Local Signup/Login Website

Manual

By Karan Arora

Overview

In this lab, you will learn how to create a website locally hosted on a virtual machine (VM). The function of this website will be to allow users to create, sign-in, and delete accounts that will be stored on a database in the VM. However, these aren't the only things the website will be able to accomplish. Let's explore the webpage and the code behind it.

Table of Contents

I. Website Interface

II. Environment-

   a. Installing PHP, phpMyAdmin, MySQL, and Apache

   b. Installing VirtualBox's Guest Additions and Configuring Your VM

   c. Configuring the SMTP Server

III. Code-

   a. Database Schema

   b. Storing Data

   c. Sending Emails and Authentication

   d. Removing Inactive Accounts

IV. Appendix-

   a. `createhandler.php`

   b. `signhandler.php`

   c. `linkhandler.php`

   d. `delete.php`

   e. `remove.php`

   f. `cron.txt`

   g. `signout.php`

   h. `index.php`

   i. `email.php`

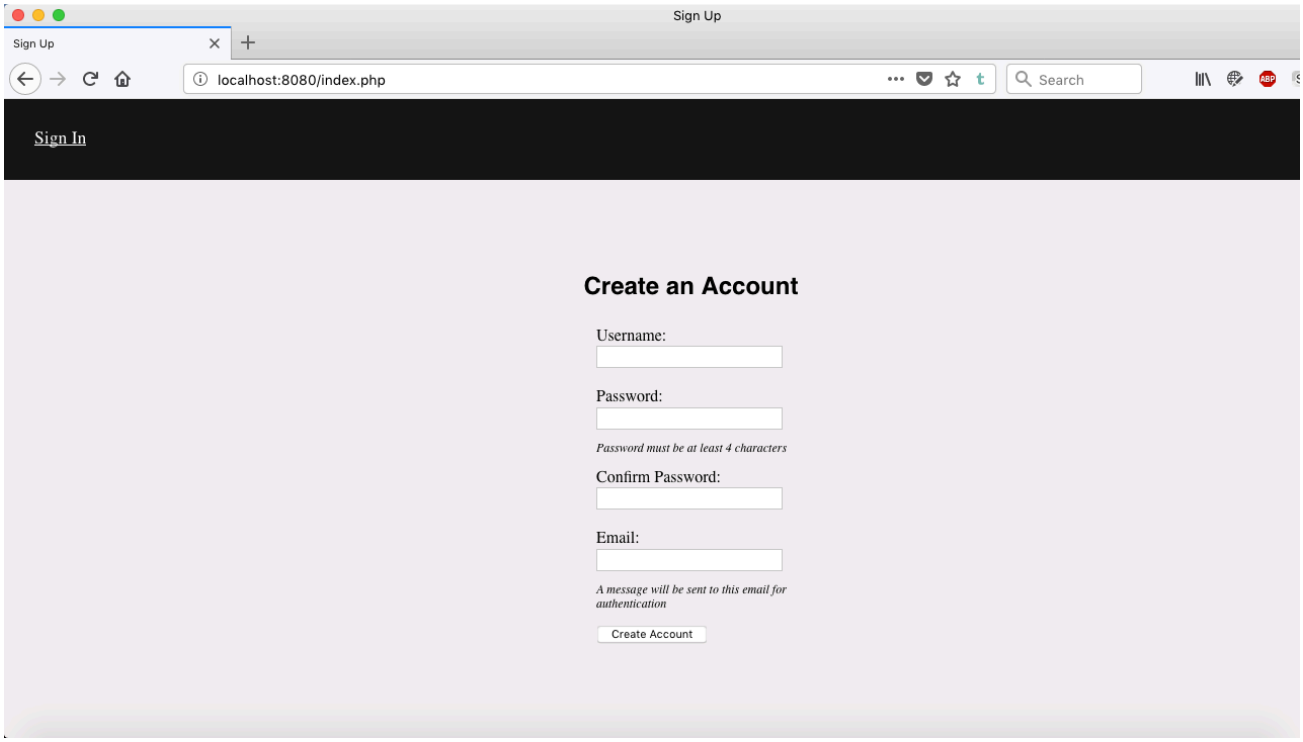
   j. `signin.php`

   k. `home.php`

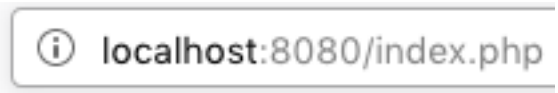
   l. `main.css`

Website Interface

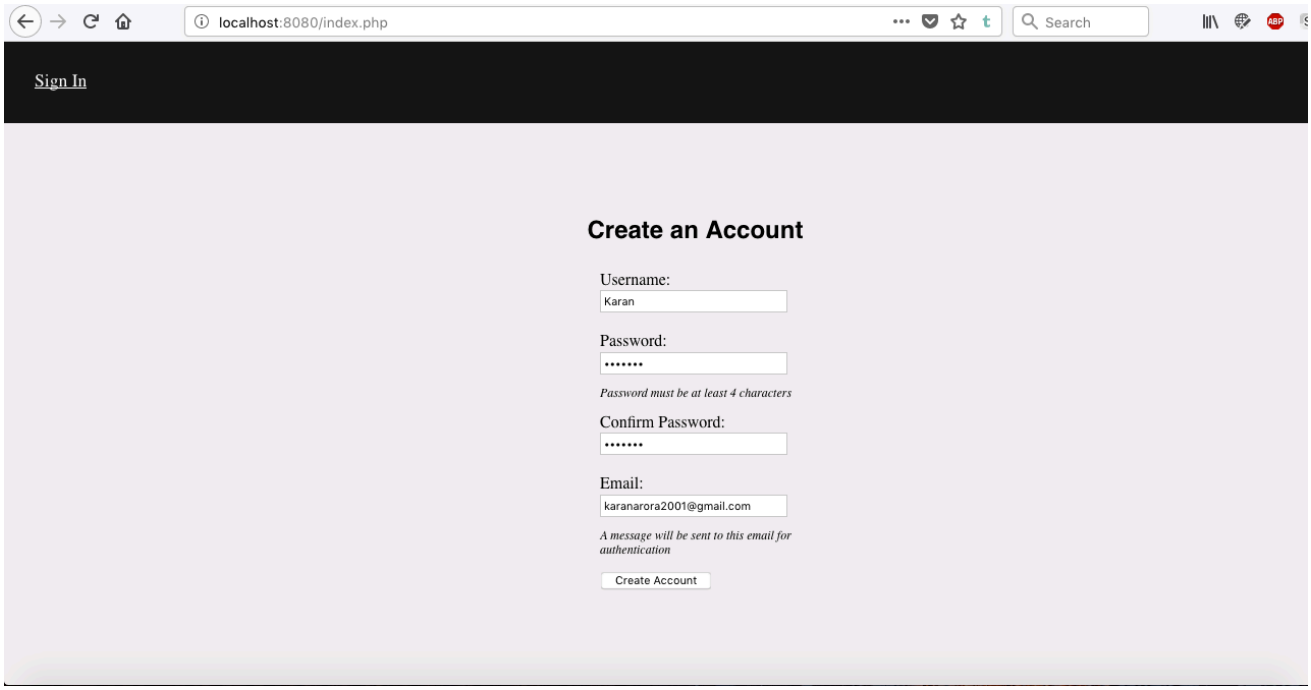
There are three main pages on this website: the account-creation page, the sign-in page, and the user's homepage. As its name implies, the account-creation page allows new users to create an account on the website. To create an account, a username, a password, and an email must be entered.

