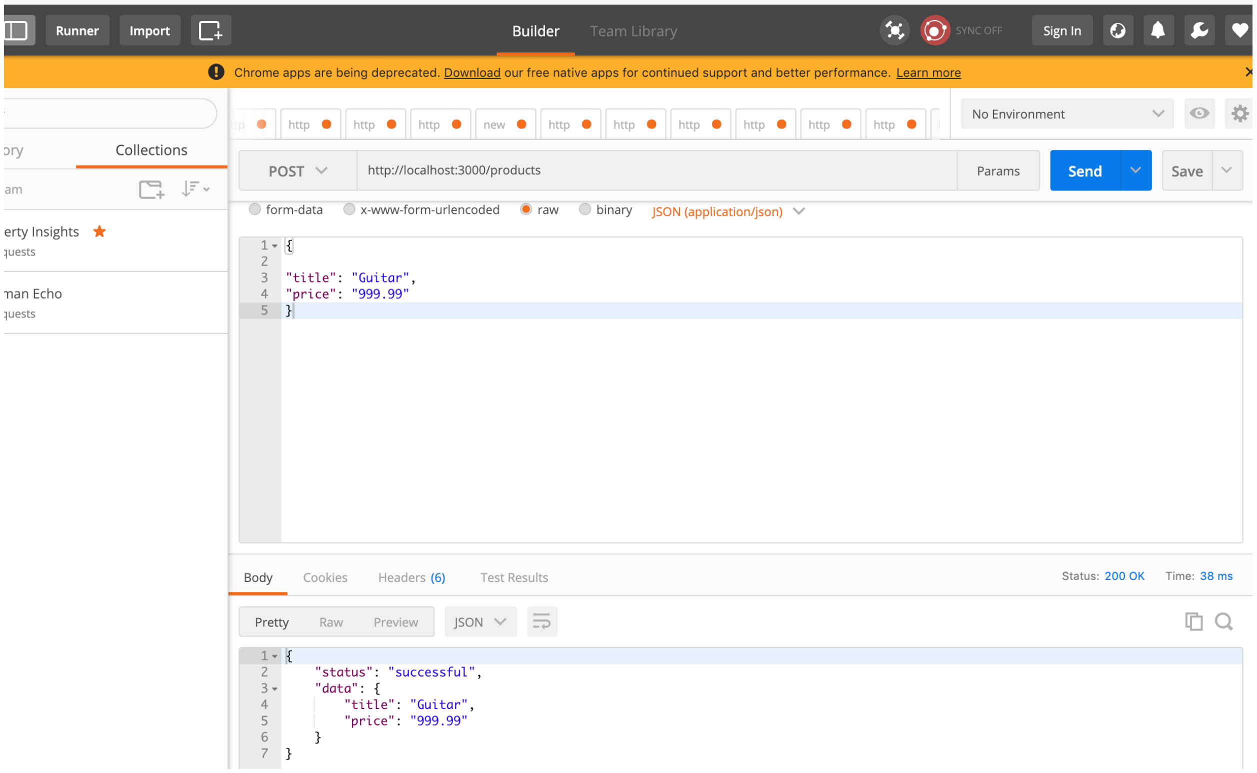
```
// 6.2
router.get('/products/:id', (req, res) => {
    var id = req.params.id;
    console.log('id:' + id);

    Products.findByPk(id).then(products => {
        res.send(products);
      })
```

6.2 output



```
{
    "id": 24,
    "title": "Drum Kit",
    "price": "1500",
    "tags": null
}
```

6.3 – POST – using POSTMAN



POST http://localhost:3000/products

raw JSON (application/json)

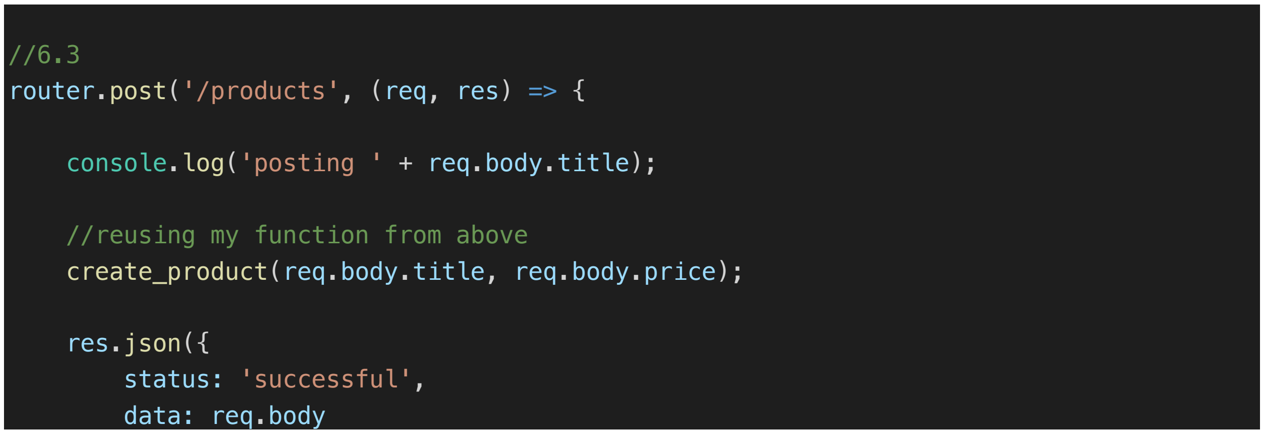
```
{

"title": "Guitar",
"price": "999.99"
}
```

