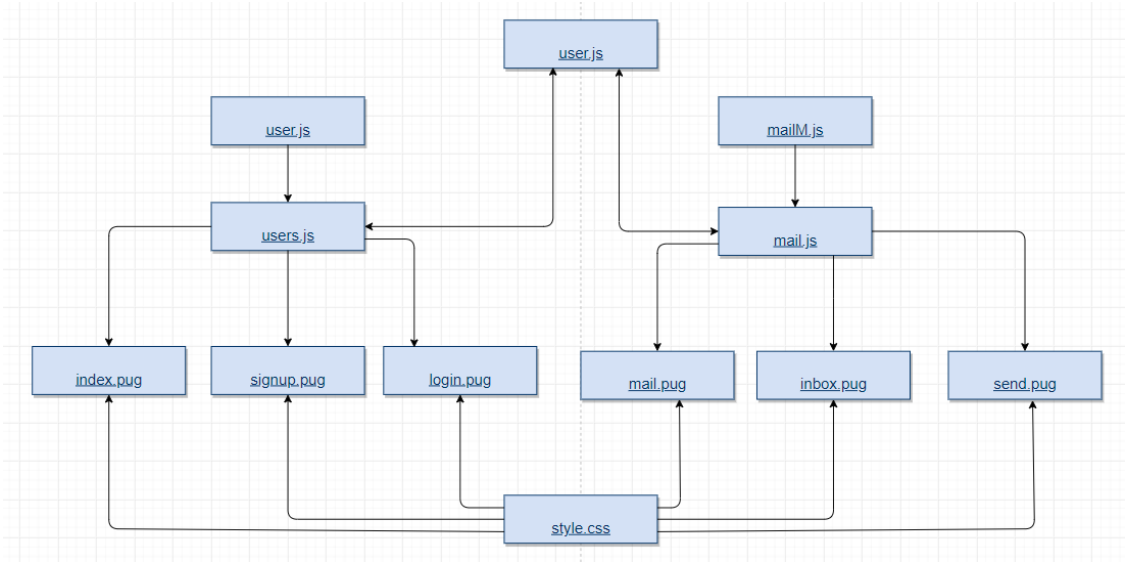
Web Tech Assignment - Cipher Messenger Website

Introduction

The aim of this assessment was to design and implement a coded messaging platform using skills gained from the previous assessment. As part of the requirement, the website includes a sign-up feature which allows users to create their own personal accounts as well as a login feature which persists their data in a MongoDB. In addition, users can send messages to other users of the site (which can then be decoded). To provide security benefits, the website has authorisation, meaning that without signing into an account you cannot access webpages as they require unique ids. To help create this website, a lot of the logic is built from a series of YouTube videos ("Creating a RESTFUL API with Node.js", 2017).

1. Software Design



Graph 1: Graphical representation of the relationship and navigation between the PUG, HTML and JS used to create the Cypher Website.

To create the website PUG, Cascading Style Sheets (CSS) and Javascript (JS) are used. In this case the website is created by a database and is accessed to via the PUG pages. These pug pages interact with the javascript files by getting GET and POST commands (Graph 1).

The javascript files contain all the main functionality of the website. Within them are the GET, POST and DELETE functions.

The PUG files are the ones that create the user interface on the web browser. They can be navigated using buttons and designed in a similar format due to the CSS file.

