WEB TECHNOLOGIES

Callum Lawson
40400640

Table of Contents

Introduction

The second coursework is to extend the first piece and implement a coded messaging platform.

The requirements are:

- Allow users to sign-up for an account
- Write plain text messages
- Encode the messages
- Send the messages to other users
- Access messages sent
- Decipher any retrieved messages

My website uses the same cipher suite as before but uses an emailing system to send emails to users with encoded messages. My site has a login/register page, a profile page, a cipher page and an e-mailing page. A user can register for an account, login and send a ciphered email, the email can be de-ciphered in the cipher page. My website features 4 pages that the user can navigate once logged in. There is a page to send an encoded message via email and a page to decode the message. User information and sent e-mails are stored in a MongoDB database. For this coursework there was quite a lot of background reading into what middleware I would need and certain pages. Mainly I used W3Schools, Stack Overflow, labs and lecture notes.

Software Design

I have a rough idea of the requirements I will need for this coursework although I will probably need to re-visit this section later and add more as I become more aware of what exactly I need.

There will need to be:

- At least 2 or 3 pages
- Login/Registration page, possibly separate
- Profile page
- My original cipher page
- A page to send encoded emails to users
- A database to store user info/emails

I will need to begin by deciding which database to use, this will probably be mongoDB either cloud or localhost as this was easy to use in the labs. I will need a new colour scheme for my pages as the cipher page is way too bright to host a register/login page. I will encrypt the user's password probably using bcrypt, so it salts it and stores it in the database. I may also decide to store user cookies for the user that is logged in. I will need to protect the profile page so only when a user logs in can they access it.

The profile page will have:

- Links to the other pages
- Information about current user

The email page will have

- A link to the profile page
- A text area for messages

- A submit button to send the e-mail
- An Encryption button
- Drop down menu to select cipher
- Shift selector for Caesar cipher

The login page will have

- 2 submit buttons for login/registration
- Login form
- Registration Form

(See Appendix A for wireframes)

Implementation

For this coursework I used the following middleware:

- Express - Framework
- Nodemailer - To send encoded messages to other users
- bodyParser – Parse requests
- ejs - Templating
- session – Save cookies
- mongoose – Create schemas to store user information
- bcrypt – hashing and salting user passwords in database
- path – utilities for file and directory paths
- nodemon – avoid having to restart the server each time

My website uses 4 pages

Login/Registration | Profile | E-mail | Cipher Page

The login/registration page stores user information in a mongoDB database using mongoose that salts the password. There is some protection on the login/registration information a user can enter, for instance the same email or username cannot be used twice, passwords must match, and wrong information cannot be entered. Once a user is on the profile page their username and email is displayed and there are links to logout(which takes the user back to the login/register page), cipher page and email page. The email page allows a user to send an encoded message using either Caesar,Base-64 or Morse. The cipher page is from Coursework 1 with updated CSS and HTML to make the page's colour scheme in line with the other pages. This project required a lot of JavaScript and some new HTML and CSS, I avoided touching things like jQuery or Ajax since I wasn't sure if they were was allowed.

The structure of my project is:

Routes/Router.js – Where all the routes are defined includes some middleware functions and contains code for emailing.

App.js – My main file where the server is running on port 5000, also handles errors, contains most of my middleware functions and connects to the database.

Package.json – A list of my dependencies

Views – Root contains 3 HTML and one ejs file.

Views/Css – Contains the CSS for my 4 pages

Views/Js – Contains 3 javascript files

Models – Contains a user schema and email schema to store information in database. Uses bcrypt

(See Appendix B for screenshots)

Critical Evaluation

This coursework proved substantially more difficult than the first one, particularly when we were only allowed to use middleware from the labs. Using nodemailer was easy to send emails although I don't like having to go into the code to change which user to send an email to, ideally, I would have liked a system to choose on the e-mail form which user to send an email to.

In terms of the requirements I think I met most of them however:

- Access messages sent
- Decipher any retrieved messages

Are a bit of a disappointment for me, its very easy to send emails in node but retrieving them is a lot more difficult. I had some success with middleware like Imap and mail-listener to download emails, but the emails were always garbled in some way and wasn't really an efficient way because it ran of the same server meaning emails were downloaded as soon as I ran the server and I didn't want to run two servers separately or run a SMTP server. In the end I chose not to implement an email retrieval system so I feel as if I missed out two of the requirements which is annoying, in hindsight If I knew how strange it is to try and fetch emails I may have went for a chatroom or private messaging system instead.

