1 - REST API, SQL and ORM

1. Learning Outcomes

On completion of this lab you will have:

- 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

2. Organisation

Please complete the exercises individually.

3. Grading

This worksheet is worth up to 10% of your overall module grade. You must attend and sign in at 6 labs in order to obtain full credit for your submitted worksheets. You may work on this worksheet during lab 1 and lab 2 with instructor assistance. You may also be asked to demonstrate your submission in order to receive credit - see below.

4. Submission

The deadline for submission is Sunday Feb 17, 2019 @23:59 through Github.

The work and submission workflow is as follows:

- Before you start working on your worksheet you must **fork** a copy of the official class repo from here
    - o https://github.com/bjg/2019-tudublin-cmpu4023.git
- Then **clone** the forked repo to your development machine
- The make a new branch in your cloned, local repo named as follows:
    - o ***<student-id>***-wks-1

  where ***<student-id>*** is something like C12345678
- When you are finished developing your worksheet solution then you must push your local repo to the remote origin for that branch
- Finally, when you are submitting your solution for grading, you will generate a pull request (PR) requesting that your branch is merged with the remote origin master branch of the official class repo above
- If you are not sure about any of the described steps here, then take a look at this worked demonstration:
    - o https://www.youtube.com/watch?v=FQsBmnZvBdc

.

5. Demonstration

You may be asked to give a brief demonstration of your submission to the lab instructor in lab 3.

6. Requirements

For this lab you will need to

- Use your own laptop with local tools or,
- Sign up for a free account with a cloud provider

7. Resources

You are free to research whatever you need to solve the problems in this lab. Some recommended resources include:

- https://nodejs.org/en/
- https://www.postgresql.org/download/windows/
- https://www.postgresql.org/download/macosx/
- http://postgresguide.com/setup/example.html
- http://massive-js.readthedocs.io/en/latest/
- http://www.craigkerstiens.com/2015/11/30/massive-node-postgres-an-overview-and-intro/
- http://expressjs.com/
- http://www.unixwiz.net/techtips/sql-injection.html
- http://mherman.org/blog/2015/10/22/node-postgres-sequelize/#.WJ9-aBKLSRt
- http://docs.sequelizejs.com/en/v3/

8. Setting Up

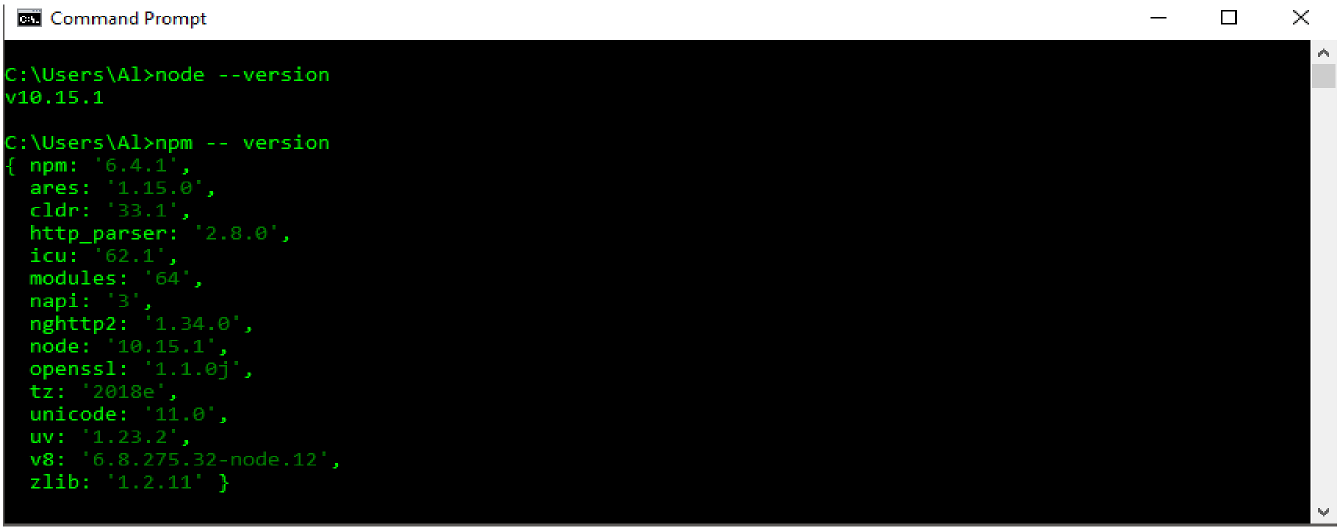
The following platform-independent tasks can be solved on Windows, Mac local Linux or Cloud Linux as you prefer

