



# Asia Developer Academy

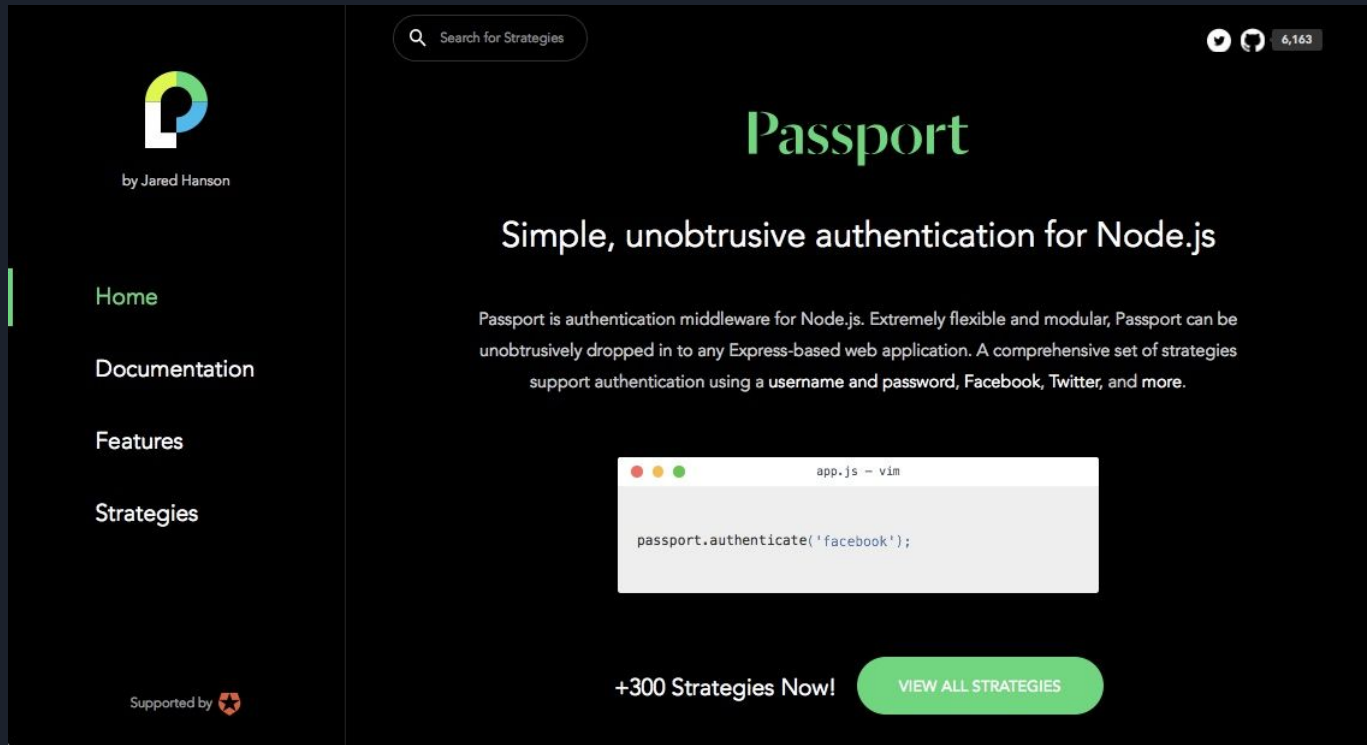
Authentication (2) with JWT



# API Revision

- 1) Create new API project.
- 2) Import the necessary Modules
- 3) Create a simple “Hello World” route.
- 4) Test the API to be working.
- 5) Connect to database.

# Passport (Revision)



The screenshot shows the Passport.js website with a dark theme. On the left is a navigation sidebar with the Passport logo and links for Home, Documentation, Features, and Strategies. The main content area features the title 'Passport' in green, a subtitle 'Simple, unobtrusive authentication for Node.js', and a paragraph describing its flexibility and modularity. A code editor window displays the code `passport.authenticate('facebook');`. At the bottom, there is a call to action '+300 Strategies Now!' and a green button labeled 'VIEW ALL STRATEGIES'. The top right corner includes a search bar, social media icons, and a user count of 6,163.


by Jared Hanson

Home

Documentation

Features

Strategies

Supported by 

Search for Strategies

6,163

## Passport

Simple, unobtrusive authentication for Node.js

Passport is authentication middleware for Node.js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies support authentication using a username and password, Facebook, Twitter, and more.

```
app.js - vim

passport.authenticate('facebook');
```

+300 Strategies Now! [VIEW ALL STRATEGIES](#)



# About JWT

JWT (*JSON Web Tokens*) is a very simple and secure authentication's strategy for REST APIs.

