**Token Based authentication in Node.js using JSON Web Tokens**

**Abstract:**Node.js based applications can be made more secured using Token Based Authentication. We will see how to use the Json Web Token package for this purpose.

Implement Token based security in Node.js applications.

Token based authentication is one of the most powerful and useful mechanisms for securing modern web applications. In ASP.NET Web API, we already have support for Token based authentication.

The application in this article is implemented using Visual Studio Code (VSCode). This is a free IDE by Microsoft used for building Modern Web and Cloud based applications. This can be installed on Windows, Linux and MAC OSX.

This new IDE is based on electron and can be downloaded from [this link](http://code.visualstudio.com/). We need to install following prerequisites for implementing application.

· Node.js  - <https://nodejs.org/en/>

· MongoDB - <https://www.mongodb.org/downloads#production> (used for creating database)

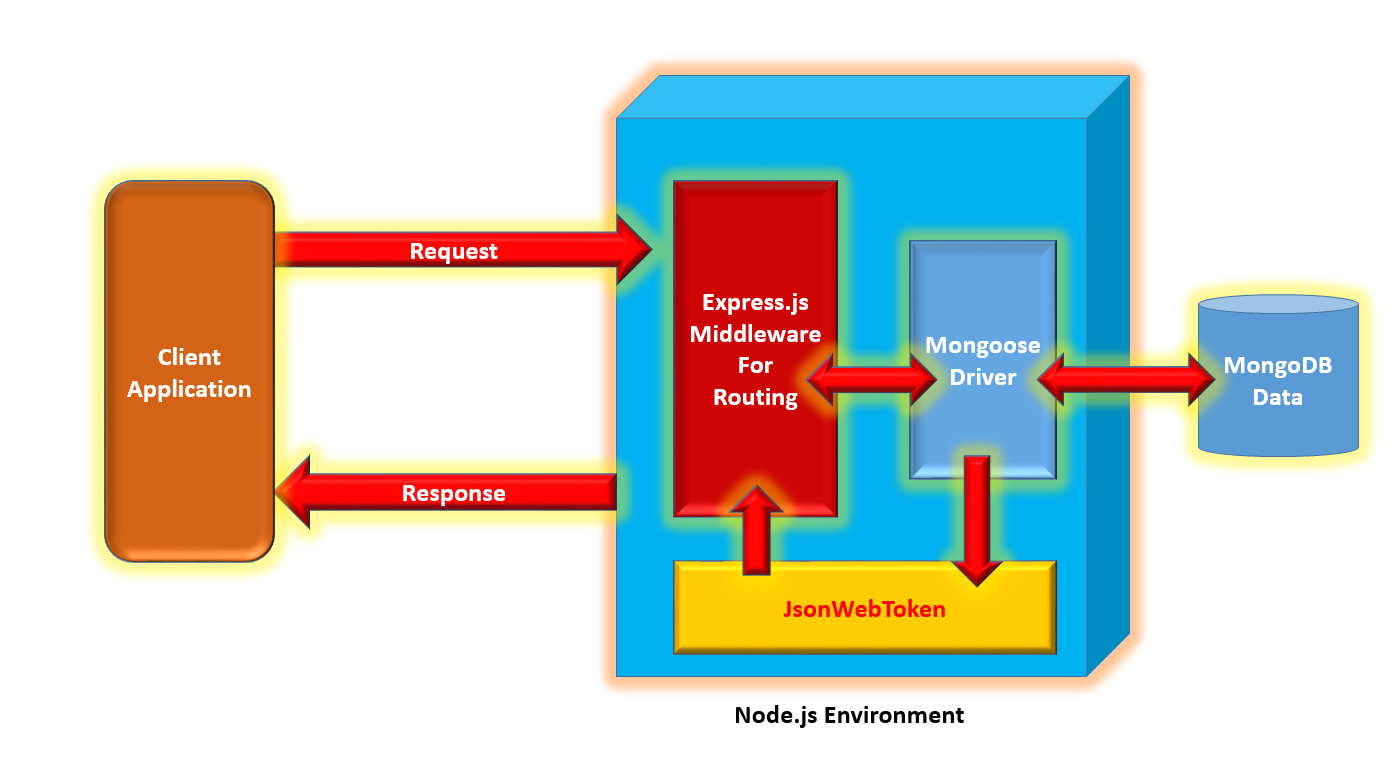
· Mongoose - The driver for performing Read/Write operations with MongoDB.

