Part One

The users table contains an ID, username, and password. The public_key and private_key fields are used in later parts.

```
CREATE TABLE users (
ID SERIAL PRIMARY KEY,
username TEXT NOT NULL,
password TEXT NOT NULL,
public_key TEXT
private_key TEXT
);
```

In order to create a hashed password using the crypt() and gen_salt() functions the pgcrypto extension needs to be created.

```
CREATE EXTENSION pgcrypto;
```

The protected resource table is "books".

```
CREATE TABLE BOOKS (
ID serial PRIMARY KEY,
TITLE text NOT NULL,
PRICE numeric NOT NULL
);
```

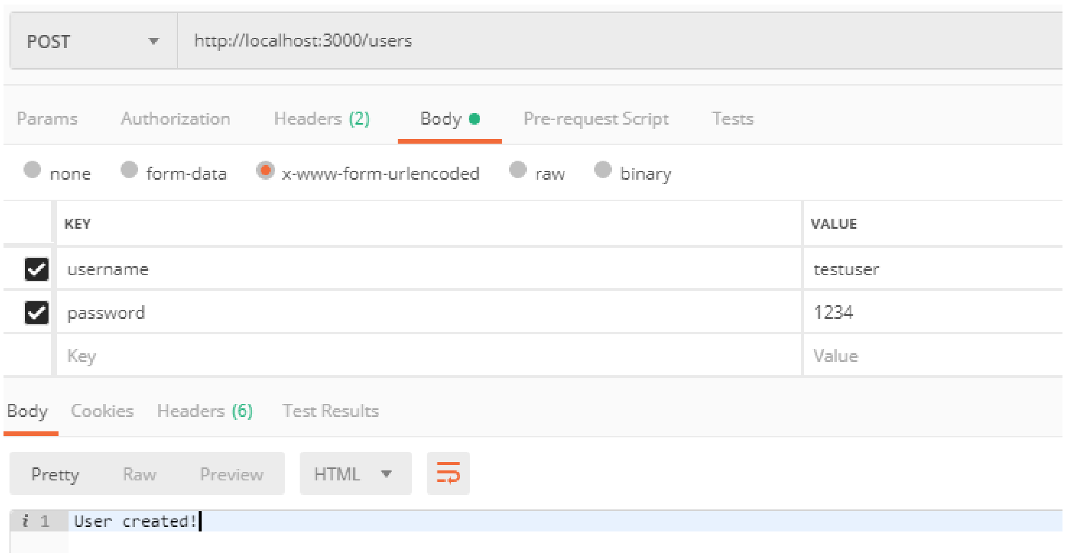
It is populated with some generic data.

```
INSERT INTO BOOKS (TITLE, PRICE) VALUES ('The Great Gatsby', 10.50);
INSERT INTO BOOKS (TITLE, PRICE) VALUES ('Ulysses', 25);
```

SQL for hashing passwords on insert:

Insert into users (username, password) values ('${user}', crypt('${pass}', gen_salt('bf', 8)));

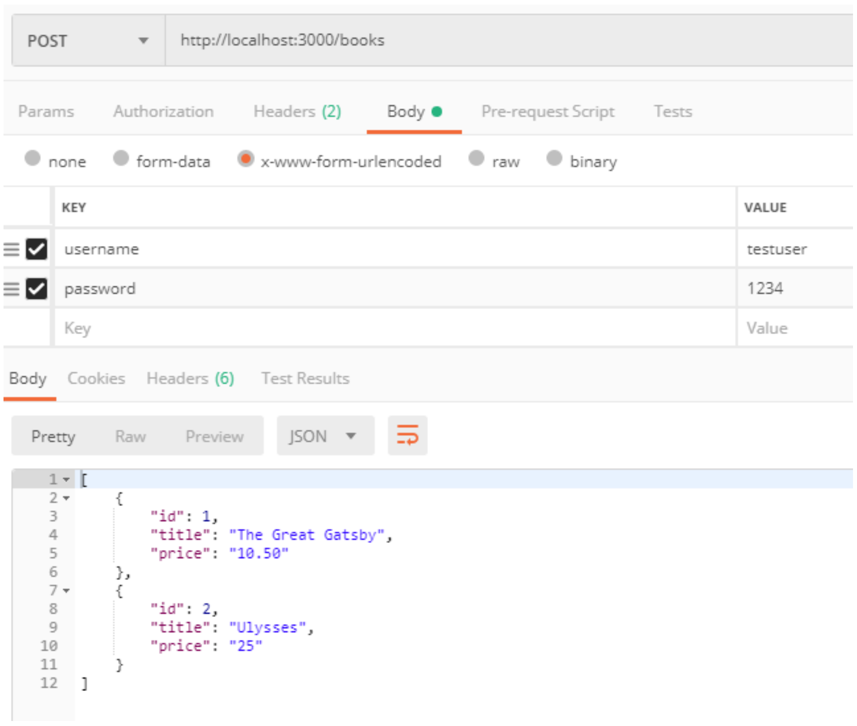
Users can be created by posting a username and password in the body to /users.



