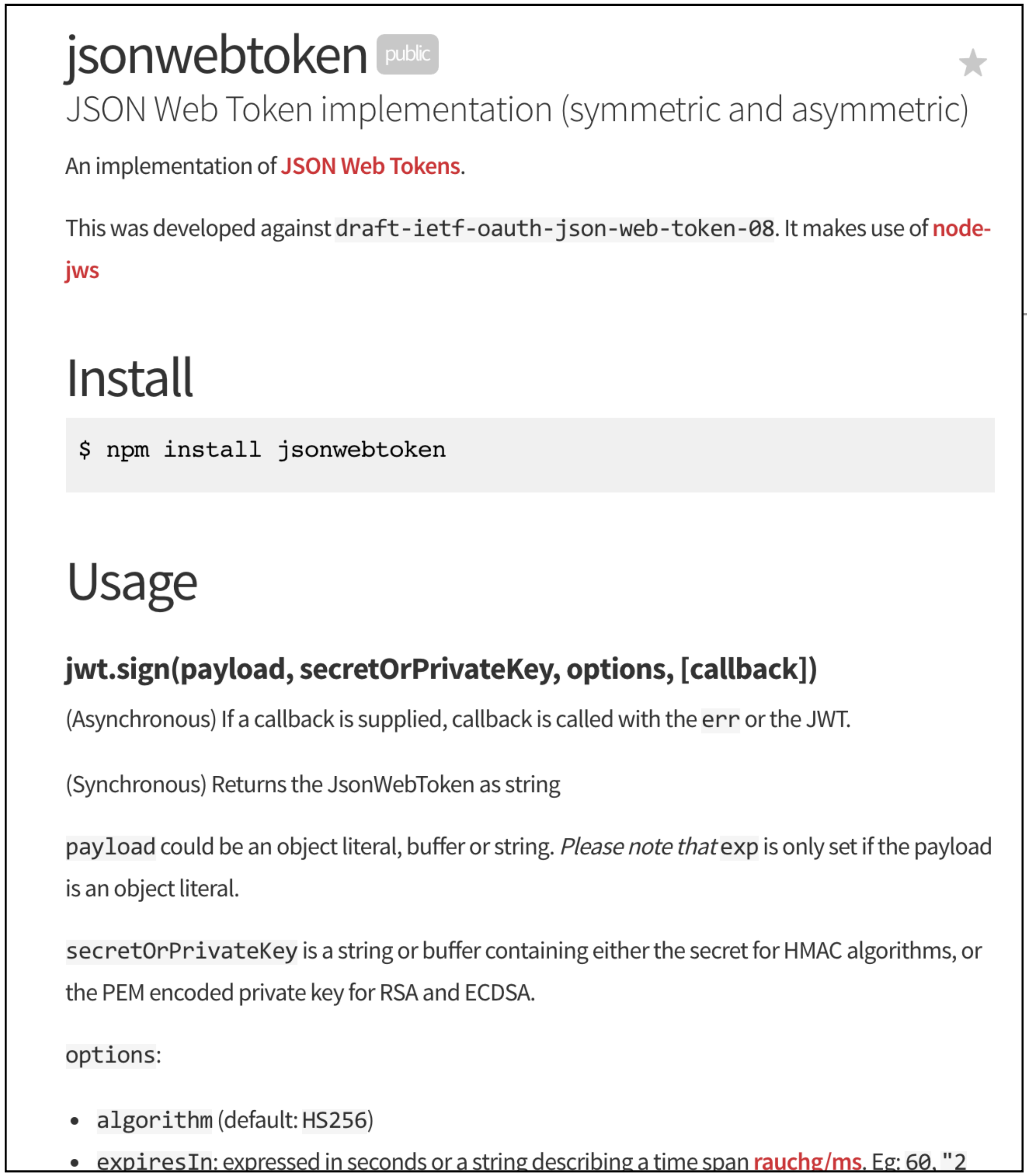
Secure Donation API

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.

