Coursework2 Report

Richard Cadelano

40315121@live.napier.ac.uk

Edinburgh Napier University – Web Technology (CSI09101)

Introduction.

The main task of this coursework will be to improve the cipher website by implementing a Platform that will allow the user to Register an account, and subsequently access through a Login interface, apart from the possibility of encrypt text the user will be allowed to send crypted e-mails. The Application will be divided into:

- A Client side: it will provide a web interface with an initial Authentication interface, the User will be able to login with is credentials, in case of a new user a Sign-up form will permit the Registration of a new account; after the access the user will be able to access to the Caesar Cipher section or the Morse code section, a new feature will allow the user to send e-mail messages in plain-text or encrypted into Morse code,
- A Server side: it represents the engine behind the appearance, this side of the application will process all the functionalities, will process all the Users data and will direct the message activity.

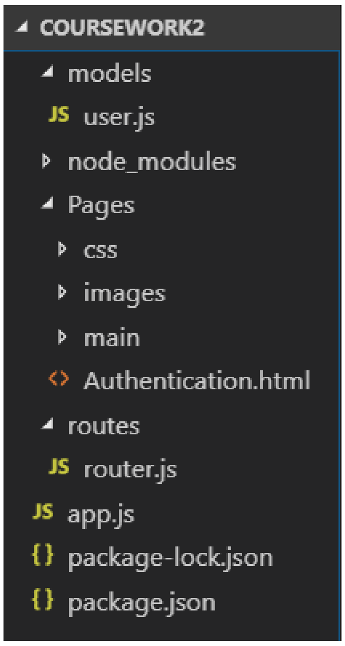
Once Authenticated the user will be able to navigate between sections with the animated buttons implemented into the main page, and a top-bar will allow the logout and the access to the message section.

In this Report will be analysed the creation and the work of the Server side, built through Node.js and an express base it represents the structure of the Application; will be analysed all the node modules used and their use.

Software Design.

Before start with any implementation is good to dedicate some time to clarify which are the core requirements for our project and the possible ways to achieve it. Before the coding activity my personal approach started with pen and paper by determining the components of the application and the possible connections, then was necessary to apply the draw to the software, that means determine the Main Routes requirement, after that the most important job was determine which modules and how use them to achieve the goal.

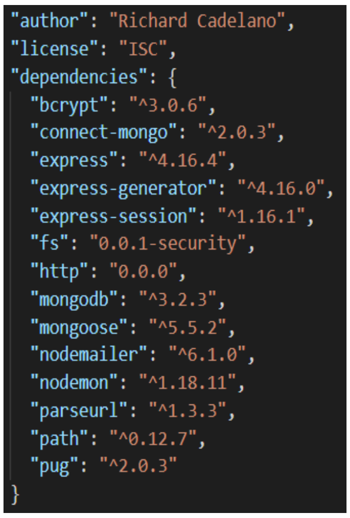
Starting from a template built with express-generator now is possible to analyse the actual Folders Structure:



```
COURSEWORK2
  models
    user.js
  node_modules
  Pages
    css
    images
    main
    Authentication.html
  routes
    router.js
  app.js
  package-lock.json
  package.json
```

As is possible to see in the picture there are different sections:

- Models: where is contained the Database Schema, the values (email and password) are stored during the registration process and checked for the Login,
- Pages: where are store all the html and CSS files relatives to the app webpages,
- Routes: here is stored the file that include all the routes setting and the functions relative to them,
- Node_modules: here are installed all the Node Dependencies installed,
- App.js: the main start file of the application, this is the centre that connects all the app elements,
- Package.json: this is the main file with app.js in this file are stored the start-up settings of the app and the dependencies used.



```
"author": "Richard Cadelano",
"license": "ISC",
"dependencies": {
  "bcrypt": "^3.0.6",
  "connect-mongo": "^2.0.3",
  "express": "^4.16.4",
  "express-generator": "^4.16.0",
  "express-session": "^1.16.1",
  "fs": "0.0.1-security",
  "http": "0.0.0",
  "mongodb": "^3.2.3",
  "mongoose": "^5.5.2",
  "nodemailer": "^6.1.0",
  "nodemon": "^1.18.11",
  "parseurl": "^1.3.3",
  "path": "^0.12.7",
  "pug": "^2.0.3"
}
```