· Express - This will be used to contain logic for implementing routing for http calls.

· JsonWebToken - provides mechanism to generate the Token and manage signing for the applications.

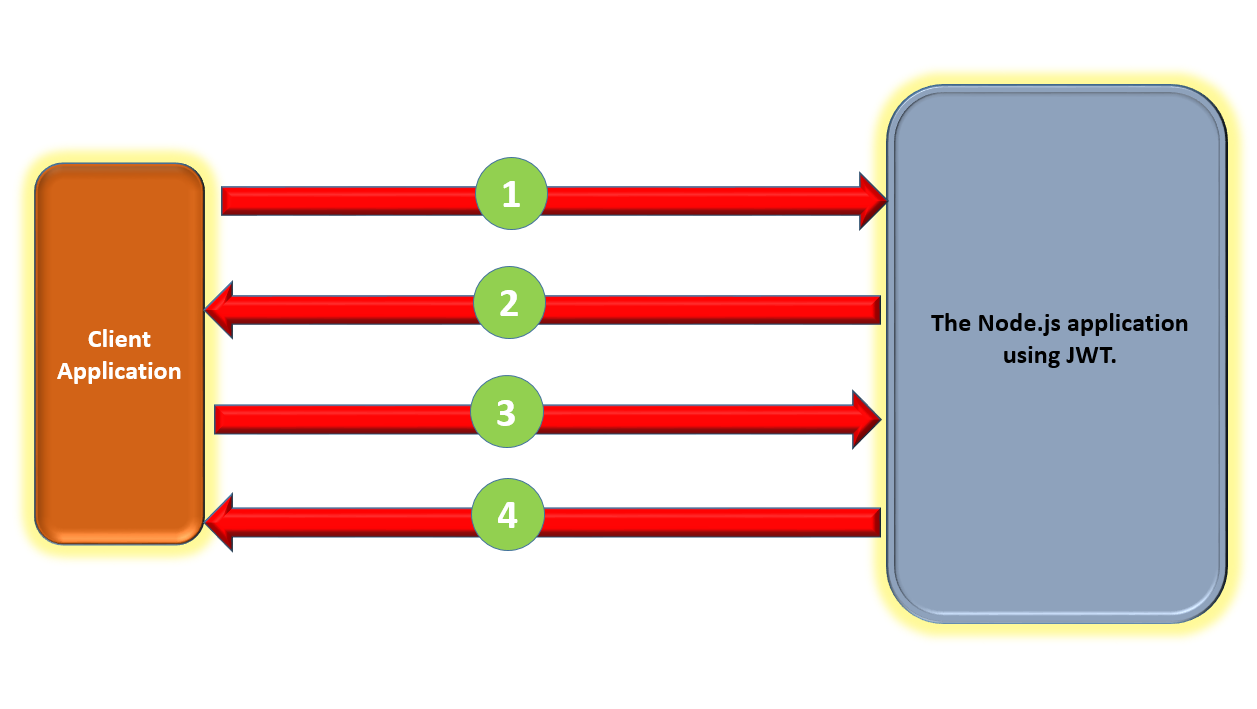
**Authentication in Node.js API with JSON Web Tokens**

The following diagram explains the architecture of the application.



The above diagram explains the technical details of the application. The Request will be accepted using Route expression exposed and managed by Express. This further uses the [Mongoose driver to connect to MongoDB database](https://www.dotnetcurry.com/nodejs/1165/connect-mongodb-using-mongoose) and contains database and collections for User Information and application data. The JSON Web Token will be used to generate token and manage the sign in for the user.

The JSON Web Token (JWT), is a token format used in the authorization headers. This token is used to manage secure communication across two different applications. The use of JWK can be explained using the following image.



1. The end-user posts the credentials (username and password) to the server applications.

2. The application receives the credentials from a Http request and validates these credentials by querying the database. If the user is validated, the token will be created based on the user information. This token is stored in the response header. This token information, which consist of token, expiration, etc. will be stored on the client-side in the browser’s localStorage or sessionStorage.

3. The token will be used for making each request from client to the application. This token information will be present in the request header. The server application is now responsible for validating an incoming token.

4. If the token is valid, then the response for the desired result will be send by the service to the client.

[Install MongoDB and configure it](https://www.dotnetcurry.com/nodejs/1161/connect-mongodb-nodejs-mongodb-driver).  Visit the command prompt and execute the following command

mongodb > Use UserInformation

This will create a new database of name **UserInformation.**We will use this to create a collection to store user information and the Application data.

