

JWT/HAPI



Full Stack Web Development

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 synchronous (`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
```



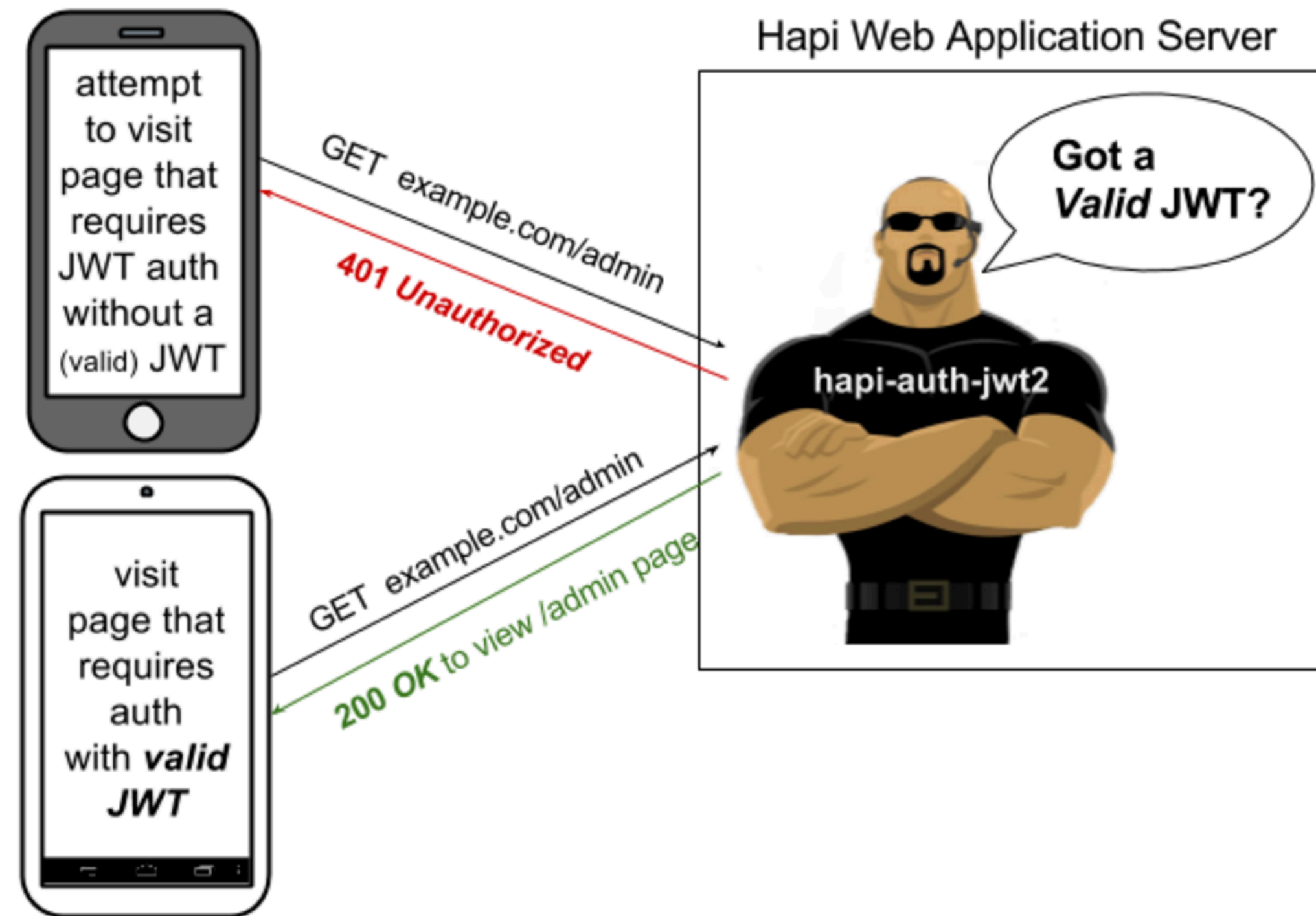
hapi-auth-jwt2 public



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**



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

0.3 node >=4.2.3

m v7.1.3

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

jsonwebtoken public

JSON Web Token implementation (symmetric and asymmetric)

An implementation of **JSON Web Tokens**.

options

- `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

- 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

```
import jwt from "jsonwebtoken";
import { db } from "../models/db.js";
import dotenv from "dotenv";

const result = dotenv.config();

export function createToken(user) {
  const payload = {
    id: user._id,
    email: user.email
  };
  const options = {
    algorithm: "HS256",
    expiresIn: "1h"
  };
  return jwt.sign(payload, process.env.cookie_password, options);
}
```

- Encode id + email
- HS256 encoding
- cookie password from .env

Utility function to decode Token

```
export function decodeToken(token) {  
  var userInfo = {};  
  try {  
    var decoded = jwt.verify(token, process.env.cookie_password);  
    userInfo.userId = decoded.id;  
    userInfo.email = decoded.email;  
  } catch (e) {  
    console.log(e.message);  
  }  
  return userInfo;  
}
```

- Recover user ID and email from token

Utility function to validate Token

