

Thomas Delaney C15300756 - Enterprise Application Development Worksheet 1

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

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

```
app.get('/users', (req, res) => {
  db.users.find({}, {
    fields: [
      "email",
      "details::json"
    ],
    order: [
      {
        field: 'created_at',
        direction: 'desc',
      }
    ]
  }).then(items => {
    let info = [];

    items.forEach(function(entry) {
      info.push({
        "email": entry['email'],
        "sex": entry['details'] && entry['details']['sex'] ? entry['details']['sex'] : "Not Disclosed"
      });
    });

    res.json(info);
  });
});
```

The route will find all the users email and details, in a JSON format so we can parse the sex out later. When the items are returned we create an empty array to store the output, and loop over the items. We use an inline if statement to check if details is and sex are available, if so then we push an object made up of the email and the sex, or "Not Disclosed" of details and or sex is not available

When requested from the browser, the output was as follows.

```
[{"email": "test2@gmail.com", "sex": "F"}, {"email": "test1@gmail.com", "sex": "M"},
{"email": "Shari.Julian@yahoo.com", "sex": "M"}, {"email": "Evelyn.Patnode@gmail.com", "sex": "M"},
{"email": "Layne.Sarver@aol.com", "sex": "M"}, {"email": "Quinton.Gilpatrick@yahoo.com", "sex": "M"},
{"email": "Graciela.Kubala@yahoo.com", "sex": "F"}, {"email": "Derek.Knittel@gmail.com", "sex": "F"},
{"email": "Theresia.Edwin@yahoo.com", "sex": "M"}, {"email": "Williams.Upson@gmail.com", "sex": "F"},
{"email": "Glen.Lanphear@yahoo.com", "sex": "Not Discolسد"},
{"email": "Ozella.Yoshimura@gmail.com", "sex": "M"},
{"email": "Samatha.Hedgpeth@yahoo.com", "sex": "F"},
{"email": "Isabel.Breeding@gmail.com", "sex": "Not Discolسد"},
{"email": "Kali.Damore@yahoo.com", "sex": "F"}, {"email": "Gudrun.Arends@gmail.com", "sex": "Not Discolسد"},
{"email": "Ivana.Kurth@yahoo.com", "sex": "Not Discolسد"},
{"email": "Kimi.Mcqueeney@gmail.com", "sex": "Not Discolسد"},
{"email": "Stacia.Schrack@aol.com", "sex": "M"}, {"email": "Claud.Westmoreland@aol.com", "sex": "Not Discolسد"},
{"email": "Eleanor.Patnode@yahoo.com", "sex": "F"},
{"email": "Sherilyn.Hamill@gmail.com", "sex": "M"}, {"email": "Russ.Mcclain@yahoo.com", "sex": "F"},
{"email": "Tonette.Alba@gmail.com", "sex": "Not Discolسد"},
{"email": "Earlean.Bonacci@yahoo.com", "sex": "Not Discolسد"},
{"email": "Shanell.Maxson@gmail.com", "sex": "M"}, {"email": "Ozella.Roles@gmail.com", "sex": "Not Discolسد"},
{"email": "Wan.Dilks@gmail.com", "sex": "M"},
{"email": "Vivian.Westmoreland@yahoo.com", "sex": "F"},
{"email": "Humberto.Jonson@yahoo.com", "sex": "Not Discolسد"},
{"email": "Salvatore.Arends@aol.com", "sex": "F"}, {"email": "Angel.Lessley@yahoo.com", "sex": "F"},
{"email": "Eve.Kump@yahoo.com", "sex": "M"}, {"email": "Mauro.Pung@yahoo.com", "sex": "F"},
{"email": "Claud.Cousineau@gmail.com", "sex": "F"}, {"email": "Zita.Breeding@gmail.com", "sex": "Not Discolسد"},
{"email": "Granville.Hedgpeth@gmail.com", "sex": "Not Discolسد"},
{"email": "Harrison.Puett@yahoo.com", "sex": "M"},
{"email": "Cortney.Strayer@gmail.com", "sex": "M"}, {"email": "Edmund.Roles@yahoo.com", "sex": "F"},
{"email": "Carmel.Bulfer@aol.com", "sex": "F"}, {"email": "Wynona.Greening@aol.com", "sex": "M"},
{"email": "Shanell.Lichtenstein@aol.com", "sex": "M"},
{"email": "Danny.Crays@gmail.com", "sex": "M"}, {"email": "Samatha.Pellegrin@yahoo.com", "sex": "Not Discolسد"},
{"email": "Jeremiah.Buonocore@yahoo.com", "sex": "Not Discolسد"},
{"email": "Derek.Crenshaw@gmail.com", "sex": "F"},
{"email": "Takako.Gilpatrick@aol.com", "sex": "M"}, {"email": "Zita.Luman@yahoo.com", "sex": "Not Discolسد"},
{"email": "Cherryl.Tarnowski@gmail.com", "sex": "Not Discolسد"},
{"email": "Ivana.Sosnowski@aol.com", "sex": "M"}, {"email": "Romaine.Birdsell@aol.com", "sex": "F"}]
```