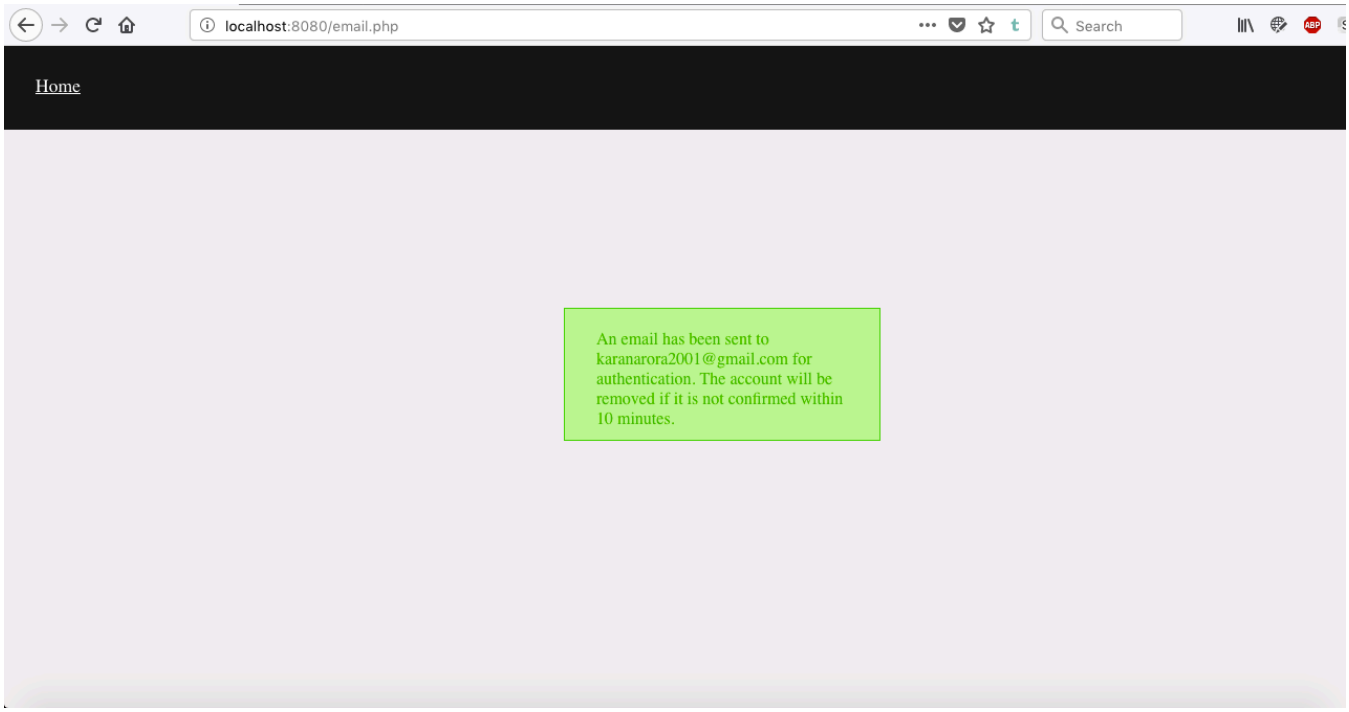


Notice how the URL contains the address `localhost:8080`, different from a normal web address. This signifies that the server for the webpage is locally hosted: it is on your computer. `8080` is the port number being forwarded to the VM, something we will discuss more in the next section. `index.php` is the file being accessed.

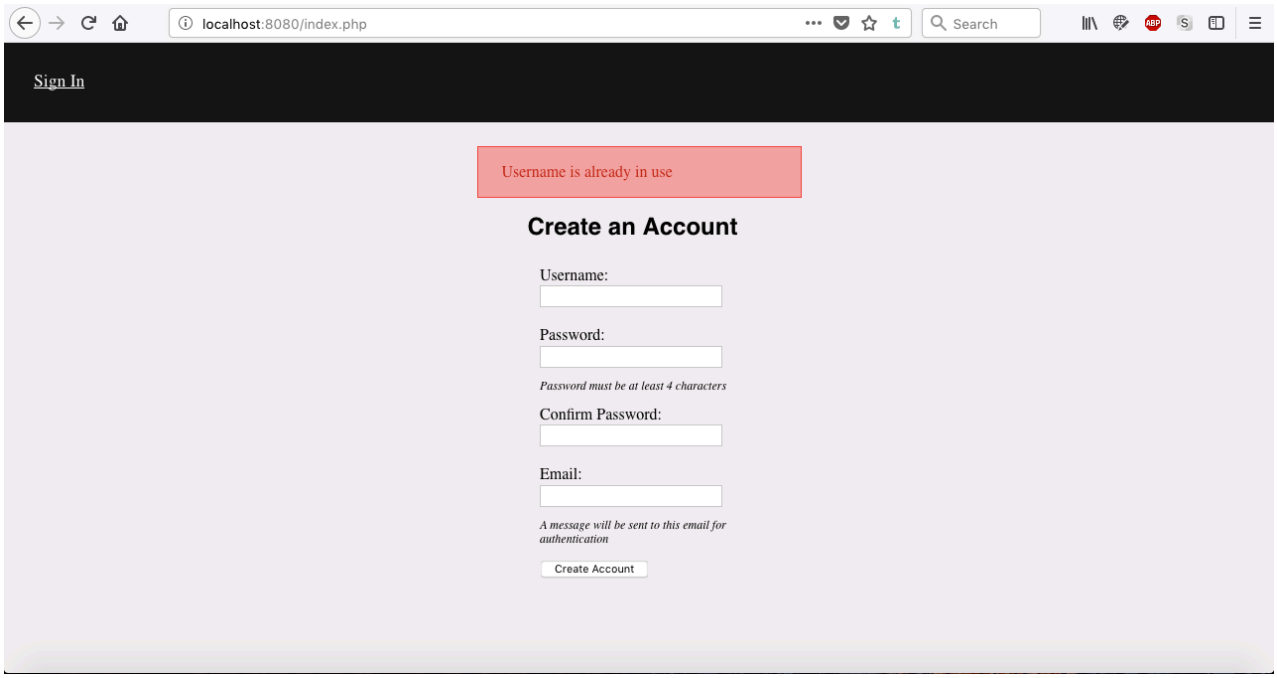


When the information is entered and the `Create Account` button is clicked, the user is taken to page stating that an email was sent to the address provided.

