Security features

Role-Based Authorization:
The service uses role-based authorization to separate access control among different user roles (admin and user).

Secure Communication:
Use SSL/TLS encryption, all communications between clients and the server are encrypted.

```
app.run(ssl_context='adhoc')
```

Audit and Logging:
Used logging library to log all significant events like login attempts, registration, password changes, and unauthorized access attempts. The log is saved in the app.log file.

Secure Session Management:
By using configurations such as SESSION_COOKIE_SECURE, SESSION_COOKIE_HTTPONLY, and SESSION_COOKIE_SAMESITE, ensuring data is secured. Also sessions will expire after 10 minutes.

Password Management:
Using hashing algorithms provided by the Werkzeug library. This ensures passwords are stored securely in the database.

Input Validation and Sanitization:
Input validation is enforced through WTForms. Usernames can only contain letters, numbers, and underscores. This helps to prevent XSS attacks. Passwords are at least 8 characters long, contain at least one letter, one number, and may contain special characters. This password

policy helps to ensure that users choose strong passwords.

```python
class RegistrationForm(Form):
    username = StringField('Username', [validators.Length(min=4, max=25), validators.DataRequired()])
    password = PasswordField('Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords must match'),
        validators.Length(min=8)
    ])
    confirm = PasswordField('Repeat Password')

class LoginForm(Form):
    username = StringField('Username', [validators.Length(min=4, max=25), validators.DataRequired()])
    password = PasswordField('Password', [validators.DataRequired()])

class ChangePasswordForm(Form):
    old_password = PasswordField('Old Password', [validators.DataRequired()])
    new_password = PasswordField('New Password', [
        validators.DataRequired(),
        validators.EqualTo('confirm', message='Passwords must match'),
        validators.Length(min=8)
    ])
    confirm = PasswordField('Repeat Password')
```

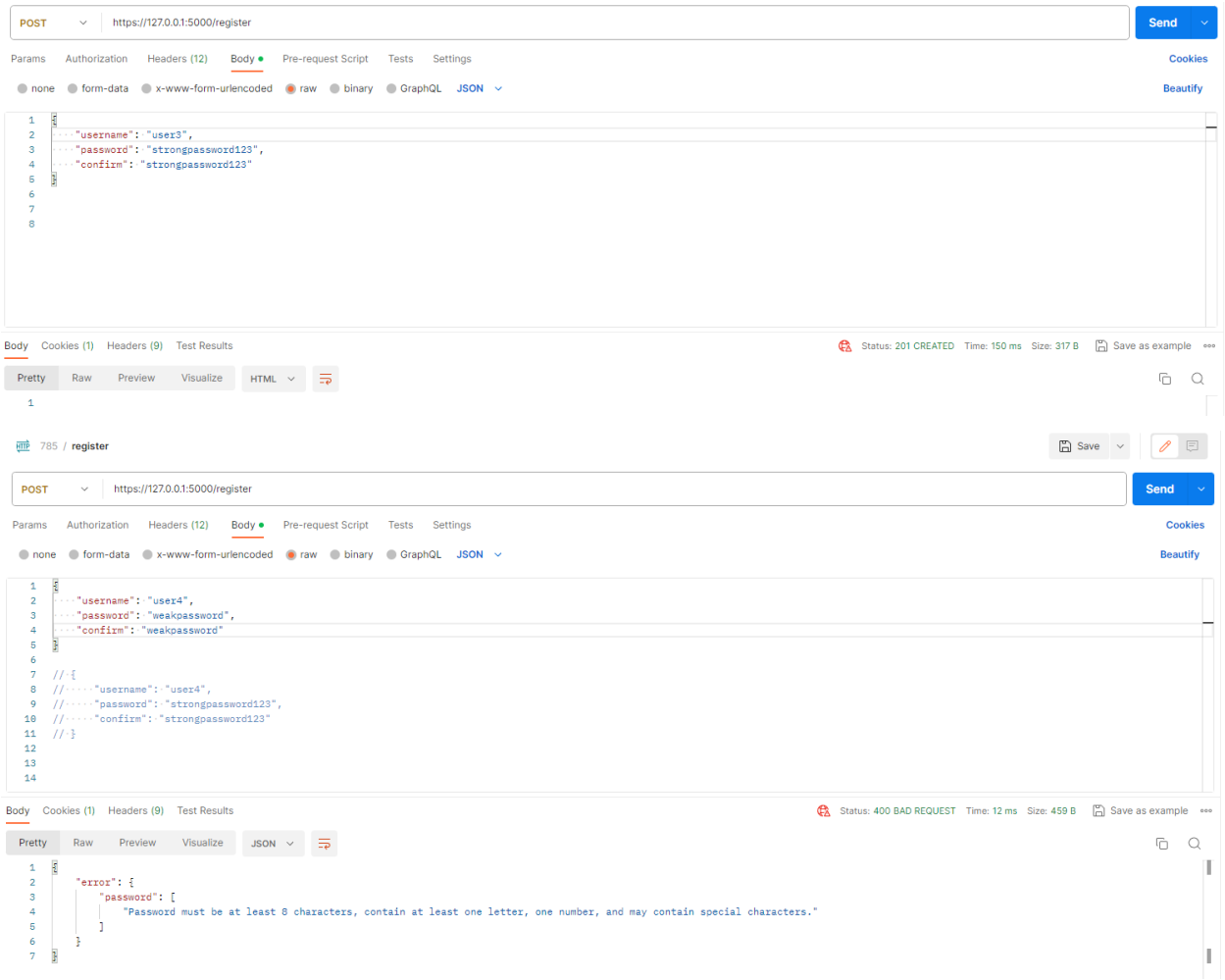
Security Headers:
Various HTTP security headers such as X-Content-Type-Options, X-Frame-Options, X-XSS-Protection, and Content-Security-Policy are set to mitigate risks.

```python
# Security headers
@app.after_request
def apply_security_headers(response):
    response.headers["X-Content-Type-Options"] = "nosniff"  # prevents MIME type sniffing
    response.headers["X-Frame-Options"] = "SAMEORIGIN"  # prevents clickjacking
    response.headers["X-XSS-Protection"] = "1; mode=block"  # protect against XSS attacks
    response.headers["Content-Security-Policy"] = "default-src 'self'"  # restricts the sources from which content can be loaded
    return response
```