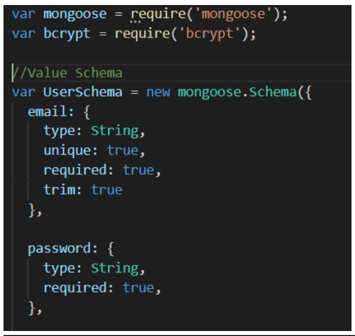
Focusing on the requirements for the Server, in this picture is possible to see all the dependencies used, most of them are installed automatically after the template creation but let's have a look to those installed externally:

- Mongoose: this module is used to create a new MongoDB Database and to manage the connections to it,
- Connect-mongo: used during the sessions to track login activities,
- Bcrypt: Used to Encrypt the password stored in the Database,
- Nodemailer: this module allows the app to send email messages through the SMTP port.

Implementation.

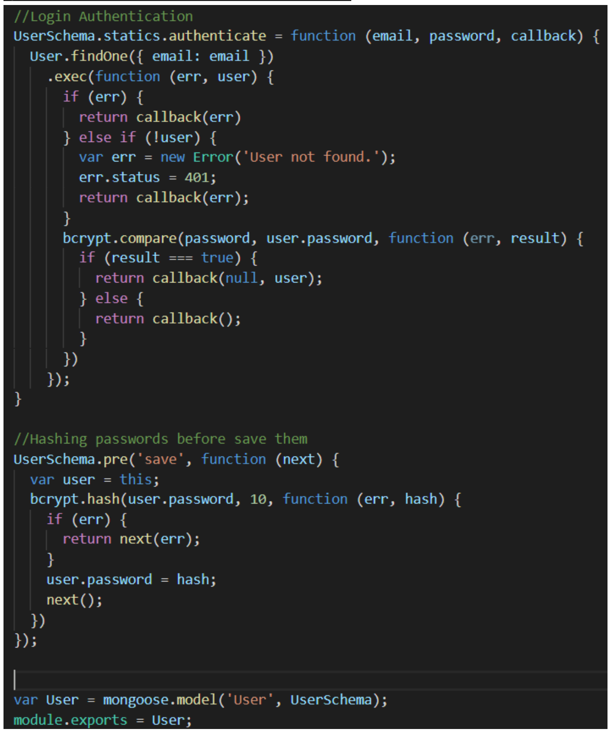
After the analysis of the Folder Structure and a briefly look at the external dependencies installed it is time to focus on the functionality implemented in this node application.

Once created the application through the express-generator module the most important thing was to develop an Authentication side to allow the users to login or register a new account, to achieve this was necessary to create a database to store the values and build a Schema to organise them:



```
var mongoose = require('mongoose');
var bcrypt = require('bcrypt');

//Value Schema
var UserSchema = new mongoose.Schema({
  email: {
    type: String,
    unique: true,
    required: true,
    trim: true
  },

  password: {
    type: String,
    required: true,
  },
```

File users.js: as we can see our Schema Require the Email, password information, without one of them is impossible to complete the Login/Register process, the Schema forbids the use of the same email more than one time.



```
//Login Authentication
UserSchema.statics.authenticate = function (email, password, callback) {
  User.findOne({ email: email })
    .exec(function (err, user) {
      if (err) {
        return callback(err)
      } else if (!user) {
        var err = new Error('User not found.');
        err.status = 401;
        return callback(err);
      }
      bcrypt.compare(password, user.password, function (err, result) {
        if (result === true) {
          return callback(null, user);
        } else {
          return callback();
        }
      })
    });
}

//Hashing passwords before save them
UserSchema.pre('save', function (next) {
  var user = this;
  bcrypt.hash(user.password, 10, function (err, hash) {
    if (err) {
      return next(err);
    }
    user.password = hash;
    next();
  })
});


var User = mongoose.model('User', UserSchema);
module.exports = User;
```

File users.js: in this function are checked the values, against the Database during the Login Process, is possible to observe the function bcrypt, used to hash the passwords during the registration process Before save them in the Database and also is used during the Login process.

