Student Number: C15440858

Module: Enterprise Application Development

Lab: 1

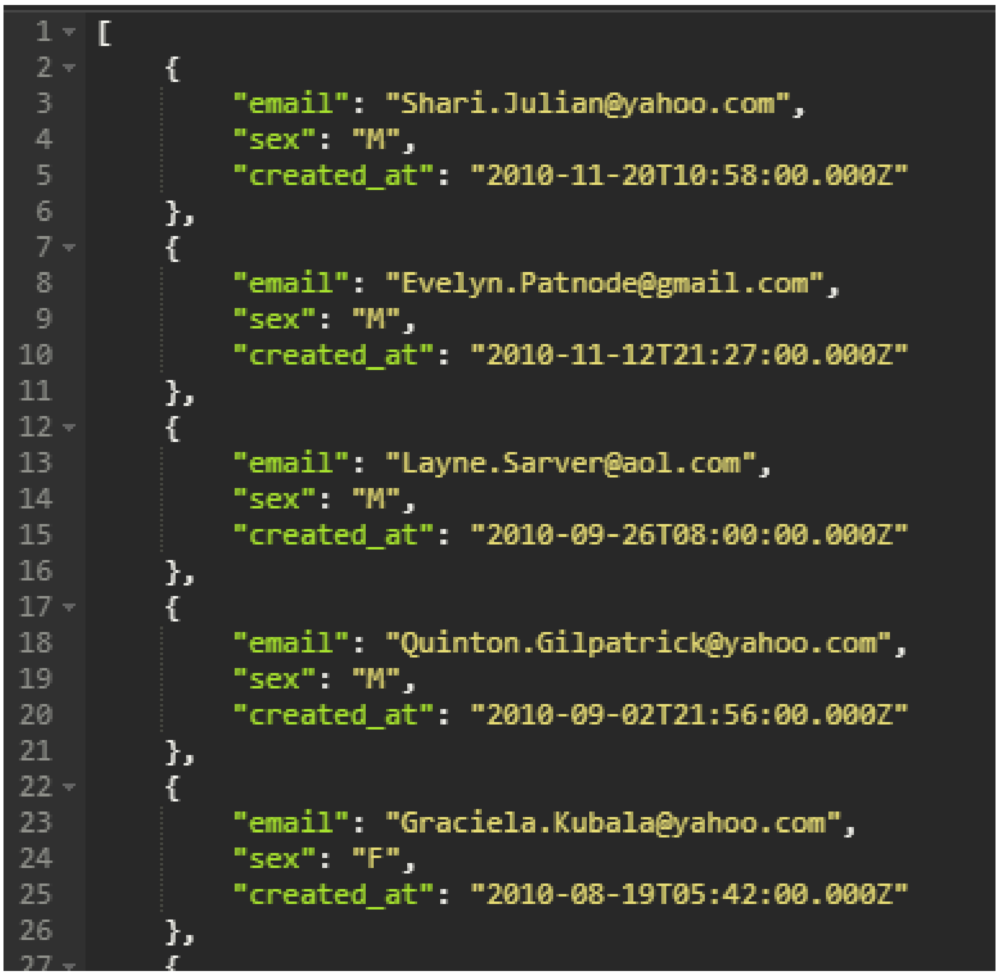
Question 1.1

GET /users

Code:

```
//GET /users
app.get('/users', (req, res, next) =>
{
    const query = "SELECT email, details->'sex' AS SEX, created_at FROM users ORDER BY created_at DESC"
    db.query(query).then(result =>
    {
        res.json(result);
    });
});
```

Output:

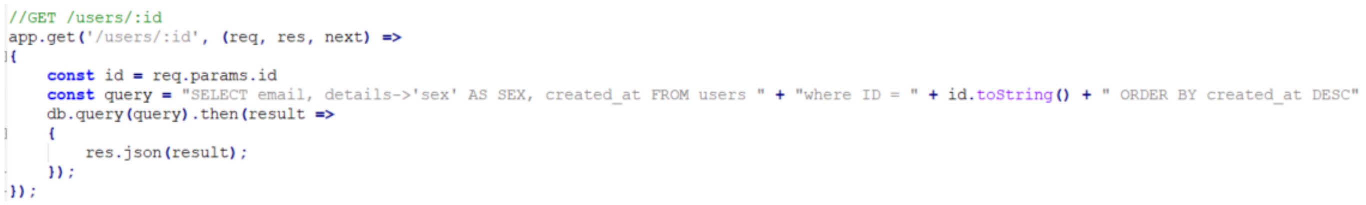


```
[
    {
        "email": "Shari.Julian@yahoo.com",
        "sex": "M",
        "created_at": "2010-11-20T10:58:00.000Z"
    },
    {
        "email": "Evelyn.Patnode@gmail.com",
        "sex": "M",
        "created_at": "2010-11-12T21:27:00.000Z"
    },
    {
        "email": "Layne.Sarver@aol.com",
        "sex": "M",
        "created_at": "2010-09-26T08:00:00.000Z"
    },
    {
        "email": "Quinton.Gilpatrick@yahoo.com",
        "sex": "M",
        "created_at": "2010-09-02T21:56:00.000Z"
    },
    {
        "email": "Graciela.Kubala@yahoo.com",
        "sex": "F",
        "created_at": "2010-08-19T05:42:00.000Z"
    },
    {
```

Question 1.2

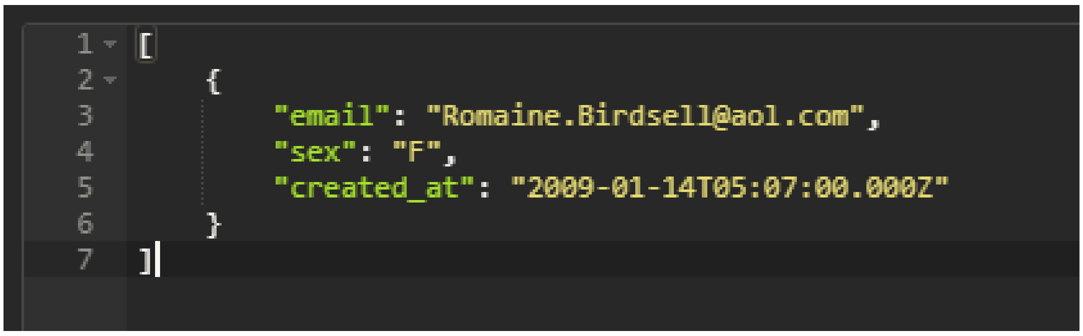
GET /users/:id

Code:



```
//GET /users/:id
app.get('/users/:id', (req, res, next) =>
{
    const id = req.params.id
    const query = "SELECT email, details->'sex' AS SEX, created_at FROM users " + "where ID = " + id.toString() + " ORDER BY created_at DESC"
    db.query(query).then(result =>
    {
        res.json(result);
    });
});
```

Output:

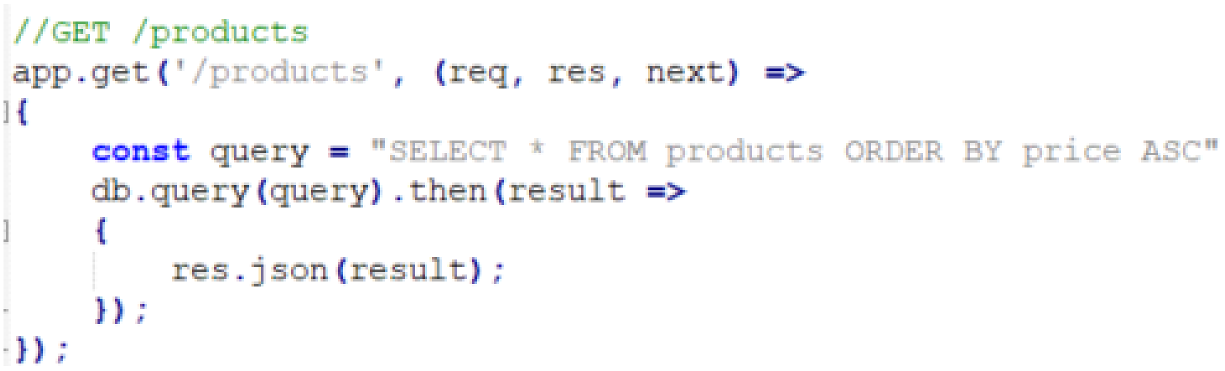


```
[
    {
        "email": "Romaine.Birdsell@aol.com",
        "sex": "F",
        "created_at": "2009-01-14T05:07:00.000Z"
    }
]
```

Question 1.3

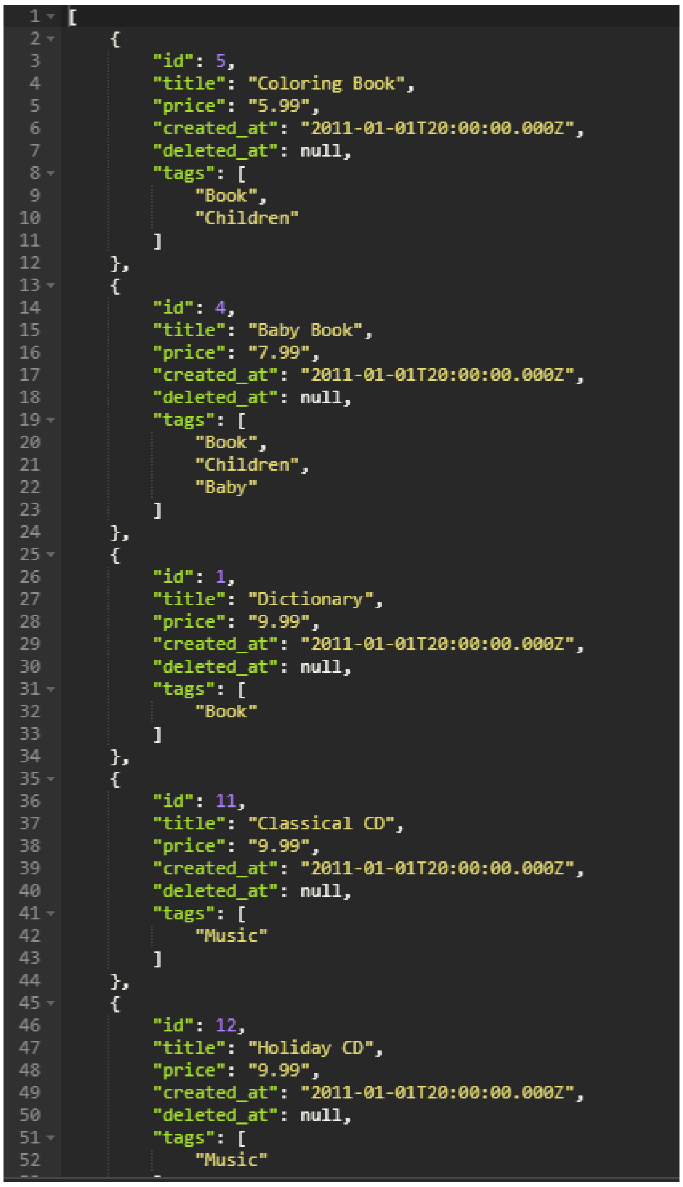
GET /products

Code:



```
//GET /products
app.get('/products', (req, res, next) =>
{
    const query = "SELECT * FROM products ORDER BY price ASC"
    db.query(query).then(result =>
    {
        res.json(result);
    });
});
```