**Node.js with JSON Web Tokens - Example**

**Step 1:**On your hard drive create a new folder of the name Node\_Token\_Auth. Open this folder in Visual Studio Code. In this folder add following files

* App.js
* Logic.js
* Index.html

**Step 2:**Right-Click on app.js and select option **Open in Command Prompt.**This will open the command prompt. Run the following command from the Command prompt:

|  |
| --- |
| Npm install express jsonwebtoken mongoose body-parser –save |

This will install packages mentioned in the project under node\_modules folder. To install TypeScript Definitions for the above packages run the following commands. This will be useful for intellisense.

|  |
| --- |
| npm install -g tsd  tsd query express --action install  tsd query body-parser --action install  tsd query mongoose --action install |

We will be creating a client application using Knockout, jQuery, bootstrap and we will host the html page using a http server. To install the http server we need the following package

|  |
| --- |
| npm  install -g http-server  npm install – cors |

The CORS package is required for Corss-Origin-Resource-Sharing.

The other client-side packages will be installed using the following commands:

|  |
| --- |
| Npm install jQuery  Npm install knockout  Npm install bootstrap |

**Step 3:**In the app.js add the following code

|  |
| --- |
| ///1. Get All Packages  var express = require('express');  var bodyParser = require('body-parser');  var mongoConnect = require('mongoose');  var jsonWebToken = require('jsonwebtoken');  var cors = require('cors');    //2. Declare the Express object  var object = express();  object.use(cors());  //3. The Express Route  var apirouting = express.Router();    //4. Set the port for communication  var communicationPort = 5050;    //5. Object to store MongoDB Connection and Secret  var impObject = {      'jwtSecret': 'xtytzt00700tytx',      'connStr': 'mongodb://localhost/UserInformation'  };      //6. The MongoDB Connection  mongoConnect.connect(impObject.connStr);  //7. The Connection object  var db = mongoConnect.Connection; // The Connection object    if (db == 'undefined') {      console.log("The Connecion issues");  }  //8. The User Schema  var userInfoSchema = mongoConnect.Schema({      UserName: String,      Password: String,  });    //9. Map Schema with the Model Object  var userModel = mongoConnect.model('userModel', userInfoSchema);  //10. Define the Schema for person  var personSchema = mongoConnect.Schema({      PersonId: String,      PersonName: String,      Email: String,      Age: String  });  //11. Map the Person Schema  var personModel = mongoConnect.model('Person', personSchema, 'Person');      //12. The Secret  object.set('jwtSecret', impObject.jwtSecret);    //13. The Body Parser to parse incoming data from request  object.use(bodyParser.urlencoded({ extended: false }));  object.use(bodyParser.json());      //14/ logic for Creating user  apirouting.post('/createuser', function (request, response) {      // UserName:request.body.UserName      //request.body.Password      console.log('The Request is being Processed');      var user = new userModel({          UserName: request.body.UserName,          Password: request.body.Password      });        //Lets save the sample user      user.save(function (error) {          if (error) {              console.log('Some Error Occured');              throw error;          }          console.log('User Created...');          response.json({ createduccessfully: true });      });  });          //15. Get all users  apirouting.get('/users', function (req, res) {      userModel.find({}, function (err, users) {          res.json(users);      });  });    //16. Lets authenticate user  apirouting.post('/authuser', function (request, response) {      //16a. we need to check if the user exist      userModel.findOne({          UserName: request.body.UserName      }, function (error, usr) {          if (error) { console.log('Some error  occured '); throw error; }          if (!usr) {              response.json({                  authsuccess: false,                  description: 'User Authentication failed because user not found.'              });          } else if (usr) {              //16b check if the received password matches with the data store              if (usr.Password != request.body.Password) {                  response.json({                      authsuccess: false,                      description: 'User Authentication failed because provided password is wrong.'                  });              } else {                  //16c. generate the token because we have the username and pasword                  //matching                  var accessToken = jsonWebToken.sign(usr, object.get('jwtSecret'), {                      //Set the expiration                      expiresIn: 3600 //we are setting the expiration time of 1 hr.                  });                  //send the response to the caller with the accesstoken and data                  console.log('Authentication is done successfully.....');                    //16d.                  response.json({                      authsuccess: true,                      description: 'Sending the Access Token',                      accessToken: accessToken                  });              }          }      });    });  //17.  apirouting.get('/persons', loadPersonData);    //Function to load PersonData  function loadPersonData(req, resp) {      console.log('In Load Person');          personModel.find({}, function (err, persons) {          resp.json(persons);      });      };    //18.  object.use(apirouting);  object.listen(communicationPort, function () {      console.log('Listening on  port 5050');  });  //object.listen(communicationPort);    console.log('started listeing on the port ' + communicationPort); |

The above code does the following (The line number in the code matches with comment number applied on the above code.)

