



# NODE.JS GLOBAL

**Authentication. Validation. Security**

FEBRUARY, 2019

# AGENDA

---

1. Validation with AJV
2. Authentication with Passport.js
3. JSON Web Tokens (JWT)
4. Security

# VALIDATION

1. By schema
2. By chaining calls (fluent interface)
3. String validation

```
const Joi = require('joi');

const schema = Joi.object().keys({
  username: Joi.string().alphanum().min(3).max(30).required(),
  password: Joi.string().regex(/^[a-zA-Z0-9]{3,30}$/),
  access_token: [Joi.string(), Joi.number()],
});

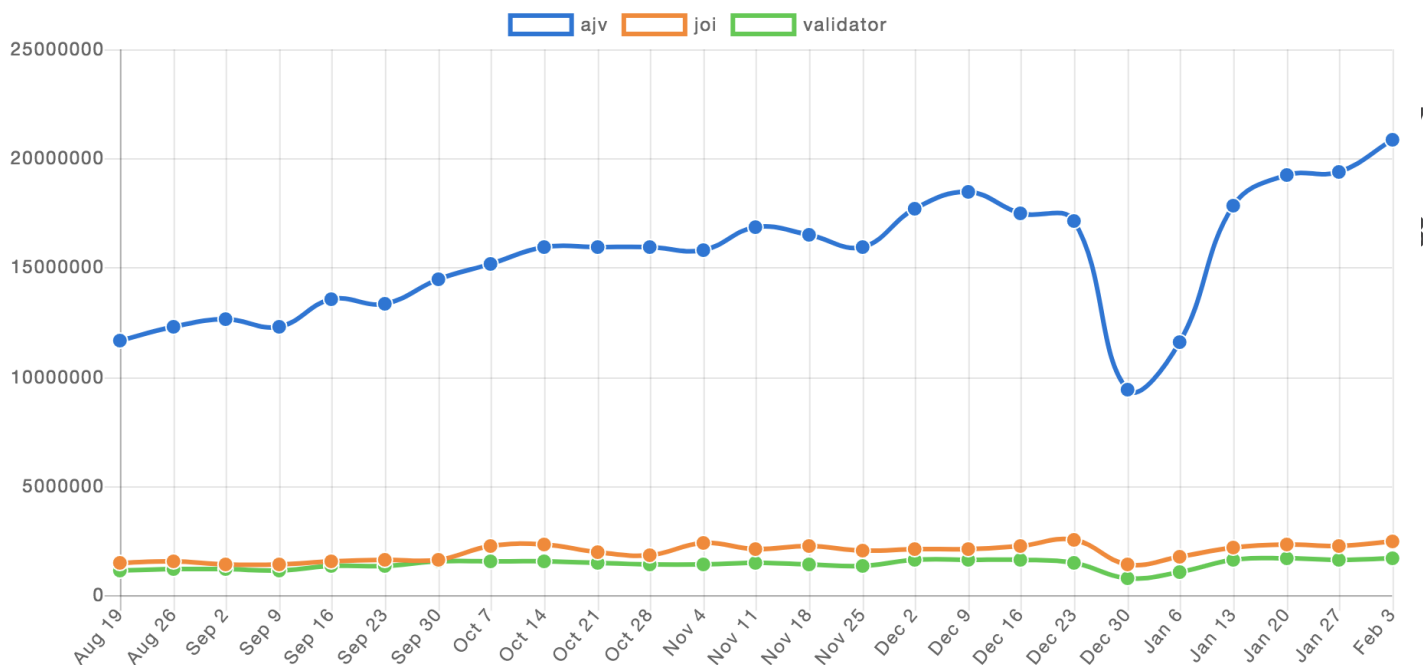
// Return result.
const result = Joi.validate({ username: 'abc', birthyear: 1994 }, schema);
// result.error === null => valid

const validator = require('validator');

validator.isEmail('foo@bar.com'); // => true
```

