Authentication and Authorization in Node.JS

Authentication

- Simply, we know that authentication is nothing but verifying the user identities for security purpose. Previously (old approach) we used server-based authentication where logged information stored in the server by creating a session for further identification. Think about those stored logged information which is going to match with the logged user for identity on each and every request to the server for serving data. This may cause performance issue while handling more authenticated response by the server.

Token-Based Authentication

Here comes token based authentication that means the server will response with a generated token on user login which will save in client instead of storing in the server to use for the further request. On each client request the token need to pass with the header which will verify in the server to serve data. The thought is much simpler, once you logged in just request with a valid token to get data on each request. Different types of NodeJS token-based authentication:

Different types of NodeJS token-based authentication

- Passport
- JSON Web Tokens (JWT)
- Bcrypt

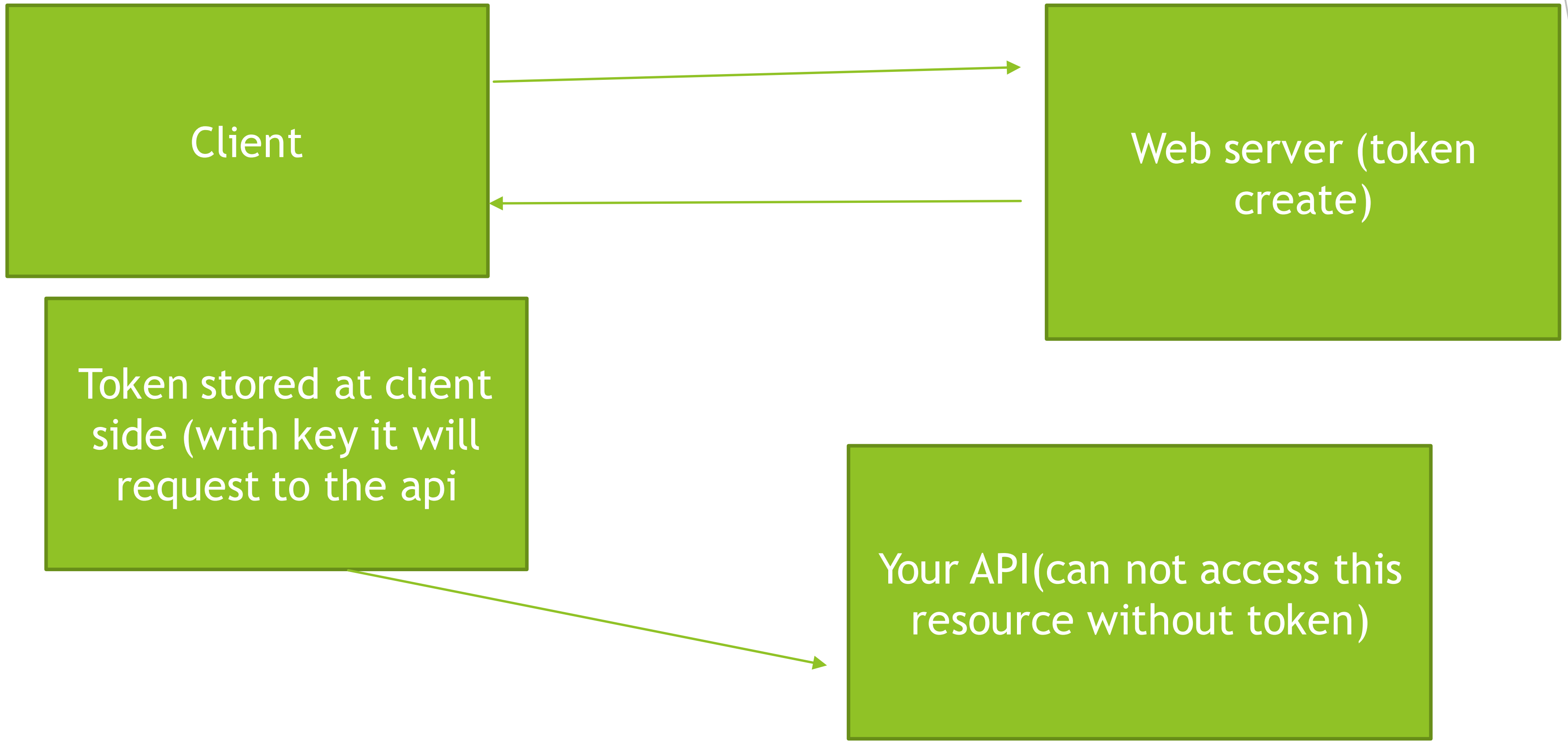
JSON Web Tokens (JWT)

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

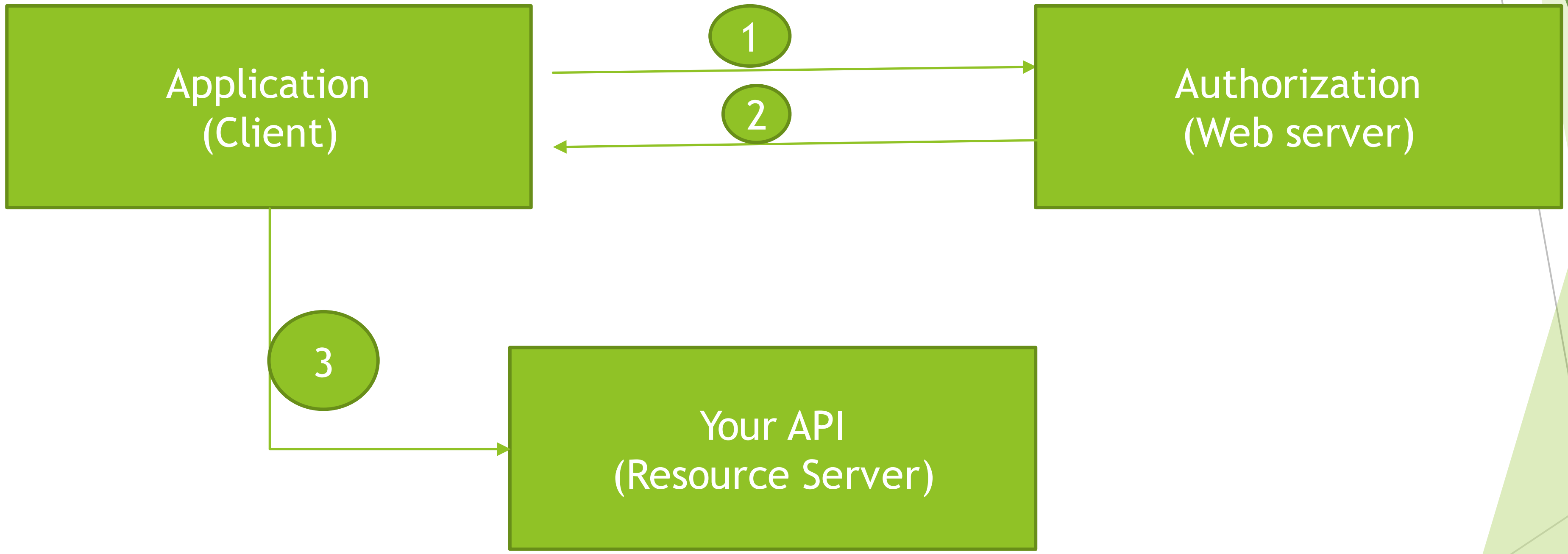
JWT string sample:

"jwtoken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwicGFzc3dvcmQiOiJhZG1pbiIsImlhdCI6MTU4OTk1NDY5MiwiZXhwIjoxNTg5OTU0OTkyfQ.w0cR2exNviVfKHl_1chRjBwt42CBJdG6bb4PryRuIg4"

Which has three part separated by “.”, where the first part with purple color is header, then the part with blue color is payload, then the signature is with green color. While a user requesting for data to server this JWT string needs to pass with the header to verify in the server for user identification. Following screen demonstrate how JWT is going to work with client request.

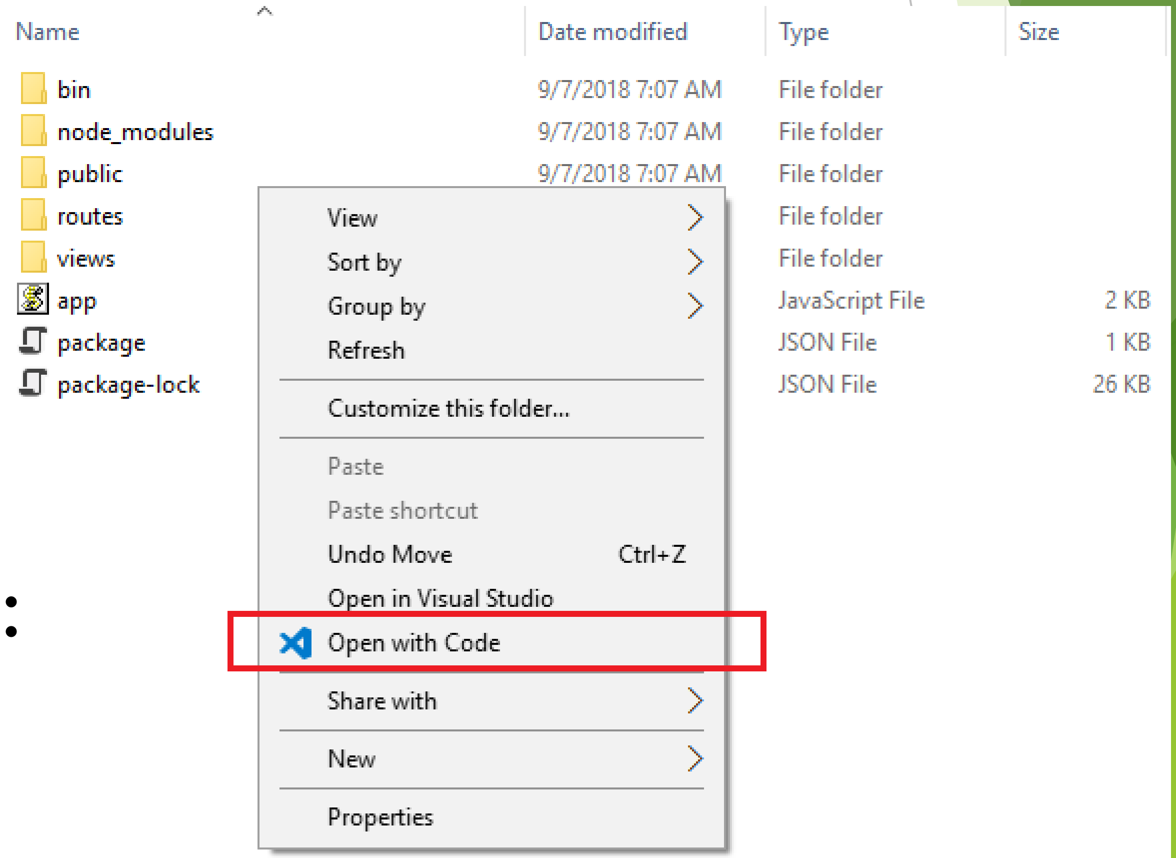

Client
Web server (token create)
Token stored at client side (with key it will request to the api
Your API(can not access this resource without token)

Screen demonstrate how JWT is going to work with client request.



Sample Application JWT Token

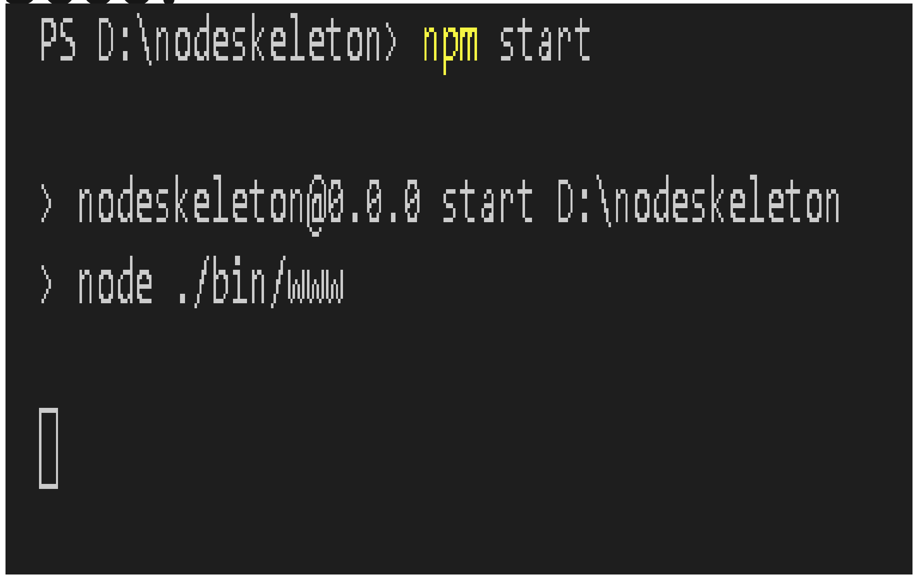
- Let's create node application with express generator.
- type command: npx express-generator. Let's open the application with Visual Studio code.



