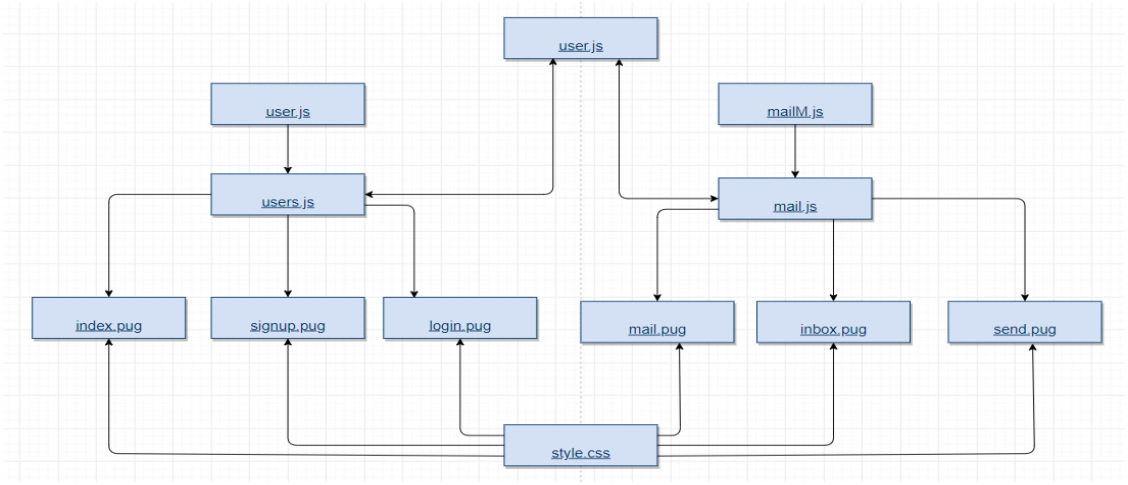
Web Tech Assignment - Cipher Messenger Website

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 must have 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 around research 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 files, a Cascading Style Sheets (CSS) and Javascript (JS)files were used. The website is created by the Javascript files which interacts with the PUG files to create the pages(Graph 1).

The javascript files contain all the main functionality of the website. Within them are the GET and POST requests. The GET methods are used when the user is requested data from a specified source. In the case of this website it would be the user signup, login and mail data. The POST method is used to send data to create a new resource, such as a new user or a new mail ("HTTP Methods GET vs POST", 2019).

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 in the style folder. The CSS file is inserted using an internal style sheet contained within the “head” of the pug (Olsson, 2014).

