2019

# COURSEWORK #2

## LOGIN SYSTEM, MESSAGE SYSTEM, SQLITE3

ALASTAIR EVANS | 40315129

# Contents

# Introduction

## Scope

The aim of this coursework is to deliver the following features:

- User account creation
- User login
- Account Information edit page
- Messaging system between users
- Send encrypted messages between users
- Make use of a database
- Host via node

## SQLITE3

SQLite3 was a requirement for the project. This type of database allowed me to create 2 tables for user account information and messages. Through the use of app.get in the server JavaScript I was able to send requests to the database to insert data into pre-configured fields.

## Node JS

This was used as the method for connecting the html of the sites to the browser instead of accessing the pages directly like in the previous coursework.

## JavaScript

This was used in the configuration of the server, a method of connecting user input via browser to the database, used to authorise user account details via username / password lookup.

## Site Design

The overall design scheme for the site will stay the same, simplistic black and white colour scheme with a side navigation bar. All new pages have been added to the side bar and due to the extra icons added the size has been decreased by 45%. A logout icon has also been created taking the user back to the login page. Smooth fade in for each page for user experience benefit remains.

# Software Design

Breakdown of the features provided:

**Login / Login Authentication**

A simple page requesting the user to insert a username and password will be created. Pressing a submit button will trigger an app.get to check the SQLite database for a username match. Should this return false the user will be redirected to an identical login page with a message saying username not found. Should the return be true, a function for user authentication will be triggered checking that the password provided matches the username provided, if true the user will be redirected to the main page. Should this return false they will be redirected to the login page with a message stating password incorrect.

**Account Creation**

An account creation page will be created allowing the user to input a username and password of their choice, these fields will be marked as required to stop entries of null / empty fields being submitted to the database. An app.get will be triggered upon pressing a button, this will cause a check to happen to see if the username already exists in the database. Should this return false, a function for create will be triggered inserting the users information into the database and redirecting them to the login page with a success message. Should there return be true (username exists) a redirect to the account creation page will happen with a message stating username taken, try again.

**Account Detail Management**

Once the user is logged in, an account creation page will be on the navigation bar. Upon load the site will populate the following fields with the information found in the database. Username, First name and Surname. There will be password field which is left blank however is required to be filled in to submit changes. New user accounts will have null or empty boxes for the first name and surname section of this site as the database will be empty.

Messaging System / Encrypting Message System

A simple messaging page will be created with 3 text boxes. A username box which is populated by an onload script pulling the username cookie into the field, this box will be read only to prevent users sending messages from users other than their self. A recipient box will be empty to allow the user to insert the username of the person they are messaging aswell as a message box for their message. A button press will trigger an app to take the information provided and insert it into a messages database.

The code from the ciphers created previously (caeser and morse code) will be inserted into the html of the messaging page. The JavaScript used by the ciphers will be duplicated and alerted to set the output area to the message box created in the plaintext messaging system. This will allow the user to select a cipher and process the output into the message box ready to be sent.

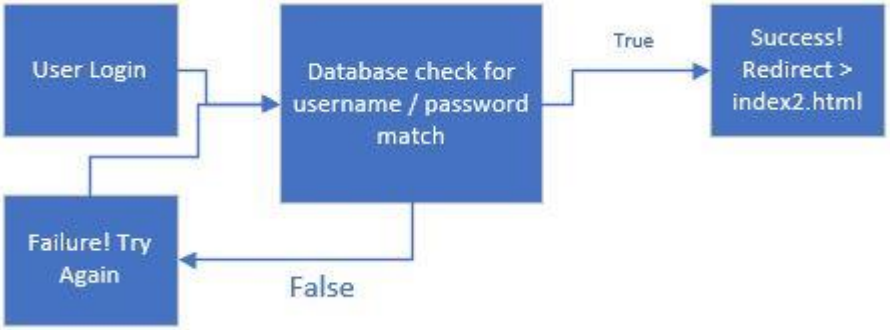
**Inbox System**

An inbox page will be created to display messages for the user. A button retrieving the mail will be used to trigger an app that will take the username from the cookie and search for sender entries in the messages database. JavaScript will be used to manipulate the output from the SQL database to improve the user experience.