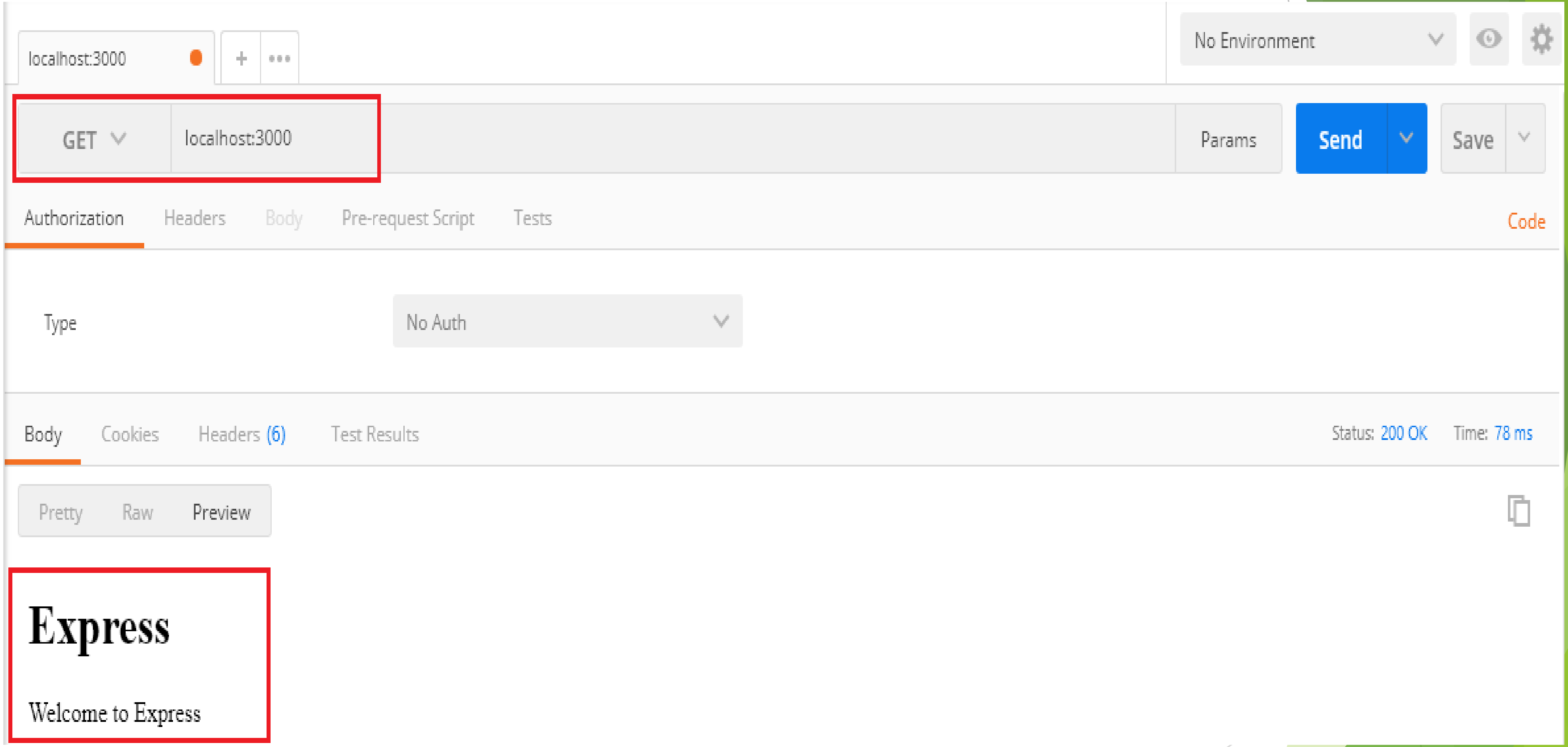
Name	Date modified	Type	Size
bin	9/7/2018 7:07 AM	File folder	
node_modules	9/7/2018 7:07 AM	File folder	
public	9/7/2018 7:07 AM	File folder	
routes		File folder	
views		File folder	
app		JavaScript File	2 KB
package		JSON File	1 KB
package-lock		JSON File	26 KB

Install Package dependency:
npm install

After installation package dependencies, time to run the application. Go to terminal tab in Visual Code then type “npm start” to start the application with predefined port 3000.



```
PS D:\nodeskeleton> npm start

> nodeskeleton@0.0.0 start D:\nodeskeleton
> node ./bin/www
```



```
GET localhost:3000
```

