CIPHER WEBSITE NODE, EXPRESS & MONGO DB REPORT

Coursework Submission 2 for SET08101 Web Tech
Edinburgh Napier University 2018-2019 Trimester 2
Aiden Henry 40344585 *April 2019*

INTRODUCTION

As part of the Authors second piece of coursework submission for Web Tech, they were tasked to create a back-end CRUD implementation for the previous cipher website submission using Node.js and the Express framework. This solution would allow for users to log in to their account, have the ability to view, send and delete encrypted messages and it could utilise from three options to provide data persistence of user data on the server, fs filesystem, an SQL database or MongoDB.

The original cipher website submission was modified to allow for user log in via the drop-down menu. This new version would provide for user log in and password check, rejection to root page if incorrect password is used, and if detected as not already being a member automatically signing them up for a new account. Once user was verified against database, they would be redirected to their user page where they can access the functions to send and receive messages. Encrypt and De-Crypt functions will be provided by the drop-down menu as in the original submission.

The server database selected for this project was MongoDB, as it was reasoned that this would automatically provide the necessary framework of the database with its well-defined commands [1]. However, as will be discussed later in the report this turned out to be a more complicated solution compared to a simple file system CRUD implementation [2]. The templating language chosen was EJS (Embedded JavaScript).

SOFTWARE DESIGN

Client Web Interface Design

User Log In

The functional requirements of this project required that a user of the website could log in to their account and be able to receive and send messages to other users. It should also provide the ability to encrypt and de crypt these messages and the ability to delete them.

This required a slight modification to the original script to add a new HTML <section> containing the elements for the user log in section via the drop-down menu. This new section would provide the client user name and password to be POST to the server using the HTML form command, this data will be checked against the database. If user name and password don't match the page will reload itself while if it is a new user a new account will be created. Once a valid user is confirmed it will redirect to their user page.

User Page

This page is almost an exact copy of the index page; however, it will contain the HTML form elements to transfer the data containing the recipient user name and message contents. It will also receive the data of messages relating to that user from the database and display this in a HTML table. This section will float below the cipher content selected from the drop-down menu and will allow users to easily copy messages across to the relevant ciphers to encrypt and de crypt.

Server Web Design

User Log In

The server will be implemented in Node.js with the Express framework to provide for the server connections, it will use the MongoDB for data storage of user names, passwords, sent and received messages. It will retrieve the data of the user attempting to log in and check that against the database, if there is no record of that user a new account will be created, and they will then be redirected to their own user page. If it finds the username but the entered password is incorrect it will redirect to the root index page. When it receives a username and password that corresponds with the database the user page will load for that user and it will retrieve the messages.

User Page

This page will load on the successful log in of a user, it will use the confirmed username to retrieve the messages from the MongoDB database and respond with that data back to the client front end to be rendered in a table. It will also handle user CRUD interaction of the database, such as sending messages (creating new entries in user document

messages), receiving messages (retrieving contents of user document messages in database and displaying the data), changing password (updating the database with new data) and deleting messages (accessing the user database and deleting message).

IMPLEMTATION

Client Web Interface

User Log In

Some redesign of the original index HTML code was required to make the client-side JavaScript functions work as intended. An example of this was the drop-down menu for cipher selection, the previous workaround using an href tag caused errors when trying to call the functions. By deleting the href tag from the menu selection restored full functionality. This new modified version will become the root directory of the webpage served by Node.js & Express.

```
<a onclick="templar_select()">Templar Cipher</a>
<a onclick="user_select()">User Log In</a>
```

An option for logging in was added to the drop-down menu, this would activate and deactivate the CSS script to “block” and “none” the user log in content. The design of this section followed the same implementation of the previous work, HTML section, image, article etc but now includes a new HTML form tag to pass user name and password to the server. The form tag will POST the data to the server address /adduser and the node.js commands will perform an authentication check against the database.

```
<section id="users">
    <h2>User Log In / Register</h2>
    <img src="images/encryption.jpg" alt="Image of a cipher with key" width="350" height="350">
    <article>
        Welcome to the User Log In and Register page.  If you sign in you
        will automatically be registered and will be able to send secret messages to other users
        of JSON. If you are not already a user then just enter the name
        and the password you would like and click log in and you will automatically
        be registered.<br>
    </article>
    <form id="formAddUser" name="adduser" method="post" action="/adduser">
        <input id="inputUserName" type="text" placeholder="username" name="username" />
        <input id="inputUserPwd" type="text" placeholder="password" name="userpwd" />
        <button class="button" id="btnSubmit" type="submit">Submit</button>
    </form>
</section>
```

