ENTERPRISE APPLICATION DEVELOPMENT – LAB 2

C15478448

ENTERPRISE APPLICATION DEVELOPMENT

COURSE: DT228/4

Lab 2

Table of Contents

Folder Structure

Authentication – Contains All Parts (1-4)

Document – This Document in .docx and .pdf

Screenshots – Screenshots for all Outputs

Note:

- To run this project you must enter in the following command in the **Authentication** folder: *npm install*

- After installing the required modules based on the package, run the following command: *npm start*

- Enter: *localhost:3000* into a browser for relevant links to showcase each part.

Problem Sets

Part 1 – Basic Authentication

Implement a users table having a username and hashed password fields. Use the postgresql crypt() and gen_salt() functions to implement the password hashing Implement a protected resource table (e.g. a "products" table) to which you can use to demonstrate your authentication features.

The user's table was created with the following steps:

1.CREATE EXTENSION pgcrypto;

2.SELECT gen_random_uuid(); (To check if a random UUID is generated)

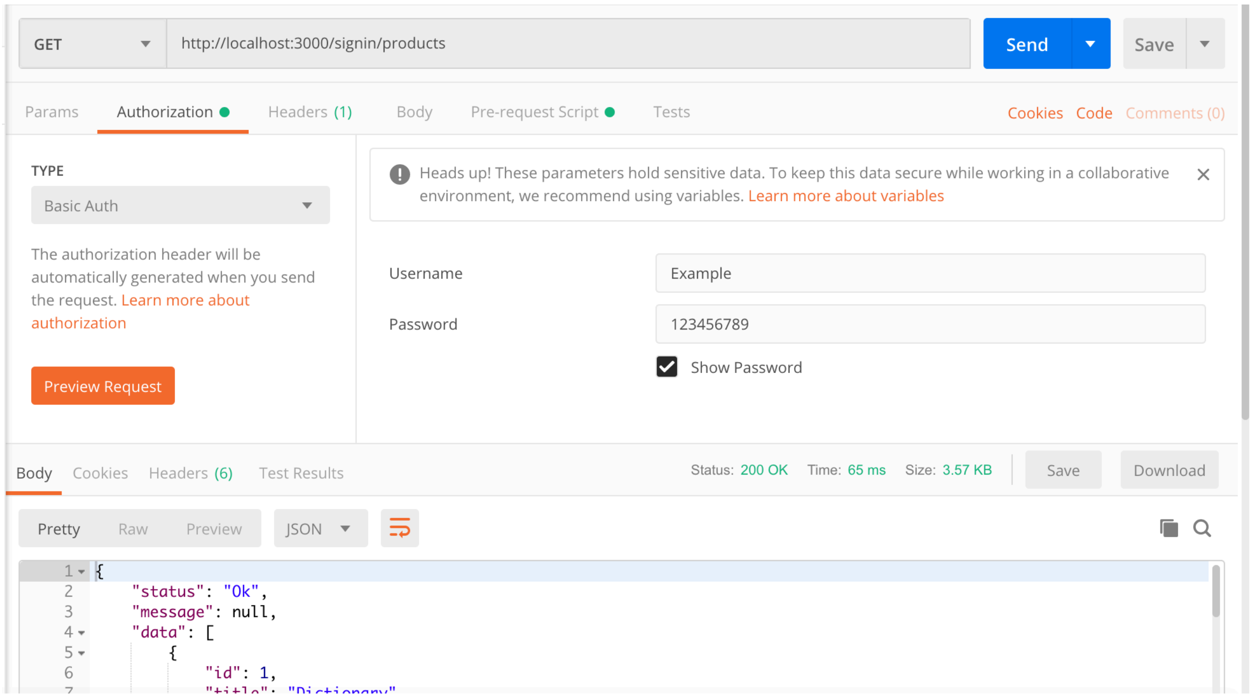
```
3.CREATE TABLE public.users (
        id uuid NOT NULL DEFAULT gen_random_uuid() PRIMARY KEY,
        email text NOT NULL,
        username text NOT NULL,
        password text NOT NULL,
        key text NOT NULL DEFAULT key_gen(20), (Part 3)
        secret_key text NOT NULL DEFAULT key_gen(40) (Part 3)
);
```

4. INSERT INTO users (email, username, password)

VALUES ('example@example.com', 'Example', crypt('123456789', gen_salt('bf', 8)));

After creating the table and inserting an entry into it as we need a user to access a protected resources which in my case I have used the Products table.

