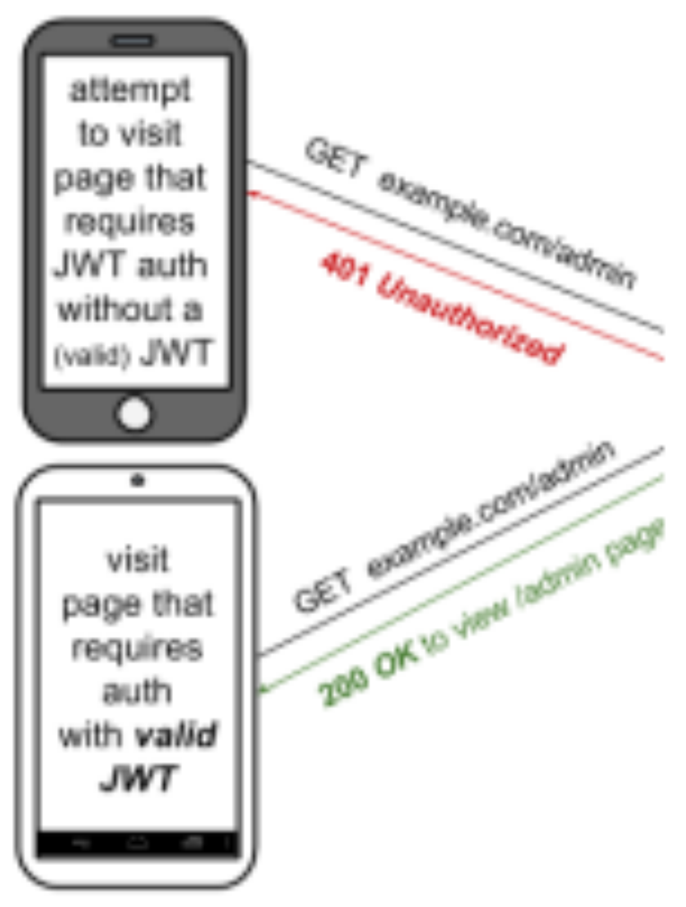
Secure Donation API

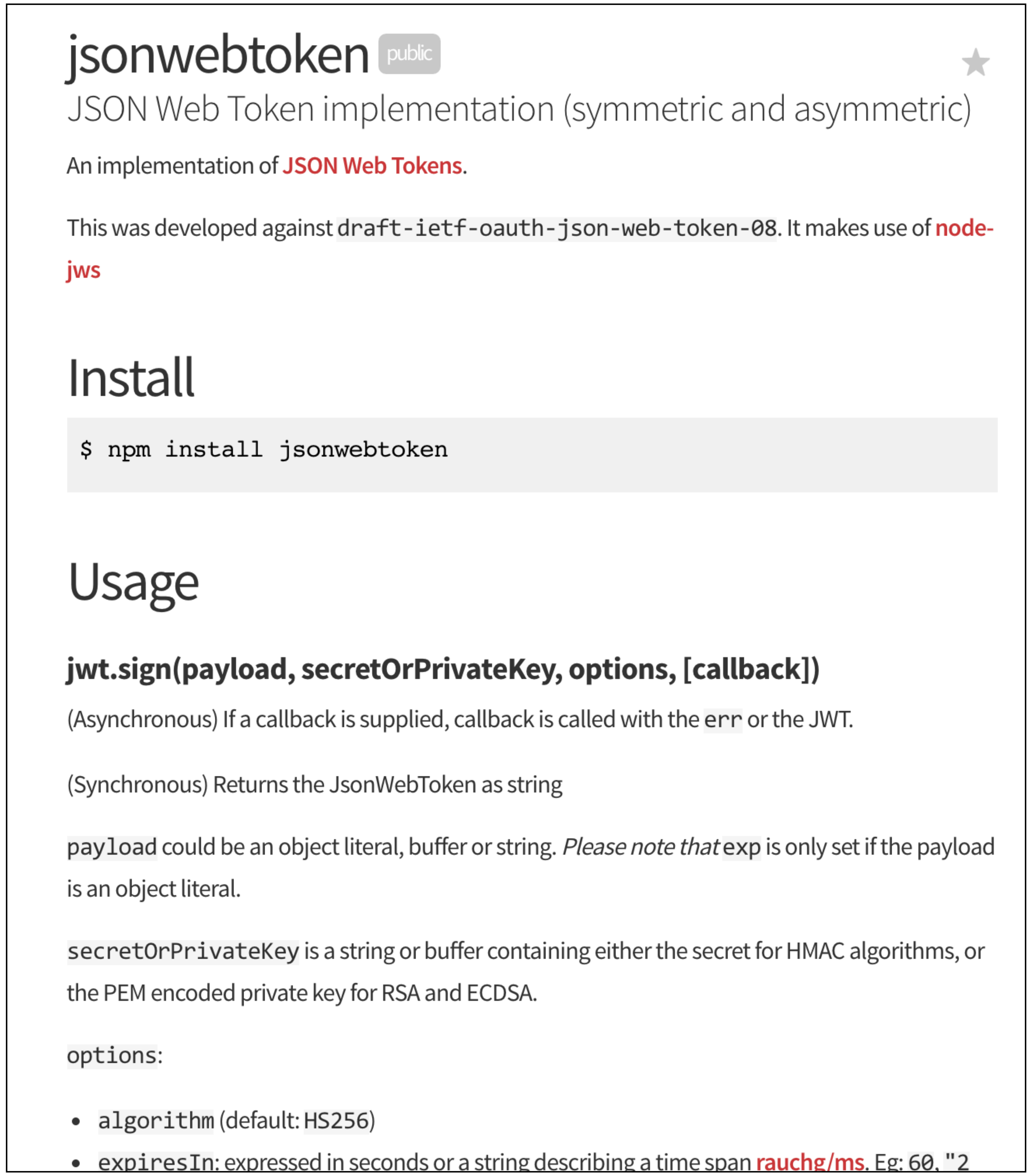
Jwt in HAPI


attempt to visit page that requires JWT auth without a (valid) JWT
GET example.com/admin
401 Unauthorized
visit page that requires auth with *valid* *JWT*
GET example.com/admin
200 OK to view /admin page

Integrating JWT into an API built using HAPI.

Agenda

- JWT Node Libraries
- Encoding & Decoding the Tokens
- The Authenticate Route
- Securing the API with a JWT Strategy
- Testing the Secured API