- Show above details of the specified user

```
app.get('/users/:id', (req, res) => {
  db.users.findOne({
    id: req.params.id
  }, {
    fields: [
      "email",
      "details::json"
    ]
  }).then(entry => {

    let user = {
      "email": entry['email'],
      "sex": entry['details'] && entry['details']['sex'] ? entry['details']['sex'] : "Not Discolسد"
    };

    res.json(user);
  });
});
```

This is similar to the previous solution; the difference is we take the id as parameter for the URL which is accessed using req.params.id. We then include this id in the database request which will only return data for the user with that id. Output is as follows:

```
[{"email": "Earlean.Bonacci@yahoo.com", "sex": "Not Disclosed"}]
```

- List all products in ascending order of price

```
app.get('/products', (req, res) => {  
  if (req.query.name == undefined) {  
    db.products.find({}, {  
      order: [  
        {  
          field: 'price',  
          direction: 'asc',  
        }  
      ]  
    }).then(items => {  
      res.json(items);  
    });  
  }  
})
```

The "if" part of this code will be used for part 2 of the worksheet. However the query here will find all the products, and then order them by the price field, in ascending order/direction. Output looks like:

```
[{"id":5,"title":"Coloring Book","price":"5.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Book","Children"]}, {"id":22,"title":"Final Year Project","price":"6.00","created_at":"2019-02-13T15:30:00.000Z","deleted_at":null,"tags":["Movie","Tragedy"]}, {"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"]}, {"id":11,"title":"Classical CD","price":"9.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Music"]}, {"id":12,"title":"Holiday CD","price":"9.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Music"]}, {"id":13,"title":"Country CD","price":"9.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Music"]}, {"id":14,"title":"Pop CD","price":"9.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Music"]}, {"id":15,"title":"Electronic CD","price":"9.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Music"]}, {"id":21,"title":"Test Movie","price":"10.50","created_at":"2019-02-13T19:50:00.000Z","deleted_at":null,"tags":["Movie","Comedy"]}, {"id":24,"title":"test_product3","price":"12.00","created_at":"2019-02-13T20:00:00.000Z","deleted_at":null,"tags":["Movie","Comedy"]}, {"id":23,"title":"test_product","price":"12.00","created_at":null,"deleted_at":null,"tags":["Movie","Comedy"]}, {"id":16,"title":"Comedy Movie","price":"14.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Movie","Comedy"]}, {"id":18,"title":"Romantic","price":"14.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Movie"]}, {"id":17,"title":"Documentary","price":"14.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Movie"]}, {"id":20,"title":"Action","price":"14.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Movie"]}, {"id":19,"title":"Drama","price":"14.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Movie"]}, {"id":3,"title":"Ruby Book","price":"27.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Book","Programming","Ruby"]}, {"id":2,"title":"Python Book","price":"29.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Book","Programming","Python"]}, {"id":8,"title":"MP3 Player","price":"108.00","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Technology","Music"]}, {"id":9,"title":"42\" LCD TV","price":"499.00","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Technology","TV"]}, {"id":6,"title":"Desktop Computer","price":"499.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Technology"]}, {"id":10,"title":"42\" Plasma TV","price":"529.00","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Technology","TV"]}, {"id":7,"title":"Laptop Computer","price":"899.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Technology"]}]
```

- Show details of the specified products

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

This is similar to get users using id solution; the parameter is taken in via the URL and used to query the product. Output is as follows:

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

- 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

```
app.get('/purchases', (req, res) => {
  db.query(
    "select name, address, email, price, quantity FROM purchases P JOIN users U"+
    "on (P.user_id = U.id) join purchase_items on (purchase_id = P.id) order by price desc;"
  ).then(items => {
    res.json(items);
  });
});
```

This query needed to be done using a raw query, as massive.js querying functions does not support joins. This query will select all fields required and will join purchases to purchase items, ordering the fields by price in descending order. Output is as follows:

```
[{"name": "Kourtney Brazell", "address": "5980 43rd
St.", "email": "Zita.Luman@yahoo.com", "price": "899.99", "quantity": 1}, {"name": "Missy
Galvez", "address": "2773 MLK
Ave.", "email": "Ozella.Roles@gmail.com", "price": "899.99", "quantity": 1}, {"name": "Kelli
Pung", "address": "5262 45th
St.", "email": "Carmel.Bulfer@aol.com", "price": "899.99", "quantity": 1}, {"name": "Shari
Mcdougale", "address": "6125 Washington
Ave.", "email": "Evelyn.Patnode@gmail.com", "price": "899.99", "quantity": 1}, {"name": "Gaye
Lanser", "address": "9768 50th
Ave.", "email": "Ivana.Kurth@yahoo.com", "price": "899.99", "quantity": 1}, {"name": "Reed
Arends", "address": "4083 10th
Ave.", "email": "Russ.Mcclain@yahoo.com", "price": "899.99", "quantity": 2}, {"name": "Fidelia
Dossey", "address": "2842 43rd
St.", "email": "Zita.Breeding@gmail.com", "price": "899.99", "quantity": 3}, {"name": "Pauletta
Verner", "address": "8926 44th
Ave.", "email": "Mauro.Pung@yahoo.com", "price": "899.99", "quantity": 1}, {"name": "Angel
Lesane", "address": "2318 MLK
Ave.", "email": "Kali.Damore@yahoo.com", "price": "899.99", "quantity": 1}, {"name": "Graig
Blatt", "address": "2872 43rd
St.", "email": "Zita.Luman@yahoo.com", "price": "899.99", "quantity": 1}, {"name": "Gaylene
Sandoval", "address": "5621 35th St.42nd
Ave.", "email": "Claud.Westmoreland@aol.com", "price": "899.99", "quantity": 1}, {"name": "Beverlee
Mcdougale", "address": "5912 44th
Ave.", "email": "Ivana.Kurth@yahoo.com", "price": "899.99", "quantity": 1}, {"name": "Divina
Hamill", "address": "2103 50th
Ave.", "email": "Gudrun.Arends@gmail.com", "price": "899.99", "quantity": 1}, {"name": "Alfonzo
Haubrich", "address": "1955 43rd
St.", "email": "Evelyn.Patnode@gmail.com", "price": "899.99", "quantity": 2}, {"name": "Reed
Larimer", "address": "4937 Washington
Ave.", "email": "Graciela.Kubala@yahoo.com", "price": "899.99", "quantity": 1}, {"name": "Alfonzo
Bodkin", "address": "8330 10th
Ave.", "email": "Zita.Luman@yahoo.com", "price": "899.99", "quantity": 4}, {"name": "Jami
Parrilla", "address": "9366 MLK
```

2. SQL Injection

```
app.get('/products', (req, res) => {
  if (req.query.name == undefined) {
    db.products.find({}, {
      order: [
        {
          field: 'price',
          direction: 'asc',
        }
      ]
    }).then(items => {
      res.json(items);
    });
  }
  else {
    //unsafe query
    const q2 = "select * from products where title = '" + req.query.name + "'";

    db.query(
      //SQL injection to delete product row

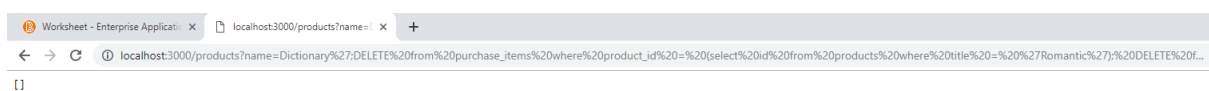
      //Dictionary;DELETE from purchase_items where product_id =
      //(select id from products where title = 'Coloring Book'); DELETE from products where title = 'Coloring Book';--
      q2
    ).then(items => {
      res.json(items);
    });
  }
});
```

This is the full query based on the previous query to get purchases. Using `req.query.name` we can get the query name parameter. If it is included then we go into the else bracket of this code snippet. "q2" is an unsafe query, it just concatenates the parameter into the query then executes it. This allows the injection query (commented out in the snippet) to be added into the query.