In terms of improvements I would have liked a better profile page that maybe allowed users to manage their accounts or reset their passwords. I decided to avoid using passport in this coursework as it seemed to do a lot of work for you with few lines of code so I wouldn't really learn much. I did end up using mongoose and I decided it was worth it because of how easy it was to set up schemas and store information in a database. I tried to limit the number of middleware functions I was using to see what was possible in standard node and express. I also would have liked a better login system, for example when a user registers an account it doesn't check if the email exists or not.

Personal Evaluation

This coursework proved quite challenging for me particularly in the beginning when I could only use node middleware from the labs, I was struggling a lot to get something going, with the freedom to choose my packages I was able to build something I feel is good for a first attempt. Progress was quite slow for the first few weeks, ideally, I would have liked to have used any node package from the beginning which probably would have led to a better website overall. I was disappointed I didn't quite meet all the requirements but that really came down to my choice of messaging system, early on I didn't take in to account how I would need to receive emails and once I had gotten a good part into my website I didn't want to change my methodology. Unlike the last coursework I felt a bit rushed once I realised how big the coursework was. Building a registration and login system early on was challenging as it was my first time working with databased as well, thankfully mongoDB is very easy to work with. I feel like the CSS, JavaScript and HTML is better this time after having more experience with it. I'm disappointed I didn't manage to fetch/download emails to de-crypt them on my site. Overall, I think my site is a reasonable attempt at balancing not relying on too many middleware functions and seeing what is capable in standard JavaScript and node express, but also utilising those functions where I needed them.

References

Creating Schemas. (n.d.). Retrieved April 2, 2019, from elegant mongodb object modeling for node.js: https://mongoosejs.com/docs/guide.html

Getting Started. (n.d.). Retrieved April 2, 2019, from elegant mongodb object modeling for node.js: https://mongoosejs.com/

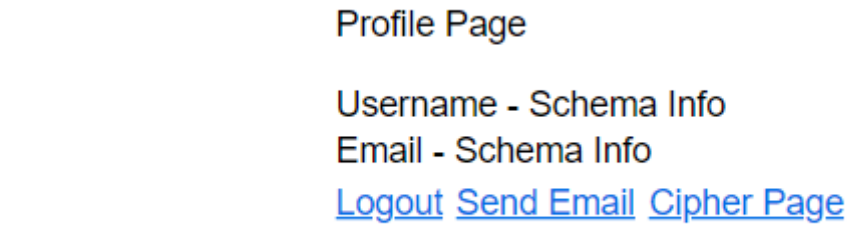
Jensen, B. (n.d.). *All You Ever Wanted to Know About Sessions In Node.js*. Retrieved April 5, 2019, from Stormpath: https://stormpath.com/blog/everything-you-ever-wanted-to-know-about-node-dot-js-sessions

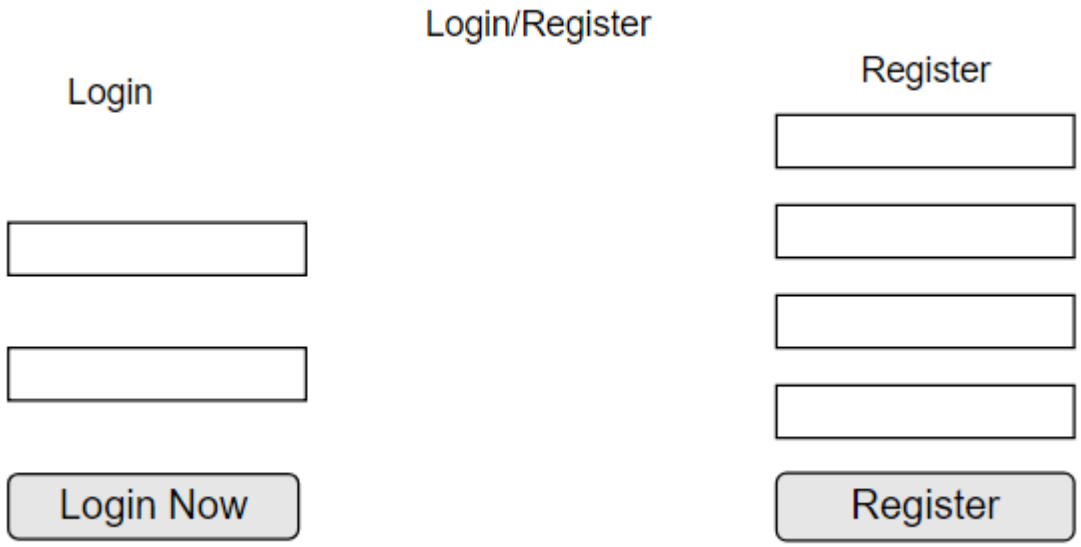
LaViska, C. (n.d.). *Hashing Passwords with Node.js and Bcrypt*. Retrieved April 3, 2019, from abeautifulsite: https://www.abeautifulsite.net/hashing-passwords-with-nodejs-and-bcrypt