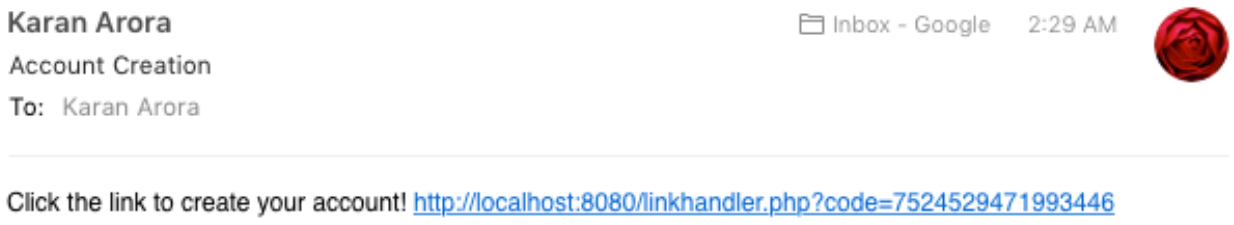




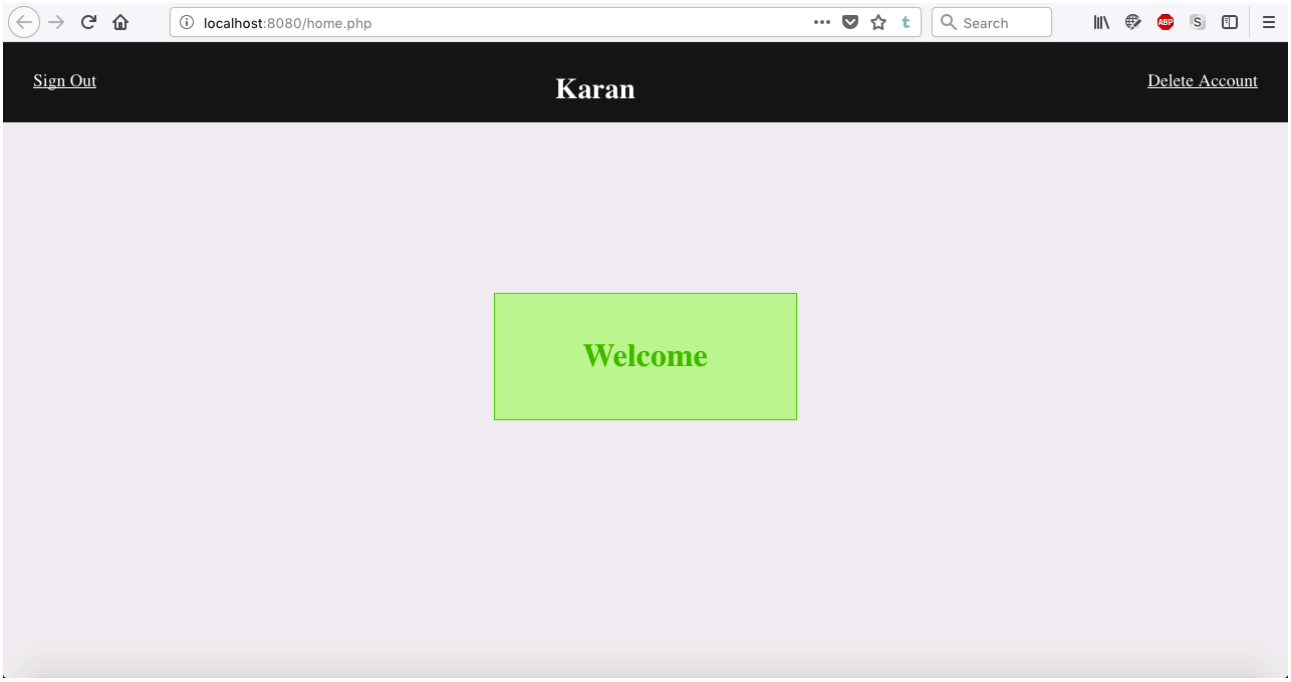
If the information given was inappropriate (i.e. the username was already taken), an error message is generated for the user instead.



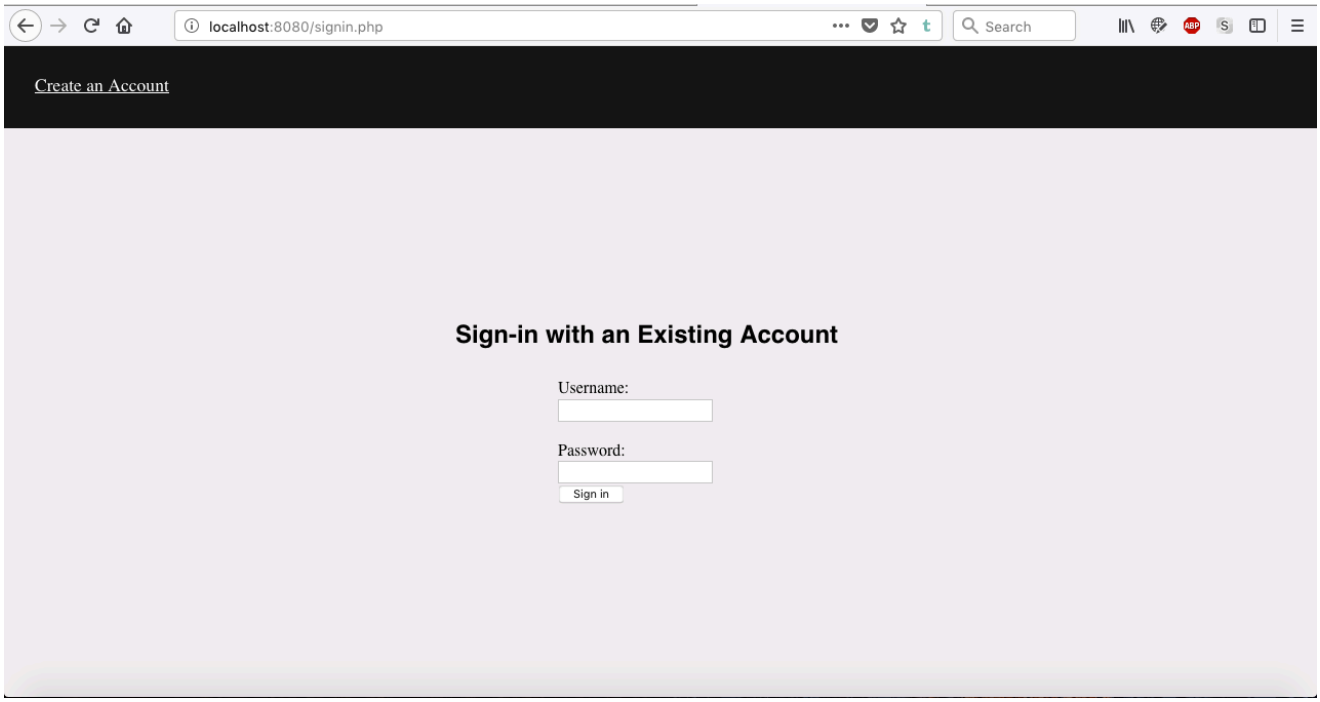
However, if the information is fine, the user's account is created but has a status of inactive. In order to activate his account, the user must check his email and click the link provided. (**Note: Because the website is locally hosted, the link must be opened on the same machine where the VM is running.)** If an account is not activated within ten minutes, it is deleted.