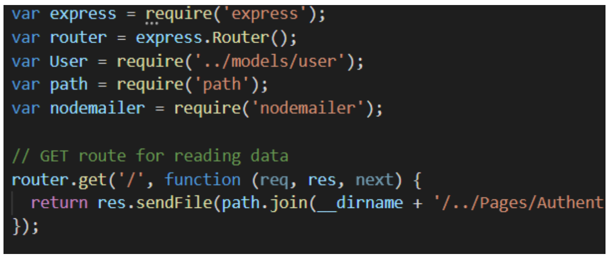
The command: mongoose.model creates the Database 'Users' following the Schema created before.

The command module.export allows the app to call this module in another js file.

The code relative to the use of mongoDB is referenced in the References paragraph.

After the creation of the Database id possible to access it by using the app: MongoDB Compass Community.

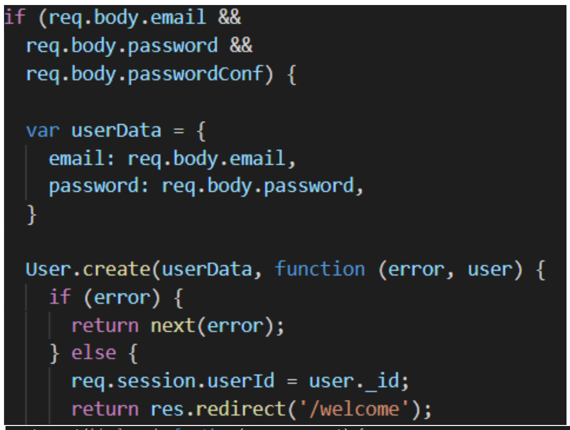
All the interactions with the app are regulated by the file router.js: it is here that we decide which page display as first or which function activate:



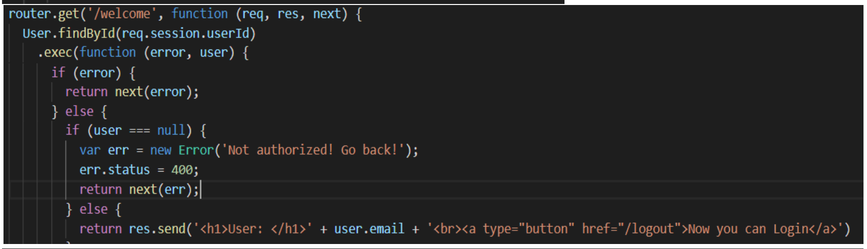
```
var express = require('express');
var router = express.Router();
var User = require('../models/user');
var path = require('path');
var nodemailer = require('nodemailer');

// GET route for reading data
router.get('/', function (req, res, next) {
  return res.sendFile(path.join(__dirname + '/../Pages/Authent
});
```

Here is possible to see the module imported, particular attention to the file user.js(Database) and the module "nodemailer"(messaging)

Here we can see the page displayed as first: our authentication form.



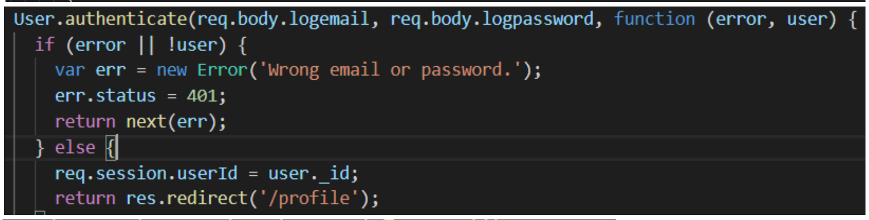
```
if (req.body.email &&
  req.body.password &&
  req.body.passwordConf) {

  var userData = {
    email: req.body.email,
    password: req.body.password,
  }

  User.create(userData, function (error, user) {
    if (error) {
      return next(error);
    } else {
      req.session.userId = user._id;
      return res.redirect('/welcome');
```



```
router.get('/welcome', function (req, res, next) {
  User.findById(req.session.userId)
    .exec(function (error, user) {
      if (error) {
        return next(error);
      } else {
        if (user === null) {
          var err = new Error('Not authorized! Go back!');
          err.status = 400;
          return next(err);
        } else {
          return res.send('<h1>User: </h1>' + user.email + '<br><a type="button" href="/logout">Now you can Login</a>')
```