```
export async function validate(decoded, request) {  
  const user = await db.userStore.getUserById(decoded.id);  
  if (!user) {  
    return { isValid: false };  
  } else {  
    return { isValid: true, credentials: user };  
  }  
}
```

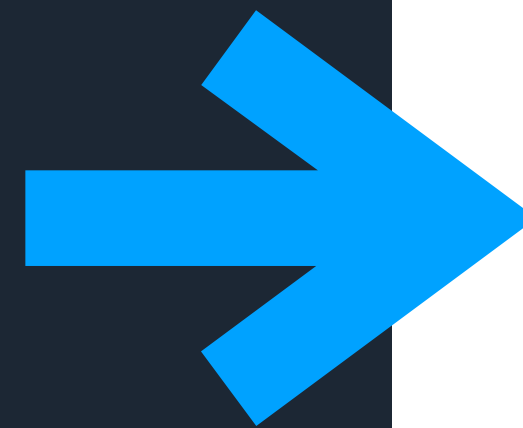
- Hapi security strategy
Validate Function

Configure JWT

server.js

```
import jwt from "hapi-auth-jwt2";
import { validate } from "../api/jwt-utils.js";
...
await server.register(jwt);
...
```

```
server.auth.strategy("jwt", "jwt", {
  key: process.env.cookie_password,
  validate: validate,
  verifyOptions: { algorithms: ["HS256"] }
});
```



- Define a new strategy “jwt”
- The default is still “session”, which uses cookie authentication

```
export async function validate(decoded, request) {
  const user = await db.userStore.getUserById(decoded.id);
  if (!user) {
    return { isValid: false };
  } else {
    return { isValid: true, credentials: user };
  }
}
```

api-routes.js

```
{ method: "POST", path: "/api/users/authenticate", config: userApi.authenticate },
```

users-api.js

```
import { createToken } from "../jwt-utils.js";
...

authenticate: {
  auth: false,
  handler: async function(request, h) {
    try {
      const user = await db.userStore.getUserByEmail(request.payload.email);
      if (!user) {
        return Boom.unauthorized("User not found");
      } else if (user.password !== request.payload.password) {
        return Boom.unauthorized("Invalid password");
      } else {
        const token = createToken(user);
        return h.response({ success: true, token: token }).code(201);
      }
    } catch (err) {
      return Boom.serverUnavailable("Database Error");
    }
  }
}
```

Authenticate API Route

- If valid user:
- Create token
- Return token as part of response

Recap : Cookie Authentication


- Set the cookie when a user is authenticated
- Cookie contains the logged in user ID



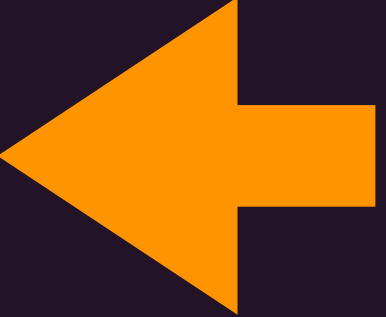
```
login: {  
  handler: async function (request, h) {  
    const { email, password } = request.payload;  
    const user = await db.userStore.getUserByEmail(email);  
    if (!user || user.password !== password) {  
      return h.redirect("/");  
    }  
    request.cookieAuth.set({ id: user._id });  
    return h.redirect("/dashboard");  
  },  
},
```

Cookie Configuration 1

- Set an auth 'strategy' before application starts
- Specifies range or parameters, including:
 - password for securing cookie
 - cookie name

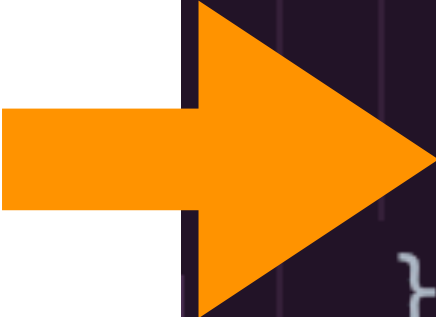


```
server.auth.strategy("session", "cookie", {  
  cookie: {  
    name: process.env.cookie_name,  
    password: process.env.cookie_password,  
    isSecure: false,  
  },  
  redirectTo: "/",  
  validateFunc: accountsController.validate,  
});  
server.auth.default("session");
```



Cookie Configuration 2

- By default hapi-auth-cookie will only allow the cookie to be transferred over a secure TLS/SSL connection.
- This may not be convenient during development so you can set the `isSecure` option to `false`.



