

Table of Contents

Project Analysis.....	2
Prospective Users	3
Researching the Problem	3
Mailchimp: an email management and design system.	3
Interview	4
Limitations	5
Objectives.	6
Initial Diagrams.....	7
Initial PHP Class Diagram	7
Initial Database System Design.....	8
Intended algorithms, techniques, and methods.	8
Prototyping parts of my Program.....	9
Communicate with the database using SQL.	9
Track whether an email has been opened or not.	10
Test Dragging and dropping.	11
Being able to send emails using php using PHPMailer. Allows attachments, embedded images and more customisation.	11
Documented Design	13
Diagram Designs and System Structure.....	13
PHP Structure Diagram (file stacking)	13
Ajax Structure Diagram.....	14
User Interface Design	14
Revised Class Diagram.....	16
Revised Database Structure	16
Fonts & Images.....	17
Password Encryption System.....	17
Email Opened Tracking System.....	17
Image Upload and Embedding System	17
Template Saving and Selecting.....	17
Key Variables.....	18
SQL Design	20
Get Templates	20
Save Templates	20
Get Admin Information.....	21
Get Emails	21

Key Algorithms	21
Communicating with Database (SELECT Templates)	21
Communicating with Database (INSERT Template)	22
Removing the attributes inside the generated html for website content to be ready to be sent (function mouldContents)	22
Generate SQL to get the Email Recipients (function generateRecipients)	22
Adding a new image to dictionary of images	24
Encryption Algorithm	25
Testing	25
Test Plan	25
Evaluation	29
Technical Solution:	33
frontadmin.php (admin)	36
postdelivery.php (admin)	42
index.php (admin)	43
header.php (admin)	44
fileupload.php (admin)	45
template.php (admin)	46
customer.php (admin)	47
database.php	48
header.php (front end)	54
index.php (front end)	54
adminjava.js	54
adminstyle.css	59
frontjava.js	59
frontstyle.css	63

Project Analysis

Project Title: Email List Manager as well as Mass Email Designer and Sender.

Andrew owns the company by the name of 'Orange Lighting'. The company has always sold lights to large scale or personal building projects, acting as the in between for the lighting suppliers and the clients as well as helping design the interiors. However, as of recent the company's owner is looking towards moving forward into the blogging and educational industry. As part of this as well alongside his normal work, Andrew needs a system of collecting email addresses from forms to be stored in a database as well as being able to design emails which are sent to these collected email addresses in mass. He wants to be

able to select emails depending on factors such as where they signed up and for what as well as other factors like whether they have opened past emails. He wants a system that manages email lists and the customers that sign up for these email lists and an email design and sending interface that utilises these newsletters as well as other factors to target customers with emails. Andrew's main problem is he wants to have a great email design interface which sends emails that look good on all possible mail clients.

He wants the emails to look visually appealing on all email clients and have a user friendly, drag and drop system to design his emails.

Andrew wants this system to include some form of admin area and the ability to sign up for these emails in his website. Ideally, this admin area would be a closed off part of his website that requires login.

Mail marketing platforms such as Mailchimp and Moosend are very widely used by businesses and online presence. They are used to allow people to easily design and manage emails in a user-friendly way. However, the problem with these tools is that they are not bespoke which can lead to third party complications as well as a limit on the specific user-friendly nature of a complex system like Mailchimp. Andrew would like a much more bespoke system, made specifically for his needs at this time to help with efficiency as well as reduce third party dependency. Services like mail-chimp are paid for regularly and the pricing can add up. A bespoke system would not be so expensive in the long run. Andrew has specifically stated that he does not enjoy using third party systems such as Mailchimp.

The system will allow Andrew to do specific targeting of audiences with information that they give him and he collects.

Prospective Users

The users that are expected to interact with this project will be in later adulthood (30-50) and therefore are likely to be technically inept. Therefore the user interface needs to be extra rigorous when it comes to input validation and the way in which the user interacts with the program.

As well as this, the fact that the users will likely be technically adept will contribute to the importance of a very clear and user-friendly nature of the user interface.

The use of the admin area of the program will be Andrew who, considering he didn't like the interface of other similar mail management systems like Mailchimp, will want a very simple, efficient and easy to learn user interface. Andrew also struggles with eyesight so fonts and potential Images should be reasonably large and easy to read. On the other hand, Andrew does have some experience, while limited, with HTML and CSS syntax which might help him with understanding the user interface.

Researching the Problem

Mailchimp: an email management and design system.

Relevant features that Mailchimp possesses:

- Designing Email contents with extensive library of widgets and templates.
- Target audience using specific variables chosen by the user.
- Landing Pages, Social Ads, and Postcards etc (helping with marketing).
- Relatively complicated to implement for a user. User designs the form on the mail chimp website. Although they have options like landing pages etc.
- Being able to view the mail list and all the emails and clients that have been collected as well as being able to add more manually and delete them too.
- The handling of emails all happens on a separate website not integrated into your own website. This is slightly better with speed

Advantages/Disadvantages of Mail Chimp.

Advantages	Disadvantages
A lot of form customisation.	Extensive customisation means complication and complexity for the user's experience.

Extensive email visual customisation.	Subscription service, paying to keep the system every month.
Multiple ways of signing up for an email list.	Relying on third party service.

Interview

Questions and Answers

What is the problem you want to find a solution to?

Andrew doesn't like email software such as Mail Chimp and therefore wants something much simpler and easy to navigate so he can send emails to his clients. He wants a means of contacting the people that sign up to his lists. This is something he hadn't had before.

How do you want your clients to sign up to your newsletters?

Andrew wants clients to sign up through forms on his website.

How do you want to design your emails?

Andrew wants a simple and clean drag and drop system to design emails before they were sent to his clients.

What factors do you want that will allow you to choose who receives these emails?

Andrew wants to be able to target clients that have or haven't opened an email of his before. He also wants to be able to target different email lists as well as be able to send to specific emails as well as target specific client names.

Where do you want to design these emails?

Andrew wants to have some kind of admin area connected to his website that he can use to design the emails.

What information do you want your clients to give you? These can be used as factors?

Andrew wants the clients to give their email, name and profession. Although with some of the forms/lists they don't need to give their profession.

What are the different email lists you want to have featured on your website?

Andrew wants to have a main list which requires the name, email and profession of the client. He also wants a product recommendation list which does not require profession. The last list is a lighting updates list which does not also need to have profession.

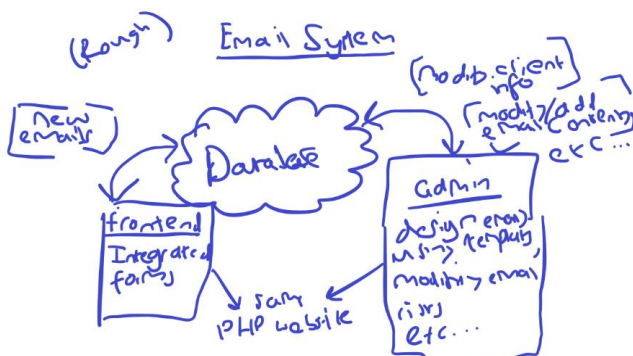
What are the different professions that your customers can select for sign up?

Designer, Architect, Retail Customer, Lighting Consultant, Electrician/Contractors.

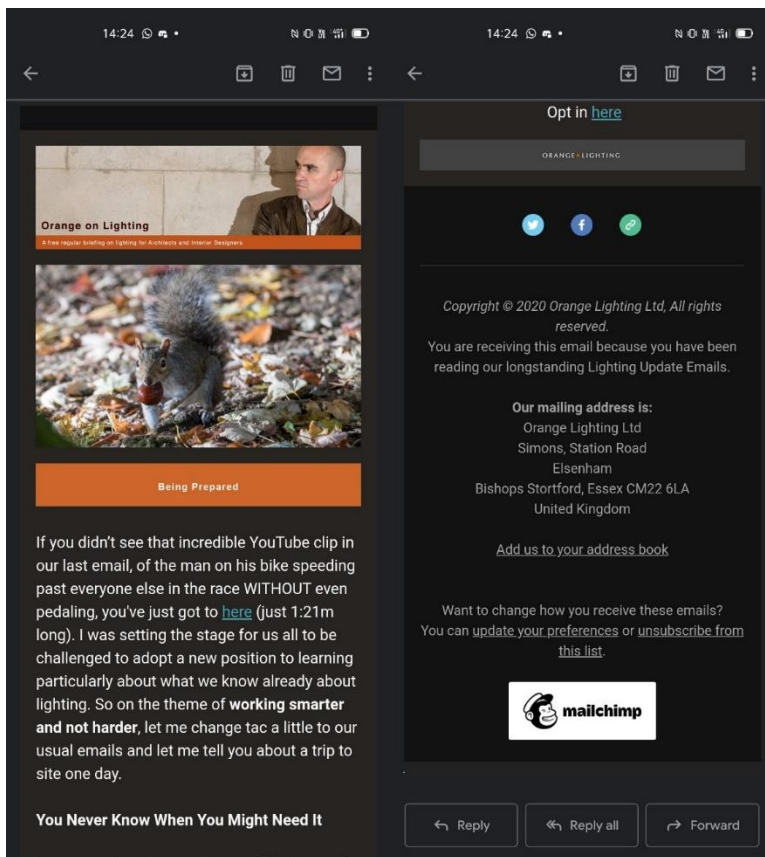
Are there any other features you would like implemented?

Andrew says he wants to have a way to create templates that he can save and use in future email designs. This is to help him with efficiency.

Initial System Design from Interview Information



Observation of current 'Orange Lighting' email design.



- Andrew currently uses mail chimp as his means of sending out customised emails to his clients
- The email uses Images as its means of adding graphical content and colour.
- Uses a very simple design with elements like text boxes, titles, Images and links.
- It uses a header and a footer to the email.
- The email has a very clear colour scheme using brown with orange highlights.
- In the footer, it tells the reader that the reason for their receiving the email (because they have already been reading emails).
- Readers have the ability to opt out of receiving emails.
- The email is watermarked by Mail Chimp – not so great for brand emails.

Limitations

The project being built is open to complexity and there will therefore be many possible future improvements and missing features. The limitations and/or missing features include:

- There is no ability to send out timed campaigns that use a tick because of the lack of means of hosting a server. (This would add extra cost).
- There is no way of Andrew adding more factors that could affect the receiver of the email.
- Due to the program being bespoke, Andrew will not have specific help from Forums and others outside of the developers of the program.
- There will not be collecting and storing of very personal information like date of birth (the users are signing up for an email list, not a login account).
- There will not be any way of Andrew adding new email lists and forms.

- The design capabilities will be limited due to a lack of available time for development e.g. changing the style of nodes e.g. the colour of text.

Objectives.

Must be included:

1: Andrew must have an admin area where he can manage his email system to perform tasks.

1.1: The admin area should only be reached through: websitename.com/admin.

1.2: The admin area should only be reached through a login page.

1.2.1: The admin area login must use a username and a password that only Andrew knows.

1.2.1: The password must be stored in an encrypted form.

1.2.2: If not logged in the browser will be redirected to the admin login page.

1.2.3: The username and password should be validated properly and there should be no way that anybody can access any areas of the admin area without being in a logged in session.

1.3: There must be the ability to log out from the admin area and the browser is redirected to the login page again.

1.4: Andrew must be able to design emails to be sent to clients. There must be a simple, interactive user interface in the admin area.

1.4.1: Andrew must be able to drag and drop elements into his central design and be able to edit these elements/nodes to make the email customised and look as he desires.

1.4.2: Elements of the design must be able to be shuffled around and rearranged.

1.4.3: Elements of the design must be able to be deleted and replaced.

1.4.4: The design nodes on the left must be labelled and easily understood as to their functionality so Andrew easily know what to drag over to the design construction box.

1.4.5: When these nodes are clicked the contents of the node should be able to be changed e.g. change text. The outline should be changed to suggest to Andrew that the node is in edit mode. When in edit mode the node should be able to be deleted. This should only be the case when dragged into construction box.

1.4.6: The emails that are sent must look good and be accessible in all web clients. Therefore, The program must take into the fact that some clients handle email contents differently such as the handling of CSS or even HTML tags. If the client does not accept the use of HTML, then remove the HTML tags and styling from the contents.

1.5: Andrew must be able to specify which emails receive the email he has designed/is designing by using a variety of factors.

1.5.1: Andrew must be able to choose what newsletter subscriptions receive the email.

1.5.2: He must also be able to use the name of the clients to decide who receives the emails.

1.5.3: Andrew must have the ability to only email clients if they haven't opened an email before.

1.5.4: Andrew must also be able to send emails to specific email addresses if he desires.

1.6: Andrew must be able to save and utilise email templates to give him a head start in his designs.

1.6.1: Andrew must be able to save the current state of his design as a template which can be retrieved at a later date.

1.6.1.1: A template name must be submitted before saving the template. There must be something inside the templates content and there must be a subject entered.

1.6.2: Andrew must be able to select a template from a drop down.

1.6.4: The current state of the email design will be lost therefore Andrew must be warned before the template is loaded.

1.7: It should only take Andrew 5 minutes to get to grips with the user interface.

2: There must be forms in the front end of Andrew's website that allows website visitors to sign up for his email lists.

2.1: There will be 3 email lists that can be signed up for on the front end of Andrew's website: Mail Email List, Lighting Updates and Recommended Products.

2.1.1: All lists will require name and email however the Main Email List will also require the profession.

2.2: All inputs will require rigid validation. The email must be in email format, the names must start with a capital.

2.3: There must input spam protection for the forms.

2.4: The users must be notified if their information is processed successfully and whether the information is not successfully processed. Notified if customer has signed up successfully, if Andrew has sent email successfully, if template has been saved successfully.

3: All SQL must be secure and protected from security risks like SQL injection. All information must be easily and quickly accessible from the database due to an efficient database design.

3.1: It must be a fully normalised database preventing any inaccurate data.

4: Andrew must be able to have a send email button which will then display 'sending...' until all the emails have been sent which then will display 'sent'.

5: When an email is opened the database must be updated so that it knows that the email has been opened by the customer.

Should be included:

1: When the user makes a big change, they should be notified which they then have to confirm their choices.

1.1: This includes when a template is deleted, when the page is refreshed, when pressing send email, when selecting a template.

2: There should be an 'add new email' form in the admin area similar to the form on the front page to allow Andrew to add his own emails to the email list.

3: Clients should be able to sign up for multiple newsletters simultaneously without causing any problems inside the program and in the database. The client should be able to sign up through different forms in the front end of the website.

4: Clients should not be able to input their email more than once which would spark an error message.

5: It should only take Andrew 5-10 minutes to fully grasp the user interface.

6: Andrew should be able to delete templates he has previously saved.

7: Andrew's emails should be able to be sent to all browsers and look good on all browsers, specifically Gmail and Outlook.

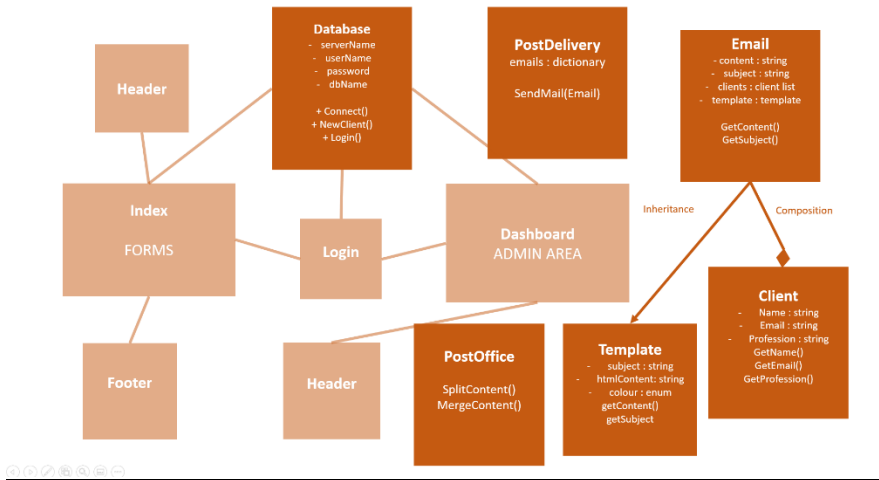
Could be included:

1: There could be the ability to manually add and or remove someone to the email list through the admin area. This should be a separate part of the admin area. This should be done through another simple user interface.

