Part 1

Creating a users table, and a restricted table products.

Inserting data to each, encrypting with crypt() and gen salt()

```
CREATE TABLE users (
    username TEXT,
    password TEXT
);

INSERT INTO users values ('Chloe_Doyle', crypt('password', gen_salt('md5')));
INSERT INTO users values ('Test2', crypt('password123', gen_salt('md5')));

create table products
  (
    ID SERIAL UNIQUE,
    prod_name TEXT,
    price FLOAT
  );

insert into products (prod_name, price) values ('Book1', 5.99);
insert into products(prod_name, price) values ('Book2', 7.99);
insert into products(prod_name, price) values('Book3', 3.99);
insert into products(prod_name, price)values ('Book4', 10.99);
```

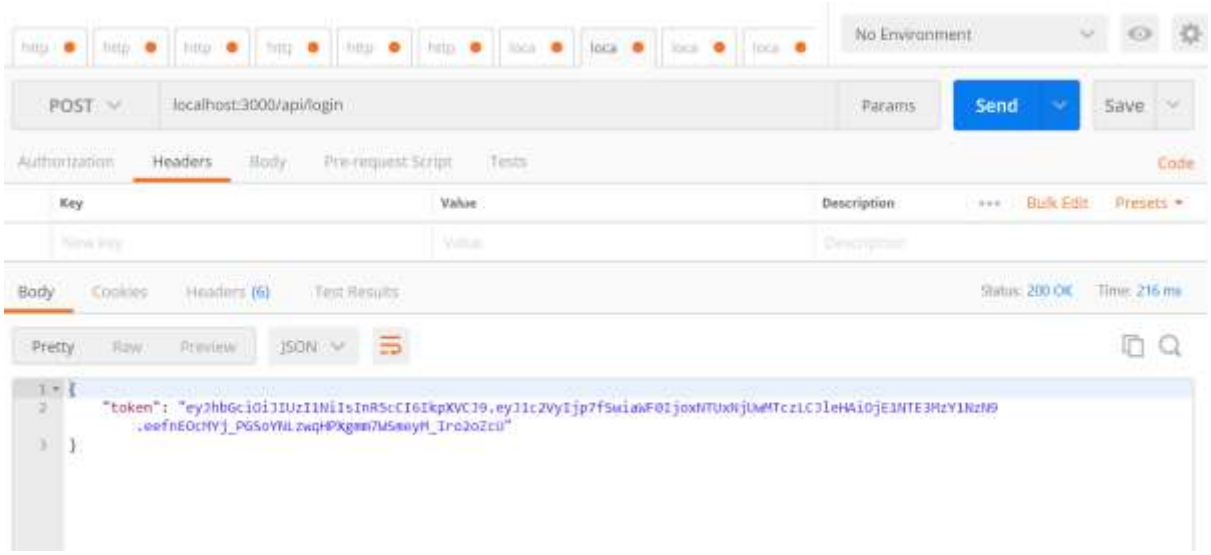
```
postgres=# SET search_path TO lab2;
SET
postgres=# select * from users;
  username   |              password
-------------+------------------------------------
 Chloe_Doyle | $1$vZogioVv$/WNfK636saKrByeZ9T6YV0
 Test2       | $1$3yoPc9Gh$8bTbb9CR4OzX3QiwVRYpz0
(2 rows)
```

```
postgres=# select * from products;
 id | prod_name | price
----+-----------+-------
  1 | Book1     |  5.99
  2 | Book2     |  7.99
  3 | Book3     |  3.99
  4 | Book4     | 10.99
  5 | Book5     | 11.99
(5 rows)

postgres=#
```

Part 2

Getting a JWT token for user Chloe_Doyle and password password.



This is the code run to create the token which expires after 24 hours.