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"
        ]
    },
    {
        "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"
```

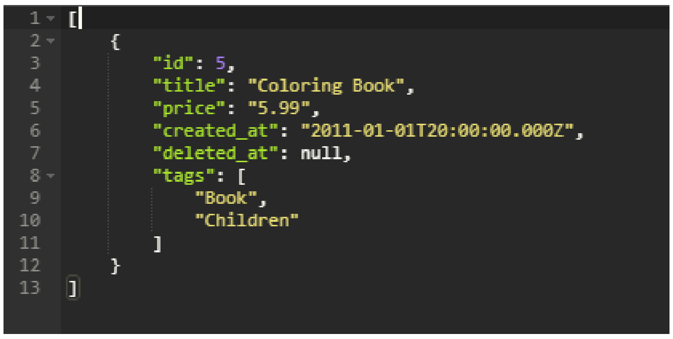
Question 1.4

GET /products/:id

Code:

```
//GET /products/:id
app.get('/products/:id', (req, res, next) =>
{
    const id = req.params.id
    const query = "SELECT * FROM products " + "where ID = " + id.toString() + " ORDER BY price ASC"

    db.query(query).then(result =>
    {
        res.json(result);
    });
});
```

Output:

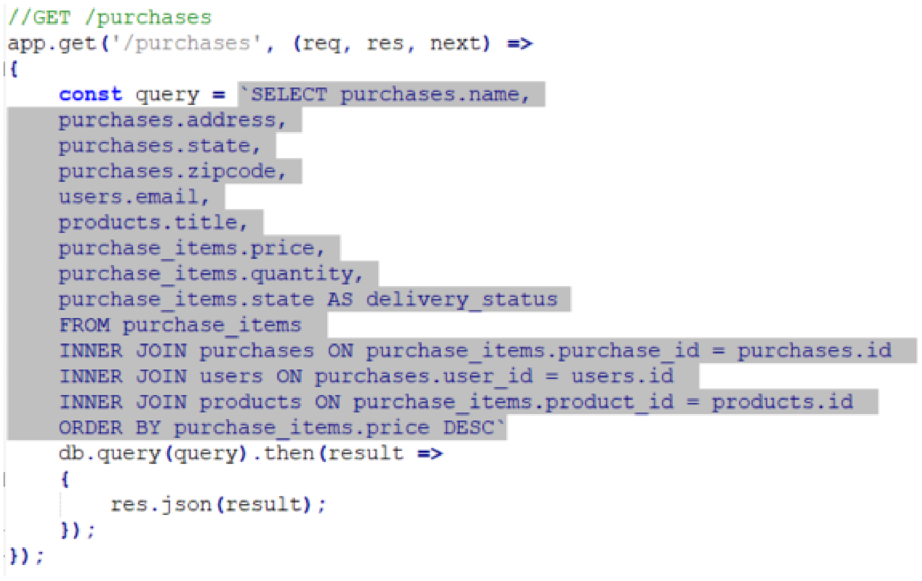


```
[
    {
        "id": 5,
        "title": "Coloring Book",
        "price": "5.99",
        "created_at": "2011-01-01T20:00:00.000Z",
        "deleted_at": null,
        "tags": [
            "Book",
            "Children"
        ]
    }
]
```

Question 1.5

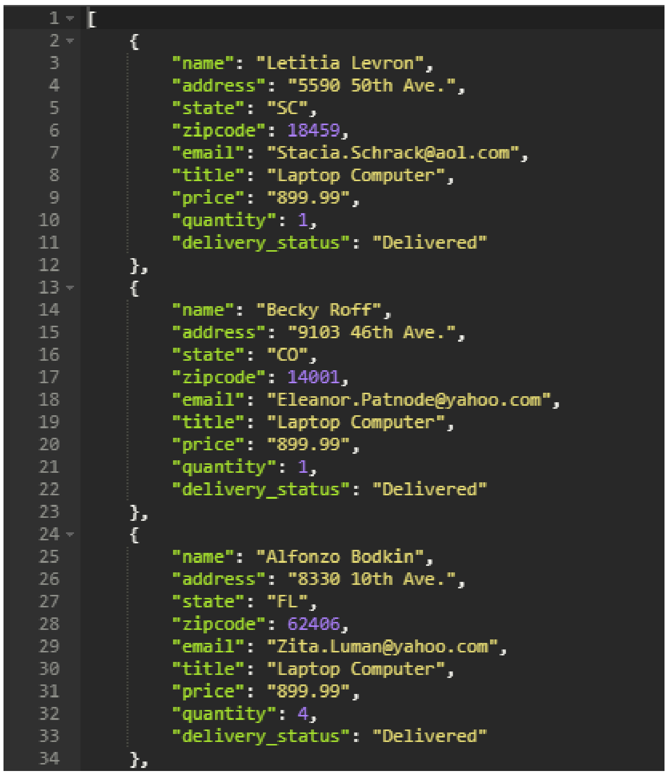
GET /purchases

Code:



```
//GET /purchases
app.get('/purchases', (req, res, next) =>
{
    const query = `SELECT purchases.name,
    purchases.address,
    purchases.state,
    purchases.zipcode,
    users.email,
    products.title,
    purchase_items.price,
    purchase_items.quantity,
    purchase_items.state AS delivery_status
    FROM purchase_items
    INNER JOIN purchases ON purchase_items.purchase_id = purchases.id
    INNER JOIN users ON purchases.user_id = users.id
    INNER JOIN products ON purchase_items.product_id = products.id
    ORDER BY purchase_items.price DESC`
    db.query(query).then(result =>
    {
        res.json(result);
    });
});
```

Output:

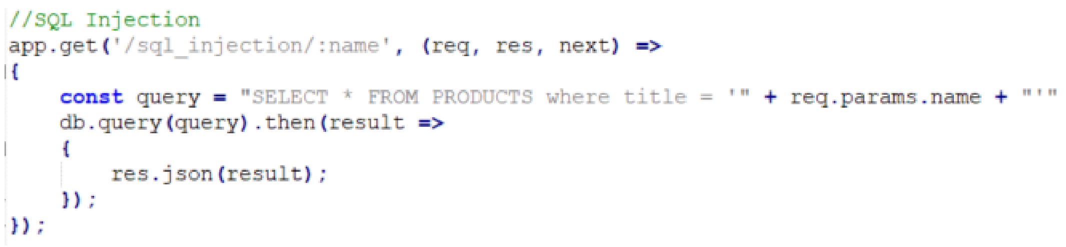


```
[
    {
        "name": "Letitia Levron",
        "address": "5590 50th Ave.",
        "state": "SC",
        "zipcode": 18459,
        "email": "Stacia.Schrack@aol.com",
        "title": "Laptop Computer",
        "price": "899.99",
        "quantity": 1,
        "delivery_status": "Delivered"
    },
    {
        "name": "Becky Roff",
        "address": "9103 46th Ave.",
        "state": "CO",
        "zipcode": 14001,
        "email": "Eleanor.Patnode@yahoo.com",
        "title": "Laptop Computer",
        "price": "899.99",
        "quantity": 1,
        "delivery_status": "Delivered"
    },
    {
        "name": "Alfonzo Bodkin",
        "address": "8330 10th Ave.",
        "state": "FL",
        "zipcode": 62406,
        "email": "Zita.Luman@yahoo.com",
        "title": "Laptop Computer",
        "price": "899.99",
        "quantity": 4,
        "delivery_status": "Delivered"
    },
```

Question 2

GET /products[?name=*string*]

In this query, we are taking in a parameter (name) and concatenating the parameter to our query. This is very bad practice as it can allow the unauthorised execution of SQL code. This can result in the unauthorised deletion of tables and alteration of data. The following code below is prone to SQL injection.