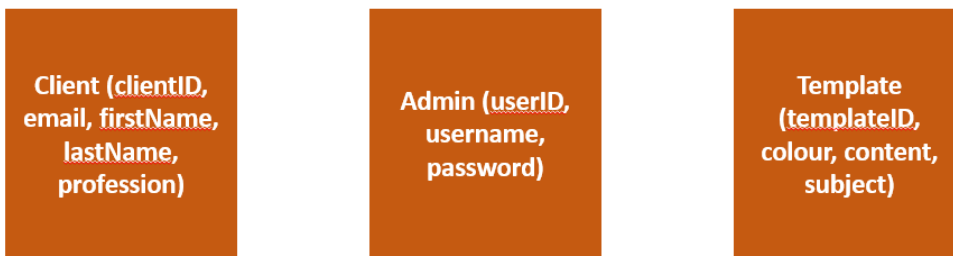
2: The clients could opt in and opt out of email list and email sending from the front end as well as from the emails being sent. This could be a link at the bottom of the emails being sent out as well as an option on the front page of the website (in the footer).

Initial Diagrams

Initial PHP Class Diagram



Initial Database System Design



Intended algorithms, techniques, and methods.

The program will use an object-oriented approach to my program. It will likely use:

- Arrays and possibly a dictionary
- Encryption
- Both selection and iteration (if, for, foreach,
- Communicating with a database (SQL).
- Namespaces
- Ajax
- PHP Objects
- PHP Class Objects.
- Constant Variables
- Private and Public Variables (get and set)
- File Upload.

- PHP sessions.
- PHP include/require
- Reading and Writing to a text file
- Regex

Programming Languages: Javascript, HTML, CSS, JQuery, SQL, PHP.

Prototyping parts of my Program

I have decided to prototype the parts of my program that I am not 100% sure I can do or do not have a huge amount of experience doing.

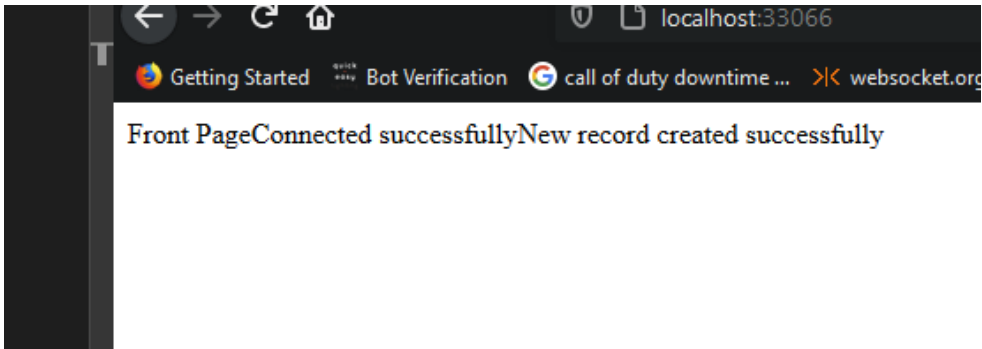
Communicate with the database using SQL.

I will be using MySQL workbench on my Home PC to store and communicate with my database. However this algorithm is not protected from SQL injection and therefore a new method will have to be found to achieve this.

```

... index.php \ X frontadmin.php index.php admin header.php admin # a
index.php
1  <?php namespace frontend;
2
3  echo "Front Page";
4
5
6
7  $servername = "localhost";
8  $username = "root";
9  $password = "123456789";
10 $dbname = "emailsystemdb";
11
12 // Create connection
13 $conn = new \mysqli($servername, $username, $password, $dbname);
14
15 // Check connection
16 if ($conn->connect_error) {
17     die("Connection failed: " . $conn->connect_error);
18 }
19 echo "Connected successfully";
20
21 $sql = "INSERT INTO customers (firstName, lastName, email, profession)
22 VALUES ('John', 'Doe', 'john@example.com', 'designer')";
23
24 if ($conn->query($sql) === TRUE) {
25     echo "New record created successfully";
26 } else {
27     echo "Error: " . $sql . "<br>" . $conn->error;
28 }
29
30 $conn->close();

```



1S
S
1 Keys
S
S

Result Grid | Filter Rows: | Edit: | Export/Impo

	idCustomer	firstName	lastName	email	profession
▶	1	John	Doe	john@example.com	designer
•	NULL	NULL	NULL	NULL	NULL

S

Code Help

(w3Schools, PHP MySQL Insert Data, n.d.) : https://www.w3schools.com/php/php_mysql_insert.asp

Track whether an email has been opened or not.

```
$body = '<p><strong style="background-color:#000;">Hello!</strong> this is my first test email</p>';

// $tracker = 'localhost:33066/admin/tracker.php?log=true&subject=' . urlencode( 'Test Email' ) . '&user=' . urlencode( 'josiahorange02@gmail.com' );
$tracker = 'localhost:33066/tracker.php?log=true&subject=' . urlencode( $subject ) . '&user=' . urlencode( $email );

$body = $body . '';

// Content

> tracker.php
<?php namespace admin;

if( !empty( $_GET['log'] ) && $_GET['log'] == 'true' && !empty( $_GET['user'] ) && !empty( $_GET['subject'] ) ) {

    //fwrite($myfile, $_GET['user'] . "\n");

    file_put_contents( 'newfile.txt', strval($_GET['user'] . "\n\n"), FILE_APPEND);

    //WILL BE STORED IN DATABASE
}
```

Tracker link is placed as the src for an image in every email. This Image does not appear but will connect to the src address giving it the information about the client, so the system knows that the client has opened the email and processed this image. This information can then be used later down the line for targeting emails.

Test Dragging and dropping.

Dragging and dropping using HTML/CSS will be essential to building my mail creation interface. Users will be able to drag elements into their email template.

Login Page:



Login Page:



```
<script>
function allowDrop(ev) {
    ev.preventDefault();
}
function drag(ev) {
    ev.dataTransfer.setData("text", ev.target.id);
}
function drop(ev) {
    ev.preventDefault();
    ev.target.appendChild(document.getElementById(data));
}
</script>
<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
<br>

<?php
echo "<a href='". $url . "frontadmin.php"><button>Login</button></a>";
```

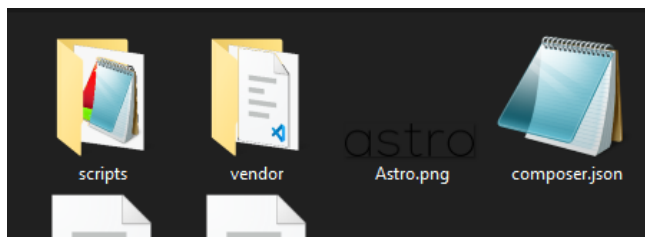
Code Help

(w3schools., n.d.) : https://www.w3schools.com/html/html5_draganddrop.asp

Being able to send emails using php using PHPMailer. Allows attachments, embedded images and more customisation.

PHPMailer is a package of code that allows you to do much more with php when it comes to sending mail.

To install PHPMailer, I had to install **Composer API – a dependency manager**. This is a dependency manager for PHP allowing me to install other php files/ extensions into my project. This was installed through Command Prompt. The installed files were a vendor and scripts folder as well as a composer file (for the dependency manager). These contained scripts containing code and classes in which I could reference from my code without too much extra effort.



```
Warning: Undefined variable $stucker in F:\Apache24\htdocs\admin\postoffice.php on line 63
2020-11-19 17:12:54 SERVER -> CLIENT: 220 smtp.gmail.com ESMTP u23am736812wmc:32 - gsmtp
2020-11-19 17:12:54 CLIENT -> SERVER: EHLO localhost
2020-11-19 17:12:54 SERVER -> CLIENT: 250-smtp.gmail.com at your service, [2.26.136.240]250-SIZE 35882577250-8BITMIME250-STARTTLS250-ENHANCEDSTATUSCODES250-PIPELINING
2020-11-19 17:12:54 CLIENT -> SERVER: STARTTLS
2020-11-19 17:12:54 SERVER -> CLIENT: 220 2.0.0 Ready to start TLS
2020-11-19 17:12:54 CLIENT -> SERVER: EHLO localhost
2020-11-19 17:12:54 SERVER -> CLIENT: 250-smtp.gmail.com at your service, [2.26.136.240]250-SIZE 35882577250-8BITMIME250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTORCHUNKING250-SMTPUTF8
2020-11-19 17:12:54 CLIENT -> SERVER: AUTH LOGIN
2020-11-19 17:12:54 SERVER -> CLIENT: 334 VXNlcm9ubWU6
2020-11-19 17:12:54 CLIENT -> SERVER: [credentials hidden]
2020-11-19 17:12:54 SERVER -> CLIENT: 334 UGFzc3dvcmQ6
2020-11-19 17:12:54 CLIENT -> SERVER: [credentials hidden]
2020-11-19 17:12:54 SERVER -> CLIENT: 250 2.1.0 OK u23am736812wmc:32 - gsmtp
2020-11-19 17:12:54 CLIENT -> SERVER: MAIL FROM: <[redacted]@gmail.com>
2020-11-19 17:12:54 SERVER -> CLIENT: 250 2.1.0 OK u23am736812wmc:32 - gsmtp
2020-11-19 17:12:54 CLIENT -> SERVER: RCPT TO: <[redacted]@gmail.com>
2020-11-19 17:12:54 SERVER -> CLIENT: 250 2.1.5 OK u23am736812wmc:32 - gsmtp
2020-11-19 17:12:55 SERVER -> CLIENT: 354 Go ahead u23am736812wmc:32 - gsmtp
2020-11-19 17:12:55 CLIENT -> SERVER: Date: Thu, 19 Nov 2020 17:12:54 -0000
2020-11-19 17:12:55 CLIENT -> SERVER: To: Josiah <[redacted]@gmail.com>
2020-11-19 17:12:55 CLIENT -> SERVER: From: Josiah <[redacted]@gmail.com>
2020-11-19 17:12:55 CLIENT -> SERVER: Reply-To: Josiah <[redacted]@gmail.com>
2020-11-19 17:12:55 CLIENT -> SERVER: Subject: Test Email
2020-11-19 17:12:55 CLIENT -> SERVER: Message-ID: <[redacted]@localhost>
2020-11-19 17:12:55 CLIENT -> SERVER: X-Mailer: PHPMailer 6.1.8 (https://github.com/PHPMailer/PHPMailer)
2020-11-19 17:12:55 CLIENT -> SERVER: MIME-Version: 1.0
2020-11-19 17:12:55 CLIENT -> SERVER: Content-Type: multipart/alternative;
2020-11-19 17:12:55 CLIENT -> SERVER: boundary="b1_PjM9v3SKDS3Pa7jqHgfW6ie7SLYOzCzhakSFUI2o"
2020-11-19 17:12:55 CLIENT -> SERVER: Content-Transfer-Encoding: 8bit
2020-11-19 17:12:55 CLIENT -> SERVER:
2020-11-19 17:12:55 CLIENT -> SERVER: This is a multi-part message in MIME format.
2020-11-19 17:12:55 CLIENT -> SERVER:
2020-11-19 17:12:55 CLIENT -> SERVER:
2020-11-19 17:12:55 CLIENT -> SERVER: -b1_PjM9v3SKDS3Pa7jqHgfW6ie7SLYOzCzhakSFUI2o
2020-11-19 17:12:55 CLIENT -> SERVER: Content-Type: text/plain; charset=us-ascii
2020-11-19 17:12:55 CLIENT -> SERVER:
2020-11-19 17:12:55 CLIENT -> SERVER: Hello! this is my first test email
2020-11-19 17:12:55 CLIENT -> SERVER:
2020-11-19 17:12:55 CLIENT -> SERVER: -b1_PjM9v3SKDS3Pa7jqHgfW6ie7SLYOzCzhakSFUI2o
2020-11-19 17:12:55 CLIENT -> SERVER: Content-Type: text/html; charset=us-ascii
2020-11-19 17:12:55 CLIENT -> SERVER:
2020-11-19 17:12:55 CLIENT -> SERVER: <p><strong style="background-colour:#000;">Hello!</strong> this is my first test email</p><img alt="" src="" width="10" height="10" border="0" />
2020-11-19 17:12:55 CLIENT -> SERVER:
2020-11-19 17:12:55 CLIENT -> SERVER: -b1_PjM9v3SKDS3Pa7jqHgfW6ie7SLYOzCzhakSFUI2o--
2020-11-19 17:12:55 CLIENT -> SERVER:
2020-11-19 17:12:55 SERVER -> CLIENT: 250 2.0.0 OK 1605809976 u23am736812wmc:32 - gsmtp
2020-11-19 17:12:55 SERVER -> CLIENT: 221 2.0.0 closing connection u23am736812wmc:32 - gsmtp
Message has been sent
```

I tested PHP mailer using my own email. Here is some of the code. I had to call the PHPMailer scripts from inside my code and I had to use Gmail's smtp information to send the emails. The good thing about PHPMailer is it allow you to easily send emails as html and if the destination mail client doesn't support html then I just remove the html tags and send it as basic string. Another great thing is that it allows you to send attachments.

```
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;

class PostOffice{

    static function SendMail() {

        // Load Composer's autoloader
        require 'vendor/autoload.php';

        // Instantiation and passing `true` enables exceptions
        $mail = new PHPMailer(true);

        try {
            //Server settings
            $mail->SMTPDebug = SMTP::DEBUG_SERVER;
            $mail->isSMTP();
            $mail->Host      = 'smtp.gmail.com';
            $mail->SMTPAuth  = true;
            $mail->Username   = 'myemail';
            $mail->Password   = 'password';
            $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
            $mail->Port= 587;

            //Recipients
            $mail->setFrom('myemail', 'Josiah');
            $mail->addAddress(myemail, 'Josiah');
            $mail->addReplyTo('myemail', 'Josiah');

            $body = '<p><strong style="background-colour:#000;">Hello!</strong> this is my first test email</p>';
```

```

        // Content
        $mail->isHTML(true);
        $mail->Subject = 'Test Email';
        $mail->Body     = $body;
        $mail->AltBody = strip_tags($body);
        $mail->send();
        echo 'Message has been sent';
    } catch (Exception $e) {
        echo "Message could not be sent. Mailer Error: {$mail->ErrorInfo}";
    }
}
}

```

Code Help

(GitHub, n.d.) : <https://github.com/PHPMailer/PHPMailer>

This code above is the default recommended code given by PHPMailer on Github. The code has been copied and then can be altered to fit the specific program.

Documented Design

The program will have a fairly complex structure. It utilises two ways of getting user input information to code. One being PHP **include/require** and the other being **Ajax**.

Ajax is used to pass user input information, primarily after a form submission, that will be dealt with behind the scenes. E.g. sending an email.

PHP 'include' allows for the utilisation of the code from the included file from within the main file. E.g. calling database functions from included database.php to get information from the database.

The structure utilises **namespaces** which helps to separate the front end from the admin side of the PHP structure.

