# 🚀 Flask JWT Authentication Using Cookies (Complete Guide)

## 1️⃣ Introduction

TechSolutions Inc. wanted to secure their Flask-based REST API using OAuth authentication with JWT tokens stored in cookies. Users log in via a Jinja2-powered web console, and upon successful authentication, they receive a JWT stored as an HTTP-only cookie. This ensures secure authentication without exposing tokens to JavaScript.

## 2️⃣ Problem Statement

The API should provide:

* ✔️ Secure authentication using \*\*JWT tokens\*\*.
* ✔️ Store JWT securely using \*\*HTTP-only cookies\*\*.
* ✔️ Protect API endpoints using \*\*JWT-based authentication\*\*.
* ✔️ Provide a \*\*login, dashboard, and logout flow\*\*.

## 3️⃣ Solution Overview

To implement this, the following technologies are used:

* ✔️ \*\*Flask\*\* - Backend framework.
* ✔️ \*\*Flask-JWT-Extended\*\* - For JWT authentication.
* ✔️ \*\*Flask-Login\*\* - For user session management.
* ✔️ \*\*Bcrypt\*\* - For password hashing.
* ✔️ \*\*Jinja2\*\* - Templating engine for the web console.

## 4️⃣ Installation Steps

* Run the following command to install dependencies:

pip install flask flask-jwt-extended flask-bcrypt flask-login

## 5️⃣ Code Implementation

### 📜 Backend Code (`app.py`)

from flask import Flask, render\_template, request, redirect, url\_for, make\_response  
from flask\_jwt\_extended import JWTManager, create\_access\_token, jwt\_required, get\_jwt\_identity  
from flask\_bcrypt import Bcrypt  
from flask\_login import LoginManager, UserMixin, login\_user, logout\_user  
import datetime  
  
app = Flask(\_\_name\_\_)  
  
# Secure Key Configurations  
app.config["SECRET\_KEY"] = "supersecurekey123"  
app.config["JWT\_SECRET\_KEY"] = "supersecretkey"  
app.config["JWT\_TOKEN\_LOCATION"] = ["cookies"]  
app.config["JWT\_COOKIE\_SECURE"] = False # Set True in production (HTTPS)  
  
jwt = JWTManager(app)  
bcrypt = Bcrypt(app)  
login\_manager = LoginManager(app)  
  
# Dummy User Database  
users = {"alice": bcrypt.generate\_password\_hash("password123").decode('utf-8')}  
  
# User Class  
class User(UserMixin):  
 def \_\_init\_\_(self, username):  
 self.id = username  
  
@login\_manager.user\_loader  
def load\_user(username):  
 return User(username) if username in users else None  
  
# Route: Show Login Page  
@app.route('/')  
def home():  
 return render\_template('login.html')  
  
# Route: Authenticate User & Set JWT Cookie  
@app.route('/login', methods=['POST'])  
def login():  
 username = request.form["username"]  
 password = request.form["password"]  
  
 if username in users and bcrypt.check\_password\_hash(users[username], password):  
 access\_token = create\_access\_token(identity=username, expires\_delta=datetime.timedelta(hours=1))  
  
 response = make\_response(redirect(url\_for('dashboard')))  
 response.set\_cookie("access\_token\_cookie", access\_token, httponly=True, samesite="Lax")  
 login\_user(User(username))  
  
 return response  
 else:  
 return "Invalid credentials", 401  
  
# Protected Route: Dashboard (Requires JWT)  
@app.route('/dashboard')  
@jwt\_required(locations=["cookies"])  
def dashboard():  
 current\_user = get\_jwt\_identity()  
 return render\_template('dashboard.html', username=current\_user)  
  
# Route: Logout (Clears JWT Cookie)  
@app.route('/logout')  
def logout():  
 response = make\_response(redirect(url\_for('home')))  
 response.set\_cookie("access\_token\_cookie", "", expires=0)  
 logout\_user()  
 return response  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

### 📜 Jinja2 Templates

#### 🔹 Login Page (`templates/login.html`)

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <title>Login</title>  
</head>  
<body>  
 <h2>Login</h2>  
 <form method="POST" action="/login">  
 <label>Username:</label>  
 <input type="text" name="username" required>  
 <br>  
 <label>Password:</label>  
 <input type="password" name="password" required>  
 <br>  
 <button type="submit">Login</button>  
 </form>  
</body>  
</html>

#### 🔹 Dashboard Page (`templates/dashboard.html`)

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <title>Dashboard</title>  
</head>  
<body>  
 <h2>Welcome, {{ username }}!</h2>  
 <p>You have successfully logged in.</p>  
 <a href="/logout">Logout</a>  
</body>  
</html>

## 6️⃣ Testing the Application

* Run the Flask server:
* python app.py
* Then, follow these steps:

1. Open `http://127.0.0.1:5000/` and log in using username `alice` and password `password123`.
2. Visit `http://127.0.0.1:5000/dashboard` to access the protected route.
3. Click `Logout` to clear the JWT and return to login.

## 7️⃣ Conclusion

By implementing this \*\*Flask JWT authentication system\*\*, TechSolutions Inc. successfully:

* ✔️ Secured authentication using \*\*JWT tokens in HTTP-only cookies\*\*.
* ✔️ Implemented \*\*role-based protected routes\*\*.
* ✔️ Enabled \*\*frontend compatibility (React, Android, etc.)\*\*.

This ensures a robust and scalable authentication system. 🚀