Type: No Auth

Status: 200 OK Time: 78 ms

Express

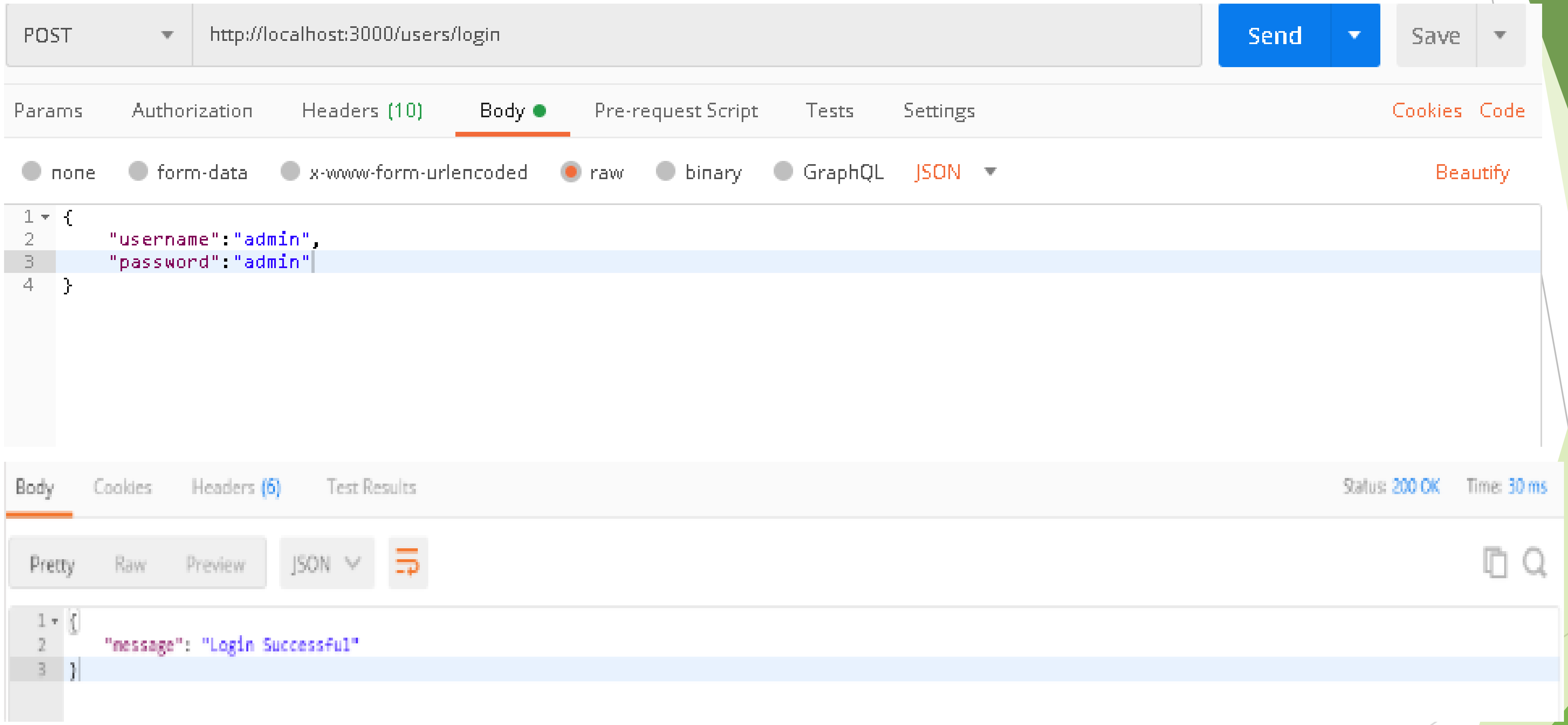
Welcome to Express

Steps:

- Let’s create a new route to user login. Go to visual code explorer to open “users.js” file then add below code snippet.

```
/* Post users Login. */
router.post('/login', function (req, res, next) {
  let userdata = {
  username: req.body.username,
  password: req.body.password
  };

  //Go to server for user varificarion
  if (userdata.username == "admin" && userdata.password == "admin") {
  res.status(200).json({
  message: 'Login Successful'
  });
  }
  else {
  res.status(401).json({
  message: 'Login Failed'
  });
  }
  });
```

Validating user with Postman. Requesting URL: https://localhost:3000/users/login



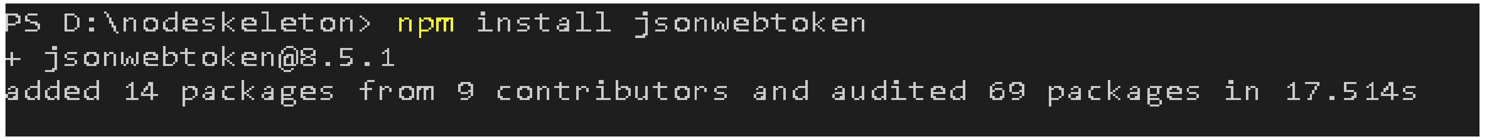
http://localhost:3000/users/login

```
{
    "username":"admin",
    "password":"admin"
}
```

Status: 200 OK Time: 30 ms

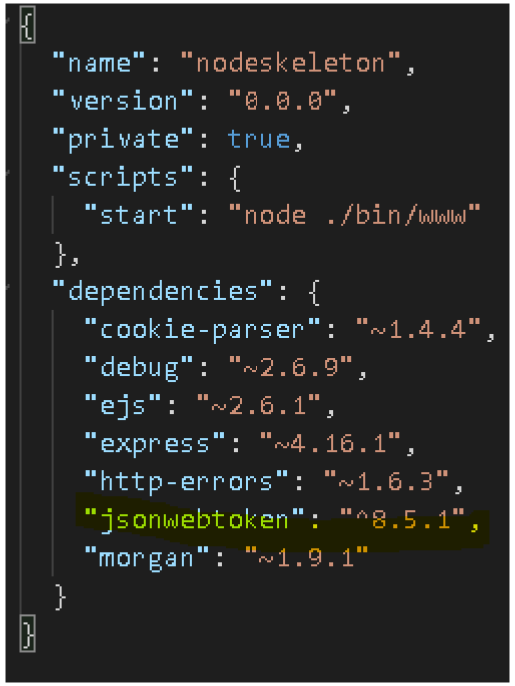
```
{
    "message": "Login Successful"
}
```

- Type “npm install jsonwebtoken” then press enter to install the package to the application.



```
PS D:\nodeskeleton> npm install jsonwebtoken
+ jsonwebtoken@8.5.1
added 14 packages from 9 contributors and audited 69 packages in 17.514s
```

- Open package.json to see the installed package.



```
{
  "name": "nodeskeleton",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "ejs": "~2.6.1",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "jsonwebtoken": "^8.5.1",
    "morgan": "~1.9.1"
  }
}
```

1. **Import jsonwebtoken :**
    1. let jwt = require('jsonwebtoken');
2. **Create Token** :jwt.sign()
    1. jwt.sign(payload : string | Buffer | object, secret: Secret, [options]: SignOptions)
    2. jwt.sign(payload : string | Buffer | object, secret: Secret, [callback: SignCallback])
    3. jwt.sign(payload : string | Buffer | object, secret: Secret, [options: SignOptions, callback: SignCallback])
3. **Create a config file**

```
module.exports = {
    secretKey: “Radixi",
    algorithm: 'HS256', //default: HS256
    };
```

