Edinburgh Napier UNIVERSITY

Coursework Report

David Pires Lazaro
40406091@live.napier.ac.uk
Edinburgh Napier University - Web Technologies (SET08101)

1 Introduction

The aim of this assignment is to extend the cypher site from the first coursework to design and implement a coded messaging platform. The solution will include server and client elements. The client element will provide a web interface to enable your users to sign-up for an account, to write plain text messages, to encode those messages (using the cyphers previously developed, to send those messages to other users, to access messages sent to them, and to decipher any received messages. The server element will persist data about messages and users and will support the client-side functionality.

For this project I used both local storage and the filesystem package to act as my database, all built upon an Express HTML template to avoid having to create every file from scratch,and using HTTP requests to interact between client-side and server-side interfaces.

2 Software Design

I started off the project by writing down all the requirements for the assignment on a piece of paper so I could then start ticking off tasks as I would complete them. The requirements for this project are to improve on the site from the first coursework and implement:
-A Sign up/Log in system
-The ability to send text messages to other users
-To encode those messages
-And finally to have a database system set up to deal with the messaging and the login systems.

Right from the start, using the filesystem node package for the sign up/log in systems seems the most straightforward as it allows us to work with the file system on our computers, and thus read, create and edit files to acts as our own database. It consisted of validating the user data, followed by sending a POST request to the server that then stores the username and password onto a JSON file.
Using that same JSON file, the log in system would just need to read the database and with a simple for each loop iterate between all the array objects and pick out the matching username and password pair. LocalStorage will also hold the current logged in user, so the messaging system can display the correct messages.

The messaging, as we'll see later, required an npm module called node-localstorage to use localStorage at the nodejs server side. Each message will be pushed onto an array that belongs to each specific username, and we've named each user's key "username + _messages" which we can then filter to output each user's messages depending on their username.

3 Implementation

As for the Client-Side of the website, again, a simple design was the preferred approach. Right at the start, you are greeted by a sign up page, which will check against the file system if your username is unique. When you log in, you are going to be redirected to /cyphers, the main page of the website. There you will find a brief introduction as to what you can do on the website and then at the bottom you will be faced with an UI that will allow you to write messages, encode them and send them to other users.

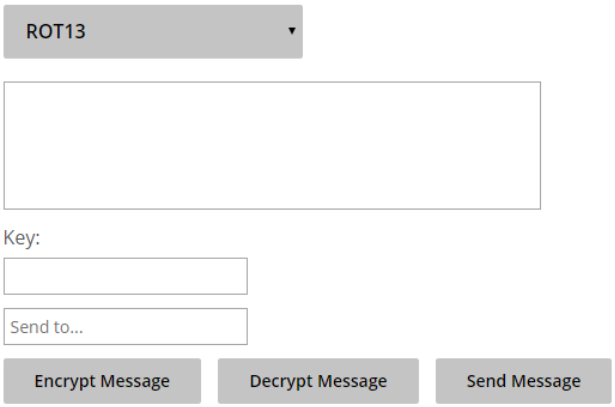

Figure 1: **UI** - Simple interface

At the bottom of the page you will find the Message board. There you will see all the messages that have been sent to you in the form of an unordered list.

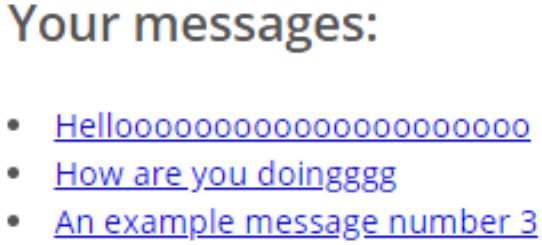

Figure 2: **Messages** - How users see messages

4 Critical Evaluation

The project turned out successful and without many detours from the main path that was originally planned. The body-parser package was one feature I didn't plan on using but turns out it does a great job at parsing the data from an HTTP request into a format which you can easily extract relevant information.

Listing 1: Body-parser

```
function postData(url = '', data = {}) {
    return fetch(url, {
        method: "POST",
        mode: "cors",
        cache: "no-cache",
        credentials: "same-origin",
        headers: {"Content-Type": "application/json",},
        redirect: "follow",
        referrer: "no-referrer",
        body: JSON.stringify(data),
    });
}

```

Struggled with connecting the server-side with the client-side, and then connecting them to the database but after hours of searching, I managed to find a zellwk.com[1] which explained the reasoning behind a CRUD application and how body-parser worked while also going through sending HTTP requests to the back-end.

Possible improvements would include a password encrypting function using bcrypt and a user authentication service to allow for admin privileges around the website such as maybe deleting messages.

5 Personal Evaluation

By far, the hardest part was the interaction between client-side and server-side which had me research for hours in hopes of finding a viable solution to my problems. Read a lot on CRUD and HTTP requests and how the server deals with PUT/GET/POST/CREATE requests.

References

[1] Z. W.K., "Building a simple crud application with express and mongodb," Jan 2016.