Screenshot are displayed on the next page.


GET http://localhost:3000/signin/products

Authorization — TYPE: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization

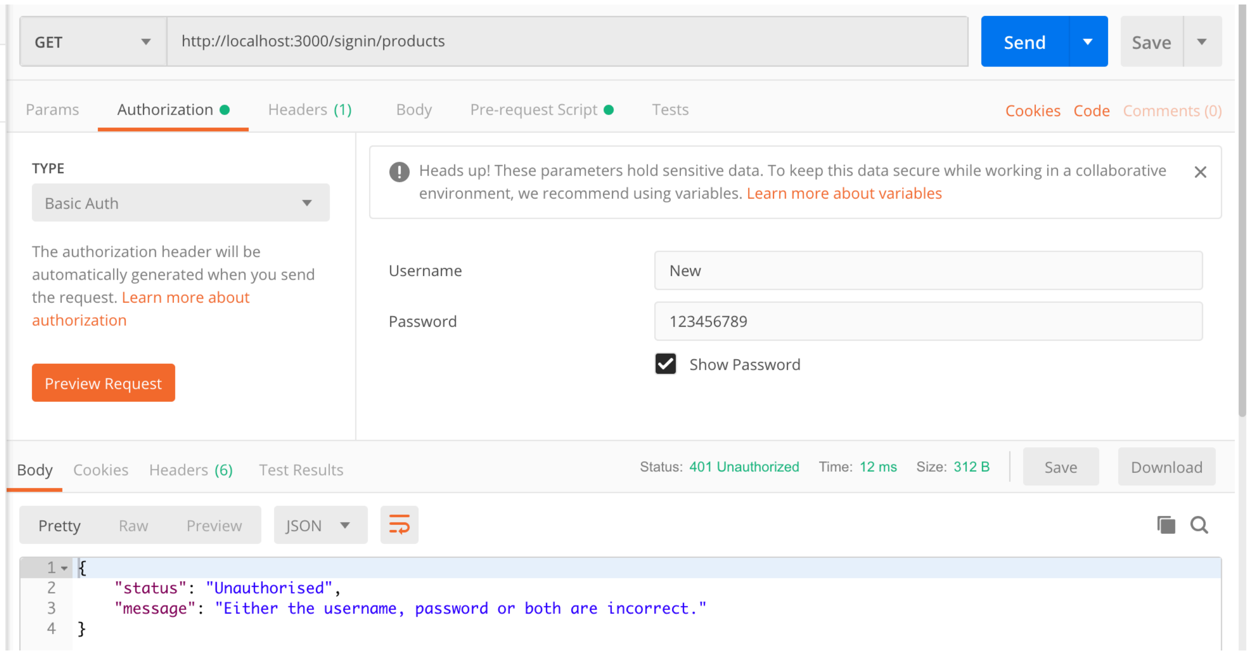
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Username: Example
Password: 123456789
☑ Show Password

Status: 200 OK Time: 65 ms Size: 3.57 KB

```
{
    "status": "Ok",
    "message": null,
    "data": [
        {
            "id": 1,
            "title": "Dictionary"
```

As you can see, passing in the correct username and password the contents of the Product table will be displayed.