KEY	VALUE
username	testuser
password	1234
Key	Value

```
User created!
```

To access books, the username and password must be posted in the body. I did this through postman using the newly created user. The usernames and hashed passwords are compared, if there is a match the results are displayed.



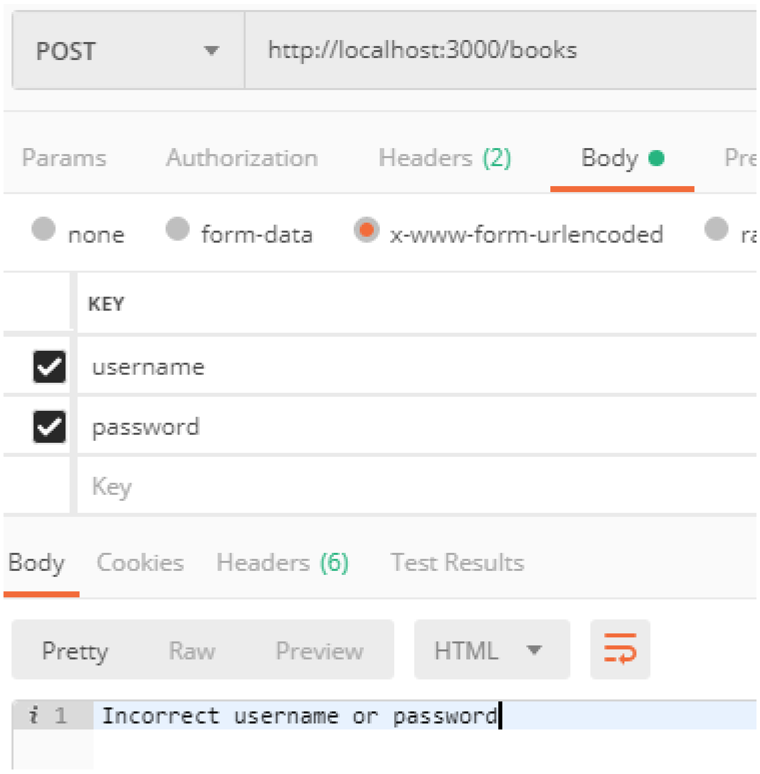
POST http://localhost:3000/books

x-www-form-urlencoded

KEY	VALUE
username	testuser
password	1234
Key	Value

```
[
    {
        "id": 1,
        "title": "The Great Gatsby",
        "price": "10.50"
    },
    {
        "id": 2,
        "title": "Ulysses",
        "price": "25"
    }
]
```

The following response is given if an incorrect password is posted.



