19th November, 2020

**Nikhil Teja Dommeti**

# REST API - User login, logout, access resources

REDFIN APPLICATION SECURITY CODING CHALLENGE

# Table of contents

## Introduction

This document describes a REST API which supports user login, logout and accessing user details. It outlines the design of the API, and the security practices which are followed to build the REST API.

## Architecture

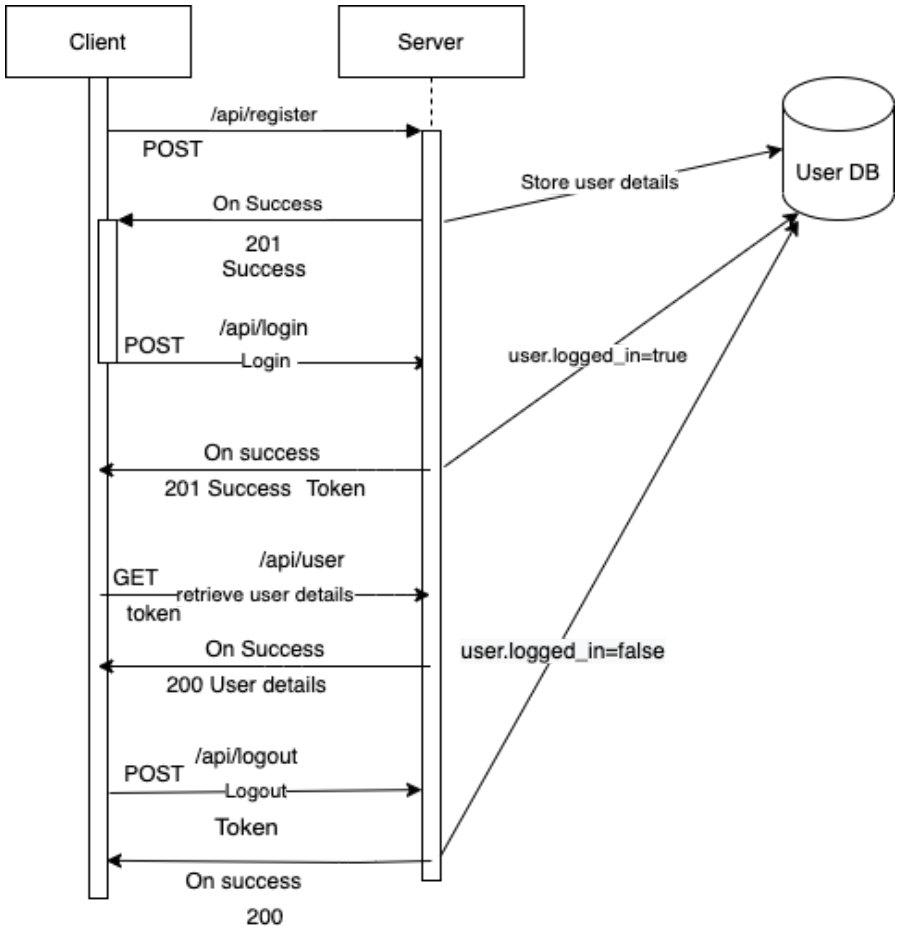


FIGURE (1): ARCHITECTURE OF THE RESTFUL API

Figure (1) describes the flow of the API and shows the client server interactions. It also shows how the endpoints can be utilized for accessing the resources.

# Secure Implementation Notes

REST API authentication and session management features are implemented in Flask without the use of standard libraries like **HTTPBasicAuth**(). Instead, implementation utilizes authorization fields from the request package to check if the username and password fields are valid. Also, JWT Tokens are created and assigned on successful login which can be utilized for accessing user details.

## 1. Secure Password Creation / Storage Strategies

OWASP Application Security Verification Standard (4.0), and NIST 800-63 Guidelines are utilized to come up with the following security policies for creating and storing passwords.

- **Password Creation Policies**
  - The implementation permits passwords of length **>= 8** characters and passwords of length **=< 64** characters as suggested in NIST 800-63 Guidelines.
  - NIST and OWASP recommend minimizing password complexity, like necessary inclusion of uppercase letters. Hence, such complexity requirements are not mandated for password creation.
  - This implementation rejects the creation of commonly used passwords by checking the user entered passwords against top 10,000 commonly used passwords, as recommended by OWASP app sec verification standard.

