Enterprise Application Development

LAB 02

David O'Neill

C15737551

Table of Contents

Question One

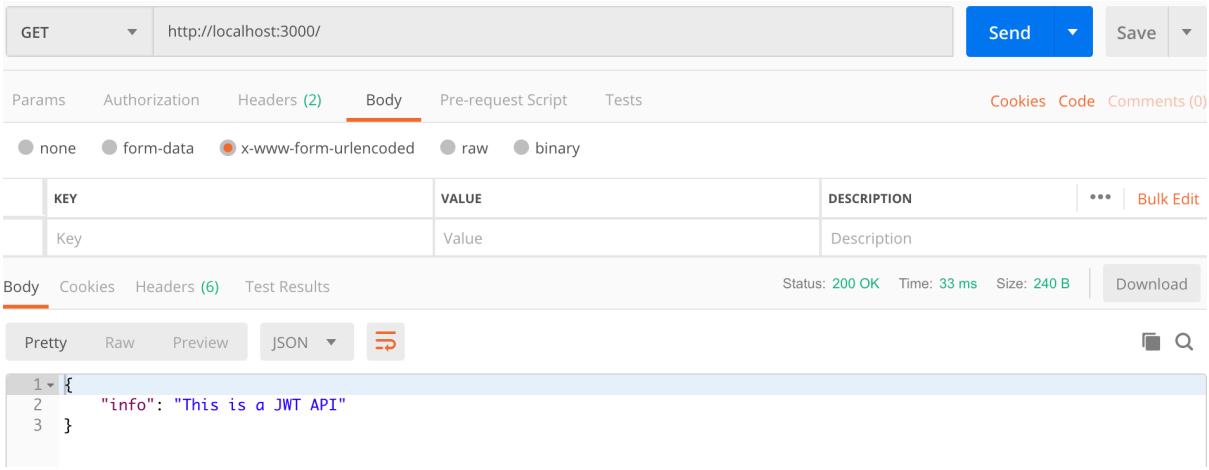
Created tables and populated them:

```
postgres=# CREATE DATABASE EAD2;
CREATE DATABASE
```

```
ead2=# CREATE EXTENSION PGCRYPTO;
CREATE EXTENSION
ead2=# CREATE TABLE users(
ead2(# id SERIAL PRIMARY KEY,
ead2(# username TEXT NOT NULL,
ead2(# password TEXT NOT NULL);
CREATE TABLE
ead2=# INSERT INTO users(
ead2(# username,password)VALUES(
ead2(# 'daveodit',
ead2(# crypt('password',gen_salt('bf')));
INSERT 0 1
ead2=# INSERT INTO users(
username,password)VALUES(
'daveoneill',
crypt('passhello',gen_salt('bf')));
INSERT 0 1
ead2=#
```

```
ead2=# SELECT * FROM users;
 id |  username  |                          password
----+------------+------------------------------------------------------------
  1 | daveodit   | $2a$06$rOrHLvPd1dyzQIHTzfsgC.LXjhNtdsojvqIGa56HTVBTPsyvI/DOK
  2 | daveoneill | $2a$06$euSMkxYBlPT6LNGVBwzDnOLP.847orJwIWWBLKCXltK6febwM2naS
(2 rows)
```

```
[ead2=# CREATE TABLE products(
[ead2(# id SERIAL PRIMARY KEY,
[ead2(# title TEXT NOT NULL,
[ead2(# price NUMERIC NOT NULL);
CREATE TABLE
[ead2=# INSERT INTO products(title,price) VALUES(
[ead2(# 'MacBook Pro',1000.50),(
[ead2(# 'Windows PC',850.99),(
[ead2(# 'Lemovo', 500.22),(
[ead2(# 'ChromeBook',700.99),(
[ead2(# 'MacBook Pro 2019',2199.99);
INSERT 0 5
[ead2=# SELECT * FROM products
[ead2-# ;
 id |       title        |  price
----+--------------------+---------
  1 | MacBook Pro        | 1000.50
  2 | Windows PC         |  850.99
  3 | Lemovo             |  500.22
  4 | ChromeBook         |  700.99
  5 | MacBook Pro 2019   | 2199.99
(5 rows)
```