```
server.auth.strategy("session", "cookie", {
  cookie: {
    name: process.env.cookie_name,
    password: process.env.cookie_password,
    isSecure: false,
  },
  redirectTo: "/",
  validateFunc: accountsController.validate,
});
server.auth.default("session");
```

- 
- Set 'standard' as the default strategy for all routes

Validate function

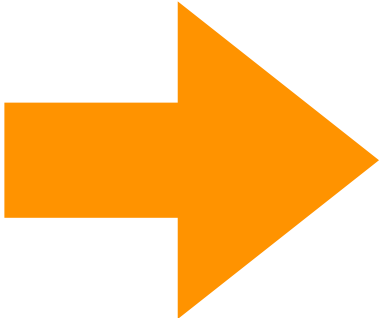
```
validateFunc: accountsController.validate,
```

- A hook function triggered when before a route guard by this strategy is to be engaged
- Has access to the cookie via session parameter
- Perform additional checks and return result of these checks


```
async validate(request, session) {  
  const user = await db.userStore.getUserById(session.id);  
  if (!user) {  
    return { valid: false };  
  }  
  return { valid: true, credentials: user };  
},
```

Annotating Routes

- All routes are now 'guarded' by default, cookie based authentication mechanism
- Any attempt to visit a route will be rejected unless valid cookie detected.
- Some routes need to be available (to signup or login for instance)
- These routes must specifically disable auth mechanism



```
server.auth.default("session");
```



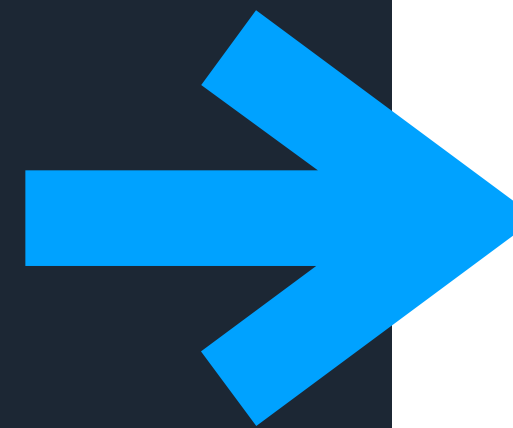
```
export const accountsController = {  
  index: {  
    auth: false,  
    handler: function (request, h) {  
      return h.view("main", { title: "Welcome to Playlist" });  
    },  
  },  
  showSignup: {  
    auth: false,  
    handler: function (request, h) {  
      return h.view("signup-view", { title: "Sign up for Playlist" });  
    },  
  },  
  signup: {  
    auth: false,  
    handler: async function (request, h) {  
      const user = request.payload;  
      await db.userStore.addUser(user);  
      return h.redirect("/");  
    },  
  },  
  showLogin: {  
    auth: false,  
    handler: function (request, h) {  
      return h.view("login-view", { title: "Login to Playlist" });  
    },  
  },  
}
```

Configure JWT

server.js

```
import jwt from "hapi-auth-jwt2";
import { validate } from "../api/jwt-utils.js";
...
await server.register(jwt);
...
```

```
server.auth.strategy("jwt", "jwt", {
  key: process.env.cookie_password,
  validate: validate,
  verifyOptions: { algorithms: ["HS256"] }
});
```



- Define a new strategy “jwt”
- The default is still “session”, which uses cookie authentication

```
export async function validate(decoded, request) {
  const user = await db.userStore.getUserById(decoded.id);
  if (!user) {
    return { isValid: false };
  } else {
    return { isValid: true, credentials: user };
  }
}
```

Hapi Security Strategy : JWT

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

```
server.auth.strategy("jwt", "jwt", {  
  key: process.env.cookie_password,  
  validate: validate,  
  verifyOptions: { algorithms: ["HS256"] }  
});
```

Validate

```
export async function validate(decoded, request) {  
  const user = await db.userStore.getUserById(decoded.id);  
  if (!user) {  
    return { isValid: false };  
  } else {  
    return { isValid: true, credentials: user };  
  }  
}
```

- Invoked on routes marked with the 'jwt' strategy.
 - Passed a decoded token
 - Check to see if ID in token == valid id in the database
- > This will determine if route can be invoked

All API Routes given JWT Strategy

```
export const playlistApi = {
  find: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request, h) {...},
    tags: ["api"],
    response: { schema: PlaylistArraySpec, failAction: validationError },
    description: "Get all playlists",
    notes: "Returns all playlists",
  },

  findOne: {
    auth: {
      strategy: "jwt",
    },
    async handler(request) {...},
    tags: ["api"],
    description: "Find a Playlist",
    notes: "Returns a playlist",
    validate: { params: { id: IdSpec }, failAction: validationError },
    response: { schema: PlaylistSpecPlus, failAction: validationError },
  },

  create: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request, h) {...},
    tags: ["api"],
    description: "Create a Playlist",
    notes: "Returns the newly created playlist",
    validate: { payload: PlaylistSpec, failAction: validationError },
    response: { schema: PlaylistSpecPlus, failAction: validationError },
  },
```

```
deleteOne: {
  auth: {
    strategy: "jwt",
  },
  handler: async function (request, h) {...},
  tags: ["api"],
  description: "Delete a playlist",
  validate: { params: { id: IdSpec }, failAction: validationError },
},

deleteAll: {
  auth: {
    strategy: "jwt",
  },
  handler: async function (request, h) {...},
  tags: ["api"],
  description: "Delete all PlaylistApi",
},
};
```

JWT/HAPI



Full Stack Web Development