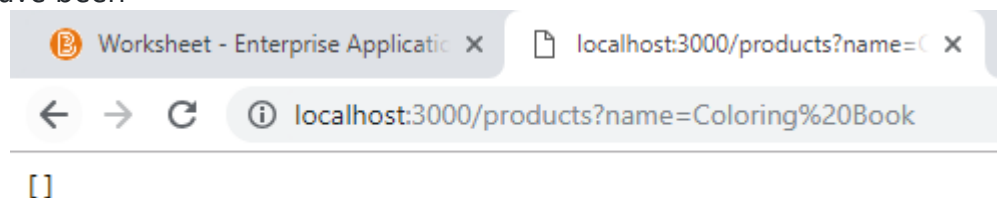
Output before injection:

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

Coloring Book exists, but when we include the injection query:



When Colouring Book is now queried, it is empty, like so, it, and all its purchase items, have been



deleted.

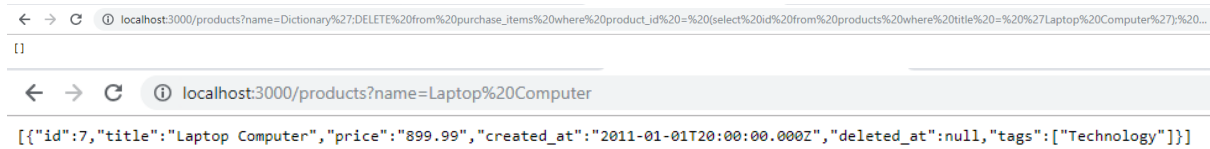
3. Two solutions to eliminate the security hole

- Parameterized Query

```
const q = "select * from products where title = $1";

db.query(
  q, [req.query.name]
).then(items => {
  res.json(items);
});
```

The parameterized query looks like this, the `$1` relates to the first element in the passed in array, which includes the parameter name. This provides extra protection against SQL injections as it includes its own use of quotes to avoid injection of quotes. When the query is ran with the same injection query (using Laptop Computer as the title), the row still remains, even with the same output as when executed with an unsafe query.



```
localhost:3000/products?name=Laptop%20Computer
[{"id":7,"title":"Laptop Computer","price":"899.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Technology"]}]
```


The stored procedure is a function that is declared as "LANGUAGE SQL STRICT IMMUTABLE" which means that no update or delete operations can occur. The function will look like this:

```
CREATE OR REPLACE FUNCTION getProductByTitle(_title TEXT) RETURNS SETOF products AS $$
SELECT * from products where title = _title;
$$ LANGUAGE SQL STRICT IMMUTABLE;
```

It is executed in the JavaScript like so:

```
db.getProductByTitle(req.query.name).then(items => {
  res.json(items);
}).catch(error => {
  console.log(error);
});
```

And the output is as follows:



```
localhost:3000/products?name=Laptop%20Computer
[{"id":7,"title":"Laptop Computer","price":"899.99","created_at":"2011-01-01T20:00:00.000Z","deleted_at":null,"tags":["Technology"]}]
```

4 and 5. Sequelized Models


```

'use strict';
module.exports = function(sequelize, DataTypes) {
  var purchase_item = sequelize.define('purchase_item', {
    price: DataTypes.FLOAT,
    quantity: DataTypes.INTEGER,
    state: DataTypes.STRING,
    purchase_id: DataTypes.INTEGER,
    product_id: DataTypes.INTEGER,
  }, {
    paranoid: false,
    updatedAt: false,
    underscored: true
  }, {
    classMethods: {
      associate: function(models) {
        purchase_item.belongsTo(models.purchase);
        purchase_item.belongsTo(models.product);
      }
    }
  });
  return purchase_item;
};

```

All the models are somewhat built like this, this is the purchase_item model, **paranoid: false** ensures that deleted_at is not included, however it would be set to true for models that include this (user, product). **updatedAt: false** ensures that updatedAt is not included in any queries on the model. **underscored: true** allows for attributes with underscores. **purchase_item.belongsTo(models.purchase);** and **purchase_item.belongsTo(models.product);** associations tell the model that it has references to products and purchases. In the respective purchase table, it would include the association **purchase.hasMany(models.purchase_item);**.