1. This block loads all the required packages for the current application. This loads Express, Mongoose, Json Web Token, cors.

2. This step declares an instance for express of name **object** and it asks the express instance to use CORS.

3. This statement defines an instance for the express route object.

4. The communication port is set to 5050.

5. This is an important step. This is a JavaScript object used to define secret for the Token, and the connection string for Mongo database.

6. This statement connects to the mongodb database.

7. This statement checks the connection status to verify whether it is successful or not.

8. This is a JavaScript object for creating Users Information. This contains UserName and Password properties.

9. This statement maps the User Schema with the Collection in the MongoDB. If this is not present then it will be created.

10. This is a JavaScript object used to define schema.

11. This statement maps the JavaScript object with the Collection in the Mongodb database.

12. This statement set the Token Secret using **jwtSecret**value.

13. The block configures the express object to use body-parser so that the posted data can be used and mapped with the JavaScript object while creating users.

14. The **post**function of the express routing object is used to define route **/createuser.**This reads the User information from the request body and maps it with the **userModel**object. This object is already mapped with the **usermodels** collectioncreated in Mongodb. This object has a **save()**function using which the data will be saved in the Mongo collection.

15. The **get**routing function is used to read all users.

16. The **post**function of the route object defines route of name **/authuser.**This function has the following specifications

* a. Read the username from the body and check if it exist or not, if not, present it with an error message as **User Authentication failed because user not found.**
* b. If the user is present but password does not match then the error message is **User Authentication failed because provided password is wrong.**
* c. If the login is successful then **sign()**function of **jsonWebToken**object will be called. This function accepts the current user information and a secret to generate the access token. We can set an expiry for the token using **expiresIn**property. The token will be responded.

17. The **get**function of the express rout object define **/persons**route. This calls the **loadPersonData**function to return all persons using **personModel.**

18. This statement configures the express route object with express object.

**Step 4:**In the logic .js, add the following code

|  |
| --- |
| function securityViewModel() {      var self = this;      //1.      self.responseData = ko.observable();        self.userName = ko.observable();        self.userRegistrationEmail = ko.observable();      self.userRegistrationPassword = ko.observable();      self.userRegistrationConfirmPassword = ko.observable();        self.userLoginEmail = ko.observable();      self.userLoginPassword = ko.observable();        self.accessToken = ko.observable();      self.refreshToken = ko.observable();      self.Persons = ko.observableArray([]);      //2.      self.registerUser = function () {            self.responseData('');            var userRegistrationInfo = {              UserName: self.userRegistrationEmail(),              Password: self.userRegistrationPassword()          };            if (self.userRegistrationConfirmPassword() === self.userRegistrationConfirmPassword()) {              alert('Passsword Match');              $.ajax({                  type: 'POST',                  url: 'http://localhost:5050/createuser',                  data: userRegistrationInfo,                  contenttype: 'application/json; charset=utf-8',                  datatype: "json"              }).done(function (data) {                  self.responseData("User is Successfully " + data);                  self.userRegistrationEmail("");                  self.userRegistrationPassword("");                  self.userRegistrationConfirmPassword("");              }).error(function (err) {                  self.responseData("Error " + err.status);              });            } else {              alert('Please check the password');          }          };      //3.      self.clear = function () {          self.userRegistrationEmail("");          self.userRegistrationPassword("");          self.userRegistrationConfirmPassword("");      };      //4.      self.login = function () {            //This is the information to pass for token based authentication          var userLogin = {              UserName: self.userLoginEmail(),              Password: self.userLoginPassword()          };          var data = 'UserName=' + userLogin.UserName + '&Password=' + userLogin.Password;          alert(data);          $.ajax({              type: 'POST',              url: 'http://localhost:5050/authuser',              contenttype: 'application/x-www-form-urlencoded',              data: data          }).done(function (resp) {              self.userName(resp.userName);              //Store the token information in the SessionStorage              //So that it can be accessed for other views              sessionStorage.setItem('UserName', resp.UserName);              sessionStorage.setItem('accessToken', resp.accessToken);              //  loadData();          }).error(function (err) {              self.responseData("Error " + err.status);          });      };      //5.      self.getData = function () {          var accessToken = sessionStorage.getItem('accessToken');          console.log(accessToken);          var authHeaders = {};          if (accessToken) {              authHeaders.Authorization = 'Bearer ' + accessToken;          }          $.ajax({              url: 'http://localhost:5050/persons',              type: 'GET',              headers: authHeaders          }).done(function (resp) {              alert('data ' + JSON.stringify(resp));              self.Persons(resp);          }).error(function (error) {              self.responseData('Error Occured ' + error.status);          });      };      };    var secure = new securityViewModel();  ko.applyBindings(secure); |