# VALIDATION LIBRARIES

Downloads in past 6 Months ▾



stars 🌟

forks 🍴

issues ⚠️

updated 🔧

created 🐣

ajv

4985

381

46

Feb 9, 2019

May 20, 2015

validator.js

12508

1068

72

Feb 3, 2019

Oct 6, 2010

joi

10737

865

50

Jan 30, 2019

Sep 16, 2012

\* <https://www.npmtrends.com/ajv-vs-joi-vs-validator>

# AJV

```
const Ajv = require('ajv');
const ajv = Ajv({ allErrors:true, removeAdditional:'all' });
const employeeSchema = require('./employee.schema');
ajv.addSchema(employeeSchema, 'new-employee');

// error mapping
function errorResponse(schemaErrors) {
  let errors = schemaErrors.map((error) => {
    return {
      path: error.dataPath,
      message: error.message
    };
  });
  return {
    status: 'failed',
    errors: errors
  };
}

// create validation middleware
function validateSchema (schemaName) {
  return (req, res, next) => {
    let isValid = ajv.validate(schemaName, req.body);
    if(!isValid) {
      res.status(400).json(errorResponse(ajv.errors));
    } else {
      next();
    }
  }
}
```



# JSON SCHEMA

```
{
  "title": "new employee",
  "description": "Properties required to create an employee",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string",
      "description": "firstname of the account user"
    },
    "lastName": {
      "type": "string",
      "description": "lastname of the account user"
    },
    "title": {
      "type": "string",
      "description": "username of account"
    },
    "isActive": {
      "type": "boolean",
      "description": "whether the users email address should be remembered"
    }
  },
  "required": ["firstName", "lastName", "title", "isActive"]
}
```

# APPLY SCHEMA

```
app.post('/employees', validateSchema('new-employee'), function (req, res) {  
  let employee = req.body;  
  
  employee.id = uuid.v4();  
  
  data.push(employee);  
  
  res.status(204).location(`/employees/${employee.id}`).send();  
});
```

# AJV RESPONSE

POST ▾

http://localhost:3000/employees

Params

Send

Authorization

Headers (1)

Body ●

Pre-request Script

Tests

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

JSON (application/json) ▾

1 ▾ {

2     "isActive": "true",

3     "title": "Senior Software Engineer",

4     "lastName": "Lomako",

5     "firstName": "Vladislav"

6   }

Body

Cookies

Headers (6)

Test Results

Status: 400 Bad Request T

Pretty

Raw

Preview

JSON ▾

1 ▾ {

2     "status": "failed",

3     "errors": [

4       {

5         "path": ".isActive",

6         "message": "should be boolean"

7       }

8     ]

9   }



# AUTHENTICATION

---

1. Passport.js
2. JSON Web Tokens

# WHAT IS PASSPORT

---

1. Authentication middleware for Node.js
2. Extremely flexible and modular
3. Can be unobtrusively used with any Express application
4. 500+ strategies to choose from



# GET IT WORK

---

1. Require passport module and initialize it
2. Configure passport with at least one Strategy
3. Specifying a route for authentication
4. Protect routes

# SETUP LOCAL STRATEGY

```
// setup local strategy
passport.use(new LocalStrategy({
  usernameField: 'firstName',
  passwordField: 'lastName',
  session: false
}, function (username, password, done) {
  let employee = _.find(data, { firstName: username });

  if (employee === undefined || employee.lastName !== password) {
    done(null, false, 'Bad username/password combination');
  } else {
    done(null, employee);
  }
}));

// setup bearer strategy
passport.use(new BearerStrategy(
  function (token, done) {
    let result = _.find(tokens, { token: token });

    if (result === undefined) {
      done(null, false);
    } else {
      done(null, result, { scope: 'all' }) → →
    }
  }
));

// a piece of JSON with data
{
  "isActive": true,
  "title": "Senior Software Engineer",
  "lastName": "Williamson",
  "firstName": "Marsh",
  "id": "cdb24955-4de2-4ae2-9419-2dd509027de6"
},

// a piece of JSON with tokens
{
  "id": "cdb24955-4de2-4ae2-9419-2dd509027de6",
  "token": "TOKEN-6e87-4baf-b942-f984d9cd0635"
},
```