- **Password Storage Policy**

  The API implementation uses standard password hashing algorithm **bcrypt** with a work factor of **13**, a work factor of 13 contains 2^13(8192) iterations . The **salt** size of 128-bit is utilized for storing the hashes of the passwords. Work factor is set to **13** after considering NIST and OWASP recommendations.

- **Logging Failed Login attempts**

  Log entry is created for every failed login attempt that can be utilized for identifying brute force and credential stuffing attacks.

- **Generic Error Messages**

  Generic error messages are implemented in the application to prevent account enumeration attacks.

## 2. Token based session management

Token-based session management technique is used to manage sessions in the REST API. Token-based session management recommendations from OWASP Application Security Verification Standard are taken into consideration.

- **JSON web token (JWT)**

  JWT contains expiration time and digital signature is chosen as session management token, as it contains all the security requirements which are suggested in OWASP Application Security Verification Standard for token-based session management. **'HS512'** is utilized to sign the token

- **'JWT token verification'**

  Token verification is performed to mitigate the attack vector where someone would tamper with a token and make use of a different

digital signature (probably a less secure version), a whitelisting approach is used to verify the value of the algorithm field in the token received to the one used when signing the token.

- **Invalidating the token**

  The token is invalidated if the token expires or if the user utilizes logout endpoint.

## Recommendations/Changes

- The implementation utilizes HTTP basic authentication, which sends credentials over plaintext. Ideally, it should be always performed over TLS 1.3.
- SQLAlchemy is used for the implementation of the SQL queries in the API. Library documentation suggests the use of SQL injection prevention techniques such as encoding of special characters (like semicolons), parameterized queries, etc. However, further analysis of the library might be required.
- Additional analysis is needed to detect any security misconfigurations such as misconfigured HTTP headers, unsecure default configurations.

## Additional notes

Verified libraries like hashlib, pyJWT and cryptographically secure pseudo-random number generators like secrets() are utilized for key generation, hashing, and digital signature schemes.

- **Current users in the user database** :
  1. username : tpratchett, password: Thats Sir Terry to you!
  2. username : kvothe, password: 3##Heel7sa*9-zRwT

# API Documentation

This section describes how the API can be used for various use cases.

## Use case: creating a new user

Example

Endpoint: ‘/api/register’ , Method: POST

Body:


```
{
        "username" : "testuser" ,
        "password": "helloworld",
        "confirm_password": "helloworld" ,
        "full_name" : "Test Name",
        "search_engine_name" : "Test Search Engine"
}
```


Response: 201 SUCCESS


```
{
    "full_name": "Test Name",
    "search_engine_name": "Test Search Engine",
    "user_id": "testuser"
}
```

## Use case: User Login

Example

Endpoint: ‘/api/login’ , Method: POST

Basic Auth Headers:

"username" : "testuser" ,

"password": "helloworld",

Response: 200 SUCCESS

{"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MSwiZXhwIjoxNjA1NTkzNjkyfQ.rFhD0f6y8aCpIoTWp1vQHSNmhvrCpM-PZKmQ7gqAJ7s" }

## Use case: Getting user details

Example

Endpoint: ‘/api/user , Method: GET

Header:

"access-token" :

"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MSwiZXhwIjoxNjA1NTkzNjkyfQ.rFhD0f6y8aCpIoTWp1vQHSNmhvrCpM-PZKmQ7gqAJ7s"

Response: 200 SUCCESS

{

"full_name": "Test Name",

"search_engine_name": "Test Search Engine",

"user_id": "testuser"

}

## Use case: user logout

Example

Endpoint: ‘/api/logout’ , Method: GET

Header:

"access-token" :
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MSwiZXhwIjoxNjA1NTkzNjkyfQ.rFhD0f6y8aCpIoTWp1vQHSNmhvrCpM-PZKmQ7gqAJ7s"