This is the Knockout View Model with following specifications (see code number above and its corresponding explanation below)

1. This block defines observable objects User Registrations information, login information, access token, etc.

2. The **registerUser**function calls the **/createuser**api from the node server to create user.

3. The **clear** function clears entered data from User interface.

4. The **login** function will pass the login user information to the server using **/authuser**api. If the user is valid, then on the successful completion of the Ajax call, the access token will be stored in the **sessionStorage**on the client.

5. The **getData**function will use the accessToken on the client side to make calls to /**persons**api. This call will pass the accessToken in the request header. The call will be successful based on the accessToken.

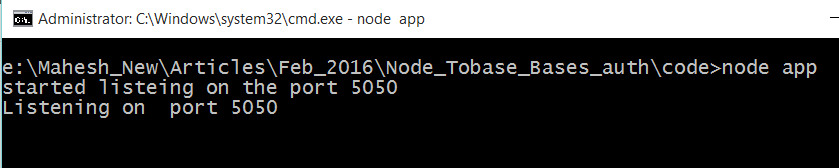
**Step 5:**Add the following markup in the Index.html and set the databinding with the viewmodel created in previous step.

|  |
| --- |
| <html>      <head>          <link rel="stylesheet" href="/node\_modules/bootstrap/dist/css/bootstrap.min.css">      </head>      <body>        <table class="table table-bordered table-striped">             <tr>                 <td>                     <table class="table table-bordered table-striped">                         <tr>                             <td>User Name:</td>                             <td>                                 <input type="text" id="unameregister" class="form-control"                                  data-bind="value:userRegistrationEmail">                             </td>                         </tr>                         <tr>                              <td>Password:</td>                             <td>                                 <input type="password" id="pwdregister" class="form-control"                                  data-bind="value:userRegistrationPassword">                             </td>                         </tr>                         <tr>                              <td>Confirm Password:</td>                             <td>                                 <input type="password" id="confirmpwdregister" class="form-control"                                 data-bind="value:userRegistrationConfirmPassword" >                             </td>                         </tr>                         <tr>                               <td>                  <input type="button" id="btnregister" value="Register" class="btn btn-success"                   data-bind="click:registerUser"/>                </td>                <td>                     <input type="button" id="btncancelregister" value="Cancel"                     class="btn btn-default" data-bind="click:clear"/>                </td>                         </tr>                     </table>                 </td>                 <td>                 <table class="table table-bordered table-striped">                   <tr>                <td>                    User Name:                </td>                <td>                    <input type="text" id="txtuname" class="form-control"                     data-bind="value:userLoginEmail"/>                </td>            </tr>             <tr>                <td>      Password:                </td>                <td>                    <input type="password" id="txtpwd" class="form-control"                     data-bind="value:userLoginPassword"/>                </td>            </tr>             <tr>                <td>                  <input type="button" id="btnlogin" value="Login" class="btn btn-success"                   data-bind="click:login"/>                </td>                <td>                     <input type="button" id="btncancel" value="Cancel" class="btn btn-default"/>                </td>            </tr>                 </table>                 </td>               </tr>            <tr>                <td>                    <input type="button" value="Load Data"  data-bind="click:getData"  class="btn btn-success">                </td>            </tr>          </table>        <hr/>        <br/>        <div> </div>        <hr/>        <table class="table table-bordered table-striped">            <thead>                <tr>                    <th>Person Id</th>                    <th>Person Name</th>                    <th>Email</th>                    <th>Age</th>                </tr>            </thead>            <tbody data-bind="foreach:Persons">                <tr>                    <td>                        <span data-bind="text:PersonId"></span>                    </td>                     <td>                        <span data-bind="text:PersonName"></span>                    </td>                     <td>                        <span data-bind="text:Email"></span>                    </td>                     <td>                        <span data-bind="text:Age"></span>                    </td>                </tr>            </tbody>        </table>      </body>      <a href="/node\_modules/jquery/dist/jquery.min.js">/node\_modules/jquery/dist/jquery.min.js</a>      <a href="/node\_modules/knockout/build/output/knockout-latest.js">/node\_modules/knockout/build/output/knockout-latest.js</a>      <a href="http://logic.js" rel="nofollow" target="\_blank">http://logic.js</a>    </html> |

