Contents

INTRODUCTION

The objective of this project is to set a platform where users can register their details and send and receive coded messages from other users. This was achieved by combining both client and server-side features. All the client files have been stored in the "public" folder where we can find the HTML, CSS and JavaScript files that performs the encode and decode of the messages. The server-side files are all located in the main root.

Resources

I took the old cypher website as a base to start to work out the functionality of it. The programs I have used:

-Node JS to work with the server, collect data from "forms" and interact with the database.

-Node modules:

- Npm as a package manager for Node JS.
- Express as a framework that provides HTML templates and allows to perform actions based on HTTP method and URL.
- SQLite as a database system to store personal details and messages from users.
- Body-parser to validate input from users and manage input under request.body property.

As a background reading I check the next books and tutorials:

- Pro Express.js
- w3schools.com
- docs.npmjs.com/about-npm/

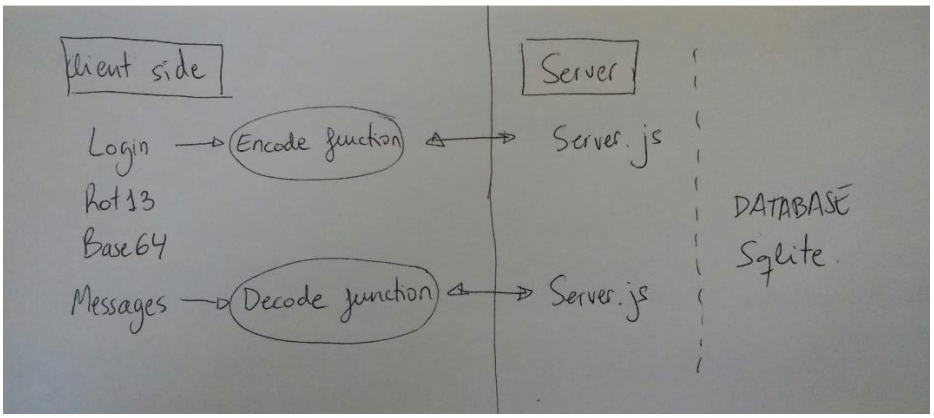
Functionality

The platform consists in:

- Login/sign up page where users can enter their personal details (first name, surname and password). A POST request from the server will authenticate the details entered by the user. If the details are already in the database, the users will be redirected to the next stage, otherwise, their details will be stored in the database created with SQLite.
- Rot 13 webpage allows the user to encode a message and enter the details of the person to the message must be delivered. A POST request inserts the encoded message into the database.
- Base 64 webpage has the same functionality than the Rot 13 webpage.
- Messages webpage where users must enter their personal details to get the messages sent to them by other users. A GET request provides the encoded message from the database to the webpage. The user can decode it right at that moment by clicking in a button that triggers the decoding function.

SOFTWARE DESIGN

Since HTML, CSS and JavaScript files for the client side was already done, I just needed to plan a strategy to include those in Node JS to work on the server side.



SITE IMPLEMENTATION

Once Node JS is being installed in the system, I have downloaded the packages needed and I refer them into the server.js file that will run the server in Node JS.

```
var express=require('express');
var app = express();
const sqlite3 = require('sqlite3').verbose();
let db = new sqlite3.Database('sqlite3/users.db');
var bodyParser = require('body-parser');


app.use(express.static(__dirname + '/public'));
app.use(bodyParser.urlencoded({ extended: false }));
```

- I include the Port listener:

```
app.listen(3000, function () {
    console.log("Server is running on port 3000");
});
```

- I add the <form> method attribute into the HTML for every POST and GET request:

```
<form action="/receiverot13" method="GET">

<form action="/receiveb64" method="GET">

<form action="/users" method="POST">

<form action ="/messagerot13" method="POST">

<form action="/messageb64" method="POST">
```

- I add the POST request to create the login/sign up user account:

```
//sign up

app.post('/users', function (request, response) {
    db.get('SELECT fname, surname, password FROM users WHERE fname=? AND surname=? AND password=?',
        [request.body.fname, request.body.surname, request.body.password], function (err, row) {
            var post = request.body;
            if (!row) {
                db.run('INSERT INTO users VALUES(?,?,?,?,?)',
                    [request.body.fname, request.body.surname, request.body.password],
                    function (err) {
                        if (err) {
                            console.log("Error: " + err);
                        }
                        else {
                            console.log('POST request received at/users');
                            response.status(200).redirect('../main/index.html');
                        }
                    });
            }
            else {
                console.log('User authenticated');
                response.redirect('../main/index.html');

            };
        });

});
```