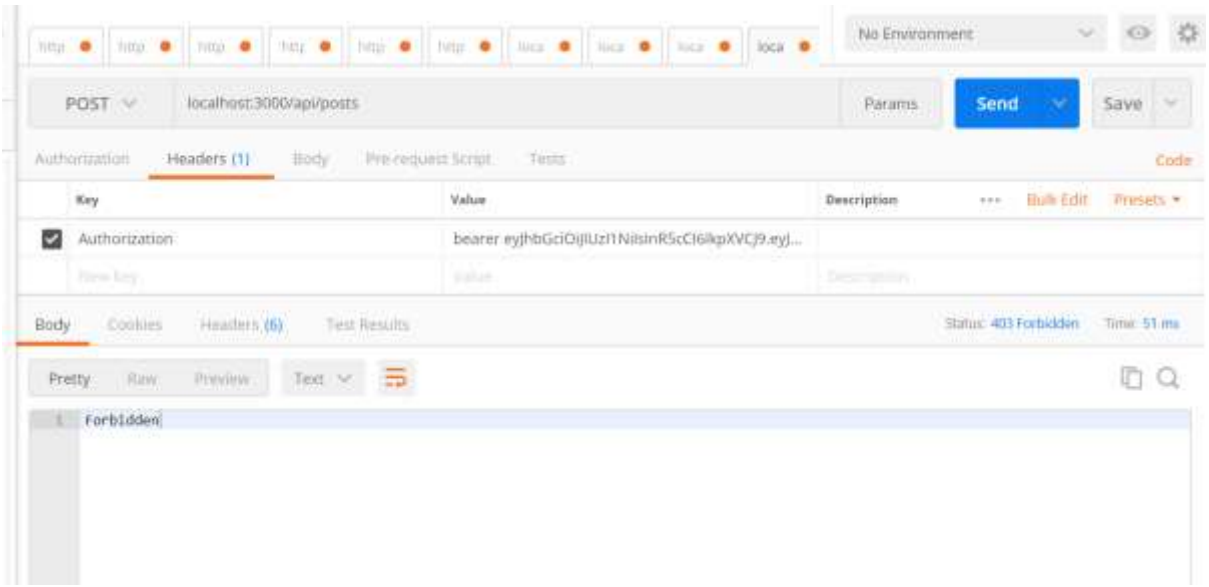
```
app.post('/api/login', (request,response)=>
{
  let username = "Chloe_Doyle"
  let password = "password"

//sequelize.query("SELECT * FROM lab2.users WHERE username = 'Chloe_Doyle' AND password= lab2.crypt('password', password);")
sequelize.query("SELECT * FROM lab2.users WHERE username = '"+ username + "' AND password= lab2.crypt('" +password +"', password);")
.then(items =>
        {
          if(items[1].rowCount != 0)
          {
            let username = items[0].username;
            let password = items[0].password;
            const user =
          {
            username,
            password
          }
            jwt.sign({user:user}, 'verysecretkey', {expiresIn: '24h'},(error,token)=>
            {
              response.json(
                {
                  token
                });
            });
          } else{response.json('fail');}
        })
        .catch(error => console.log(error));
});
```