Once the user clicks this link, his account is active and will not be removed. Then, user is automatically logged in, and his homepage appears.



Now that he has an account, the user can log-in by entering his credentials through the sign-in page.



Like the account-creation page, an error message appears when the information entered is incorrect.

When the user is logged-in, he is able to delete his account or log-out. If the user deletes his account, he is not able to log-in using the same credentials, and would have to create a new one. A user can log-out to allow other people to access their own accounts, or to prevent his information from being viewed by anyone else who uses the same computer.

A feature that is not shown upfront about the website is its capability to maintain a session, or meeting. This means that if a user exits his browser or tab without logging-out, he is able to reopen the webpage and be automatically logged-in to his account and directed to his homepage.

Environment

In order to set up the website, you must have the proper tools installed and configured. These instructions were created using VirtualBox, version 5.1.22, and the Ubuntu operating system, version 16.04. This manual will not detail how to create the virtual machine, but links to the proper downloads for VirtualBox and Ubuntu can be found at https://www.virtualbox.org/wiki/Download_Old_Builds_5_1 and https://www.ubuntu.com/download/desktop. In addition to having a virtual machine, you will need to:

1. Install and configure PHP, phpMyAdmin, MySQL, and Apache on your VM
2. Configure VM for port forwarding, and install Virtual Box's Guest Additions so your host machine and VM can share files
3. Configure your VM's mail settings so PHP's `mail()` is functional

Installing PHP, phpMyAdmin, MySQL, and Apache-

In the world of webpages, PHP is commonly used as a backend for websites. That is, PHP provides the instructions that tell the server what to do when a request is sent or a form is submitted, for example. phpMyAdmin is a powerful tool that will allow us to view and edit our MySQL databases through a webpage interface. MySQL is a language used to communicate to databases, allowing one to edit, select, insert, and delete data, to name just a few of its capabilities. Apache is the free and open-source webserver software we will be using to host the webpage. As you will see, when all of these resources are used together, some amazing things can be created.

First, to install PHP, launch and log-in to your VM. Open the terminal, and enter:

```
sudo apt-get update
```

to update any packages already on your machine. Next, type:

```
sudo apt-get install php7.0
```

Answer 'y' to any questions asked by the terminal for all commands unless told to do otherwise.

Now, enter:

```
sudo apt-get install apache2 libapache2-mod-php7.0
```

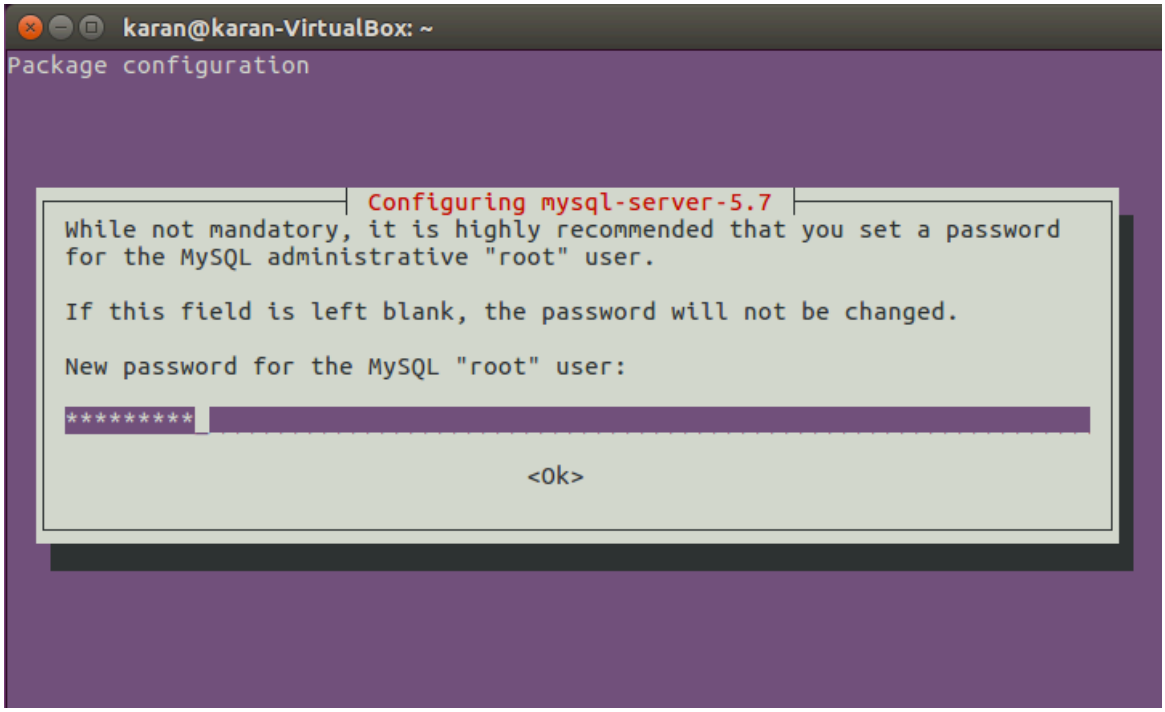
to install Apache.

Finally, to get the packages needed for MySQL, type:

```
sudo apt-get install php7.0-mysql mysql-server
```

```
sudo apt-get update
sudo apt-get install php7.0
sudo apt-get install apache2 libapache2-mod-php7.0
sudo apt-get install php7.0-mysql mysql-server
```

Enter whatever password you want when the configuration box appears. Just make sure you don't forget it! Notice that your username is preset to "root".