**Step 6:**Right-click on app.js and select the **Open in Command Prompt**option. Run the following command

|  |
| --- |
| Node app |

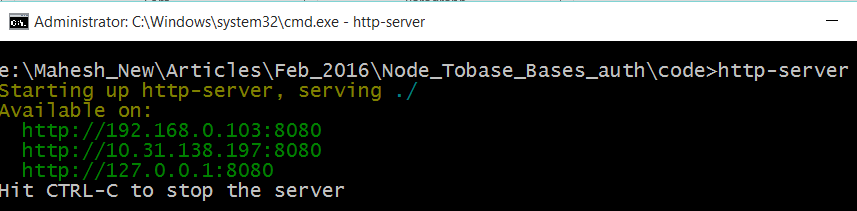
This will start the Node server on 5050 port with all the APIs created using express.



Right-Click on index.html and select **Open in Command Prompt.**Run the following command from the command prompt.

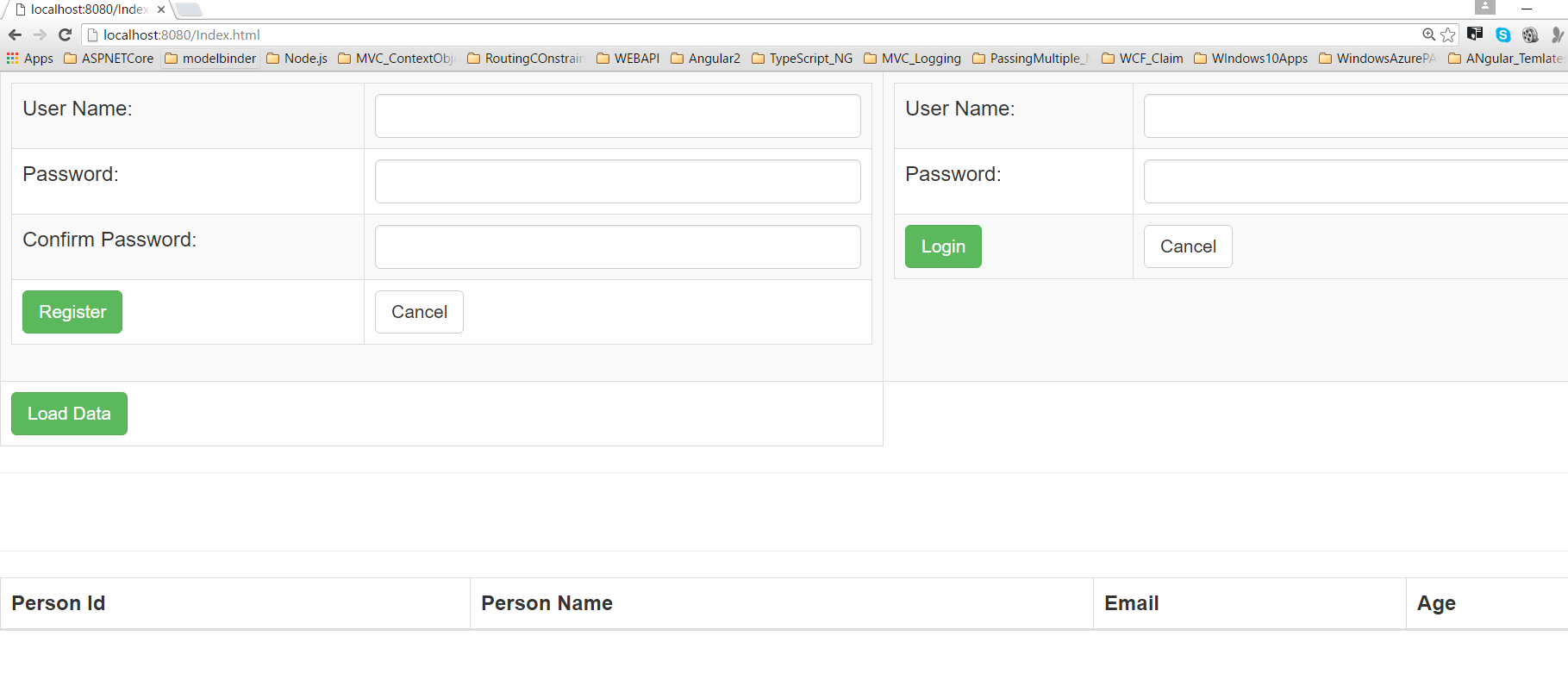
|  |
| --- |
| http-server |

This will start the Http Server on port 8080



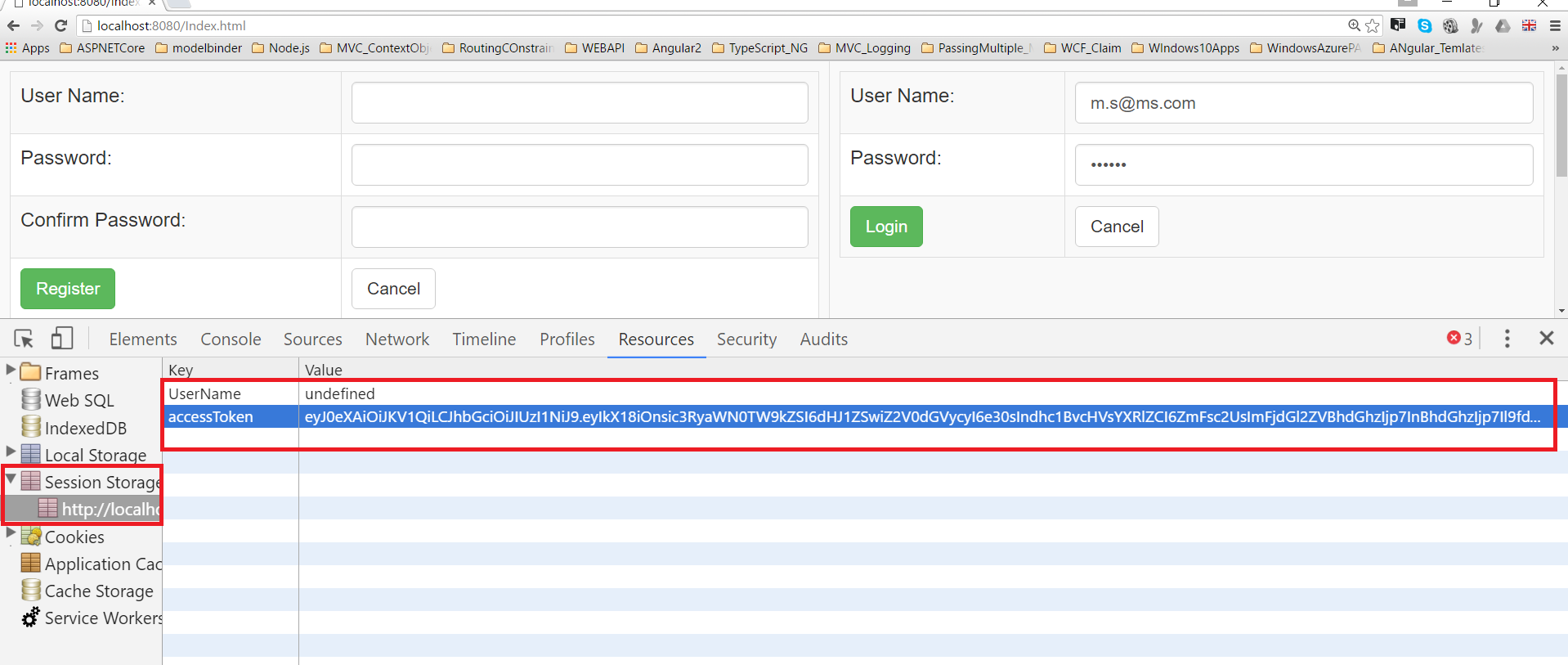
Open the browser and enter the following address