Database Security:
Using SQLAlchemy for database interactions, the service benefits from its defense against SQL injection

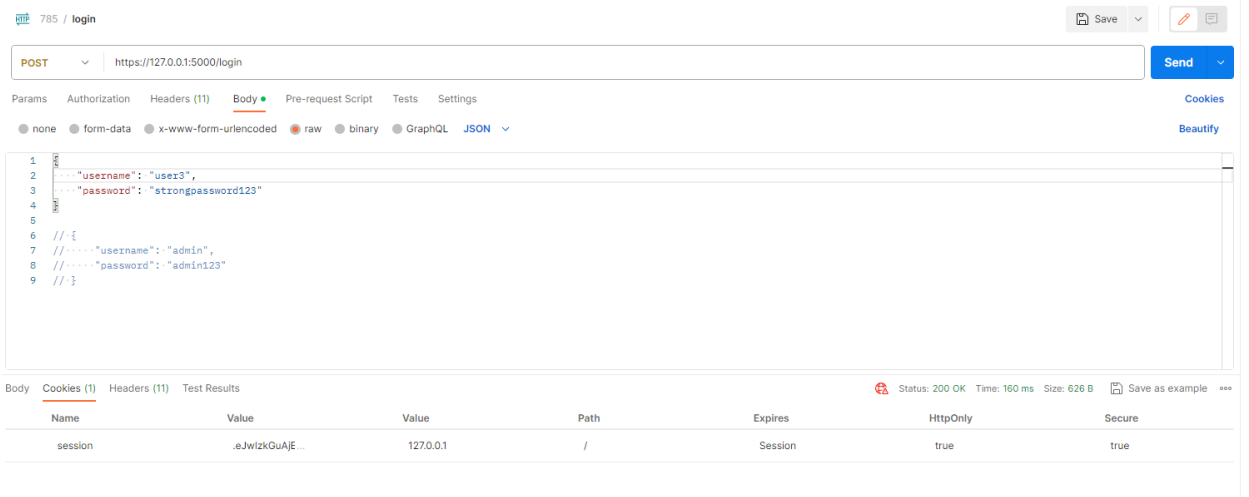
We use postman to test our APIs, here are the tests we did to check functionality:

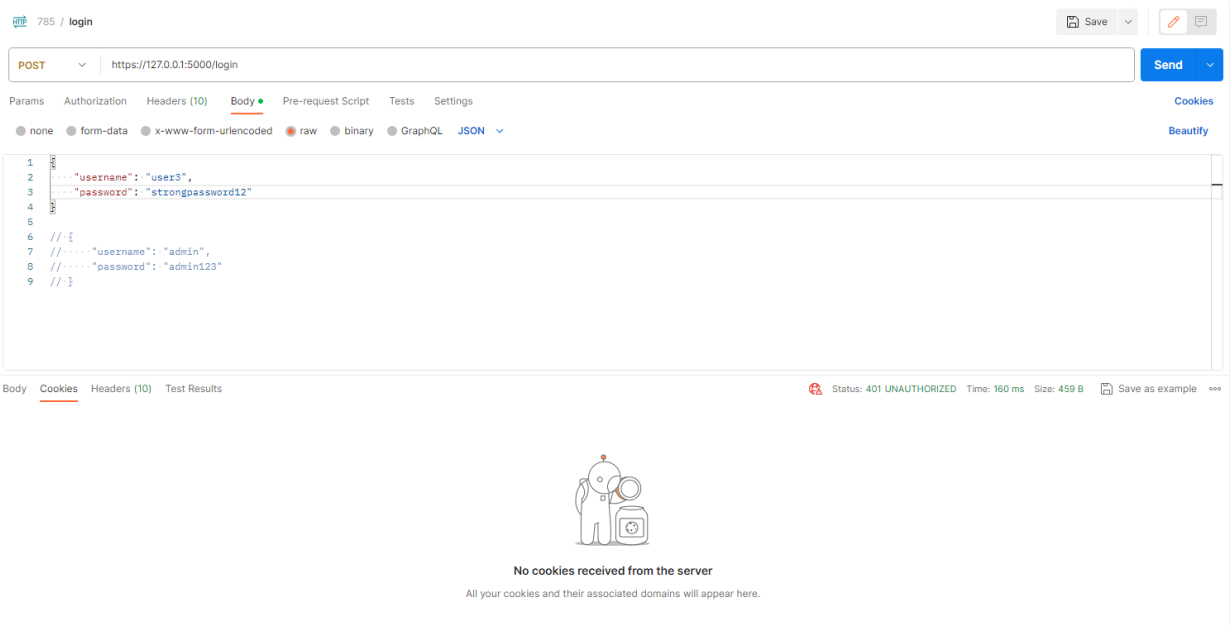
POST /register:


POST
https://127.0.0.1:5000/register
Send
Params Authorization Headers (12) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"username": "user3",
"password": "strongpassword123",
"confirm": "strongpassword123"
}
Body Cookies (1) Headers (9) Test Results
Status: 201 CREATED Time: 150 ms Size: 317 B Save as example
Pretty Raw Preview Visualize HTML

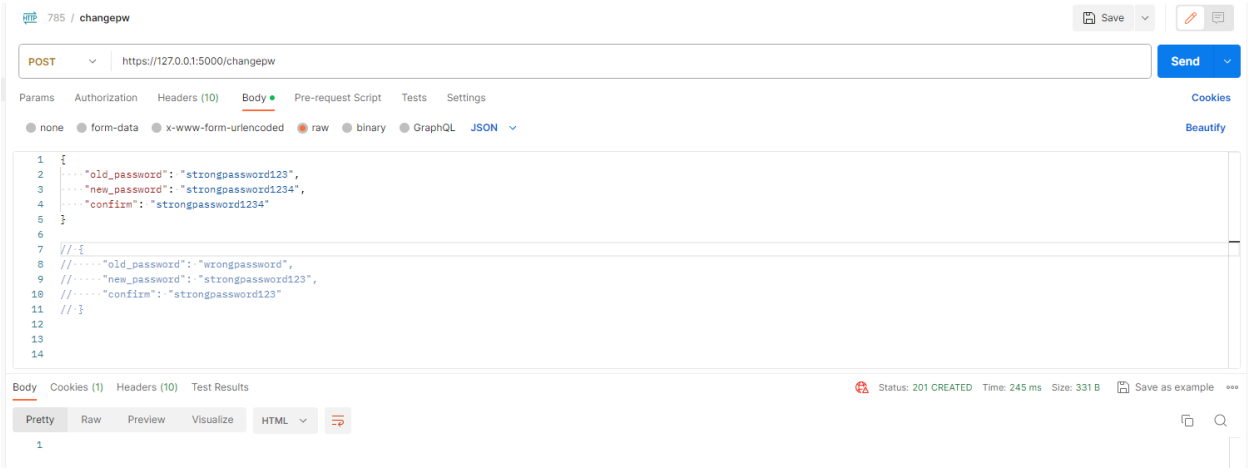
785 / register
Save
POST
https://127.0.0.1:5000/register
Send
Params Authorization Headers (12) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"username": "user4",
"password": "weakpassword",
"confirm": "weakpassword"
}
// {
// "username": "user4",
// "password": "strongpassword123",
// "confirm": "strongpassword123"
// }
Body Cookies (1) Headers (9) Test Results
Status: 400 BAD REQUEST Time: 12 ms Size: 459 B Save as example
Pretty Raw Preview Visualize JSON
{
"error": {
"password": [
"Password must be at least 8 characters, contain at least one letter, one number, and may contain special characters."
]
}

