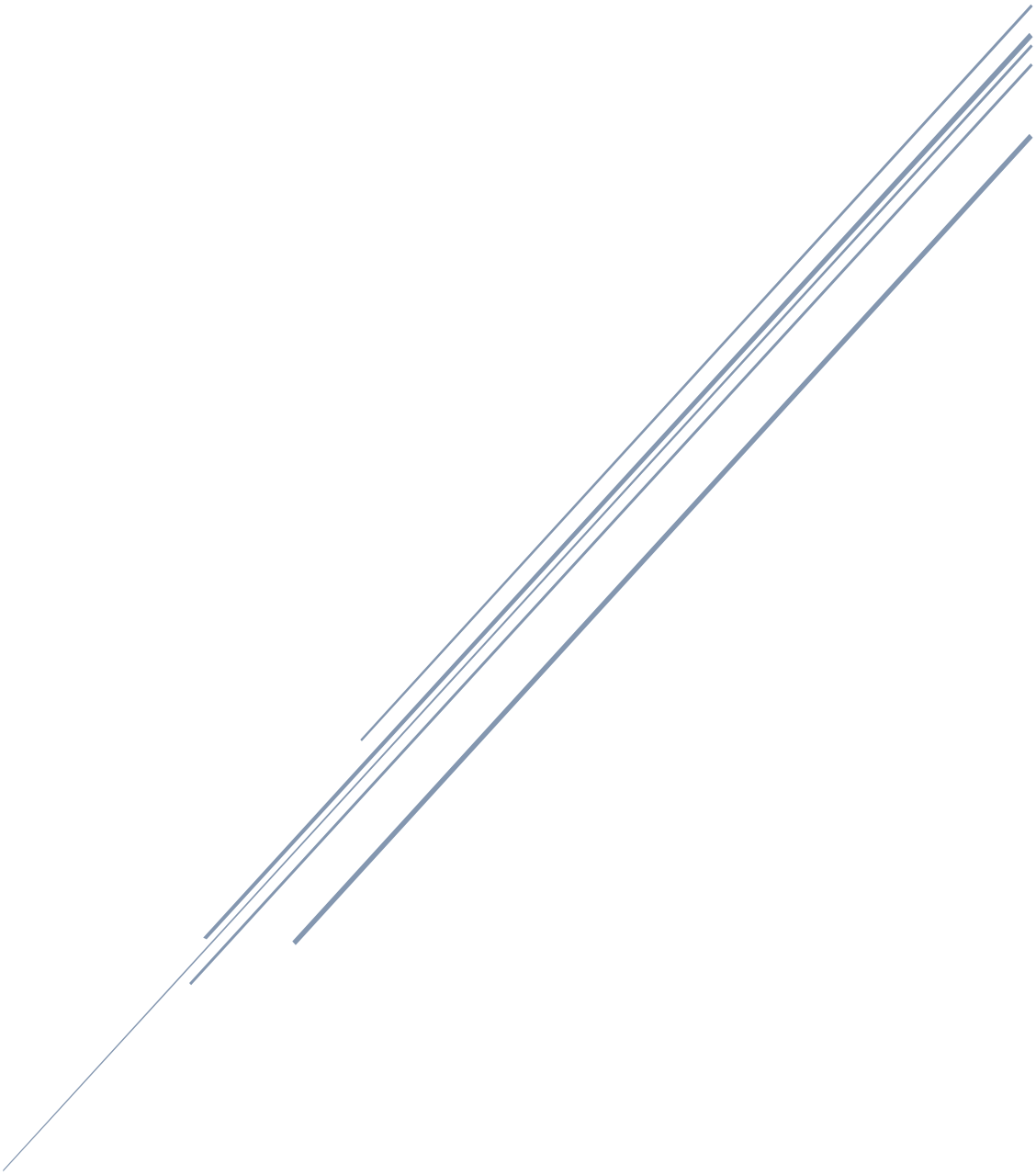
COURSEWORK 2 REPORT

Matthew Nichol



Napier
Web Tech

Coursework 2 Report

Matthew Nichol
40405649@live.napier.ac.uk
Edinburgh Napier University – SET 08101

1 – Introduction

The aim of this coursework is to develop an express application that allows users to send encrypted messages to each other using the ciphers created in Coursework one. The users will be able to create a profile on the website, encrypt messages to other users and view messages sent to them. All messages sent are anonymous, so if the user wants the recipient to know who sent the message it is recommended to sign it in the message body. When viewing received messages, the user's signs in and it will display the number of messages they have and each messages information, such as Cipher type, Shift number (if applicable), encoded message and decoded message.

