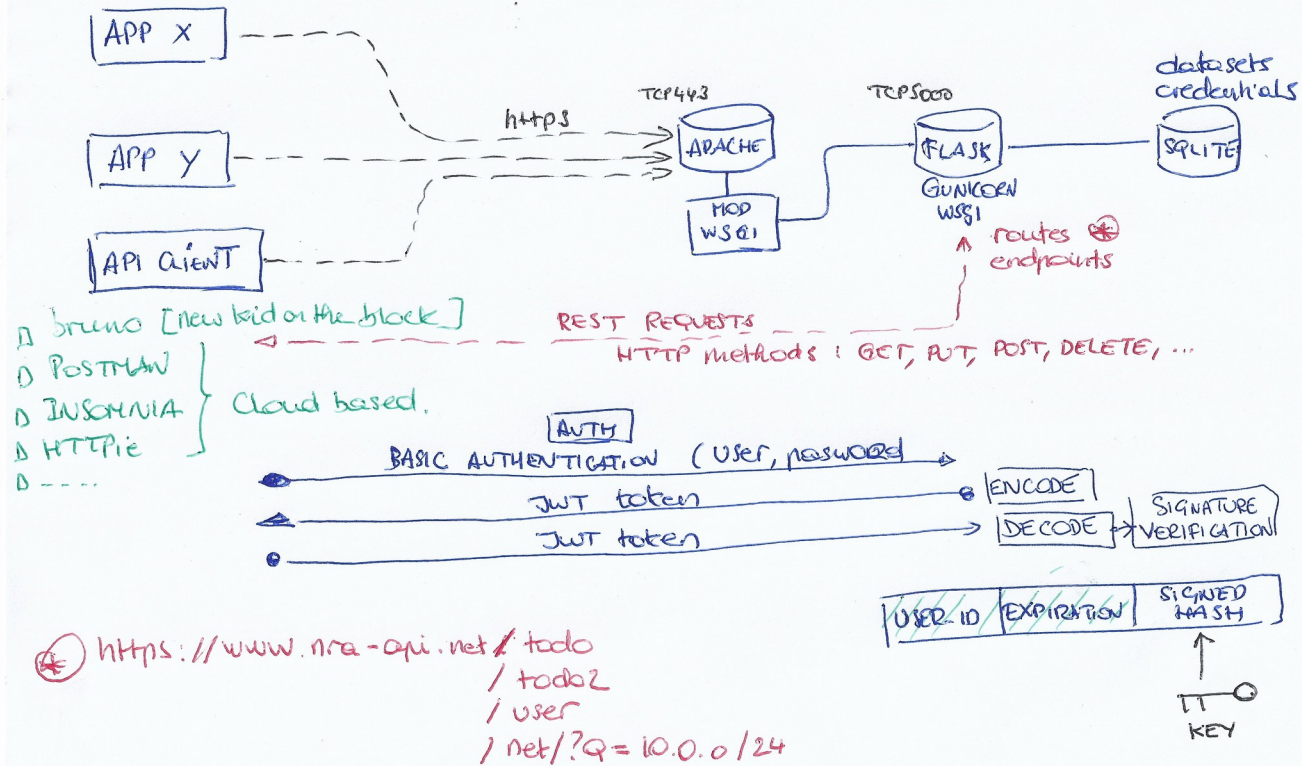


Flask / JWT REST API Environment





[Debugger](#)

[Libraries](#)

[Introduction](#)

[Ask](#)

Crafted by



Auth0
by Okta



JSON Web Tokens are an open, industry standard **RFC 7519** method for representing claims securely between two parties.

JWT.IO allows you to decode, verify and generate JWT.

[LEARN MORE ABOUT JWT](#)

[SEE JWT LIBRARIES](#)



Manifesto

Bru

Sponsors

Blog

About

Pricing

Support



21,502

Download

Re-Inventing the API Client

Bruno is a Fast and Git-Friendly Opensource API client, aimed at revolutionizing the status quo represented by Postman, Insomnia and similar tools out there.

Bruno stores your collections directly in a folder on your filesystem. We use a plain text markup language, Bru, to save information about API requests.

You can use git or any version control of your choice to collaborate over your API collections.

Bruno is offline-only. There are no plans to add cloud-sync to Bruno, ever. We value your data privacy and believe it should stay on your device. Read our long-term vision [here](#).