```
//SQL Injection
app.get('/sql_injection/:name', (req, res, next) =>
{
    const query = "SELECT * FROM PRODUCTS where title = '" + req.params.name + "'"
    db.query(query).then(result =>
    {
        res.json(result);
    });
});
```

The above code is expecting the input through the "*name*" parameter. Since the parameter is simply concatenated to the query that will be run, this means a user can expand the query through the parameter.

For example:

localhost:3000/products/'; DELETE from products where id = 30 OR title = '

will cause the following SQL to be executed.

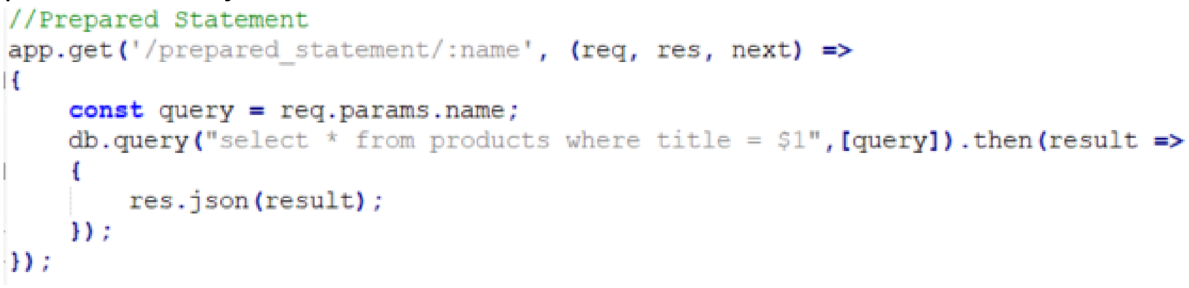
SELECT * FROM PRODUCTS where title = ''; DELETE from products where id = 30 OR title = ''

Question 3

SQL Injection Prevention

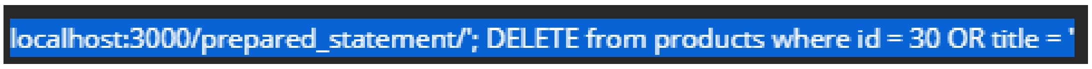
SQL Injection is a code injection technique that is used to attack data systems. It allows an attacker to execute their own query on a system in which they do not have permissions. There are a few ways to prevent this, such as:

- **Parameterised Query –** By preparing the query with parameters rather than concatenating them to the query, we can eliminate SQL injection totally. For example, the following code is a prepared statement and will prevent SQL Injection.



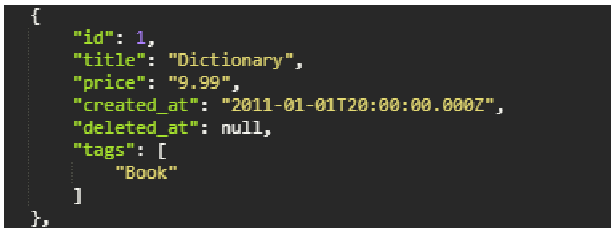
```
//Prepared Statement
app.get('/prepared_statement/:name', (req, res, next) =>
{
    const query = req.params.name;
    db.query("select * from products where title = $1",[query]).then(result =>
    {
        res.json(result);
    });
});
```

  When we try to perform SQL Injection, it does not work.



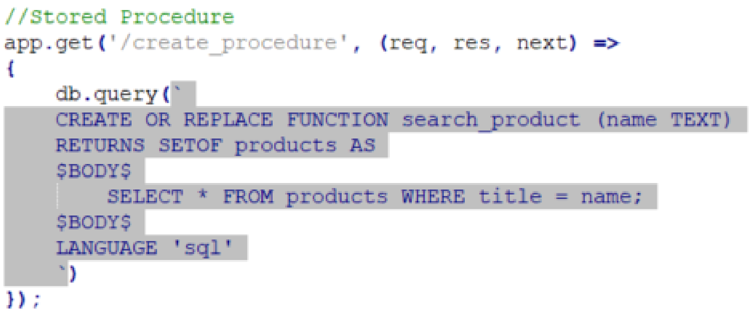
```
localhost:3000/prepared_statement/'; DELETE from products where id = 30 OR title = '
```

  We can see this as the product is still there.



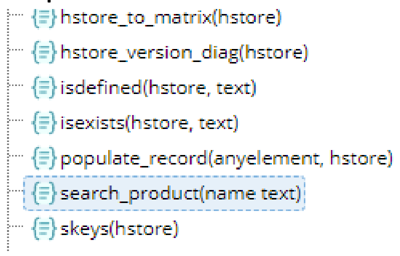
```
{
    "id": 1,
    "title": "Dictionary",
    "price": "9.99",
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null,
    "tags": [
        "Book"
    ]
},
```

- **Stored Procedure –** A stored procedure is a statement in a relational database that can be reused and shared by multiple programs. A benefit to this is that it prevents SQL Injection because similarly to parameterised queries, it prepares the queries before executing them. Firstly, we can create the procedure through JavaScript code.



```
//Stored Procedure
app.get('/create_procedure', (req, res, next) =>
{
    db.query(`
    CREATE OR REPLACE FUNCTION search_product (name TEXT)
    RETURNS SETOF products AS
    $BODY$
        SELECT * FROM products WHERE title = name;
    $BODY$
    LANGUAGE 'sql'
    `)
});
```

The procedure is now in our database and is ready to run.

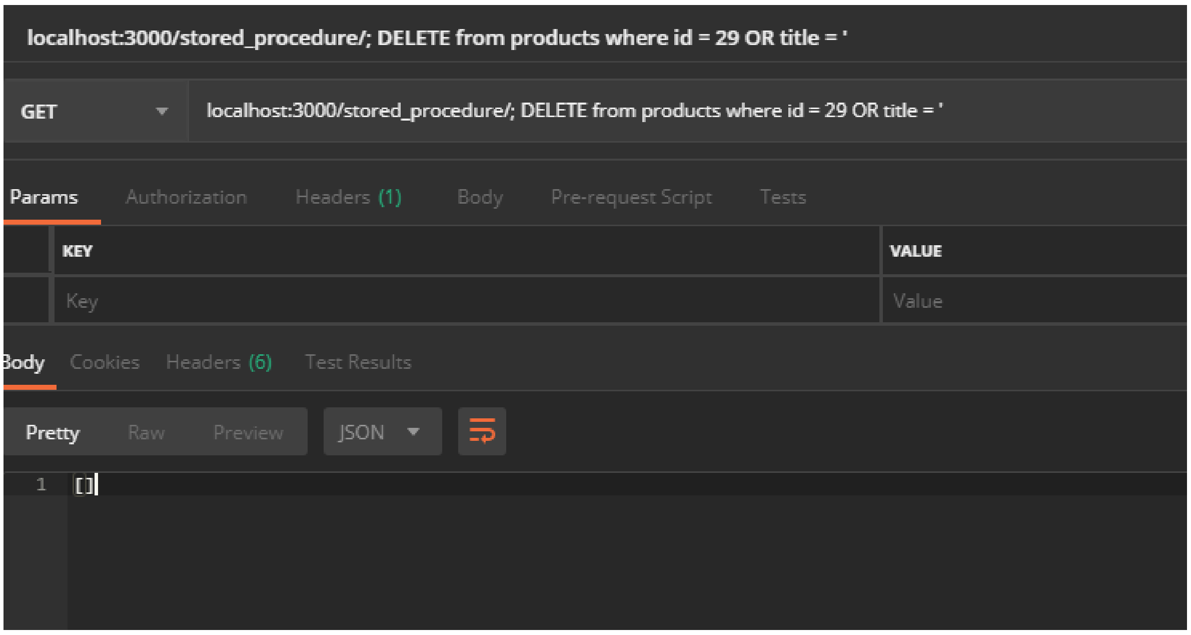


```
hstore_to_matrix(hstore)
hstore_version_diag(hstore)
isdefined(hstore, text)
isexists(hstore, text)
populate_record(anyelement, hstore)
search_product(name text)
skeys(hstore)
```

The code for the running of the procedure:

```
app.get('/stored_procedure/:name', (req, res, next) =>
{
    const name = req.params.name;
    db.query(`SELECT * FROM search_product($1)`, [name]).then((products) =>
    {
        res.json(products);
        res.end();
    });
});
```

When we try to inject SQL, we are unable to.

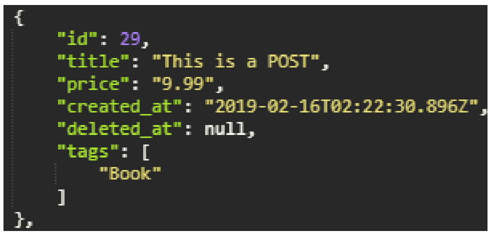


localhost:3000/stored_procedure/; DELETE from products where id = 29 OR title = '

GET localhost:3000/stored_procedure/; DELETE from products where id = 29 OR title = '

```
[]
```

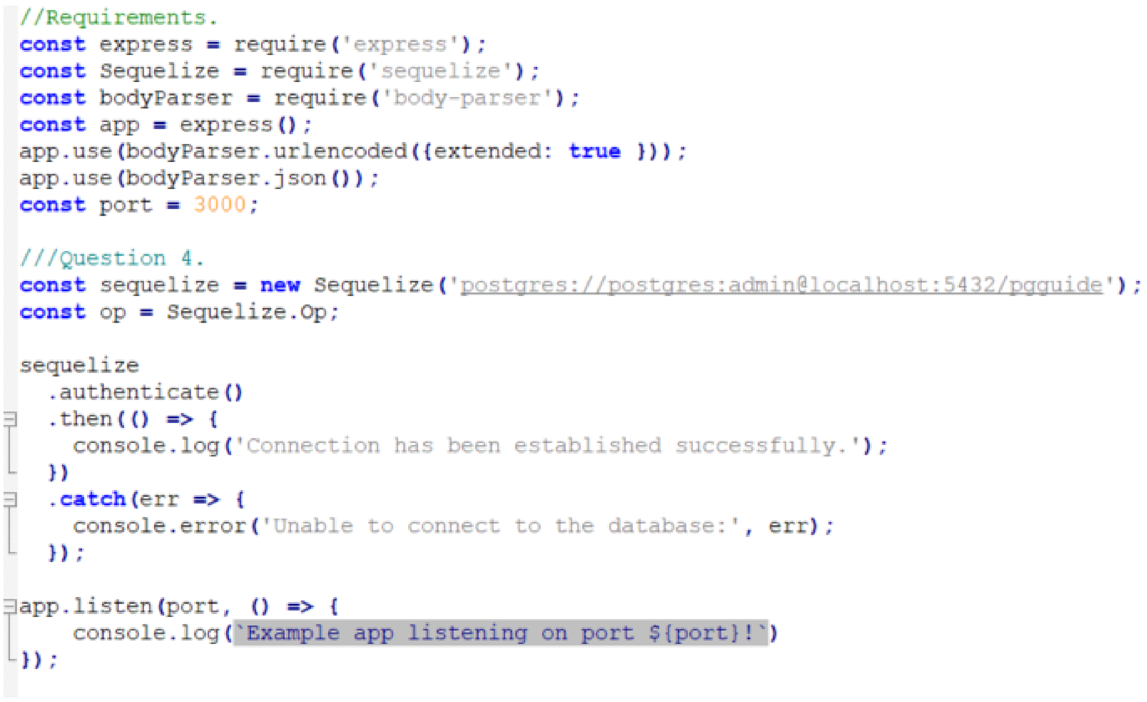
The SQL Injection failed as the product is still in the table.