Response: 200 SUCCESS


```
{
    "Message": "Logout Successful"
}
```

## Screenshots of API working in POSTMAN:

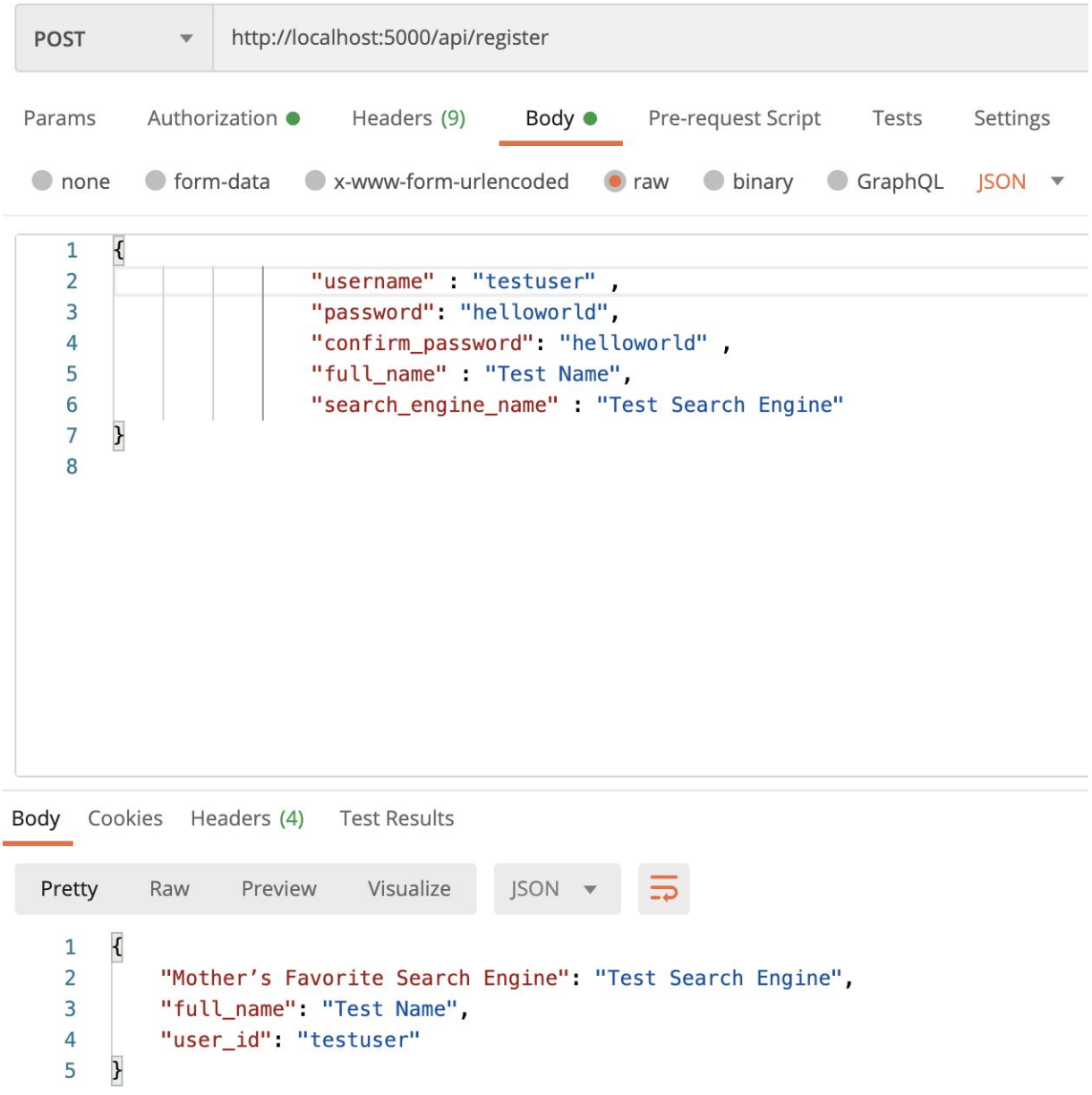


FIGURE (2): User registration in Postman

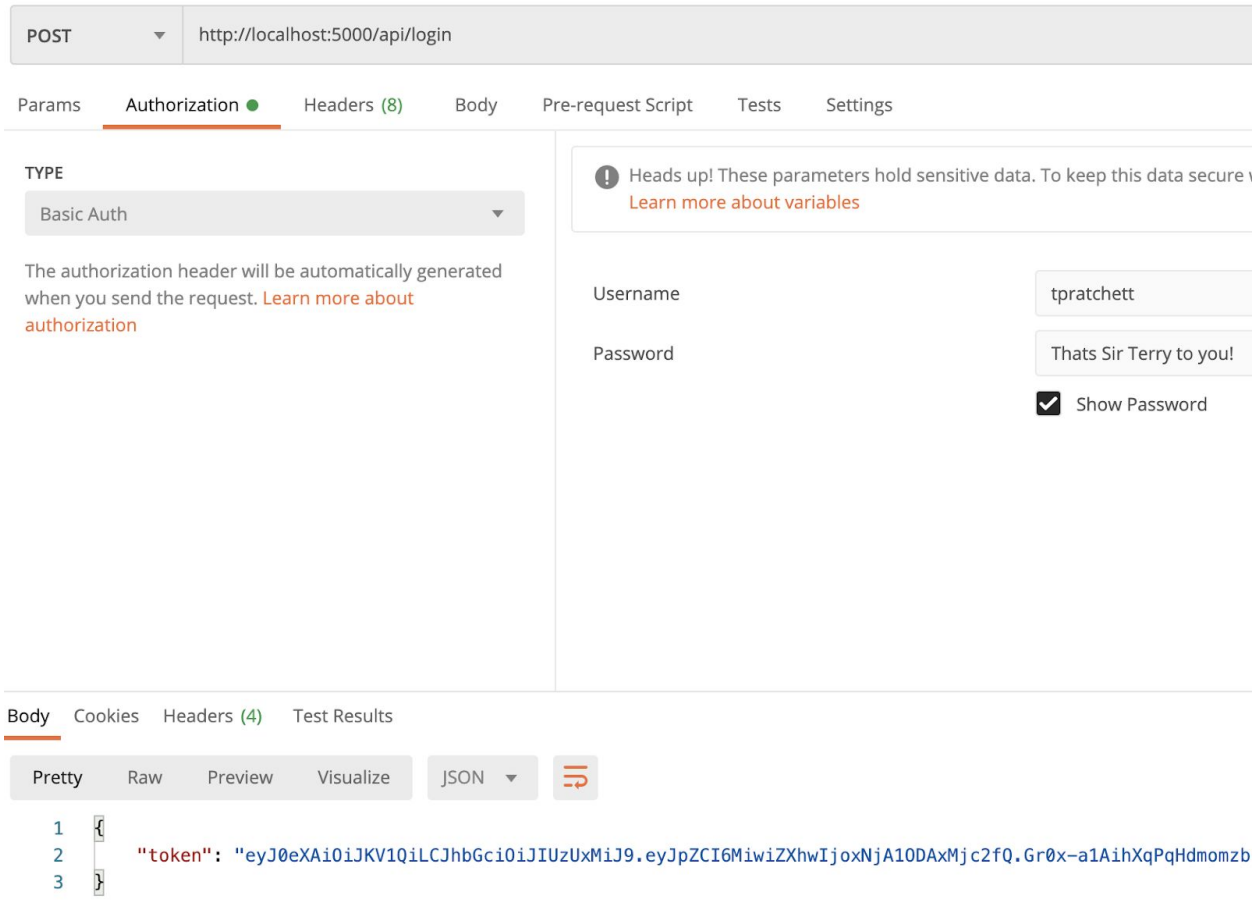


FIGURE (3): User login using POSTMAN