jsonwebtoken public

JSON Web Token implementation (symmetric and asymmetric)

An implementation of JSON Web Tokens.

This was developed against `draft-ietf-oauth-json-web-token-08`. It makes use of node-jws

Install

```
$ npm install jsonwebtoken
```

Usage

jwt.sign(payload, secretOrPrivateKey, options, [callback])

(Asynchronous) If a callback is supplied, callback is called with the `err` or the JWT.

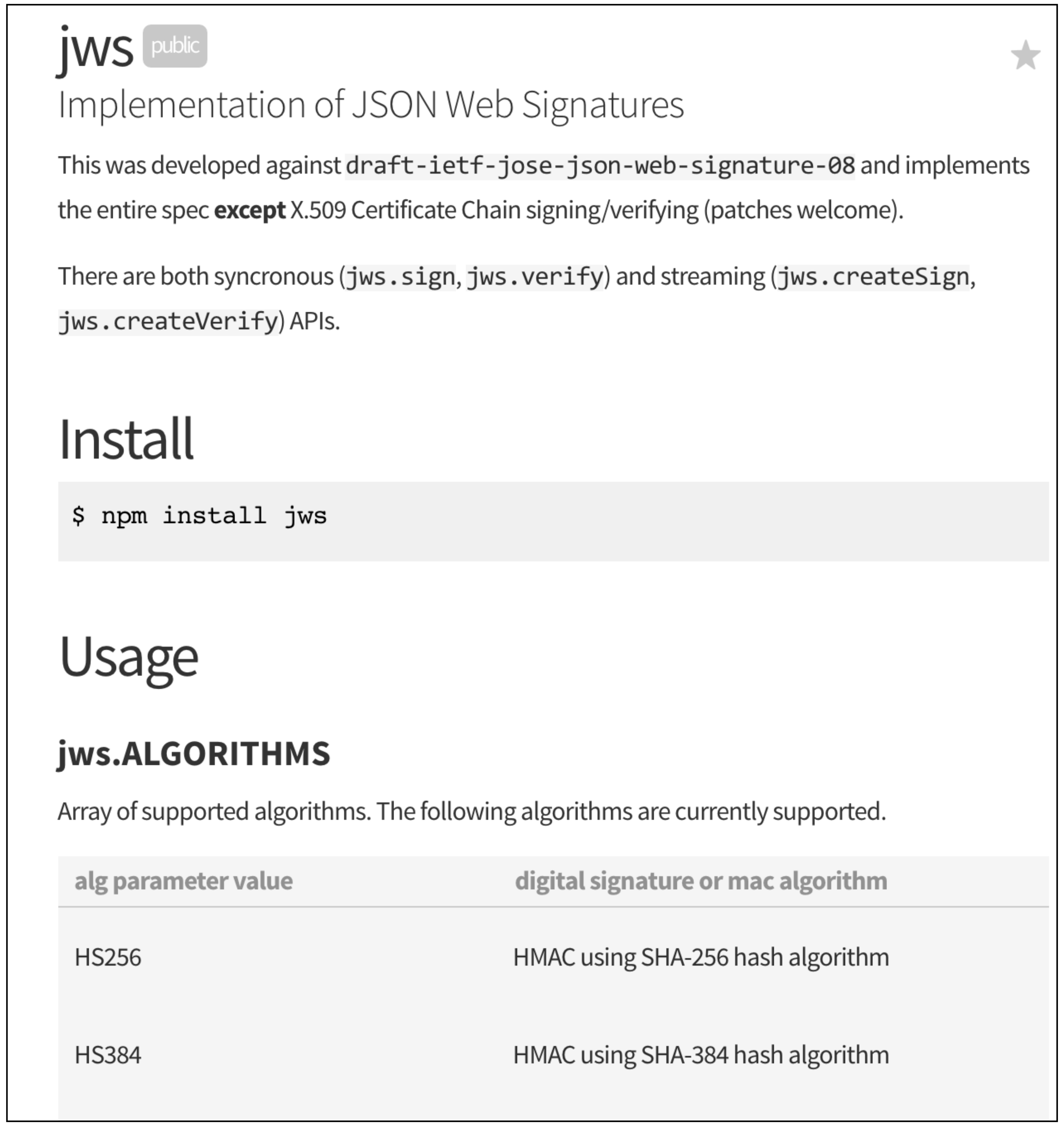
(Synchronous) Returns the JsonWebToken as string