6. API using Sequelize.js

- List all products


```
// get all products
router.get('/products', function(req, res) {
  if (req.query.name == undefined){
    models.product.findAll({}).then(function(products) {
      res.json(products);
    });
  }
  else{
    models.product.find({
      where: {
        title: req.query.name
      }
    }).then(function(products) {
      res.json(products);
    });
  }
});
```

If name is defined then we use models.products.find and a where clause to query the database using the parameter name. Output is:

← → ↻ ⓘ localhost:5000/products?name=Dictionary

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

- Show details of the specified products

```
router.get('/products/:id', function(req, res) {
  models.product.find({
    where: {
      id: req.params.id
    }
  }).then(function(products) {
    res.json(products);
  });
});
```

Where clause and parameter id used to query the specified product, result is:

← → ↻ ⓘ localhost:5000/products/1

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

- Create a new product instance

```
router.post('/products', function(req, res) {
  models.product.create({
    title: req.body.title,
    price: req.body.price,
    created_at: req.body.created_at,
    deleted_at: null,
    tags: req.body.tags
  }).then(function(product) {
    res.json(product);
  });
});
```

The router post function lets us post a product instance, the models.product.create will use the sequelized product model to create an instance and add it to the database. The CURL command and result is as follows:

```
PS G:\4th Year\Semester 2\Enterprise Application Development\Worksheet 1\Parts 4-6\node-postgres-sequelize>
curl -d "title=test_product5&price=12.00&created_at=2019-02-13T20:00:00.000Z&tags={\"Movie\", \"Comedy\"}" http://127.0.0.1:5000/products
{"id":25,"title":"test_product5","price":"12.00","created_at":"2019-02-13T20:00:00.000Z","deleted_at":null,"tags":["Movie","Comedy"]}
```

And when queried using the ID we can see that the instance has indeed been created:

```
localhost:5000/products/25
{"id":25,"title":"test_product5","price":"12.00","tags":["Movie","Comedy"],"deleted_at":null,"created_at":"2019-02-13T20:00:00.000Z"}
```

- Update an existing product

```
router.put('/products/:id', function(req, res) {
  models.product.find({
    where: {
      id: req.params.id
    }
  }).then(function(product) {
    if(product){
      product.updateAttributes({
        title: req.body.title,
        price: req.body.price,
        created_at: req.body.created_at,
        deleted_at: req.body.deleted_at,
        tags: req.body.tags
      }).then(function(product) {
        res.send(product);
      });
    }
  });
});
```

To update a product, you select a product then use the updateAttributes method to update the rows attributes with the PUT request body data. CURL command looks like (updating test_product5 to

test_product5_updated):

```
PS G:\4th Year\Semester 2\Enterprise Application Development\Worksheet 1\Parts 4-6\node-postgres-sequelize> curl
-X PUT --data "title=test_product5_updated" http://127.0.0.1:5000/products/25
{"id":25,"title":"test_product5_updated","deleted_at":null,"created_at":"2019-02-13T20:00:00.000Z"}
```

```
localhost:5000/products/25
{"id":25,"title":"test_product5_updated","price":"12.00","tags":["Movie","Comedy"],"deleted_at":null,"created_at":"2019-02-13T20:00:00.000Z"}
```

- Delete an existing product

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

  models.purchase_item.destroy({
    where: {
      id: req.params.id
    }
  }).then(function(product) {
    res.json(product);
  }).catch(function(error) {
    res.json(error['message']);
  });

  models.product.destroy({
    where: {
      id: req.params.id
    }
  }).then(function(product) {
    res.json(product);
  }).catch(function(error) {
    res.json(error['message']);
  });
});
```

To delete a product, you select a product then use the destroy method to delete the row. Since the product 's id is a foreign key in the purchase_items table, we need to delete all rows where the id is used in purchase_items.

Using the ID 20 (Action), the result is as follows:

```
PS G:\4th Year\Semester 2\Enterprise Application Development\Worksheet 1\Parts 4-6\node-postgres-sequelize>
curl -X DELETE http://127.0.0.1:5000/products/20
```

```
localhost:5000/products/20
null
```