**Node-JWT token implementation**

Token verification was removed from the test suites as the JWT. This JWT token is related to the user sessions when a user logs in to the application. It will be handled on the node backend to allow user to navigate and use the application.

More information on the JWT token and authorization: <https://jwt.io/introduction>

**web-application/node/controllers/utility.js**

This file has the functions to handle the validation, encryption and decryption of JWT token required for API calls being made to the node backend. These api calls are not to the EHR systems rather they are when users are accessing the scriptchain application.

EncryptToken: {payload, expiresIn = 300) => {

const token = jwt.sign(payload, “santosh”, {expiresIn: expiresIn});

const encrypted = CryptoJS.AES.encrypt(token, “secret key 123”).toString();

return encrypted;

}

DecryptToken: (encrypted) => {

const bytes = CryptoJS.AES.decrypt(encrypted, ‘secret key 123’);

const decrypted = bytes.toString(CryptoJS.enc.Utf8);

Payload = {};

jwt.verify(decrypted, “santosh”, (err, decodedValue) => {

if(err) {

payload = {‘error’: true, ‘error\_message’: err.message};

Return;

}

payload = decodedValue;

});

return payload;

},

**web-application/node/controllers/verifiedController.js**

Takes the token and decrypts it by sending the token to the security\_utils.js file function

const decryptedToken = Utility.DecryptToken(req.body.jwtToken);

if(decryptedToken[‘error’]) {

return res.status(401).json(message: decryptedToken[‘error\_message’]});

}

**web-application/node/controllers/healthProviderController.js**

This controller comes in play when we are trying to create new healthcare provider user in the DB. In that situation, we are creating a JWT token for the session of the user which contains following details:

* First Name
* Last Name
* Company Name
* Role in Company
* Email
* Password
* Phone
* Photo
* EHR

This will be signed using “Santosh” and a token will be created. It will be encrypted using the following function:

EncryptToken: {payload, expiresIn = 300) => {

const token = jwt.sign(payload, “santosh”, {expiresIn: expiresIn});

const encrypted = CryptoJS.AES.encrypt(token, “secret key 123”).toString();

return encrypted;

}

Just like the healthcarecontroller, all the other controllers wherever the API call is being made we make use of these functions to encrypt and decrypt the token and verify the user before working on the request.

**CryptoJS**

<https://cryptojs.gitbook.io/docs/>

This is a JS implementation of standard and secure cryptographic algorithms. It supports following hashing algos such as MD5, SHA-1, SHA-2 etc. It also supports cipher algorithms AES, DES, Rabbit, etc.

Explanation on the encryption function for token.

2 variables are passed to the function:

1. Payload: which contains the request or user session information.
2. ExpiresIn: This variable contains the value in seconds for how long the token will be valid.

EncryptToken: {payload, expiresIn = 300) => {

/\* jwt.sign is a function that is part of npm library which lets you create a token with a key signature \*/

const token = jwt.sign(payload, “santosh”, {expiresIn: expiresIn});

/\* Here we are going to be converting the token which contains the payload, expire value and the sign with a key “santosh”. Once the token is encrypted, we are converting it to string using a simple toString function. \*/

const encrypted = CryptoJS.AES.encrypt(token, “secret key 123”).toString();

// After we have the token encrypted we are just returning it.

return encrypted;

}

Explanation on the decrypt function

// Variable that we pass is the encrypted token

DecryptToken: (encrypted) => {

// decrypt function in the cryptoJS library helps to decrypt the token with a key that we used to encrypt the token with.

const bytes = CryptoJS.AES.decrypt(encrypted, ‘secret key 123’);

const decrypted = bytes.toString(CryptoJS.enc.Utf8);

Payload = {};

// Next we are verifying the decrypted key with “santosh” key that we used to sign the key initially with

jwt.verify(decrypted, “santosh”, (err, decodedValue) => {

if(err) {

payload = {‘error’: true, ‘error\_message’: err.message};

Return;

}

payload = decodedValue;

});

return payload;

},