2. Implementation

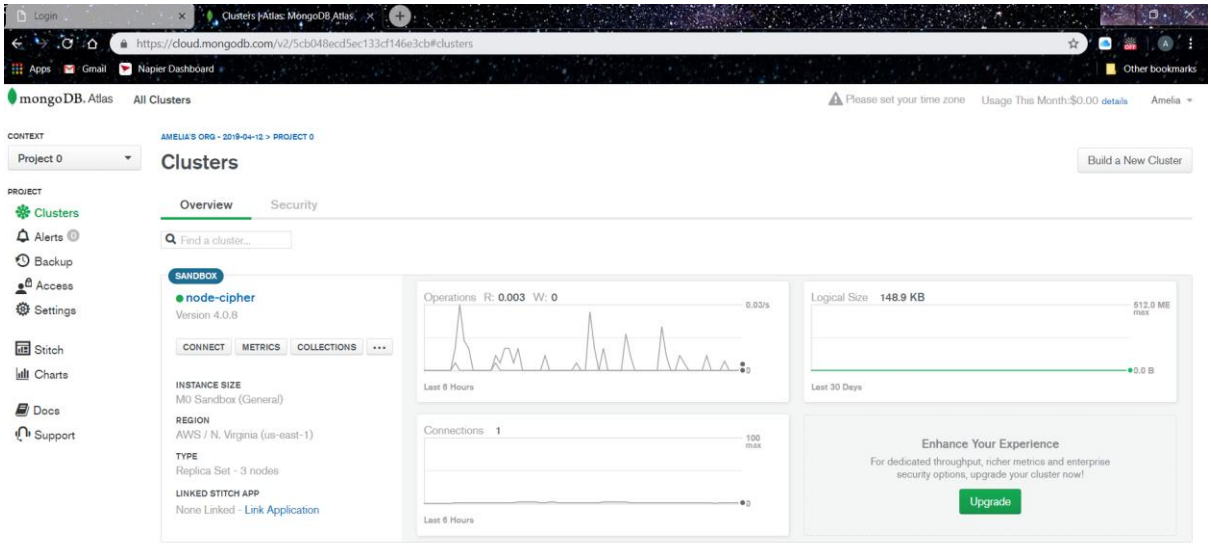


Image 1: Image of the MongoDB Altas created from the website ("MongoDB Cloud Database Solutions", 2019)

MongoDB Atlas is a cloud database service created by the company who made the Mondo DB (local). Instead of requiring the installation a whole mongoDB and run all the commands there, the Atlas is able to deal with the deployment and management on cloud service providers (in this case AWS was used) ("MongoDB Cloud Database Solutions", 2019). To ensure that the website created could connect to the database, an administration role was added, which was hard coded into the app.js.

To connect the MongoDB and the website together, an additional node file called mongoose was installed ("Automattic/mongoose", 2019). Mongoose is an object data modelling library which allows the user to create their own environment for their data as well as providing structures connected to MongoDB servers. It is particularly useful when used alongside JavaScript files and JSON data formatting making it suitable for this assignment ("Object Modeling in Node.js with Mongoose", 2019).

To ensure that users data is protected within the database an additional package called bcrypt.js was installed ("node.bcrypt.js", 2019). Bcrypt is a password hashing function which is resilient against brute force attacks.