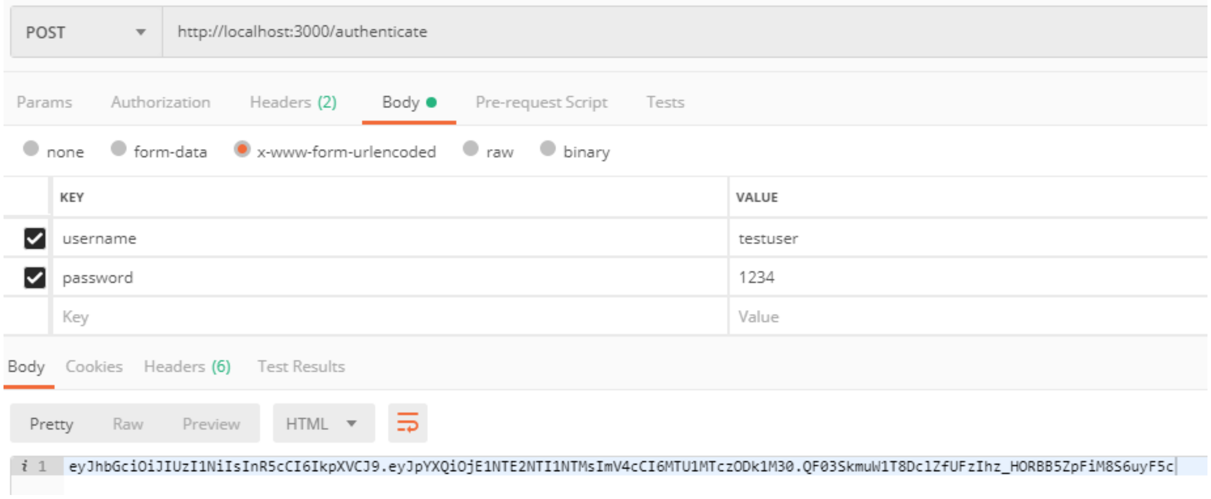
POST http://localhost:3000/books

KEY
username
password
Key

```
Incorrect username or password
```

Part Two

The /authenticate endpoint is used to validate user credentials (username and password) and to then generate a JWT token that expires in 24 hours. The token can then be used to access the protected resource /books.