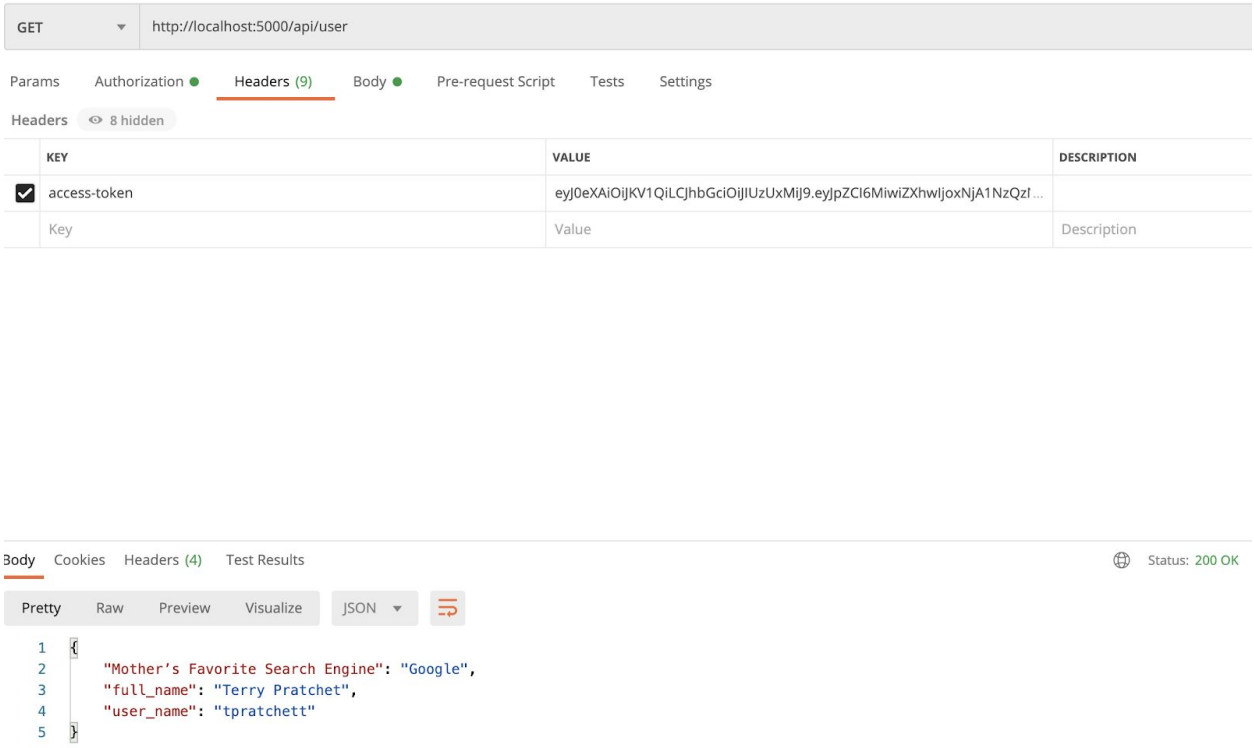


FIGURE (4): retrieving user using Postman

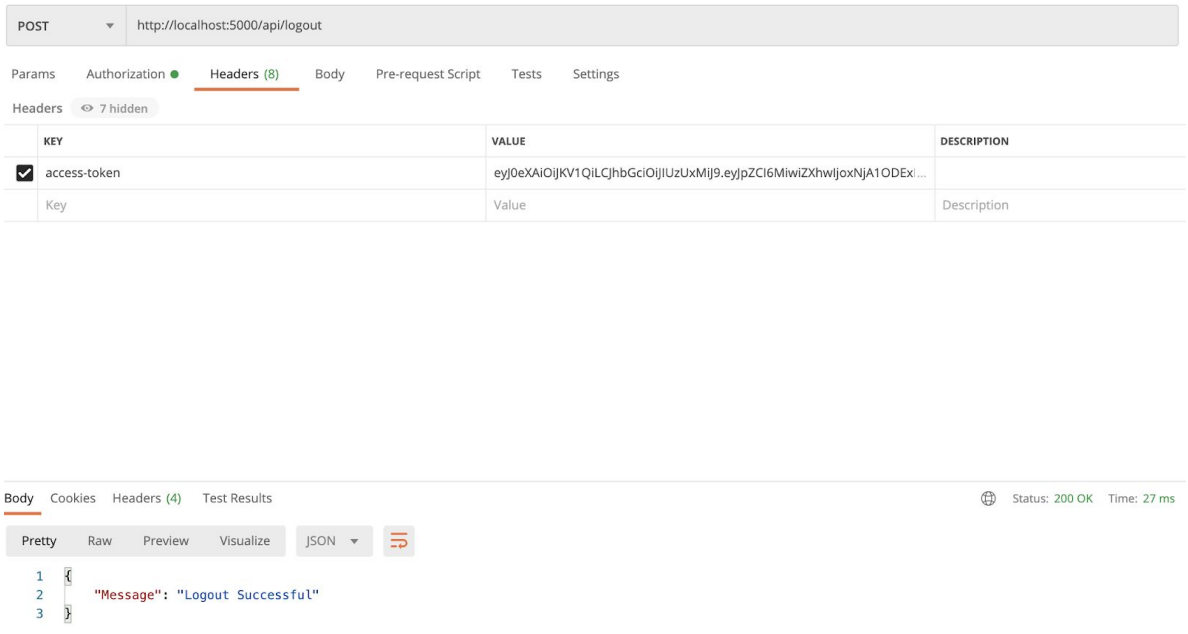


FIGURE (5): Logging out using Postman

## References

[1] Flask Documentation
(https://flask.palletsprojects.com/en/1.1.x/)
[2] SQLAlchemy
(https://www.sqlalchemy.org/)
[3] OWASP Secure Software Development
(https://wiki.owasp.org/index.php/OWASP_Secure_Software_Development_Lifecycle_Projec)
[4] NIST Special Publications
(https://pages.nist.gov/800-63-3/sp800-63b.html)
[5] Building RESTFul API using Flask
(https://blog.miguelgrinberg.com/post/restful-authentication-with-flask)