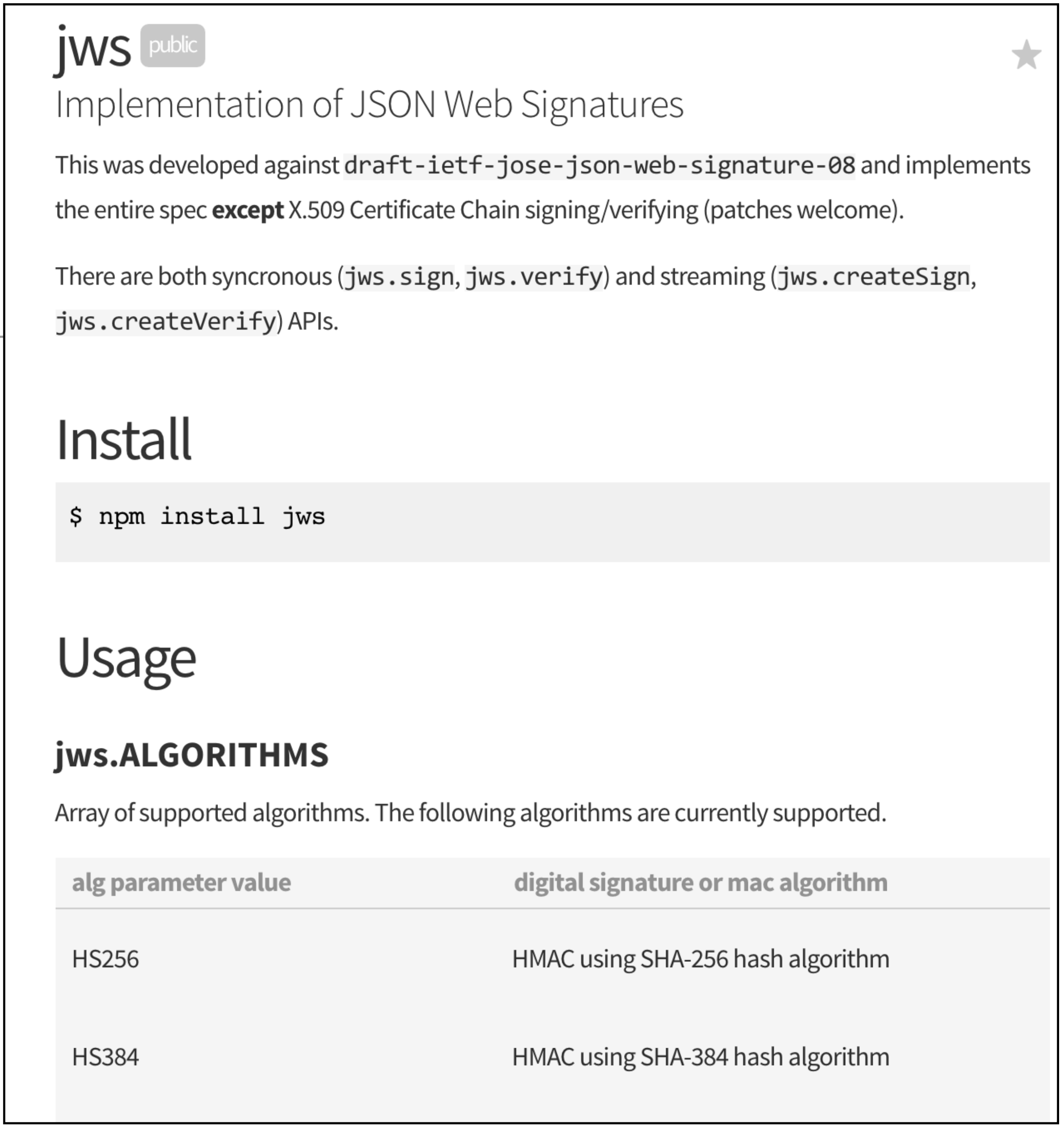
(Synchronous) Returns the JsonWebToken as string