1

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

(*) This tutorial guide assumes you are using NodeJS as your server environment. However, you are free to choose whatever server environment you like but you have to do the appropriate translation of the instructions here.



```
C:\Users\Al>node --version
v10.15.1

C:\Users\Al>npm -- version
{ npm: '6.4.1',
  ares: '1.15.0',
  cldr: '33.1',
  http_parser: '2.8.0',
  icu: '62.1',
  modules: '64',
  napi: '3',
  nghttp2: '1.34.0',
  node: '10.15.1',
  openssl: '1.1.0j',
  tz: '2018e',
  unicode: '11.0',
  uv: '1.23.2',
  v8: '6.8.275.32-node.12',
  zlib: '1.2.11' }
```

2

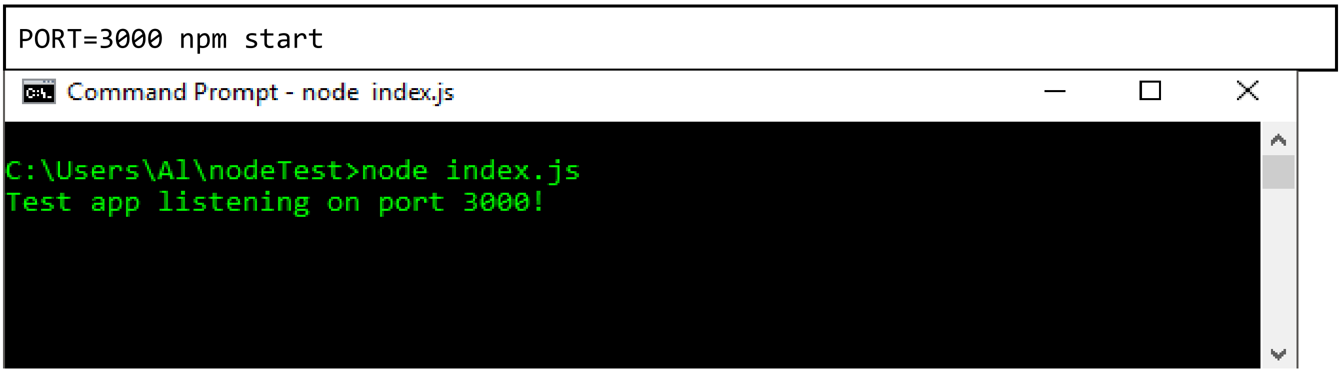
Create a new project folder, change into it and run the following

```
npm init
(entry point: index.js)
npm install express --save
```

In your folder, create an `index.js` with the contents from the **Express** tutorial example at https://expressjs.com/en/starter/hello-world.html

Run and verify that the your endpoint is responding at your designated port. For example on Mac or Linux

```
PORT=3000 npm start
```



```
C:\Users\Al\nodeTest>node index.js
Test app listening on port 3000!
```

3

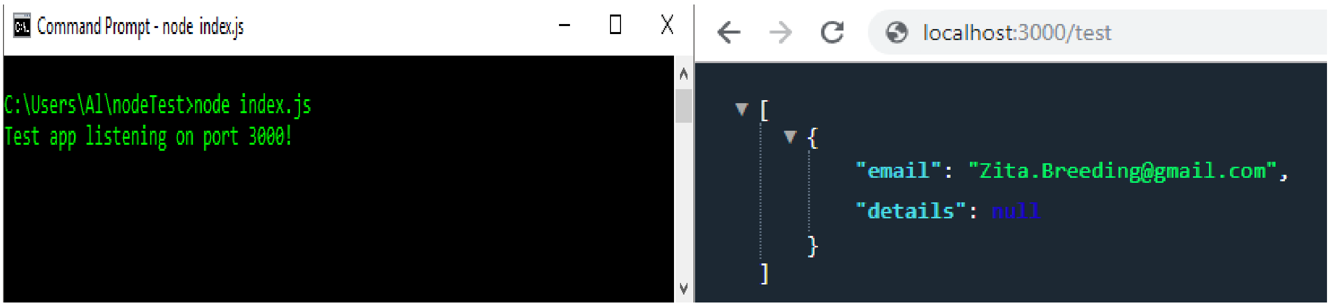
Install a recent of Postgres (*) on your laptop or sign up for a free cloud-based provider (**)

(*) This tutorial guide assumes you have used Postgres as your database. You are free to use any relational database you choose but:

- It must be relational (so no MongoDB, etc)
- An you must be prepared to translate any Postgres-specific instructions given here for your chosen database

(**) In some circumstances when using Postgres in the cloud, you may encounter college firewall problems communicating to your database instance. In this case, if available, access the Internet via a tethered phone or other 3G/4G access point

Verify that your Postgres server instance is working for your chosen target environment



```
C:\Users\Al\nodeTest>node index.js
Test app listening on port 3000!
```

```
[
  {
    "email": "Zita.Breeding@gmail.com",
    "details": null
  }
]
```