Javascript / JQuery

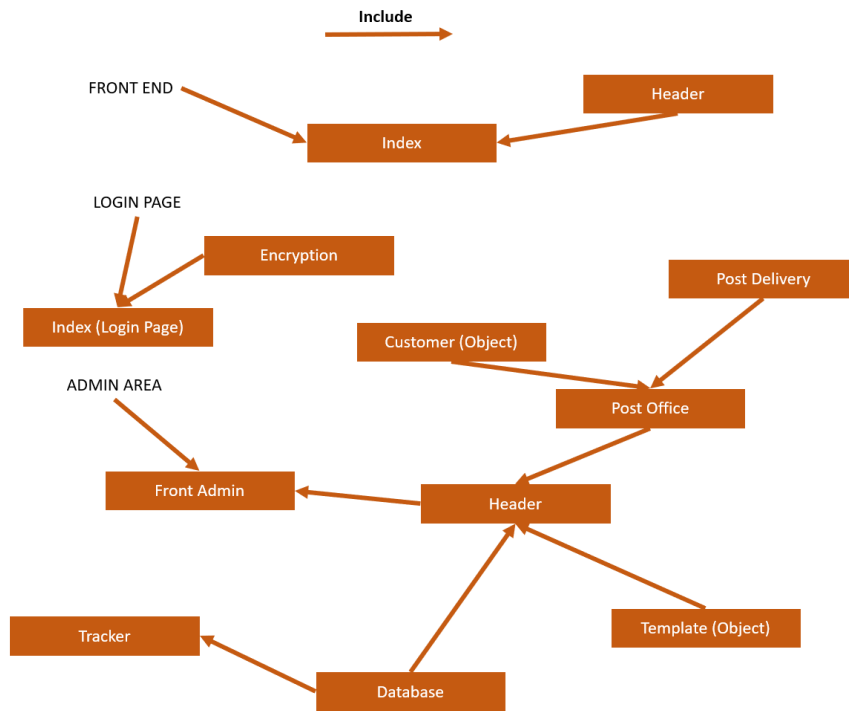
Javascript will be accessed through two ways – one from within the PHP file, calling from the HTML <script> tag. This is mainly for Ajax calls. The other will be from a separate JS file called from the header file.

Diagram Designs and System Structure

PHP Structure Diagram (file stacking)

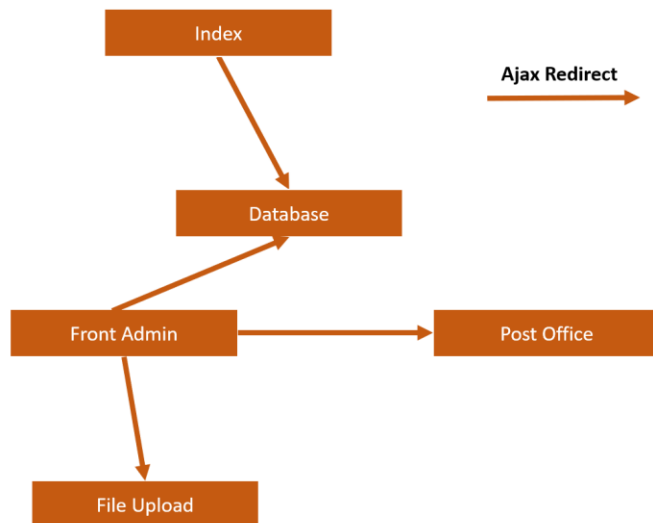
This is not all the files used and all file associations. These are the connections between files that use the 'include' syntax to stack the php files. When a file is included, it allows for the other php files to utilise the code inside the included file.

The Header php file, not only allows for a consistent header styling but also is the users access to all the back end code.



Ajax Structure Diagram

These are the ajax redirects for when I need to deal with information inputted by the user without a change of page on the user side.



User Interface Design

Front End



Sign Up Forms

The sign up forms shown on the front end are not the same as the ones that will be used by Andrew. These are just for development purposes before the system is moved over to Andrew's website.

Profession: A select input with options chosen by Andrew. Customers can choose one of these if signing up for the Main Email List.

First Name, Last Name: These two string inputs must start with a capital letter.

Email: This string input must fit the email format.

Main Email List

First Name

Last Name

Email

Profession

3 Forms

There are 3 forms to sign up to 3 different email lists/newsletters. These are all shown on the front page although realistically they would be scattered throughout the website.

Lighting Recommendation

First Name

Admin Area

Email Subject: A String That will be used as the Email Subject when sent

Template Select: A Selection Input which displays all saved templates

Template Name: A string that represents the template being saved



Construction Nodes

These nodes can be dragged into the construction box on the right. Each node does a different thing specified by their titles. These nodes can be dragged and edited but cannot be deleted and cannot be overwritten.

Title

Text

Space

Image

Submit

Image Upload
Andrew browses his computer for an image and then once selected, the image gets uploaded into the images folder and then the preview image refers to the newly uploaded image. This image is then embedded into the email when sent.

Email Subject

Template: 1

Template Name

Choose Recipients

Send Email

Select Template

Save as Template

Preview

Construction Controls

Templates
This is where Andrew (user) can both save the current state of the construction box and subject as a template, entering a template name as well as well as be able to load previously saved templates to bring back the saved state

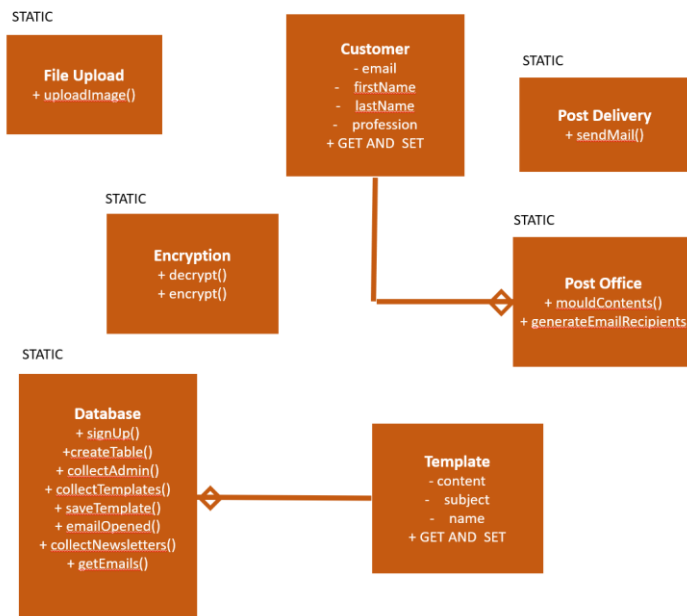
Recipients
Andrew must click the choose recipients button to open the pop up which allows Andrew to change the factors determining who receives the email

Construction Box

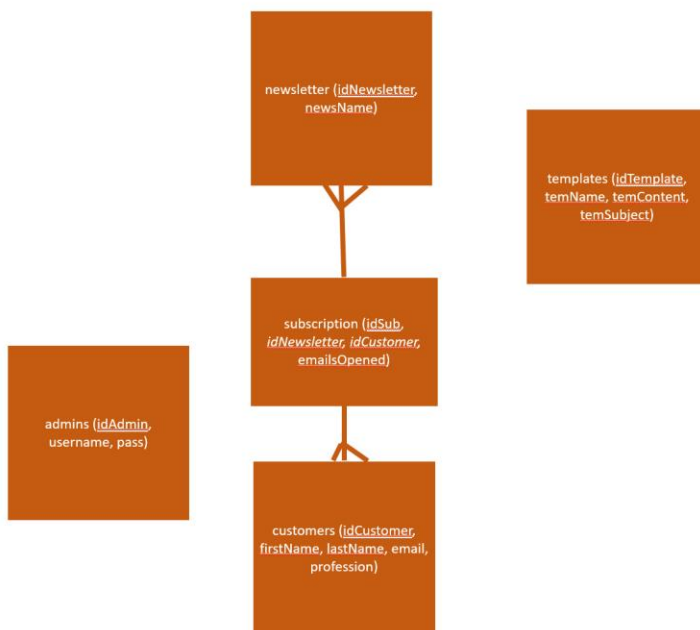
In the construction box, when a node is dragged in, a new available drop spot is created so another node can be dragged in. Nodes can be dragged around, swapped, edited and deleted.

Revised Class Diagram

This is not all the files and code but these are the classes that are used. Interactions only shown when the classes are referenced from inside the other class.



Revised Database Structure



Fonts & Images

Fonts and Images get separate folders in the admin area. When an Image gets uploaded it gets uploaded into the images folder. All fonts being used in the website are stored in the fonts folder.

The user interface will be using Open Sans as the font on the admin area. This allows for a more pleasing, user friendly and readable admin area for Andrew. This will be stored in a fonts folder and references in adminstyle.css.

Password Encryption System

The encryption system is simple. All the encryption algorithms are performed in encryption.php. The key used in the encryption is stored in encryptionkey.txt on the web server (localhost during development) and the cypher text is stored in the database. When the cyphertext needs to be decrypted on login, the key is retrieved from the text file and used to decrypt the cypher text retrieved from the database. The plain text is then encrypted again with a new random key, the same length as the plain text and the cycle continues.

Email Opened Tracking System

When an email is sent, there will be an image attached with the src of tracker.php on the web server. The src will pass in the email address of the customer who has opened the email and what newsletter the email is associated with. tracker.php will use this information to update the database, incrementing the emailsOpened value as to know an email has been opened by the customer. This information can then be used a factor/variable by Andrew to specify email recipients in the future.

Image Upload and Embedding System

The image node will allow Andrew to browse and select an image from his computer. When this Image is selected, the image's information including its address on the computer is sent through ajax to fileupload.php where it uploaded to the images folder. The preview in the admin area is then updated to reference the newly uploaded local file. When the email is sent, the images information is passed through with the other email information so it can be embedded by PHPMailer.

Template Saving and Selecting

The template system is simple. Andrew enters a template name and makes sure the subject and content is filled in in the construction box in the admin area. The template is saved into the database using the algorithm in database.php. When selecting the template the templates and their associated content, subject and name are retrieved from the database and then once one has been selected, the content and subject is placed into the construction box.

Key Files (Not all Files).

frontadmin.php	This is the admin area php file. This is the file that Andrew will spend most of his time. It loads
----------------	---

	all the user interface and brings together a lot of the functionality of the admin area.
index.php (admin)	This is the login page. The file is stored in the admin folder and is therefore called when the /admin/ is called in the URL. This is where Andrew can login to his admin area.
customer.php & template.php	These files are just classes which are used as objects to store information. customer.php is used to store information about the customer and template.php is used to store information about all the templates.
fileupload.php	This file contains the code used to upload an image selected by Andrew using information about the images local location and name.
postoffice.php and postdelivery.php	These two files contain the code use to prepare and send the emails. postoffice is used to prepare the email and postdelivery is used to send it.
encryption.php	This file does what it says on the tin. It contains the encryption and decryption methods used to encrypt and decrypt the admin password.
adminjava.js	adminjava.js contains a lot of the javascript needed for the admin area to function.
database.php	database file contains all the SQL algorithms to communicate with the database.
header.php (admin)	This file is the inbetween for most of the files used in the admin area. These files are included through the header file which is called by frontadmin.php. It also loads in the header and the css and js files.
Index.php (front)	index.php on the front end contains all the forms needed for customers to sign up for the newsletters.
encryptionkey.txt	This text file is used to store the encryption key that was used to encrypt the current cypher text stored in the database.

Key Variables (Not all Variables)

Variable Name	Description	Type
	frontadmin.php	
templates	This is a store of all the templates that have been created and stored on the database. Each one is assigned a key in a php array. An associative array would not work because each template needs its own key value. Templates is an array of the Template class instances which allows the storing of multiple values associated with each template (content and subject).	Array of Object Instances

newsletters	This array variable stores all the newsletter names so they can be displayed in the recipients popup and can be chosen by Andrew.	Array
recipientBox	This is the JavaScript object for the popup box that shows when choosing the recipients of the email that has been designed. Allows for the box to be visible or not.	JS Object
subject and contents	these are temporary variables to store the contents and subject that has been designed.	String
tempContent & tempSubject	these variables store the template content and subject.	String
recipientInfo	recipient info stores the factors determining who receives the email being sent.	Array
images	this dictionary array stores the image information (the image id and address) so it can be embedded into the email.	Dictionary Array
	header.php (admin)	
details	this variable stores the admin information retrieved from database that needs to be typed into the login form to allow the user to access the admin area. Used to validate.	Array
	postdelivery.php	
bodyContent and subject	This the body and subject that has been prepared and is ready to be sent to the specified users.	String
mail	this variable an instance of the PHPMailer package allowing for the sending of the email.	Object Instance
customer	this object variable stores the customer information.	Object
images	this dictionary array contains all the necessary images info so PHPMailer can embed the images.	Dictionary Array
	postoffice.php	
content and subject and name	these are the public properties of the Template Class. They store the associated content, subject and name with the given template.	String
customers	stores all the customer information of the emails that will receive the email being sent.	array
sql	The sql string that has been generated that will get all the customers information required that will receive the email being sent.	string
recipientInfo	This array stores all the recipient factors that will influence who receives the email being sent.	
	template.php	
email, firstName, lastName and profession	These variables store the information associated with the customer object instance.	String
	customer.php	
dbservername, dbusername, dbpassword, dbname:	these are constant global variables that store the information allowing for connection to the database.	Constant Strings
	database.php	
sql	this variable stores the sql statement that will be executed. It then stores the sql after being prepared for execution.	String

conn	this variable stores the mysql connection, allowing for communication with the database.	Object Instance
result	for select statements which result in the return of information, the result variable holds this resulting data which is then distributed using loops.	Object
	encryption.php	
firstname, lastname, email, profession, identifier	these are all javascript variables temporarily holding the information submitted by clients. encryption.php	String
cypherTxt	The cypher text currently in process.	String
plaintext	The password in unencrypted form	String
rndKey & key	The key being used to encrypt or decrypt the password.	string

Regex Design

Regex strings can be used to validate a string to fit the desired format. This helps to create defensive programming and helps prevent any SQL attacks. It also just allows for my consistency with inputs.

These are my Regular Expression designs.

Email Validation

```
^[a-z0-9]+@[a-z0-9]+(\.[a-z]+)+$
```

This regular expression allows for basic email validation. There must be either a string of lowercase letters or numbers and then an @ symbol and then another string of lowercase letters and number and then the sequence of dot and then lowercase string which can be repeated.

Name Validation

```
^[A-Z][a-z]*$
```

This a very simple regular expression that requires the first letter to be a capital letter and then the rest to be lowercase letters

SQL Design

Get Templates

```
SELECT temName, temContent
FROM templates
```

Save Templates

```
INSERT INTO (temName, temContent, temSubject)
VALUES ( subject, content, subject )"
```

Get Admin Information

```
SELECT username, pass
FROM admins
```

Get Emails

```
SELECT customers.email, customers.firstName, customers.lastName, customers.profession
FROM customers
INNER JOIN subscription
ON customers.idCustomer = subscription.idCustomer
WHERE customers.email <> "
AND
subscription.idNewsletter =
(SELECT idNewsletter
FROM newsletters
WHERE newsName = 'email list name')
AND
subscription.emailsOpened > 0
AND
customers.firstName LIKE '%firstname%'
OR customers.lastName LIKE '%lastname%'
```

Some Key Algorithms (Structured English + Pseudocode)

Some of the pseudocode used is mixed with some structured English – this is often used when language specific syntax would be required.

Communicating with Database (SELECT Templates)

This algorithm is an example of a SELECT SQL statement. In this example, the template names and content are being retrieved from the database.

```
conn = connect to database using username, servername, password and name.
if (error has occurred with during connection)
    print "connection failed"
sql = "SELECT temName, temContent FROM templates"
result = query using sql and conn
if (number of rows in result is more than 0)
    while ( row = fetch next row from result )
        templates = push into array new Template(row[temName], row[temContent])
else
    print "no results"
close connection
```

return templates.