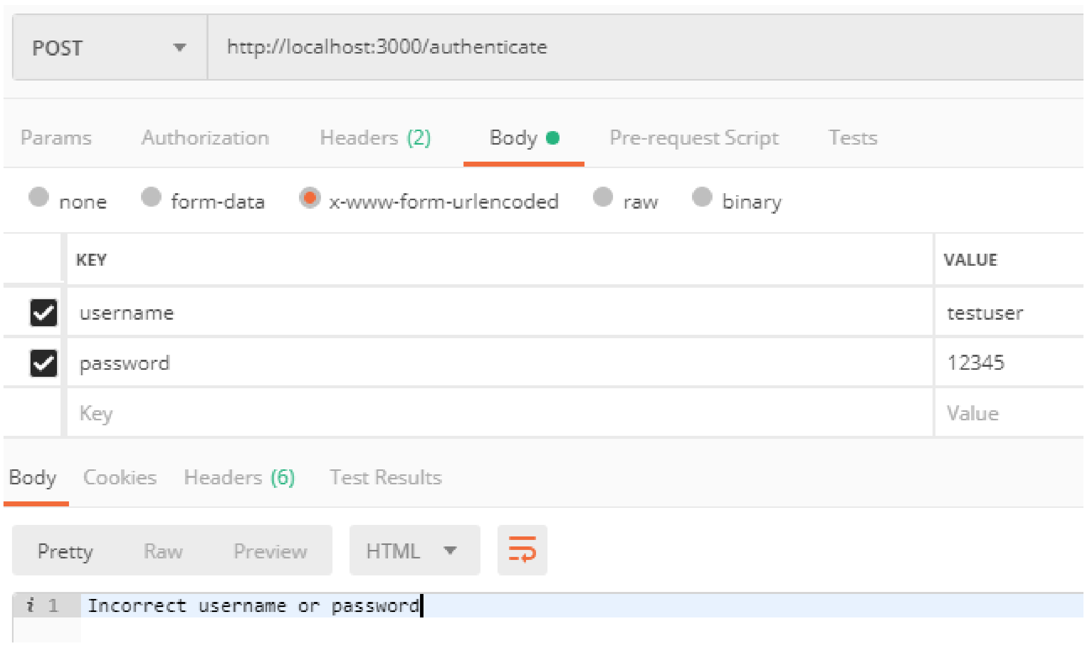
POST http://localhost:3000/authenticate

Body: x-www-form-urlencoded

	KEY	VALUE
✓	username	testuser
✓	password	1234
	Key	Value

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1NTE2NTI1NTMsImV4cCI6MTU1MTczODk1M30.QF03SkmuW1T8DclZfUFzIhz_HORBB5ZpFiM8S6uyF5c
```

Incorrect password results in :



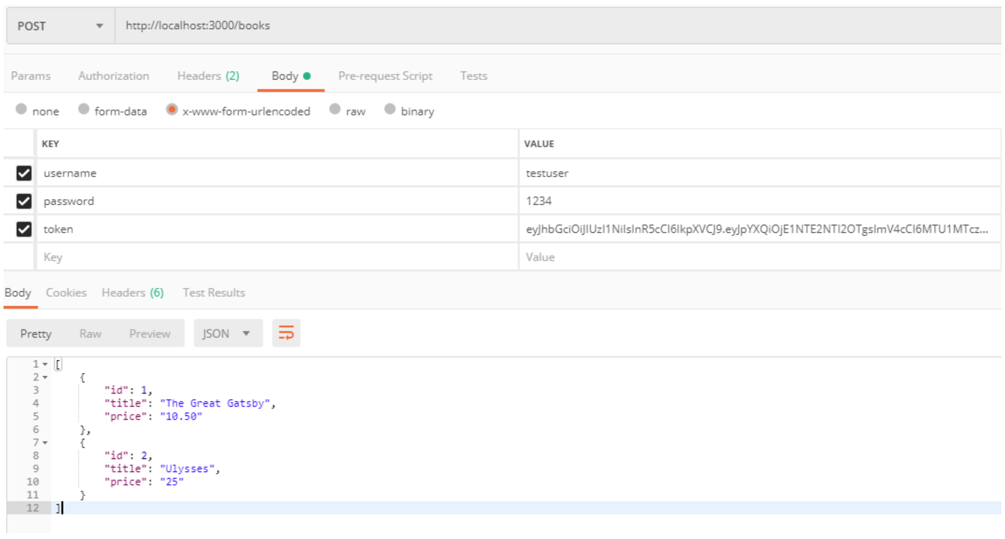
POST http://localhost:3000/authenticate

Body: x-www-form-urlencoded

	KEY	VALUE
✓	username	testuser
✓	password	12345
	Key	Value

```
Incorrect username or password
```

A valid token can then be used to access the /books resource by posting.



```
POST    http://localhost:3000/books
```

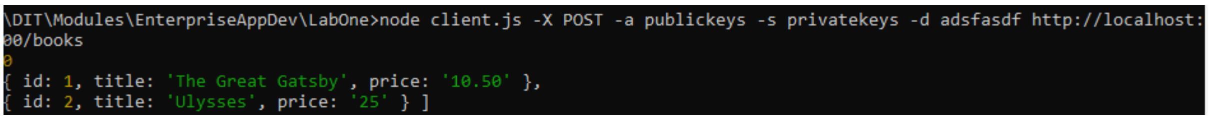
	KEY	VALUE
✓	username	testuser
✓	password	1234
✓	token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1NTE2NTI2OTgsImV4cCI6MTU1MTcz...
	Key	Value

```
[
    {
        "id": 1,
        "title": "The Great Gatsby",
        "price": "10.50"
    },
    {
        "id": 2,
        "title": "Ulysses",
        "price": "25"
    }
]
```

Part Four

The HMAC signature is created from the data and private key on the server end, and on the client. The server verifies that a user with the corresponding private key exists using the public key passed by the client. If so, the server independently generates its own signature and compares them.

Sending the client request with a known public and private key results in:



```
\DIT\Modules\EnterpriseAppDev\LabOne>node client.js -X POST -a publickeys -s privatekeys -d adsfasdf http://localhost:
00/books
0
{ id: 1, title: 'The Great Gatsby', price: '10.50' },
{ id: 2, title: 'Ulysses', price: '25' } ]
```

Incorrect keys returns "Authentication Failed"