<http://localhost:8080/Index.html>

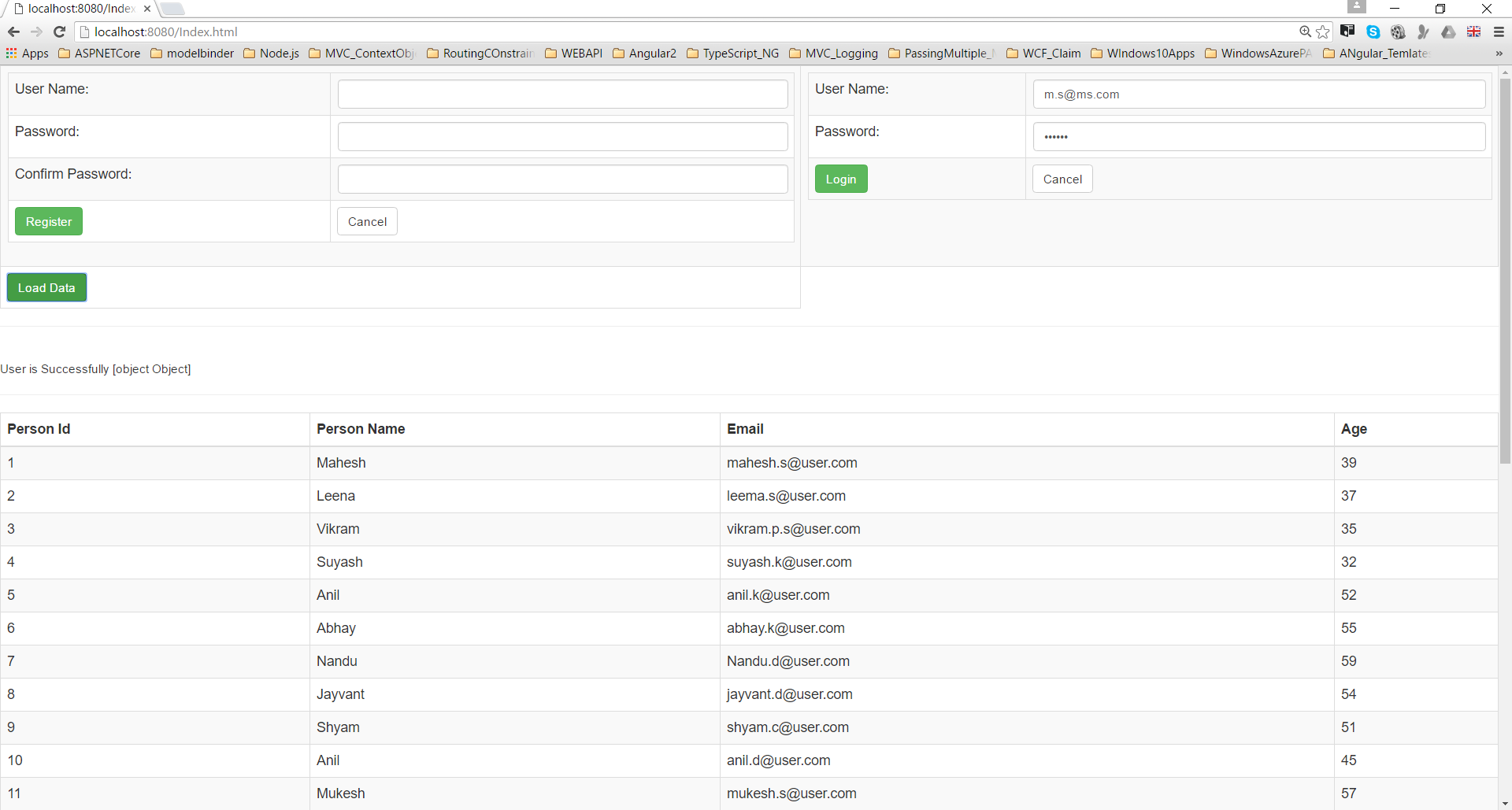


Enter UserName, Password and Confirm password and click on the register button. Once the user is created, enter the same username and password in the right-hand side text boxes and click on the **Login**button as shown in the following image.

(Note: Open the developer tool in IE using F12 (Chrome CTRL+Shift+I) and the sessionStorage will show Access Token)



The Access Token is stored on the client side. Click on the **Load Data** button, the person data will be loaded as shown in the following image



**Conclusion:** Node.js based applications can be made more secured using Token Based Authentication. The Json Web Token package is of much importance here.