Communicating with Database (INSERT Template)

This algorithm is an example of an INSERT SQL statement that inserts data into the database. In this example, template name and content are being inserted into the database.

```

conn = connect to database using username, servername, password and name.
if (error has occurred with during connection)
    print "connection failed"
sql = "INSERT INTO (temName, temContent) VALUES ( subject, content )"
if ( the query was successful )
    print "New record created successfully"
else
    print "error has occurred"
close connection
Post Office Preparing Content for Email delivery
If ( post[subject] and post[content] from ajax call )
    content = MouldContents(post[content])
    PostDelivery.SendMail(content, post[subject])

```

Removing the attributes inside the generated html for website content to be ready to be sent (function mouldContents)

This very simple algorithm stripes the construction box content of all the unnecessary elements and tags before it is sent as email content.

```

MouldContents(content)
    content = stringreplace(*all attributes but style* with "")
    return content

```

Generate SQL to get the Email Recipients (function generateRecipients)

This algorithm uses the factors Andrew has inputted to generate the appropriate SQL to get the emails that will receive the email being sent.

```

generateRecipients(recipientInfo)
    If ( recipientInfo[0] == "" )
        sql = "SELECT customers.email, customers.firstName, customers.lastName, customers.profession
        FROM customers INNER JOIN subscription ON customers.idCustomer = subscription.idCustomer
        WHERE customers.email <> ""
        if ( recipientInfo[1] != "None" )
            sql = sql + " AND "
            sql = sql + "subscription.idNewsletter = (SELECT idNewsletter FROM newsletters WHERE
            newsName = " + recipientInfo[1] + ")"

```

```

If ( recipientInfo[2] == "true" )
    sql = sql + " AND "
    sql = sql + "subscription.emailsOpened < 1"

If (recipientInfo[3] != "")
    sql = sql + " AND "
    sql = sql + "customers.firstName LIKE '%" + recipientInfo[3] + "%' OR customers.lastName LIKE '%"
    + recipientInfo[3] + "%"

else
    sql = "SELECT email, firstName, lastName, profession FROM customers WHERE email LIKE '%" +
    recipientInfo[0] + "%"

return sql

```

Designing the Email – Nodes and drag and drop

```

function deletenode()
    remove targetnode html

function drag()
    set transfer data from dragging node

function insertafter(referenceNode, newNode)
    insert node after reference node

function drop()
    get transferdata from original node being copied
    if ( node being dragged is already in construction box )
        if ( target drop position doesn't have any node attached )
            append node to target drop position
        else
            nodecopy = copy original node
            nodecopy2 = copy node currently inside target position

            parentnode1 = parent of original node
            parentnode2 = parent of node currently in target position

            nodecontrols1 = a copy of the controls for the original node
            nodecontrols = a copy of the controls for the parent node currently in target position.

            empty the inner html for the parent elements to remove the nodes

            append nodecopy to parentnode2
            append nodecopy2 to parentnode1
            *this swaps the nodes*

    Else
        nodecopy = copy of original node
        counter = get counter from the html

```

```

increment the counter in the html

add 'customnode' to the nodecopy

if ( the id of the original node is the image node )
    add counter to the id of nodecopy
    add counter to the id of the image element in nodecopy
else
    add counter to the id of nodecopy
    if ( the target drop position contains a node )
        append nodecopy to the drop position

        div = create div
        div inner html = '<div class="customnode" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>'

        append div to construction box ready for a new node insertion

        divcontrol = create div
        id of divcontrol = 'nodecontrol' + counter
        class of divcontrol = 'nodecontrol'
        nodecontrol inner html = '<button onclick="deletenode(event)">Delete</button>'
        insert the control div after the target drop position

```

Adding a new image to dictionary of images.

This algorithm processes the image just selected in the construction box. It gets the data required about the image being uploaded as well, sends it off for uploading and adds it to the dictionary so it can be embedded once the email is sent.

```

addImage(formId)

form = get form by formId
imgPath = get value of image upload

imgPaths = split image path by /
imgName = get final split from imgPaths array

Send Image upload form post information using 'form' to fileupload.php

change image src to new uploaded image src

imageId = "nodeImage" + formId
imageAddress = new address of uploaded image

images[] = images[] + [ imageId => imageAddress ]

```


Encryption Algorithm

The encryption algorithm encrypts the password before It is stored in the database and stores the generated key into a text file stored on the server. (or localhost).

length = length of plain text.

characters = !"#\$%&'()*+,-

./0123456789;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

for (i = 0 to length of plain text)

 rndKey = rndKey + random letter using characters

put rndKey into txt document

ascii = array

for (i = 0 to length of rndKey)

 ascii [i] = plaintext[1] as ascii + rndKey[i] as ascii

 while (ascii[i] > 126)

 ascii[i] = 33 + ascii[i] – 126

cypherTxt = "

for (i = 0 to ascii length)

 cypherTxt = ascii[i] as char

return cypherTxt

Testing

Test Plan

 Erroneous data
 Normal data
 Boundary data

Test No.	Action	Data	Reason/Explanation	Expected Result
1	Drag nodes over to construction box and swap around inside construction box.		Shows successfully designing an Email before the email is sent.	Nodes appear in construction box if dragged over and when dragged over one another can be swapped back and forth. When a new node is added to construction box, a new drop position appears.
2	Browse for Image and then select an Image for upload from local storage.	File: A png Image	Successfully upload Image which is previewed in the construction box	Image is uploaded into images folder and then referenced inside the admin area for a preview of the Image being uploaded.
3	Enter the url into browser that would be followed if image	email = testemail@est.com	Successfully Record in database when Email Opened.	In customer table for the customer's to Main Email List row in the database, the emailOpened attribute

	was loaded in email. (Because localhost is used to store tracker.php, the tracker file cannot be referenced in email).	news = 'Main' Subject = 'TestSubject'		will be incremented which can then be used as a factor.
4	Input factors into form and then send email.	Newsletter: 'Lighting Updates'	Successfully Send and Receive Email using recipient Factors.	The email(s) fitting the factors chosen (part of the Lighting Updates newsletter) will receive the email. The prepared email signed up to Lighting Updates will receive the email.
5	Input incorrect recipient factors before sending email.	Newsletter : 'Main'	Send email to incorrect factors so email is not received.	Email is not received because they are not part of the Main newsletter. The email not received by prepared email.
6	Input data into 'Main Email List' sign up form.	email = 'testemail@t est.com' firstName = 'Test' lastName = 'Name'	Successfully sign up to Newsletters/ Email Lists from the front end.	New customer is inserted into database (customers table) and new subscription inserted (subscription table) connecting customer with the Main Email List.
7	Enter incorrect data into 'Main Email List'.	email = 'testemail@t est.com' firstName = 'test' lastName = 'Name'	Fail to Sign Up for Newsletter/Email list from front end.	Error message warning user the incorrect data has been entered and the sign up has failed.
8	Sign up for a new email list using previously used email on a different newsletter (Recommended Products)	email = 'testemail@t est.com' firstName = 'Test' lastName = 'Name' Newsletter = Recommended Products	Successfully sign up for another new email list using same email without the same customer being inserted into database twice.	Connection between previously inserted customer and new Lighting Updates is created but customer table not changed
9	Enter a template name before saving template. (with content and subject entered)	templateName = 'Test Template' content and subject = any input	Successfully Save a custom template.	New template inserted into database (template table) and when page is refreshed, template comes up for selection.
10	Enter a template name and subject but do not enter any	templateName and subject =	Fail to Save a custom template.	Error message comes up saying that no all contents has been entered.

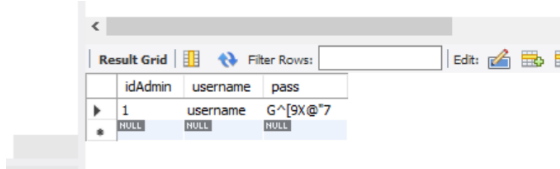
	content and then save template.	'Template Fail content = no input'		
11	Enter correct admin login information	'username = 'username' password = 'password'	Successfully Login to Admin area.	Redirected into the admin area.
12	Enter incorrect admin login information.	'username = 'username' password = 'wrongpassword'	Fail to login to admin area.	Warning message showed and remain on login page.
13	Enter URL to admin area (localhost::33066/admin/frontadmin.php)	'localhost::33066/admin/frontadmin.php' into browser	Denied from admin files and admin area If not logged in.	User redirected to the admin login page.
14	Browse for non-Image and then upload.	File: A word document.	Uploading incorrect input for image	The file is not uploaded and nothing is previewed in admin.
15	Enter URL to admin area (localhost::33066/admin/)	'localhost::33066/admin/' into browser	Automatically taken to admin area if already logged in and attempting to go to login page.	User directed to the frontadmin.php admin area user interface.
16	Send Email with Image and text and subject to Gmail and Outlook.	Subject input, Image and text in contents.	Emails can be sent to a variety of mail services specifically, Gmail and Outlook.	All will receive the email with the correct styling and the Image loads.

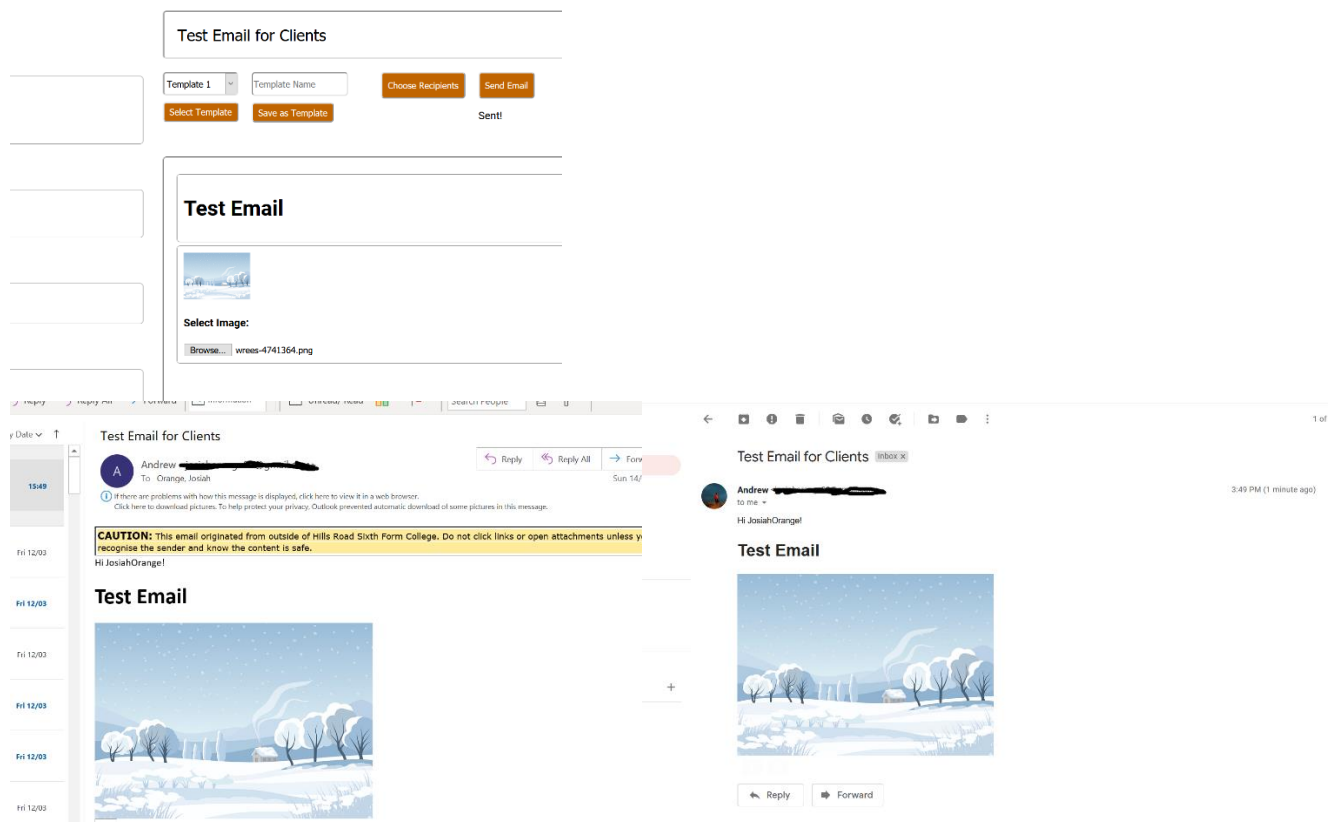
Testing Video & Results

<https://youtu.be/S--ccbK2nd4>

----- As Expected ----- Not Entirely Expected ----- Not expected

Test No.	Actual Result
1	Success. The nodes can be dragged and dropped, swapped and deleted. A drop position is created when a new node is dropped in. Occasionally buggy such as certain parts of the node wont react when node dropped on. [3:30]
2	Success. The Image is uploaded into image folder and then previewed in admin area.

	[4:58]
3	The emailOpened value is successfully incremented. Unfortunately this cannot be done through an Image references both because of the tracker.php being hosted locally and some mail clients like gmail blocking image src. [6:21]
4	Successfully sends email according to the factor being signed up for lighting updates. Both the text and Image is sent through to the email. The email that was signed up just previous into the lighting updates form receives the email. [3:55]
5	Email not received because the email is being sent to everyone signed up to the Main Email List and the receiving email is not. Receiving email previously signed up in video. [4:36]
6	Success. Customer successfully signs up for the Main Email List once all inputs are validated. [0:14]
7	Error message sent to user telling them to input the correct value formats. [0:00]
8	Successful. New subscription link is added but not a new repeated customer. [0:36]
9	Successfully saved template which can then be selected and loaded into construction box. [2:17]
10	Fails to save template. Warning pop up asking user to fill in all contents. [2:01]
11	Successfully logs into admin area using the correct username and password (the password being decrypted). Password is re encrypted.  [1:20]
12	Failed to log in. Error message shows at the top. [1:04]
13	Successfully redirected back to admin login page. [1:32]

14	<p>The word document selected is not uploaded and not previewed, nothing is previewed. However the preview is attempted so the preview box flashes as it looks for the file uploaded – which it does not find. No errors occur but it looks a bit strange.</p> <p>[6:04]</p>
15	<p>Successfully redirected to admin area.</p> <p>[1:50]</p>
16	<p>Emails were successfully received by Gmail and Outlook – as seen by these screen shots. The Image and the text was displayed and with the correct styling.</p>  <p>The screenshots show the email creation interface and the resulting emails in Gmail and Outlook. The interface includes a 'Test Email for Clients' form with fields for 'Template 1', 'Template Name', 'Choose Recipients', 'Send Email', 'Select Template', and 'Save as Template'. Below the form is a 'Test Email' preview showing a winter landscape image and a 'Select Image' button. The Gmail and Outlook views show the email being received, with the subject 'Test Email for Clients', the sender 'Andrew [redacted] To: Orange, Josiah', and the body text 'Hi JosiahOrange!'. The email also includes a cautionary note about the origin of the email and a large winter landscape image.</p>

Evaluation

I am happy with my technical solution and how it has turned out. I feel it has achieved almost all of the objectives I set out to achieve. It is effective in having the ability to send to different mail clients, with or without HTML/CSS capabilities. It can save and select templates, have an effective design system and has the ability to choose who does or does not receive the email being sent. The admin area is simple and user friendly for Andrew who does not have a problem navigating it.

Possible Improvements

However, there are a few ways the program can be extended and improved. This is partially due to the openness of the programs concept as well as the time limitations. These could be:

- A way to delete and modify templates that have previously been saved.
 - A way to add and modify admin accounts from the admin area.
 - A way to add and modify factors that can influence who receives the email being designed
 - A way to add, remove and modify email lists.
 - More extensive construction box capabilities and more nodes. E.g. the ability to changes the styling of the nodes.
 - The ability to choose both who will **and** will not receive the email.
 - A preview of all emails that will receive the email depending on the factors currently selected.
- An option to send test emails to Andrew's own email before sending it off to all the recipients.

Most of these improvement could be made with some extra time and are not out of the realms of possibility. Not many bugs have been found but the user interface could definitely be improved with extra and improved features.