A clear messages button will be added. When pressed it will take the user cookie and delete the appropriate lines when found in the messages database.

# Page Design

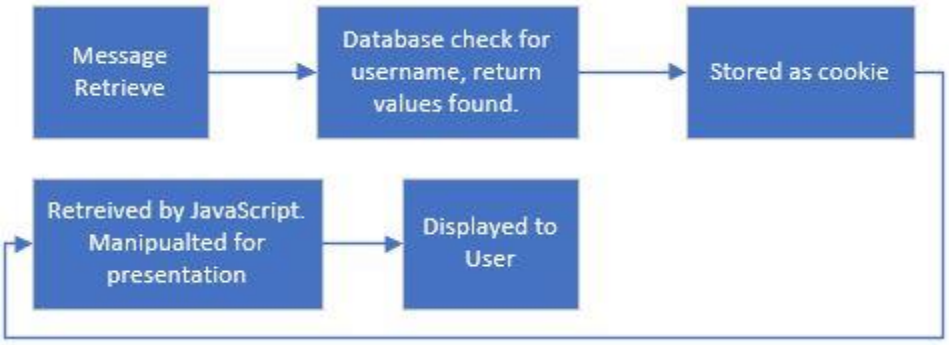
**Login Flow Diagram**




**Account Management Flow Diagram**


**Send Message Flow Diagram**


**Retrieve Mail Flow Diagram**




**Updated Navigation Bar**

# Implementation

Account Creation

This is the function used to create a new user, should the username taken from the webpage not match any entry in the username field it will return a value of false. This triggered a db.run to insert the information stored in the variables to the database for use in the login process. If returned true, the user will be redirected to another account creation page prompting the user to try a different username.

```
app.get('/signup', function (req, res) {

    var PW = req.query.Password;
    var UN = req.query.Username;

    userCHECK(UN, function(existing){
        if(existing == true){
        res.redirect('/loginUT.html');
    }
        if(existing == false){
        db.run('INSERT INTO UserAccounts(Password, Username) VALUES (?, ?)', [PW, UN]);
        res.redirect('/loginS.html');
    }
  });
})
```

Login Authentication

This is the function used to check if a user exists and if the password passed into the function matches what's in the database. Should it match, a true false is returned and is called back to the app that forwarded the information for checking.

```
nction Auth(username, password, cb){
 var user = false;
db.get("SELECT * FROM UserAccounts WHERE Username = '" + username + "' AND Password = '"+password+"' ", function(err,row){
   if(err) console.log(err);

   if(typeof row == "undefined"){
     user = false;
     cb(user);
   }else{
     user = true;
     cb(user);
   }
 })
```

**Send Message System**

This app takes the information from the fields on the web page and inserts them into the appropriate fields within the messages database. Once complete it will redirect the user to a message page with a success message.

```
app.get('/message', function (req, res){

  var Sender = req.query.userNAME;
  var MSG = req.query.MSG;
  var REC = req.query.REC;

    db.run('INSERT INTO Messages(Recipient, MSG, Sender) VALUES (?, ?, ?)', [REC, MSG, Sender]);

    res.redirect('/messageSENT.html');
```

**Retrieve Message System**

This is the function used for retrieving mail from the messages database. A username is passed into the function and its checked against the recipient field. Once data is found the result is stored as a variable and passed back via call back. It is then later processed by a separate JavaScript for presentation reasons. Db.all is used as it checks the entire database before moving further on in the script. A db.get would stop at the first entry found and pass it back before checking the next line causing the site to crash.

```
function checkMAIL(username, cb){

  var user = false;
  db.all("SELECT MSG, Sender FROM Messages WHERE Recipient = '" + username + "' ", function(err, row){
    if(err) console.log(err);


    else

        user = true;
        var MSG = row;
        cb(user, MSG);
```

**Encrypted Message System**

This was a simple change of the JavaScript used for the ciphers. Once the html properties for the cipher was implemented into the message page, a duplicate was made with a minor alteration to the output area. The output area was changed to the message box of the mail system allowing the user to easily send the output of the cipher.

```
var test = document.getElementById('MSG');
    test.value = cipher_text.join("");
```

# Evaluation

**Comparison Requirement vs Delivered**

A system for login, account creation, detail updates, messaging and encrypted messing incorporating the ciphers used previously were successfully implemented. The website design overall is smooth and in form with regards to the design scope. Easy to navigate with information provided to assist the user with operation.