Basic Authentication

The screenshot shows a REST client interface with a dark theme. The top bar displays the method 'GET' and the URL 'http://localhost:5000/login'. Below this, a tabbed interface includes 'Params', 'Body', 'Headers', 'Auth' (which is selected and underlined), 'Vars', 'Script', and 'Assert'. Under the 'Auth' tab, there is a 'Basic Auth' dropdown menu. Below the dropdown, there are two input fields: 'Username' with the value 'Bruno' and 'Password' with the value 'geheim'. To the right of the 'Auth' tab, there are tabs for 'Response', 'Headers', 'Timeline', and 'Tests'. The 'Response' tab is selected and underlined. It shows a status of '200 OK' in green, with a response time of '525ms' and a size of '191B'. The response body is a JSON object with a single key 'token' containing a long alphanumeric string. The response is displayed in a syntax-highlighted format with line numbers 1, 2, and 3 on the left.

```
GET http://localhost:5000/login
```

Params Body Headers Auth Vars Script Assert

Tests Docs

Basic Auth ▾

Username

Bruno

Password

geheim

Response Headers Timeline Tests

200 OK 525ms 191B

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJw  
dWJsaWNfaWQiOiI0NGYwNjUxOS1jYmMzLTQzYmMtOTU2Ni01NGVky  
2FiYWJhNTgiLCJleHAiOiJlE3MTkzMjE0ODd9.Gp1_Fu9bNavNtcsCU  
TU1vruBfArr6yuGUfLxu_lTr2o"  
3 }
```

Request with valid token

GET http://localhost:5000/net/?specific_net=3.3.3.0/8

Params¹ Body Headers¹ Auth Vars Script Assert

Tests Docs

Name	Value	
x-access-token	JTU1vruBfArr6yuGUfLxu_ITr2o	<input checked="" type="checkbox"/>

+ Add Header

Response Headers Timeline Tests

200 OK 7ms 25B

```
1 {  
2   "message": "TOKYO"  
3 }
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkdWJsawNfawQioiI0NGYWNjUxOS1jYmZLTQzYmMtOTU2Ni01NGVkyY2FiYWJhNTgiLCJleHAiOiJlE3MTkzMjE0ODd9.Gp1_Fu9bNavNtcsCUTU1vruBfArr6yuGUfLxu_ITr2o  
{'public_id': '44f06519-cbc3-43bc-9566-54edcababa58', 'exp': 1719321487}  
127.0.0.1 - - [25/Jun/2024 14:51:06] "GET /net/?specific_net=3.3.3.0/8 HTTP/1.1" 200 -
```

Request with tampered / expired token

The screenshot displays a REST client interface with a GET request to `http://localhost:5000/net/?specific_net=3.3.3.0/8`. The 'Headers' tab is active, showing a single header: `x-access-token` with the value `.rr6yuGUf_TAMPERED_HERE__`. This header entry is highlighted with an orange rectangle. The 'Response' tab shows a 401 Unauthorized status with a response body of `{ "message": "Token is invalid!" }`. The status bar at the top right indicates '401 Unauthorized', '3ms', and '37B'.

GET `http://localhost:5000/net/?specific_net=3.3.3.0/8`

Params¹ Body Headers¹ Auth Vars Script Assert

Tests Docs

Name	Value	
x-access-token	.rr6yuGUf_TAMPERED_HERE__	<input checked="" type="checkbox"/>

+ Add Header

Response Headers Timeline Tests 401 Unauthorized 3ms 37B

```
1 {  
2   "message": "Token is invalid!"  
3 }
```


Token Encoding

```
app.config['SECRET_KEY'] = 'thisissecret'
```

```
@app.route('/login')
def login():
    auth = request.authorization

    if not auth or not auth.username or not auth.password:
        return make_response('Could not verify', 401, {'WWW-Authenticate': 'Basic realm="Login required!"'})

    user = User.query.filter_by(name=auth.username).first()

    if not user:
        return make_response('Could not verify', 401, {'WWW-Authenticate': 'Basic realm="Login required!"'})

    if check_password_hash(user.password, auth.password):
        token = jwt.encode({'public_id': user.public_id,
                           'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=30)},
                           app.config['SECRET_KEY'],
                           algorithm="HS256")
        return jsonify({'token': token})

    return make_response('Could not verify', 401, {'WWW-Authenticate': 'Basic realm="Login required!"'})
```

Token Decoding / Verification

```
app.config['SECRET_KEY'] = 'thisissecret'
```

```
def token_required(f):  
    @wraps(f)  
    def decorated(*args, **kwargs):  
        token = None  
  
        if 'x-access-token' in request.headers:  
            token = request.headers['x-access-token']  
            print(token)  
        if not token:  
            return jsonify({'message': 'Token is missing!'}), 401  
  
        try:  
            data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=["HS256"])  
            current_user = User.query.filter_by(public_id=data['public_id']).first()  
        except Exception as error:  
            return jsonify({'message': 'Token is invalid!'}), 401  
  
        return f(current_user, *args, **kwargs)  
  
    return decorated
```