Evaluation of Objectives

----- Objective met ----- Objective met somewhat ----- Objective not met

1: Andrew must have an admin area where he can manage his email system to perform tasks.

1.1: The admin area should only be reached through: websitename.com/admin.

[The user is redirected to websitename.com/admin when trying to reach any admin area files therefore the only way of getting into the admin area is logging in through /admin/]

1.2: The admin area should only be reached through a login page.

[The user has to login through websitename.com/admin otherwise he is redirected to the /admin/ login page]

1.2.1: The admin area login must use a username and a password that only Andrew knows.

[The username and password chosen by Andrew is stored in the database (the password in encrypted form) which is used to authenticate the login]

1.2.1: The password must be stored in an encrypted form in the database.

[The password is encrypted with the encryption key stored in a text file. The cyphertext is stored in the database]

1.2.2: If not logged in the browser will be redirected to the admin login page.

1.2.3: The username and password should be validated properly and there should be no way that anybody can access any areas of the admin area without being in a logged in session.

[The username and password are authenticated and the password is encrypted every time Andrew logs in]

1.3: There must be the ability to log out from the admin area and the browser is redirected to the login page again.

[There is a log out button at the top of the admin area in the header so Andrew can log out of the admin area]

1.4: Andrew must be able to design emails to be sent to clients. There must be a simple, interactive user interface in the admin area.

1.4.1: Andrew must be able to drag and drop elements into his central design and be able to edit these elements/nodes to make the email customised and look as he desires.

[This objective has been met however the final design system could be extended and improved by adding the ability to copy and paste nodes]

1.4.2: Elements/nodes of the design must be able to be shuffled around and rearranged.

[Nodes can be swapped, moved around by dragging them over each other]

1.4.3: Elements of the design must be able to be deleted and replaced.

[When a node is clicked, a delete button is displayed, allowing Andrew to delete the node. An empty drop position is left behind which can be replaced]

1.4.4: The design nodes on the left must be labelled and easily understood as to their functionality so Andrew easily know what to drag over to the design construction box.

[They design nodes have titles and also have preview text to give Andrew an Idea of what it is he is dragging into the construction box]

1.4.5: When these nodes are clicked the contents of the node should be able to be changed e.g. change text. The outline should be changed to suggest to Andrew that the node is in edit mode. When in edit mode the node should be able to be deleted. This should only be the case when dragged into construction box.

1.4.6: The emails that are sent must look good and be accessible in all web clients. Therefore, The program must take into the fact that some clients handle email contents differently such as the handling of CSS or even HTML tags. If the client does not accept the use of HTML, then remove the HTML tags and styling from the contents.

[This is done successfully through PHPMailer which allows for a different contents when HTML is not allowed]

1.5: Andrew must be able to specify which emails receive the email he has designed/is designing by using a variety of factors.

1.5.1: Andrew must be able to choose what newsletter subscriptions receive the email.

1.5.2: He must also be able to use the name of the clients to decide who receives the emails.

1.5.3: Andrew must have the ability to only email clients if they haven't or have opened an email before.

[This system is in place however it cannot work In its current state for a couple of reasons. While in development, the system is on localhost and therefore the email recipients cannot reference the tracker.php file. Also, some email clients block the ability to have img src specified e.g. Gmail.]

1.5.4: Andrew must also be able to send emails to specific email addresses if he desires.

[This is done through a factor where Andrew can enter an email or a part of an email which will send to all matching emails]

1.6: Andrew must be able to save and utilise email templates to give him a head start in his designs.

1.6.1: Andrew must be able to save the current state of his design as a template which can be retrieved at a later date.

[The template system works well. The contents and subject and template name is saved into the database and can be selected and loaded back into the construction box at a later date]

1.6.1.1: A template name must be submitted before saving the template. There must be something inside the templates content and there must be a subject entered.

[If any of these inputs are not satisfied, Andrew is alerted and the template is not saved]

1.6.2: Andrew must be able to select a template from a drop down.

[Andrew is given a drop down of all the templates which he can select and load in the admin area]

1.6.4: The current state of the email design will be lost therefore Andrew must be warned before the template is loaded. [A pop up warning is shown as well as a warning that the page is being refreshed]

1.7: It should only take Andrew 5 minutes to get to grips with the user interface.

2: *There must be forms in the front end of Andrew's website that allows website visitors to sign up for his email lists.*

2.1: There will be 3 email lists that can be signed up for on the front end of Andrew's website: Mail Email List, Lighting Updates and Recommended Products

[3 forms are shown for 3 different email lists which can be individually signed up for]

2.1.1: All lists will require name and email however the Main Email List will also require the profession.

2.2: All inputs will require rigid validation. The email must be in email format, the names must start with a capital.

2.3: There must input spam protection for the forms.

[Forms are emptied once submitted to prevent accidental input spam and you cannot input the same email more than once. However, spam is still possible]

2.4: The users must be notified if their information is processed successfully and whether the information is not successfully processed. Notified if customer has signed up successfully, if Andrew has sent email successfully, if template has been saved successfully, if not logged in successfully.

[Status text is used to notify users I they have completed something successfully or not successfully in all of these instances]

3: *All SQL must be secure and protected from security risks like SQL injection. All information must be easily and quickly accessible from the database due to an efficient database design.*

[A secure PHP communication with the database is used using SQL prepare syntax]

3.1: It must be a fully normalised database preventing any inaccurate data.

4: Andrew must be able to have a send email button which will then display 'sending...' until all the emails have been sent which then will display 'sent'.

5: When an email is opened the database must be updated so that it knows that the email has been opened by the customer.

[This system only partially works. An image with src is removed by a lot of mail clients like Gmail and tracker.php is stored locally during development and therefore the tracking does not work faultlessly. However the image src can be placed into a browser and the value is incremented in the database as it would do if the system worked in its entirety]

Should be included:

1: When the user makes a big change, they should be notified which they then have to confirm their choices.

1.1: This includes when a template is deleted, when the page is refreshed, when pressing send email, when selecting a template.

[Users are asked to confirm their choices with a pop up confirmation box in all these instances]

3: Clients should be able to sign up for multiple newsletters simultaneously without causing any problems inside the program and in the database. The client should be able to sign up through different forms in the front end of the website.

[Users are able to sign up for all the email lists simultaneously without a problem. There are no repeated customer emails]

4: Clients should not be able to input their email more than once in the same email list which would spark an error message.

[Clients cannot input their email more than once however it does not spark an error, only prevents the sql input]

5: It should only take Andrew 5-10 minutes to fully grasp the user interface.

6: Andrew should be able to delete templates he has previously saved.

[This has not been implemented purely because of time constraints]

7: Andrew's emails should be able to be sent to all browsers and look good on all browsers, specifically Gmail, Outlook and Apple Mail.

Could be included:

1: There could be the ability to manually add and or remove someone to the email list through the admin area. This should be a separate part of the admin area. This should be done through another simple user interface.

[There is no ability to manually add or remove emails from the admin area purely because of time constraints]

2: The clients could opt in and opt out of email list and email sending from the front end as well as from the emails being sent. This could be a link at the bottom of the emails being sent out as well as an option on the front page of the website (in the footer).

[There is no ability to opt out of an email list, partially because of time constraints and partially because code to do so could not be referenced from an email while the program is in local]

Andrew's Evaluation and Comments

(Written by Andrew Orange – The Client)

Extensions:

Label Content placeholders for usability advice and more labels and descriptions around the admin area.

Saving email during construction to build the email over a few days.

Preview summary of email - which list has been selected, how many emails are being sent from the batch, what the email will look like.

Test email easy to send off during the creation process to see how it is received on multiple platforms - desktop, tablet, phone etc.

Positives:

Drag & drop system is easy to use.

Template creation and selection easy and useful.

Very quick and efficient sending.

A great start to an essential system for any sales orientated business.

Snags:

On screen preview whilst creating an email - even if the construction is very close to the real email that would be sufficient ie. widths of images, content style of mixed image and text.

'Enter Text Here' placeholder should disappear after clicking into placeholder

moving content placeholders within construction can leave a blank space that is confusing

Text WISIWYG missing - rich html content required - bold text, hyperlinking especially. Being able to make smaller changes in the nodes.

Technical Solution:

Key Parts of the Technical Solution

These are some of the techniques and features that have been used throughout the technical solution. They link to one of the files that contain the feature specified.

[Arrays and dictionary](#)

[Encryption](#)

Both selection and iteration ([if else](#), [for](#), [foreach](#), [while](#), [switch case](#))

[Communicating with a database \(SQL\) Cross Table SQL.](#)

[Namespaces](#)

[Ajax](#)

[PHP Objects](#)

[PHP Class Objects.](#)

[Constant Variables](#)

[Private and Public Variables \(get and set\)](#)

[PHP sessions.](#)

[PHP include/require](#)

[Reading and Writing to a text file](#)

[Files and File Upload](#)

[Regex](#)

[PHP POST](#)

File organisation.

[Exception Handling](#)

[User input validation](#) – defensive programming.

[Call by Reference](#)

Any Notable Research References

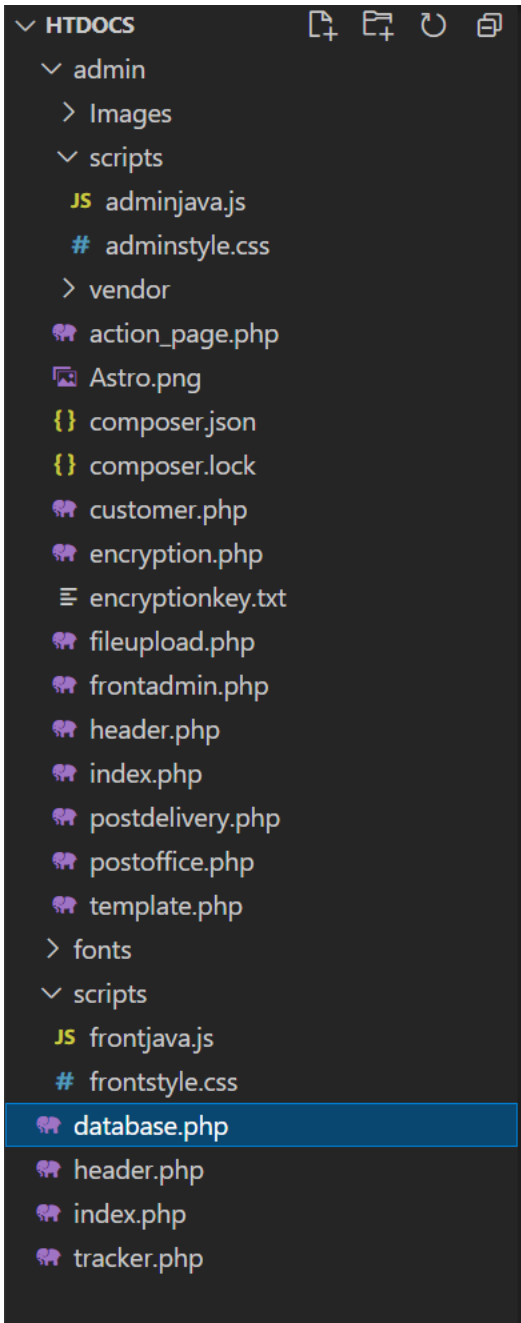
(w3Schools, PHP File Upload, n.d.) : https://www.w3schools.com/php/php_file_upload.asp

(w3Schools, How TO - CSS/JS Modal, n.d.) : https://www.w3schools.com/howto/howto_css_modals.asp

File Structure

The left Image shows the file system that allows **for grouping of subroutines** and classes as well as allowing for **direct access to files**. The admin files being stored in a separate admin folder means the user has to type /admin/ into the url to reach index.php inside the admin folder. Fonts, Images and scripts are all stored in their own separate folders. encryptionkey.txt is stored besides the encryption.php file for direct access.

You can also see the Composer PHPMailer files and the installed Composer Dependency Manager.



frontadmin.php (admin)

```

<?php namespace admin;
include 'header.php';

use Database;
use \admin\Template;

$templates = array();
Database::collectTemplates($templates); //collecting templates and their info using call by reference

$newsletters = array();
Database::collectNewsletters($newsletters); ?> //collecting newsletters and their info using call by reference

<!-- ///// Nodes to be dragged into the contruction box ///// -->
<div id="builder-wrapper">

    <div id="builder-left">

        <h2 class="node-title">Title</h2>
        <div class="draggable" id="drag1" draggable="true" ondrop="nodrop(event)" ondragstart="drag(event)">
            <h1 contenteditable="true">Enter Title Here</h1>
        </div>
        <div class="overlay"></div>

        <h2 class="node-title" >Text</h2>
        <div class="draggable" id="drag2" draggable="true" ondrop="nodrop(event)" ondragstart="drag(event)">
            <p contenteditable="true">Enter Text Here.</p>
        </div>

        <h2 class="node-title">Space</h2>
        <div class="draggable" id="drag3" draggable="true" ondrop="nodrop(event)" ondragstart="drag(event)">
            <div class="space-node" contenteditable="false" style="padding:20px;"></div>
        </div>

        <h2 class="node-title" >Image</h2>
        <div class="draggable" id="drag4" draggable="true" ondrop="nodrop(event)" ondragstart="drag(event)">
            <img id="node-image" class="node-images" width="100" height="auto" />
            <h4>Select Image:</h4>
            <form id="image-upload-form" enctype="multipart/form-data" method="post">
                <input id="image-info" name="image-info" type="file" onchange="document.getElementById('node-
image').src = window.URL.createObjectURL(this.files[0]);addImage(-1)"/>
            </form>
        </div>

    </div>

    <div id="builder-right">

        <form style="width:100%;" action="">
            <input placeholder="Email Subject" id="email-subject" type="text">
        </form>

        <!-- ///// Construction Box Controls ///// -->
        <div id="construction-controls">

            <div class="icmulti ic-node">
                <!-- Choosing Template -->
                <form id="tem-form" method="POST" action="<?=$_SERVER['PHP_SELF'];?>">

```

```

<select name="template" id="template-choose">
  <?php
    for ($x = 0; $x <= sizeof($templates) - 1; $x++) {
      echo '<option value="' . $templates[$x]->name . '">' . $templates[$x]->name . '</option>';
    }
  ?>
</select>
</form>
<input onclick="return confirm('Selecting Template will remove everything currently in construction box!')" form="
tem-form" value="Select Template" type="submit">
</div>

<!-- Saving new template -->
<div id="space-right" class="icmulti ic-node">

  <input placeholder="Template Name" id="template-name" type="text">
  <input onclick="if(confirm('Save current Email Contents and Subject as Template?'))saveTemplate()" value="Save as
Template" type="button">
  <p style="color:green;" id="save-status"></p>
</div>
<!-- Choose Recipients (open popup) -->

<div class="ic-node">
  <input onclick="openRecipient()" id="open-recipient" type="button" value="Choose Recipients">
</div>
<!-- Send Email Button -->

<div class="ic-node">
  <input onclick="if(confirm('Send Email? This will be sent to all matching emails!')) sendEmail();" value="Send Ema
il" type="button">
  <p id="email-status"></p>
</div>

</div>

<input id="counter" style="display:none;" value="0" type="text">

<!-- ///// Construction Box ///// -->
<div id="construction-box">
  <div id="drop-position">
    <div class="custom-node" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
  </div>
</div>

<br>

</div>
</div>

<!-------Pop Up Box----->

<div id="recipient" class="pop-up">

  <!-- Choosing the Recipients Pop Up -->
  <div class="recipient-wrapper">
    <span class="close">&times;</span>

    <form action="">
      <h1>Find Recipients</h1>
      <h2>Specify an Email:</h2>
      <p>Email contains:</p>
      <input id="recemail" type="text">
      <h2>Or choose Parameters:</h2>