**Possible Improvements / Problems**

Reworking the messaging system would be an area of improvement as it is causing some issues with decoding of morse code. The method used is simple in terms of function however creates some issues with presenting the data to the user resulting in the use of regex to tidy the data before presenting. Due to this I was unable to implement “Easy” decoding when receiving mail. It will need to be manually copied over to one of the cipher pages.

Password encryption would be a great improvement to the site as it currently does not offer this. To improve security when generating cookies I do not generate a password cookie until the user changes their password as well as clearing cookies at appropriate times.

# Personal Evaluation

The only experience I have with JavaScript is the previous coursework however I have previously studied a class about programming in python, so the skills were transferable in terms of reading and writing. No experience with SQLite3 but the use of database browser software allowed for easy configuration and maintenance of the database. Configuring node was covered in a lab which was help in figuring out how to configure it.

Much research was required for every aspect of this coursework as everything was relatively new to me. Many hours spent creating and testing parts of code that others would take 10 minutes to write.

As expected, lecture notes, labs, general discussion, google and stackoverflow have been very helpful in creating code and solving problems as they repeatedly arise. The great thing about having a resource like the internet is that every problem you can have someone else has had it, advice is never far away.

# Summative Evaluation

I feel I have delivered all of the basic requirements for this coursework majority of them working to full functionality. With better time management I could have produced a fully working easy decode system I would of most likely attempted doing the messaging system in a different way, if possible. I am grateful for the deadline extension as I have been burdened with unexpected responsibilities in other classes, so it has given me much needed time to balance the workload. I found the difficulty between coursework1 and coursework2 to be massive however after many hours of research I have managed to produce a working system.

# References


**Guidance for retrieving cookies.**

W3schools.com. (2019). *JavaScript Cookies*. [online] Available at: https://www.w3schools.com/js/js_cookies.asp [Accessed 19 Apr. 2019].

Guiselin, M. (2019). *How to display the value of a cookie in a .html file*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/23315854/how-to-display-the-value-of-a-cookie-in-a-html-file [Accessed 19 Apr. 2019].

**Guidance for SQL**

SQLite Tutorial. (2019). *Deleting Data in SQLite Database from a Node.js Application*. [online] Available at: http://www.sqlitetutorial.net/sqlite-nodejs/delete/ [Accessed 19 Apr. 2019].

W3schools.com. (2019). *Node.js MySQL Select From*. [online] Available at: https://www.w3schools.com/nodejs/nodejs_mysql_select.asp [Accessed 19 Apr. 2019].

101, C. (2019). *node.js - sqlite3 read all records in table and return*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/39106668/node-js-sqlite3-read-all-records-in-table-and-return [Accessed 19 Apr. 2019].

Codecademy Forums. (2019). *Why use db.each() instead db.all() or db.get() in node-sqlite?*. [online] Available at: https://discuss.codecademy.com/t/why-use-db-each-instead-db-all-or-db-get-in-node-sqlite/381382 [Accessed 19 Apr. 2019].

**Guidance for variable manipulation**

W3schools.com. (2019). *JavaScript String replace() Method*. [online] Available at: https://www.w3schools.com/jsref/jsref_replace.asp [Accessed 19 Apr. 2019].

**Images for Navigation Bar**

https://www.123rf.com/photo_17569954_painted-carrier-pigeon-vector-illustration.html

https://www.iconfinder.com/icons/174115/cog_gear_settings_sprocket_icon