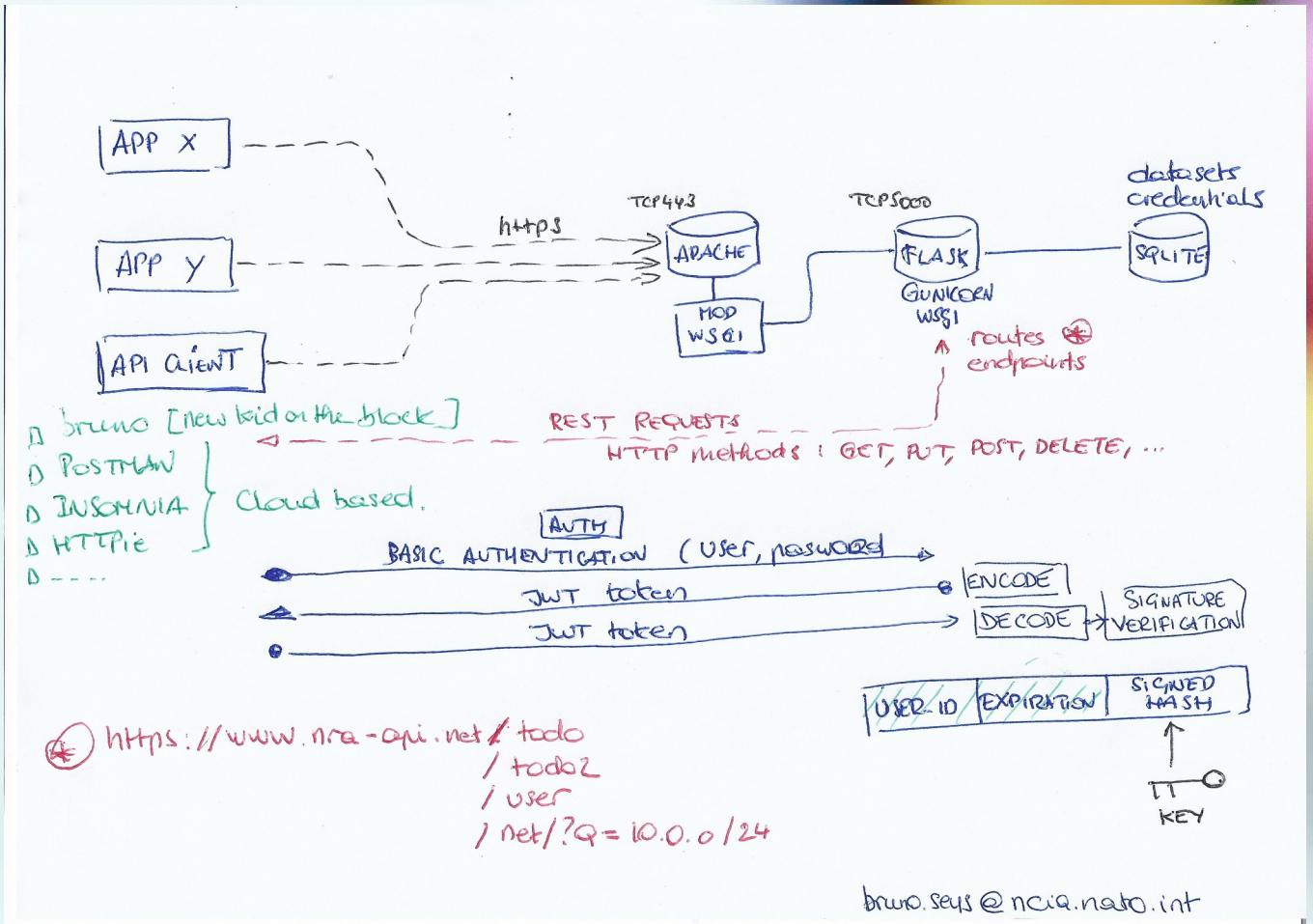
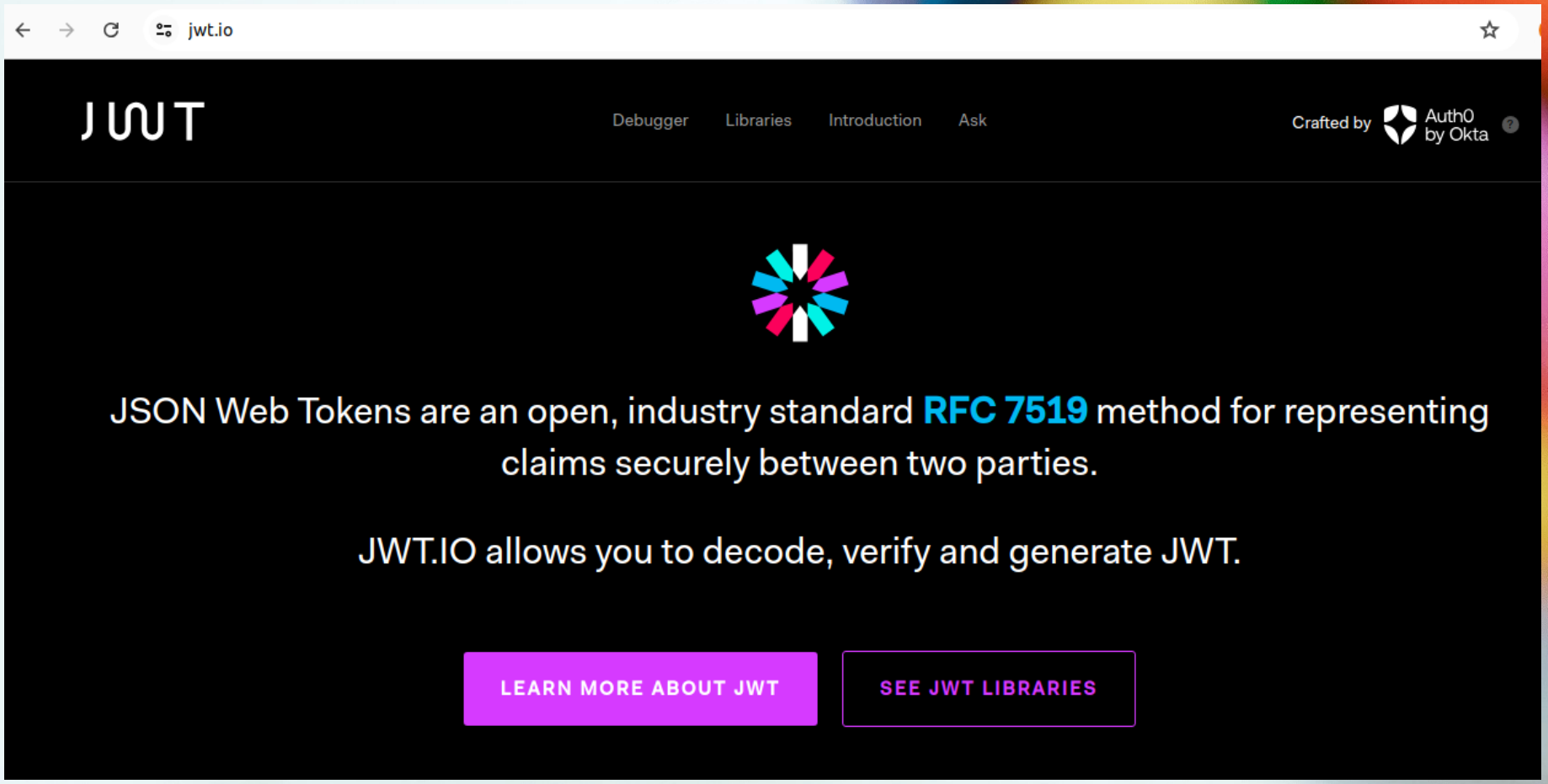
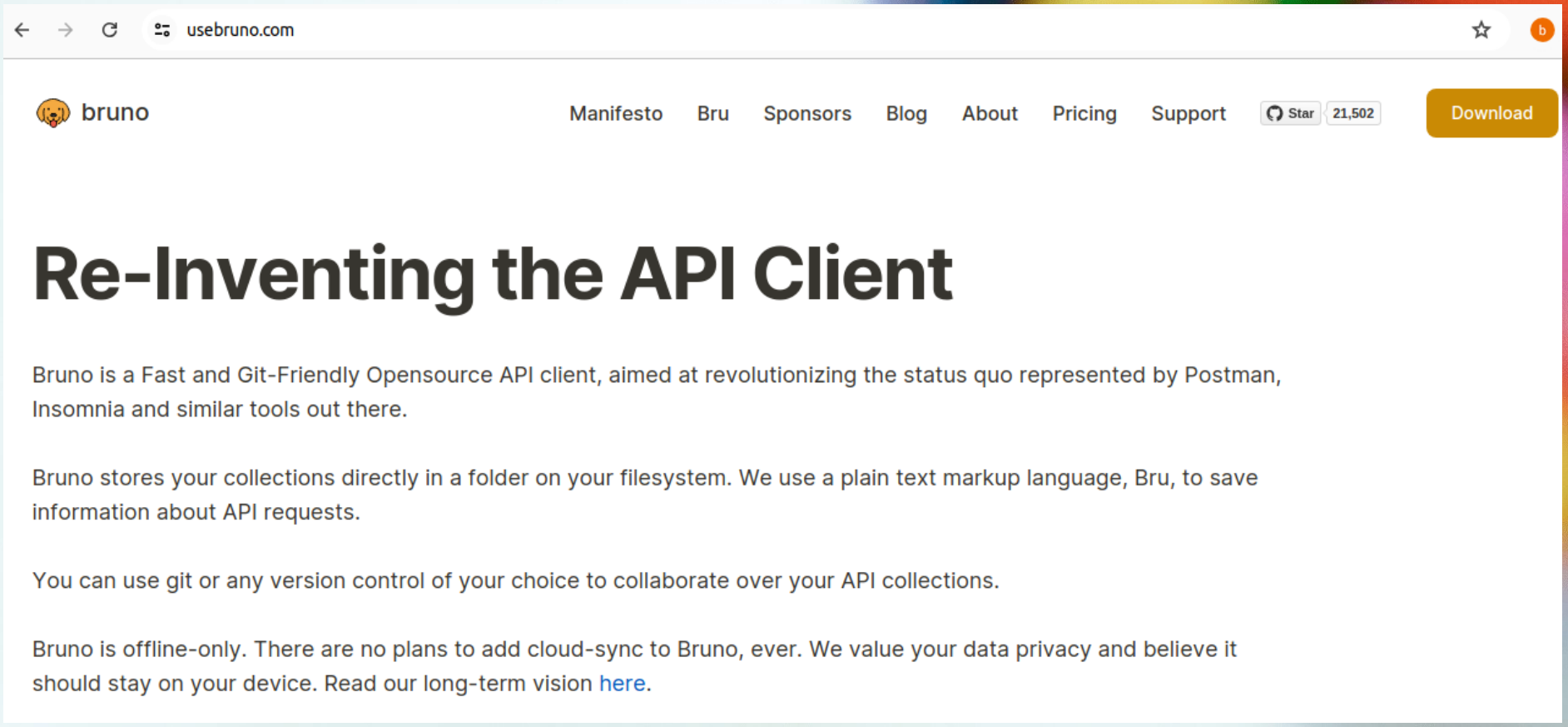
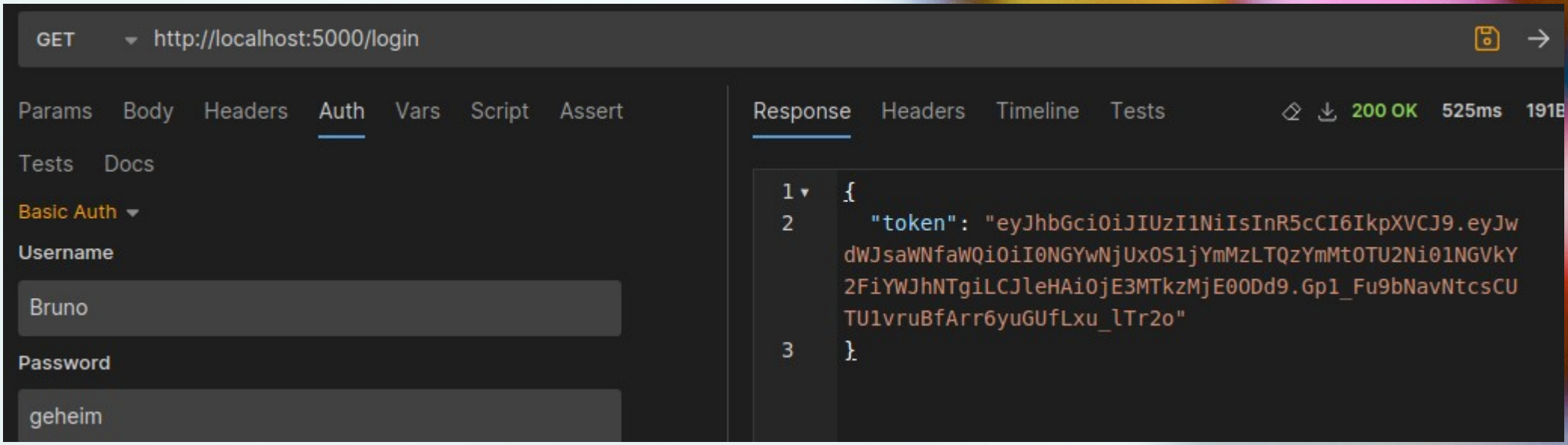
Flask / JWT REST API Environment



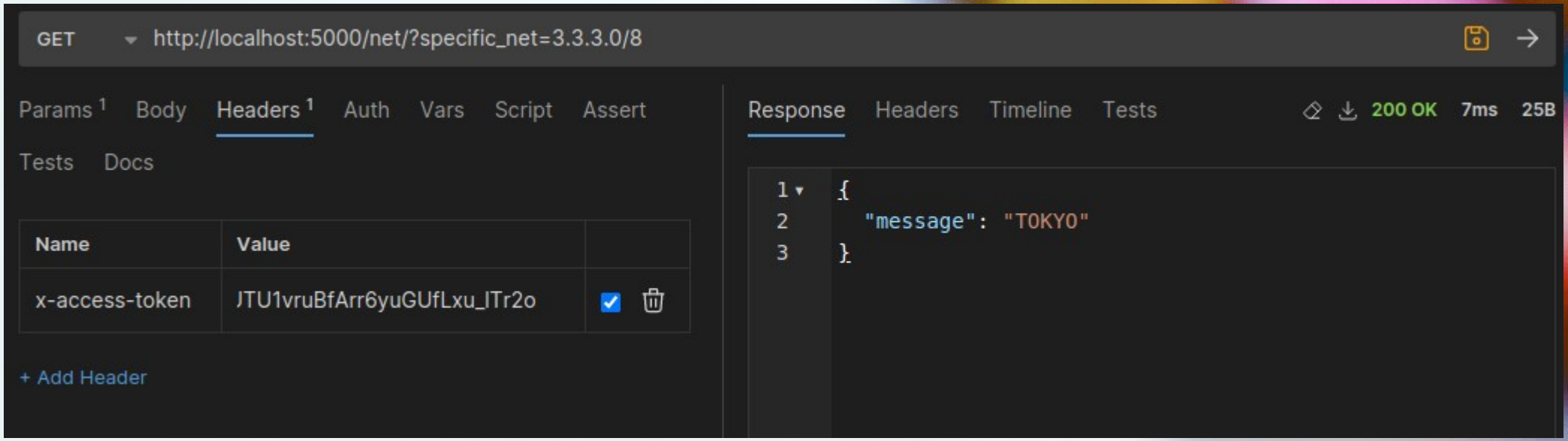