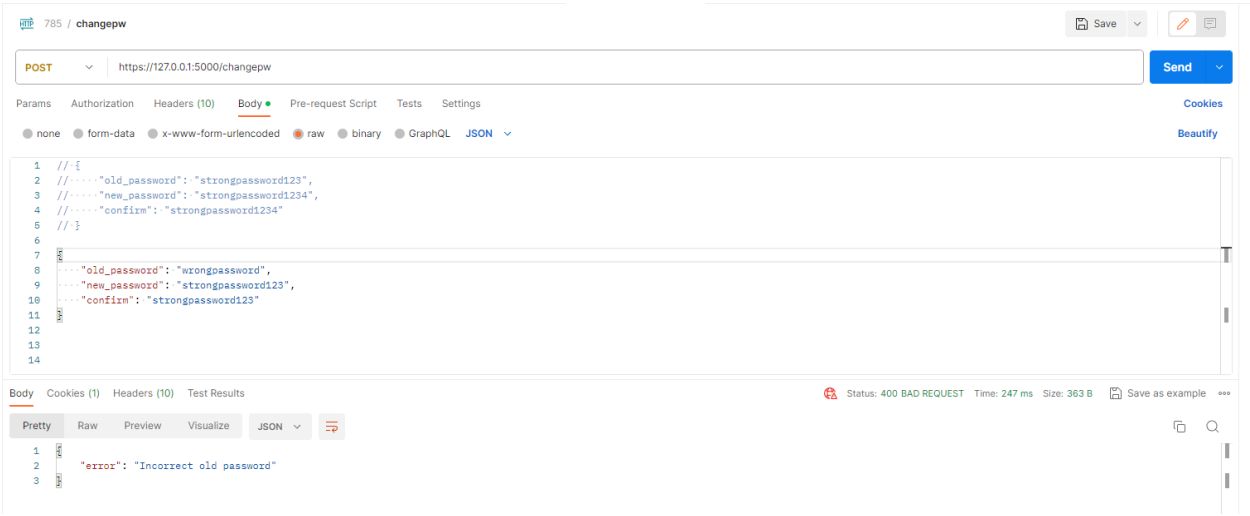
POST /login


785 / login
Save
POST
https://127.0.0.1:5000/login
Send
Params Authorization Headers (11) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"username": "user3",
"password": "strongpassword123"
}
// {
// "username": "admin",
// "password": "admin123"
// }
Body Cookies (1) Headers (11) Test Results
Status: 200 OK Time: 160 ms Size: 626 B Save as example
Name Value Value Path Expires HttpOnly Secure
session .eJwlzkGuAjE... 127.0.0.1 / Session true true