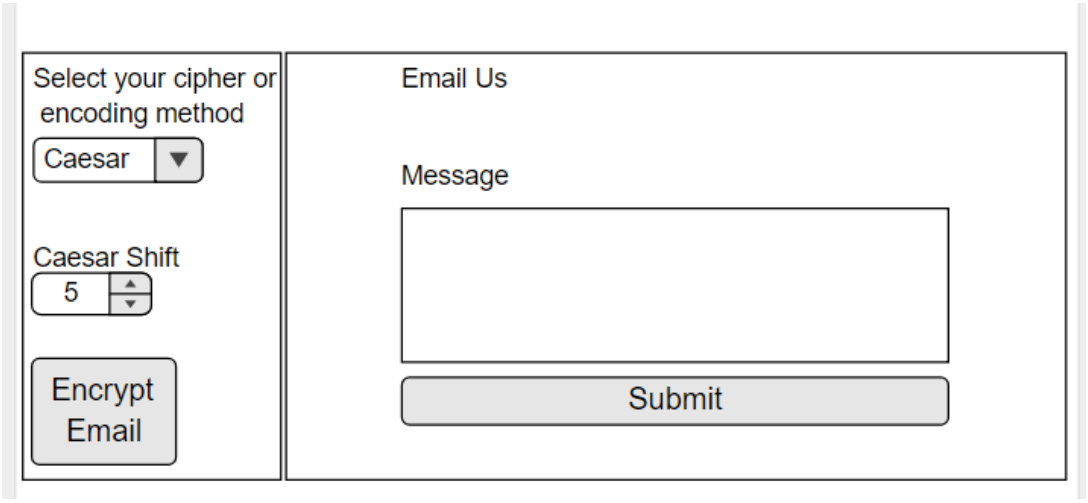
Reinman, A. (n.d.). *Nodemailer*. Retrieved April 1, 2019, from Nodemailer: https://nodemailer.com/about/

Wilson, D. (n.d.). *body-parser*. Retrieved April 6, 2019, from npm: https://www.npmjs.com/package/body-parser