GET http://localhost:3000/signin/products

Authorization — TYPE: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Username: New
Password: 123456789
☑ Show Password

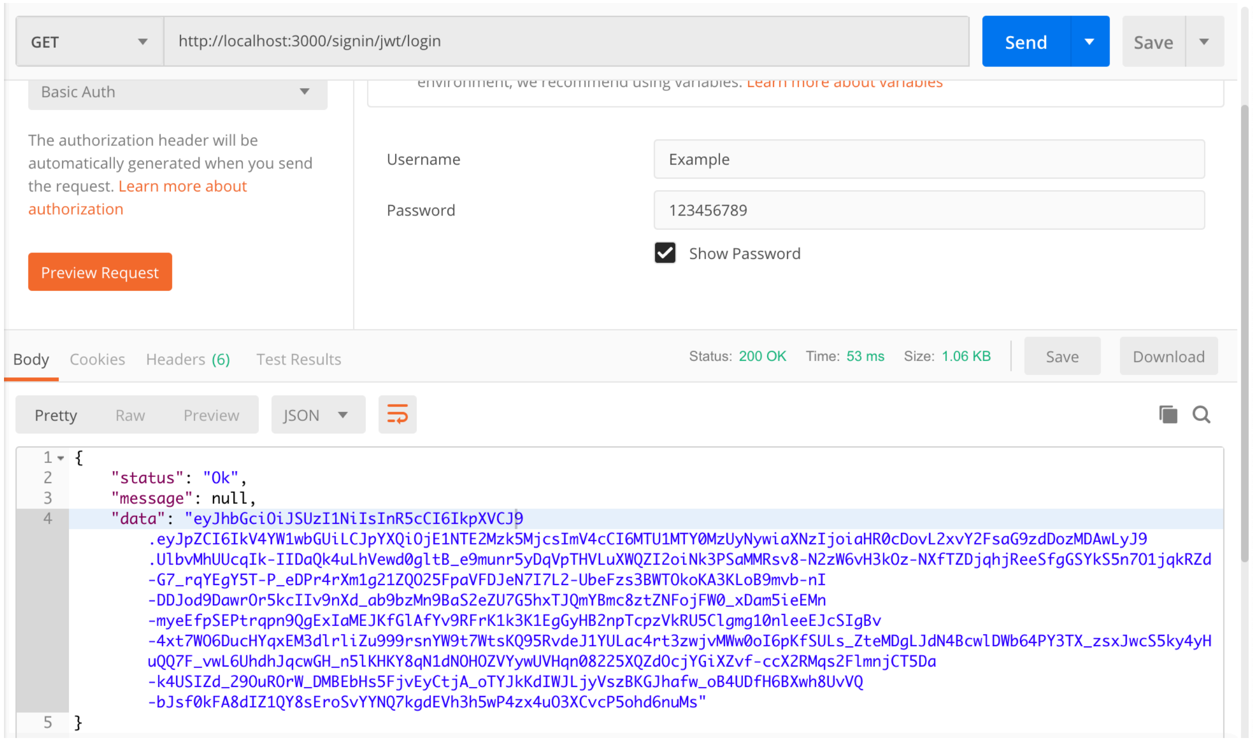
Status: 401 Unauthorized Time: 12 ms Size: 312 B

```
{
    "status": "Unauthorised",
    "message": "Either the username, password or both are incorrect."
}
```

Entering the incorrect details to access the desired resources will return a status code of 401 which is unauthorised.

Part 2 – JWT

Implement a JWT-secured version of the API based on the users table from the previous step...



http://localhost:3000/signin/jwt/login

The authorization header will be automatically generated when you send the request. Learn more about authorization