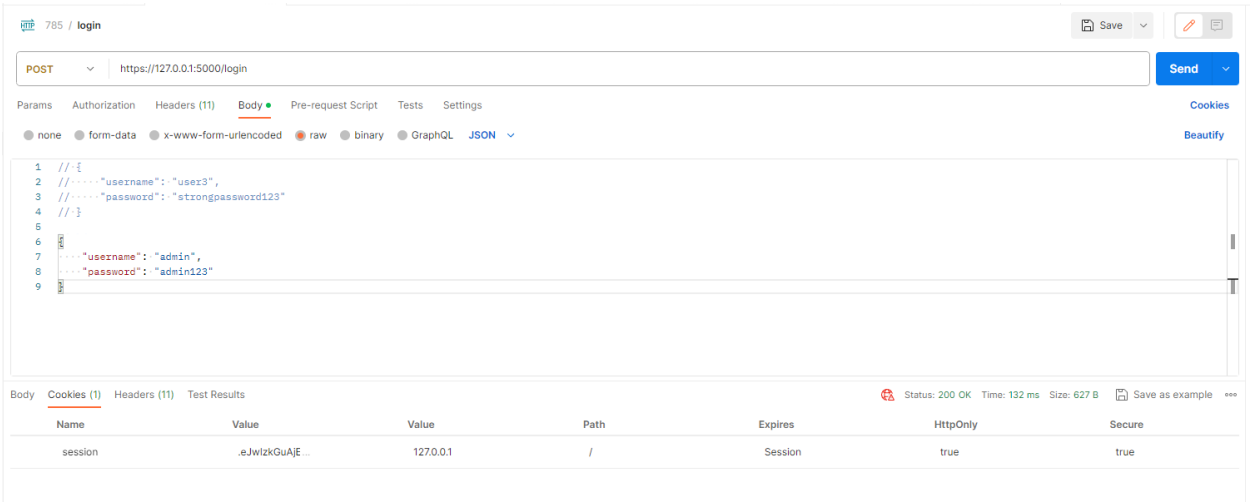
POST /changepw



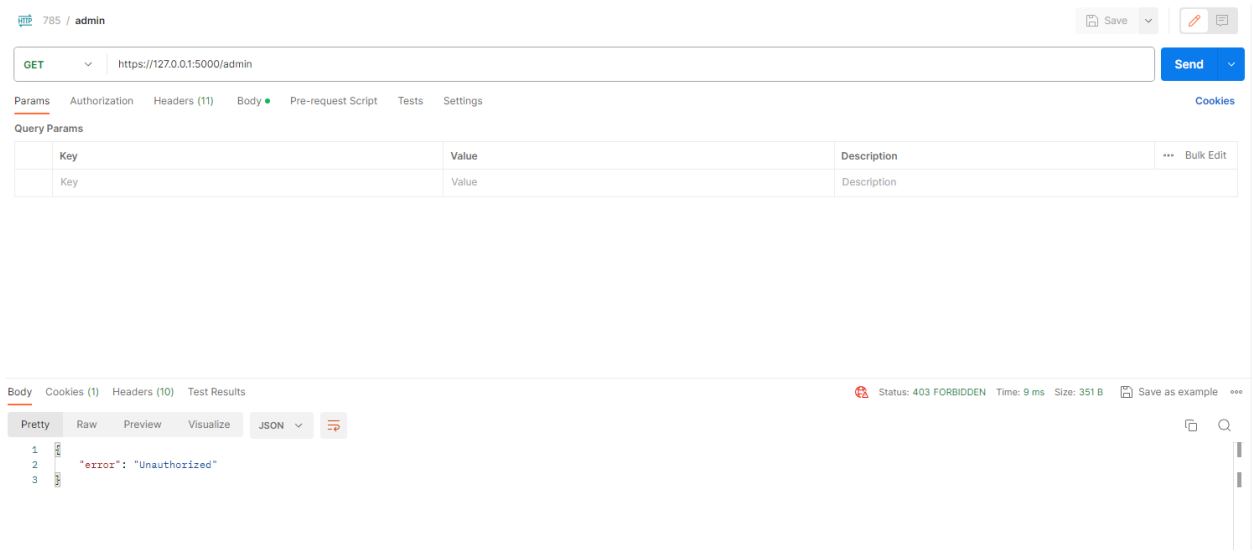


GET /admin

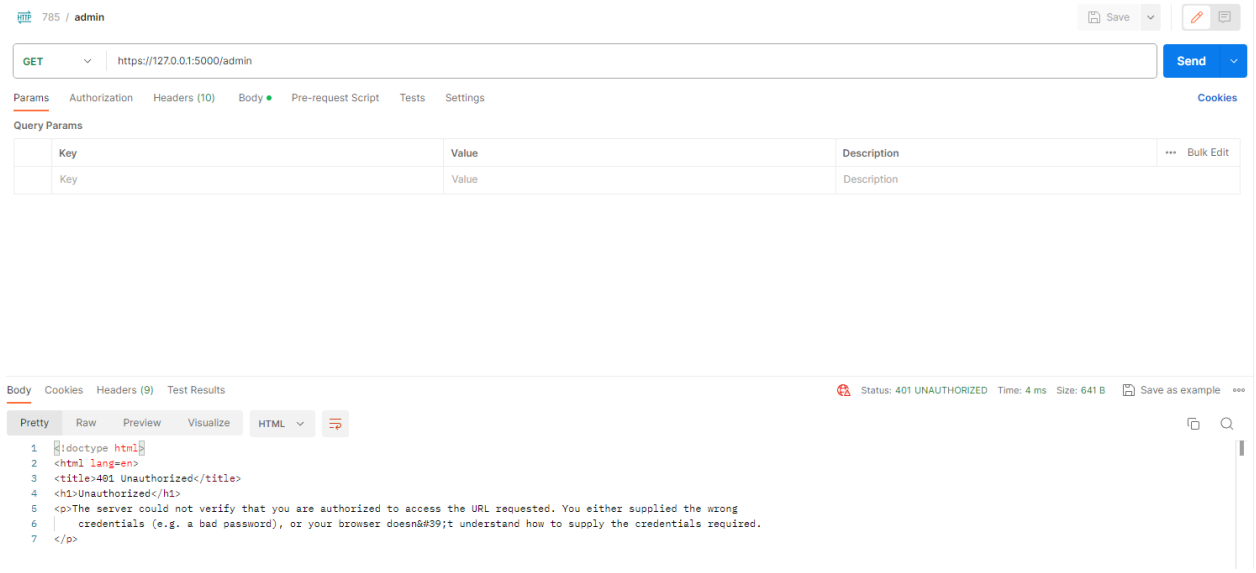
1. Success login



2. Logged in as user

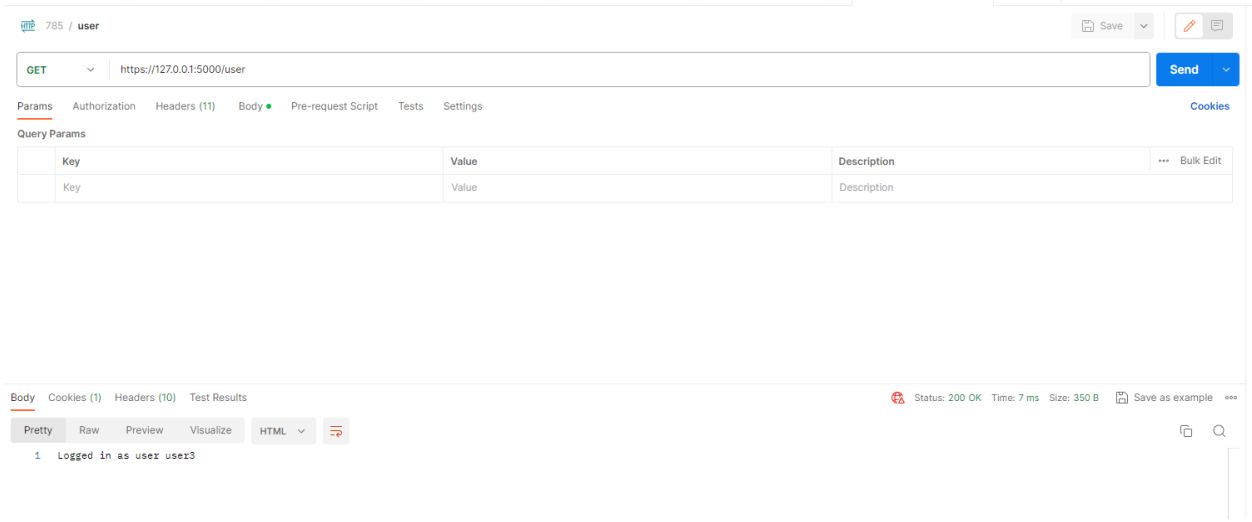


3. Failed login

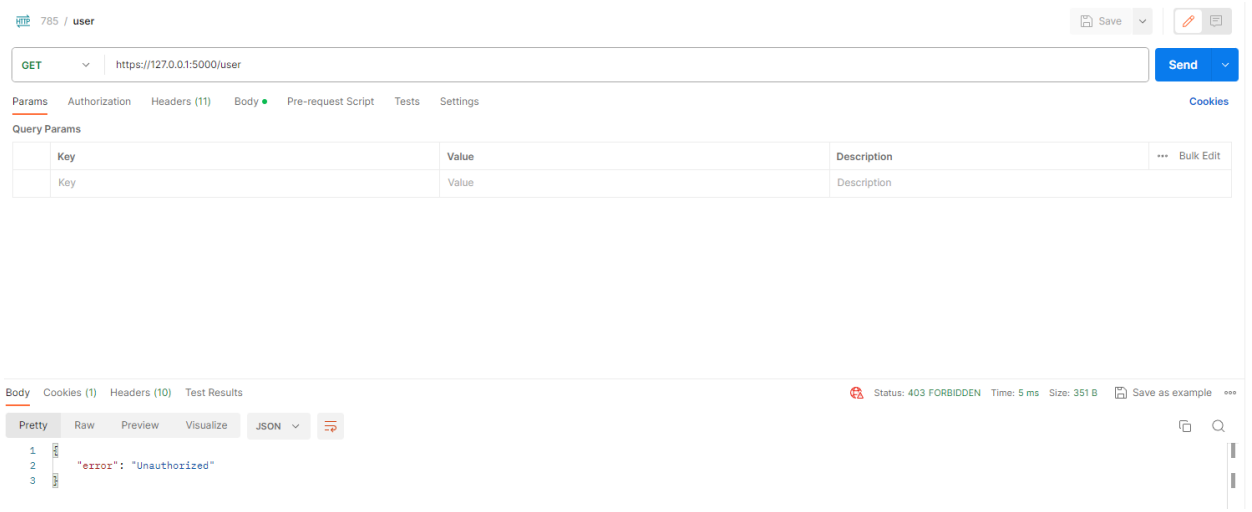


GET /user