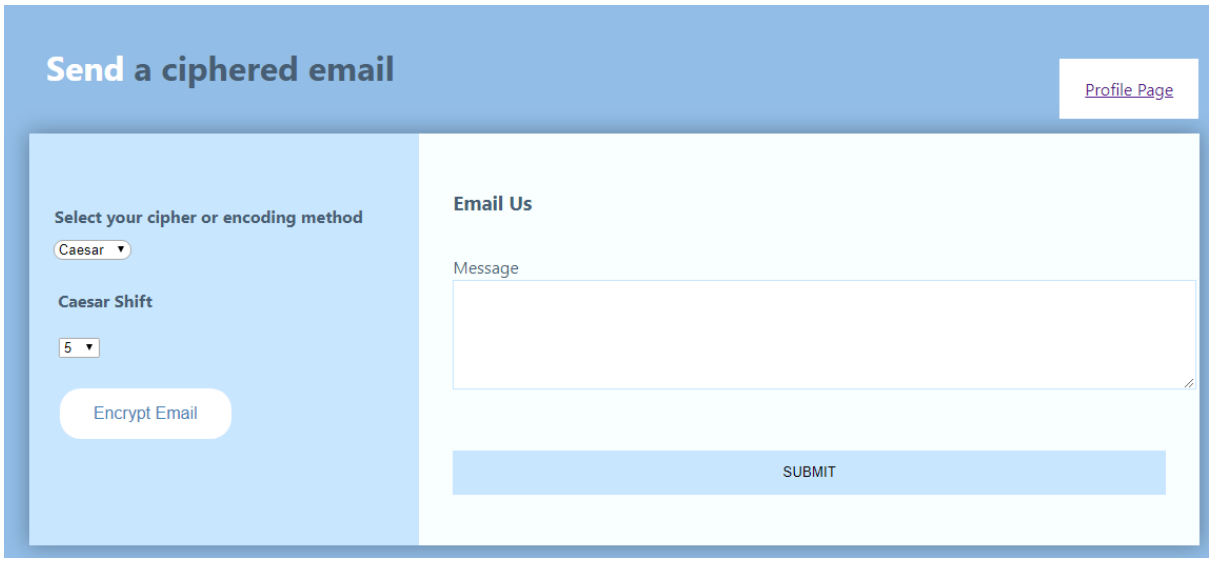
Appendix A - Wireframes

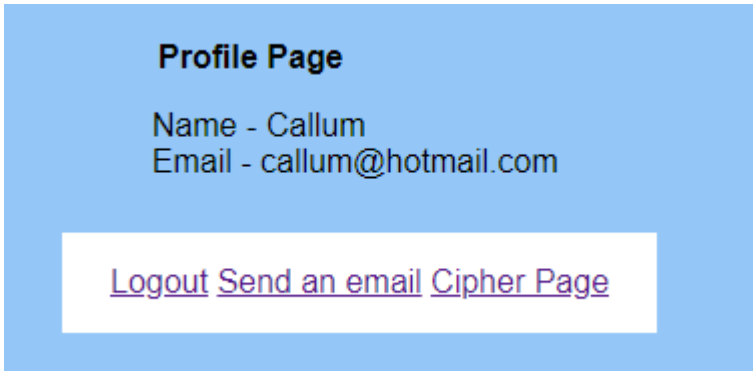


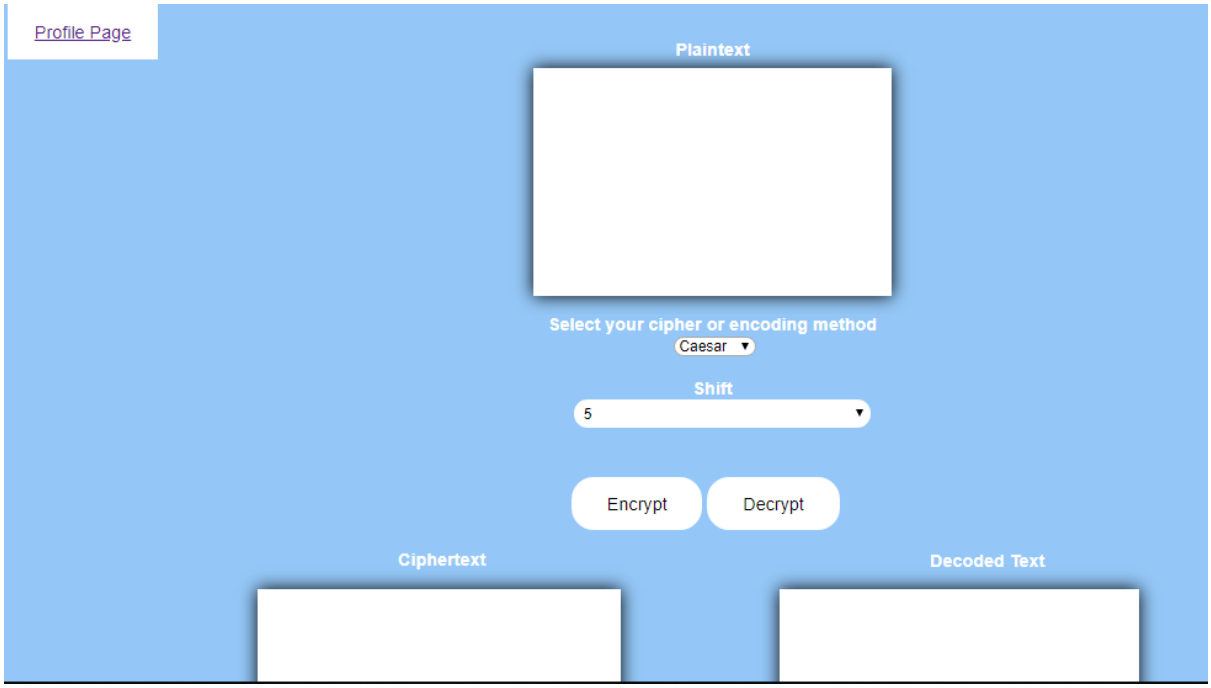




Appendix B – Implementation Screenshots








User login and Registration
Login
Register
calle590@hotmail.com
LOGIN NOW
E-mail
Username
Password
Confirm Password
REGISTER