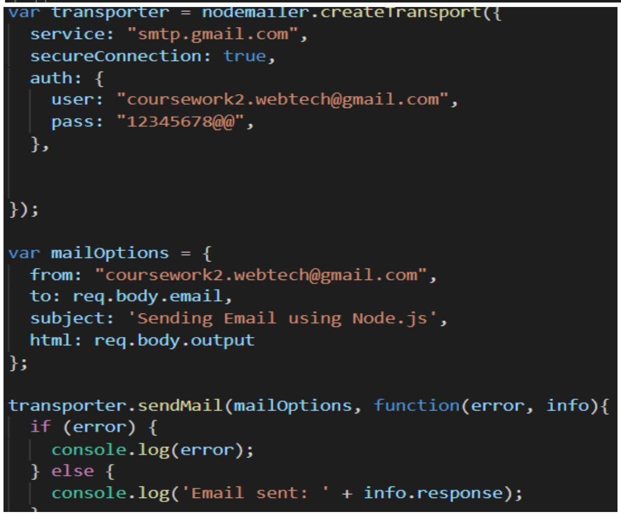
Sign-up Process: as is possible to see the app catch the information from the Register form and store them into the variable UserData, then through the function User.create the values are stored in the database and the user is redirect to the route "/welcome" (bottom picture) where will be displayed a message with the username and a button to redirect the new user to the Login.

The code relative to the use of authentication is refenced in the References paragraph.



```
User.authenticate(req.body.logemail, req.body.logpassword, function (error, user) {
  if (error || !user) {
    var err = new Error('Wrong email or password.');
    err.status = 401;
    return next(err);
  } else {
    req.session.userId = user._id;
    return res.redirect('/profile');
```

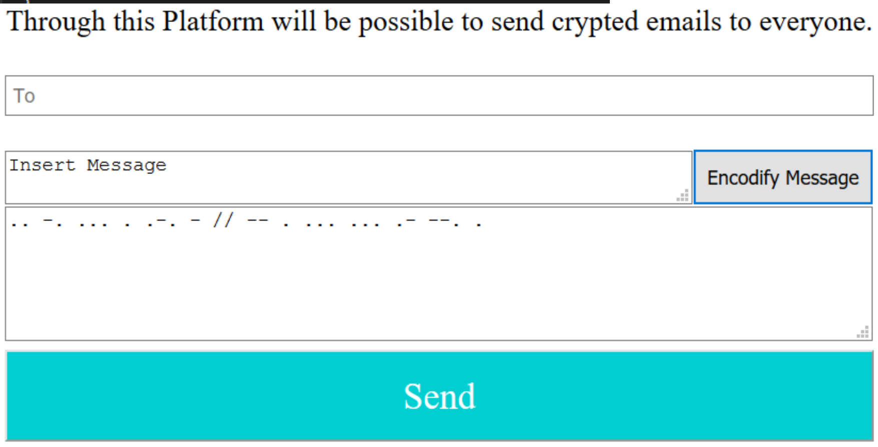
Login Function: Through the command User.authenticate, the app check the values inserted into the Login Form against the values stored in the Database, if successful the user will be addressed to the route:"/profile.



```
var transporter = nodemailer.createTransport({
  service: "smtp.gmail.com",
  secureConnection: true,
  auth: {
    user: "coursework2.webtech@gmail.com",
    pass: "12345678@@",
  },


});

var mailOptions = {
  from: "coursework2.webtech@gmail.com",
  to: req.body.email,
  subject: 'Sending Email using Node.js',
  html: req.body.output
};

transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
  } else {
    console.log('Email sent: ' + info.response);
```

Message Function: through the module "nodemailer" is possible to send emails from the app, to make that possible is necessary build a variable transporter, that contains: the protocol used and the Data of the sender email, I have created one gmail account dedicated to the coursework.

With the variable mailOptions the app catch the information of receiver and message from the dedicated text-boxes.



Through this Platform will be possible to send crypted emails to everyone.