`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
```

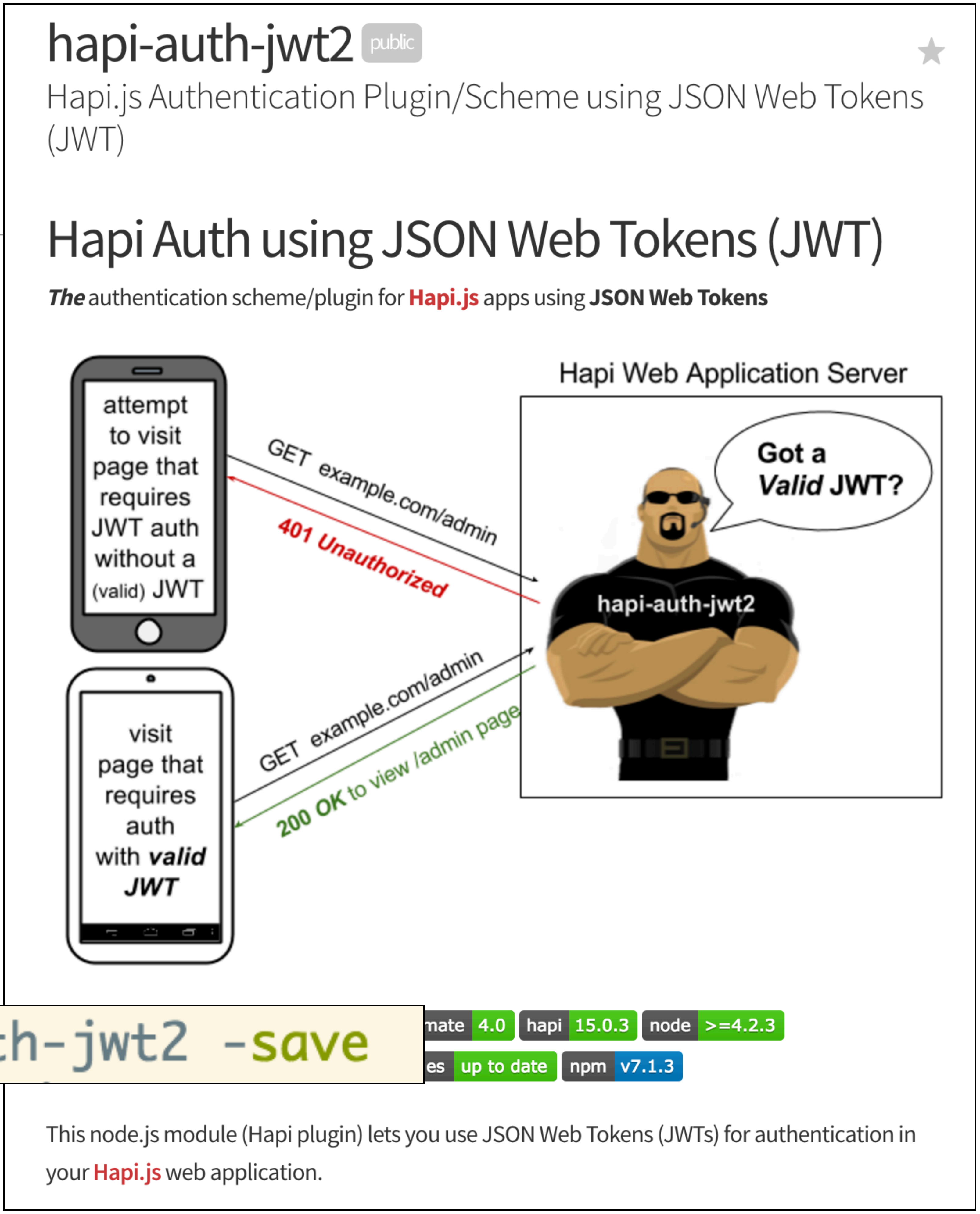
Usage

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 -save
```