- I add the POST request to save the messages that users want to send:

```
//message

app.post('/messagerot13', function (request, response) {
    var post = request.body;
    db.run("UPDATE users SET messagerot13 = ? WHERE fname = ? AND surname = ?",
        [request.body.messagerot13,request.body.fname,request.body.surname],
        function (err) {
            if (err) {
                console.log("Error: " + err);
            }
            else {
                console.log('POST request /users updated');
                response.send('Message is being sent');
            }
        });
});

app.post('/messageb64', function (request, response) {
    var post = request.body;
    db.run("UPDATE users SET messageb64 = ? WHERE fname = ? AND surname = ?",
        [request.body.messageb64, request.body.fname, request.body.surname],
        function (err) {
            if (err) {
                console.log("Error: " + err);
            }
            else {
                console.log('POST request /users updated');
                response.send('Message is being sent');
            }
        });
});

app.listen(3000, function () {
     console.log("Server is running on port 3000");
});
```

- I add the GET request to send the message from the database to the user side when they enter their details:

```
//routes

app.get('/', function (request, response) {
    response.send('Hello, world');
});


app.get('/receiverot13', function (request, response) {
    console.log('GET request received at /users');
    var post = request.body;

    db.get('SELECT messagerot13 FROM users WHERE fname = ? AND surname = ?',
        [request.body.fname, request.body.surname, request.body.messagerot13], function (row) {
            if (row) {
                response.send(document.getElementById("messagedecoderot13").innerHTML = row);

            }
            else {
                console.log("No message");
                response.status(200).redirect('../messages/index.html');
            }
        });
});

app.get('/receiveb64', function (request, response) {
    console.log('GET request received at /users');
    var post = request.body;
    db.get('SELECT messageb64 FROM users WHERE fname = ? AND surname = ?',
        [request.body.fname, request.body.surname],
        function (row) {
            if (!row) {
                console.log("No message");
                response.status(200).redirect('../messages/index.html');
            }
            else {
                function showMessage(comments) {
                    var commentsSection = document.getElementById("messagereceivedB64");
                    for (var i = 0; i < comments.length; i++) {
                        commentsSection.innerHTML = messageb64[i];
                        response.end();
                    };

                };

            };

        });
});
```

EVALUATION OF IMPLEMENTATION

COMPARISON AGAINST REQUIREMENT

This project is very user-friendly, easy to use and very intuitive platform. The development plan was clear and concise.

In terms of functionality, users can perfectly navigate the website and encode and decode their messages.

The login section performs well, and its objective is either authenticate users or add new users to the system. This is essential to move on the next step: sending the messages.

The cyphers sections are aimed to store the messages- already encoded in the client side- in the database. The messages must be stored along with the recipient to be ready to be displayed in the client side when recipients enter their details.

The messages webpage has an effective strategy to display the messages. The GET request is being set and text area's ready to display the messages that users would receive from the database when entering their details. Server works but it doesn't display messages for some reason, however, the decoding section applied in the client side is still available for users.

POSSIBLE IMPROVEMENTS

Besides the issue on the messages section, the login/sign up webpage could have been implemented with a more user-friendly design. The functionality of the section is good, and it guarantees the good performance of the rest of the platform, but I miss some message from the server to let user know whether they have been given access to the app or they have signed up.

PERSONAL EVALUATION

The learning process was long and challenging due to the lack of time. There was needed a lot of documentation to understand the basics of Node JS and how this programme interacts with the others, the browsers… After Node JS basics, there was also the Express framework and the numerous modules available. That involved quite a lot more reading again to see how to install them into the app and the benefits they bring to the development of the project.

During the development of this platform I have encountered several issues related to programmes installation, JavaScript coding, POST and GET requests from "forms" in HTML, attributes crashes like "name" and "id" in the same element, database syntax combined with JavaScript coding.

I approach every of the issues encountered by researching many different resources like online tutorials in Youtube, online documentation, recommended books and articles, public web development platforms, …

In summary, this app involved a lot of work including research, development process plan, coding and testing. The result is not perfect, but I am very satisfied with my progress because I have learned a lot from it.

REFERENCES

Books and articles:

- Pro Express.js

Online documentation:

- W3school.com
- Doc.npmjs.com
-

Youtube tutorials:

- Build your first web application using HTML, CSS, JavaScript, Node JS and SQLite: (www.youtube.com/watch?v=RaNpujNOAHg&list=WL&index=3&t=0s)
- Inserting new data into SQLite, PetsApp v2 done (Node.js+Express Part 11) (www.youtube.com/watch?v=Dzy_32qE_dI&list=WL&index=6&t=35s)

Online development portals:

- Stackoverflow.com
- Github.com