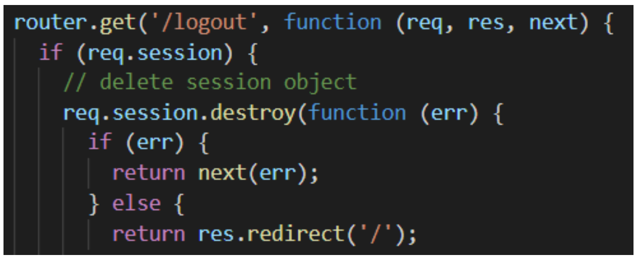
To

Insert Message

```
.. -. ... . .-. - // -- . ... ... .- --. .
```

Message Box: as is possible to see is possible to encrypt the message into Morse code before sending it.

The code relative to the module nodemailer is referenced in the Reference paragraph.

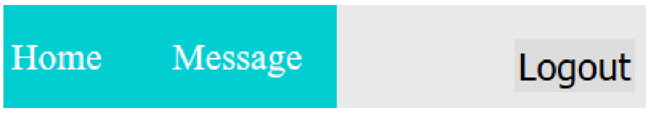


```
router.get('/logout', function (req, res, next) {
  if (req.session) {
    // delete session object
    req.session.destroy(function (err) {
      if (err) {
        return next(err);
      } else {
        return res.redirect('/');
```

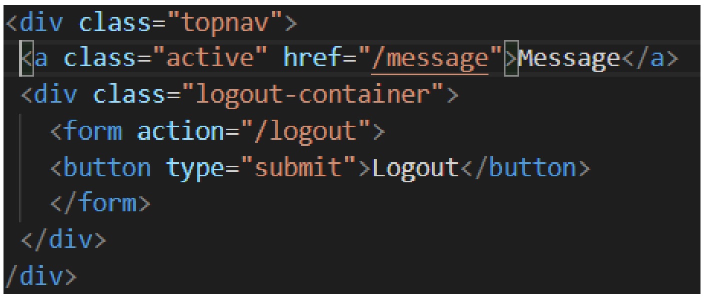
Logout function: once pressed the logout button the session will be closed, and the user will be redirected to the login page.

All the modules are called inside the main file “app.js”, at the launch the app will create the Database, if not existing. All the textbox are identified by a particular id to allow the app to parse the information required to run the functions, to parse the text I have used the module: bodyParser.

After the login the user will be able to access to the message page or logout through the top navbar:

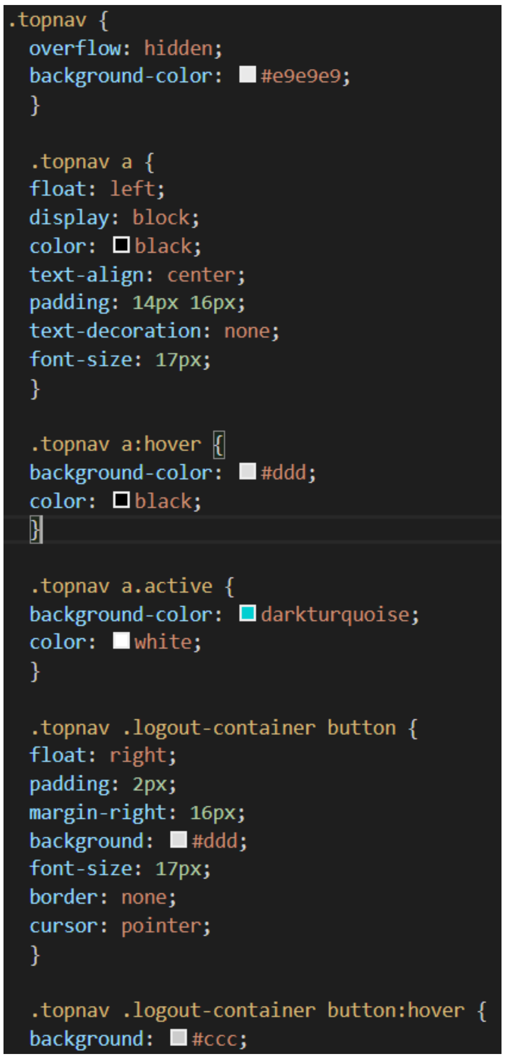


Navbar visualised on the browser.



```
<div class="topnav">
 <a class="active" href="/message">Message</a>
 <div class="logout-container">
   <form action="/logout">
   <button type="submit">Logout</button>
   </form>
 </div>
/div>
```