4. **Register the module with the application in app.js.**

```
global.config = require('./config');
```

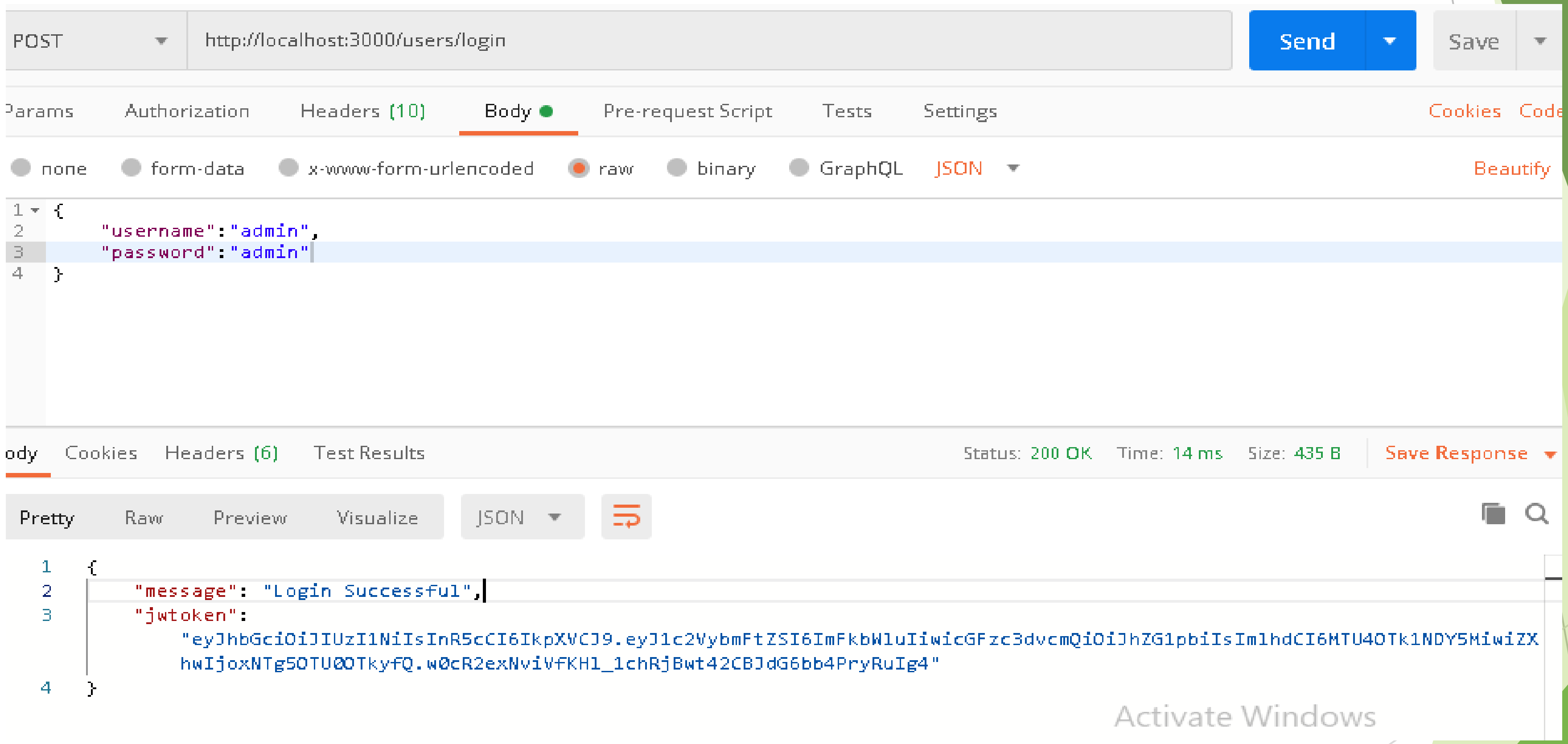
5. **Pass user data login post method to create token**

```
let token = jwt.sign(userdata, global.config.secretKey, {
  algorithm: global.config.algorithm,
  expiresIn: '5m'
  });
```

6. **In response add new property “jwtoken” with return value of token string**

```
res.status(200).json({
  message: 'Login Successful',
  jwtoken: token
  });
```

Using postman browse with URL :https://localhost:3000/users/login. From the following screen as we can see the token is generated.



```
POST  http://localhost:3000/users/login
```

```
{
    "username":"admin",
    "password":"admin"
}
```

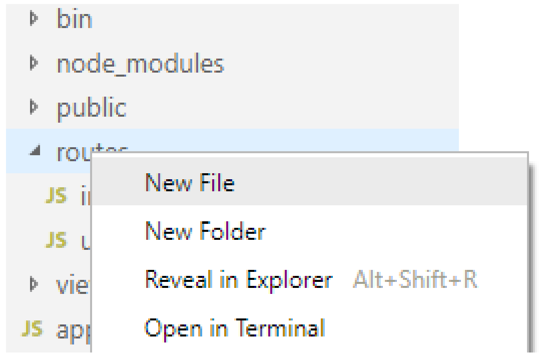
Status: 200 OK Time: 14 ms Size: 435 B

```
{
    "message": "Login Successful",
    "jwtoken":
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwicGFzc3dvcmQiOiJhZG1pbiIsImlhdCI6MTU4OTk1NDY5MiwiZX
        hwIjoxNTg5OTU0OTkyfQ.w0cR2exNviVfKHl_1chRjBwt42CBJdG6bb4PryRuIg4"
}
```

The generated token is valid for five minute you may configure it seven days by “7d” to increase the validity.

