Enterprise Application Development

Lab 1

# Part 1.

//gets all users  
app.get('/users', (req, res) => {  
  
 **let** query = 'SELECT id, email, details FROM users ORDER BY created\_at DESC';  
  
 req.app.get('db').query(query).then(users => {  
  
 res.json(users)});  
});  
  
  
//gets users with matching id number  
app.get('/users/:id', (req, res) => {  
  
 **let** id = req.params.id;  
 **let** query = "SELECT id, email, details FROM users WHERE id = '" + id + "'";  
  
 req.app.get('db').query(query).then(users => {  
  
 res.json(users)});  
});

//gets all products  
//Commented out so the other query can run  
/\*  
app.get('/products', (req, res) => {  
  
 let query = "SELECT \* from products";  
  
 req.app.get('db').query(query, [req.query.name]).then(products => {  
  
 res.json(products);  
 });  
  
});  
\*/

app.get('/products/:id', (req, res) => {  
  
 **let** query = "select \* FROM products WHERE id= '" + req.params.id + "'" + " ORDER BY price ASC";  
  
 req.app.get('db').query(query).then(products => {  
  
 res.json(products);  
 });  
});  
  
  
app.get('/purchases', (req, res) => {  
  
 query = "SElECT purchase\_items.price, purchase\_items.quantity, purchase\_items.state, purchases.name AS RECEIVER\_NAME"  
 + ", purchases.address AS RECEIVER\_ADDRESS, users.email AS PURCHASERS\_EMAIL from purchase\_items "  
 + "INNER JOIN PURCHASES ON (purchase\_items.purchase\_id=purchases.id) INNER JOIN USERS ON (purchases.user\_id=users.id) ORDER BY price DESC";  
  
 req.app.get('db').query(query).then(purchases => {  
 res.json(purchases);  
 });  
  
});

Above is the creation of the HTTP API endpoints, /users, /users/:id, /products, /products/:id, and /purchases. This section of code is included in the index.js file in the main folder for the lab. This facilitates the HTTP GET operation, connecting to the local pgguide database through massive and express, it returns the matching values

# Part 2.

if(req.query.name != null){  
 query = "SELECT \* FROM products WHERE title = '" + req.query.name + "'"  
}  
  
req.app.get('db').query(query).then(products => {  
 res.json(products);  
});

This is included in the index.js file of the main folder of the lab. This is similar to the /products HTTP API endpoint created in the previous section, however this section acknowledges when a name is entered. This checks the name against the products table title column and returns items that match the entered value.

# Part 3

//paramaterised query  
//by storing the first param in $1 and this automatically espaces the paramaters  
//let query = "SELECT \* FROM products WHERE title = $1";

The above is a paramaterised query included in the index.js file of the main folder. This query prevents SQL injection as the $1 value is replaced with the user input. This avoid the user being able to take advantage of bad SQL code and changing things within the database.

//This uses the stored procedure in the psql database to extract the data  
//A psql procedure was called getproductbyname  
**let** query = "SELECT getproductbyname('" + req.query.name + "')";  
  
req.app.get('db').query(query , [req.query.name]).then(products => {  
 res.json(products);  
});

The code above shows a stored procedure in the index.js file of the main folder. The code calls a procedure getproductbyname, that is included in the pgguide database. This is pre-compiled and pre-parsed, so users will not be able to injection SQL code that can alter the database, preventing any SQL injection.

# Part 4

I tried to use migrations for this section, but they wouldn’t work. I manually created models based off online examples. Below is a copy of the users.js model created, all the models follow a similar format to this and are included in the models folder.

'use strict';  
module.exports = (sequelize, DataTypes) => {  
 **const** users = sequelize.define('users', {  
 id: {  
 type: DataTypes.INTEGER,  
 primaryKey: **true**,  
 autoIncrement: **true** },  
 details: DataTypes.STRING  
 }, {});  
  
 users.associate = **function**(models) {  
 // associations can be defined here  
 };  
  
  
 **return** users;  
};

# Part 5

The following code creates randomised data to populate the database and is included in the server.js file. This was ran once to add some extra data to the local pgguide database, then commented out.

db.products.bulkCreate(  
 times(10, () => ({  
 title: faker.lorem.sentence(),  
 price: random(1, 10),  
 tags: faker.lorem.sentence()  
 }))  
 );  
  
 db.users.bulkCreate(  
 times(250, () => ({  
 details: faker.lorem.sentence()  
 }))  
 );  
  
 db.purchases.bulkCreate(  
 times(100, () => ({  
 name: faker.lorem.sentence(),  
 address: faker.lorem.sentence(),  
 zipcode: random(1, 20)  
 }))  
 );  
  
 db.purchase\_items.bulkCreate(  
 times(100, () => ({  
 price: random(1, 10),  
 quantity:random(1, 20),  
 state: faker.lorem.sentence()  
 }))  
 );

# Part 6

The following code is from the products-api-pt6 js file. It implements CRUD operations on the products HTTP API endpoint as described in the worksheet. Each is indicated with comments as to which operation it is referring to.

module.exports = (app, db) => {  
  
  
 //Gets all products in the database  
 app.get( "/products", (req, res) =>  
 db.products.findAll().then((result) =>  
 res.json(result))  
 );  
  
 //Gets products with matching id  
 app.get( "/products/:id", (req, res) =>  
 db.products.findById(req.params.id).then((result) =>  
 res.json(result))  
 );  
  
  
 //creates products entry with title and price  
 app.post('/products', **function** (req, res) {  
 //values for new entry  
 db.products.create({  
 title: req.body.title,  
 price: req.body.price  
 }).then(**function** (products) {  
  
 res.json(products);  
 });  
 });  
  
 //updates title and price of product with matching id  
 app.put( "/products/:id", (req, res) =>  
 //values for update  
 db.products.update({  
 title: req.body.title,  
 price: req.body.price  
 },  
  
 {  
 where: {  
 id: req.params.id  
 }  
 }).then( (result) => res.json(result) )  
 );  
  
  
 //deletes product with matching id  
 app.delete( "/products/:id", (req, res) =>  
 db.products.destroy({  
 where: {  
 id: req.params.id  
 }  
 }).then( (result) =>  
 res.json(result) )  
 );  
}