`payload` could be an object literal, buffer or string. *Please note that* `exp` is only set if the payload is an object literal.

`secretOrPrivateKey` is a string or buffer containing either the secret for HMAC algorithms, or the PEM encoded private key for RSA and ECDSA.

`options`:

- `algorithm` (default: `HS256`)
- `expiresIn`: expressed in seconds or a string describing a time span rauchg/ms. Eg: `60`, `"2`



jws public

Implementation of JSON Web Signatures

This was developed against `draft-ietf-jose-json-web-signature-08` and implements the entire spec **except** X.509 Certificate Chain signing/verifying (patches welcome).

There are both syncronous (`jws.sign`, `jws.verify`) and streaming (`jws.createSign`, `jws.createVerify`) APIs.

Install

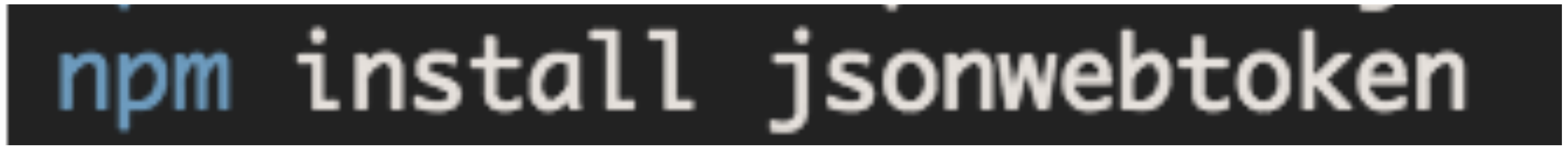
```
$ npm install jws
```

Usage

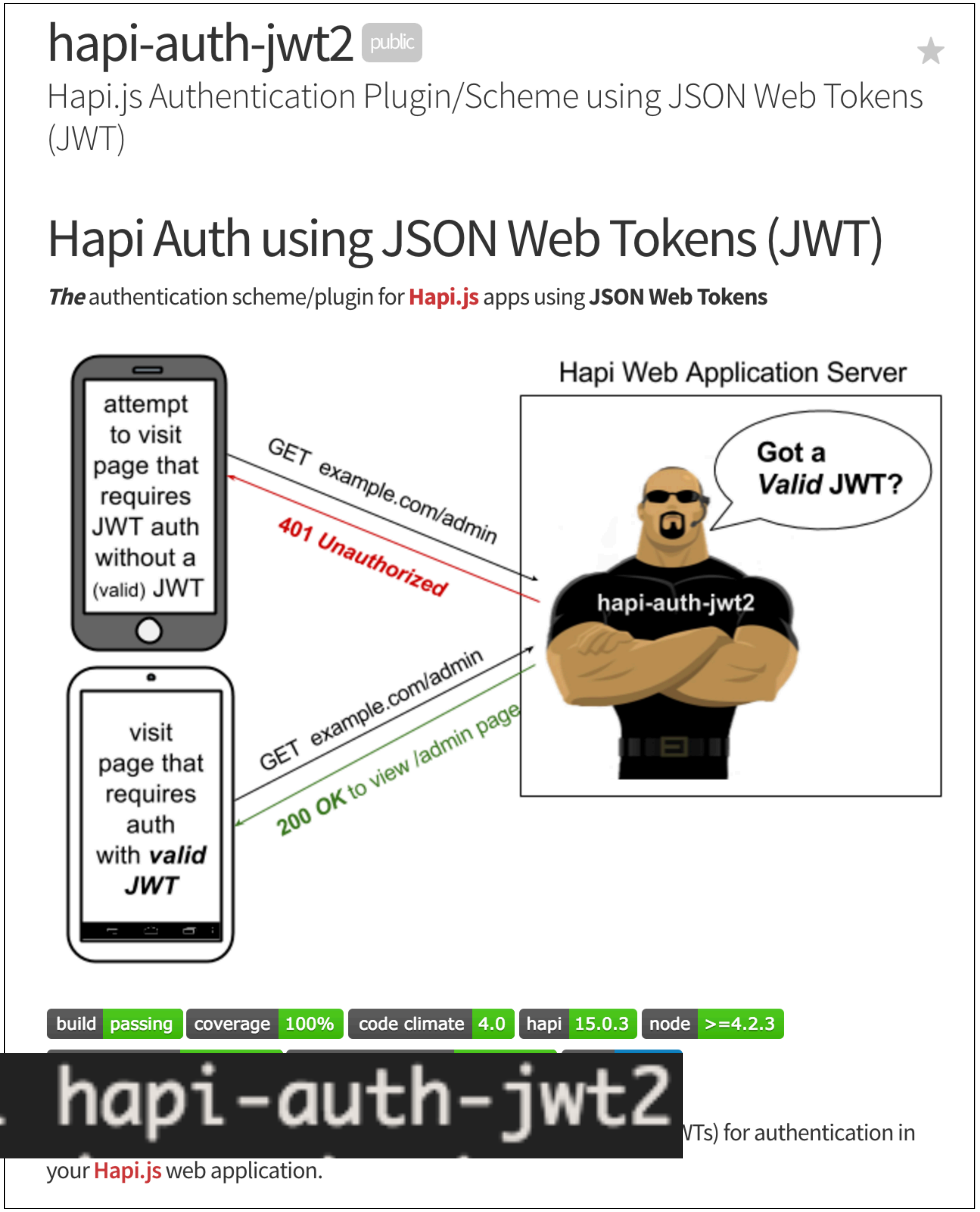
jws.ALGORITHMS

Array of supported algorithms. The following algorithms are currently supported.

alg parameter value	digital signature or mac algorithm
HS256	HMAC using SHA-256 hash algorithm
HS384	HMAC using SHA-384 hash algorithm



```
npm install jsonwebtoken
```