# CONNECT WITH EXPRESS

```
// setup authentication route with local strategy
app.post('/authenticate', passport.authenticate('local', { session: false }), function (req, res) {
  let token = _.find(tokens, { id: req.user.id });

  res.json(token);
});

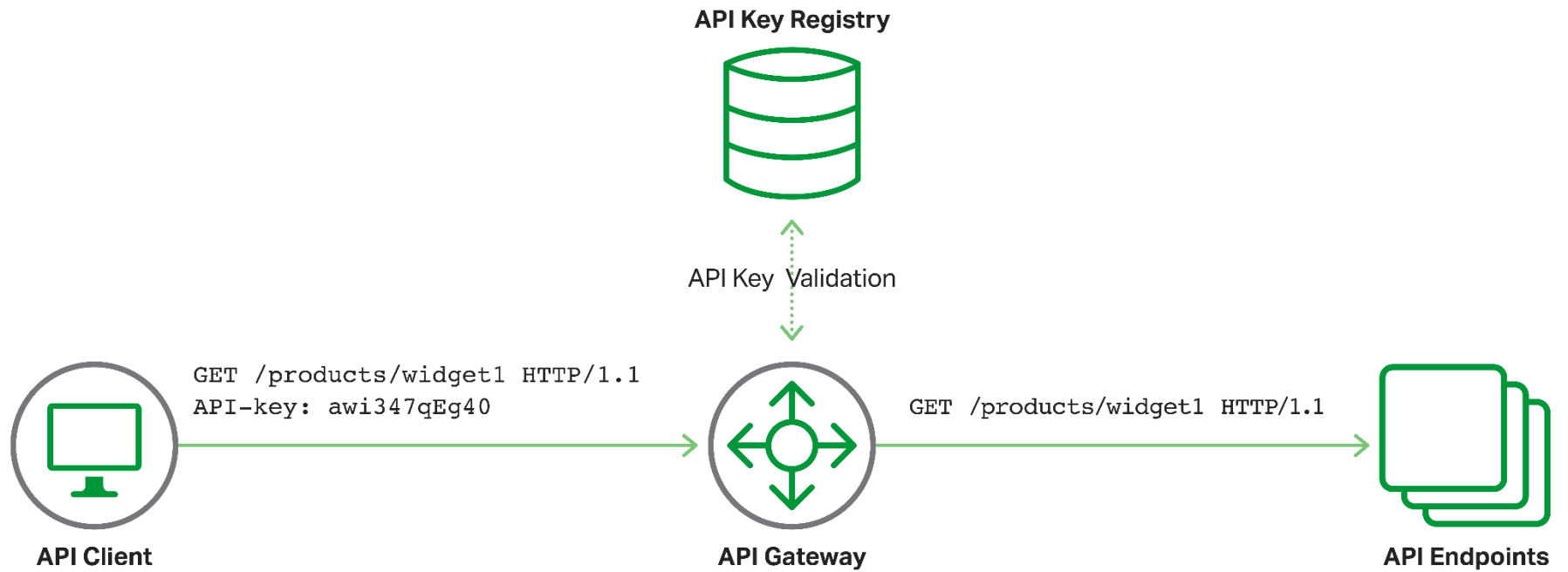
// protect endpoint with bearer strategy
app.get('/employees', passport.authenticate('bearer', { session: false }), function (req, res) {
  res.json(data);
});
```