2.2. Website Design

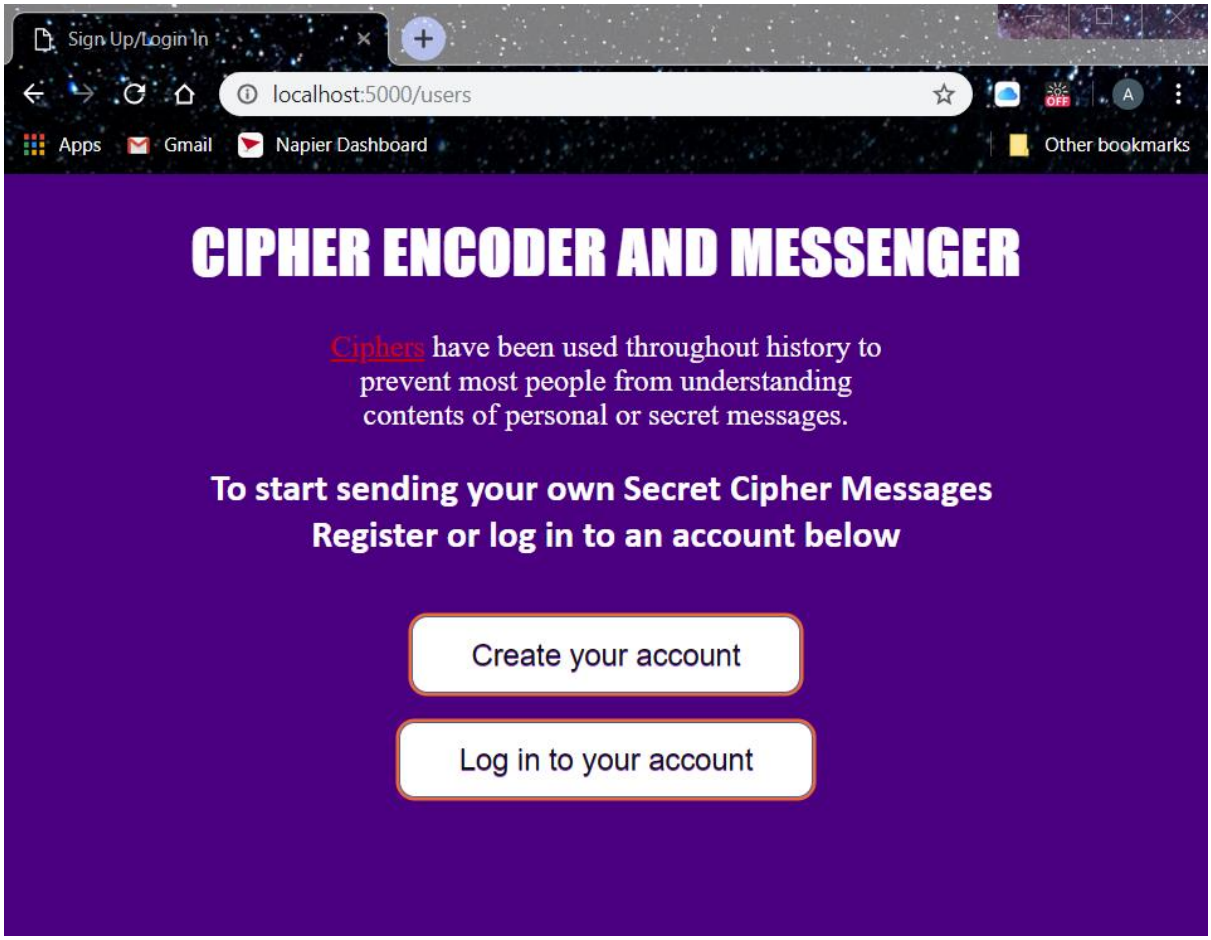


Image 1: Home page of the Website (using url: localhost:5000/users)

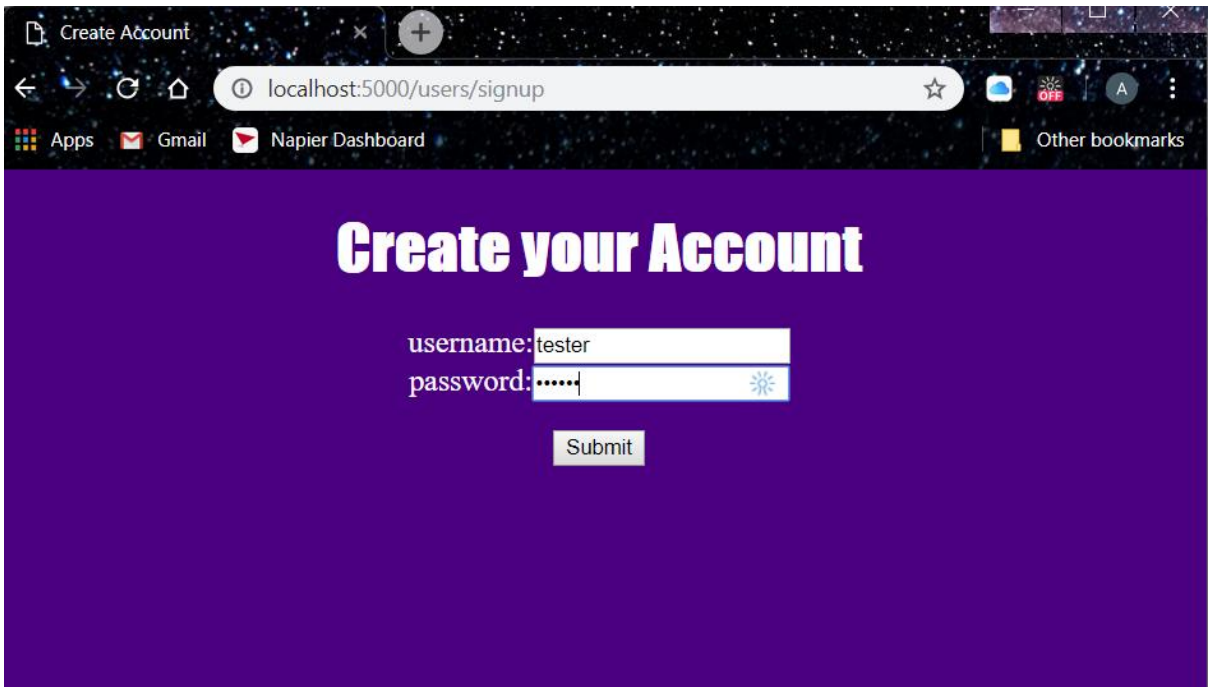


Image 2: Sign up page to create an account for the website (using url:localhost:5000/users/signup)