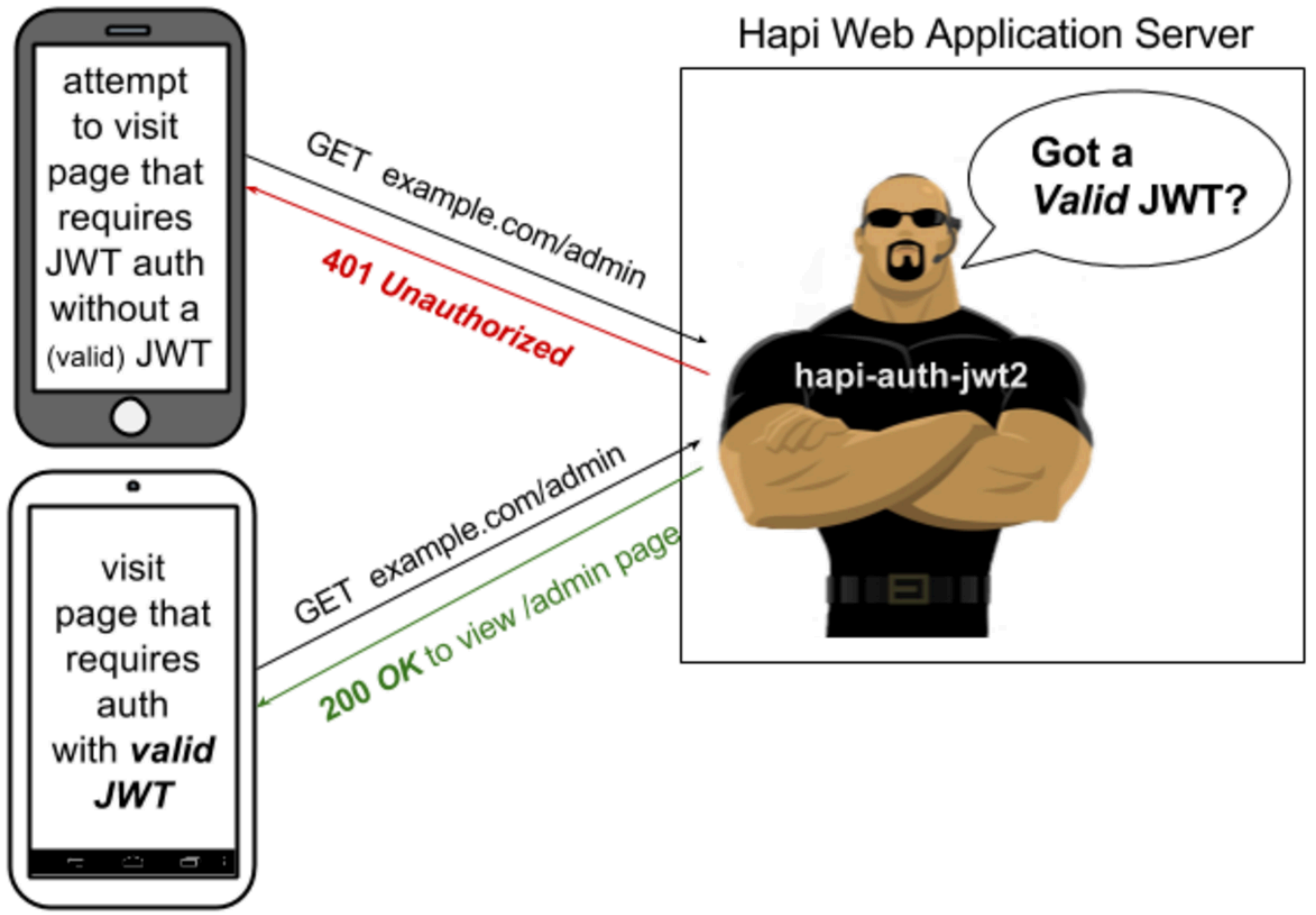


hapi-auth-jwt2 public

Hapi.js Authentication Plugin/Scheme using JSON Web Tokens (JWT)

Hapi Auth using JSON Web Tokens (JWT)

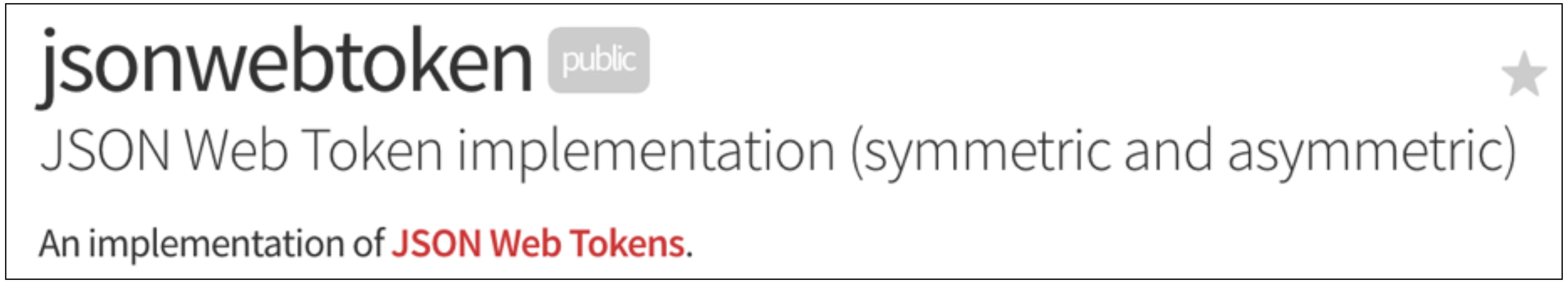
The authentication scheme/plugin for **Hapi.js** apps using **JSON Web Tokens**


attempt to visit page that requires JWT auth without a (valid) JWT
GET example.com/admin
401 Unauthorized
Hapi Web Application Server
Got a Valid JWT?
hapi-auth-jwt2
visit page that requires auth with *valid* *JWT*
GET example.com/admin
200 OK to view /admin page

build passing | coverage 100% | code climate 4.0 | hapi 15.0.3 | node >=4.2.3

```
npm install hapi-auth-jwt2
```

Ts) for authentication in your **Hapi.js** web application.


jsonwebtoken public

JSON Web Token implementation (symmetric and asymmetric)

An implementation of JSON Web Tokens.

- jwt.sign(payload, secretOrPrivateKey, options, [callback])
  - (Asynchronous) If a callback is supplied, callback is called with the err or the JWT.
  - (Synchronous) Returns the JsonWebToken as string
- payload could be an object literal, buffer or string.
- secretOrPrivateKey is a string the secret for HMAC

options

- algorithm (default: HS256)
- expiresIn: expressed in seconds or a string describing a time span rauchg/ms. Eg: 60, "2 days", "10h", "7d"
- notBefore: expressed in seconds or a string describing a time span rauchg/ms. Eg: 60, "2 days", "10h", "7d"
- audience
- issuer
- jwtid
- subject
- noTimestamp
- header

Utility functions to generate Token

```
const jwt = require('jsonwebtoken');

exports.createToken = function (user) {

  const payload = {
    id: user._id,
    email: user.email,
  };

  const options = {
    algorithm: 'HS256',
    expiresIn: '1h',
  };

  return jwt.sign(payload, 'secretpasswordnotrevealedtoanyone', options);
};
```