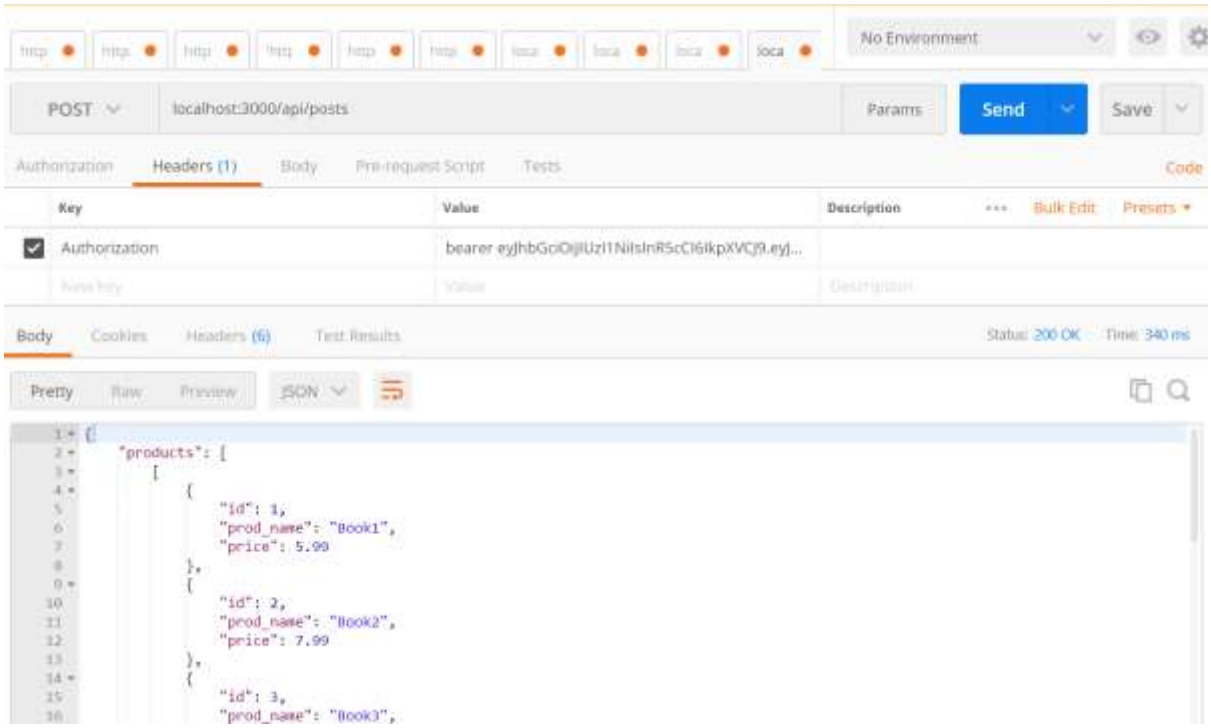
I then verify the token using the following code

```
//authorization:Bearer <token_given>
function verifyToken(request, response, next)
{
  const bearerHeader = request.headers['authorization'];
  if (typeof bearerHeader !== 'undefined')
  {
    const bearer = bearerHeader.split(' ');
    const bearerToken = bearer[1];
    request.token = bearerToken;
    next();
  }
  else
  {
    response.sendStatus(403);
  }
}
```

If the token is not valid, and error 403 is thrown,.



When a valid token is put in. This is the result, of being able to view the products table.




```
app.post('/api/posts', verifyToken, (request, response)=>
{
  jwt.verify(request.token, 'verysecretkey', (error, Auth_Data)=>
  {
      if(error)
      {
        response.sendStatus(403);
      }
      else
        {
          sequelize.query("SELECT * FROM lab2.products")
          .then(products =>
          {
            response.json
            ({
              products,
              Auth_Data})
          })

      }
  });

});
```

Part 3

I extended the user table to a new apikeys table, which inherits from users.

```
CREATE TABLE apikeys
(
  access_key bit(160),
  secretkey bit(320)
) INHERITS (users);
```

```
postgres=# \d apikeys
       Table "lab2.apikeys"
   Column   |   Type   | Modifiers
------------+----------+-----------
 username   | text     |
 password   | text     |
 access_key | bit(160) |
 secretkey  | bit(320) |
Inherits: users
```

Part 4

Created a client.js file. Signature hmac is created, and the secret key is verysecretkey. The signature Is then passed to the server through the header.

```
const hmac = crypto.createHmac('sha256', secret).digest("hex");
const request = require("request")
//hmac question - part 4

var header =
{
    'x-signature': hmac
};
```

Then in the server,the signature is checked to see if it is the one created by the header is the same as the one sent by the client. If it is the same, 200 message is sent, if not error 403.

```
var header =
{
    'x-signature': hmac
};

var options =
{
    url: 'http://localhost:3000/api/hmac',
    method:'GET',
    headers: header,
}

request(options, function (error, response, body) {
    if (!error && response.statusCode == 200)
    {
        console.log(body);
    }
    else
    {
      console.log(error);
    }
})
```

```
PS C:\Users\Chooolie\Documents\Final year - semester 2\EAD\Lab2> node .\client.js
OK
PS C:\Users\Chooolie\Documents\Final year - semester 2\EAD\Lab2>
```

This shows OK for a response from the client, and an error if not

```
app.get('/api/hmac', (request,response) =>
{
  clients_signature = request.headers['x-signature'];
  const secret = 'verysecretkey';
  servers_signature = crypto.createHmac("sha256", secret).digest("hex");

  if(clients_signature === servers_signature)
  {
    response.sendStatus(200);
    console.log("SUCCESS");
  }
  else
  {
    response.sendStatus(403);
    console.log("FAIL");
  }
})
```