Status: 200 OK Time: 38 ms

JSON

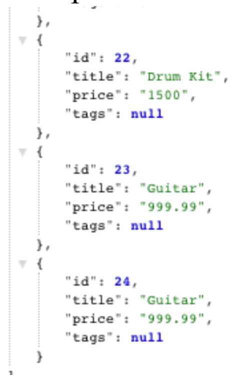
```
{
    "status": "successful",
    "data": {
        "title": "Guitar",
        "price": "999.99"
    }
}
```

6.3 endpoint

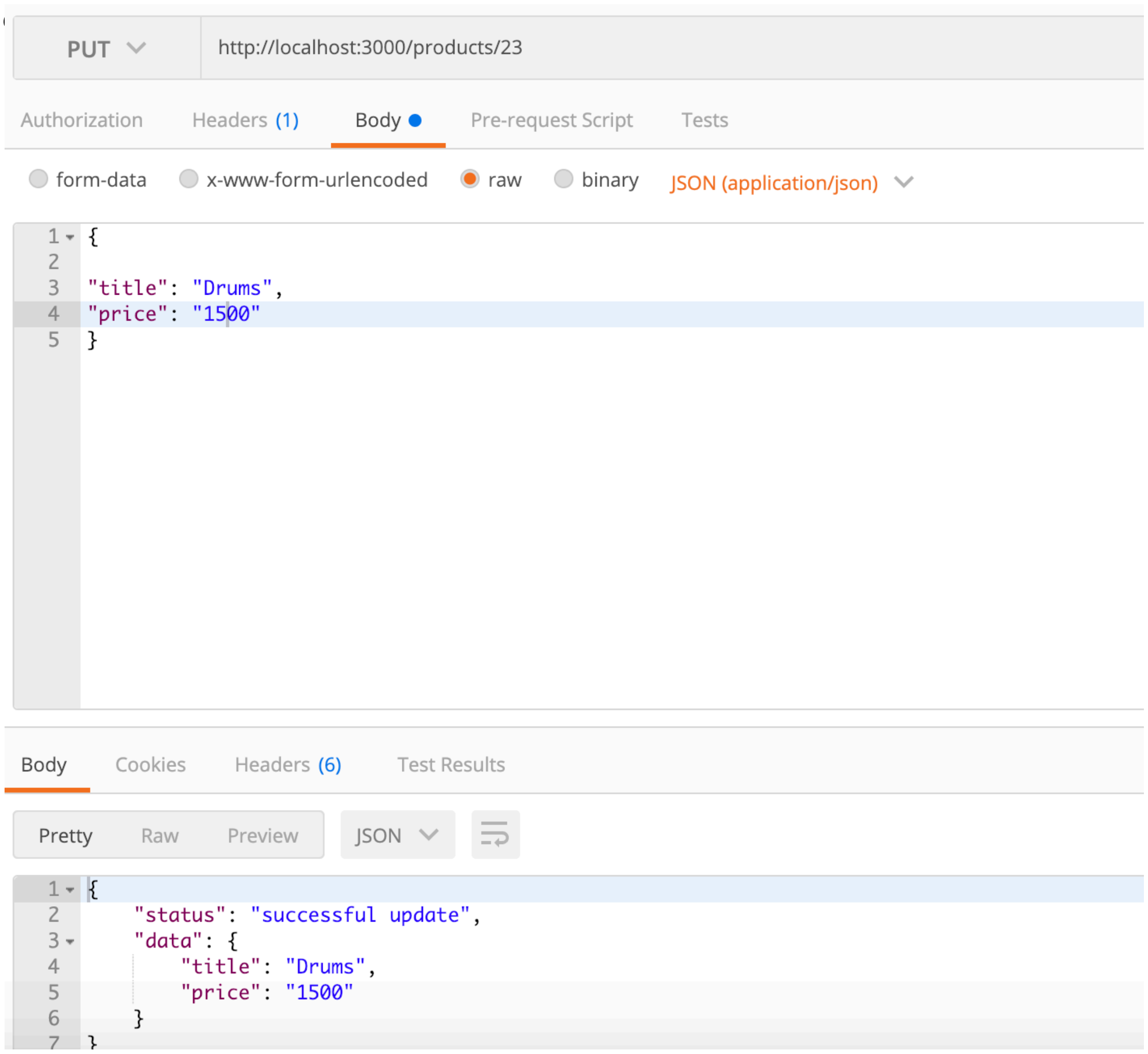


```
//6.3
router.post('/products', (req, res) => {

    console.log('posting ' + req.body.title);

    //reusing my function from above
    create_product(req.body.title, req.body.price);

    res.json({
        status: 'successful',
        data: req.body
    });

}); // request response
```