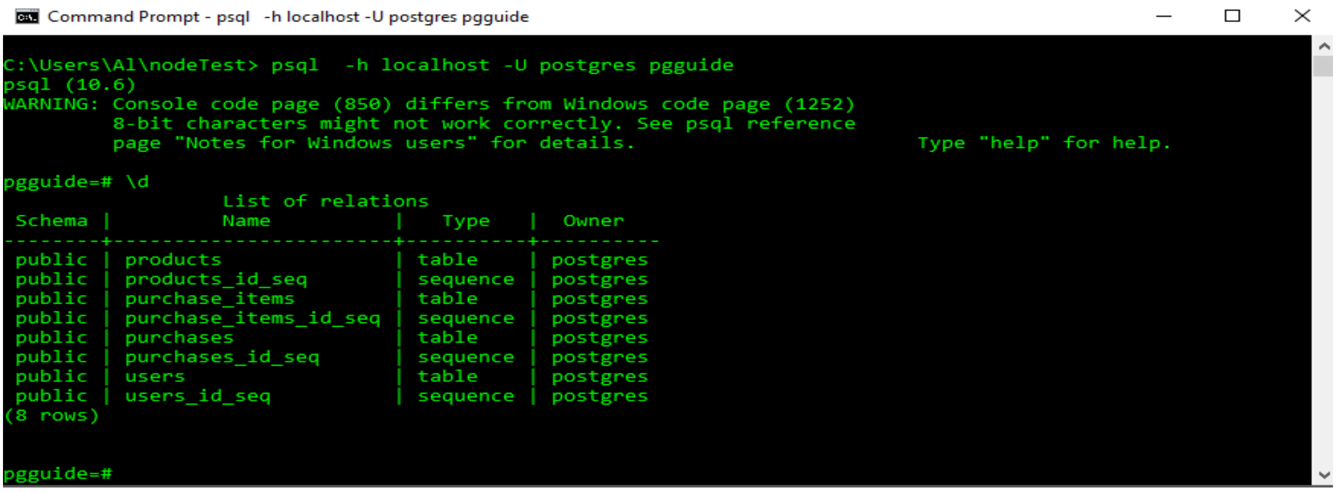
4

Load the sample database from the Postgres Guide

http://postgresguide.com/setup/example.html

Inspect the schema and table data using the psql client, e.g.

```
\d
\d products
table products;
```



```
C:\Users\Al\nodeTest> psql  -h localhost -U postgres pgguide
psql (10.6)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.                  Type "help" for help.

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=#
```

5 Install Massive JS from http://massive-js.readthedocs.io/en/latest/

Massive JS uses database reflection to create Javascript APIs to allow CRUD operations on a specified schema

Following the tutorial at https://dmfay.github.io/massive-js/,
rework the example models to access the pgguide sample database installed in step 4 above (i.e. skip creating the example user model and focus on the user, products, purchases and purchase_items tables from the step 4 sample database instead.

NOTE:
It may be necessary to add a primary key to the purchase_items table to make it play well with MassiveJS's table reflection facility as follows:

```
ALTER TABLE purchase_items ADD PRIMARY KEY (id);
```

9. Problem Sets

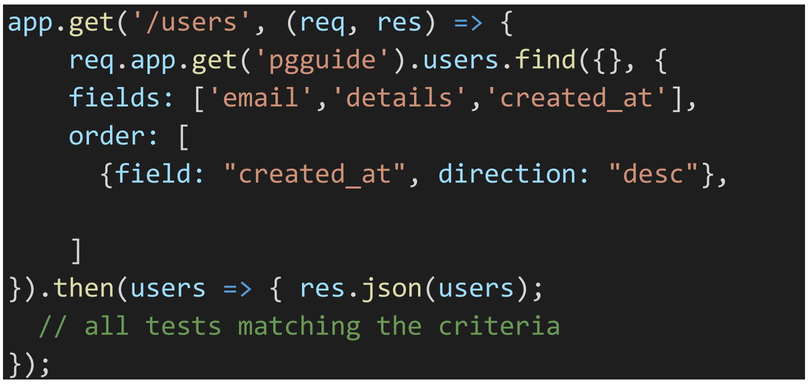
For your lab submission, take screenshots or cut-n-paste your solutions into a document or zip archive your generated solutions which you will submit through webcourses

1	Using Node, Express and Massive create the following HTTP API endpoints serving the following resources as JSON documents	20 Marks