```

```

<p>Newsletter:</p>

<!-- Choose Newsletter as factor -->
<select id="rec-news">
  <option value="None">None</option>
  <?php
    for ($x = 0; $x <= sizeof($newsletters) - 1; $x++) {
      echo ' <option value="' . $newsletters[$x] . '">' . $newsletters[$x] . ' </option>';
    }
  ?>
</select>

<p>Hasn't opened an email before:</p>
<input id="rec-check" type="checkbox">

<p>Name contains:</p>
<input id="rec-name" type="text">

</form>
</div>
</div>

<!------- SCRIPTS ----->
<?php
if(isset($_POST['template'])) {

  //fill in construction box with chosen template. Getting the content ready to pass through javascript
  $tempContent = "";
  $tempSubject = "";
  for ($x = 0; $x <= sizeof($templates) - 1; $x++) {
    if($templates[$x]->name == $_POST['template']){
      $tempContent = $templates[$x]->content;
      $tempSubject = $templates[$x]->subject;
      $tempContent = str_replace("'", "\'", $tempContent);
      $tempContent = str_replace('"', '\"', $tempContent);
      $tempContent = str_replace(array("\n", "\r", "\r\n"), '', $tempContent);
    }
  }
  echo '<script>fillInTemplate("' . $tempContent . '", "' . $tempSubject . '")</script>';
} ?>

<script>

function sendEmail() //ajax for sending an email.
{
  document.getElementById("email-status").innerHTML= "Sending...";

  for (const [imageId, imageAddress] of Object.entries(images)) { //temporarily change the sizing of the images in the construction box before being sent as content
    document.getElementById(imageId).src = "cid:" + imageId;
    document.getElementById(imageId).width = "400";
  }

  //sending the email ajax. Passing the subject, content and the chosen recipient factors.
  jQuery(function($) {

    var subject = ""; //stores email subject
    var contents = ""; //stores email content

    subject = document.getElementById("email-subject").value; //get subject
    contents = document.getElementById("construction-box").innerHTML; //get content

```

```

if(subject && contents){
  $.ajax
  ({
    type:'post',
    url:'postoffice.php',
    data:
    {
      _subject:subject,
      _contents:contents,
      _recipientInfo:recipientInfo, //sending recipient variables
      _images:images, //sending image information
    },
    success: function (response)
    {
      document.getElementById("email-status").innerHTML= "Sent!";
    }
  });
}
else{
  document.getElementById("email-status").innerHTML= "";
  alert("Fill in Required");
}
for (const [imageId, imageAddress] of Object.entries(images)) { //revert image sizing
  document.getElementById(imageId).width = "100";
}
return false;
})
}

images = {}; // images variable will store the emails embedded image information. Passed through sendEmail ajax.

function addImage(uploadFormId){ //uploadFormId is the id to identify location of image being added in the email content.

  if (uploadFormId != -1){ //If the image is in content construction box, Get image information.
    var form = document.getElementById("image-upload-form" + uploadFormId);
    var imgNamePath = document.getElementById("image-info" + uploadFormId).value;
  }
  else {
    var form = document.getElementById("image-upload-form");
    var imgNamePath = document.getElementById("image-info").value;
  }

  //get image file name from the path.
  imgNamePaths = imgNamePath.split('\\');
  imgName = imgNamePaths[imgNamePaths.length - 1]

  jQuery(function($) {

    $.ajax( {
      url: 'fileupload.php',
      type: 'POST',
      data: new FormData( form ), //send the form post data from specific image submission form.
      processData: false,
      contentType: false,
      success: function (response){},
      error: function (request, error)
      {
        },
      } );
    return false;
  } );

  if (uploadFormId != -1){ //change the src of image in contents to the new src of uploaded image.

```

```

        document.getElementById("node-image" + uploadFormId).src = "http://localhost:33066/admin/images/" + imgName;
    }
    else{
        document.getElementById("node-image").src = "http://localhost:33066/admin/images/" + imgName;
    }
    //
    var imageId = "node-image" + uploadFormId;
    var imageAddress = "images/" + imgName;
    var tempImage = { [imageId] : imageAddress, };
    images = Object.assign(images, tempImage); //image information added to images variable.
}

function saveTemplate()
{
    //ajax for saving a new template. This passes the content, subject and the chosen template name.
    jQuery(function($) {
        var subject = "";
        var contents = "";

        subject = document.getElementById("email-subject").value;
        contents = document.getElementById("construction-box").innerHTML;
        temName = document.getElementById("template-name").value;

        //prevents any string escapes.
        contents = contents.replace(/'/g, "\\'");
        subject = subject.replace(/'/g, "\\'");
        temName = temName.replace(/'/g, "\\'");
        contents = contents.replace(/"/g, "\\\"");
        subject = subject.replace(/"/g, "\\\"");
        temName = temName.replace(/"/g, "\\\"");

        if(subject && contents && temName && contents.includes("nodecontrol")){
            $.ajax
            ({
                type:'post',
                url:'../database.php',
                data:
                {
                    _subject:subject,
                    _contents:contents,
                    _temName:temName,
                },
                success: function (response)
                {
                }
            });
            document.getElementById("save-status").innerHTML= "Template Saved";

        }
        else{
            alert("Error: Fill in all Contents");
        }
        return false;
    });
}

var recipientInfo = ["", "", "", ""]; //variable storing the recipient information that will be sent through sendEmail function.

function saveRecipient(){
    //saves the recipient factor information into a variable accessible by the whole php file.
    recNews = document.getElementById("rec-news").value;

```



```

recCheck = document.getElementById("rec-check").checked;
recName = document.getElementById("rec-name").value;
recEmail = document.getElementById("recemail").value;

recipientInfo[0] = recNews;
recipientInfo[1] = recCheck;
recipientInfo[2] = recName;
recipientInfo[3] = recEmail;
}

//functions for opening and closing the popup box
var recipientBox = document.getElementById("recipient");

document.getElementsByClassName("close")[0].onclick = function() {
    recipientBox.style.display = "none";
    saveRecipient();
}

document.getElementById("open-recipient").onclick = function() {
    recipientBox.style.display = "block";
}

document.getElementById("open-recipient").onclick = function() {
    recipientBox.style.display = "block";
}

window.onclick = function(event) {
    if (event.target == recipientBox) {
        recipientBox.style.display = "none";
        saveRecipient();
    }
}
</script>

```

postoffice.php (admin)

```

<?php namespace admin;

include 'postdelivery.php';
include 'customer.php';

//for testing
if (!class_exists('Database')){ //only include database if not already included. (for testing mainly)
    include '../database.php';
}

use \admin\Customer;
use Database;
use \admin\PostDelivery;
$images = []; //images variable which will take in the images variable from ajax call.

if(isset($_POST['_contents']))
{
    //preparing to send the email. Mould the email content removing all the information associated with the admin area.
    $content = PostOffice::mouldContents($_POST['_contents']);

    $sql = PostOffice::generateEmailRecipients($_POST['_recipientInfo']);

    $images = $_POST['_images'];
    $customers = array();
    Database::getEmails($sql, $customers);
    foreach ($images as $imageId => $imageAddress){

```

```

$txt = $images[$imageId];
}
//send the emails to all the recipients.
for ($x = 0; $x <= count($customers); $x++) {
    PostDelivery::sendMail($content, $_POST['_subject'], $customers[$x], $images,$_POST['_recipientInfo'][0]);
}
}

class PostOffice{

    static function mouldContents($content){
        $content = preg_replace('/class=".*?"/', '', $content);
        $content = preg_replace('/id=".*?"/', '', $content);
        $content = preg_replace('/ondrop=".*?"/', '', $content);
        $content = preg_replace('/ondragover=".*?"/', '', $content);
        $content = preg_replace('/ondragstart=".*?"/', '', $content);
        $content = preg_replace('/draggable=".*?"/', '', $content);
        $content = preg_replace('/<button>.*?</button>/', '', $content);
        $content = preg_replace('/contenteditable=".*?"/', '', $content);
        $content = preg_replace('/border:.*?/', 'border:none;', $content);
        $content = preg_replace('/<form.*?>.*?</form>/', '', $content);
        $content = preg_replace('/<h4>.*?</h4>/', '', $content);
        return $content;
    }

    // generates the sql to get all the desires recipients depending on the factors inputted by the user (Andrew).
    static function generateEmailRecipients($recipientInfo){

        if($recipientInfo[3] == ""){

            $sql = "SELECT customers.email, customers.firstName, customers.lastName, customers.profession FROM customers INNER
JOIN subscription ON customers.idCustomer = subscription.idCustomer WHERE customers.email <> ''";
            if ($recipientInfo[0] != "None"){
                $sql = $sql . " AND ";
                $sql = $sql . "subscription.idNewsletter = (SELECT idNewsletter FROM newsletters WHERE newsName = '" . $recipientI
tInfo[0] . "')";
            }
            if ($recipientInfo[1] == "true"){
                $sql = $sql . " AND ";
                $sql = $sql . "subscription.emailsOpened = NULL";
            }
            if ($recipientInfo[2] != ""){
                $sql = $sql . " AND ";
                $sql = $sql . "customers.firstName LIKE '%" . $recipientInfo[2] . "%' OR customers.lastName LIKE '%" . $recipientI
nfo[2] . "%'";
            }
        }
        else{
            $sql = "SELECT email, firstName, lastName, profession FROM customers WHERE email LIKE '%" . $recipientInfo[3] . "%'";
        }
        return $sql;
    }
}

```

postdelivery.php (admin)

```

<?php namespace admin;

// use the classes from the PHPMailer plugin.
use PHPMailer\PHPMailer\PHPMailer;

```

```

use PHPMailer\PHPMailer\SMTP;
use PHPMailer\PHPMailer\Exception;

class PostDelivery{

    //----- Send Email using the passed recipient information. -----//
    static function sendMail($bodyContent, $subject, $customer, $images, $newsName) {

        $bodyContent = "Hi " . $customer->firstName . $customer->lastName . "!" . $bodyContent;
        // Import PHPMailer classes into the global namespace
        // Load Composer's autoloader
        require 'vendor/autoload.php';
        // Instantiation and passing `true` enables exceptions
        $mail = new PHPMailer(true);

        try {
            //Server settings
            $mail->SMTPDebug = SMTP::DEBUG_SERVER;                          // Enable debug
            $mail->isSMTP();                                                  // Send using SMTP
            $mail->Host = 'smtp.gmail.com';                                    // Set the SMTP server to send through
            $mail->SMTPAuth = true;                                           // Enable SMTP authentication
            $mail->Username = '[GMAIL EMAIL]';                                // set SMTP username
            $mail->Password = '[SMTP PASSWORD]';                             // set SMTP password
            $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;               // Enable encryption
            $mail->Port = 587;                                                 // TCP port to connect to

            //Recipients
            $mail->setFrom('[ANDREWS EMAIL]', 'Andrew');
            $mail->addAddress($customer->email, $customer->firstName);          // recipient
            $mail->addReplyTo('[ANDREWS EMAIL]', 'Andrew');
            $tracker = 'ErrorInfo}";
        }
    }
}

```

index.php (admin)

```

<?php namespace admin;

include '../database.php';
include 'encryption.php';

use \admin\Encryption;
use Database;
session_start(); //session for login information.
//----- Login Page -----//

```

```

$details = Database::collectAdmin();
$details[1] = Encryption::decrypt(strval($details[1]));
$detUser = strval($details[0]);
$detPass = strval($details[1]);

if(isset($_SESSION["password"])&& isset($_SESSION["username"])) //if already logged in send to admin area
{
    if( $_SESSION["password"] == $details[1] && $_SESSION["username"] == $details[0]){
        header('Location: http://' . $_SERVER['SERVER_NAME'] . ':' . $_SERVER['SERVER_PORT'] . '/admin/frontadmin.php');
        exit();
    }
}

if($_SERVER['REQUEST_METHOD']=='POST') //if trying to log in.
{
    $_SESSION["username"] = $_POST['username'];
    $_SESSION["password"] = $_POST['password'];
    $details = Database::collectAdmin(); //get admin info from database then decrypt it.
    $details[1] = Encryption::decrypt($details[1]);

    if( $_SESSION["password"] == $details[1] && $_SESSION["username"] == $details[0]){
        echo "correct!";
        $cypherTxt = Encryption::encrypt($_SESSION["password"]);
        Database::updateAdmin($cypherTxt, $_SESSION["username"]);

        header('Location: http://' . $_SERVER['SERVER_NAME'] . ':' . $_SERVER['SERVER_PORT'] . '/admin/frontadmin.php');
        exit();
    }
    else {
        echo '<p style="padding:20px;color:red">Incorrect Login Info - Try again!</p>';
    }
}
?>

<link rel="stylesheet" href="scripts/adminstyle.css">

<div id="login-area">
    <h1>Login to Admin</h1>
    <form method="post" action="<?=$_SERVER['PHP_SELF'];?>">
        <label for="">username</label>
        <input name="username" type="text">
        <label for="">password</label>
        <input name="password" type="text">
        <input type="submit" value="click">
    </form>
</div>

```

header.php (admin)

```

<?php namespace admin;?>

<!DOCTYPE html>
<html>
    <head> <!-- enqueueing js and css files as well as jquery capabilities-->
        <link rel="stylesheet" href="scripts/adminstyle.css">
        <script src="scripts/adminjava.js"></script>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <?php session_start(); //starts session allowing for the storing of login information.
    include 'template.php'; //including all necessary php files. Header acts as in between for functionality.
    include '../database.php';
    include 'postoffice.php';

```

```

include 'encryption.php';

use \admin\Encryption;
use Database;

if (isset($_POST['logout'])){
    $_SESSION["password"] = "";
    $_SESSION["username"] = "";
    header("Refresh:0");
}

//----- Check if the user is logged in or not -----//
$details = Database::collectAdmin();
$details[1] = Encryption::decrypt($details[1]);
$detUser = strval($details[0]);
$detPass = strval($details[1]);

if(isset($_SESSION["password"])&& isset($_SESSION["username"]))
{
    //if not logged in then sent to the login page.
    if( $_SESSION["password"] == $details[1] && $_SESSION["username"] = $details[0]){
    }
    else{
        header('Location: http://' . $_SERVER['SERVER_NAME'] . ':' . $_SERVER['SERVER_PORT'] . '/admin');
        exit();
    }
}
else{
    header('Location: http://' . $_SERVER['SERVER_NAME'] . ':' . $_SERVER['SERVER_PORT'] . '/admin');
    exit();
}
?>

<header>
<h1>Admin Area</h1>
<form method="POST">
    <input onclick="return confirm('Log out of Admin Area?')" value="Log Out" name="logout" id="logout" type="submit">
</input>
</form>
</header>
</head>

```

fileupload.php (admin)

```

<?php

// Check if image file is a actual image or fake image
if(isset($_FILES["image-info"])) {

    FileUpload::uploadImage();
}

class FileUpload{

    static function uploadImage(){

        $target_dir = "images/";
        $target_file = $target_dir . basename($_FILES["image-info"]["name"]);
        $canUpload = true;
        $imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

        // Check if the file is actually an image
        $check = getimagesize($_FILES["image-info"]["tmp_name"]);
    }
}

```

```

if($check !== false) {
    $canUpload = true;
} else {
    $canUpload = false;
}

// Check if file already exists
if (file_exists($target_file)) {
    $canUpload = false;
}

// Check file size
if ($_FILES["image-info"]["size"] > 2000000) {
    $canUpload = false;
}

// Allow certain image formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg" && $imageFileType != "gif" ) {
    $canUpload = false;
}

// is the image able to be uploaded?
if ($canUpload == true) {
    // upload file
    (move_uploaded_file($_FILES["image-info"]["tmp_name"], $target_file));
}
}
}

```

tracker.php (admin)

```

<?php namespace admin;

include 'database.php';

use Database;

if( !empty( $_GET['log'] ) && $_GET['log'] == 'true' && !empty( $_GET['user'] ) && !empty( $_GET['subject'] ) ){
    Database::emailOpened($_GET['user'], $_GET['news']);
}

