

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.

POST http://localhost:3000/users

Params Authorization Headers (2) Body Pre-request Script Tests

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

KEY	VALUE
username	testuser
password	1234
Key	Value

Body Cookies Headers (6) Test Results

Pretty Raw Preview HTML

1 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.

The screenshot shows a Postman interface for a POST request to `http://localhost:3000/books`. The 'Body' tab is selected, and the data type is 'x-www-form-urlencoded'. The request body contains two form fields: 'username' with value 'testuser' and 'password' with value '1234'. The response is displayed in the 'Body' tab, showing a JSON array of two book objects. The first book has an id of 1, title 'The Great Gatsby', and price 10.50. The second book has an id of 2, title 'Ulysses', and price 25.

KEY	VALUE
username	testuser
password	1234

```
[{"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.

The screenshot shows a Postman interface for a POST request to `http://localhost:3000/books`. The 'Body' tab is selected, and the data type is 'x-www-form-urlencoded'. The request body contains two form fields: 'username' with value 'testuser' and 'password' with value '1234'. The response is displayed in the 'Body' tab, showing an HTML message: 'Incorrect username or password'.

KEY	VALUE
username	testuser
password	1234

```
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

Params

Authorization

Headers (2)

Body ●

Pre-request Script

Tests

● none

● form-data

● x-www-form-urlencoded

● raw

● binary

	KEY	VALUE
<input checked="" type="checkbox"/>	username	testuser
<input checked="" type="checkbox"/>	password	1234
	Key	Value

Body

Cookies

Headers (6)


Test Results

Pretty

Raw

Preview

HTML ▼



i 1

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiJlbnTEZTI1NTMsImV4cCI6MTU1MTCzODk1M30.QF03Skmw1T8Dc1ZfUFzIhz_HORB85ZpFiM8S6uyF5c

Incorrect password results in :

POST

http://localhost:3000/authenticate

Params

Authorization

Headers (2)

Body

Pre-request Script

Tests

none

form-data

x-www-form-urlencoded

raw

binary

KEY	VALUE
<input checked="" type="checkbox"/> username	testuser
<input checked="" type="checkbox"/> password	12345
Key	Value

Body

Cookies

Headers (6)

Test Results

Pretty

Raw

Preview

HTML

1

Incorrect username or password

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

POST http://localhost:3000/books

Params Authorization Headers (2) **Body** Pre-request Script Tests

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

KEY	VALUE
<input checked="" type="checkbox"/> username	testuser
<input checked="" type="checkbox"/> password	1234
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiE1NTE2NTI2OTg5MTU1MTcz...
Key	Value

Body Cookies Headers (6) Test Results

Pretty Raw Preview JSON

```

1  [
2    {
3      "id": 1,
4      "title": "The Great Gatsby",
5      "price": "10.50"
6    },
7    {
8      "id": 2,
9      "title": "Ulysses",
10     "price": "25"
11   }
12 ]

```

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:3000/books
{ id: 1, title: 'The Great Gatsby', price: '10.50' },
{ id: 2, title: 'Ulysses', price: '25' } ]

```

Incorrect keys returns “Authentication Failed”