GET /users

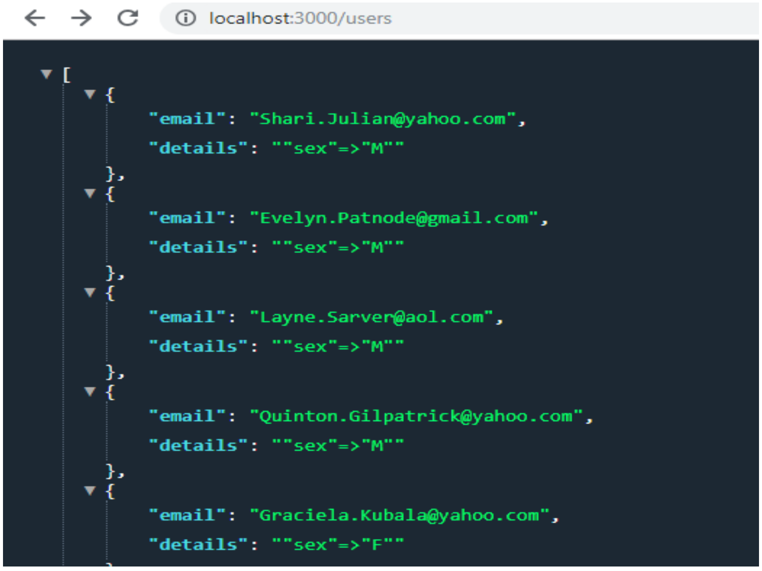
List all users email and sex in order of most recently created. Do not include password hash in your output

Query:



```
app.get('/users', (req, res) => {
    req.app.get('pgguide').users.find({}, {
    fields: ['email','details','created_at'],
    order: [
      {field: "created_at", direction: "desc"},

    ]
}).then(users => { res.json(users);
  // all tests matching the criteria
});
```

output:



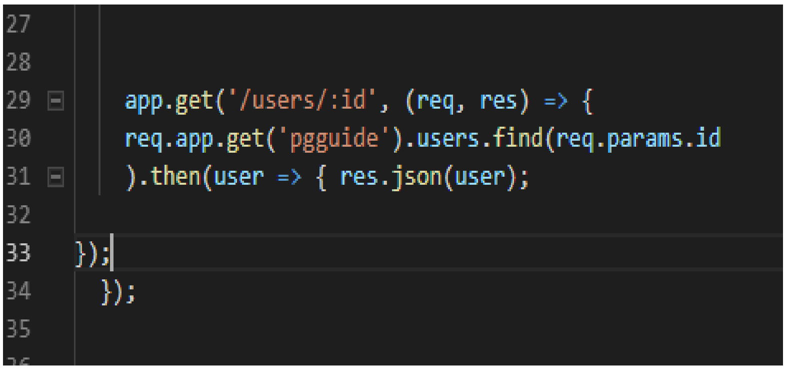
localhost:3000/users

```
▼ [
   ▼ {
         "email": "Shari.Julian@yahoo.com",
         "details": ""sex"=>"M""
     },
   ▼ {
         "email": "Evelyn.Patnode@gmail.com",
         "details": ""sex"=>"M""
     },
   ▼ {
         "email": "Layne.Sarver@aol.com",
         "details": ""sex"=>"M""
     },
   ▼ {
         "email": "Quinton.Gilpatrick@yahoo.com",
         "details": ""sex"=>"M""
     },
   ▼ {
         "email": "Graciela.Kubala@yahoo.com",
         "details": ""sex"=>"F""
```

GET /users/:id

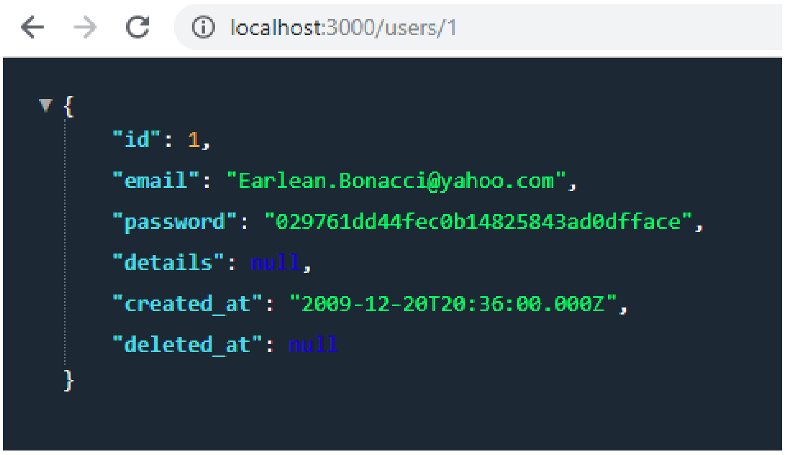
Show above details of the specified user

Query:



```


    app.get('/users/:id', (req, res) => {
    req.app.get('pgguide').users.find(req.params.id
    ).then(user => { res.json(user);

});
  });

```

outPut:



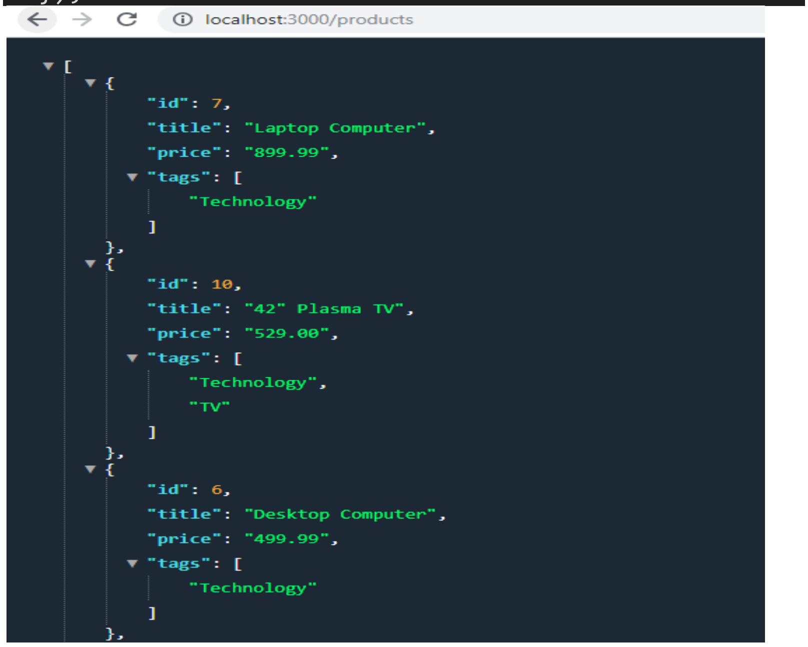
```
localhost:3000/users/1

▼ {
      "id": 1,
      "email": "Earlean.Bonacci@yahoo.com",
      "password": "029761dd44fec0b14825843ad0dfface",
      "details": null,
      "created_at": "2009-12-20T20:36:00.000Z",
      "deleted_at": null
  }
```

```
GET
/products
Query:
```

List all products in ascending order of price

```
app.get('/products', (req, res) => {
    req.app.get('pgguide').products.find({}, {
      fields: ['id','title','price','tags'],
       order: [
        {field: "price", direction: "desc"},

         ]
    }).then(user => { res.json(user);
  // all tests matching the criteria
});
  });
```

```
outPut:
```

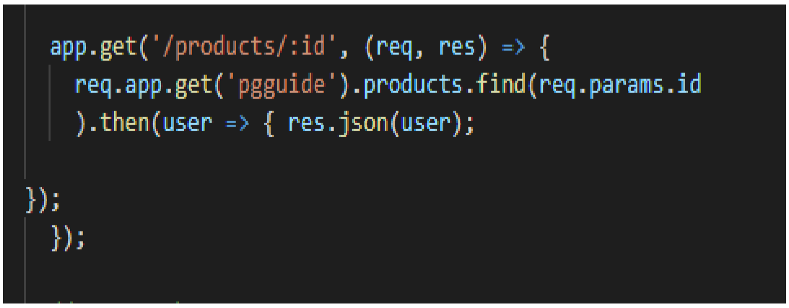


```
localhost:3000/products

▼ [
  ▼ {
        "id": 7,
        "title": "Laptop Computer",
        "price": "899.99",
      ▼ "tags": [
            "Technology"
        ]
    },
  ▼ {
        "id": 10,
        "title": "42" Plasma TV",
        "price": "529.00",
      ▼ "tags": [
            "Technology",
            "TV"
        ]
    },
  ▼ {
        "id": 6,
        "title": "Desktop Computer",
        "price": "499.99",
      ▼ "tags": [
            "Technology"
        ]
    },
```

GET /products/:id

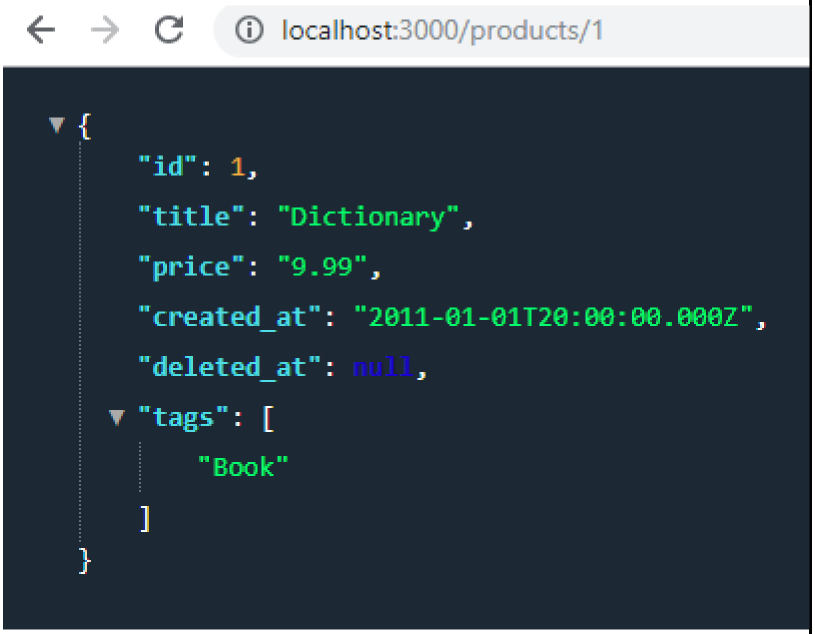
Show details of the specified products

Query:



```
app.get('/products/:id', (req, res) => {
  req.app.get('pgguide').products.find(req.params.id
  ).then(user => { res.json(user);

});
 });
```