When developing the express application, several reading materials were used to further develop knowledge on the topics. As there was no experience in Node JS or Express the books "Beginning Node.js" by Basarat Ali Syed and "Pro Express.js" by Azat Mardan were read. These gave a good foundation to the understanding on the core concepts of development of the app. When developing the databases, the book "The definitive guide to SQLite" by Michael Owens was used to get to grips with the syntax of queries and how the databases interprets data. In some cases, during development, my JavaScript knowledge escaped me so quick references to "JavaScript quick Syntax reference" by Mikael Olsson to quickly jog the memory from time to time.

2 - Design

Before starting the actual coding of the app, it was important to thoroughly read the coursework descriptor as this would give me a good understanding of the requirement of the website. These requirements were:

- Users will be able to sign up to the website
- Have plain text messages encrypted using selected ciphers
- Send encoded messages to other users on the site
- Be able to view messages sent to them, encoded and decoded
- Server must have data persist on users and messages

To get a better understanding on how I wanted the website to look and function, I first drew up some diagrams of file hierarchy layouts (fig 1). Such as where databases files would be located, where the JS and CSS files would be located.

For each page on the website, I want it to follow the same design as each other. I was quite fond of the Black on Blue design of Coursework one, so I intend for this to remain for Coursework two. However, each page will have the same layout as each other, to keep a constant structure throughout the website. I drew up a mock webpage to show this base structure (fig 2).

3 – Implementation

This website was created with the intent for users to be able to send encrypted messages to registered users. The website is composed of several separate files and apps that combine to provide the working product. These are:

Node.JS and Express.JS (fig 3)

Node.JS was used to run the server that will be used to view the website in action and for getting additional modules required to run the website. Express is most of the server-side code, it handles things like routing (such as get, post requests), the use of static files in webpages such as JavaScript and CSS files.

Sqlite3 (fig 4)

Sqlite3 was used for the databases being used on the website. This worked in conjunction with the website by creating two separate database files (one for users, other for messages) that can be accessed to view or add rows to the database.

PUG files (fig 5)

The template engine PUG was used for displaying content to the user. This was used because it allows for a template to be set, and additional webpages to simply build off that template.

4 – Evaluation

The requirements set out for the website were:

- Users will be able to sign up to the website
- Have plain text messages encrypted using selected ciphers
- Send encoded messages to other users on the site
- Be able to view messages sent to them, encoded and decoded
- Server must have data persist on users and messages

Looking at these I believe my website to have achieved every section. As users can create accounts, send encrypted messages and view them and when the server is switched off and on the data on users and messages remains.

As the website does meet these requirements it's important to look at how it meets these but how could they have been improved. Here a couple of ideas in regards to the requirements on how the website could be improved.

Users can sign up to the website

In my implementation a visitor to the website can very easily sign up to the website, however there is no checking whether this user already exists. This would cause havoc with how messages are retrieved by the server should two users have the same name.

This could be remedied by simply checking the users requested username with the users database and declining the submission should the username already be in use.

Have plain text messages encrypted

This section of the website is done rather well in my opinion. As the user can stay on the same webpage and choose between multiple different ciphers. It could have been improved by allowing users to take their encoded message and placing it back in the text box for deciphering rather than forcing the user to copy and paste should they wish to do that.

Be able to view messages sent to them

While my implementation allows for a user to login to their inbox and view all messages very easily I believe there is an issue with user experience. There is no way for a user to delete previous messages sent to them. When there are only three or four messages this isn't a major issue, but when considering upscaling if a very popular user has 100s of messages being sent to them this could be really annoying for a user. Simply adding a button that deletes the selected message from the database would remedy this issue.

5 – Self Evaluation

When developing the second piece of coursework, I learned many new pieces of software, languages and improved on some I already knew well. It was fun to learn exactly how a website handles each separate request and how complicated routes can become should the website require it, also understanding how data is stored in a database and how it can be accessed for verification and display purposes was a real eye-opener in how validations are really completed.

The biggest issue I ran into was learning SQLite3, I had a real issue with understanding how the tables were created and how data was entered to the tables. Thankfully, after reading "The definitive guide to SQLite" by Michael Owens and several YouTube videos I was finally able to understand how it all works.

I'm happy with how the website turned out, it hits all the notes required and hits them well in my opinion. However, I believe if I sat down a bit longer in the design stage and considered how the data would react with each other I could have predicted the issue with same usernames in signing up for the website.