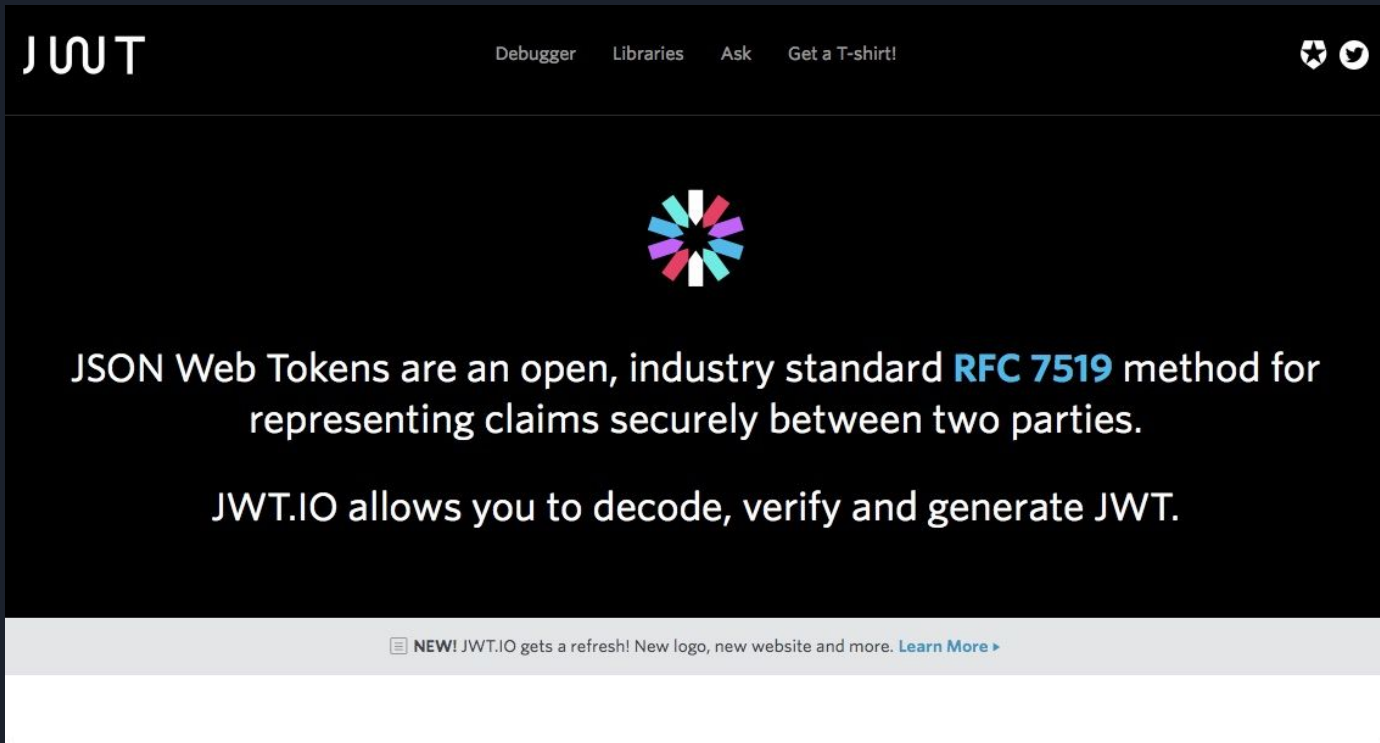
It is an *open standard* for web authentications and is totally based in JSON token's requests between client and server.



# Authentication flow

1. Client makes a request once by sending their login credentials and password;
2. Server validates the credentials and, if everything is right, it returns to the client a JSON with a token that encodes data from a user logged into the system;
3. Client, after receiving this token, can store it the way it wants, whether via LocalStorage, Cookie or other client-side storage mechanisms;
4. Every time the client accesses a route that requires authentication, it will only send this token to the API to authenticate and release consumption data;
5. Server always validate this token to allow or block a customer request.


# About JWT : Refer JWT.io



The image shows a screenshot of the JWT.io website. The top navigation bar is dark with the 'JWT' logo on the left and links for 'Debugger', 'Libraries', 'Ask', and 'Get a T-shirt!' on the right. Social media icons for GitHub and Twitter are also present. The main content area features a colorful circular logo with a starburst pattern. Below the logo, the text reads: 'JSON Web Tokens are an open, industry standard **RFC 7519** method for representing claims securely between two parties.' and 'JWT.IO allows you to decode, verify and generate JWT.' At the bottom, a light gray banner contains a news item: 'NEW! JWT.IO gets a refresh! New logo, new website and more. [Learn More >](#)'

**JWT**

Debugger Libraries Ask Get a T-shirt!