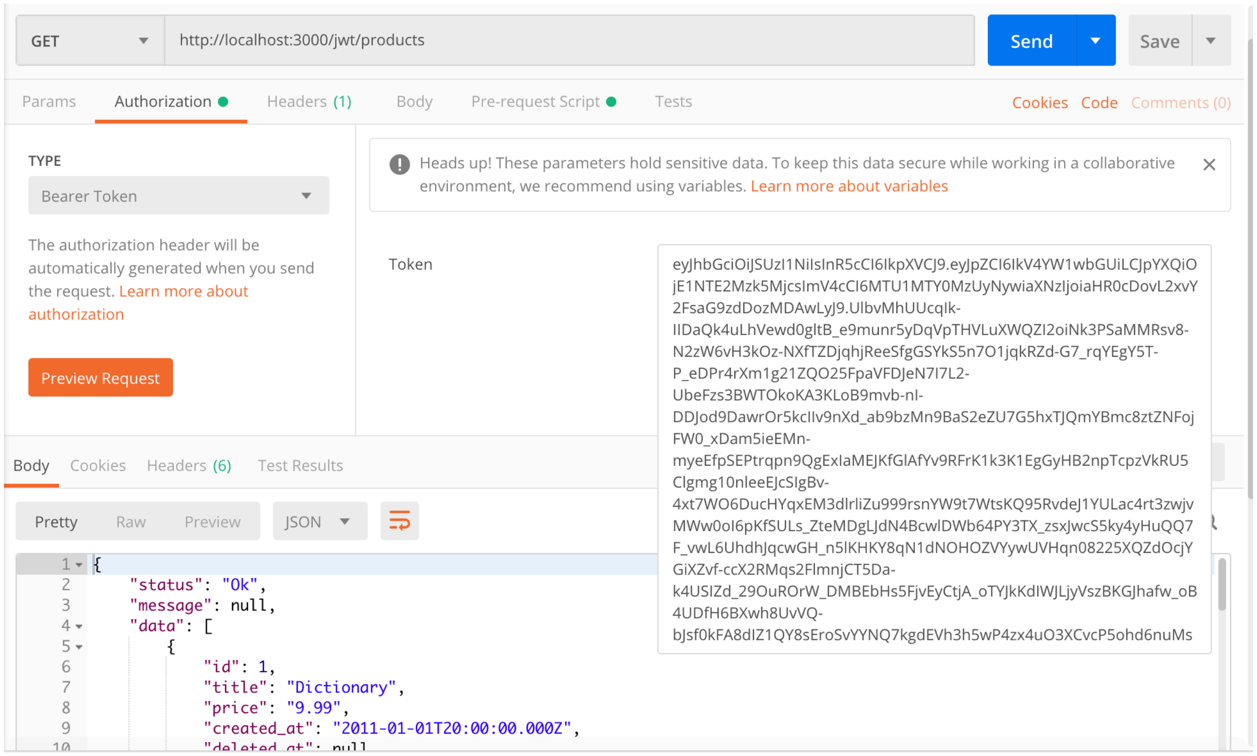
Username: Example
Password: 123456789
Show Password ✓

Status: 200 OK Time: 53 ms Size: 1.06 KB

```
{
    "status": "Ok",
    "message": null,
    "data": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJpZCI6IkV4YW1wbGUiLCJpYXQiOjE1NTE2Mzk5MjcsImV4cCI6MTU1MTY0MzUyNywiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDozMDAwLyJ9
        .UlbvMhUUcqIk-IIDaQk4uLhVewd0gltB_e9munr5yDqVpTHVLuXWQZI2oiNk3PSaMMRsv8-N2zW6vH3kOz-NXfTZDjqhjReeSfgGSYkS5n7O1jqkRZd
        -G7_rqYEgY5T-P_eDPr4rXm1g21ZQO25FpaVFDJeN7I7L2-UbeFzs3BWTOkoKA3KLoB9mvb-nI
        -DDJod9DawrOr5kcIIv9nXd_ab9bzMn9BaS2eZU7G5hxTJQmYBmc8ztZNFojFW0_xDam5ieEMn
        -myeEfpSEPtrqpn9QgExIaMEJKfGlAfYv9RFrK1k3K1EgGyHB2npTcpzVkRU5Clgmg10nleeEJcSIgBv
        -4xt7WO6DucHYqxEM3dlrliZu999rsnYW9t7WtsKQ95RvdeJ1YULac4rt3zwjvMWw0oI6pKfSULs_ZteMDgLJdN4BcwlDWb64PY3TX_zsxJwcS5ky4yH
        uQQ7F_vwL6UhdhJqcwGH_n5lKHKY8qN1dNOHOZVYywUVHqn08225XQZdOcjYGiXZvf-ccX2RMqs2FlmnjCT5Da
        -k4USIZd_29OuROrW_DMBEbHs5FjvEyCtjA_oTYJkKdIWJLjyVszBKGJhafw_oB4UDfH6BXwh8UvVQ
        -bJsf0kFA8dIZ1QY8sEroSvYYNQ7kgdEVh3h5wP4zx4u03XCvcP5ohd6nuMs"
}
```

Similar to part 1 where the username and password has been supplied to view the desired resources although this time a token system is being used.