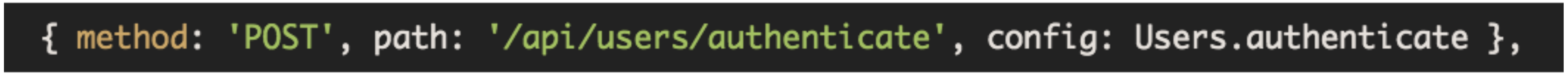
- Encode user database ID + email

Utility function to decode Token

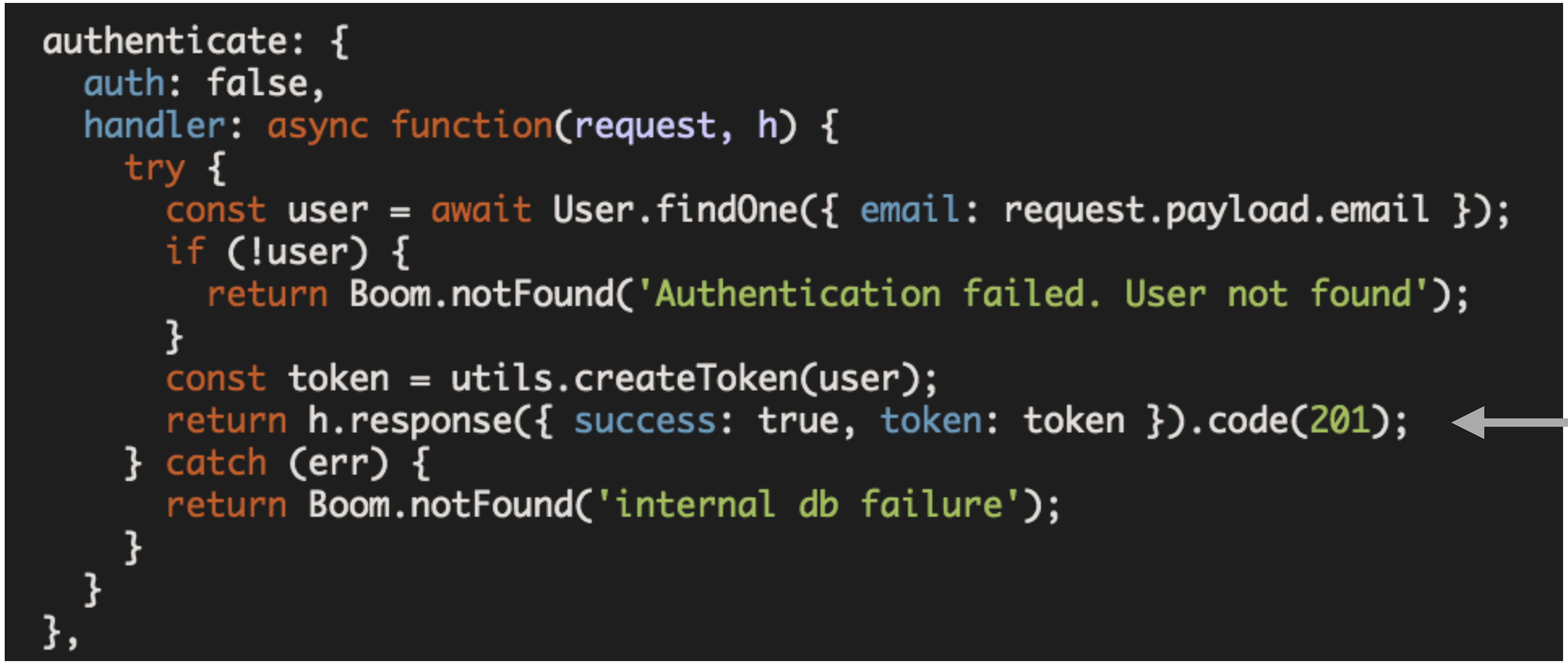
```
const jwt = require('jsonwebtoken');

exports.decodeToken = function (token) {
  const userInfo = {};
  try {
    var decoded = jwt.verify(token, 'secretpasswordnotrevealedtoanyone');
    userInfo.userId = decoded.id;
    userInfo.email = decoded.email;
  } catch (e) {
  }

  return userInfo;
};
```

- Recover the user database ID + email

Authenticate API Route



```
{ method: 'POST', path: '/api/users/authenticate', config: Users.authenticate },
```



```
authenticate: {
  auth: false,
  handler: async function(request, h) {
    try {
      const user = await User.findOne({ email: request.payload.email });
      if (!user) {
        return Boom.notFound('Authentication failed. User not found');
      }
      const token = utils.createToken(user);
      return h.response({ success: true, token: token }).code(201);
    } catch (err) {
      return Boom.notFound('internal db failure');
    }
  }
},
```

Authenticate route returns token, encoded using the utility function

Hapi Security Strategy : Cookies

- 'Standard' strategy specifies range or parameters, including:
  - password for securing cookie
  - cookie name
  - time to live (expiry)
- All routes are now 'guarded' by default, cookie based authentication mechanism

```
...
server.auth.strategy('standard', 'cookie', {
  password: 'secretpasswordnotrevealedtoanyone',
  cookie: 'donation-cookie',
  isSecure: false,
  ttl: 24 * 60 * 60 * 1000,
});

server.auth.default({
  strategy: 'standard',
});

...
```