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.

NEW! JWT.IO gets a refresh! New logo, new website and more. [Learn More >](#)



# Passport Revision

- 1) Create new User Object Model. In addition, add User group , eg: Admin, User
- 2) Create API for register and authenticate.
- 3) Add methods for password encryption and comparePassword in User model.
- 4) Test the API



# Passport config file.

1) Create a config file, config.js with following information:

```
module.exports = {  
  'secret': 'thisissecretkey',  
  'database': 'mongodb://wanmuz:abcd1234@ds113136.mlab.com:13136/friday_tuto  
};
```





# Auth.js

```
var JwtStrategy = require('passport-jwt').Strategy;

var ExtractJwt = require('passport-jwt').ExtractJwt;

var User = require('./user.js');

var config = require('./config.js');

var passport = require('passport');

var params = {

  secretOrKey: config.secret,

  jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken()

};
```



# Auth.js (2)

```
module.exports = function() {
  var strategy = new JwtStrategy(params, function(payload, done) {
    User.findOne({id: payload.id}, function(err, user) {
      if (err) {
        return done(err, false);
      }
      if (user) {
        done(null, user);
      } else {
        done(null, false);
      }
    });
  });
  passport.use(strategy);
  return {
    initialize: function() {
      return passport.initialize();
    },
    authenticate: function() {
      return passport.authenticate("jwt", {
        session: false
      });
    }
  };
};
```

## Change login/authenticate to follows:

```
1 // Check if password matches
2 user.comparePassword(req.body.password, function(err, isMatch) {
3   if (isMatch && !err) {
4     // Create token if the password matched and no error was thrown
5     var token = jwt.sign(user.toJSON(), config.secret, {
6       expiresIn: 10080 // in seconds
7     });
8     res.json({ success: true, token: 'JWT ' + token });
9   } else {
0     res.send({ success: false, message: 'Authentication failed. Pass
1   }
2 });
3 }
4 });
5 }
```

Add to route:

```
9  });  
0  
1  
2  app.use(auth.initialize());  
3  
4  
5  // Bring in defined Passport Strategy
```

```
router.get('/dashboard', auth.authenticate(), function(req, res) {  
  res.send('It worked! User id is: ' + req.user._id + '.');  
});  
  
app.use('/api'.router):
```

# Test the API : Authenticate

The screenshot displays a REST client interface with the following details:

- Request:** Method: POST, URL: localhost:3000..., Params: empty.
- Response:** Status: 200 OK, Time: 599 ms, Size: 586 B.
- Response Body:** Pretty JSON view showing a successful authentication response:

```
1 {  
2   "success": true,  
3   "token": "JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9  
   .eyJfaWQiOiI1YTJiNGE2OTljMjIxYzJiY2VjMGU2N2QlLCJlbWFpbCI6Indhbm11  
   eJg2QGdtYWlsLmNvbSIsInBhc3N3b3JkIjoiejojJDEwJFV6OUVwZS9ZZDFQum9QT  
   0swZ2FFbHVidlNsN01nNnowOTJHMM1tRk1IVVNrYzBibmUxNWxDIiwiaXN192IjowLC  
   Jyb2x1IjojQ2xpZW50IiwiaWF0IjoxNTEyODAyMjc5LjE1MTI4MTIzNTF  
   9.eainIPMXiQ-59MDu420FSkxpxcDi9hNAAmzKk6kqWSM"  
4 }
```
- Auth:** No Auth selected.

# Test the API : Authenticate

The screenshot shows a REST client interface with the following details:

- URLs:** localhost:3000/api/d, localhost:3000/api/a, localhost:3000/api/d, localhost:3000/api/a, localhost:3000/api/d, localhost:3000
- Environment:** No Environment
- Method:** GET
- URL:** localhost:3000...
- Params:** Params
- Status:** 200 OK
- Time:** 242 ms
- Size:** 253 B
- Response Body:** It worked! User id is: 5a2b4a699c221c2bcec0e67d.

The Headers tab is active, showing the following headers:

Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Content..	applicati...			
<input checked="" type="checkbox"/>	Authoriz...	Bearer e...			
	New key	Value	Description		



# Revision : Call the API

Use app and Web to call the API. It should have:

- 1) Login page
- 2) Second page where API call will have list of users.
- 3) List of users can only be shown with Authenticated user.