The first step is to login and retrieve a token, once the login has been a success a token will be generated, and you may use this token to access the desired resources which in my case is the contents of the Products table.



GET http://localhost:3000/jwt/products

TYPE: Bearer Token

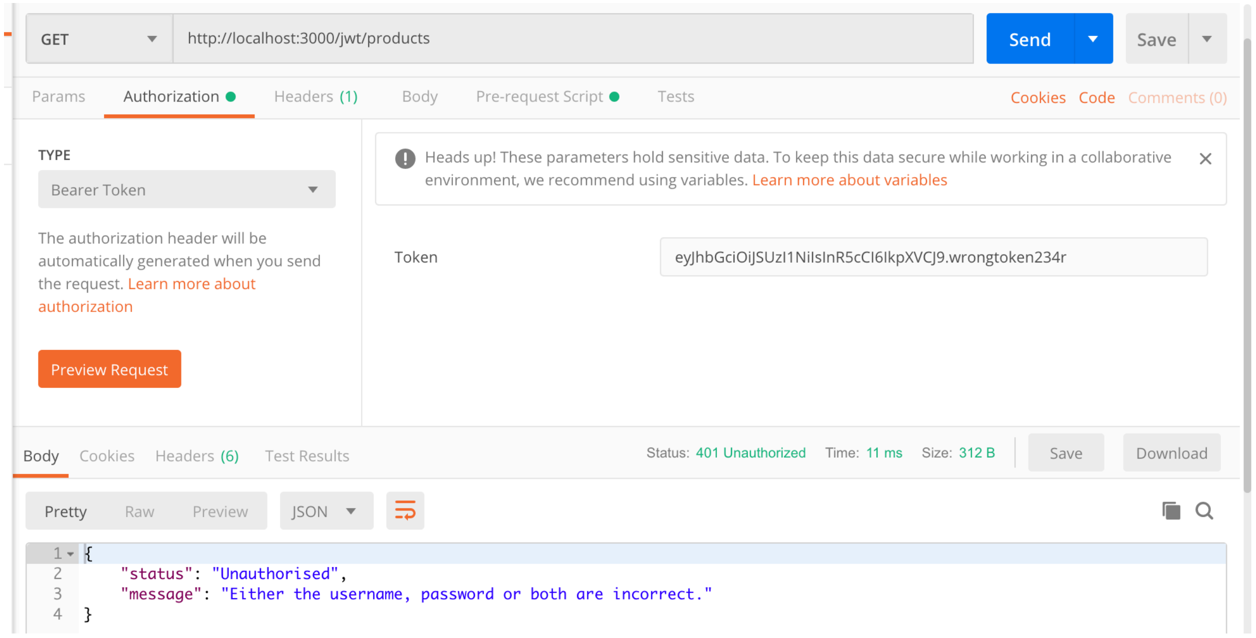
The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IkV4YW1wbGUiLCJpYXQiOjE1NTE2Mzk5MjcsImV4cCI6MTU1MTY0MzUyNywiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDozMDAwLyJ9.UlbvMhUUcqIk-IIDaQk4uLhVewd0gltB_e9munr5yDqVpTHVLuXWQZI2oiNk3PSaMMRsv8-N2zW6vH3kOz-NXfTZDjqhjReeSfgGSYkS5n7O1jqkRZd-G7_rqYEgY5T-P_eDPr4rXm1g21ZQO25FpaVFDJeN7I7L2-UbeFzs3BWTOkoKA3KLoB9mvb-nI-DDJod9DawrOr5kcIlv9nXd_ab9bzMn9BaS2eZU7G5hxTJQmYBmc8ztZNFojFW0_xDam5ieEMn-myeEfpSEPtrqpn9QgExIaMEJKfGlAfYv9RFrK1k3K1EgGyHB2npTcpzVkRU5Clgmg10nleeEJcSIgBv-4xt7WO6DucHYqxEM3dlrliZu999rsnYW9t7WtsKQ95RvdeJ1YULac4rt3zwjvMWw0oI6pKfSULs_ZteMDgLJdN4BcwlDWb64PY3TX_zsxJwcS5ky4yHuQQ7F_vwL6UhdhJqcwGH_n5lKHKY8qN1dNOHOZVYywUVHqn08225XQZdOcjYGiXZvf-ccX2RMqs2FlmnjCT5Da-k4USIZd_29OuROrW_DMBEbHs5FjvEyCtjA_oTYJkKdIWJLjyVszBKGJhafw_oB4UDfH6BXwh8UvVQ-bJsf0kFA8dIZ1QY8sEroSvYYNQ7kgdEVh3h5wP4zx4uO3XCvcP5ohd6nuMs

```
{
    "status": "Ok",
    "message": null,
    "data": [
        {
            "id": 1,
            "title": "Dictionary",
            "price": "9.99",
            "created_at": "2011-01-01T20:00:00.000Z",
            "deleted_at": null
```