```

template.php (admin)

```

<?php namespace admin;

//storing all the templates and their information.
class Template {
    private string $content; //stores the email content that will be stored and retrieved from database.
    private string $subject; //stores the subject for the template
    private string $name; //stores the template name

    function __construct(string $name, string $cont, string $sub) {
        $this->content = $cont;
        $this->subject = $sub;
        $this->name = $name;
    }

    // PHPs version of Get Set.
    function getName(){
        return $this->name;
    }
}

```

```

    }
    function getSubject(){
        return $this->subject;
    }
    function getContent(){
        return $this->content;
    }
    function setContent($value){
        $this->content = $value;
    }
    function setName($value){
        $this->name = $value;
    }
    function setSubject($value){
        $this->subject = $value;
    }
}

public function __set($ID,$value) {
    switch($ID) {
        case 'name':
            return $this->setName($value);
        case 'subject':
            return $this->setSubject($value);
        case 'content':
            return $this->setContent($value);
    }
}

public function __get($ID) {
    switch($ID) {
        case 'name':
            return $this->getName();
        case 'subject':
            return $this->getSubject();
        case 'content':
            return $this->getContent();
    }
}
}

```

customer.php (admin)

```

<?php namespace admin;

//stores each customer and their information (each email sign up) that will be sent an email

class Customer {

    private string $email; //customers email
    private string $firstName; //customers first name
    private string $lastName; //customers last name
    private string $profession; //customer profession (if applicable)

    function __construct(string $email, string $firstName, string $lastName, string $profession) {
        $this->email = $email;
        $this->firstName = $firstName;
        $this->lastName = $lastName;
        $this->profession= $profession;
    }
}

```

```
// PHPs version of Get Set.

function getEmail(){
    return $this->email;
}
function getFirstName(){
    return $this->firstName;
}
function getLastName(){
    return $this->lastName;
}
function getProfession(){
    return $this->profession;
}
function setLastName($value){
    $this->lastName = $value;
}
function setEmail($value){
    $this->email = $value;
}
function setFirstName($value){
    $this->firstName = $value;
}
function setProfession($value){
    $this->profession = $value;
}

public function __set($ID,$value) {
    switch($ID) {
        case 'email':
            return $this->setEmail($value);
        case 'firstName':
            return $this->setFirstName($value);
        case 'lastName':
            return $this->setLastName($value);
        case 'profession':
            return $this->setProfession($value);
    }
}

public function __get($ID) {
    switch($ID) {
        case 'email':
            return $this->getEmail();
        case 'firstName':
            return $this->getFirstName();
        case 'lastName':
            return $this->getLastName();
        case 'profession':
            return $this->getProfession();
    }
}
}
```

database.php

```
<?php
use \admin\Template;
use \admin\Customer;

// constants for database information.
define("dbServerName", "localhost");
```



```

define("dbUsername", "root");
define("dbPassword", "local336236!");
define("dbName", "emailsystemdb");

if(isset($_POST['_temName']))
{
    Database::saveTemplate();
}
?>
<?php
if(isset($_POST['_firstName']))
{
    if(isset($_POST['_identifier']) == "1"){
        Database::signUp();
    }

    if(isset($_POST['_identifier']) == "2"){
        Database::signUp();
    }
}

class Database{

    //Signing up to a newsletter
    static function signUp() {

        // Create connection
        $conn1 = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
        // Check connection
        if ($conn1->connect_error) {
            die("Connection failed: " . $conn1->connect_error);
        }
        //SQL string
        $sql = "SELECT email from customers WHERE email='" . $_POST['_email'] . "'"; //Find if email already entered into data
base
        $sql = $conn1->prepare($sql);
        $sql->execute();
        $result = $sql->get_result();    //Recieve Results from SQL query

        if (!empty($result) && $result->num_rows > 0) { //If email already exists in the database.
            // Create connection
            $conn2 = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
            // Check connection
            if ($conn2->connect_error) {
                die("Connection failed: " . $conn2->connect_error);
            }
            //SQL string
            $sql = "SELECT idSub
FROM subscription
WHERE idCustomer =
(SELECT idCustomer
FROM customers
WHERE email = '" . $_POST['_email'] . "') && idNewsletter =
(SELECT idNewsletter
FROM newsletters
WHERE idNewsletter = '" . $_POST['_formId'] . "') ";
            //Recieve Results from SQL query
            $sql = $conn2->prepare($sql);
            $sql->execute();
            $result = $sql->get_result();
            if (!empty($result) && $result->num_rows > 0) {} //If email is already signed up for that newsletter - do nothing.
            else{ //If email exists but is not signed up for that newsletter / mail list then sign them up.
                // Create connection

```

```

$conn3 = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
// Check connection
if ($conn3->connect_error) {
    die("Connection failed: " . $conn2->connect_error);
}
//SQL string
$sql = "INSERT INTO subscription (idNewsletter, idCustomer)
VALUES ((SELECT idNewsletter from newsletters WHERE idNewsletter=" . $_POST['_formId'] . "),(SELECT idCustomer fro
m customers WHERE email='" . $_POST['_email'] . "'))";

$sql = $conn3->prepare($sql);
if ($sql->execute()) {
} else {
}
$conn3->close();
//If customer does not yet have a profession assigned then assign them the profession they have selected.
// Create connection
$conn4 = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
// Check connection
if ($conn4->connect_error) {
    die("Connection failed: " . $conn4->connect_error);
}
//SQL string
$sql = "UPDATE customers
SET profession = '" . $_POST['_profession'] . "'
WHERE email = '" . $_POST['_email'] . "'
AND profession = ''
OR profession = NULL";

$sql = $conn4->prepare($sql);
$sql->execute();
$conn4->close();
}
}
else { //If email does not yet exist in database insert into customer table as well as sign them up for a newsletter /
mail list.

// Create connection
$conn2 = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
// Check connection
if ($conn2->connect_error) {
    die("Connection failed: " . $conn2->connect_error);
}
//SQL string
$sql = "INSERT INTO customers (firstName, lastName, email, profession)
VALUES ('" . $_POST['_firstName'] . "', '" . $_POST['_lastName'] . "', '" . $_POST['_email'] . "', '" . $_POST['_pro
fession'] . "'));

$sql = $conn2->prepare($sql);
$sql->execute();
$conn2->close();

// Create connection
$conn3 = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
// Check connection
if ($conn3->connect_error) {
    die("Connection failed: " . $conn2->connect_error);
}
//SQL string
$sql = "INSERT INTO subscription (idNewsletter, idCustomer)
VALUES (
(SELECT idNewsletter
FROM newsletters

```

```

        WHERE idNewsletter=" . $_POST['_formId'] . "),(SELECT idCustomer from customers WHERE email='" . $_POST['_email']
        . "'));

        $sql = $conn3->prepare($sql);
        $sql->execute();
        $conn3->close();
    }
    $conn1->close();
}

//Gets admin information for login validation

static function collectAdmin(){

    // Create connection
    $conn = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    //SQL string
    $sql = "SELECT username, pass
    FROM admins";
    //Recieve Results from SQL query
    $sql = $conn->prepare($sql);
    $sql->execute();
    $result = $sql->get_result();
    if ($result->num_rows > 0) {
        // output data of each row
        while($row = $result->fetch_assoc()) {
            $details = array($row["username"], $row["pass"]);
            return $details;
        }
    } else {
    }
    $conn->close();
}

//Gets the templates available for loading.

static function collectTemplates(&$templates){

    // Create connection
    $conn = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    //SQL string
    $sql = "SELECT temName, temContent, temSubject
    FROM templates";
    //Recieve Results from SQL query
    $sql = $conn->prepare($sql);
    $sql->execute();
    $result = $sql->get_result();
    if ($result->num_rows > 0) {
        // output data of each row
        while($row = $result->fetch_assoc()) {
            array_push($templates, new Template($row["temName"], $row["temContent"], $row["temSubject"]));
        }
    } else {
    }
}

```

```

$conn->close();
return $templates;
}

//Saves a new template into the database for use

static function saveTemplate(){

    // Create connection
    $conn = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    //SQL string
    $sql = "INSERT INTO templates (temSubject, temContent, temName)
    VALUES ('" . $_POST['_subject'] . "', '" . $_POST['_contents'] . "', '" . $_POST['_temName'] . "')";
    $sql = $conn->prepare($sql);

    $sql->execute();
    $conn->close();
}

//updates the cypher text stored in the database.
static function updateAdmin($encryptPass, $username){

    $encryptPass = str_replace("'", "''", $encryptPass);

    // Create connection
    $conn = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    //SQL string
    $sql = "UPDATE admins
    SET pass = '" . $encryptPass . "'
    WHERE username = '" . $username . "' ";

    $sql = $conn->prepare($sql);
    $sql->execute();
    $conn->close();
}

//Checks if an email has previously been opened by a customer

static function emailOpened($user, $news) {

    if ($news != ''){
        // Create connection
        $conn = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
        // Check connection
        if ($conn->connect_error) {
            die("Connection failed: " . $conn->connect_error);
        }
        //SQL string
        $sql = "UPDATE subscription
        SET emailsOpened=emailsOpened + 1
        WHERE idCustomer=
        (SELECT idCustomer
        FROM customers
        WHERE email= '" . $user . "' )
        AND idNewsletter="
    }
}

```

```

        (SELECT idNewsletter
        FROM newsletters
        WHERE newsName= ' " . $news . "'');

    $sql = $conn->prepare($sql);
    $sql->execute();
    $conn->close();
}

}

//Gets all newsletters that are available for signup. (For the recipient choosing popup)

static function collectNewsletters(&$newsletters){

    // Create connection
    $conn = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    //SQL string
    $sql = "SELECT newsName
    FROM newsletters";
    //Recieve Results from SQL query
    $sql = $conn->prepare($sql);
    $sql->execute();
    $result = $sql->get_result();
    if ($result->num_rows > 0) {
        // output data of each row
        while($row = $result->fetch_assoc()) {
            array_push($newsletters, $row["newsName"]);
        }
    } else {
    }
    $conn->close();
    return $newsletters;
}

//Gets all recipients that match the factors chosen by andrew. (using the generated sql statement)

static function getEmails($sql, &$customers){

    // Create connection
    $conn = new \mysqli(dbServerName, dbUsername, dbPassword, dbName);
    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }
    $sql = $conn->prepare($sql);
    //Recieve Results from SQL query
    $sql->execute();
    $result = $sql->get_result();
    if (!empty($result) && $result->num_rows > 0) {
        // output data of each row
        while($row = $result->fetch_assoc()) {
            //saved as new customer instance
            array_push($customers, new Customer($row["email"], $row["firstName"], $row["lastName"], $row["profession"]));
        }
    } else {
    }
    $conn->close();
    return $customers;
}

```

```
}
}
```

header.php (front end)

```
<?php namespace frontend;
session_start();?>

<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="scripts/frontstyle.css">
    <script src="scripts/frontjava.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <header>
    <h1>Front Page</h1>
  </header>
</head>
```

index.php (front end)

```
<?php namespace frontend;
include 'header.php'; ?>

<script>
//ajax for the clients submission of a newsletter sign up (to be inserted in to database).
function clientSubmit(form)
{
  jQuery(function($) {
    var firstname = "";
    var lastname = "";
    var email = "";
    var profession = "";
    var identifier = "";
    var formId = "";

    firstName = document.getElementById("firstname" + form).value;
    lastName = document.getElementById("lastname" + form).value;
    email = document.getElementById("email" + form).value;
    identifier = document.getElementById("identifier" + form).value;

    if (form == 1){
      profession = document.getElementById("profession" + form).value;
    }

    formId = document.getElementById("formid" + form).value;
    var validEmail = valEmail(email);
    var validName1 = valName(firstName);
    var validName2 = valName(lastName);

    if(firstName && validEmail && validName1 && validName2){
      $.ajax
      ({
        type: 'post',
        url: 'database.php',
        data:
        {
          _firstName: firstName,
          _lastName: lastName,
          _email: email,
```

```

        _identifier:identifier,
        _formId:formId,
        _profession:profession,
    },
    success: function (response)
    {
        document.getElementById("status" + form).style.color="green";
        document.getElementById("status" + form).innerHTML="Successfully Signed Up!";
        document.getElementById("firstname" + form).value="";
        document.getElementById("lastname" + form).value="";
        document.getElementById("email" + form).value="";
        document.getElementById("profession" + form).value="";
    }
    });
}
else{
    document.getElementById("status" + form).style.color="red";
    document.getElementById("status" + form).innerHTML="Incorrect Values!";
}
return false;
});
}
</script>

<!--sign up for for main newsletter-->
<form class="signup-form" name="form" onsubmit="return=false;" >
    <h2>Main Email List</h2>
    <input value="1" id="formid1" style="display:none;" type="text">
    <label for="">First Name</label>
    <input id="firstname1" name="firstname" type="text">
    <label for="">Last Name</label>
    <input id="lastname1" name="lastname" type="text">
    <label for="">Email</label>
    <input id="email1" name="email" type="text">
    <select id="profession1" name="profession">
        <option value="Designer">Designer</option>
        <option value="Architect">Architect</option>
        <option value="Retail Customer">Retail Customer</option>
        <option value="Lighting Consultant">Lighting Consultant</option>
        <option value="Electrician/Contractor">Electrician/Contractor</option>
        <option value="Other">Other</option>
    </select>
    <input id="identifier1" name="identifier" style="display:none;" value="1" type="text">
    <input type="button" onclick="clientSubmit(1);" name="submit" value="Sign Up" />
    <p id="status1"></p>
</form>

<form class="signup-form" name="form" onsubmit="return=false;" >
    <h2>Lighting Updates</h2>
    <input value="2" id="formid2" style="display:none;" type="text">
    <label for="">First Name</label>
    <input id="firstname2" name="firstname" type="text">
    <label for="">Last Name</label>
    <input id="lastname2" name="lastname" type="text">
    <label for="">Email</label>
    <input id="email2" name="email" type="text">
    <input id="identifier2" name="identifier" style="display:none;" value="2" type="text">
    <input type="button" onclick="clientSubmit(2);" name="submit" value="Sign Up" />
    <p id="status2"></p>
</form>

<form class="signup-form" name="form" onsubmit="return=false;" >
    <h2>Recommended Products</h2>

```

```

<input value="3" id="formid3" style="display:none;" type="text">
<label for="">First Name</label>
<input id="firstname3" name="firstname" type="text">
<label for="">Last Name</label>
<input id="lastname3" name="lastname" type="text">
<label for="">Email</label>
<input id="email3" name="email" type="text">
<input id="identifier3" name="identifier" style="display:none;" value="3" type="text">
<input type="button" onclick="clientSubmit(3);" name="submit" value="Sign Up" />
<p id="status3"></p>
</form>

```

adminjava.js

```

window.onload = function() {
  var nodeImages = document.getElementsByClassName("node-images");
  function updateImages() {
    Object.keys(nodeImages).forEach(key => updateImage(key));
  }

  function updateImage(key){
    if (nodeImages[key].src != ""){
      nodeImages[key].src = nodeImages[key].src.split("?")[0];
    }
  }
  setInterval(updateImages, 2000);
}
window.onbeforeunload = function(event)
{
  return confirm("Confirm refresh: Data may be lost.");
};
//function called when user clicks on their window.
window.addEventListener('click', function(e){

  var allNodes = document.getElementsByClassName("draggable") //gets all nodes with the ability to be dragged
  for (i = 0; i < allNodes.length; i++) {
    //if element clicked on is a draggable node, pause dragging capabilities and show the node controls
    if (allNodes[i].contains(e.target)){

      allNodes[i].setAttribute('draggable', false);
      allNodes[i].style.border="1.5px solid orange";
      allNodes[i].style.cursor="text";
      if(allNodes[i].classList.contains("custom-node")){ //if it is a node inside constructionbox

        var nodeIdentifier = allNodes[i].id.charAt(allNodes[i].id.length-
1); //get the nodes unique number to be able to target its controls
        var x = document.getElementById("nodecontrol" + nodeIdentifier);
        x.style.display = "initial"; //display the nodes controls to user
      }
      //if the element clicked on isnt a draggable node, resume all dragging capabilities and turn off controls
    }
    else{
      allNodes[i].setAttribute('draggable', true);
      allNodes[i].style.border="0.5px solid #888";
      allNodes[i].style.cursor="pointer";
      if(allNodes[i].classList.contains("custom-node")){

        var nodeIdentifier = allNodes[i].id.charAt(allNodes[i].id.length-1);
        var a = document.getElementById("nodecontrol" + nodeIdentifier);
        a.style.display = "none";
      }
    }
  }
}