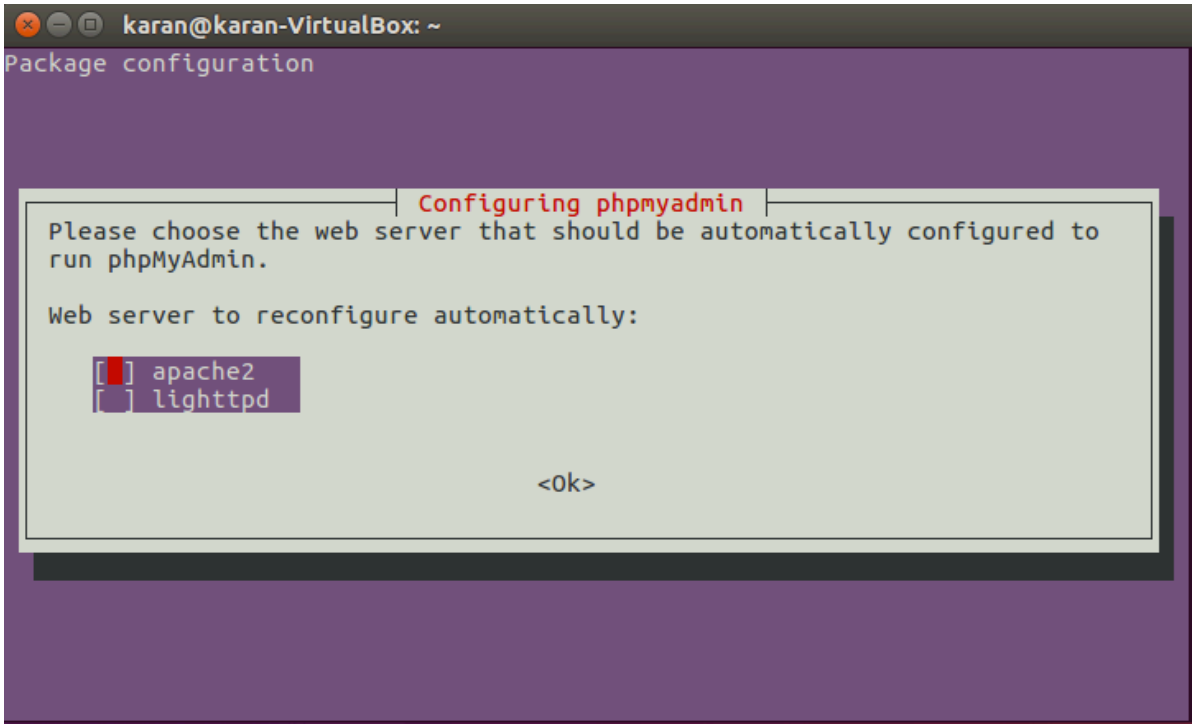


Press the `down arrow` key and then `enter` to select `<Ok>`.

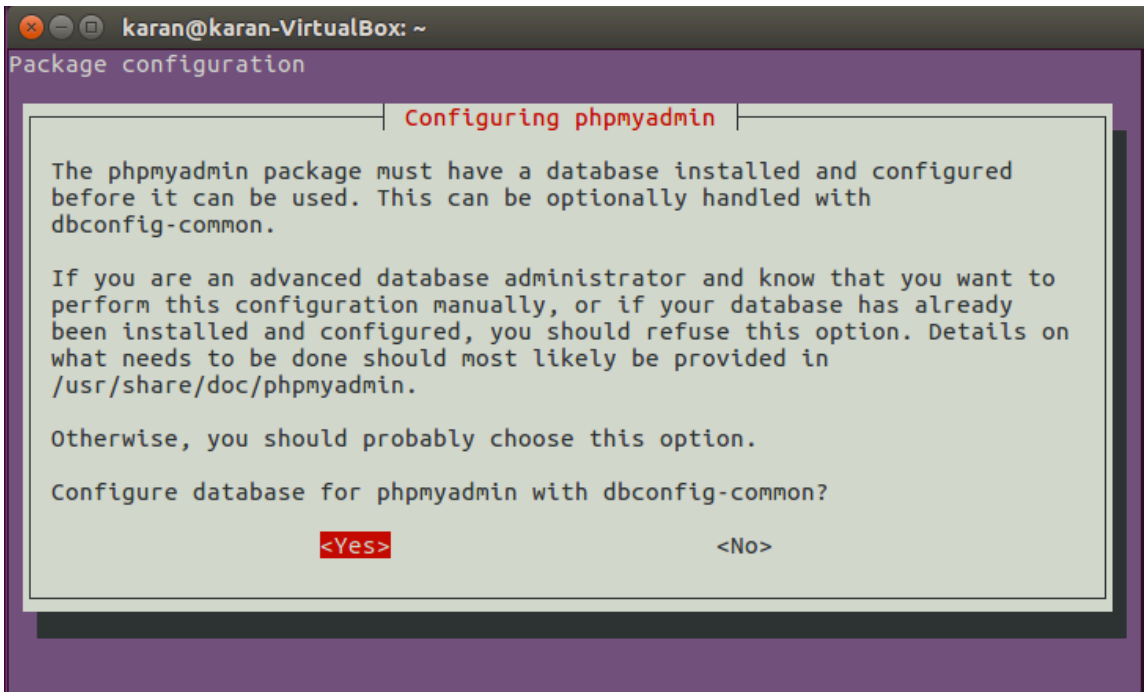
Now, to get everything required for phpMyAdmin, enter:
`sudo apt-get install phpmyadmin php-mbstring php-gettext`

```
sudo apt-get install phpmyadmin php-mbstring php-gettext
```

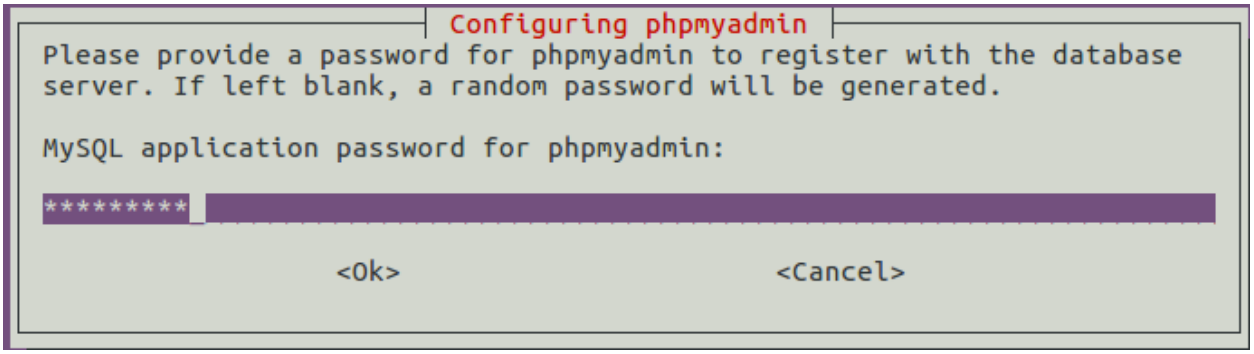
When the installation prompts you to choose between `apache2` or `lighttpd`, choose `apache2` by hitting `enter` when the red box is next to it:



Then, when you're asked to configure phpMyAdmin with `dbconfig-common`, select `Yes`:



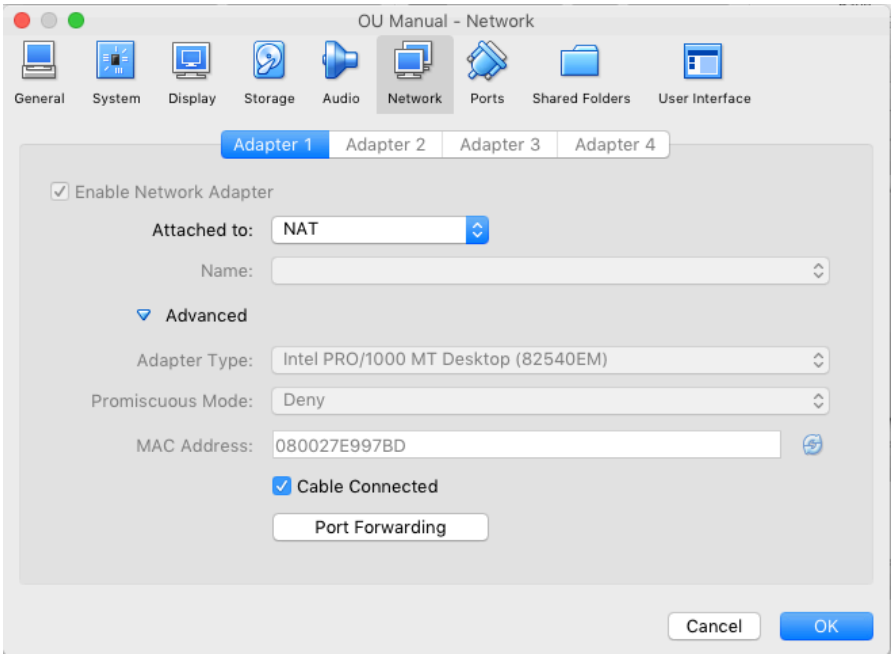
Next, enter the password you previously provided when installing MySQL:

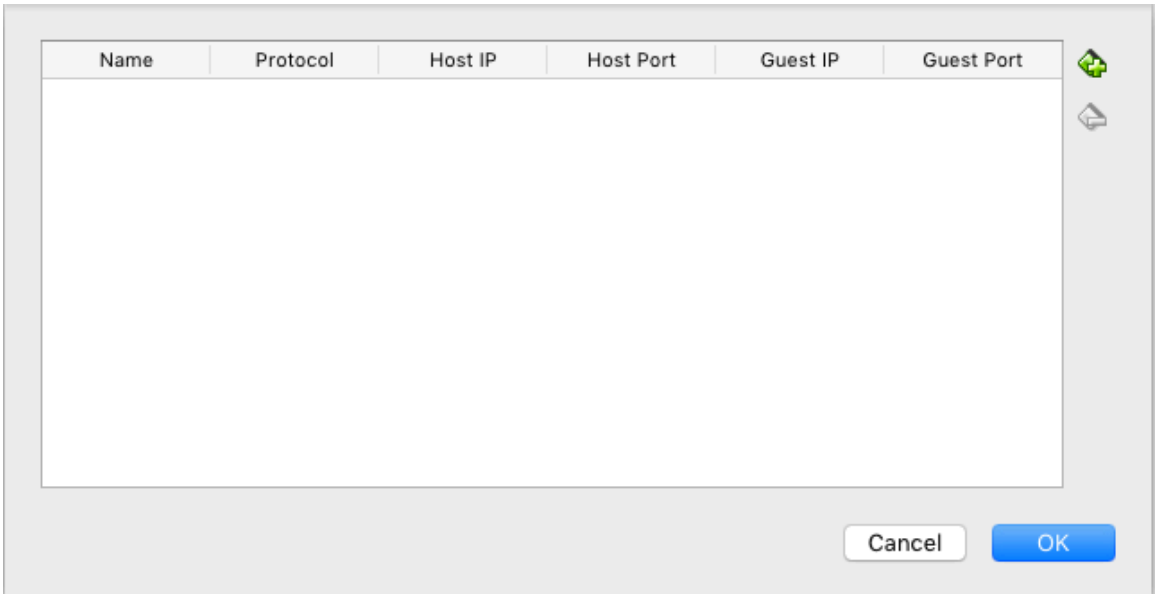


Finally, after phpMyAdmin is done installing, configure PHP and Apache 2 by entering:
`sudo a2enmod rewrite`

Installing VirtualBox's Guest Additions and Configuring Your VM-

Before we can start accessing our new Apache server, many more things need to be configured. First, click on your VM's settings in the VirtualBox menu. From the settings, click on Network < Advanced < Port Forwarding.

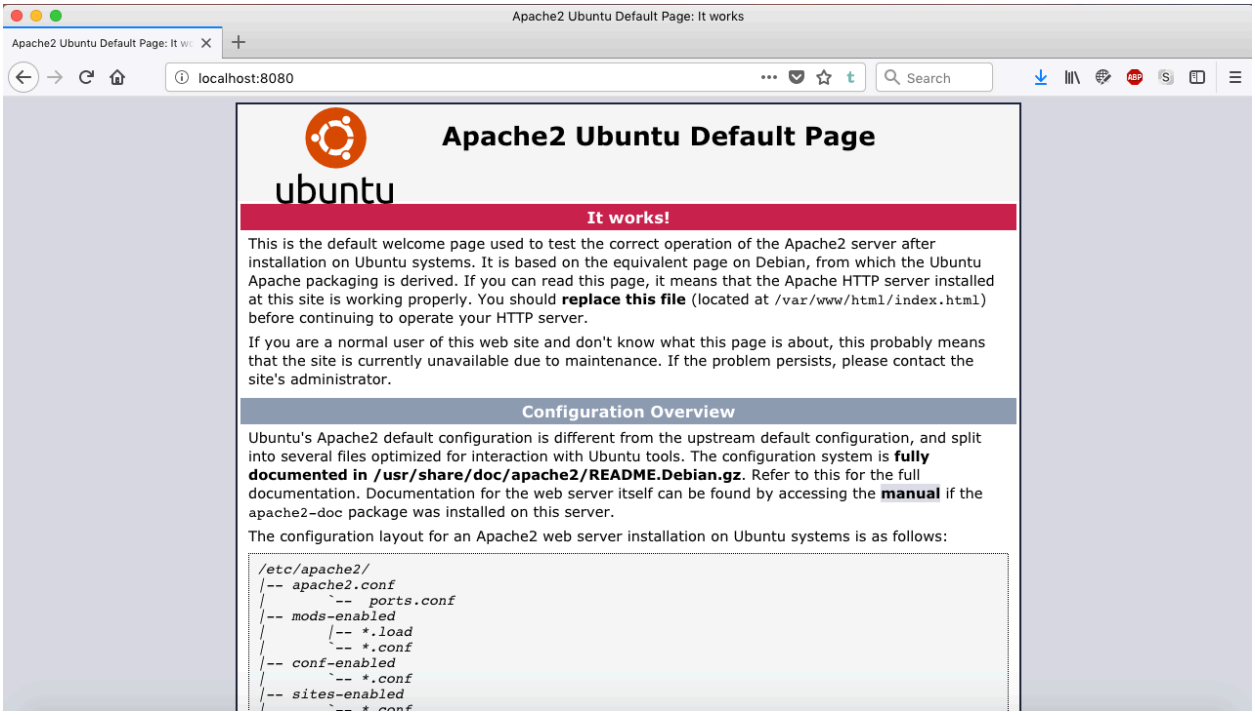




Then, click the green plus on the right, and change the host port number field to `8080` and the guest port number section to `80`. This configuration tells your host machine (your physical computer) that all requests sent to the port `8080` from your browser should be given to your VM through port `80`, the port for the `http` protocol. This protocol is used to establish connections over the internet. This explains the link from the first section, `localhost:8080/index.php`. The `:8080` tells the browser to access port `80` of `localhost`, which directs to our Apache server. From there, the server delivers the content `index.php` to the user.

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
http	TCP		8080		80

Now, if you type in `localhost:8080` on your host machine, the default page for Apache should appear!