Authorization

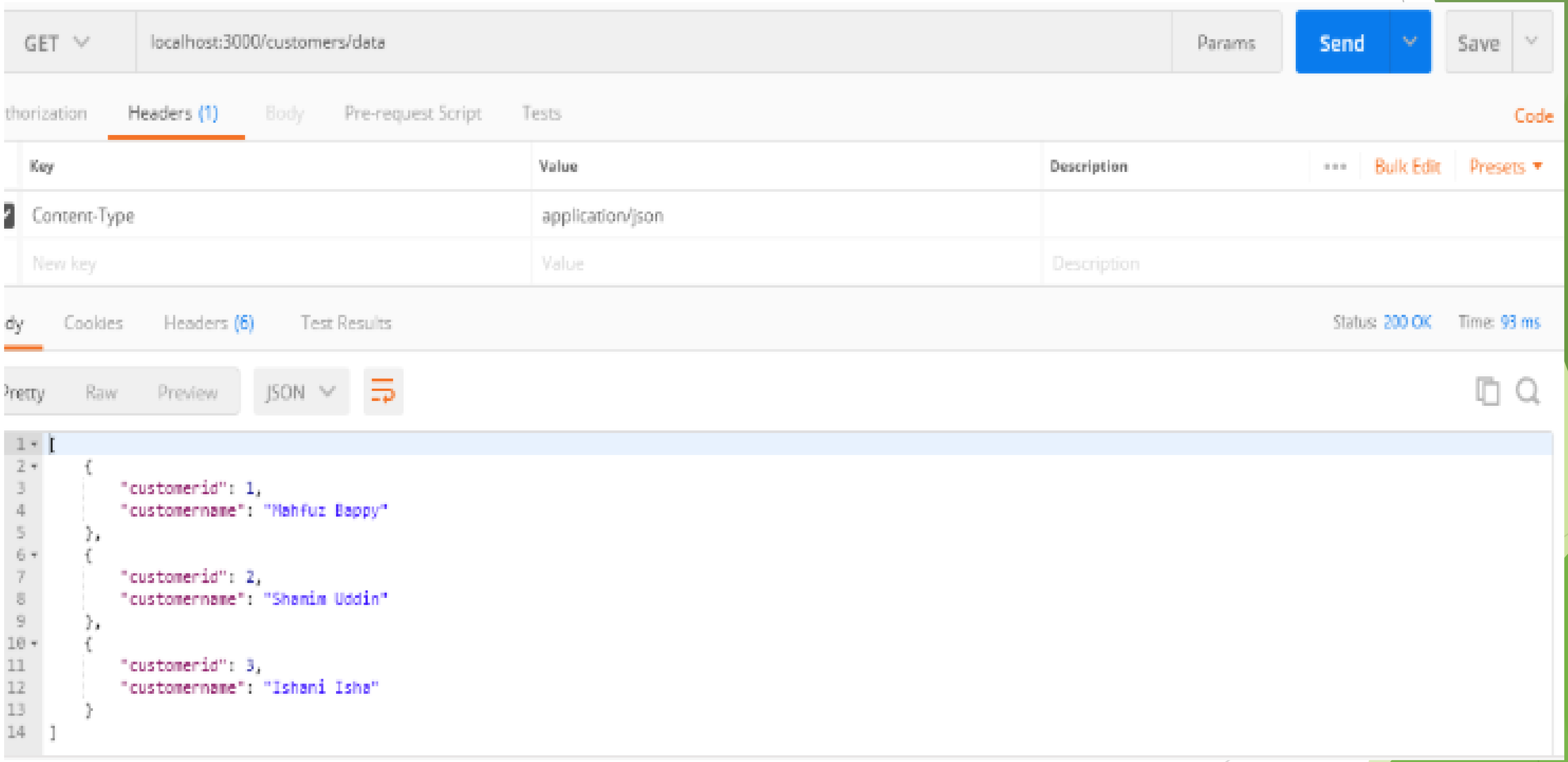
- we are going to create a customer module which is going to handle the client request by verifying the bearer token to serve data. Create a new customers.js file by the following screenshot in visual code explorer,



Customer.js file following code.

```
var express = require('express');
 var router = express.Router();
  /* GET customers listing without protection. */
 router.get('/data', function (req, res, next) {
    let customerdata = [
    {
    customerid: 1,
    customername: 'Mahfuz Bappy'
    },
    {
    customerid: 2,
    customername: 'Shamim Uddin'
    },
    {
    customerid: 3,
    customername: 'Ishani Isha'
    }
    ];

    res.json(customerdata);
    });;

 module.exports = router;
```

As you can see in postman while we are browsing by entering URL :https://localhost:3000/customers/data serve
is responding with list of data without any security validation,
which is not secured that anyone can easily access data from server.



localhost:3000/customers/data

Key	Value	Description
Content-Type	application/json	
New key	Value	Description

```
[
    {
        "customerid": 1,
        "customername": "Mahfuz Bappy"
    },
    {
        "customerid": 2,
        "customername": "Shamim Uddin"
    },
    {
        "customerid": 3,
        "customername": "Ishani Isha"
    }
]
```

Verify Token

In this portion we are going to verify the accessed token for user authorization using "jwt.verify()" method.

Method Definition

- jwt.verify(token: string, secret: Secret)
- jwt.verify(token: string, secret: Secret, [options]: VerifyOptions)
- jwt.verify(token: string, secret: Secret, [callback: VerifyCallback])
- jwt.verify(token: string, secret: Secret, [options: VerifyOptions, callback: VerifyCallback])

- Following code sample is to verify token. The first argument is the token string, then the next argument is secret key and the last one is the algorithm which used to decoding the token.

Accessing Token

Here we are going to access the token from request header by the key name “x-access-token”, which generated on user login.

```
var express = require('express');
var router = express.Router();
router.use(function (req, res, next)
 {
 var    token    =    req.headers['x-access-
token'];
console.log(token);
}
);
module.exports = router;
```