```



```

    }
  });

function deleteNode(ev){
  ev.target.parentElement.style.display = "none";
  ev.target.parentElement.previousElementSibling.innerHTML = "";
}

function allowDrop(ev) {
  ev.preventDefault();
}

function nodrop(ev){
  ev.preventDefault();
}

//set data transfer data to id of node being dragged
function drag(ev) {
  ev.dataTransfer.setData("text", ev.target.id);
}

//inserting a node after a reference node
function insertAfter(referenceNode, newNode) {
  referenceNode.parentNode.insertBefore(newNode, referenceNode.nextSibling);
}

function drop(ev) {

  ev.preventDefault();
  if(ev.target.class != "draggable"){ //if the dropping position doesnt already contain a node
    var data = ev.dataTransfer.getData("text"); //get the data transfer data (the id of the dragging node)
    // if the node being dragged was inside construction box
    if(document.getElementById(data).classList.contains("custom-
node") && ev.target.parentElement.id != document.getElementById(data).id && ev.target.parentElement.id != "" && ev.target.
id != "construction-box")
    {
      if( ev.target.childNodes.length == 0 && ev.target.className != "space-
node"){ //if the destination position is empty then append the dragging node
        alert((ev.target.classList));
        //copy the original nodes controls.
        var nodeCopyControls = document.getElementById(data).parentElement.nextElementSibling.cloneNode(true);
        //remove the original node controls
        document.getElementById(data).parentElement.nextElementSibling.remove();
        ev.target.appendChild(document.getElementById(data));
        insertAfter(ev.target, nodeCopyControls);

      }

      //if the destination position already has a node inside then swap the nodes.
      else{

        var nodeCopy = document.getElementById(data).cloneNode(true);
        var parent = ev.target.parentElement;
        var nodeCopy2 = parent.cloneNode(true);

        var parent1 = document.getElementById(data).parentElement;
        var parent2 = parent.parentElement;

        var nodeCopyControls = document.getElementById(data).parentElement.nextElementSibling.cloneNode(true);
        var nodeCopy2Controls = parent.parentElement.nextElementSibling.cloneNode(true);

        parent1.innerHTML = "";
        parent2.innerHTML = "";
      }
    }
  }
}

```

```

    parent1.nextElementSibling.remove();
    parent2.nextElementSibling.remove();

    parent2.appendChild(nodeCopy);
    insertAfter(parent1, nodeCopy2Controls);

    parent1.appendChild(nodeCopy2);
    insertAfter(parent2, nodeCopyControls);

}
}
// if the original node being dragged wasnt inside construction box then prepare node for construction box, giving it a unique id.
else{
    var nodeCopy = document.getElementById(data).cloneNode(true);
    var counter = parseInt(document.getElementById("counter").value);

    document.getElementById("counter").value = counter + 1;
    nodeCopy.className = "custom-node draggable";

    // if the node is an image node
    if(document.getElementById(data).id == "drag4"){

        nodeCopy.children.item(0).id = nodeCopy.children.item(0).id + counter;
        nodeCopy.id = document.getElementById(data).id + "custom" + counter;

        imageUploadForm = "<form id=\"image-upload-form\" + counter + \"\" enctype=\"multipart/form-data\" method=\"post\"><input id=\"image-info\" + counter + \"\" name=\"image-info\" type=\"file\" onchange=\"document.getElementById('node-image\" + counter + \").src = window.URL.createObjectURL(this.files[0]);addImage(\" + counter + \")\"/></form>"
        nodeCopy.removeChild(nodeCopy.children.item(2));
        nodeCopy.innerHTML = nodeCopy.innerHTML + imageUploadForm;

    }
    nodeCopy.id = document.getElementById(data).id + "custom" + counter;

    if( ev.target.childNodes.length == 0){ //if the drop position is empty then append the altered node and add the controls

        ev.target.appendChild(nodeCopy);
        var nodeControls = document.createElement("div");
        nodeControls.id = 'nodecontrol' + counter;
        nodeControls.className = 'nodecontrol';
        nodeControls.innerHTML = '<button class="delete-button" onclick="deleteNode(event)">Delete</button>';
        //var nodecontroldef = '<div id="nodecontrol" + counterplusone + \"\" class="nodecontrol"><button onclick="deleteNode(event)">Delete</button></div>';

        insertAfter(ev.target, nodeControls);
        //add new empty drop position
        const div = document.createElement('div');
        div.id = "drop-position"
        div.innerHTML = '<div class="custom-node" ondrop="drop(event)" ondragover="allowDrop(event)"></div>';
        document.getElementById("construction-box").appendChild(div);
    }
}
}
}

//<div id="nodecontrol" + counterplusone + \"\" class="nodecontrol"><button onclick="deleteNode(event)">Delete</button></div>
>
function fillInTemplate(tempContent, tempSubject){ //empties and fills up the contruction box with the template
    document.getElementById("construction-box").innerHTML = "";

```

```

document.getElementById("construction-box").innerHTML = tempContent;
document.getElementById("email-subject").value = tempSubject;
}

```

adminstyle.css

```

/*Adding fonts*/
@font-face {
  font-family: 'OpenSans';
  src: url('../..../fonts/OpenSans-Light.ttf');
  font-style: normal;
  font-weight: 300;
  font-display: swap;
}
@font-face {
  font-family: 'OpenSans';
  src: url('../..../fonts/OpenSans-Regular.ttf');
  font-style: normal;
  font-weight: 400;
  font-display: swap;
}
@font-face {
  font-family: 'OpenSans';
  src: url('../..../fonts/OpenSans-Bold.ttf');
  font-style: normal;
  font-weight: 700;
  font-display: swap;
}
@font-face {
  font-family: 'Roboto';
  src: url('../..../fonts/Roboto-Thin.ttf');
  font-style: normal;
  font-weight: 100;
  font-display: swap;
}
@font-face {
  font-family: 'Roboto';
  src: url('../..../fonts/Roboto-Light.ttf');
  font-style: normal;
  font-weight: 300;
  font-display: swap;
}
@font-face {
  font-family: 'Roboto';
  src: url('../..../fonts/Roboto-Regular.ttf');
  font-style: normal;
  font-weight: 400;
  font-display: swap;
}
@font-face {
  font-family: 'Roboto';
  src: url('../..../fonts/Roboto-Italic.ttf');
  font-style: italic;
  font-weight: 400;
  font-display: swap;
}
@font-face {
  font-family: 'Roboto';
  src: url('../..../fonts/Roboto-Medium.ttf');
  font-style: normal;
  font-weight: 500;

```

```

    font-display: swap;
}
@font-face {
    font-family: 'Roboto';
    src: url('../..../fonts/Roboto-Bold.ttf');
    font-style: normal;
    font-weight: 700;
    font-display: swap;
}
body{
    margin:0;
    padding:0;
    font-family: Roboto;
    font-weight:400;
}
html{
    height:100%;
}
/*Styling for all nodes in contruction box.*/
.custom-node {
    width: 100%;
    min-height: 70px;
    border:none;
}
/*Shows when construction box node is hovered over.*/
.custom-node:hover{
    border:1px solid grey;
}
#custom-node{
    padding:10px;
}
/*General styling for all nodes*/
.draggable{
    cursor: pointer;
    border: solid;
    border-color: #888;
    border-width: 0.5px;
    border-radius: 5px;
    padding: 10px;
    margin-top:5px;
}
/*Styling for main area of the admin area which allows for the left and right section to be positioned*/
#builder-wrapper{
    display: flex;
    justify-content: left;
    padding-top:30px;
    max-width:1500px;
    margin:40px;
    height:100%;
}
/*These are the left and right sections that sit next to each other inside builder-wrapper.*/
#builder-left{
    width:50%;
    padding:30px;
    padding-left:0px;
}
#builder-right{
    width:50%;
}
/*Styling for the construction box wrapper*/
#construction-box{
    border: solid;
    border-width: 1px;

```

```

display:flex;
justify-content: left;
flex-direction: column;
height:auto;
overflow: hidden;
height:100%;
padding:20px;
border-radius: 5px ;
}
/*Header styling*/
header{
display:flex;
justify-content: center;
align-items: center;
background-color: #c06400;
margin:0px;
}
header p{
cursor: pointer;
padding:20px;
padding-top: 10px;

}
header a{
color:black;
text-decoration: none;
}
header h1{
padding:20px;
color:white;
}
/*Log out button styling*/
header input{
background-color: #fff;
border-radius: 5px;
padding:5px;
padding-right: 10px;
padding-left: 10px;

}
/*Controls for each node (the delete button) initially not displayed.*/
.nodecontrol{
display: none;
}
/*Styling for the controls for the construction box above the construction box*/
#construction-controls{
display:flex;
padding-top:20px;
padding-bottom: 20px;
align-items:flex-start;
justify-content: left;
flex-direction:row;
}
#construction-controls input{
height:40px;
border-radius: 5px;
box-shadow: none;
background-color:#c06400;
color:white;
}
#construction-controls select{
height:40px;
border-radius: 5px;

```

```

    box-shadow: none;
    background-color: white;
}
/* Styling for the Email Subject Input */
#email-subject{
    padding:20px;
    width:94%;
    font-size: 1.5rem;
    border-radius: 5px;
}
/*Styling for pop up. Not initially displayed and has a fixed position on the web page.*/
.pop-up {
    display: none; /* Hidden by default */
    position: fixed; /* Stay in place */
    z-index: 1; /* Sit on top */
    padding-top: 100px; /* Location of the box */
    left: 0;
    top: 0;
    width: 100%; /* Full width */
    height: 100%; /* Full height */
    overflow: auto; /* Enable scroll if needed */
    background-color: rgb(0,0,0); /* Fallback color */
    background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
}
/*Wrapper inside pop up box*/
.recipient-wrapper {
    background-color: #fefefe;
    margin: auto;
    padding: 40px;
    border: 1px solid #888;
    width: 80%;
}
/* The Close Button */
.close {
    color: #aaaaaa;
    float: right;
    font-size: 28px;
    font-weight: bold;
}
.close:hover,
.close:focus {
    color: #000;
    text-decoration: none;
    cursor: pointer;
}
.node-title{
}
.delete-button{
    background-color: #ff1717;
    color:#fff;
    border-radius: 5px;
}
#template-choose{
    height:100%;
}
.ic-node{
    display:flex;
    flex-direction: column;
    align-items:flex-start;
    padding-right: 20px;
}
#construction-controls .icmulti input{

```

```

    height:30px;
}
#construction-controls .icmulti select{
    height:35px;
}
#tem-form{
    margin-bottom:10px;
}
#construction-controls #template-name{
    margin-bottom: 10px;
    background-color: white;
    color:initial;
}
#space-right{
    margin-right:30px;
}
/* Spacing for the Login Area. */
#login-area{
    padding:50px;
}

```

frontjava.js

```

function valEmail(email){ //Validates the string inputed to make sure it fits the email forma using regex.
    const check = /^[a-z0-9]+@[a-z0-9]+(\.[a-z]+)+$/;
    return check.test(String(email).toLowerCase());
}
function valName(name){ //makes sure string input starts with a capital to fit name format usign regex.
    const check = /^[A-Z][a-z]*$/;
    return check.test(String(name));
}

```

frontstyle.css

```

body{
    margin:0;
    padding:0;
    height:100%;
}
html{
    height:100%;
}
/*Header stlyes*/
header{
    display:flex;
    justify-content: center;
    align-items: center;
    background-color: #c06400;
    margin:0px;
}
header p{
    cursor: pointer;
    padding:20px;
    padding-top: 10px;
}
header h1{
    padding:20px;
    padding-top: 10px;
}
.nodecontrol{
    display: none;
}

```

```

/*Styles for sign up forms*/
.signup-form{
padding:30px;
display:flex;
flex-direction: column;
width:50%;
}
.signup-form input{
height:30px;
margin-top:10px;
margin-bottom:10px;
}
.signup-form select{
height:30px;
margin-top:10px;
margin-bottom:10px;
}
}

```

encryption.php

```

<?php namespace admin;

class Encryption{

    static function decrypt($cypherTxt){ //Decrypting the cypher text using the one time pad key stored in a text file on
the server.

        $key = file_get_contents("encryptionkey.txt"); // Get key.
        if (strlen($cypherTxt) != strlen($key)){ // If the cypher text might lose the space value if its at the end of th
string - this reverts this problem.
            $cypherTxt .= ' ';
        }
        for ($i = 0; $i < strlen($key); $i++){ // Decrypting algorithm. Shfiting the ascii values back to the plain text a
ccording to the key.

            $ascii[$i] = ( (int) ord ( $cypherTxt[$i] )) - ( (int) ord ( $key[$i] ));
            while ((int)$ascii[$i] < 33){ //Values must stay between the ascii values 33 and 126 so they can be stored in
a string.
                $ascii[$i] = 33 - ((int)$ascii[$i]);
                $ascii[$i] = 126 - (int) $ascii[$i];
            }
        }
        $plainTxt = '';
        for ($x = 0; $x < count($ascii); $x++){ //assigning the asscii values onto plain text converting to a char.
            $plainTxt .= chr ((int) $ascii[$x]);
        }
        return $plainTxt;
    }

    static function encrypt($plainTxt){

        $length = strlen($plainTxt);
        $characters = '!"#$%&\'()*+,-./0123456789;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~'; //all possible characters that can be
a part of the one time pad key
        $charactersLength = strlen($characters);
        $rndKey = '';
        for ($i = 0; $i < $length; $i++) {

```



```

        $rndKey .= $characters[rand(0, $charactersLength - 1)];
    }
    file_put_contents("encryptionkey.txt", $rndKey); //store generated one time key
    $ascii = array();
    for ($i = 0; $i < strlen($rndKey); $i++){ //encryption algorithm
        $ascii[$i] = ( (int) ord ( $plainTxt[$i] ) ) + ( (int) ord ( $rndKey[$i] ) );
        while ((int)$ascii[$i] > 126){ //Values must stay between the ascii values 33 and 126 so they can be stored in
a string.
            $ascii[$i] = 33 + ((int)$ascii[$i] - 126);
        }
    }
    $cypherTxt = '';
    for ($x = 0; $x < count($ascii); $x++){
        $cypherTxt .= chr ((int) $ascii[$x]);
    }
    return $cypherTxt;
}
}

```

Bibliography

GitHub. (n.d.). *PHPMailer* . Retrieved from GitHub: <https://github.com/PHPMailer/PHPMailer>

w3Schools. (n.d.). *How TO - CSS/JS Modal*. Retrieved from w3Schools:

[ahttps://www.w3schools.com/howto/howto_css_modals.asp](https://www.w3schools.com/howto/howto_css_modals.asp)

w3Schools. (n.d.). *PHP File Upload*. Retrieved from w3Schools: https://www.w3schools.com/php/php_file_upload.asp

w3Schools. (n.d.). *PHP MySQL Insert Data*. Retrieved from w3Schools:

https://www.w3schools.com/php/php_mysql_insert.asp

w3schools. (n.d.). *HTML Drag and Drop API*. Retrieved from w3schools.:

https://www.w3schools.com/html/html5_draganddrop.asp