Simple html code to insert the navbar in the page with the active buttons.



```
.topnav {
  overflow: hidden;
  background-color: #e9e9e9;
  }

  .topnav a {
  float: left;
  display: block;
  color: black;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
  font-size: 17px;
  }

  .topnav a:hover {
  background-color: #ddd;
  color: black;
  }

  .topnav a.active {
  background-color: darkturquoise;
  color: white;
  }

  .topnav .logout-container button {
  float: right;
  padding: 2px;
  margin-right: 16px;
  background: #ddd;
  font-size: 17px;
  border: none;
  cursor: pointer;
  }

  .topnav .logout-container button:hover {
  background: #ccc;
```

CSS code to determinate the style of the top navbar: as is possible to see the active buttons are characterised by a background colour turquoise, on the other side the logout button presents a grey shape.

Critical Evaluation.

Examining the Coursework Requirements and my personal work, the main discrepancy could be found into the message side of the application: one of the main goals of this coursework was to allow the users to exchange coded messages between themselves, I have implemented a platform that allows the user to send coded email outside the app, so the first thing to implement would be for sure an "user-limited" messaging platform, another implementation needed could be an user menu to allow users to read the messages and encode/decode them. Checking on the Authentication side the function implemented is able to store the data inside the Database and check the components during the Login Process, a possible improvement could be using a different and more accurate authentication module to increase the standard of security and simplify the coding process. Also an Administrative page that could allow the "owner" of the platform to decide if block or delete a user could be a nice and useful improvement.

Personal Evaluation.

During the lectures and the labs done in the university days I had the occasion to set the basics to build this platform, a good explanation of the Node.js server application and a little about databases helped me to draw the first scheme, obviously the class work was only an input and this coursework gave me the occasion to study and learn much more about Developing Server apps through node.js and express. After the Exercise in the Lab, that helped me to understand the basics of the express module the first challenge was implement an authentication for the platform, and decide which kind of Database use, my initial choice was create an Sqlite database locally through the app, but after some researches I have opted for create a MongoDB Database, that made the interactions between the app and the Database easier and, consequently, the coding. I have found the module suggested for hashing passwords well descripted and not so difficult to implement. After the Authentication another challenge was represented by developing the message platform, my original plan was to implement into my old webpage a system to send a message for each cypher, but at the end I have opted for create another different route and page for that, implementing the morse code to encode the message before sending it, the absence of a user menu and the possibility to send emails outside the platform made me try to implement the actual message platform, by creating a dedicated gmail account. The tutorials on W3 schools and the "ProexpreesJs" book on the Reading List were very helpful to gain information and see practical examples of the modules.

References.

[1]Nick Karnik(2018)Introduction to Mongoose for MongoDB[Online] Available at: https://medium.freecodecamp.org/introduction-to-mongoose-for-mongodb-d2a7aa593c57 Accessed on:12/3/2019

[2]Daniel Deutsch(2017)Starting with Authentication (A tutorial with Node.js and MongoDB)[Online] Available at: https://medium.com/createdd-notes/starting-with-authentication-a-tutorial-with-node-js-and-mongodb-25d524ca0359 Accessed on:19/03/2019

[3]Azat Mardan(2014)Pro Express,js.Apress. Part2: Deep Api Refence.

[4]npm(2019)bcrypt[Online] Available at: https://www.npmjs.com/package/bcrypt Accessed on 20/03/2019

[5]W3schools.How To Create a Login Form[Online] Available at: https://www.w3schools.com/howto/howto_css_login_form.asp Accessed on 22/03/2019

[6]W3schools.Responsive Navigation Bar[Online] Available at: https://www.w3schools.com/howto/howto_js_topnav_responsive.asp Accessed on 3/04/2019

[7]W3schools.The Nodemailer Module[Online] Available at: https://www.w3schools.com/nodejs/nodejs_email.asp Accessed on 11/04/2019