2. Implementation

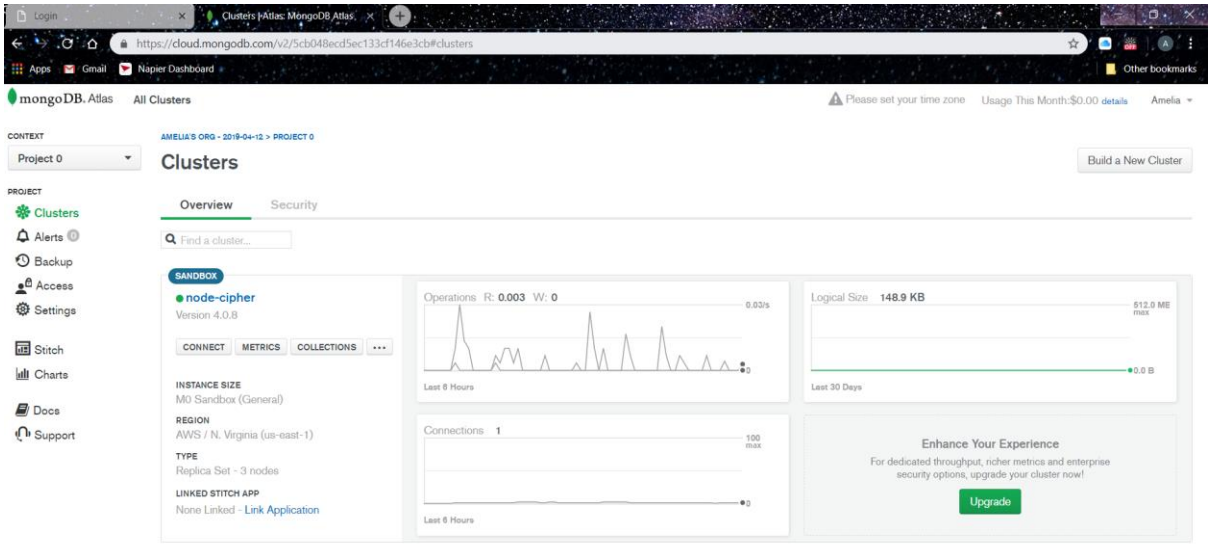


Image 1: Image of the MongoDB Altas created for the website ("MongoDB Cloud Database Solutions", 2019)

MongoDB Atlas is a cloud database service created by the company who made the MondoDB (local). Instead of requiring the installation of a whole MongoDB, the Atlas can deal with the deployment and management on cloud service providers (in this case AWS was used) ("MongoDB Cloud Database Solutions", 2019). A database is an effective way of storing information as it gives a solid-persistence strategy (Syed, 2014). So, it felt appropriate to add a database to store the user information and persist messages. For 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.js 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 Modelling in Node.js with Mongoose", 2019). Mongoose schemas were created in the user.js and mailM.js (in the models folder) which were used to format the user and mail data.

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. This is because a potential attacker would have to test a lot more keys to access a user's account.

In addition, to help develop the website, nodemon reload was installed. Nodemon automatically restarts and runs code edited and saves, meaning that you do not have to continually stop and restart the main app.js ("remy/nodemon", 2019).

2.2. Website Design

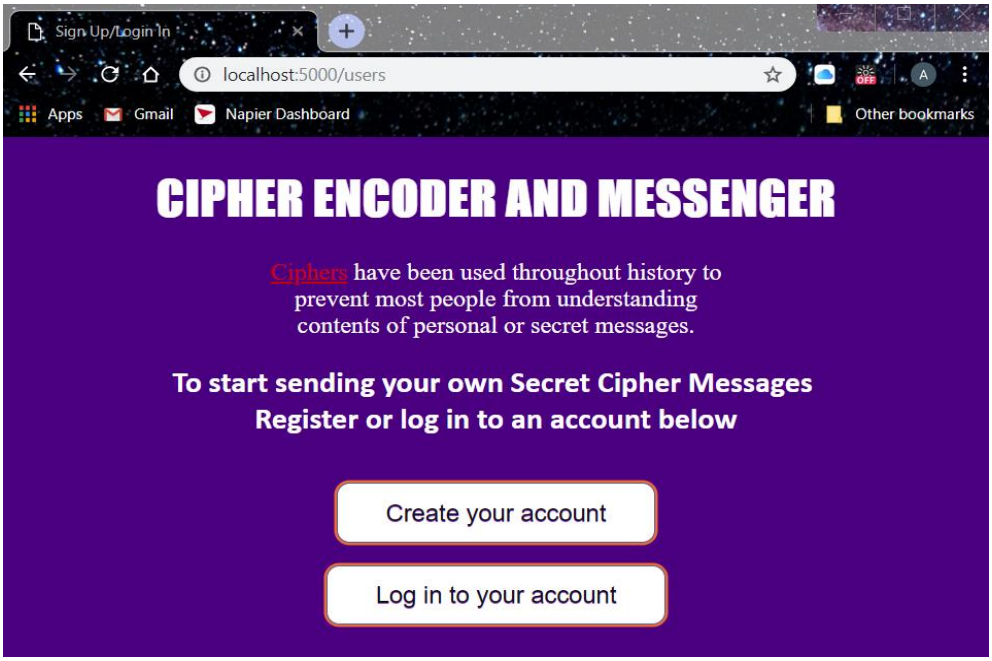


Image 1: Home page of the Website (using url: localhost:5000/users)