6 – Appendices

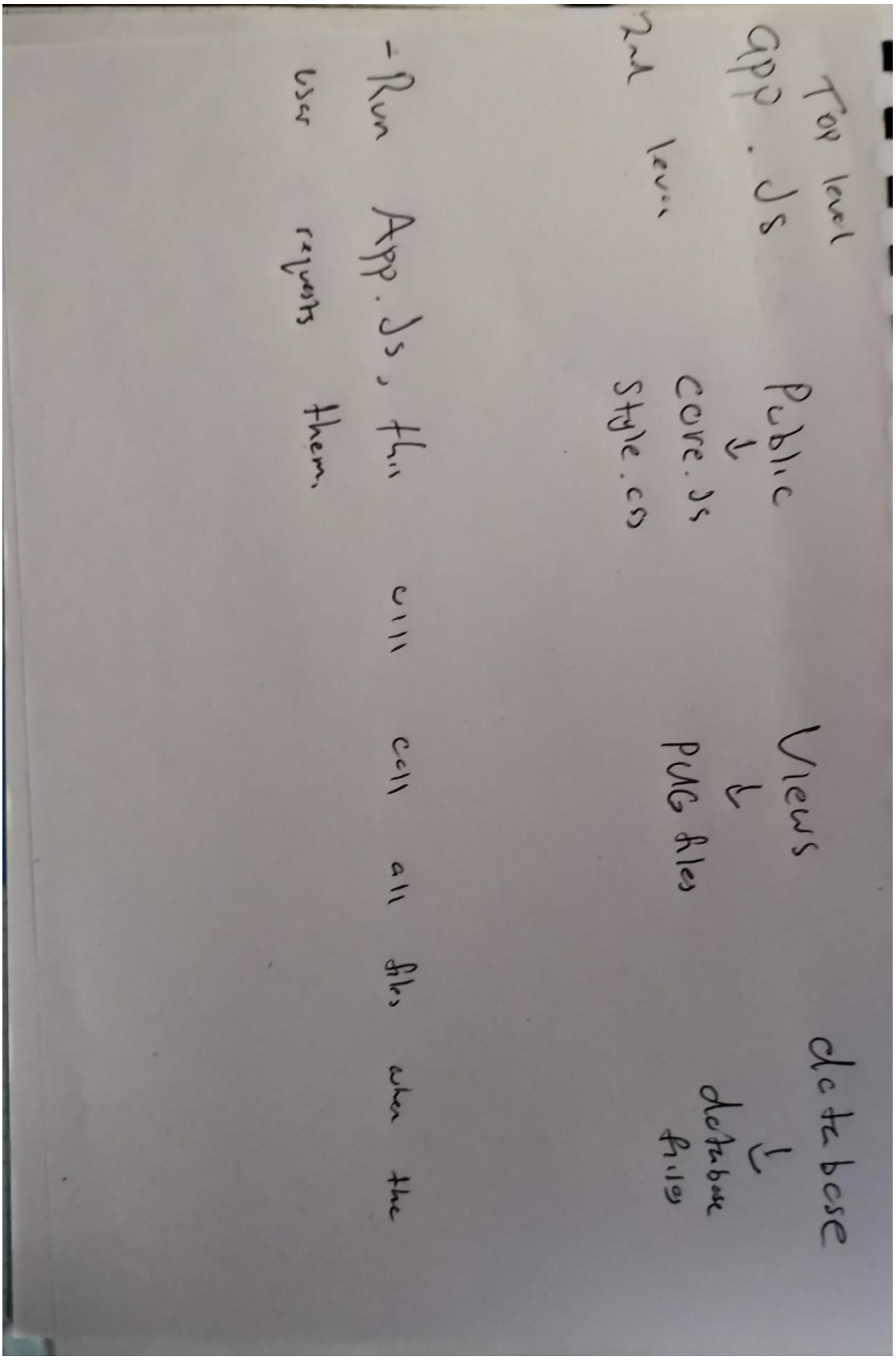


Figure 1 File Hierarchy

Figure 2 Template Example

```
JS app.js ●   JS test.js   JS core.js

1   //Required Modules
2   const express = require('express');
3   const app = express();
4   const path = require('path');
5   const BodyParser = require('body-parser');
6   const sqlite3 = require('sqlite3').verbose();
7   const http = require('http');
8   const bcrypt = require('bcrypt');
9
10  //Set up global variables being used
11  var authenticated = false;
12  var port = process.env.PORT || 5000;
13
14  //Connection to Databases
15  var db = new sqlite3.Database('./databases/users.db');
16  var msgdb = new sqlite3.Database('./databases/inbox.db');
17
18  //Creates Inbox Database
19  msgdb.serialize( function(){
20      msgdb.run('create table if not exists '
21      + 'inbox ('
22      + 'recipiant text,'
23      + 'body text,'
24      + 'cipher text,'