User Page

Once user authentication has occurred, they will be redirected to their user page. This page is almost an exact copy of the Index page, but it contains an HTML form tag to POST message data to the server address of /messages also it contains a HTML table tag to display the received and sent message data associated with that user from the database. A HTML form tag will provide a “Delete” button to remove the messages from the database.

```
<section id="logged_in">
        <form id="messager" name="messager_name" method="post" action="/messages">
        <p>Enter the user name to send to:</p><input id="send_to_user" type="text" pl
        <p>Enter the message you wish to send:</p><input id="sending_message" type="t
        <br><button class="button" id="btnSubmit" type="submit">Submit</button>
        </form>
</section>
```

As the viewer runs on EJS, it was simple to implement the dynamic HTML table of the messages received by using embedded JavaScript.

```
<table id="received_message_table">
    <tr id="received_table">
        <th>FROM</th>
        <th>MESSAGE</th>
        <td>
            <form id="delete_message" method="post" action="/delete/received">
                <button class="button" name="delete" type="submit">Delete</button>
            </form>
        </td>
    </tr>
    <% for (i=0; i < received_from.length; i++){ %>
    <tr>
        <td><%= received_from[i] %></td>
        <td><%= in_message[i] %></td>

    </tr>
    <% } %>
</table>
```

The next screenshot shows final version of the User page after Node.js and MongoDB implementation (in this case test user Eve).

Server Implementation

MongoDB

Before coding of the node.js / express framework could begin, it needed to have a data repository to work with. The method chosen for this project was the MongoDB and additional library monk solution, which was installed using the “npm” command on the command line interface. This was a local version of MongoDB and not the cloud based one. Once installed, a CLI was opened to access the mongo.exe and a new collection called user_data was created, and the service was then started.

This tells the app that we want to use the user_data collection on the MongoDB and that it should use monk to carry out this operation.

```
var monk = require('monk');
var db = monk('localhost:27017/user_data');
```

To allow Node.js and the Express framework to gain access to this database, extra lines of code were inserted into the app.js file.

```
// Make our db accessible to the router
app.use(function(req,res,next){
  req.db = db;
  next();
});
```

Node.js & Express Routes Index.js

The file index.js is perhaps the most important file of the project as it handles user CRUD requests such as user log in, creation of new users, retrieving and sending messages and deleting messages.

User Log In

The client facing web app will POST the username and password information to the server address /adduser. This script would activate when a post request was received for that address and start accessing the MongoDB. It then extracts the data from the POST and checks the user details against the one stored in the database. If authentication is successful it will then redirect to their User page.

```
// check if the username matches the password
if (result != null && result.pwd == userPwd)
{
  console.log('Correct username and password: ' + userName + ' ' + userPwd)
  user_ID = userName;
  res.redirect('userlist/'+ user_ID);
}
```

If the authentication failed it will then reload the main route html and user will have to try log in again.

```
// if password does not match username
else if (result != null && result.pwd != userPwd)
{
  console.log('Incorrect password: ' + result.pwd + ' ' + userPwd)
  res.redirect('/')
}
```

While if the user logging in does not have an account on the database, it will create the user and the relevant document fields for messages on the database and redirect to the User page.

```
else
{
// Creating username, password and array for sent /recieved
// messages to Database
user_ID = userName;
collection.insert({
"user" : userName,
"pwd" : userPwd,
"from": [],
"received_message" : [],
"to" : [],
"sent_message": []
}, function (err, doc) {
if (err) {
    // If failed return error
    res.send("Failed to write to database.");
}
else {
    // And forward to success page
    console.log('Adding user to DB: ' + userName + ', with password: ' + userPwd)
    res.redirect("userlist/':user_ID");
}
```