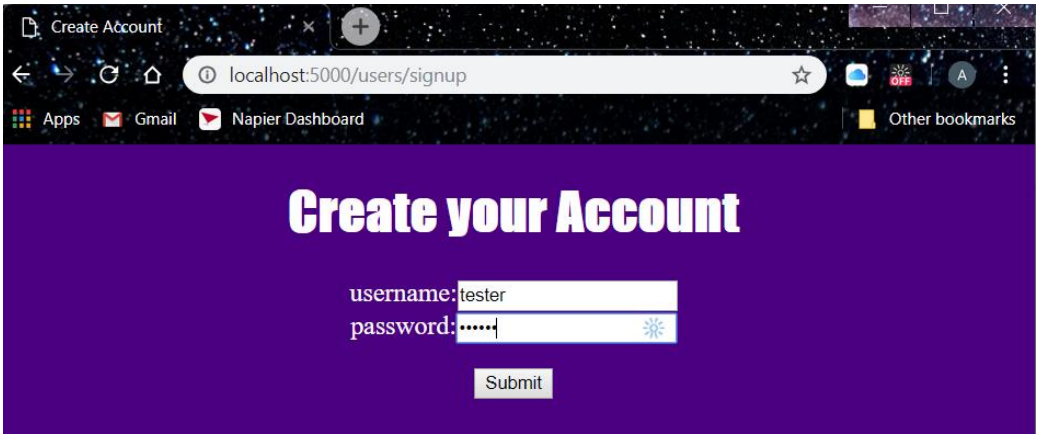


Image 2: Sign up page to create an account for the website

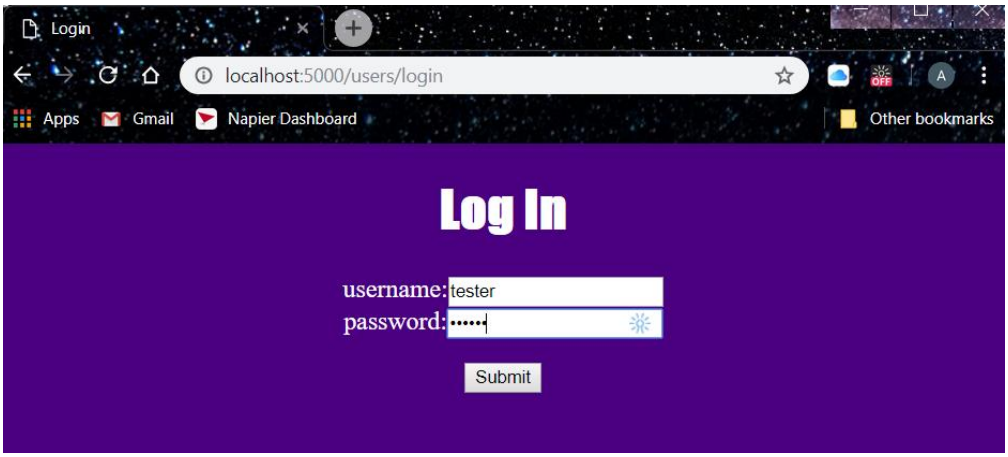


Image 3: Log In page to sign into a create account

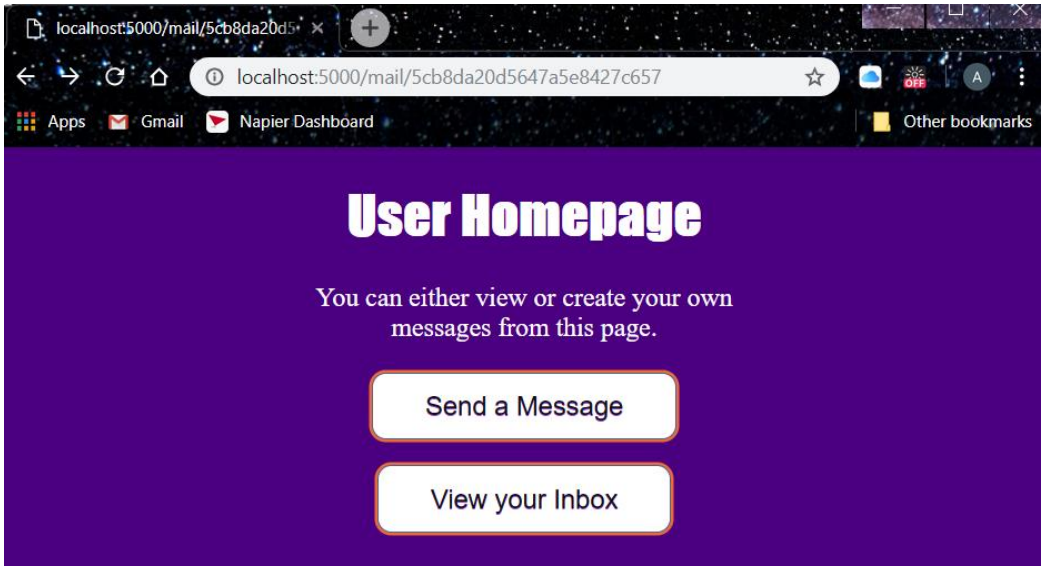


Image 4: User Homepage (they can either navigate to sending their own message or their inbox)

