Cipher Web Technologies Project

Using Caesar and Vigenere’s Ciphers, NGINX and NODE.js

Author

Ronan Coyne

40417360@live.napier.ac.uk

Date

06 March 2019

Insert relevant location disclaimers

# Introduction:

The problem I was presented with was to design a web page using html, JavaScript and CSS to implement that enables a user to encode text messages using at least two different classical cyphers or encoding schemes.

My solution was to implement the Caesar Cipher which is also known as Rot13 cipher, this is a simple encoding mechanism that simply shifts the letter by the chosen key in the alphabet/character set. I chose this as it is relatively simple and had experience implementing it in python. My second cipher was the Vigenere cipher, this was chose as I am familiar with how it works and thought it would be an interesting implementation in JavaScript to test my knowledge of loops and input validation while keeping the algorithm similarly easy to the Caesar Cipher.

In addition to this I had to provide a usable UI with a nice aesthetic using CSS and html.

I implemented this using node.js and Nginx in the back-end while pulling in Jquery to send requests from the front end.

# Software Design:

My approach was to begin by figuring out the

After doing this preparation I felt ready to build my webpage.

# Implementation:

My implementation uses a homepage (index.html) which allows you login or register:

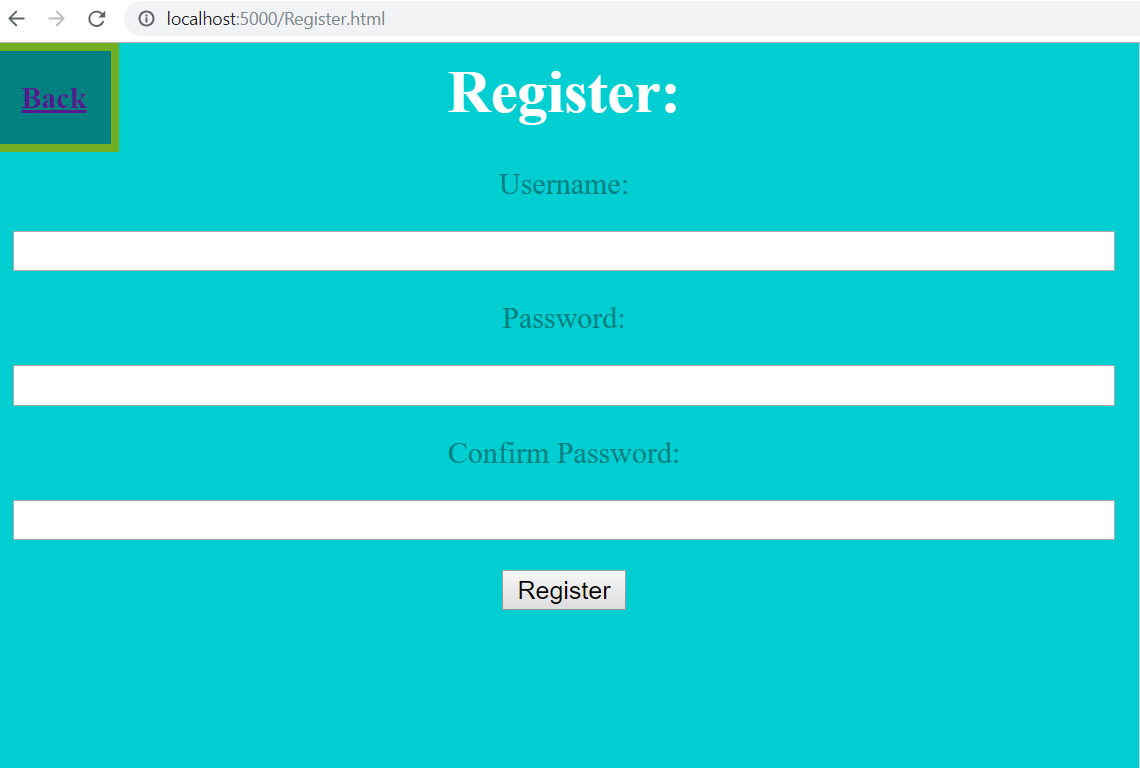
## Homepage:

### 

## Login Page:

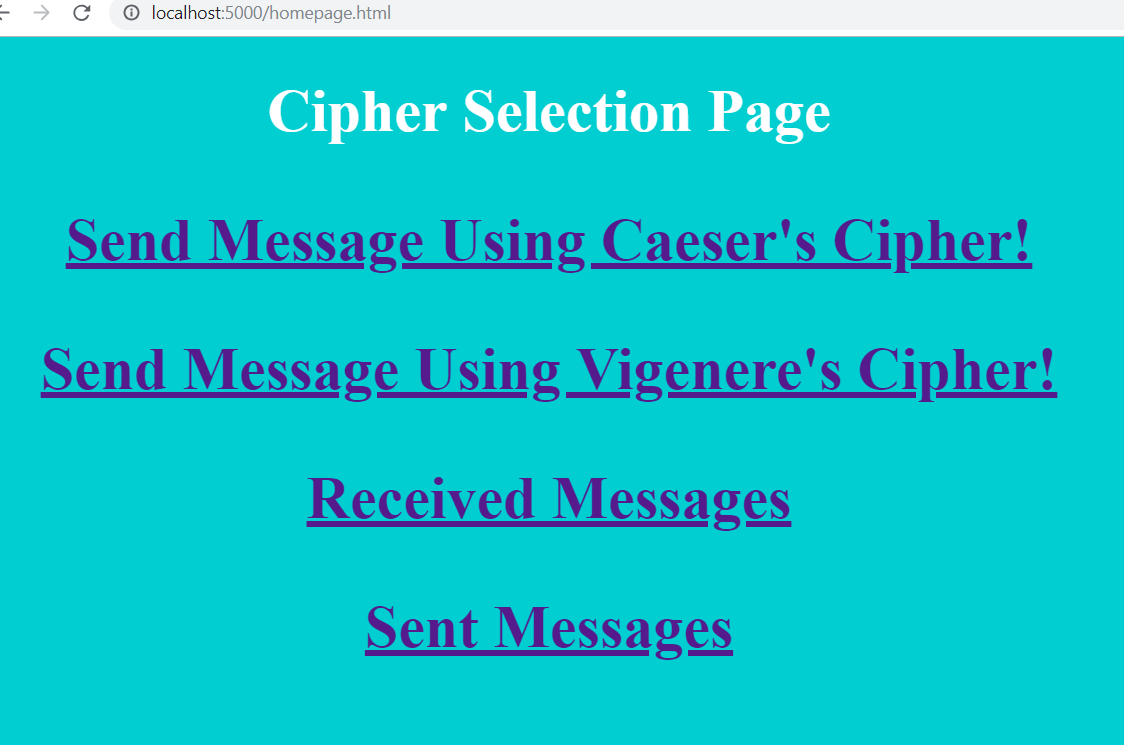
### 

## Register Page:



After going through one of the methods either logging in or registering you are brought to here which allows you to select what you want to do:

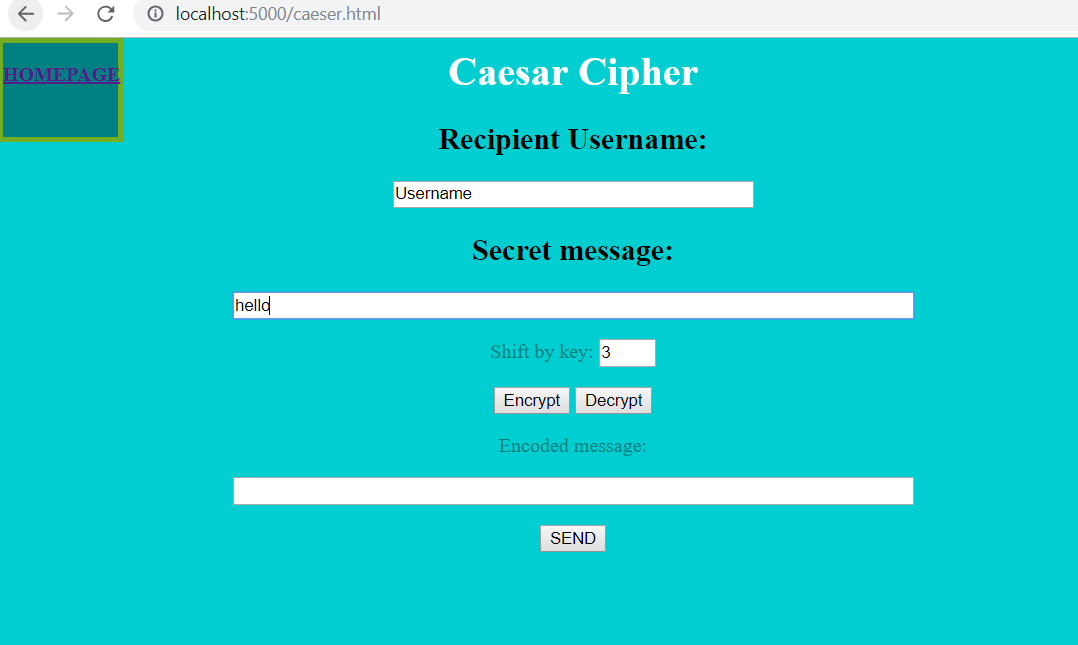
### Cipher Selection Page:



Upon clicking on one of the cipher links it brings you to either of these pages:

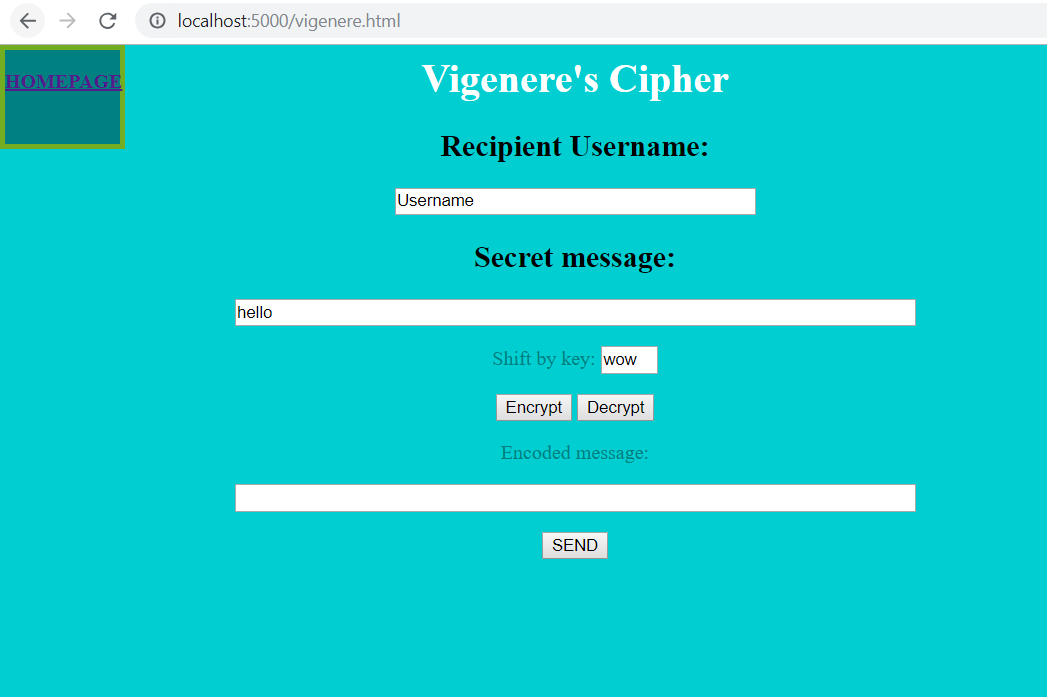
### Caesar Cipher Page:

This page allows you select a user, encrypt your message using the Caesar cipher with your own key between 0-35 as it includes numbers and send them the Encrypted message.