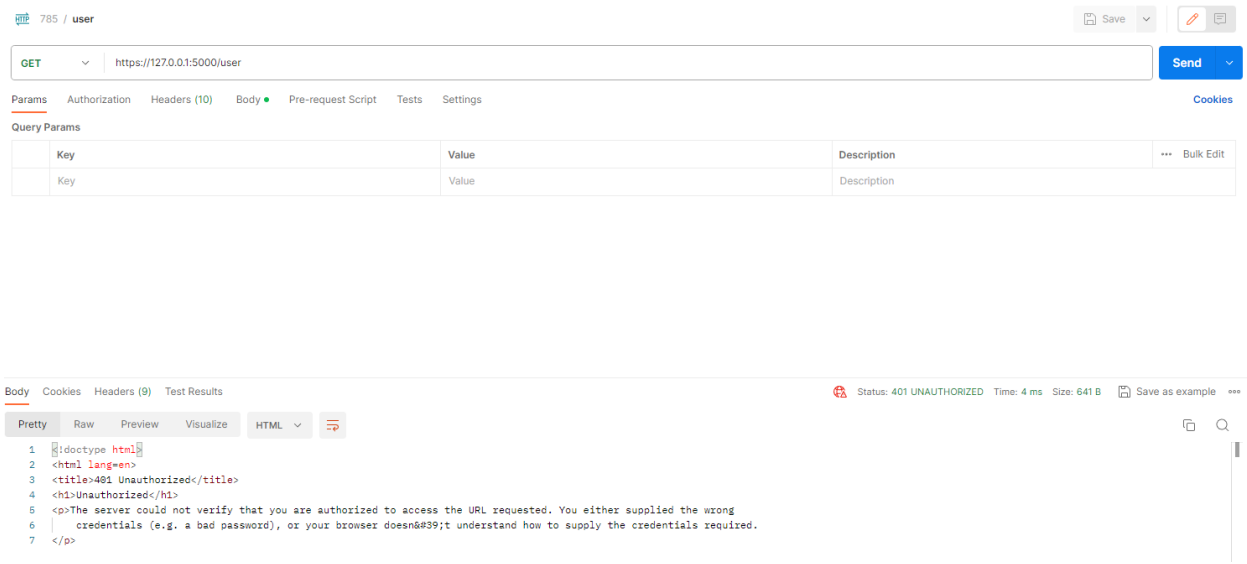
1. Success login



2. Logged in as admin



3. Login failure


785 / user
Save
GET
https://127.0.0.1:5000/user
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description

Body
Cookies
Headers (9)
Test Results
Status: 401 UNAUTHORIZED
Time: 4 ms
Size: 641 B
Save as example
Pretty
Raw
Preview
Visualize
HTML
<!doctype html>
<html lang=en>
<title>401 Unauthorized</title>
<h1>Unauthorized</h1>
The server could not verify that you are authorized to access the URL requested. You either supplied the wrong
credentials (e.g. a bad password), or your browser doesn't understand how to supply the credentials required.