Annotating Routes

- All routes are ‘guarded’ by default, cookie based authentication mechanism
- Any attempt to visit a route will be rejected unless valid cookie detected.
- Some routes are publicly available (signup or login)

```
...

server.auth.default({
  strategy: 'standard',
});

...
```

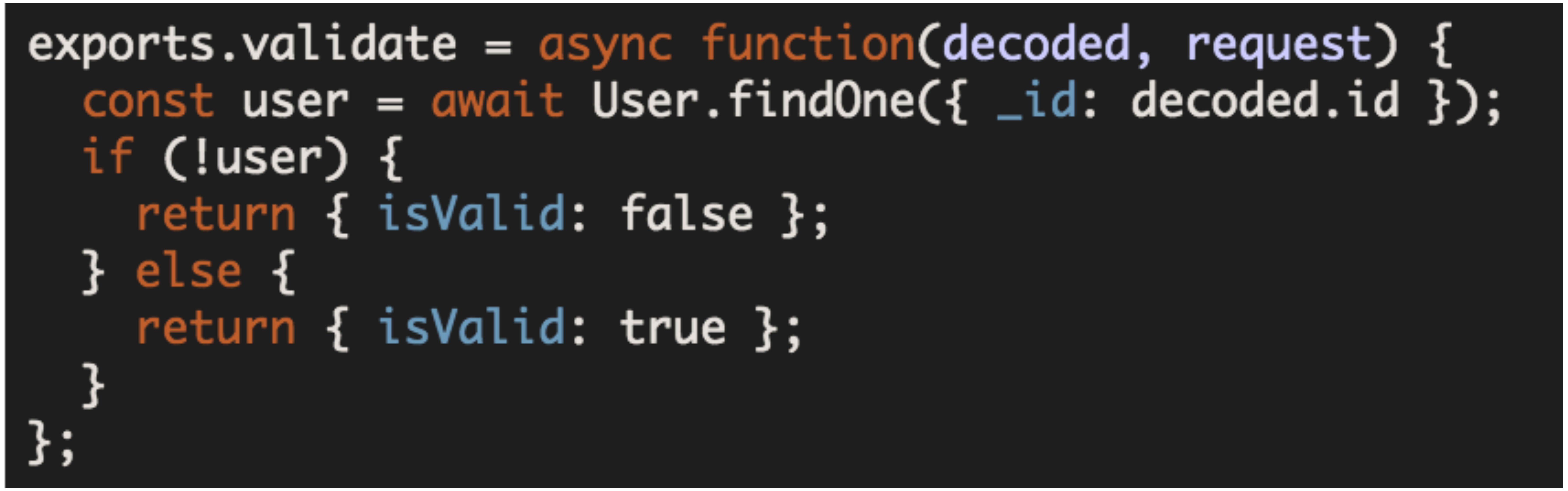
```
...
login: {
  auth: false,


signup: {
  auth: false,


...
```

Hapi Security Strategy : JWT

- Install additional strategy ‘jwt’ to be used for the API routes.
- Specifies private key + crypto algorithms
- Specifies **validateFunc** - which will be invoked to validate the token prior to triggering a route.

```
server.auth.strategy('jwt', 'jwt', {
  key: 'secretpasswordnotrevealedtoanyone',
  validateFunc: utils.validate,
  verifyOptions: { algorithms: ['HS256'] },
});
```

validateFunc



```
exports.validate = async function(decoded, request) {
  const user = await User.findOne({ _id: decoded.id });
  if (!user) {
    return { isValid: false };
  } else {
    return { isValid: true };
  }
};
```

- Invoked on routes marked with the ‘jwt’ strategy.
- Passed a decoded token
- Check to see if ID in token == valid id in the database
- Invoked callback with err, true/false