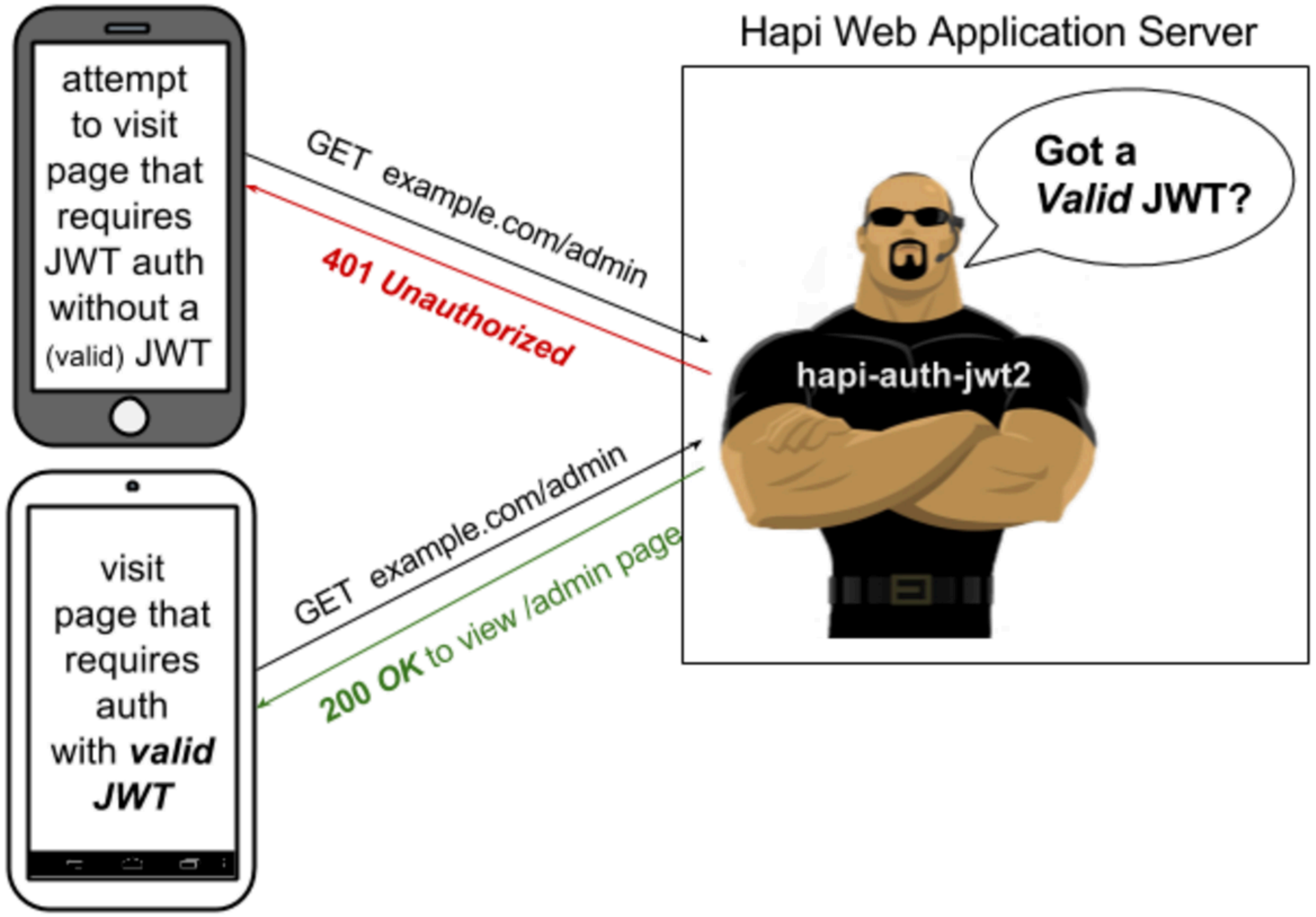


hapi-auth-jwt2 public

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**

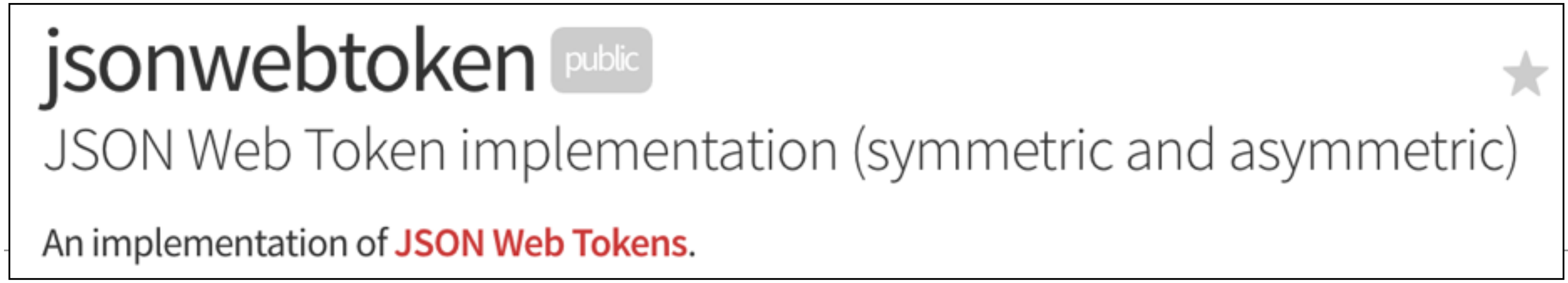

attempt to visit page that requires JWT auth without a (valid) JWT
GET example.com/admin
401 Unauthorized
Hapi Web Application Server
Got a Valid JWT?
hapi-auth-jwt2
visit page that requires auth with valid JWT
GET example.com/admin
200 OK to view /admin page

mate 4.0 | hapi 15.0.3 | node >=4.2.3

es up to date | npm v7.1.3

This node.js module (Hapi plugin) lets you use JSON Web Tokens (JWTs) for authentication in your Hapi.js web application.

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


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 functions 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: UsersApi.authenticate },
```

Authenticate route returns token, encoded using the utility function

```
exports.authenticate = {
  auth: false,
  handler: function (request, reply) {
    const user = request.payload;
    User.findOne({ email: user.email }).then(foundUser => {
      if (foundUser && foundUser.password === user.password) {

        const token = utils.createToken(foundUser);
        reply({ success: true, token: token }).code(201);

      } else {