output:



localhost:3000/products/1

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

GET /purchases

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

Query:

```
app.get('/purchases', (req, res) => {
  req.app.get('pgguide').query(
 '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 = user_id \
  inner join purchase_items on
purchase_items.purchase_id = purchase_id inner
join\
  products on purchase_items.product_id =
product_id'

).then(user => { res.json(user);
  // all tests matching the criteria
});
  });
```

Test each of these endpoints serves the expected data and briefly show how you did this

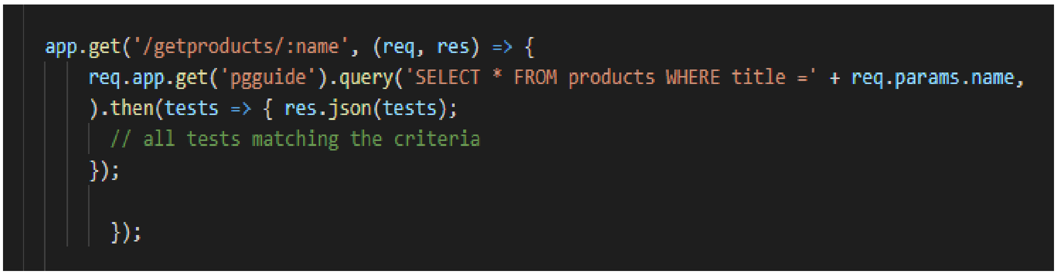
2 Building on your solution to part 1 for the API to the `products` resource from the `pgguide` database, extend the product indexing endpoint to allow the filtering of products by name as follows

20 Marks

```
GET /products[?name=string]
```

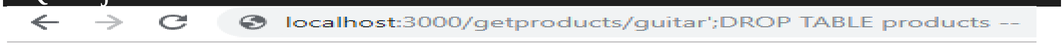
For your solution you should implement the query (badly) in such a way as to allow an attacker to inject arbitrary SQL code into the query execution. Show, using your badly implemented approach, how an attacker can craft a query string to allow the deletion of a product from the products table.

For convenience, you can continue to use MassiveJS to interface with the database.



```
app.get('/getproducts/:name', (req, res) => {
    req.app.get('pgguide').query('SELECT * FROM products WHERE title =' + req.params.name,
    ).then(tests => { res.json(tests);
      // all tests matching the criteria
    });

      });
```

SQL Injection



localhost:3000/getproducts/guitar';DROP TABLE products --

3 Provide two solutions to eliminate the security hole in your approach from the previous section as follows:

15 Marks

- Using a parameterised query
- Using a stored procedure using SQL or PLPGSQL whichever you prefer

Explicitly show that the injection attack is not now possible for each of your solutions

Again, you can just use MassiveJS as your database interface library here too.

<table>
<tr>
<td></td>
<td>

```
  app.get('/getproducts/:name', (req, res) => {
    req.app.get('pgguide').products.find(req.params.name
    ).then(user => { res.json(user);

});
  });
```

</td>
<td></td>
</tr>
<tr>
<td><b>4</b></td>
<td>

Create a brand new Express project using the Sequelize ORM. Install and configure Sequelize and wire it up to the `pgguide` database.. Verify that you have basic connectivity before proceeding.

Create Sequalize migrations for the `pgguide` sample database

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

</td>
<td>15 Marks</td>
</tr>
<tr>
<td><b>5</b></td>
<td>

Use your models and Javascript code to populate the database with some additional test data for all of the models above

```
router.post('/create_test_data',jsonParser,(req,res,next)=>{
    console.log('calling post product')
   // req.setEncoding('UTF-16')
    console.log(req.body)

products.create({

    id:req.body.id,
    title:req.body.title,
    price: req.body.price,
    tags: req.body.tags
,
  })

  .then(product =>{
      res.send({
          status:200,
          product
      })
  })
  .catch(err => console.log(err))

});
```

</td>
<td>10 Marks</td>
</tr>
<tr>
<td><b>6</b></td>
<td>

Reimplement the RESTful API using Sequelize and Express for your system. Your API should support the following CRUD operations as follows, returning JSON responses

</td>
<td>20 Marks</td>
</tr>
</table>

<table>
<tr>
<td>

GET /products[?name=string]