—> This will determine if route can be invoked

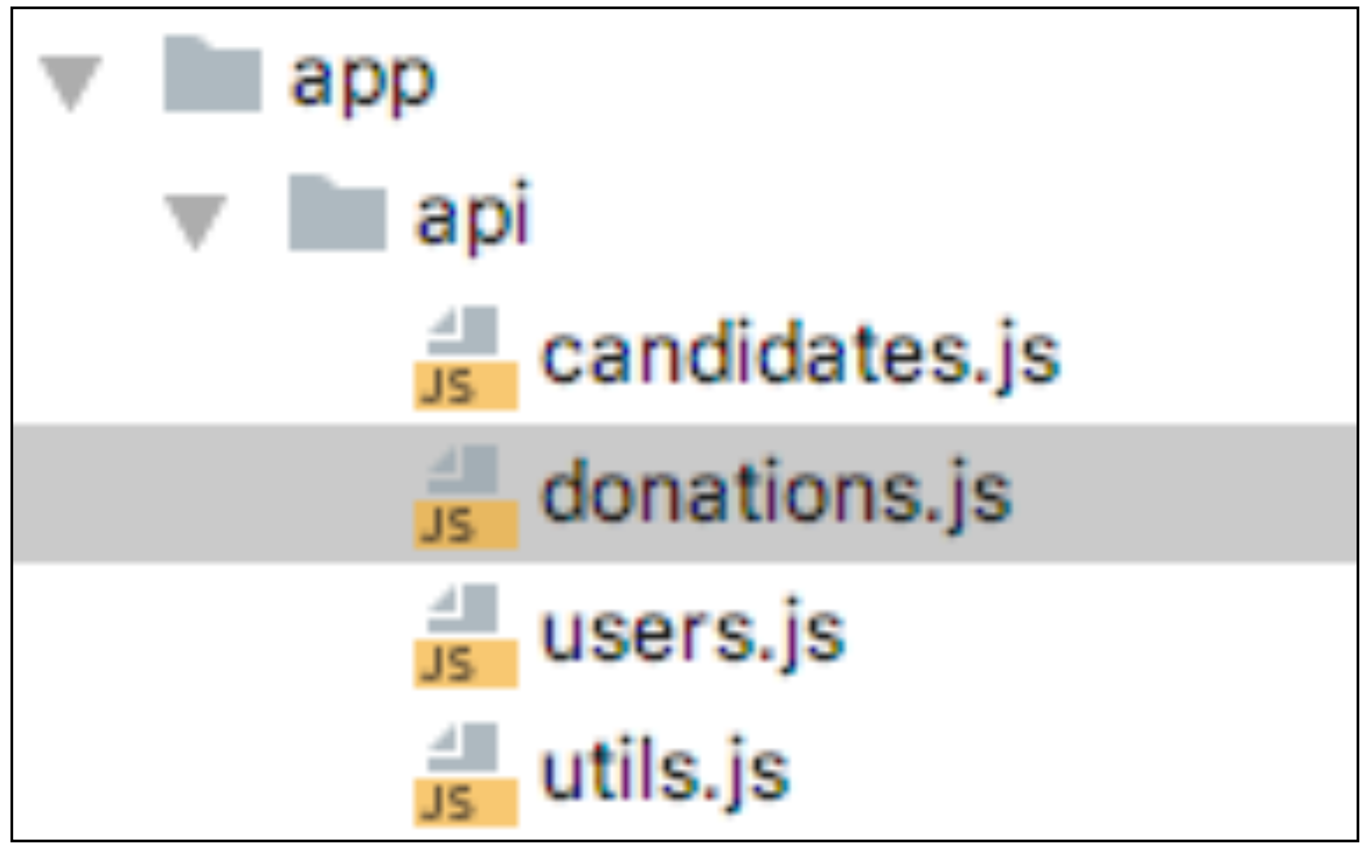
gerUserIdFromRequest

```
exports.getUserIdFromRequest = function(request) {
  var userId = null;
  try {
    const authorization = request.headers.authorization;
    var token = authorization.split(' ')[1];
    var decodedToken = jwt.verify(token, 'secretpasswordnotrevealedtoanyone');
    userId = decodedToken.id;
  } catch (e) {
    userId = null;
  }
  return userId;
};
```

- utility method to decode token, recover and return user id

All API Routes given JWT Strategy

```
server.auth.strategy('jwt', 'jwt', {
  key: 'secretpasswordnotrevealedtoanyone',
  validateFunc: utils.validate,
  verifyOptions: { algorithms: ['HS256'] },
});
```



Strategy

```
makeDonation: {
  auth: {
    strategy: 'jwt',
  },

  handler: async function(request, h) {
    const userId = utils.getUserIdFromRequest(request);
    let donation = new Donation(request.payload);
    const candidate = await Candidate.findOne({ _id: request.params.id });
    if (!candidate) {
      return Boom.notFound('No Candidate with this id');
    }
    donation.candidate = candidate._id;
    donation.donor = userId;
    donation = await donation.save();
    return donation;
  }
},
```

Testing Auth

- New method in DonationService class to authenticate users

```
suite(‘Auth API tests', function () {

  let users = fixtures.users;
  let newUser = fixtures.newUser;

  const donationService = new DonationService(fixtures.donationService);

  setup(async function () {
    await donationService.deleteAllUsers();
  });

  test('authenticate', async function () {
    const returnedUser = await donationService.createUser(newUser);
    const response = await donationService.authenticate(newUser);
    assert(response.success);
    assert.isDefined(response.token);
  });

});
```

Client Autnhenticate Method

- Perform an authentication post request and retrieve the token
- Set the token as a header on all subsequent requests…
- … until removed by a clearAuth method

```
class DonationService {
  constructor(baseUrl) {
    this.baseUrl = baseUrl;
  }

  async authenticate(user) {
    try {
      const response = await axios.post(this.baseUrl + '/api/users/authenticate', user);
      axios.defaults.headers.common['Authorization'] = 'Bearer ' + response.data.token;
      return response.data;
    } catch (e) {
      return null;
    }
  }

  async clearAuth(user) {
    axios.defaults.headers.common['Authorization'] = '';
  }
```