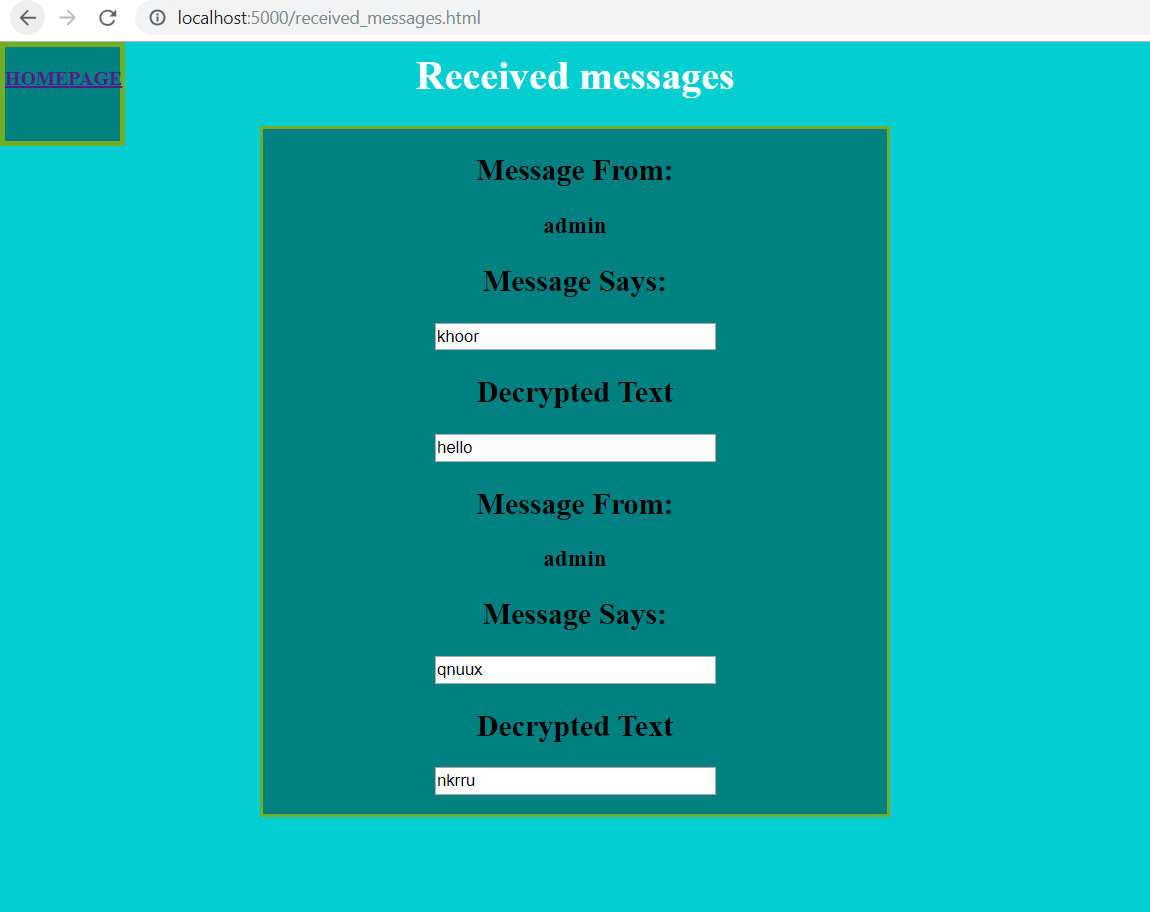
### Vigenere Cipher Page:

This page also allows you to send an encrypted message to another user but this time using Vigenere’s Cipher with a string key of any length.



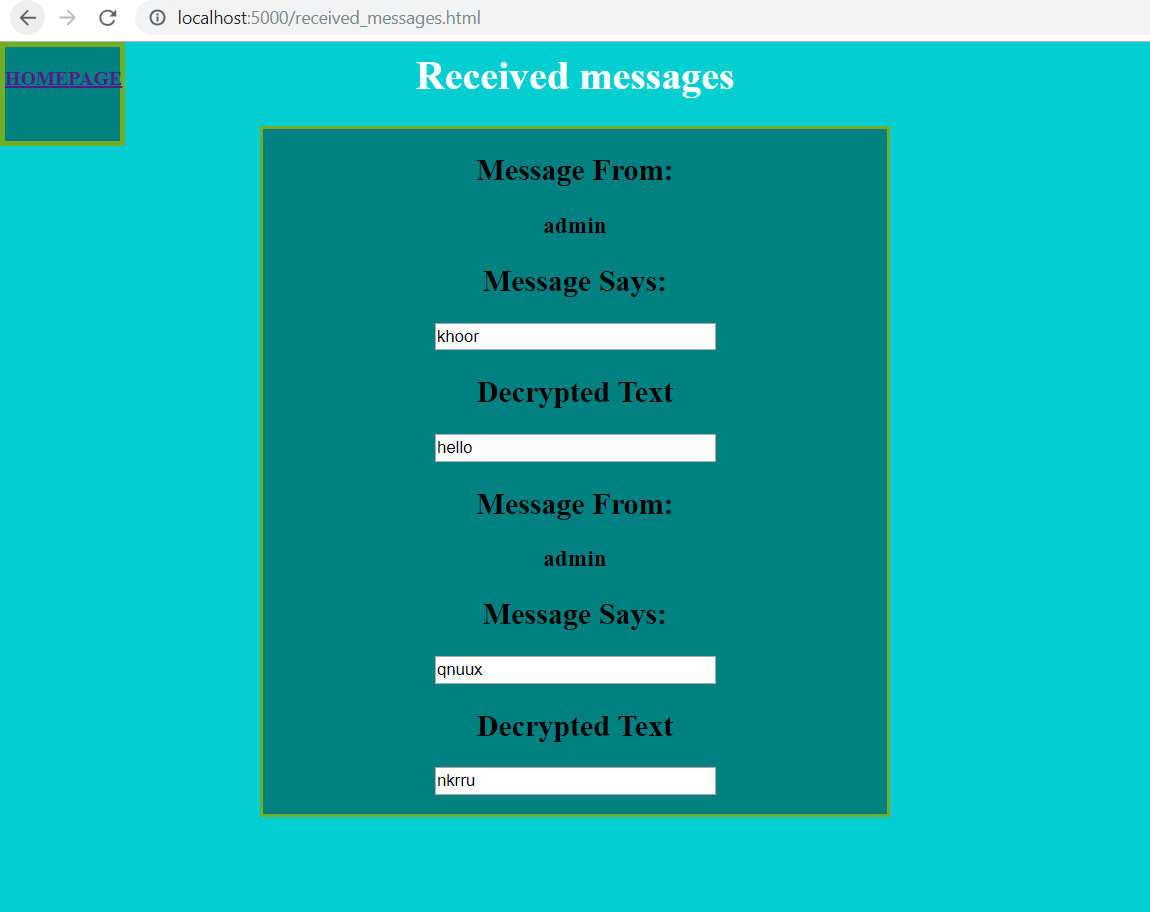
### Received Messages page:

In this page you can view the messages you have been sent as well as their unencrypted versions. It will dynamically expand as you get more messages.



### Semt Messages page:

In this page you can view the messages you have sent as well as their unencrypted versions. It will dynamically expand as you get more messages.



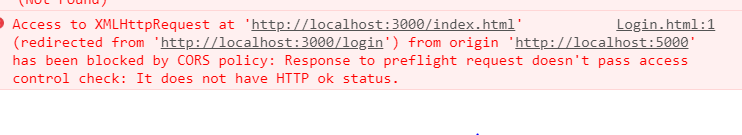
I used these webpages in conjunction with external stylesheets and external JavaScript files to create a modular solution and reduce re-use of code. The css was kept simple due to issues with the CORS Header that still haven’t been solved. Leading to the apps login redirect

# Evaluation:

I believe my implementation fulfils all the requirements of the coursework.

1. *an area where users can login / sign up.*

I do not fully cover this as I have spent days trying to solve a problem with the CORS header on Nginx and no matter what I try and how many different methods / hacks I try I cannot get this to do a cross origin request. I have tried reverse proxying the server through Nginx, explicitly allowing all origins to connect in the conf file, attempted work-arounds using JSONp but nothing I try seems to work. The app correctly inserts the users in the database and can query them correctly but when trying to direct the user to the next page it throws this error.



Now so far in my development career no error has beaten my powers of googling however attached is a screenshot of my google history searches related purely to this. It’s in a separate file as it is ridiculously long and would ruin the structure of the report. The app works correctly apart from this and all the data from these functions is passed correctly so maybe hosted on a different server would work as nginx does not seem to apply my changes to the headers. But I did not realise how big an obstacle this would be as it seemed quite a simple fix but after days of debugging it has won. All the functions data interactions work correctly and all functions return correct results ( query for username and password for login function returns correct row as object ready to be used, register correctly inserts users into database) it is just unable to interact with the static pages.

1. *An area where you can encode messages and send them to other users,*

*The cipher pages cover this as they allow you to just encode or send to*

*Other users if you wish.*

1. *a way to select between different cyphers.*

*The home / selection page covers this.*

1. *Area to view messages you have been sent and decrypt them,*

*The view received messages covers this as well as the sent messages page which allows to see your previously sent messages*

1. *The server element will persist data about messages and users and will support the client-side functionality.*

*This is done through the SQLlite3 Database which will persist the messages and user data.*

### Possible improvements:

Barring the missed requirement outlined above I believe I meet the requirements quite well however would like to improve it by fixing this issue and by adding more cipher pages as well as improving my CSS skills as they are quite lacking. I believe focusing more on the end user experience rather than cool functionality would be a great development step for me in my software development skills.

# Personal Evaluation:

I believe this was a great project for my personal development as it taught me that small errors can have a huge impact on your final product and not to think that a few google workarounds can fix anything as well as teaching me a lot about using node.js to deliver interactive web applications.