User Page Receiving Messages

When successful authentication has taken place, the users will be redirect to their own page and be able to access their messages. This required the user data to be recovered from the MongoDB and sent back to be displayed by the client. This was achieved by taking the user name variable from the log in section and using that to recover the data. This data was then parsed into separate arrays and assigned to variables which was then sent back by responding with a render of the userlist page and the relevant variables.

```
/* GET User page and messages. C.READ.U.D */
router.get('/userlist/:user_ID', function(req, res) {
  // set DB
  var db = req.db;
  // set DB collection
  var collection = db.get('user_data');
  // find user data
  collection.find({user: user_ID,},
    function(e,docs){
      var parsedArray = Object.values(docs[0]);
      // respond with userlist.ejs and these variables
      res.render('userlist', {
        "user" : parsedArray[1],
        "received_from" : parsedArray[3],
        "in_message" : parsedArray[4],
        "sent_to" : parsedArray[5],
        "out_message" : parsedArray[6]
        });
  });
});
```

User Page Sending Messages

For the user to send messages to other users, this route needed to be able to add these messages to the correct user document on the database. The data is sent by the client in a HTML form with the POST method to the server address /messages. It would receive the request, open the database, parse the recipient user name and message information and add it to their own sent message document and to the recipients received document. This was accomplished by using the MongoDB collection command “update” and by “pushing” the relevant values to the correct documents.

```
/* POST User messages. CREATE.R.UPDATE.D */
router.post('/messages', function(req, res){
  // set DB
  var db = req.db;
  var collection = db.get('user_data');
  // find user and message contents
  var toUser = req.body.to_user;
  var mess = req.body.to_message;
  // strip white space from message
  var toMess = mess.replace(/ /g,'')
  // find recipient entry and update received messages on DB
  collection.update({"user": toUser},
  {$push: {"from": user_ID, "received_message": toMess}},{multi:true});
  console.log('Updated Recipient DB ')
  // find user entry and update sent messages on DB
  collection.update({"user": user_ID},
  {$push: {"to": toUser, "sent_message": toMess}}
  );
  console.log('Updated User DB ')
})
```

User Page Deleting Messages

As part of the functional requirements, the option to delete messages in the user document was required. This was achieved by receiving the POST data sent from the “Delete” button to the server address /delete/received or sent. This script would read the database contents then using the MongoDB command “findOneAndUpdate” would find the user document and then “pop” the first item out of the received from / sent to section and the corresponding messages. It would then re render the userlist page with updated messages information.

```
/* Delete messages. C.R.U.DELETE */
router.post('/delete/sent', function(req, res){
  // set DB variable.
  var db = req.db;
  // set collection.
  var collection = db.get('user_data');
  collection.update({"user": user_ID},
  {$pop: {"to": -1, "sent_message": -1}},{multi:true}
  );
  // find user data
  collection.find({user: user_ID,},
    function(e,docs){
      var parsedArray = Object.values(docs[0]);
      // respond with userlist.ejs and these variables
      res.render('userlist', {
        "user" : parsedArray[1],
        "received_from" : parsedArray[3],
        "in_message" : parsedArray[4],
        "sent_to" : parsedArray[5],
        "out_message" : parsedArray[6]
        });
    });
  });
```

As was previously mentioned MongoDB was chosen because it seemed to be a fully functioning database framework, however, it was discovered during implementation that there was no way to delete specific content (slice) from an array in a MongoDB. The only way to provide the delete function was to use the pop method which limited the options to removing data from the beginning or the end of the array.

CRITICAL EVALUATION

The project requirements discussed previous in this paper stated that the submitted work should meet the following criteria:

1. Web interface for User log in.
2. Web interface for User page:
    a. To send messages.
    b. For Received messages.
    c. For Sent messages.
    d. To Encrypt messages.
    e. To Decrypt messages.
3. Server using Node.js and Express.
4. Data persistence of users and messages.

(1) An extra HTML section containing the user log in was added to the original website submission allowing access to this content through the drop-down menu. This would access the server and on confirmation of authentication, redirect to their userlist page.