25      + 'shift numeric,'
26      + 'plaint text)');
27  });
28
29  //Creates User Database
30  db.serialize( function(){
31      db.run('create table if not exists '
32          + 'profiles ('
33          + 'u_name text,'
34          + 'hashed text)');
```

Figure 3 Snippit of App.js (nodejs and express.js)

```
function table_creation_profiles(data) {

    db.serialize( function(){
        var stmt = db.prepare('insert into profiles values (?,?)');

        stmt.run([data.u_name, data.hashed]);

        stmt.finalize();
    });
};


function table_creation_inbox(data){
    msgdb.serialize( function(){

        var stmt = msgdb.prepare('insert into inbox values (?,?,?,?,?)');
        stmt.run([data.recipiant, data.body, data.cipher, data.shift, data.plain]);

        stmt.finalize();
    });
};
```

Figure 4 SQLite being used to insert data

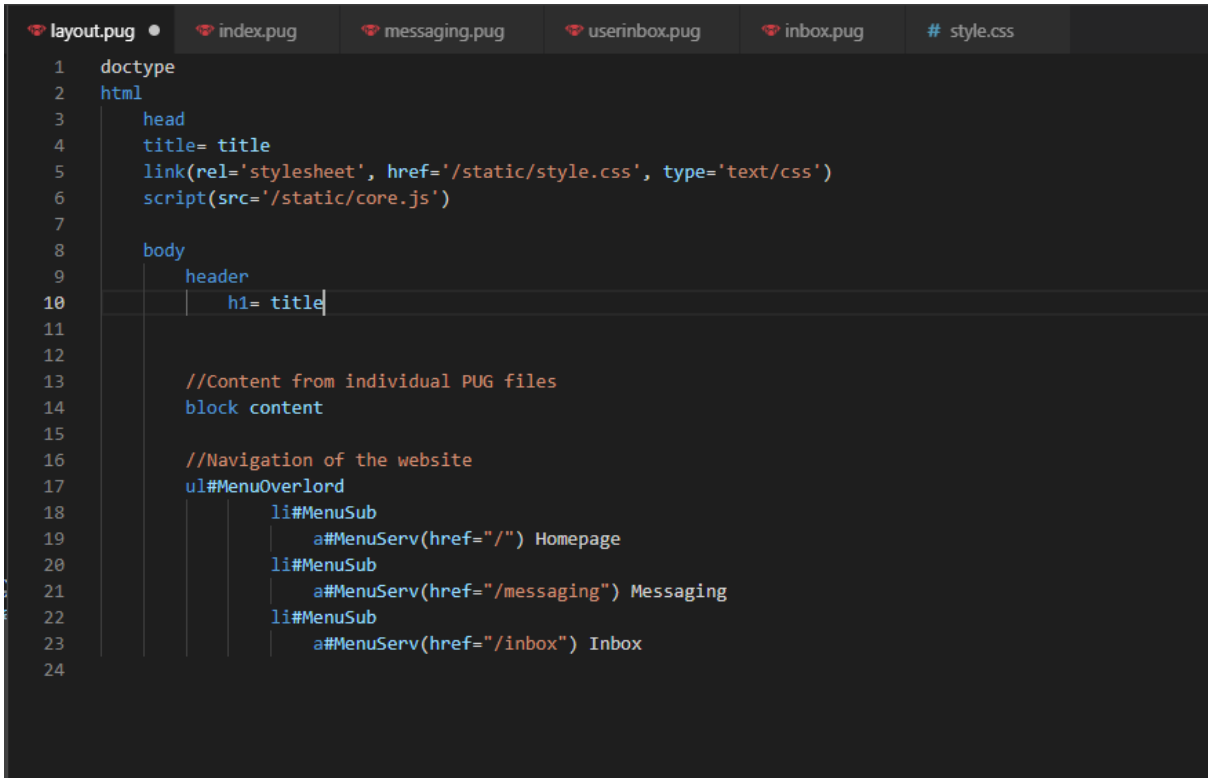


Figure 5 Layout PUG file example