6.3 output



```
    },
  ▼ {
        "id": 22,
        "title": "Drum Kit",
        "price": "1500",
        "tags": null
    },
  ▼ {
        "id": 23,
        "title": "Guitar",
        "price": "999.99",
        "tags": null
    },
  ▼ {
        "id": 24,
        "title": "Guitar",
        "price": "999.99",
        "tags": null
    }
```


PUT http://localhost:3000/products/23

Authorization | Headers (1) | Body | Pre-request Script | Tests

form-data | x-www-form-urlencoded | [✓ raw] | binary | JSON (application/json)

```
{

"title": "Drums",
"price": "1500"
}
```

Body | Cookies | Headers (6) | Test Results

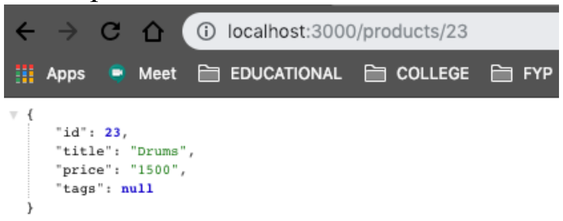
Pretty | Raw | Preview | JSON

```
{
    "status": "successful update",
    "data": {
        "title": "Drums",
        "price": "1500"
    }
}
```

6.4 endpoint

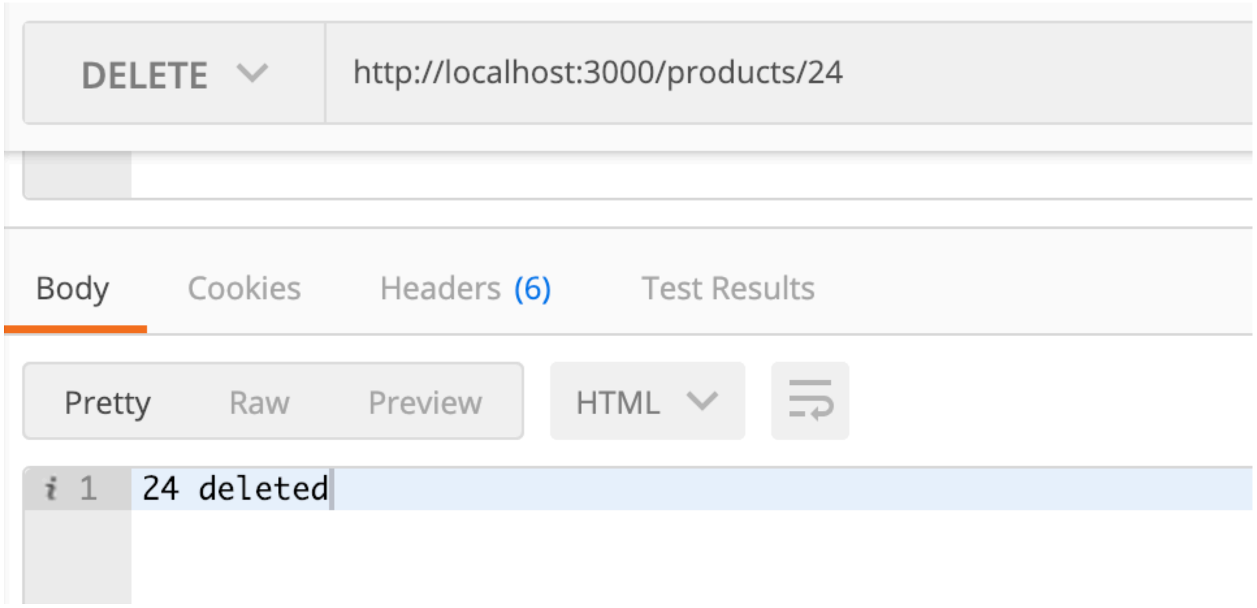
```
//6.4
router.put('/products/:id', (req,res) =>{
    var id = req.params.id;
    console.log('PUT - Updating');
    Products.update({
    updatedAt: new Date(),
    title: req.body.title,
    price: req.body.price,
    tags: req.body.tags,
  }, {
    where: {
        id: id
    }
  });
  // UPDATE product SET updatedAt = x title = x price = x tags = x WHERE id =
param.id;
  res.json({
    status: 'successful update',
    data: req.body
});
});
```

6.4 output



```
{
    "id": 23,
    "title": "Drums",
    "price": "1500",
    "tags": null
}
```

6.5 DELETE – Remove an existing product



```
24 deleted
```

6.5 endpoint

```
//6.5
router.delete('/products/:id', (req,res) =>{
    var id = req.params.id;

    console.log('DELETING');
    Products.destroy({
        where: {
            id:id
        }
    });
    res.send(id + ' deleted');
});
```

Note that the record will not show up if you do an /products endpoint