```
{
    "id": 29,
    "title": "This is a POST",
    "price": "9.99",
    "created_at": "2019-02-16T02:22:30.896Z",
    "deleted_at": null,
    "tags": [
        "Book"
    ]
},
```

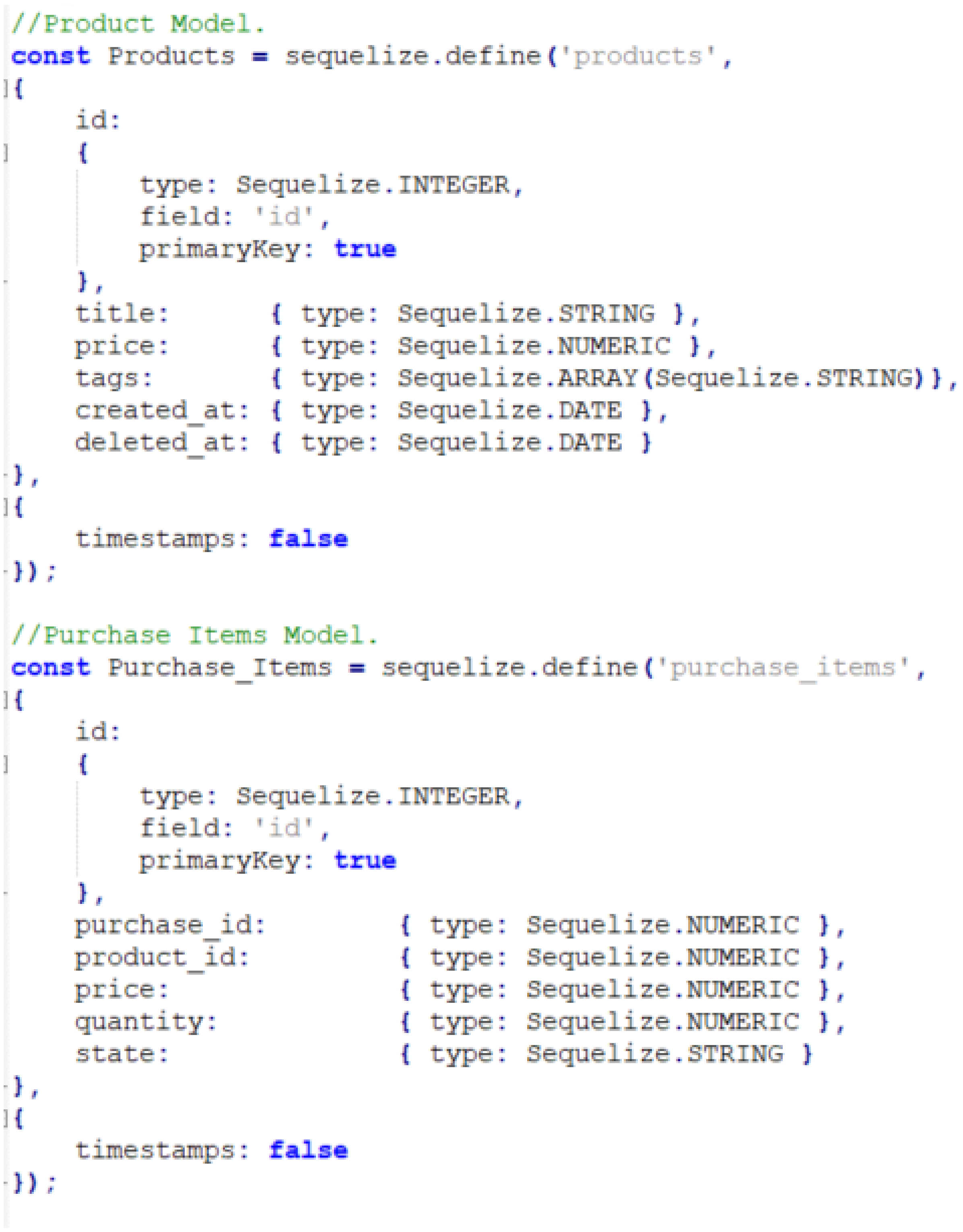
Question 4

Code:



```
//Requirements.
const express = require('express');
const Sequelize = require('sequelize');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.urlencoded({extended: true }));
app.use(bodyParser.json());
const port = 3000;

///Question 4.
const sequelize = new Sequelize('postgres://postgres:admin@localhost:5432/pgguide');
const op = Sequelize.Op;

sequelize
  .authenticate()
  .then(() => {
    console.log('Connection has been established successfully.');
  })
  .catch(err => {
    console.error('Unable to connect to the database:', err);
  });

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

Code:

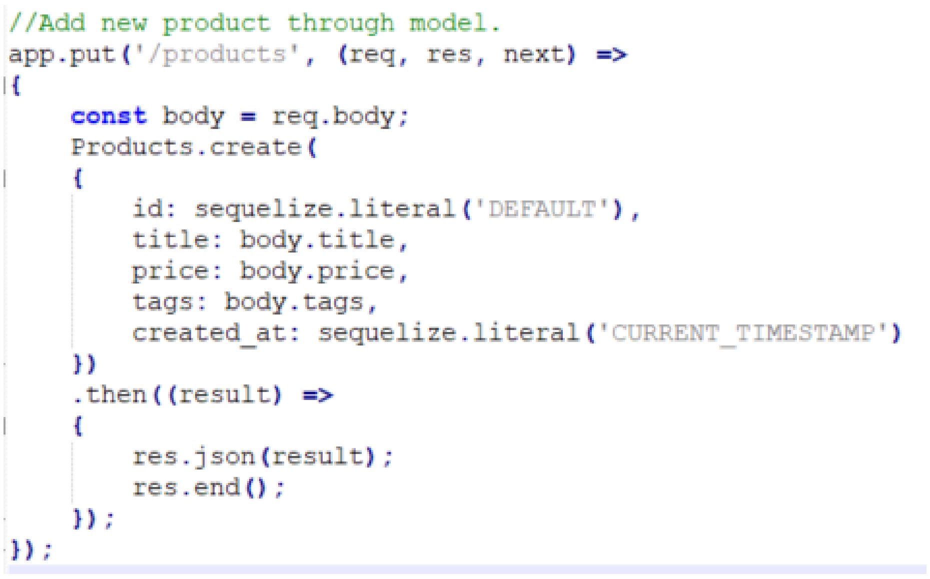


```
//Product Model.
const Products = sequelize.define('products',
{
    id:
    {
        type: Sequelize.INTEGER,
        field: 'id',
        primaryKey: true
    },
    title:      { type: Sequelize.STRING },
    price:      { type: Sequelize.NUMERIC },
    tags:       { type: Sequelize.ARRAY(Sequelize.STRING)},
    created_at: { type: Sequelize.DATE },
    deleted_at: { type: Sequelize.DATE }
},
{
    timestamps: false
});

//Purchase Items Model.
const Purchase_Items = sequelize.define('purchase_items',
{
    id:
    {
        type: Sequelize.INTEGER,
        field: 'id',
        primaryKey: true
    },
    purchase_id:        { type: Sequelize.NUMERIC },
    product_id:         { type: Sequelize.NUMERIC },
    price:              { type: Sequelize.NUMERIC },
    quantity:           { type: Sequelize.NUMERIC },
    state:              { type: Sequelize.STRING }
},
{
    timestamps: false
});
```

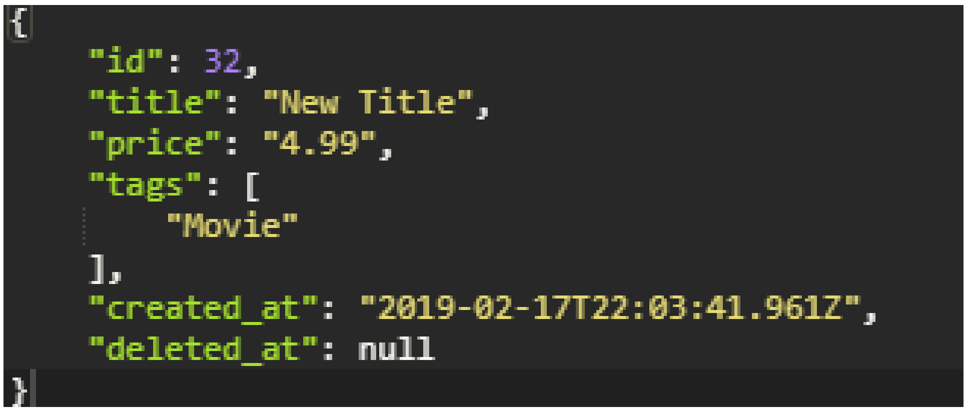
```
//Purchases Model.
const Purchases = sequelize.define('purchases',
{
    id:
    {
        type: Sequelize.INTEGER,
        field: 'id',
        primaryKey: true
    },

    created_at:          { type: Sequelize.DATE },
    name:                { type: Sequelize.STRING },
    address:             { type: Sequelize.STRING },
    state:               { type: Sequelize.STRING },
    zipcode:             { type: Sequelize.NUMERIC },
    user_id:             { type: Sequelize.NUMERIC },
},
{
    timestamps: false
});

//User Model.
const Users = sequelize.define('users',
{
    id:
    {
        type: Sequelize.INTEGER,
        field: 'id',
        primaryKey: true
    },
    email:      { type: Sequelize.STRING, field: 'email' },
    password:   { type: Sequelize.STRING, field: 'password' },
    details:    { type: Sequelize.HSTORE, field: 'details' },
    created_at: { type: Sequelize.DATE },
    deleted_at: { type: Sequelize.DATE }
},
{
    timestamps: false
});
```