(2) A secondary HTML document was created for the user page (userlist).

(2,a) An HTML form tag was added to the userlist page to allow a recipient user information and message to be sent.

(2,b) On successful log in, the server will redirect the user to their user page along with the message data retrieved from the database, this would then be rendered in an HTML table tag viewable by the user.

(2,c) As with the received messages, the sent messages are retrieved from the database and displayed in an HTML table tag.

(3) The server is running Node.js with the Express framework to respond to the user entered HTTP method requests.

(4) The project used the MongoDB and the monk library to store the user and message data. Note: this was a local implantation of MongoDB and not cloud.

On evaluation of the solution, it is believed by the Author that major improvements and extra functionality should have been implemented to provide a more rounded messaging service and user administration.

A major barrier to planning and implementation of the server and corresponding JavaScript code was the prior coursework submission. As this website ran from one page, assigning different routes in the index.js for the different functions was impossible. This seemed to complicate the project to such a degree that the option to re-create the original work over multiple pages was considered, but ultimately rejected due to time constraints.

The MongoDB is a powerful tool for those who have had training, the Author realised that they had misused the database to try to create an array of array's for user information instead of utilising the key : value methods, this directly lead to the problems with removing content from the database and having to resort to the pop method.

The following are a list of functionalities that the Author wanted to implement but due to unforeseen circumstances was unable to complete the work in the allotted time (Please see Personal Note in Appendix),

- User administration page.
- Ability to change user password.
- Ability to delete user account.
- Encrypted passwords.
- HTTPS (self-signed certificate).
- Group chat.

PERSONAL EVALUATION

When the project began, the Author had zero experience with Node.js, the Express framework or MongoDB. Implementation of Node.js and the express framework progressed smoothly, due to the experience gained of JavaScript generally from the previous coursework. However, progress was slow during implementation of MongoDB due to unfamiliar mongo commands to implement CRUD. Ultimately, MongoDB proved to be more of a barrier for this project than a simple File System CRUD solution, which if this project was being redone would now be the preferred solution for data persistence.

CONCLUSION

Overall the Author believes that they have produced a solution that has met the functional requirements of the project. However, it is not as well rounded nor does it have the extra functionality that the Author would have liked to implement prior to beginning the project.

Though with the experience gained with CRUD implementations and the HTTP methods with Node.js request and respond, has given the Author the deeper understanding of the development and implementation process of a web app.

References

[1] MongoDB, "The MongoDB 4.0 Manual," 10 04 2019. [Online]. Available: https://docs.mongodb.com/manual/.

[2] W3Schools.com, "Node.js File System Module," 08 04 2019. [Online]. Available: https://www.w3schools.com/nodejs/nodejs_filesystem.asp.

Bibliography

[1] Stack Overlow, "MongoDB check to see if values exist," 30 03 2015. [Online]. Available: https://stackoverflow.com/questions/29355134/mongodb-check-to-see-if-values-existnode-js. [Accessed 01 04 2019].

[2] Mozilla, "CSP: script-src," MDN web docs, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src. [Accessed 29 03 2019].

[3] D. Banas, "NodeJS MongoDB Tutorial," 18 01 2016. [Online]. Available: https://www.youtube.com/watch?v=Do_Hsb_Hs3c. [Accessed 02 04 2019].

[4] C. Buecheler, "The Dead-Simple Step-By-Step Guide for Front-End Developers to Getting Up and Running With Node.JS, Express, and MongoDB," Closebrace, 04 04 2019. [Online]. Available: https://closebrace.com/tutorials/2017-03-02/the-dead-simple-step-by-step-guide-for-front-end-developers-to-getting-up-and-running-with-nodejs-express-and-mongodb. [Accessed 29 03 2019].

[5] A. Kumar, "10 Most Common Commands for MongoDB Beginners," DZone, 17 10 2017. [Online]. Available: https://dzone.com/articles/top-10-most-common-commands-for-beginners. [Accessed 04 04 2019].

APPENDIX

Personal Note:

The Author suffered a loss to the family in the week prior to hand in, that is why the quantity and quality of this report and submitted code is not up to the Authors usual standards.