Now, the token is copied, and the type of authorization is used is the Bearer Token instead of the Basic Auth. Once the correct token has been entered and it is correct the contents of the Product table will be able to be viewed.



GET http://localhost:3000/jwt/products

TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.wrongtoken234r

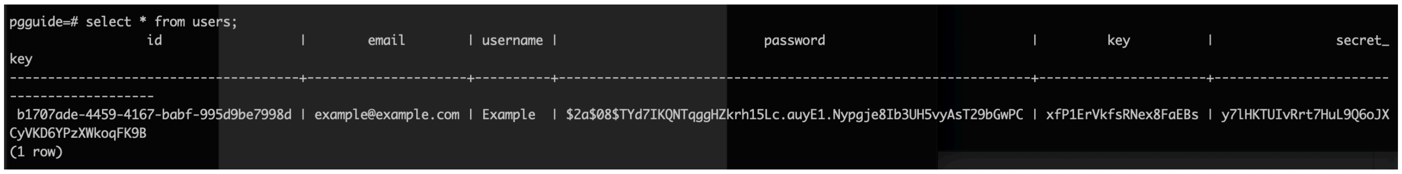
Status: 401 Unauthorized Time: 11 ms Size: 312 B

```
{
    "status": "Unauthorised",
    "message": "Either the username, password or both are incorrect."
}
```

Entering the wrong token will return an error code of 401. Access denied.

Part 3 – User’s Table Extended

Extend the users table or add another linked “apikeys” table to include an access key (160 bits) and secret key (320 bits)



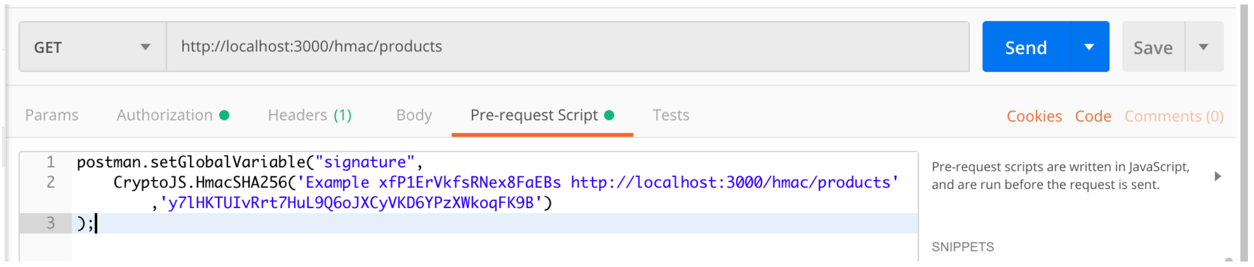
```
pgguide=# select * from users;
                  id                  |        email        | username |                           password                           |         key          |                     secret_
key
--------------------------------------+---------------------+----------+--------------------------------------------------------------+----------------------+-----------------------
-------------------
 b1707ade-4459-4167-babf-995d9be7998d | example@example.com | Example  | $2a$08$TYd7IKQNTqggHZkrh15Lc.auyE1.Nypgje8Ib3UH5vyAsT29bGwPC | xfP1ErVkfsRNex8FaEBs | y7lHKTUIvRrt7HuL9Q6oJX
CyVKD6YPzXWkoqFK9B
(1 row)
```