Question 5

Products

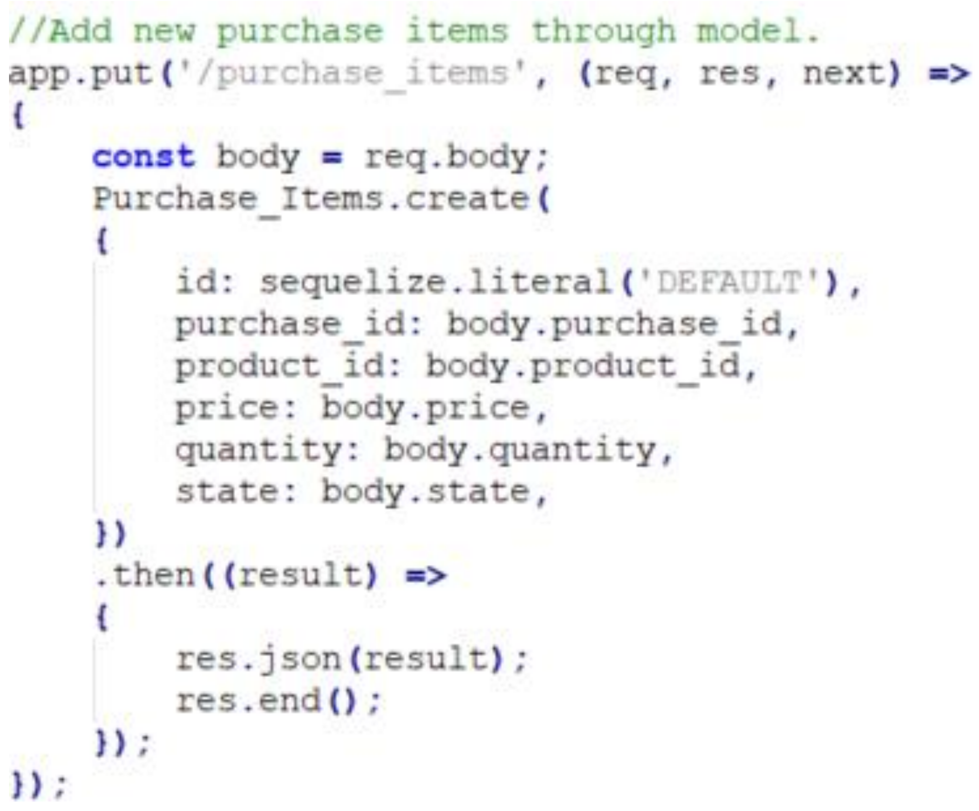


```
//Add new product through model.
app.put('/products', (req, res, next) =>
{
    const body = req.body;
    Products.create(
    {
        id: sequelize.literal('DEFAULT'),
        title: body.title,
        price: body.price,
        tags: body.tags,
        created_at: sequelize.literal('CURRENT_TIMESTAMP')
    })
    .then((result) =>
    {
        res.json(result);
        res.end();
    });
});
```

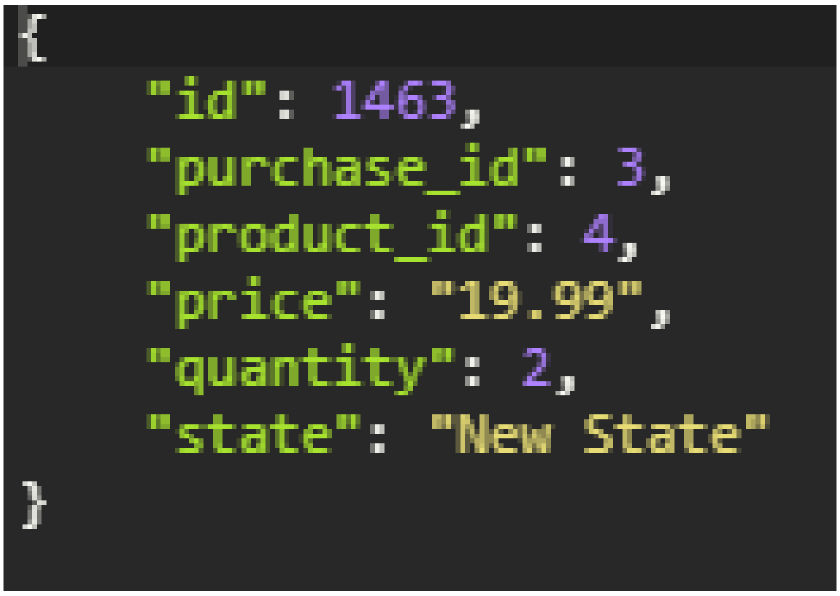


```
{
    "id": 32,
    "title": "New Title",
    "price": "4.99",
    "tags": [
        "Movie"
    ],
    "created_at": "2019-02-17T22:03:41.961Z",
    "deleted_at": null
}
```

Purchase Items

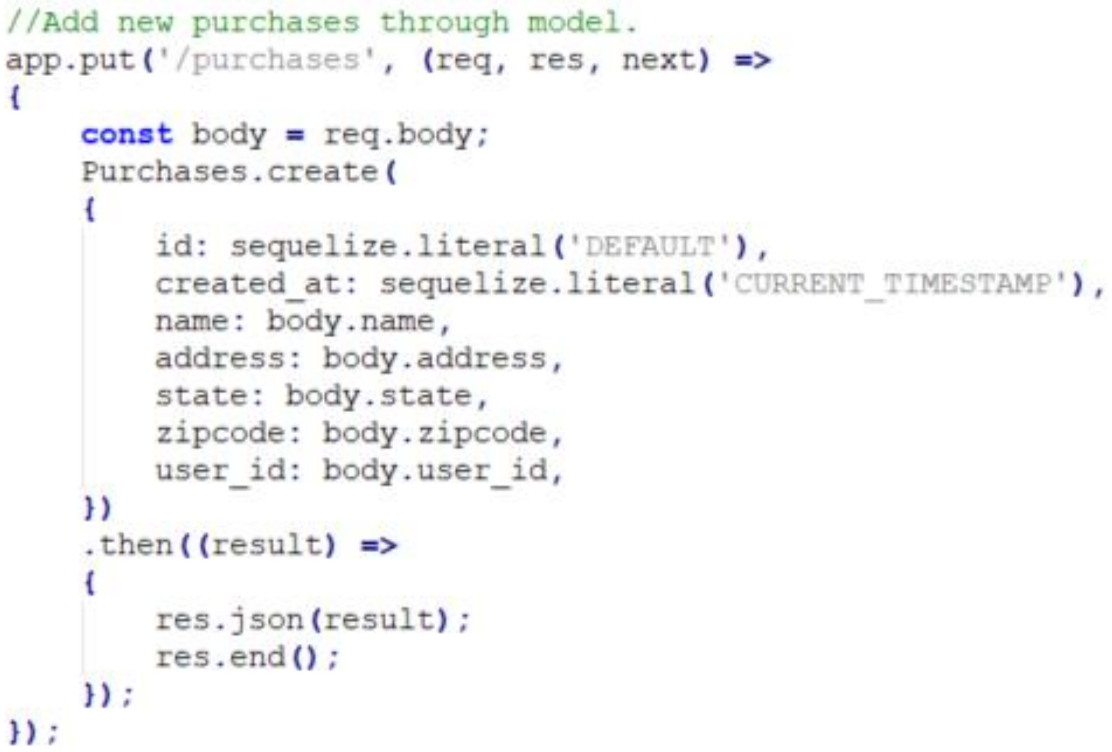


```
//Add new purchase items through model.
app.put('/purchase_items', (req, res, next) =>
{
    const body = req.body;
    Purchase_Items.create(
    {
        id: sequelize.literal('DEFAULT'),
        purchase_id: body.purchase_id,
        product_id: body.product_id,
        price: body.price,
        quantity: body.quantity,
        state: body.state,
    })
    .then((result) =>
    {
        res.json(result);
        res.end();
    });
});
```

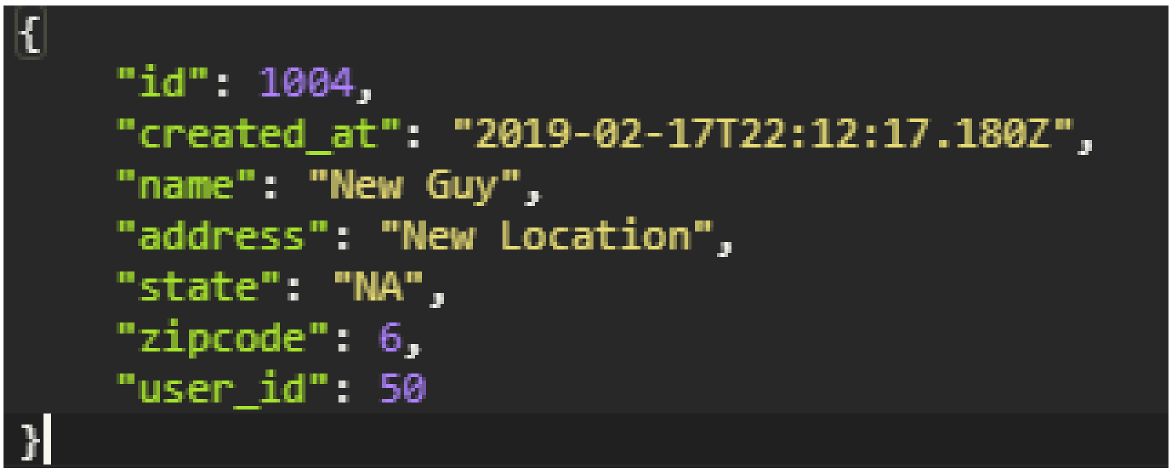


```
{
    "id": 1463,
    "purchase_id": 3,
    "product_id": 4,
    "price": "19.99",
    "quantity": 2,
    "state": "New State"
}
```