        reply({ success: false, message: 'Authentication failed. User not found.' }).code(201);
      }
    }).catch(err => {
      reply(Boom.notFound('internal db failure'));
    });
  },

};
```

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

```
...

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

...
```

- Any attempt to visit a route will be rejected unless valid cookie detected.
- Some routes are publicly available (signup or login)

```
...
exports.signup = {
  auth: false,
  handler: function (request, reply) {
    reply.view('signup', { title: 'Sign up for Donations' });
  },

};

exports.login = {
  auth: false,
  handler: function (request, reply) {
    reply.view('login', { title: 'Login to Donations' });
  },

};
...
```

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

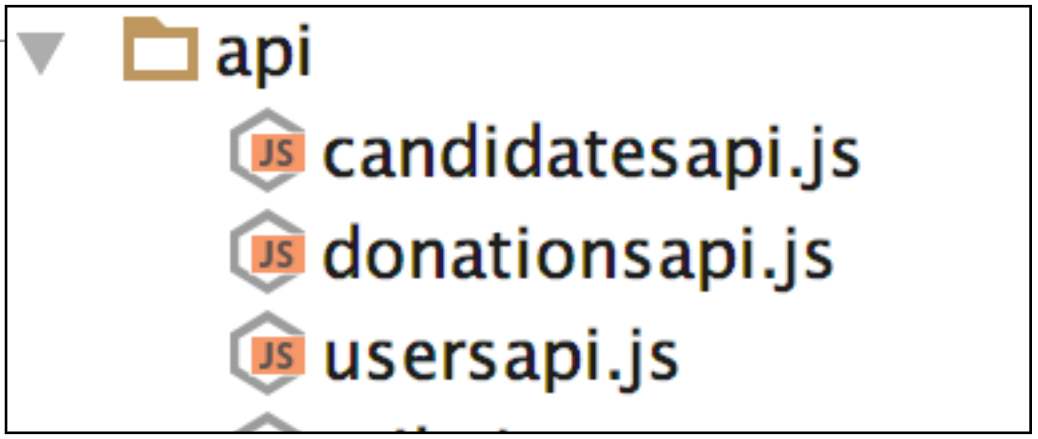
```
exports.validate = function (decoded, request, callback) {
  User.findOne({ _id: decoded.id }).then(user => {
    if (user != null) {
      callback(null, true);
    } else {
      callback(null, false);
    }
  }).catch(err => {
    callback(err, false);
  });
}
```

- 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

All API Routes given JWT Strategy

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