Upon extending the user’s table, 2 new columns have been added, the key columns (160 bits) and the secret_key (320 bits) column as instructed.

Both of these keys are auto generated upon every new entry. These keys are used for the last part. (HMAC)

Part 4 – HMAC

Implement a Hash-based message authentication (HMAC) scheme to secure the API….



GET http://localhost:3000/hmac/products

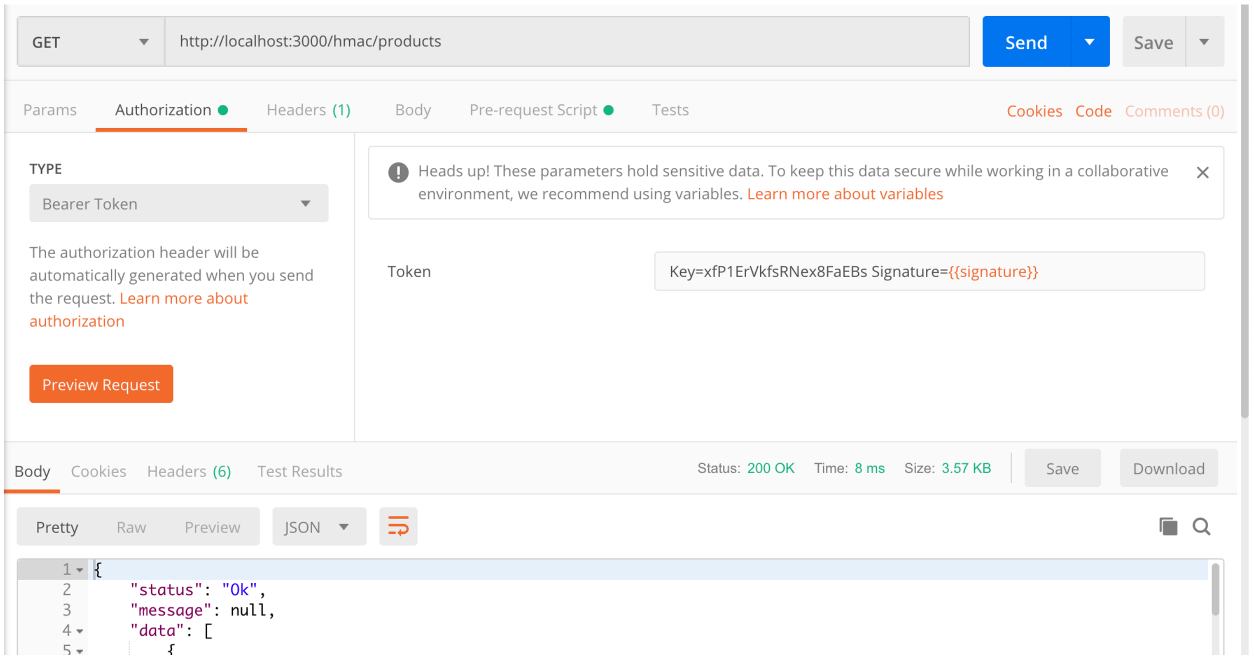
Pre-request Script

```
postman.setGlobalVariable("signature",
    CryptoJS.HmacSHA256('Example xfP1ErVkfsRNex8FaEBs http://localhost:3000/hmac/products'
        ,'y7lHKTUIvRrt7HuL9Q6oJXCyVKD6YPzXWkoqFK9B')
);
```

Pre-request scripts are written in JavaScript, and are run before the request is sent.