# REQUESTS

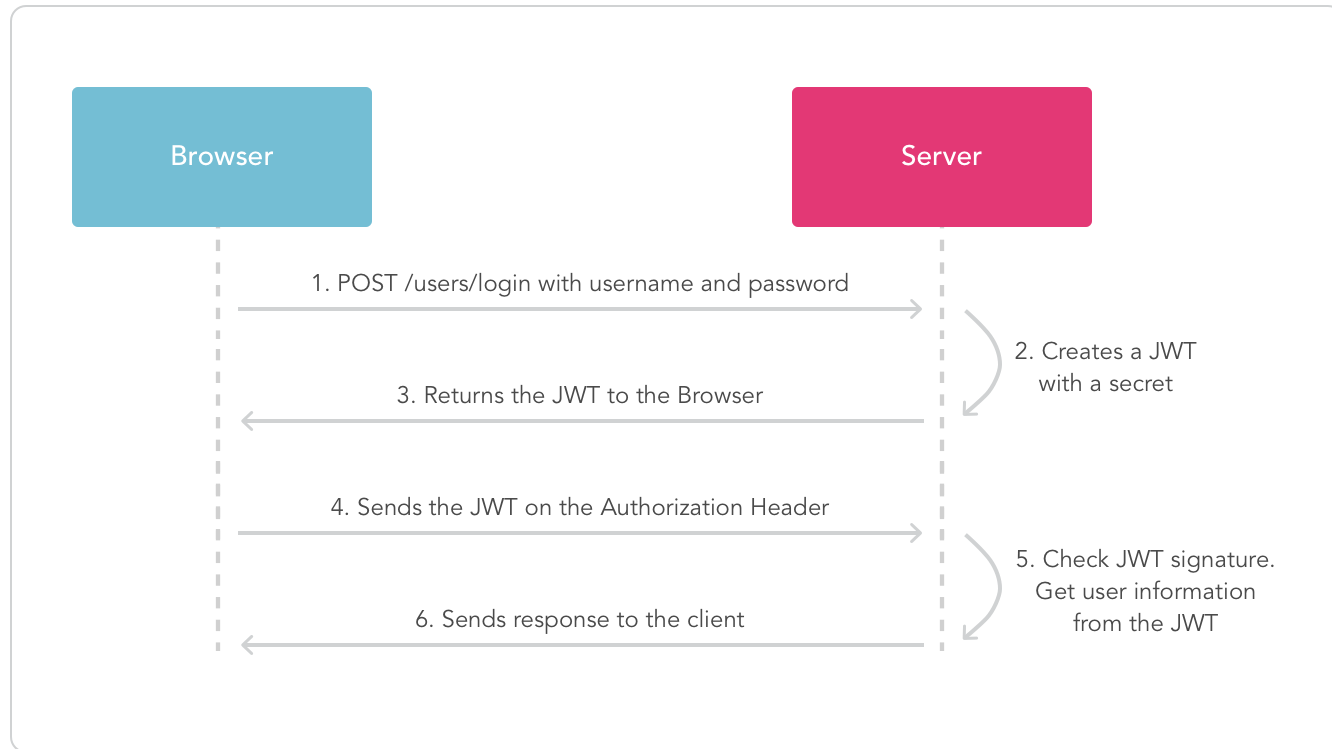
---

1. POST request to the authentication route with data in body  
As a result get JSON with ID and TOKEN in response
2. GET request to the protected authentication with bearer token

# REGULAR TOKEN AUTH



# HOW DOES IT WORK





## COMPACT AND SELF-CONTAINED

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjEwNjYyOTQxNDY5LjJVA950rM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

Decoded

EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

## HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```

HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    secret
)

```

\* <https://jwt.io/>

# SETUP JWT

```
app.post('/authenticate ', function (req, res) {
  let employee = _.find(data, { firstName: req.body.firstName });

  if (employee === undefined || employee.lastName !== req.body.lastName) {
    res.status(403).send({ success: false, message: 'Bad username/password combination.' });
  } else {
    let payload = { "sub": employee.id, "isActive": employee.isActive };
    let token = jwt.sign(payload, 'secret', { expiresIn: 10 });
    res.send(token);
  }
});

// middleware for token check
function checkToken(req, res, next) {
  let token = req.headers['x-access-token'];

  if (token) {
    jwt.verify(token, 'secret', function(err, decoded) {
      if (err) {
        res.json({ success: false, message: 'Failed to authenticate token.' });
      } else {
        // some business logic here
        next();
      }
    });
  } else {
    res.status(403).send({ success: false, message: 'No token provided.' });
  }
}
```

# PROTECT ENDPOINT WITH JWT

```
// middleware for token check
function checkToken(req, res, next) {
  let token = req.headers['x-access-token'];

  if (token) {
    jwt.verify(token, 'secret', function(err, decoded) {
      if (err) { ...
      } else {
        // some business logic here
        next();
      }
    });
  } else { ...
  }
}