Verifying Token

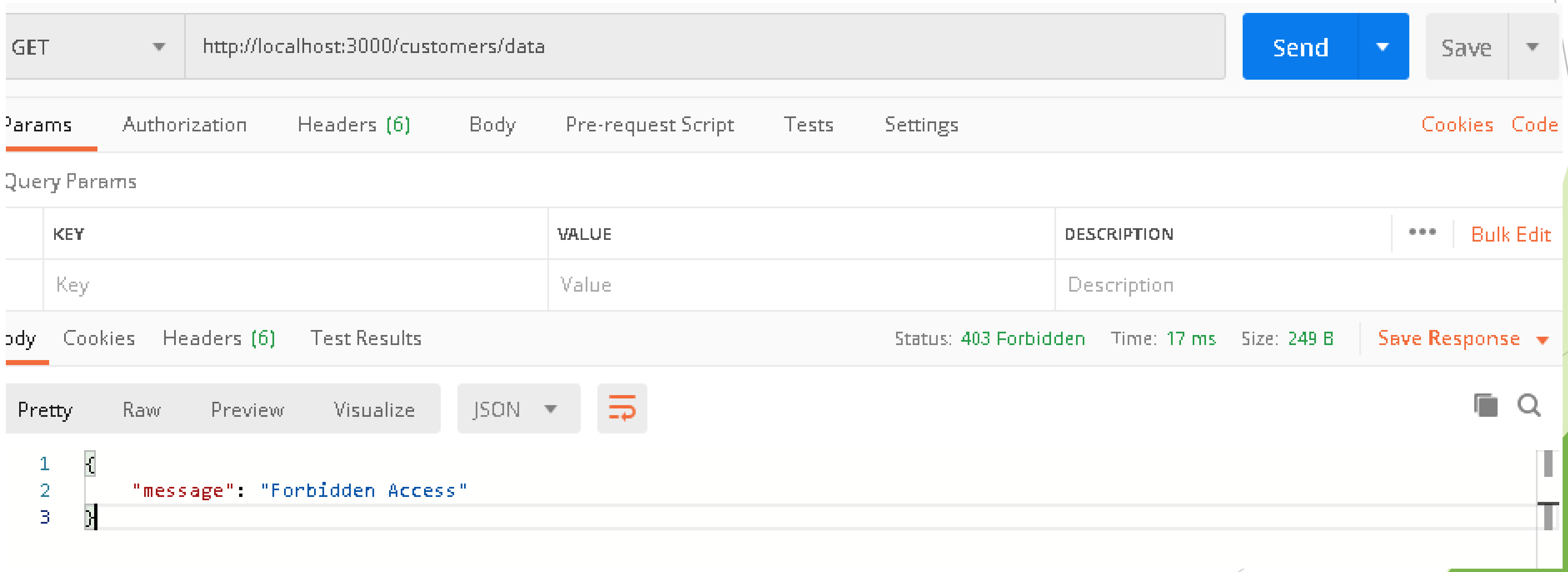
verify the accessed token for user authorization using “jwt.verify()” method

```
jwt.verify(token, global.config.secretKey,
 {
 algorithm: global.config.algorithm

 }, function (err, decoded) {
 if (err) {
 let errordata = {
 message: err.message,
 expiredAt: err.expiredAt
 };
 console.log(errordata);
 return res.status(401).json({
 message: 'Unauthorized Access'
 });
 }
 req.decoded = decoded;
 console.log(decoded);
 next();
 });
 } else {
 return res.status(403).json({
 message: 'Forbidden Access'
 });
```

Customes.js write following code:

```
let verifyToken = require('./verifytoken');

router.get('/data',verifyToken, function (req,
res, next) {
```

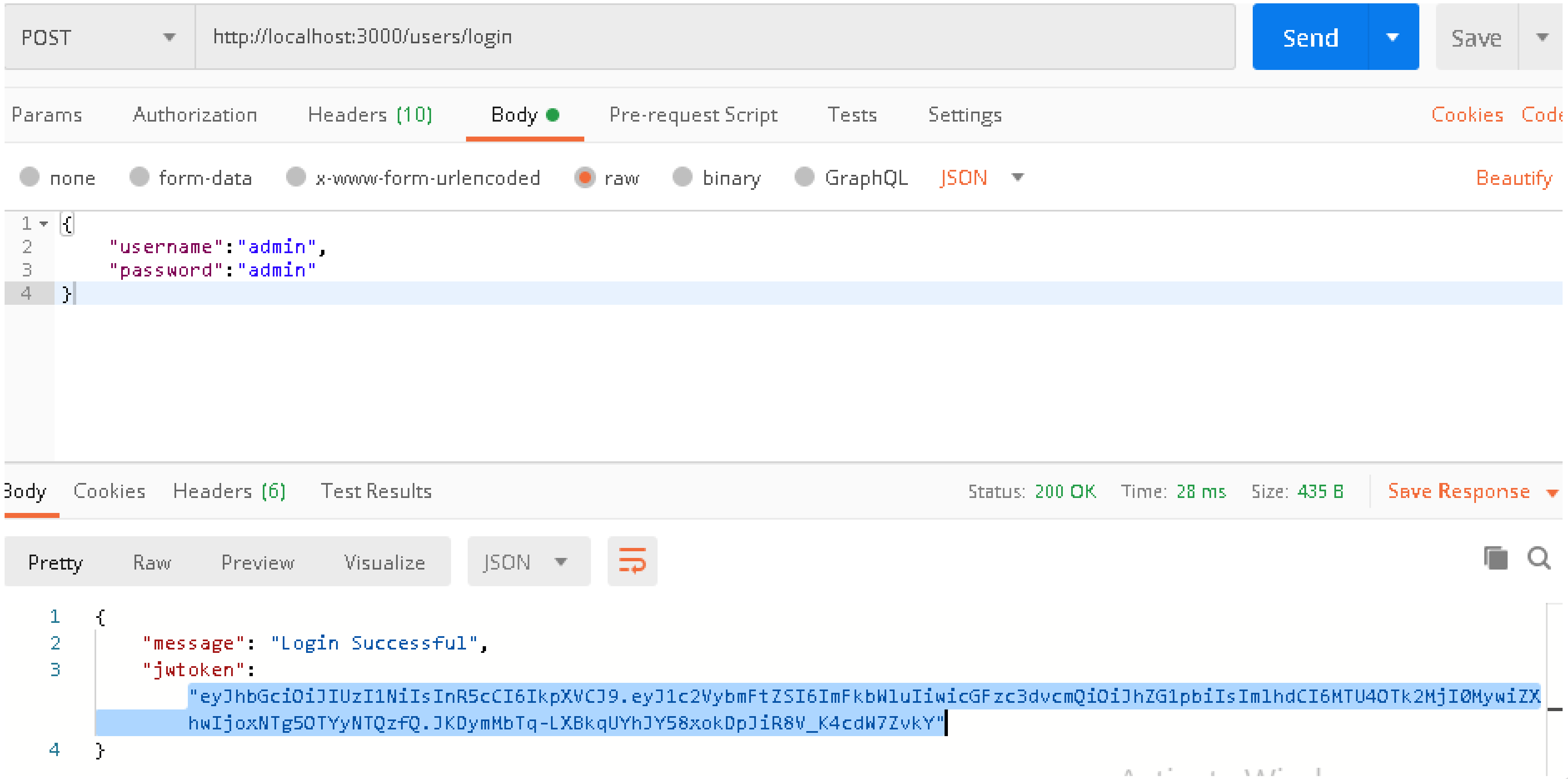
Now, if we try to access http://localhost:3000/customers/data without passing in jwt token