```
api
    candidatesapi.js
    donationsapi.js
    usersapi.js
```

Strategy

```
exports.makeDonation = {

  auth: {
    strategy: 'jwt',
  },

  handler: function (request, reply) {
    const donation = new Donation(request.payload);
    donation.candidate = request.params.id;
    donation.donor = utils.getUserIdFromRequest(request);
    donation.save().then(newDonation => {
      reply(newDonation).code(201);
    }).catch(err => {
      reply(Boom.badImplementation('error making donation'));
    });
  },

};
```

Auth Unit Test

- Simple sanity test
- Doesn't check for correct error codes
- Auth fully encapsulated in **donationService** class

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

  let users = fixtures.users;
  let candidates = fixtures.candidates;

  const donationService = new DonationService(fixtures.donationService);

  test('login-logout', function () {
    var returnedCandidates = donationService.getCandidates();
    assert.isNull(returnedCandidates);

    const response = donationService.login(users[0]);
    returnedCandidates = donationService.getCandidates();
    assert.isNotNull(returnedCandidates);

    donationService.logout();
    returnedCandidates = donationService.getCandidates();
    assert.isNull(returnedCandidates);
  });
});
```

Access should be denied

Logged in, we should get a (perhaps empty) candidate list

Logged out, should get null.

DonationService

```
class DonationService {

  ...

  login(user) {
    return this.httpService.setAuth('/api/users/authenticate', user);
  }

  logout() {
    this.httpService.clearAuth();
  }

  ...
}
```

- New functions **login** and **logout**
- These defer to **setAuth** and **clearAuth** functions in SyncHttpService

SyncHttpService - setAuth() & clearAuth()

- Post the user credentials to the service
- If success (201), then recover the token
- Store the Token in **authHeadder** attribute
- Clear the header in **clearAuth**

```
class SyncHttpService {

  constructor(baseUrl) {
    this.baseUrl = baseUrl;
    this.authHeadder = null;
  }

  setAuth(url, user) {
    const res = request('POST', this.baseUrl + url, { json: user });
    if (res.statusCode == 201) {
      var payload = JSON.parse(res.getBody('utf8'));
      if (payload.success) {
        this.authHeadder = { Authorization: 'bearer ' + payload.token, };
        return true;
      }
    }

    this.authHeadder = null;
    return false;
  }

  clearAuth() {
    this.authHeadder = null;
  }

...
}
```

SyncHttpService

- Remaining methods pass the token (if present)

```
class SyncHttpService {

  constructor(baseUrl) {
    this.baseUrl = baseUrl;
    this.authHeadder = null;
  }

  get(url) {
    var returnedObj = null;
    var res = request('GET', this.baseUrl + url, { headers: this.authHeadder });
    if (res.statusCode < 300) {
      returnedObj = JSON.parse(res.getBody('utf8'));
    }

    return returnedObj;
  }

  post(url, obj) {
    var returnedObj = null;
    var res = request('POST', this.baseUrl + url, { json: obj, headers: this.authHeadder });
    if (res.statusCode < 300) {
      returnedObj = JSON.parse(res.getBody('utf8'));
    }

    return returnedObj;
  }

  delete(url) {
    var res = request('DELETE', this.baseUrl + url, { headers: this.authHeadder });
    return res.statusCode;
  }
}
```