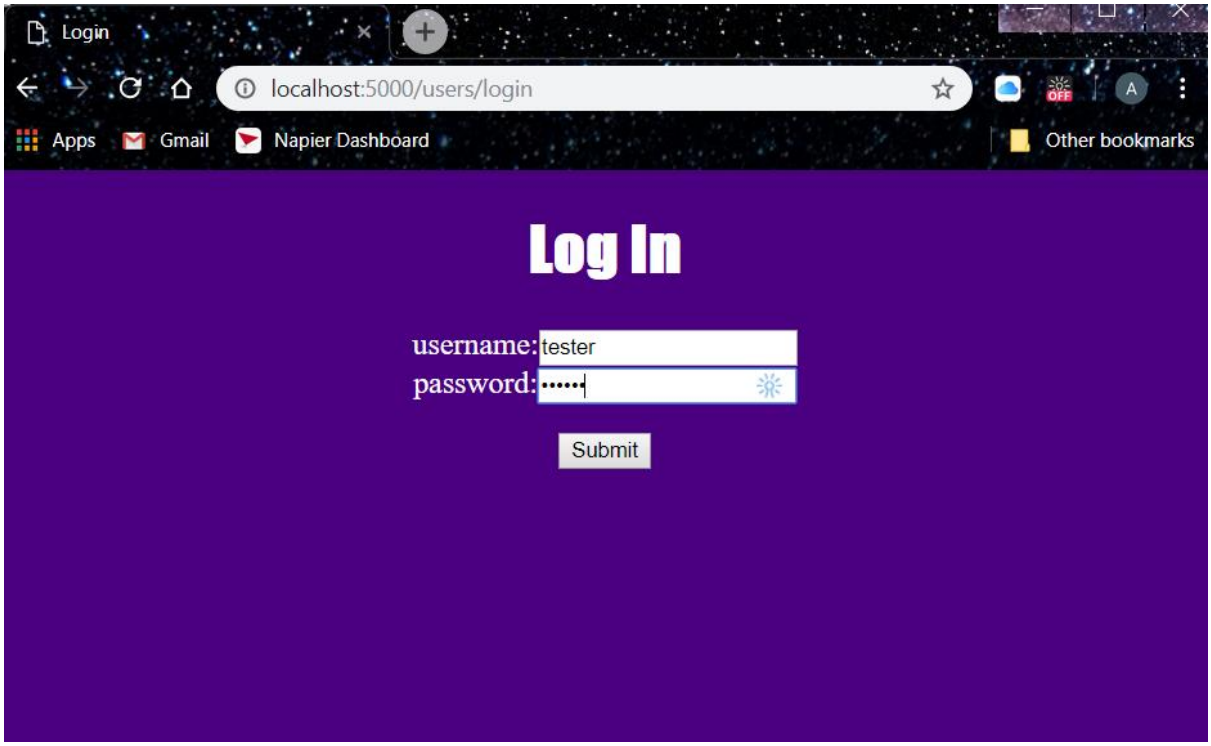


Image 3: Log In page to sign into a create account (using [url: localhost:5000/users/login](localhost:5000/users/login))