Status: 403 Forbidden Time: 17 ms Size: 249 B

```
{
    "message": "Forbidden Access"
}
```

Go to postman browse with URL : https://localhost:3000/users/login to regenerate the token by using valid login details.



http://localhost:3000/users/login

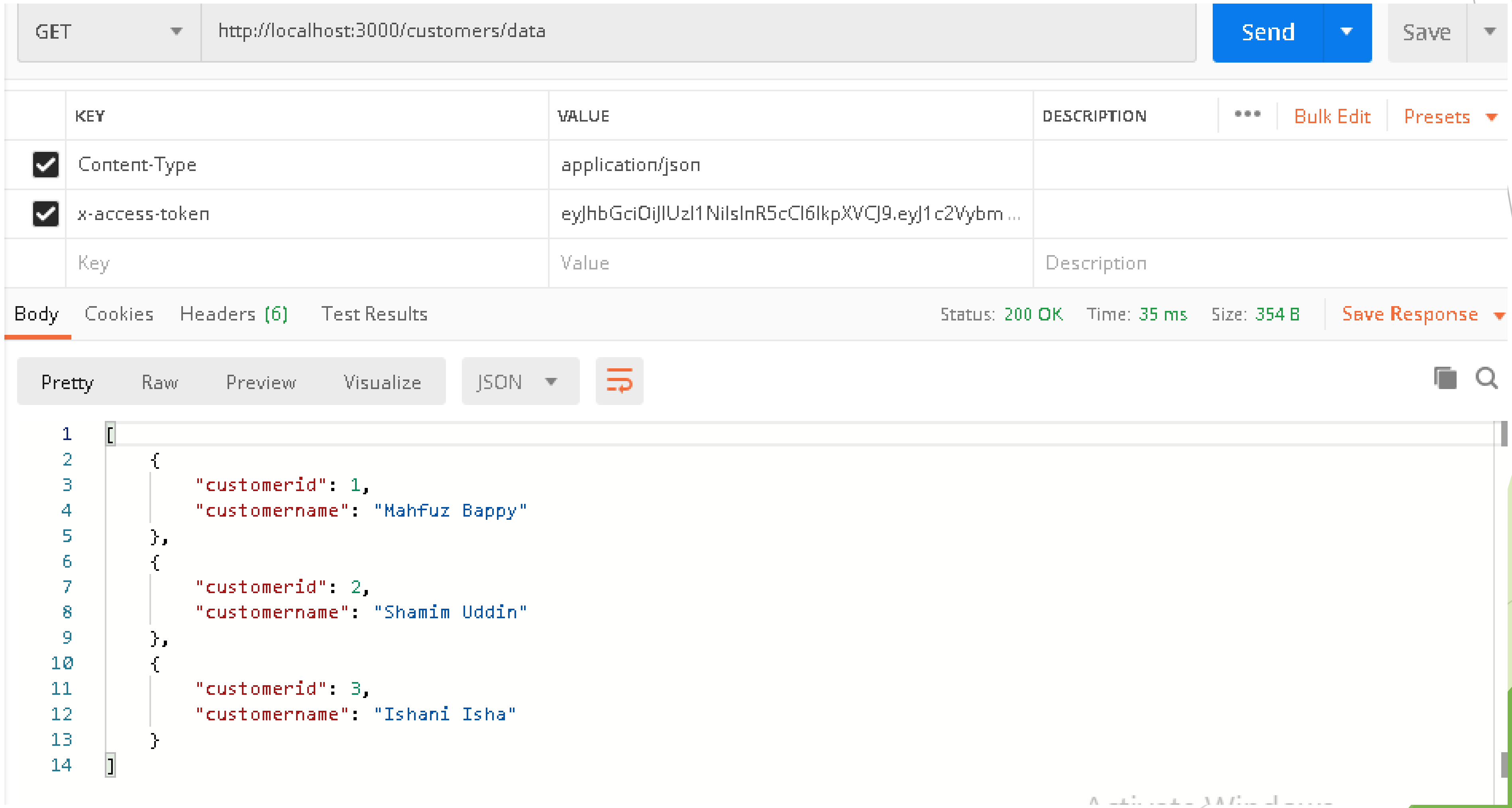
```
{
    "username":"admin",
    "password":"admin"
}
```

Status: 200 OK Time: 28 ms Size: 435 B

```
{
    "message": "Login Successful",
    "jwtoken":
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwicGFzc3dvcmQiOiJhZG1pbiIsImlhdCI6MTU4OTk2MjI0MywiZX
        hwIjoxNTg5OTYyNTQzfQ.JKDymMbTq-LXBkqUYhJY58xokDpJiR8V_K4cdW7ZvkY"
}
```

Then enter URL :https://localhost:3000/customers/data pass generated token in header “x-access-token” by copying and paste it to value section.

After that click on send button to send the request to server.



http://localhost:3000/customers/data

KEY	VALUE	DESCRIPTION
Content-Type	application/json	
x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybm ...	
Key	Value	Description

Status: 200 OK Time: 35 ms Size: 354 B

```
[
    {
        "customerid": 1,
        "customername": "Mahfuz Bappy"
    },
    {
        "customerid": 2,
        "customername": "Shamim Uddin"
    },
    {
        "customerid": 3,
        "customername": "Ishani Isha"
    }
]
```

That’s it, finally we know how to secure server request by token based authentication without storing any information

Summary:

- About JSON Web Token (JWT)
- Using JWT to secure ExpressJS API
- Using router level middleware
- Passing values within header with client request
- Accessing passed value from header