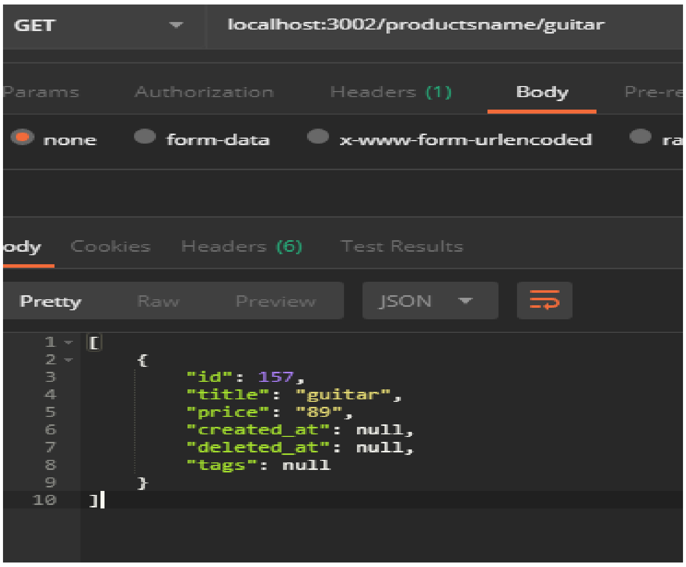
Query:

output:

</td>
<td>

List all products

```
router.get('/productsname/:title', function
(req, res) {
    products.findAll({

        where:
        {
            title: req.params.title
        }

    }).then(products => res.json(products))
  });
```



```
[
    {
        "id": 157,
        "title": "guitar",
        "price": "89",
        "created_at": null,
        "deleted_at": null,
        "tags": null
    }
]
```

</td>
</tr>
<tr>
<td>

GET /products/:id

Query:

</td>
<td>

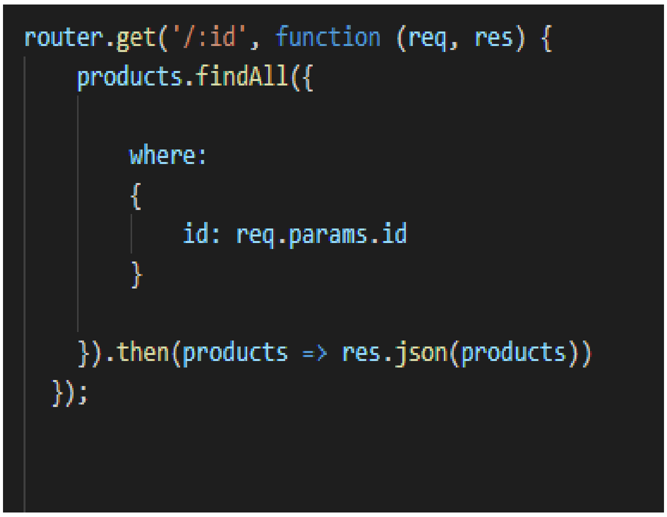
Show details of the specified products

</td>
</tr>
</table>

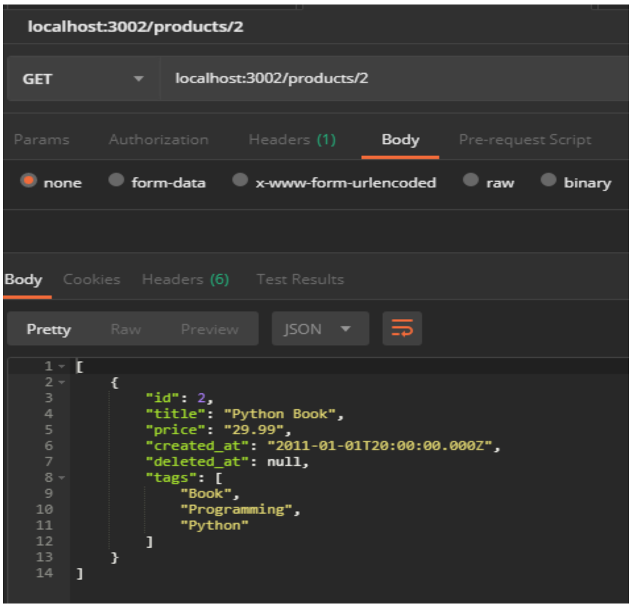
<table>
<tr>
<td>

output:

</td>
<td>



```
router.get('/:id', function (req, res) {
    products.findAll({

        where:
        {
            id: req.params.id
        }

    }).then(products => res.json(products))
  });
```



localhost:3002/products/2

GET localhost:3002/products/2

```
[
    {
        "id": 2,
        "title": "Python Book",
        "price": "29.99",
        "created_at": "2011-01-01T20:00:00.000Z",
        "deleted_at": null,
        "tags": [
            "Book",
            "Programming",
            "Python"
        ]
    }
]
```

</td>
</tr>
<tr>
<td>

`POST /products`

</td>
<td>

Create a new product instance

</td>
</tr>
<tr>
<td>

`PUT /products/:id`

Query:

</td>
<td>

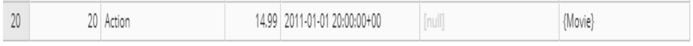
Update an existing product

```
router.put('/updatetitle/:id',jsonParser,(req,
res,next)=>{
    console.log('calling post product')
   // req.setEncoding('UTF-16')
    console.log(req.body)

products.update({


    title:req.body.title
```

</td>
</tr>
</table>

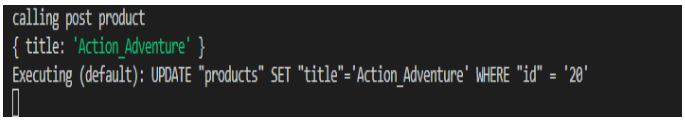
```
  },{

      where: {
                id: req.params.id
              }
    }
  )

  .then(product =>{
      res.send({
          status:200,
          product
      })
  })
  .catch(err => console.log(err))

});
```

Db before:



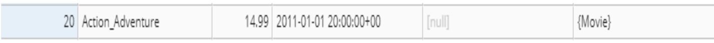
```
20    20 Action    14.99 2011-01-01 20:00:00+00    [null]    {Movie}
```

Output:



```
calling post product
{ title: 'Action_Adventure' }
Executing (default): UPDATE "products" SET "title"='Action_Adventure' WHERE "id" = '20'
```

DB after:



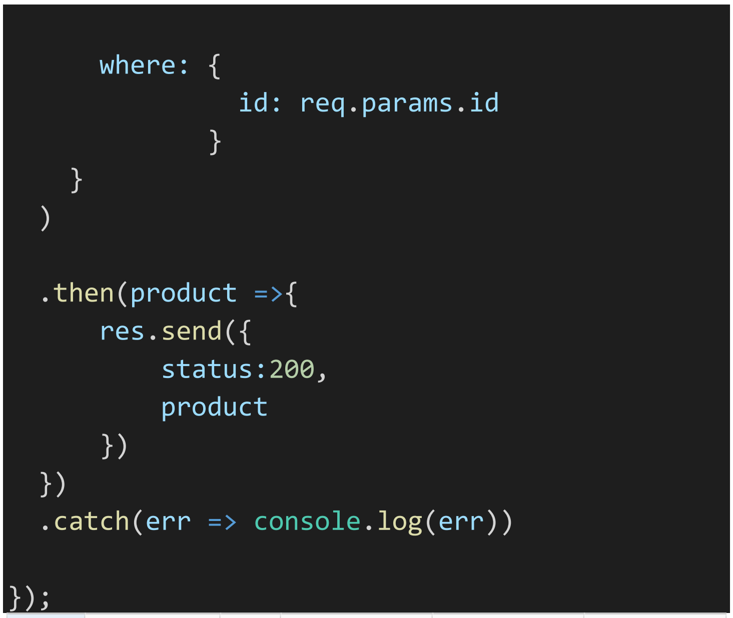
```
   20 Action_Adventure    14.99 2011-01-01 20:00:00+00    [null]    {Movie}
```

`DELETE /products/:id`

Remove an existing product

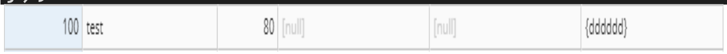
Query:

```
router.delete('/deleteproduct/:id',jsonParser,
(req,res,next)=>{

    console.log(req.body)

products.destroy({
```

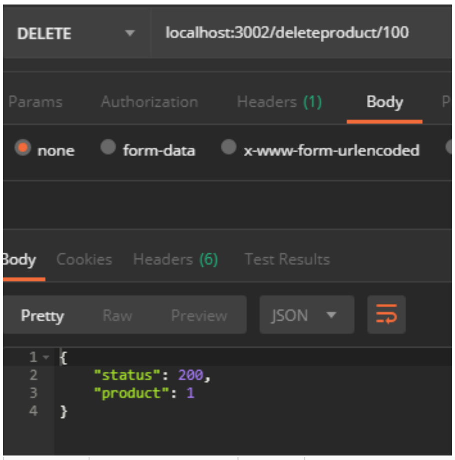


```
      where: {
                id: req.params.id
              }
    }
  )

  .then(product =>{
      res.send({
          status:200,
          product
      })
  })
  .catch(err => console.log(err))

});
```

Db before:



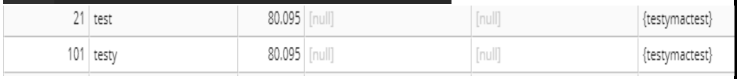
100	test	80	[null]	[null]	{dddddd}



localhost:3002/deleteproduct/100

```
{
    "status": 200,
    "product": 1
}
```

Db after



21	test	80.095	[null]	[null]	{testymactest}
101	testy	80.095	[null]	[null]	{testymactest}

Show test cases for each of the API endpoint REST operations