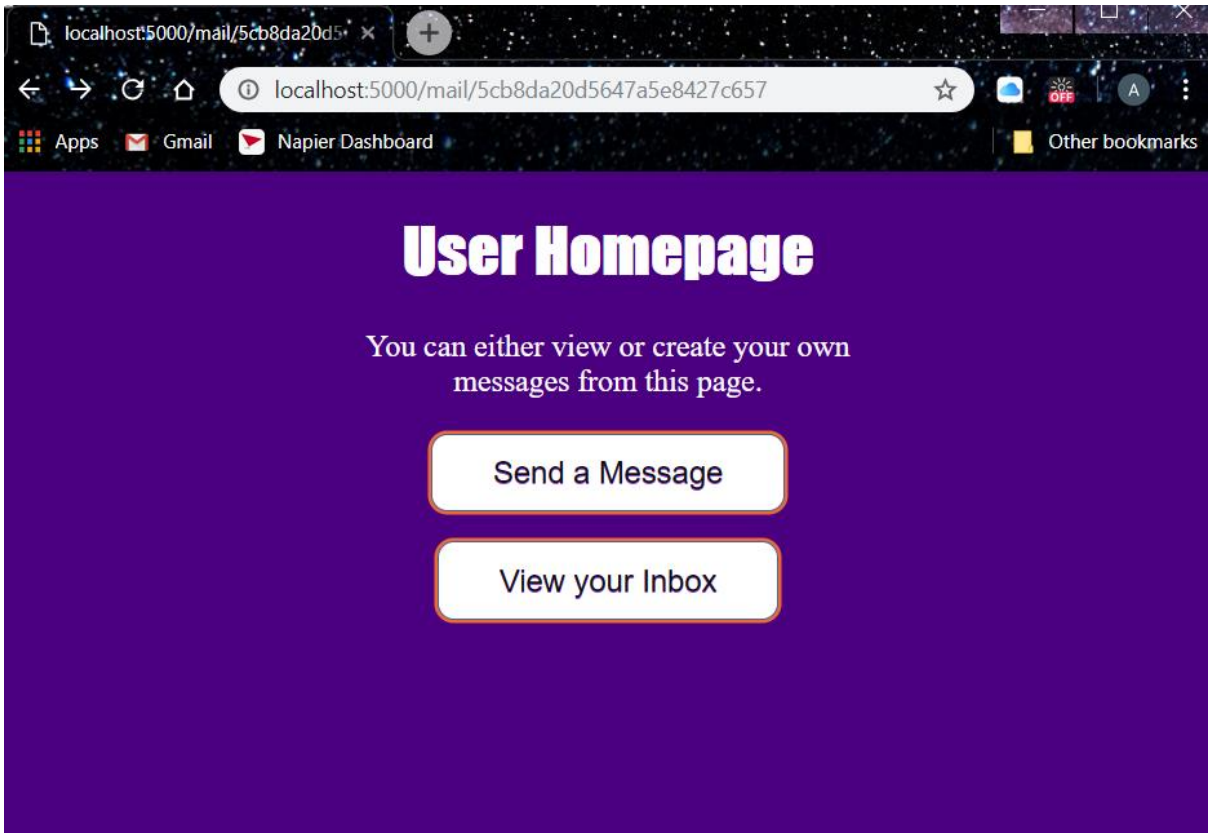


Image 4: User Homepage (using url localhost:5000/mail/(unique id))

3. Critical Evaluation

Overall, I believe that my website is functional and is easy for a user to navigate. A user is able to create an account and it is persisted through the use of a mongoDB meaning that they would be able to access it multiple time. Although it does not meet all the requirements (i.e. allowing the users to send encoded messages), the basic functionality of a messenger website has been created.

Security measures to prevent someone from using a brute force technique to access users' details was to ensure that there was no specific error message given out that could help aid any attack. Instead, if an error presents itself the page will refresh, or will return the user back to the homepage. Additionally, to protect the users' data the brcypt hash function was used so that it could not be accessed by anyone outwith and within the server. This was also enhanced by using a "type=password" method in the pug files to prevent others from viewing the users inputting their passwords.

To conclude, the creation of this website has perhaps been one of the most challenging aspects of the module. On reflection, further background reading would have been helpful as I struggled with many of the connections between the javascript files and the PUG files, particularly with the connection of the users to the message functions.

3.1. Improvements

Improvements made to the website would to be add a cipher function to allow users to encode and decode test to one another. There should also be a security measure to ensure that only the people receiving or sending messages should have access to them. If possible, I would have implemented my check-auth.js function which would have been used to check a function, specific to each user, so that they could only access their own personal data. Therefore, meaning that it would allow for appropriate authorisation of the website. This would have also been a security improvement. Also, I believe the website could have benefited from a patch method specifically in the user data, allowing them to update their user names and pass words.

References

Automattic/mongoose. (2019). Retrieved from https://github.com/Automattic/mongoose

Creating a REST API with Node.js. (2017). Retrieved from https://www.youtube.com/watch?v=0oXYLzuucwE&list=PL55RiY5tL51q4D-B63KBnygU6opNPFk_q&index=1

MongoDB Cloud Database Solutions. (2019). Retrieved from https://cloud.mongodb.com/

node.bcrypt.js. (2019). Retrieved from https://github.com/kelektiv/node.bcrypt.js/

Object Modeling in Node.js with Mongoose. (2019). Retrieved from https://devcenter.heroku.com/articles/nodejs-mongoose