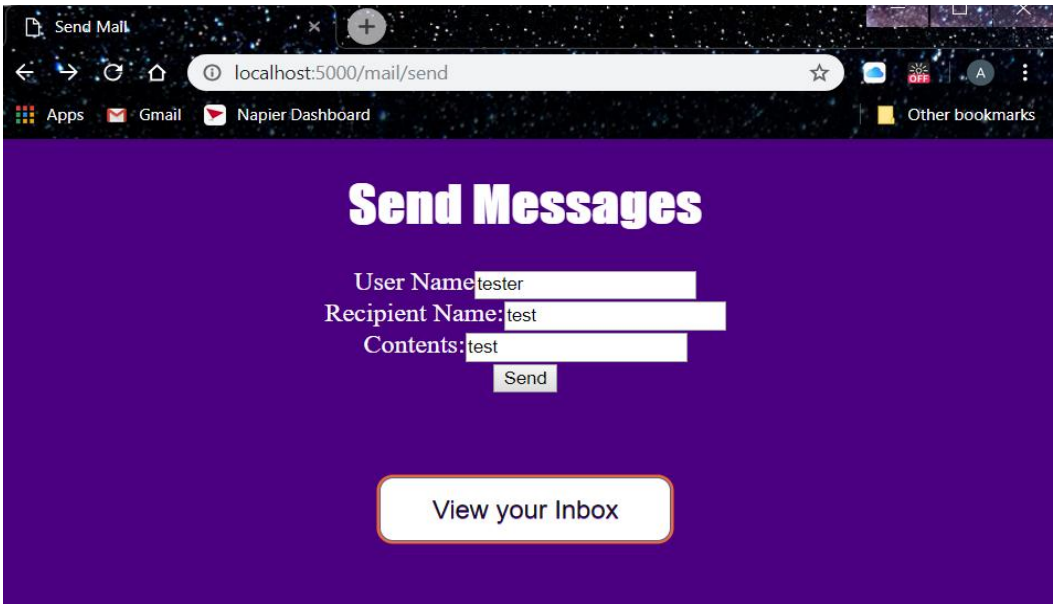


Image 5: Page to create and send your own messages

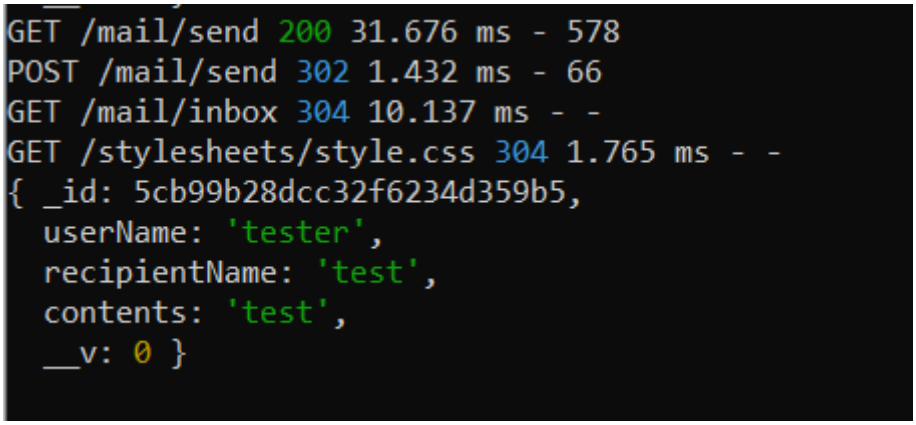


Image 6: Messages created and stored in the database

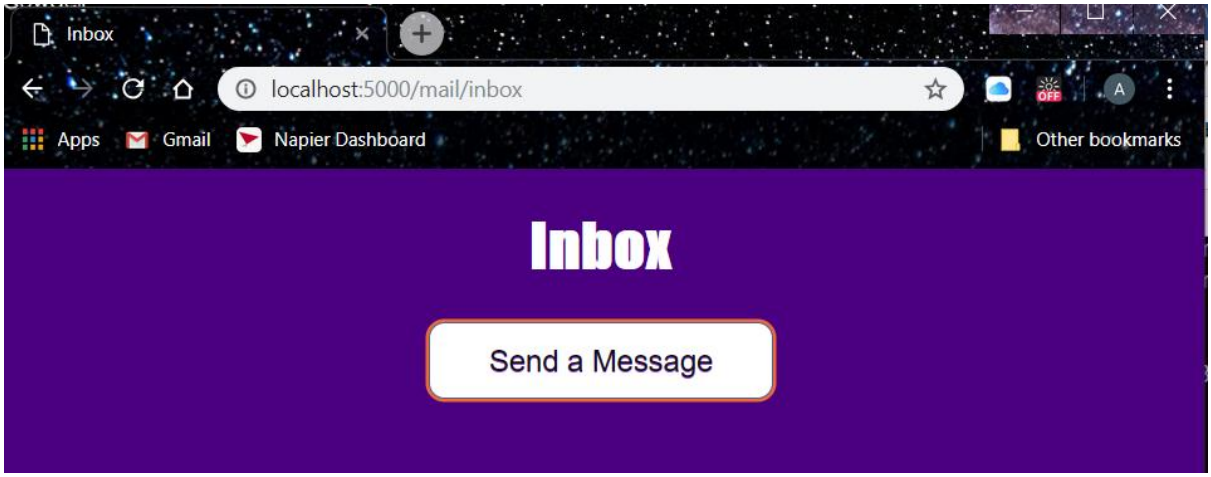


Image 6: Inbox Page

The overall design of the website is simple and designed to be easy for the user to navigate. By using buttons, the user can switch between the pages. The webpage is set up so that the user follows a chain of events. Initially, there is the index page which provides information about ciphers (including a hyperlink to a Wikipedia page: https://en.wikipedia.org/wiki/Cipher). The user has then the option to create a new account or skip to logging into their previous account. After logging in the user is take to their "home page" where they can either access their inbox or create a new message to send to other users.

3. Critical Evaluation

Overall, I believe that my website is basic but functional and is easy for a user to navigate. A user can create an account and it is persisted with a MongoDB meaning that they would be able to access it multiple times and their data will still be stored. Although it does not meet all the requirements (i.e. allowing the users to send encoded messages and functional user inbox), the basic functionality of a messenger website has been started.

