Web Technologies coursework#2 report

Cipher website with messaging platform

Carson Sanders 40429321

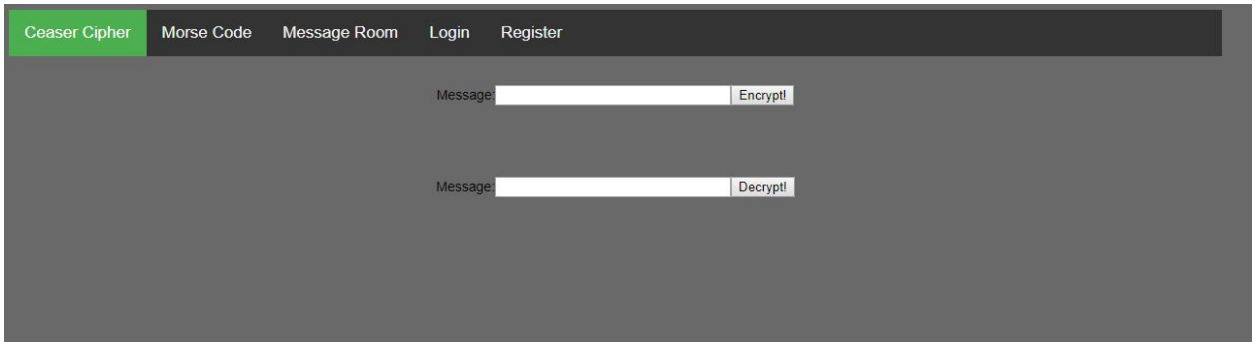
Introduction

This report details the design and implementation of the messaging platform for the second coursework of the Web Technologies module. The website features two message ciphers: a caesar cipher, and a morse code encrypter. It also extends its functionality from the first coursework by adding a messaging platform, and a way for users to login and register accounts for the website.

Software design

The website was designed using Node.js for server management, Express for serving web pages and managing routing, and SQLite3 storing data.

The plan for the website was to extend the design from the first coursework, by adding three new pages to the navigation bar: a Messageroom to handle messaging, a login page, and a registration page where users would create new accounts.



Note the three additional links in the navigation bar.

Data for user messages and username/password information is stored in a SQLite3 database.

Implementation

The express file index.js handles routing. For our static pages that handle the original ciphers from coursework 1, all that is needed is a simple get function that serves the static webpage. The encryption is handled with a Javascript file.

However for the login page and message room, we need a way to handle obtaining new information from the user. This is done with a POST request that store the information given in out SQL server.

```
/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

router.get('/ceaser', function(req, res, next) {
    res.render('cipher', {title: 'ceaser'});
});

router.get('/morse', function(req, res, next) {
    res.render('cipher', {title: 'morse'});
});


router.get('/messageroom', function(req, res, next) {
    var dbMessages = "";

    let sql = `SELECT message,
                      username
                      FROM messages`;

    db.all(sql, [], (err, rows) => {
        if (err) {
            throw err;
        }
    rows.forEach((row) => {
        console.log(row.message);
        dbMessages += row.username + " " + row.message + " ";
    });
    res.render('messageroom', {messages: dbMessages});
    });

})
```

```
router.post('/login', function(req, res, next) {
    var inputUser = req.body.username;
    var inputPass = req.body.password;

    var dbUser;
    var dbPass;
    var passCheck = false;

    let sql = `SELECT username,
                      password
                      FROM users`;

    db.all(sql, [], (err, rows) => {
        if (err) {
            throw err;
        }
    rows.forEach((row) => {
        if(row.username == inputUser && row.password == inputPass) loggedIn = true;
    });
    if (loggedIn){
        loginStatus = "loggedin";
        username = inputUser;
    }else loginStatus = "loginfail";
    res.redirect('/login');
    });
})
```

```
router.post('/messageroom', function(req, res, next) {
    var message = req.body.message;
    var messageUser;

    if(username == "") messageUser = "Anon:";
    else messageUser = username + ":";

    db.run(`INSERT INTO messages(message, username) VALUES(?,?)`, [message, messageUser],function(err){
        if (err){
            return console.log(err.message);
        }
        res.redirect('/messageroom');
    });

})
```

New messages are displayed by retrieving the messages from the database, and inputting them into a PUG variable. This allows us to send and update the website as messages come in without altering an html document directly.

```
doctype html
html
    head
        title= title
        messages = messages
        username = username
        link(rel='stylesheet', href='/stylesheets/style.css')
    body

        div(class="topnav")
            a(href="/ceaser") Ceaser Cipher
            a(href="/morse") Morse Code
            a(class="active" href="/messageroom") Message Room
            a(href="/login") Login
            a(href="/register") Register


        form(method="POST" action="/messageroom") Message:
            input(name='message', type='text', size='30')
            input(type='submit')

        p(id="mroom") #{messages}
```

Note the #{messages}, the variable that stores all the messages from the database.

Critical Evaluation

All requirements listed in the coursework description have been met. Users can register new accounts and log in, as well as send messages to other users. These messages can then be converted/encrypted using the two original ciphers from the first coursework.

While all requirements are met, The messaging system is a bit clunky to say the least. It works in very much the same way to an old school chat room. There is only one messaging area where everyone can send messages. This brings a couple of issues: mainly scalability. While this design is fine for a small amount of users, a large population of users would quickly flood the message room with messages and it would be very difficult to communicate.

Personal Evaluation

Personally I enjoyed learning the new web technologies of Node.js, Express, and SQLite3. It took a lot of time transitioning from developing static web pages with raw HTML and Javascript files to developing with Node.js and Express, however once the dots finally connected, imagining the possibilities with developing with a server and

database was enjoyable. The Aha! Moment that followed hours of frustration almost made the frustration worthwhile.