jwt.io
JWT
Debugger
Libraries
Introduction
Ask
Crafted by
Auth0
by Okta
JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.
JWT.IO allows you to decode, verify and generate JWT.
LEARN MORE ABOUT JWT
SEE JWT LIBRARIES


usebruno.com
bruno
Manifesto
Bru
Sponsors
Blog
About
Pricing
Support
Star
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

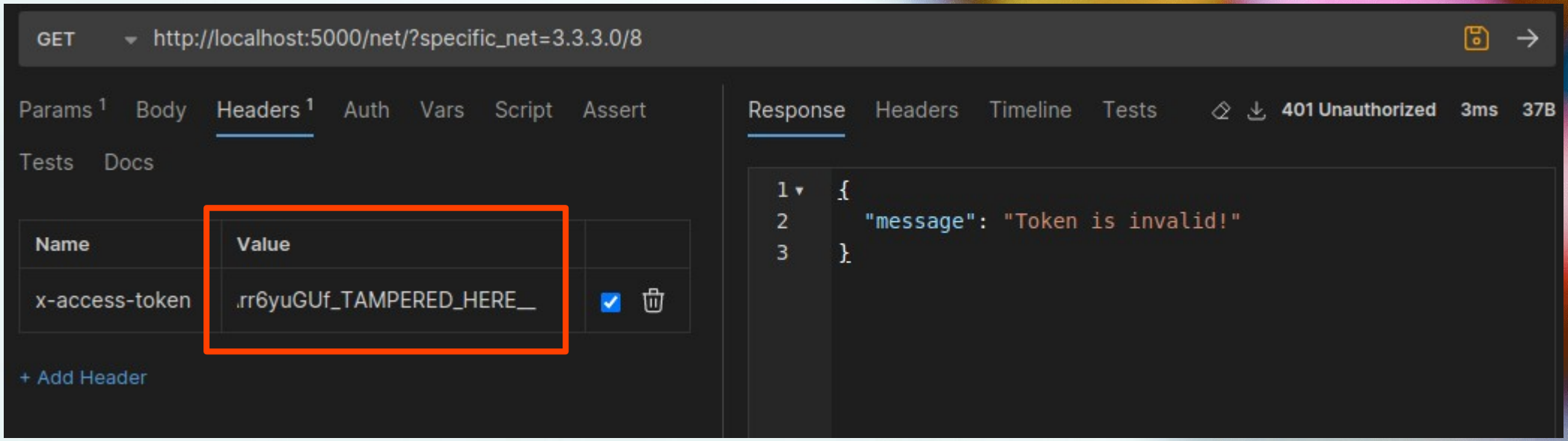


Request with valid token




```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwdWJsaWNfaWQiOiI0NGYwNjUxOS1jYmMzLTQzYmMtOTU2Ni01NGVkY2FiYWJhNTgiLCJleHAiOjE3MTkzMjE0ODd9.Gp1_Fu9bNavNtcsCUTU1vruBfArr6yuGUfLxu_lTr2o
{'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



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
```