Security measures implemented 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 bcrypt hash function was used so that it could not be accessed by anyone out with 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 and this year. I have spent a few weeks on the assessment and still have much room for improvement. 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 function (inbox).

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. It would have been also beneficial to work on the command in inbox to display messages that have been sent and received to their user.

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.

In addition, I would have liked to have my delete function working within the server to allow a user to either delete their account or to delete messages. 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

expressjs. (2019). Retrieved from https://github.com/expressjs

HTTP Methods GET vs POST. (2019). Retrieved from https://www.w3schools.com/tags/ref_httpmethods.asp

K, A., & Nadal, S. (2017). Pug (Jade) HTML form. Retrieved from https://stackoverflow.com/questions/43090770/pug-jade-html-form?fbclid=IwAR0rIe1UCHgmxGSUwbJFfczD-jlEro2BM4Ik12QrY1oaCa5TyAxDc3sUS5A

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

Olsson, M. (2014). Using CSS. CSS Quick Syntax Reference Guide, 1-4. doi: 10.1007/978-1-4302-6491-0_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/

remy/nodemon. (2019). Retrieved from https://github.com/remy/nodemon

Syed, B. (2014). Persisting Data. *Beginning Node.Js*, 165-180. doi: 10.1007/978-1-4842-0187-9_8