In order to get this to work a pre-request Script must be created to create the HMAC signature as we need both the client and server signatures for comparisons to access the desired resources.

The script consists of creating a variable called **signature** as the signature must be stored somewhere and it will be stored in the global variable **signature**. Next up the username is provided, the key and the route. Lastly, the secret key would also be provided.



GET http://localhost:3000/hmac/products

Authorization

TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

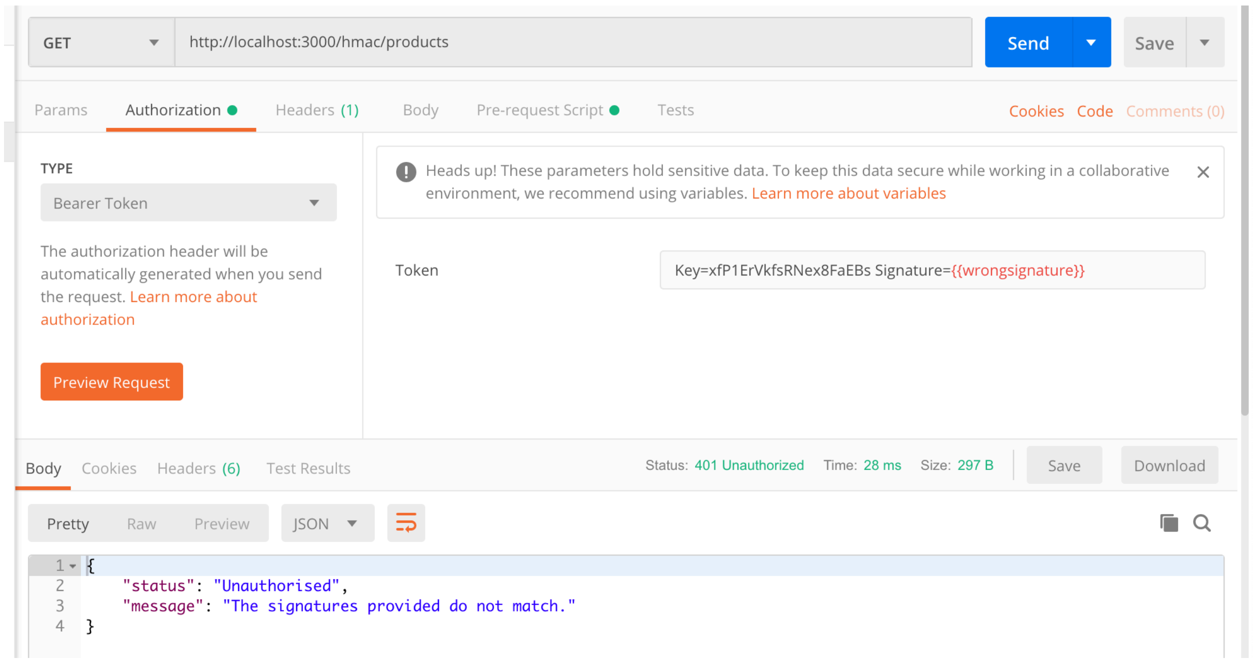
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Token: Key=xfP1ErVkfsRNex8FaEBs Signature={{signature}}

Status: 200 OK Time: 8 ms Size: 3.57 KB

```
{
    "status": "Ok",
    "message": null,
    "data": [
        {
```

With the pre-request script defined a signature is created, the client will also create a signature and upon doing so the server will take the key provided by the client and it will validate if the user and the secret key exists in the database. If the secret key exists, then it will create the same signature on the server, and it will compare the two signatures to see if they match. The screenshot above displays the happy path if both of the signatures match.



http://localhost:3000/hmac/products

TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

Token: Key=xfP1ErVkfsRNex8FaEBs Signature={{wrongsignature}}

Status: 401 Unauthorized Time: 28 ms Size: 297 B

```
{
    "status": "Unauthorised",
    "message": "The signatures provided do not match."
}
```

Above is what happening if the signatures do not match. An error code of 401 will be returned to inform of an access issue.