GET localhost:3000/users

No Environment

GET localhost:3000/users Send

Params Authorization Headers Body Pre-request Script Tests Cookies Code

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 35 ms Size: 449 B

Pretty Raw Preview JSON


```
{
    "status": 200,
    "users": [
        {
            "id": 1,
            "username": "daveodit",
            "password": "$2a$06$rOrHLvPd1dyzQIHTzfsgC.LXjhNtdsojvqIGa56HTVBTPsyvI/DOK"
        },
        {
            "id": 2,
            "username": "daveoneill",
            "password": "$2a$06$euSMkxYBlPT6LNGVBwzDnOLP.847orJwIWWBLKCXltK6febwM2naS"
        }
    ]
}
```


You can also create a user through here rather than in postgres command line:

```
app.post('/users', (req, res) => {
    let query
    query = req.app.get('db').createUser([req.body.username, req.body.password],
        function(err, result) { if (err) { return next(err); } });
    return query.then(username => res.json(username))
});
```

View protected products table while authenticated:

http://localhost:3000/products

POST http://localhost:3000/products Send Save

Params Authorization Headers (3) Body Pre-request Script Tests Cookies Code Comments (0)

none form-data x-www-form-urlencoded raw binary

KEY	VALUE	DESCRIPTION
☑ username	tester	
☑ password	password	
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 65 ms Size: 454 B Download

Pretty Raw Preview JSON


```
[
    {
        "id": 1,
        "title": "MacBook Pro",
        "price": "1000.50"
    },
    {
        "id": 2,
        "title": "Windows PC",
        "price": "850.99"
    },
```

```
app.post('/products', (req, res) => {
    let query
    query = req.app.get('db').authUser([req.body.username, req.body.password],
        function(err, result) { if (err) { return next(err); } });
    query.then(username => {
        if (username === undefined) {
            return res.status(401).send('INVALID')
        } else {
            req.app.get('db').products.find({}, {}).then(products => {
                res.json(products);
            });
```

SQL used in Q.1;

```
SELECT username
  FROM "users"
 WHERE username = lower($1)
       AND password = crypt($2, password);
```

```
INSERT INTO "users"(username, password)
     VALUES (lower($1), crypt($2, gen_salt('bf', 8)))
  RETURNING username;
```

Question Two

Verifying a user and giving them a JWT Token:

```
/ Q.2
app.post("/login", (req, res) => {
    let query
    query = req.app.get('db').authUser([req.query.username, req.query.password],
        function(err, result) { if (err) { return next(err) } })
    query.then(username => {
        if (username === undefined) {
            return res.status(401).send('INVALID')
        } else {
            var token = jwt.sign({ username: req.query.username }, config.secret, { expiresIn: "20h" })
            res.status(200).json({
                message: "Authorised",
                username: req.query.username,
                expiresIn: "20h",
                token
            })
        }
    })
```

http://localhost:3000/login?username=tester&password=password

POST http://localhost:3000/login?username=tester&password=password Send Save

Params Authorization Headers (2) Body Pre-request Script Tests Cookies Code Comments (0)

KEY	VALUE	DESCRIPTION
☑ username	tester	
☑ password	password	
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 34 ms Size: 441 B Download

Pretty Raw Preview JSON


```
{
    "message": "Authorised",
    "username": "tester",
    "expiresIn": "20h",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJ1c2VybmFtZSI6InRlc3RlciIsImlhdCI6MTU1MTYzNzk2MiwiZXhwIjoxNTUxNzA5OTYyfQ
        .ApY_Gx6rwpkl1y5CkSm68KOidPizP3Gi1hPtLHYE_8"
}
```


JWT Token Decoded:

JWT Debugger Libraries Introduction Ask Get a T-shirt! Crafted by Auth0

Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InRlc3RlciIsImlhdCI6MTU1MTYzNzk2MiwiZXhwIjoxNTUxNzA5OTYyfQ.ApY_Gx6rwpkl1y5CkSm68KOidPizP3Gi1hPtLHYE_8

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

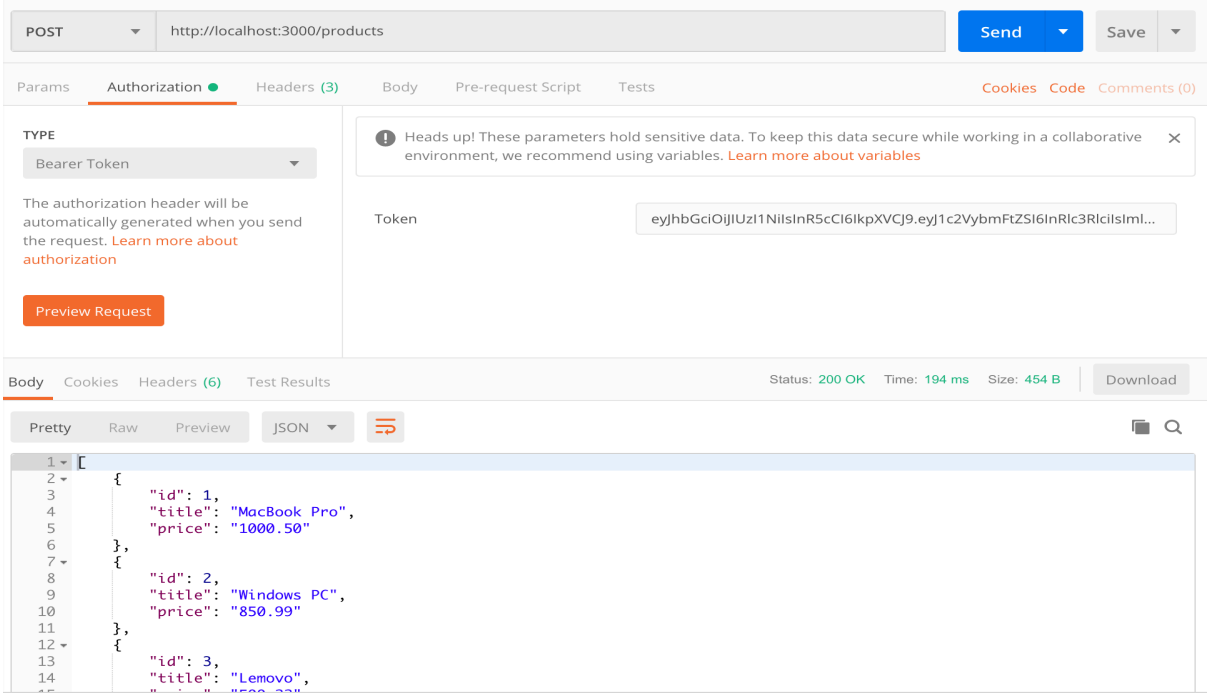

PAYLOAD: DATA


```
{
  "username": "tester",
  "iat": 1551637962,
  "exp": 1551709962
}
```


VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) secret base64 encoded
```

Protected Resource Products Table:



Config.js file holds the secret key:

```
module.exports = {
    secret: 'secretykey'
};
```

Bearer Token created with the function:

```
function authenticate(req, res, next) {
    var bearerHeader = req.header["authorization"]
    if (typeof bearerHeader !== 'undefined') {
        const bearer = bearerHeader.split("")
        const BearerToken = bearer[1]

        jwt.verify(BearerToken, config.secret, function(err, decoded) {
            if (!err) {
                console.log("Success")
                next()
            } else {
                res.status(401).send("Authentication failed")
            }
```

Applying it to the users table too:

```
app.post("/products/", authenticate, (req, res) => {
    req.app.get('db').products.find({}, {}).then(products => {
        res.sendStatus(200).json(products)
    })
})
app.post("/users", authenticate, (req, res) => {
    req.app.get('db').users.find({}, {}).then(users => {
        res.sendStatus(200).json(users)
    })
})
```

Question Three

Dropped current users table and create and new one that includes two new fields:

```
[ead2=# DROP TABLE users;
DROP TABLE
```

```
[ead2=# CREATE TABLE users(
id SERIAL PRIMARY KEY,
username TEXT NOT NULL,
password TEXT NOT NULL,
access_key varchar(40),
secret_key varchar(80));
CREATE TABLE
```

Access_Key, Secret_Key on new users table

```
[ead2=# INSERT INTO users (username, password, access_key, secret_key) VALUES ('te
stuser', crypt('password', gen_salt('bf', 8)), '4dje2e45sh7d6b4d8b3647f499b7b1b44
5176s5w' ,'h480bk5f34e27c6257hebsh10ed0403dc66c0a27fh1788f6869abb6756a55w78138bcd
fe03j84725');
INSERT 0 1
```

Sample New users Table:

```
 id | username |                          password                          |               access_key               |                                secret_key
----+----------+------------------------------------------------------------+----------------------------------------+----------------------------------------------------------------
---------------------
  1 | testuser | $2a$08$yVUQi6GBOx/fa8oFOkBDSOaFPeNrKkYG14vBJf7Qv3GIPgljnU3he | 4dje2e45sh7d6b4d8b3647f499b7b1b445176s5w | h480bk5f34e27c6257hebsh10ed0403dc66c0a27fh1788f6869abb6756
a55w78138bcdfe03j84725
```

Question Four

Implemented the HMAC and the client and then calling it with the following Command and returning the protected products table:

```
Davids-MacBook-Pro:lab2-1jwt davidoneill$ node client.js -X POST -a 4dje2e45sh7d
6b4d8b3647f499b7b1b445176s5w -s h480bk5f34e27c6257hebsh10ed0403dc66c0a27fh1788f6
869abb6756a55w78138bcdfe03j84725 -d ead2 http://localhost:3000/products
200
[ { id: 1, title: 'MacBook Pro', price: '1000.50' },
  { id: 2, title: 'Windows PC', price: '850.99' },
  { id: 3, title: 'Lemovo', price: '500.22' },
  { id: 4, title: 'ChromeBook', price: '700.99' },
  { id: 5, title: 'MacBook Pro 2019', price: '2199.99' } ]
```

HMAC and Client Code:

```
function authenmac(req, res, next) {
    header = req.headers.authorization;
    key = header.slice(16, 26);
    signature = header.slice(37, );
    db.query(
        "SELECT access_key, secret_key from users where access_key = '" + access_key + "';"
    ).then(users => {
        access_key = users[0].access_key;
        secret_key = users[0].secret_key;
        url = "http://" + req.headers.host + req.url;
        body = req.body.value;
        const data = `${url}${body}${access_key}`;
        const signature = CryptoJS.HmacSHA256(data, secret_key);
        hmachead = `HMAC-SHA256 Key=${access_key} Signature=${Base64.stringify(signature)}`;
        if (hmachead == req.headers.authorization) {
            next();
        } else {
            res.status(401)
        }
    })
}
```

```
const axios = require("axios");
const argv = require("minimist")(process.argv.slice(2));
const CryptoJS = require("crypto-js");
const Base64 = require("crypto-js/enc-base64");

axios(argv._[0], {
    method: argv.X.toLowerCase(),
    headers: { Authorization: token() },
    data: {
        value: argv.d
    }
}).then(function(res) {}).catch(error => console.log(error));

function token() {
    const data = `${argv._}${argv.d}${argv.a}`;
    const signature = CryptoJS.HmacSHA256(data, argv.s);
    return `HMAC-SHA256 Key=${argv.a} Signature=${Base64.stringify(signature)}`;
}
```