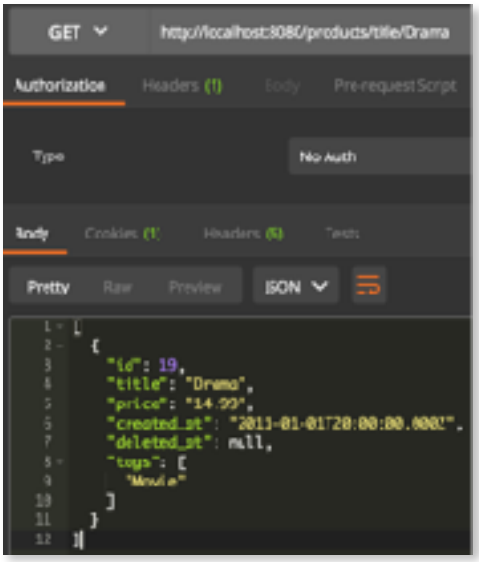
Enterprise Application Development
Lab 2

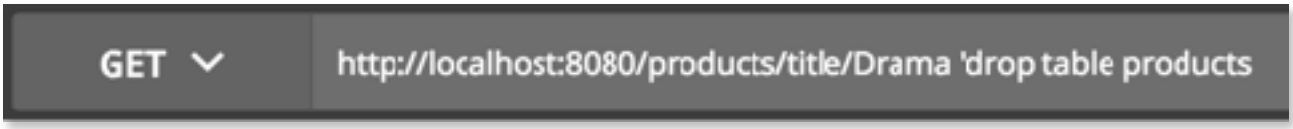
Jonathan Riordan
C13432152

Part 1.

Filter products by name



Drop a table in the url. The products table will be dropped.



Following the drop of the table, the url will change to

Part 2

Parameterised query, prevent sql injection.

```
// prduct names safe paramertised query
app.get('/products/secure1/:title', function(req, res) {
    var title = req.params.title;

    db.run("Select * from products where title = $1",[title],function (err, result) {
        res.send(result);
    });


});
```

Function to find product by title.

```
pgguide=# create or replace function find_procucts(t character varying)
returns setof products
AS
$$
    select * from products where title = t;
$$
language sql;
CREATE FUNCTION
pgguide=#
```

Part 3.

First, we create a database to hold our information. I created a database with psql called “court”

```
jonathan:~ Jonathan$ createdb court
jonathan:~ Jonathan$ psql --dbname court
psql (9.6.2)
Type "help" for help.

court=#
```

I ran the curl commands to create models judge, courtroom, participants and case. With each model, depending on the attribute, I added validation.

Part 4. - Test Data
Examples oh how I included test data was as follows, when the user enters in “insert” at the end of the url, the following Javascript code will be executed.

```
// Courtroom data
models.CourtRoom.create({
    id: 1,
    number: 201
}),

models.CourtRoom.create({
    id: 2,
    number: 300
}),
```

This is similar for the other models as well. This is how I inserted test data into my database.

Part 5 - CRUD

The following are examples of the curl commands I used to do the CRUD operations.

Insert:
curl --data "id=4&name=Adam Bari&room=9&ext=ext5" http://127.0.0.1:3000/judge

Update:
curl -X PUT --data "result=false" http://127.0.0.1:3000/case/10

Delete:
curl -X DELETE http://127.0.0.1:3000/participant/6

Retrieve:
To retrieve a single record, I would get the id for either the judge, courtroom, participant or case and then return the data in son format. The code below is similar for all the models.

```
// Get single record
router.get('/judge/:id', function(req, res) {
  models.Judge.find({
    where: {
      id: req.params.id
    }
  }).then(function(judge) {
    res.json(judge);
  });
});
```

Part 6 - Validation for Courtroom so it cant be doubled booked on a certain date.
I got the courtroom id and date entered in by the user. Fro this, I performed a where ORM query to retrieve all the data that has the same courtroom id and date. If the results returned were greater than 0, this means the courtroom has been booked for that date. As a result, an error message will be displayed informing the user that the courtroom is already booked for that date. Else, if the counter returned from the ORM where query is equal to 0, then the data is inserted into the database as the courtroom is free.

```
models.Case.findAll({
  where: {
    courtroom_id: c_id,
    start_date: date
  }
}).then(function(cases){
    if(cases.length > 0) {
        //res.json(cases);
        res.send("Error, room is already booked on this date.\n");
        return;
    } else {

            models.Case.create({
            judge_id: req.body.judge_id,
            courtroom_id: req.body.courtroom_id,
            claimant_id: req.body.claimant_id,
            respondent_id: req.body.respondent_id,
            start_date: req.body.start_date,
            duration: req.body.duration,
            result: req.body.result
        })

    }
})
.then(function(part) {
    res.json(part);
});
```