NodeJS

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

Agenda

- Building Authentication in Node Js
  - Introducing Token Authentication
  - JSON Web Token (JWT)
  - Using Bcrypt

Building Authentication in Node.js

Introducing Token Authentication

- In any application we need to authenticate the user when making the request to the server.
- Authenticated users means, user's information which is stored in the database.
- There are two ways to achieve this authentication.
  - Cookie-based authentication
  - Token-based authentication

Cookie-based authentication:

- This works like a traditional web application.
- Sets a session cookie on a login page that will then be used by all later requests.
- This requires building forms on the Node side that are traditional HTTP-generated pages.
- It is easier to set up, but it is inflexible.

Building Authentication in Node.js cont.

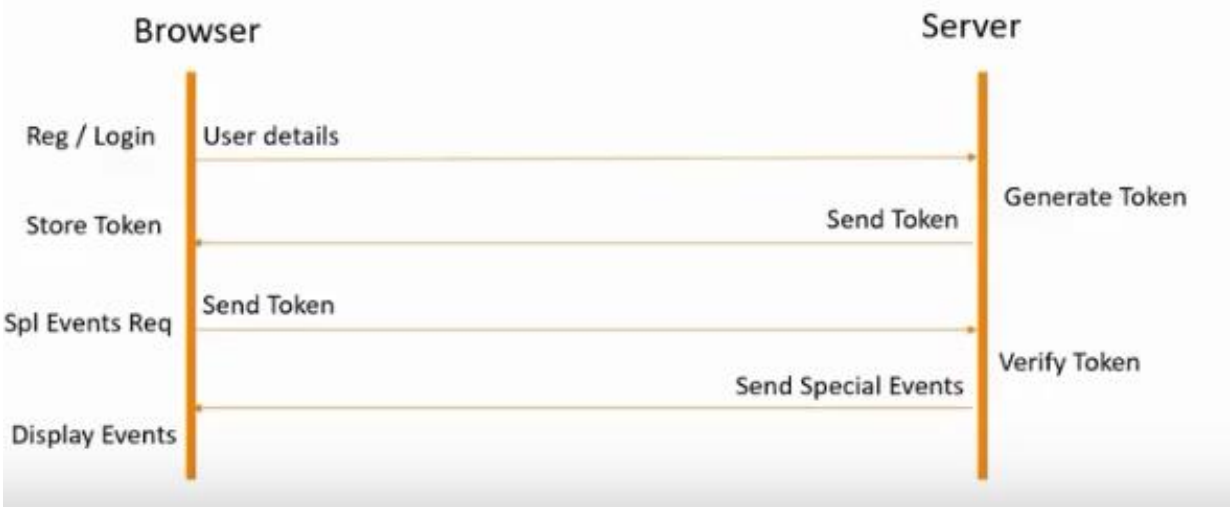
Token-based authentication:

- This works more like an API. It uses a signed token that the client must send on every request.
- The client can get the token via an API.
- It also eases debugging and troubleshooting since we just need the token to interact with the API.
- It also allows the API to be accessed via a mobile app in the same way.
- It helps with the performance since the token is validated using an algorithm rather than a database hit on every request.
- It decouples the client app from the server which makes it easier to understand the login flow in the system

Token-based authentication cont.

How token based authentication works?

- User attempt to login with their credential
- Server verifies the credentials, generate **JSON Web Token** and sent to the client.
- Client save the **JWT** in **local storage** or any other storage mechanism.
- Again if a client wants to request a protected route or resource, then it sends **JWT** in a request header.
- When the time is expired, the user has to login again to get a new token.
- Server verifies the JWT and if it is valid then returns status 200 response with the information.
- If JWT is invalid, then it gives unauthorized access or restricted message.



JSON WEB TOKEN (JWT)

- JWT is a standard format designed for authentication purpose.
- It's a JSON object which is considered to be the safest way to represent a set of information between two parties.
- The token is composed of header, payload and a signature.
- So a JWT is a string with the following format:

  header.payload.signature – xxxxx.yyyyy.zzzzz

  - Header – consists of 2 parts – type of the token (JWT) and hashing algorithm being used.
  - Payload – is the data that is stored inside the JWT.
  - Signature – is used to verify the token.
  - You can visit https://jwt.io/ for more information on the format.

JSON WEB TOKEN (JWT)

- The format of JWT Token is as following.

HEADER: ALGORITHM & TOKEN TYPE

{
“alg”: “HS256”,
“typ”: “JWT”
}

PAYLOAD:DATA

{
“email”: “krunallathiya10@gmail.com”,
“_id”: “5a7c9bd8fc3e501c94aa6035”,
“iat”: 1518120124,
“exp”: 1518127324
}

JSON WEB TOKEN (JWT) cont.

- First use npm package to download jsonwebtoken
- To generate a new token we use the sign method passing the payload and secrete key and any options if required.

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

- The token is send to the frontend as response and the same token is send back to the server with every subsequent request.
- To verify the token send back from the frontend, we use the verify method passing in the token itself along with the same secrete key used for sign and any options if required.

  jwt.verify(token, secreteOrPrivateKey, [options, callback])

Using Bcrypt

- Bcrypt is a library in NPM which hash and compare password in Node
- Bcrypt is a de facto way to hash and store passwords
- Supports both sync and async methods
- Install Bcrypt : **npm install –save Bcrypt**
- Include the module in the application : **const bcrypt = require('bcrypt');**

Using Bcrypt

Synchronous Method:

- Hash the password in using bcrypt:
- First argument is the “password”
- Second argument is the number

```
var hash = bcrypt.hashSync("my password", 10);
```

To verify the password later:

- When the password is correct the compareSync will returns true while in the second case the password is incorrect, the compareSync will returns false.

```
bcrypt.compareSync("my password", hash); //
bcrypt.compareSync("not my password", hash);
```

Using Bcrypt

Asynchronous Method:

- Hash the password and store in the DB:

```
bcrypt.hash('myPassword', 10, function(err, hash) {
  // Store hash in database
});
```

- Verify the hashed password later:

```
bcrypt.compare('somePassword', hash, function(err, res) {
  if(res) {
   // Passwords match
  } else {
   // Passwords don't match
  }
});
```