// protect endpoint with JWT token
app.get('/employees', checkToken, function (req, res) {
  res.json(data);
});
```

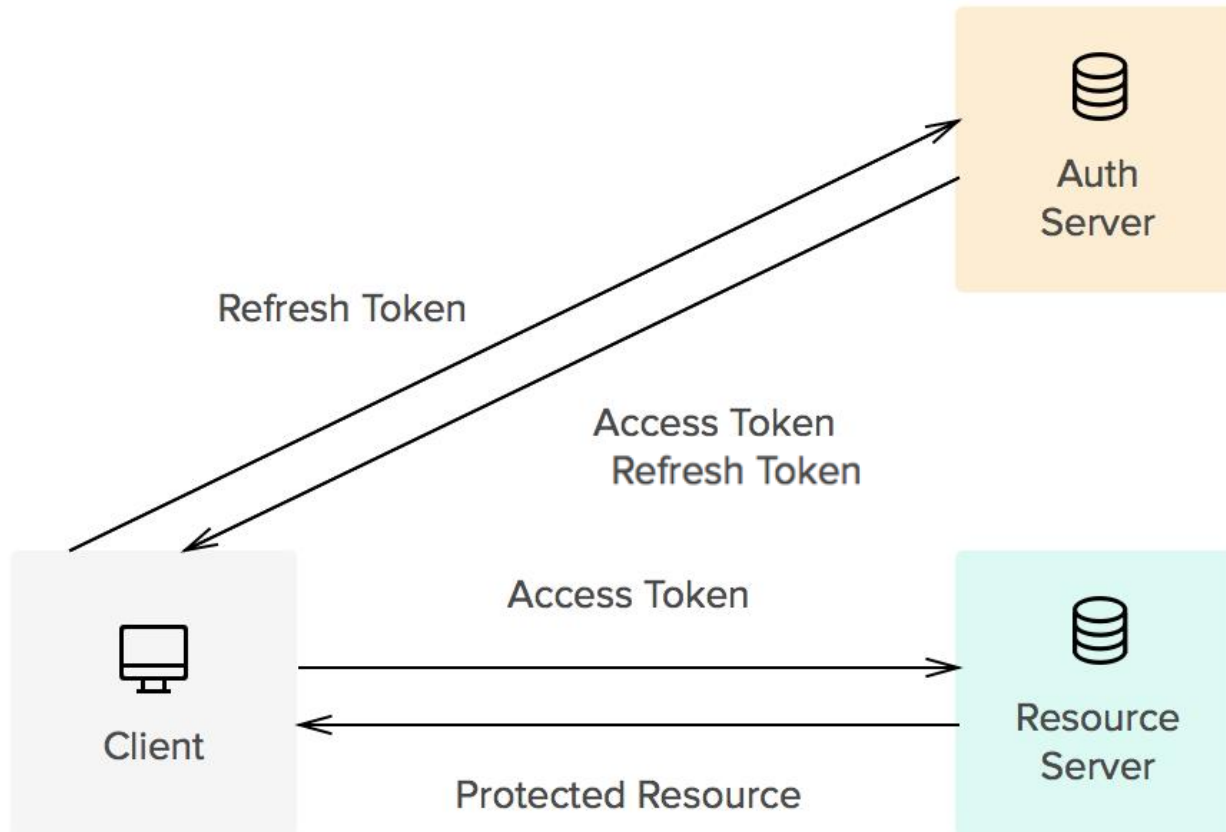
GET ▾	http://localhost:4000/employees	Params	Send ▾		
Authorization	Headers (1)	Body	Pre-request Script	Tests	
	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	<u>x-access-token</u>	token.shouldbe.here			

# JWT PROS AND CONS

---

- No server-side storage
- Easy to use (a lot of libs)
- Can be used across services
- One key rules all
- Can't be revoked permanently (in general)

# ACCESS TOKEN AND REFRESH TOKEN



# OWASP TOP 10

## OWASP Top 10 2013

A1 Injection

A2 Broken Auth and Session Management

A3 Cross-Site Scripting (XSS)

A4 Insecure Direct Object References

A5 Security Misconfiguration

A6 Sensitive Data Exposure

A7 Missing Function Level Access Control

A8 Cross-Site Request Forgery (CSRF)

A9 Using Components with Known Vulnerabilities

A10 Unvalidated Redirects and Forwards

## OWASP Top 10 2017

A1 Injection

A2 Broken Authentication

A3 Sensitive Data Exposure

A4 XML External Entities (XXE)

A5 Broken Access Control

A6 Security Misconfiguration

A7 Cross-Site Scripting (XSS)

A8 Insecure Deserialization

A9 Using Components with Known Vulnerabilities

A10 Insufficient Logging & Monitoring

# NODE SECURITY

---

1. Static analysis
2. Sensitive data on the client side
3. Running process with superuser rights
4. Vulnerabilities in dependencies
5. Known vulnerabilities in your code
6. Error logging

# API SECURITY

---

1. HTTPS
2. Access Control
3. Restrict HTTP methods
4. Input validation
5. Error handling
6. Audit logs
7. Security headers
8. Sensitive information in HTTP requests



# HELMET

1. Hide Powered-By
2. HSTS
3. No Cache
4. .. and more

```
const express = require('express');  
const helmet = require('helmet');  
const app = express();  
  
app.use(helmet());
```



**HELMET**

# USEFUL LINKS

---

- [Design by contract](#)
- [JSON Schema](#)
- [Custom Passport strategy](#)
- [node-jsonwebtoken](#)
- [Open Web Application Security Project \(OWASP\)](#)
- [REST API Security Cheat Sheet](#)
- [Express Production Security Best Practices](#)
- [Retire.js](#), [Node Security Platform](#), [Snyk](#)
- [Helmet](#)



# **NODE.JS GLOBAL**

**MIDDLEWARE. FRAMEWORKS  
BY  
VLADISLAV LOMAKO  
MOHAMADALI GANJI**