Purchases

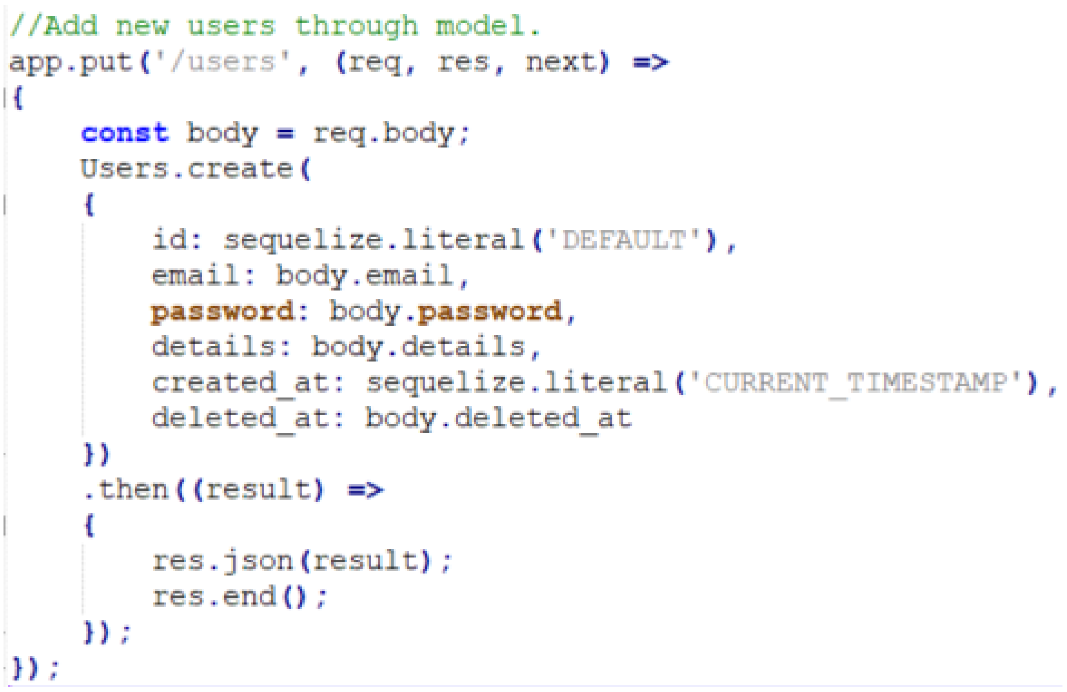


```
//Add new purchases through model.
app.put('/purchases', (req, res, next) =>
{
    const body = req.body;
    Purchases.create(
    {
        id: sequelize.literal('DEFAULT'),
        created_at: sequelize.literal('CURRENT_TIMESTAMP'),
        name: body.name,
        address: body.address,
        state: body.state,
        zipcode: body.zipcode,
        user_id: body.user_id,
    })
    .then((result) =>
    {
        res.json(result);
        res.end();
    });
});
```

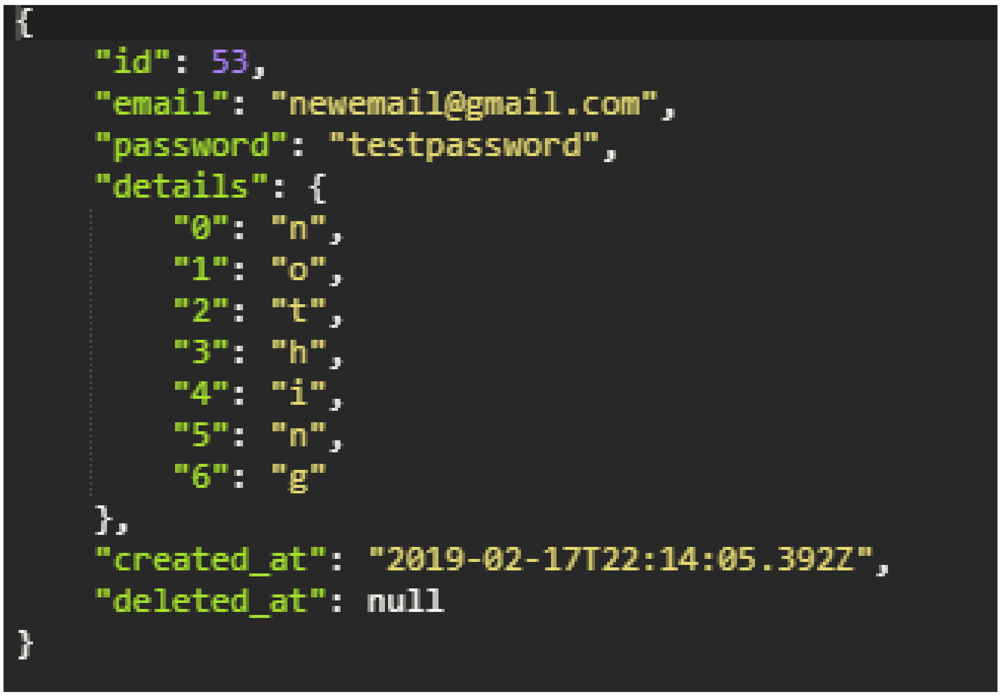


```
{
    "id": 1004,
    "created_at": "2019-02-17T22:12:17.180Z",
    "name": "New Guy",
    "address": "New Location",
    "state": "NA",
    "zipcode": 6,
    "user_id": 50
}
```

Users



```
//Add new users through model.
app.put('/users', (req, res, next) =>
{
    const body = req.body;
    Users.create(
    {
        id: sequelize.literal('DEFAULT'),
        email: body.email,
        password: body.password,
        details: body.details,
        created_at: sequelize.literal('CURRENT_TIMESTAMP'),
        deleted_at: body.deleted_at
    })
    .then((result) =>
    {
        res.json(result);
        res.end();
    });
});
```



```
{
    "id": 53,
    "email": "newemail@gmail.com",
    "password": "testpassword",
    "details": {
        "0": "n",
        "1": "o",
        "2": "t",
        "3": "h",
        "4": "i",
        "5": "n",
        "6": "g"
    },
    "created_at": "2019-02-17T22:14:05.392Z",
    "deleted_at": null
}
```

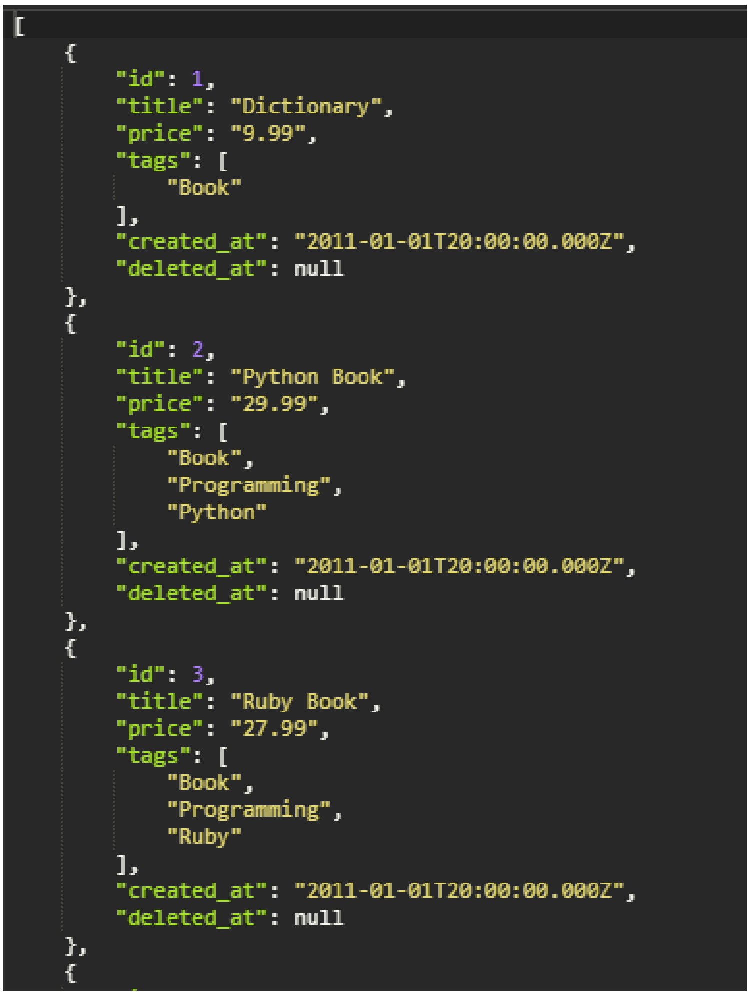
Question 6.1

GET /products

Code:

```
//GET /products
app.get('/products', (req, res, next) =>
{
    Products.findAll(
    {
        order:
        [
            ['id', 'ASC']
        ]
    }).then((result) =>
    {
        res.json(result);
    });
});
```

Output:

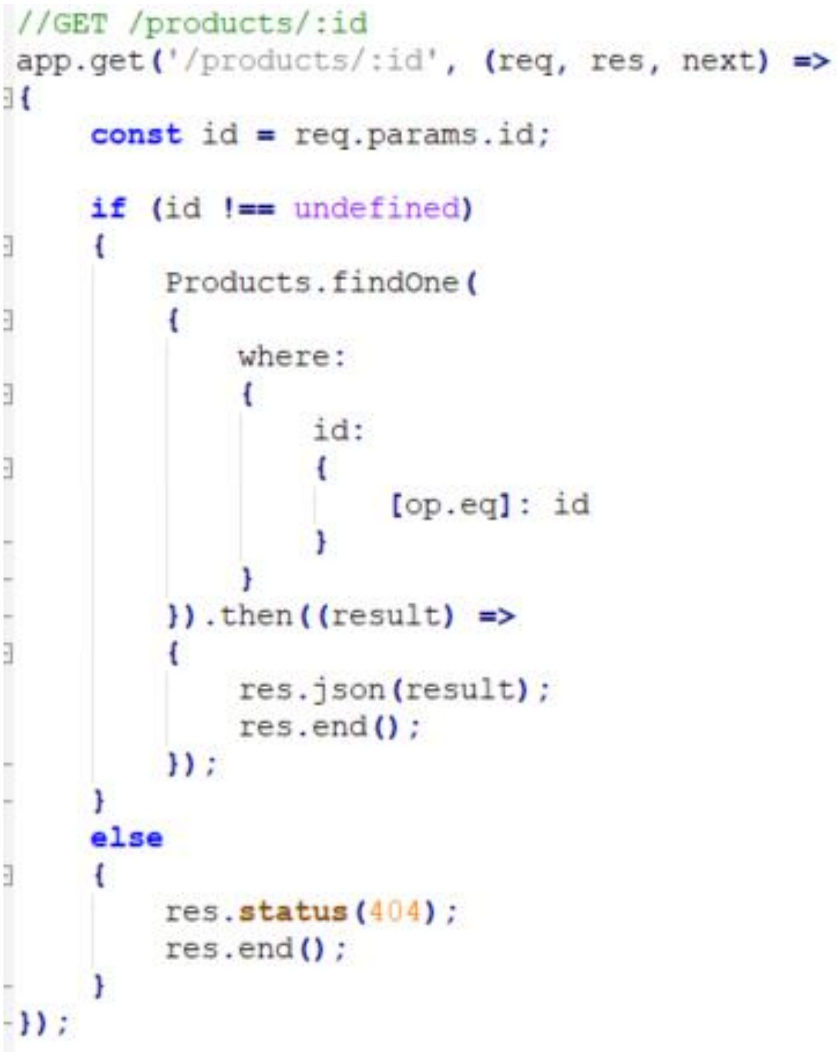


```
[
    {
        "id": 1,
        "title": "Dictionary",
        "price": "9.99",
        "tags": [
            "Book"
        ],
        "created_at": "2011-01-01T20:00:00.000Z",
        "deleted_at": null
    },
    {
        "id": 2,
        "title": "Python Book",
        "price": "29.99",
        "tags": [
            "Book",
            "Programming",
            "Python"
        ],
        "created_at": "2011-01-01T20:00:00.000Z",
        "deleted_at": null
    },
    {
        "id": 3,
        "title": "Ruby Book",
        "price": "27.99",
        "tags": [
            "Book",
            "Programming",
            "Ruby"
        ],
        "created_at": "2011-01-01T20:00:00.000Z",
        "deleted_at": null
    },
    {
```

Question 6.2

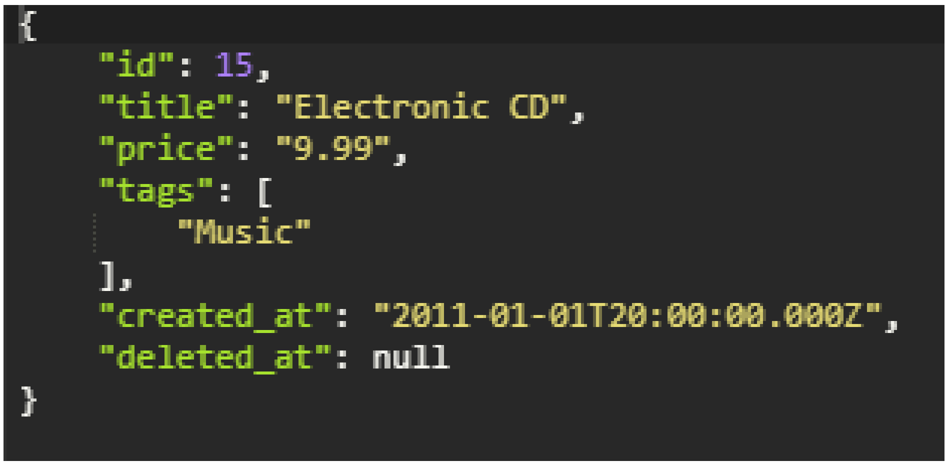
GET /products/:id

Code:



```
//GET /products/:id
app.get('/products/:id', (req, res, next) =>
{
    const id = req.params.id;

    if (id !== undefined)
    {
        Products.findOne(
        {
            where:
            {
                id:
                {
                    [op.eq]: id
                }
            }
        }).then((result) =>
        {
            res.json(result);
            res.end();
        });
    }
    else
    {
        res.status(404);
        res.end();
    }
});
```

Output:

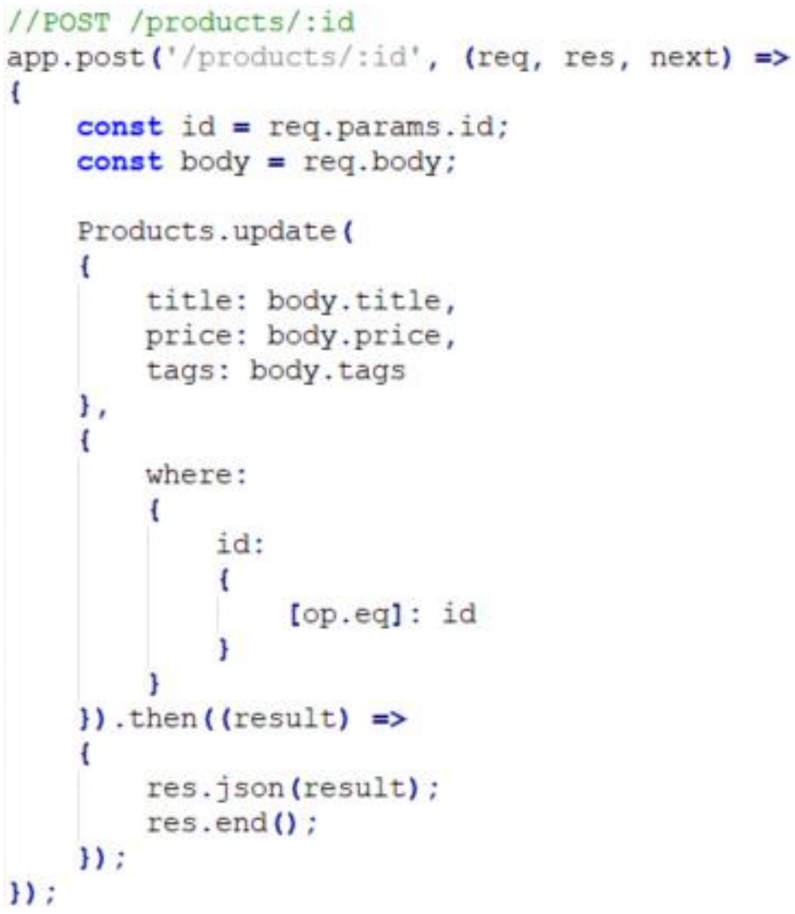


```
{
    "id": 15,
    "title": "Electronic CD",
    "price": "9.99",
    "tags": [
        "Music"
    ],
    "created_at": "2011-01-01T20:00:00.000Z",
    "deleted_at": null
}
```

Question 6.3

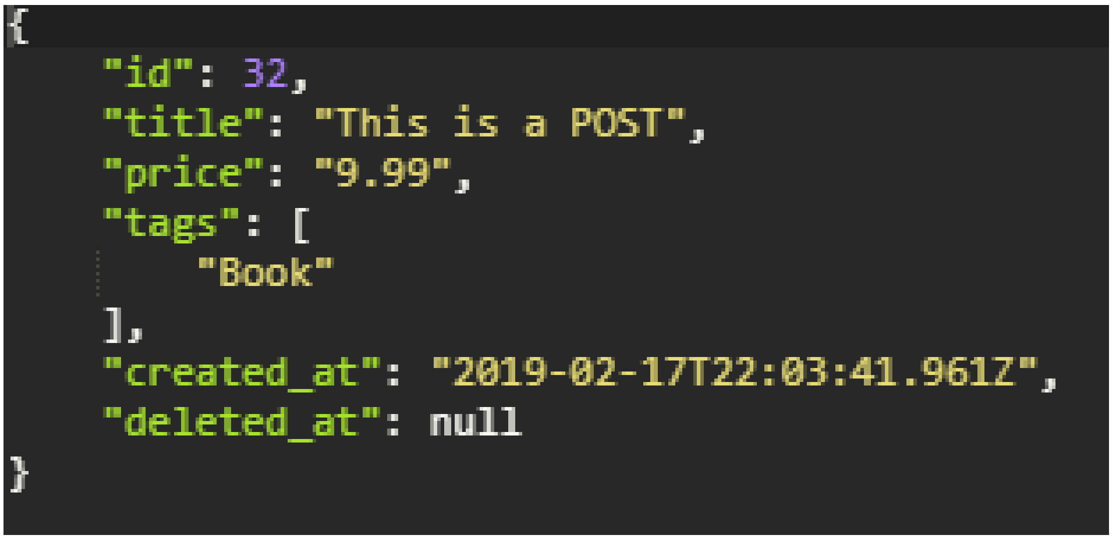
POST /products/:id

Code:



```
//POST /products/:id
app.post('/products/:id', (req, res, next) =>
{
    const id = req.params.id;
    const body = req.body;

    Products.update(
    {
        title: body.title,
        price: body.price,
        tags: body.tags
    },
    {
        where:
        {
            id:
            {
                [op.eq]: id
            }
        }
    }).then((result) =>
    {
        res.json(result);
        res.end();
    });
});
```

Output:

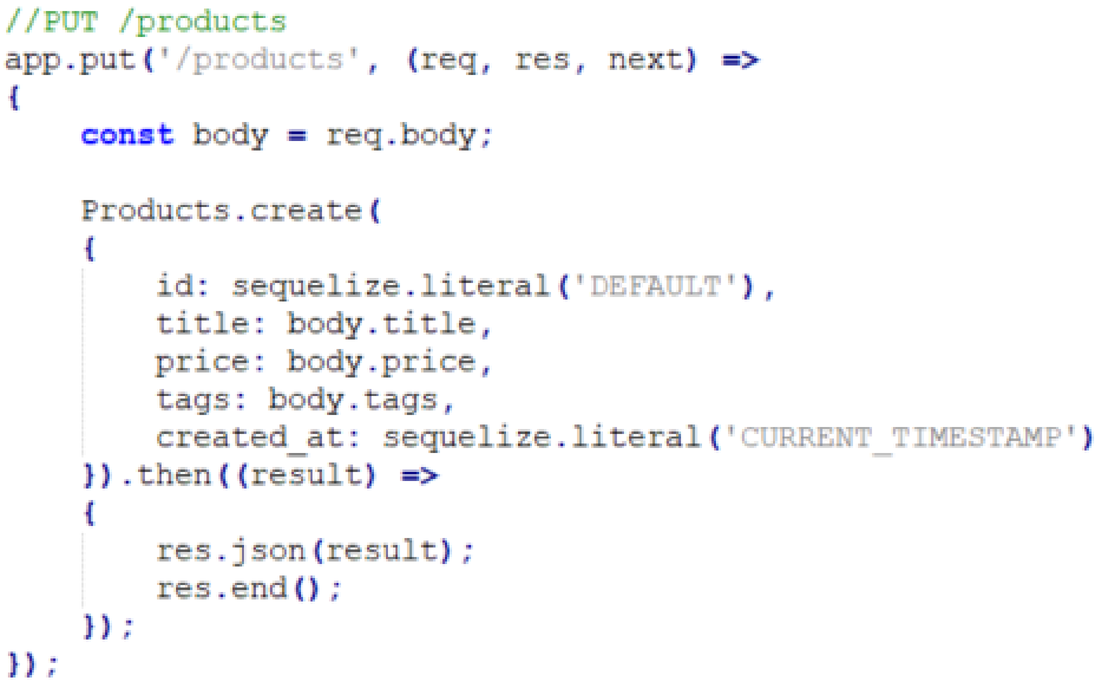


```
{
    "id": 32,
    "title": "This is a POST",
    "price": "9.99",
    "tags": [
        "Book"
    ],
    "created_at": "2019-02-17T22:03:41.961Z",
    "deleted_at": null
}
```

Question 6.4

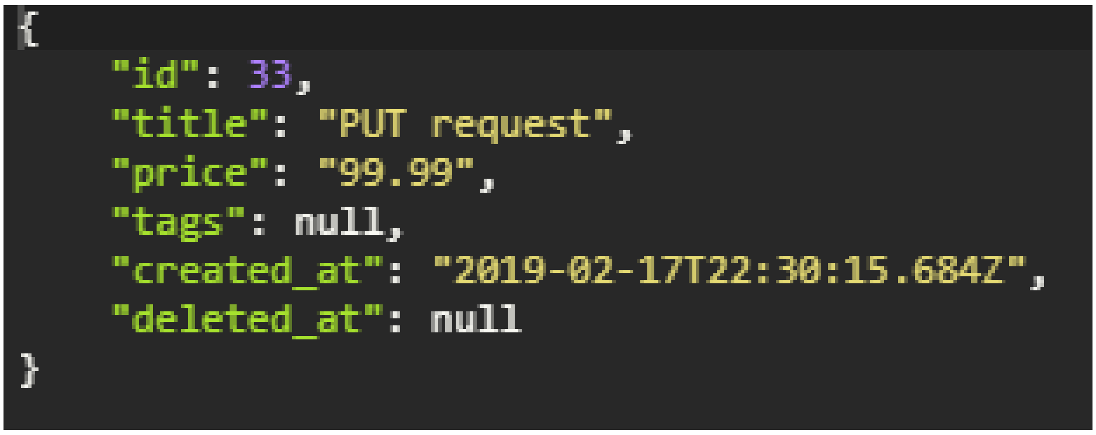
PUT /products

Code:



```
//PUT /products
app.put('/products', (req, res, next) =>
{
    const body = req.body;

    Products.create(
    {
        id: sequelize.literal('DEFAULT'),
        title: body.title,
        price: body.price,
        tags: body.tags,
        created_at: sequelize.literal('CURRENT_TIMESTAMP')
    }).then((result) =>
    {
        res.json(result);
        res.end();
    });
});
```

Output:

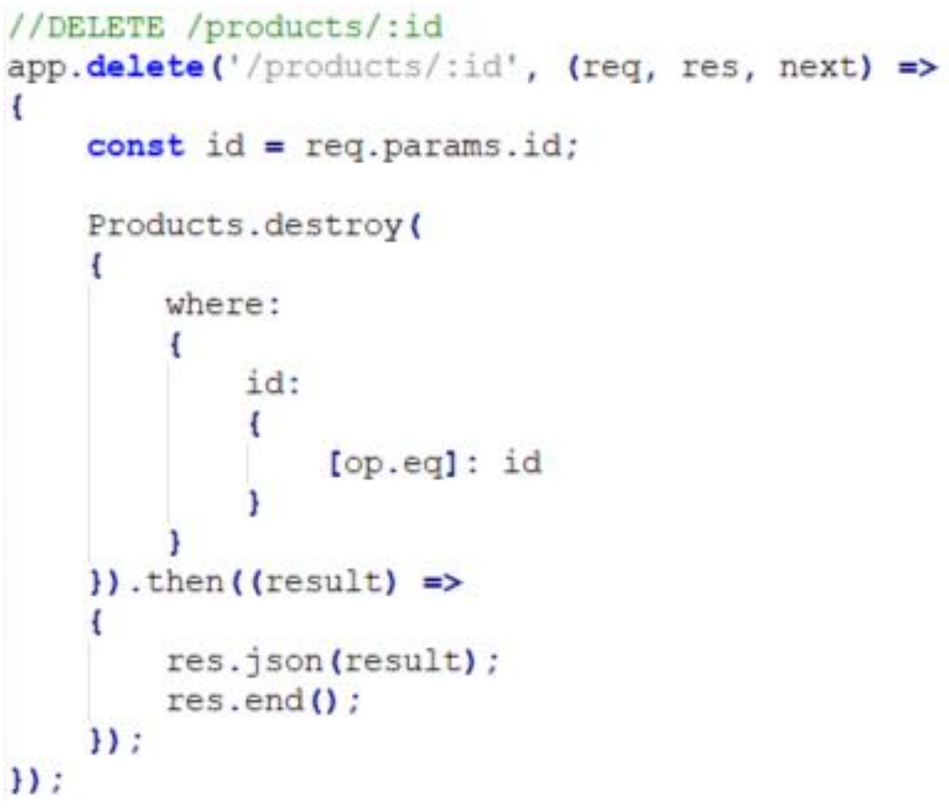


```
{
    "id": 33,
    "title": "PUT request",
    "price": "99.99",
    "tags": null,
    "created_at": "2019-02-17T22:30:15.684Z",
    "deleted_at": null
}
```

Question 6.5

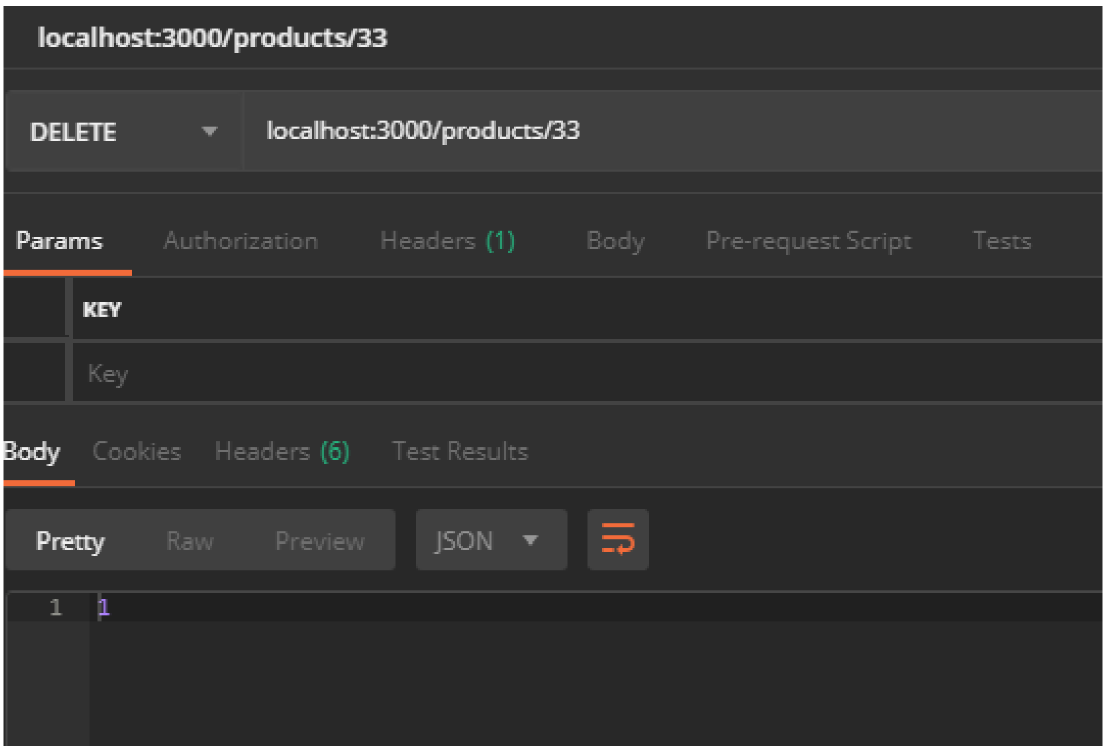
DELETE /products/:id

Code:



```
//DELETE /products/:id
app.delete('/products/:id', (req, res, next) =>
{
    const id = req.params.id;

    Products.destroy(
    {
        where:
        {
            id:
            {
                [op.eq]: id
            }
        }
    }).then((result) =>
    {
        res.json(result);
        res.end();
    });
});
```

Output:



localhost:3000/products/33

DELETE localhost:3000/products/33

```
1
```