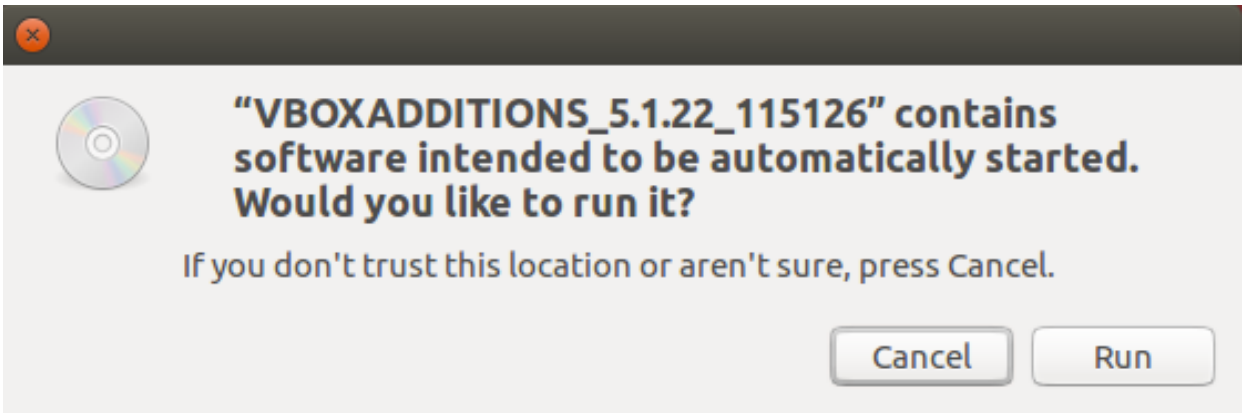


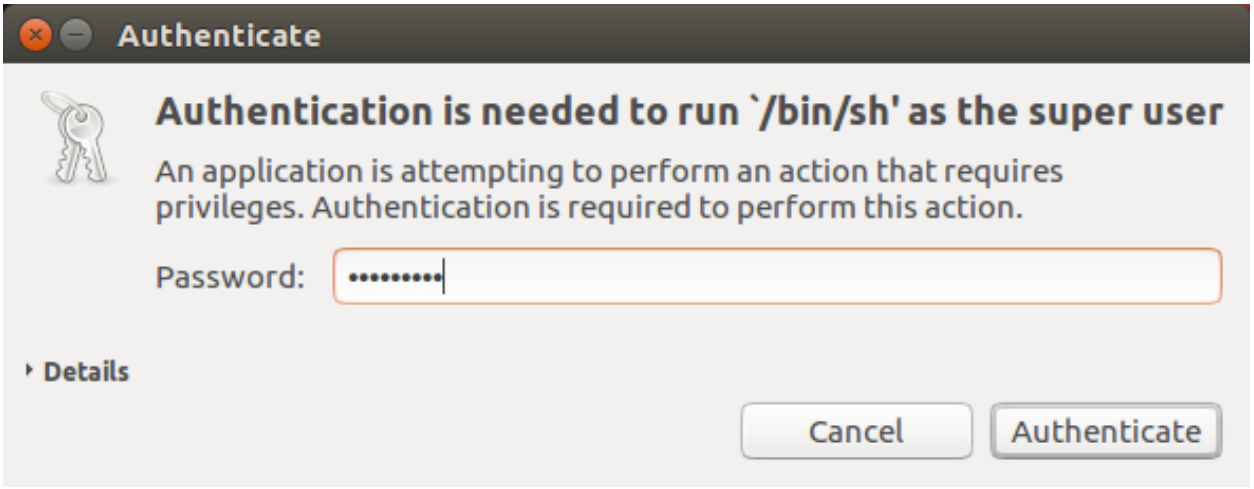
In order to include your own webpages on the server, we have to share files from your host machine to the VM. First, run the command:

`sudo apt-get install build-essential module-assistant`

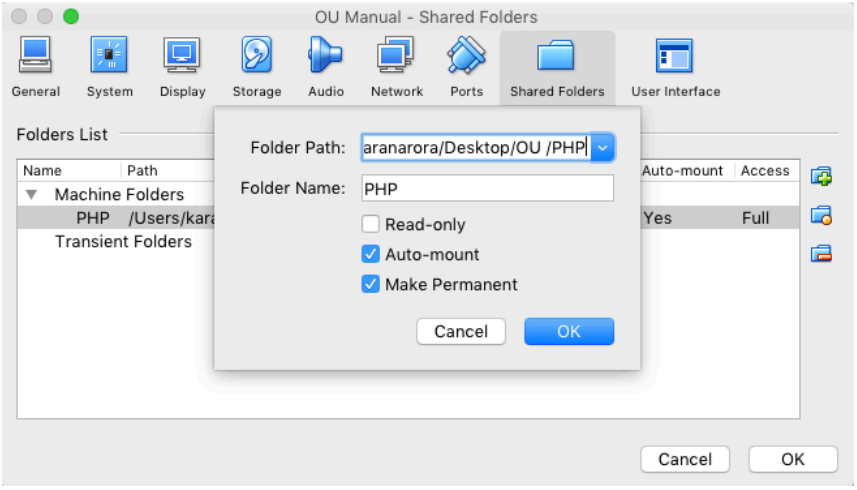
```
sudo apt-get install build-essential module-assistant
```

This sets up the packages needed for Guest Extensions to work on the VM, which are required to share files. Again, answer 'y' to any questions asked. Then, select the `Devices` tab at the top of your screen, and click `Insert Guest Additions CD Image`. Enter your passcode, and wait for the software to install. Restart your VM.





Now, create a folder on your host machine that will store all of your scripts. Afterwards, access the VM's settings from the VirtualBox menu again, and go to `Shared Folders`. Then, click on the button with a green plus on it on the right, and enter that folder's path. Finally, select the `Automount` and `Make Permanent` options.



Go back to your VM, and enter:

```
sudo mount -t vboxsf <YourFolderName> /var/www/html
```

into terminal. If the command executes without any errors, it was performed successfully. `/var/www/html` is a location accessed by Apache. Whatever content inside that folder will be opened by your browser.

In my case, the command would look like:

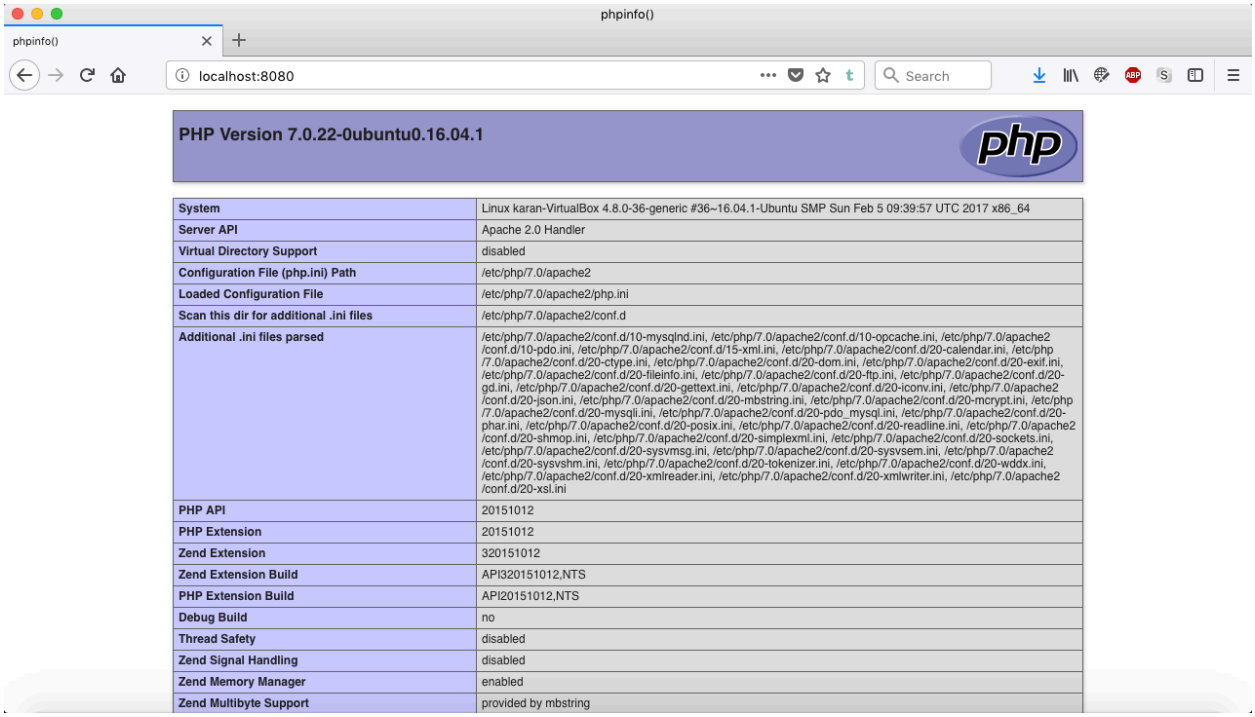
```
sudo mount -t vboxsf PHP /var/www/html
```

Create a PHP file using any text editor and type:

```
<?php print phpinfo(); ?>
```

Name this file `index.php`, and save it in the shared folder.

Refresh `localhost:8080`, and a new page displaying all your system information should appear!

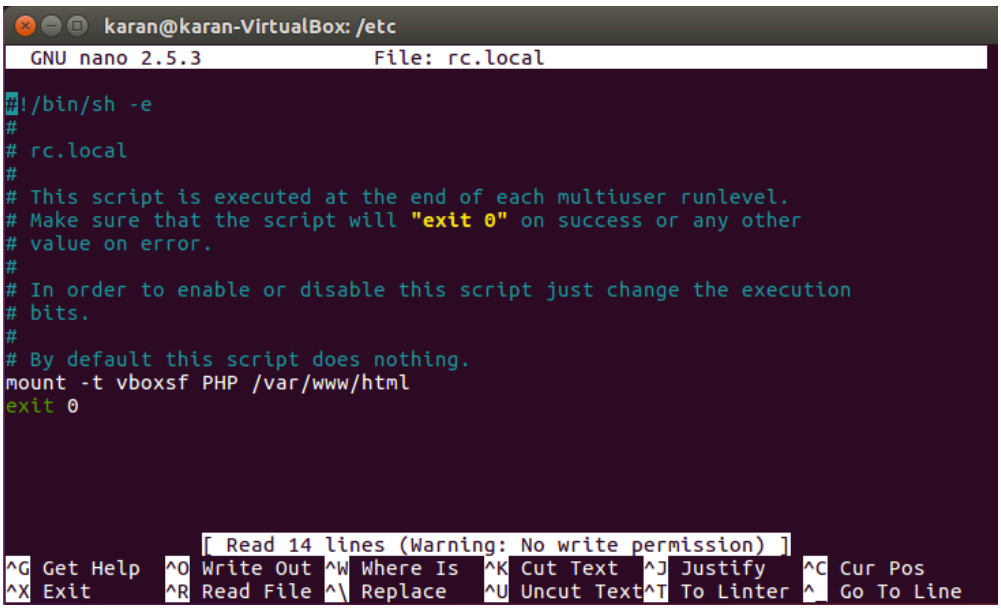


System	Linux karan-VirtualBox 4.8.0-36-generic #36~16.04.1-Ubuntu SMP Sun Feb 5 09:39:57 UTC 2017 x86_64
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/apache2
Loaded Configuration File	/etc/php/7.0/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/apache2/conf.d
Additional .ini files parsed	/etc/php/7.0/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.0/apache2/conf.d/10-opcache.ini, /etc/php/7.0/apache2/conf.d/10-pdo.ini, /etc/php/7.0/apache2/conf.d/15-xml.ini, /etc/php/7.0/apache2/conf.d/20-calendar.ini, /etc/php/7.0/apache2/conf.d/20-ctype.ini, /etc/php/7.0/apache2/conf.d/20-dom.ini, /etc/php/7.0/apache2/conf.d/20-exif.ini, /etc/php/7.0/apache2/conf.d/20-fileinfo.ini, /etc/php/7.0/apache2/conf.d/20-ftp.ini, /etc/php/7.0/apache2/conf.d/20-gd.ini, /etc/php/7.0/apache2/conf.d/20-gettext.ini, /etc/php/7.0/apache2/conf.d/20-iconv.ini, /etc/php/7.0/apache2/conf.d/20-json.ini, /etc/php/7.0/apache2/conf.d/20-mbstring.ini, /etc/php/7.0/apache2/conf.d/20-mcrypt.ini, /etc/php/7.0/apache2/conf.d/20-mysqli.ini, /etc/php/7.0/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.0/apache2/conf.d/20-phar.ini, /etc/php/7.0/apache2/conf.d/20-posix.ini, /etc/php/7.0/apache2/conf.d/20-readline.ini, /etc/php/7.0/apache2/conf.d/20-shmop.ini, /etc/php/7.0/apache2/conf.d/20-simplexml.ini, /etc/php/7.0/apache2/conf.d/20-sockets.ini, /etc/php/7.0/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.0/apache2/conf.d/20-sysvsem.ini, /etc/php/7.0/apache2/conf.d/20-sysvshm.ini, /etc/php/7.0/apache2/conf.d/20-tokenizer.ini, /etc/php/7.0/apache2/conf.d/20-wddx.ini, /etc/php/7.0/apache2/conf.d/20-xmlreader.ini, /etc/php/7.0/apache2/conf.d/20-xmlwriter.ini, /etc/php/7.0/apache2/conf.d/20-xsl.ini
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS
PHP Extension Build	API20151012,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring

Now, so you don't have to repeat this process every time you turn on the VM, go to your root directory by typing `cd /` in terminal. Then enter `cd etc` to go to the `etc` directory. Type `sudo nano rc.local` to edit the `rc.local` file.

```
cd /
cd etc
sudo nano rc.local
```

Again add the command `mount -t vboxsf <YourFileName> /var/www/html` to it like so:



Hit `ctrl+X`, then `y`, then `enter` to save your changes.

Finally, in order to get phpMyAdmin running, again go to your root directory by typing `cd /` . Enter `cd etc` and then `cd apache2`, and configure `apache2.conf` by typing `sudo nano apache2.conf`.

```
cd /
cd etc
cd apache2
sudo nano apache2.conf
```

At the end of the file, append the command:
`Include /etc/phpmyadmin/apache.conf`

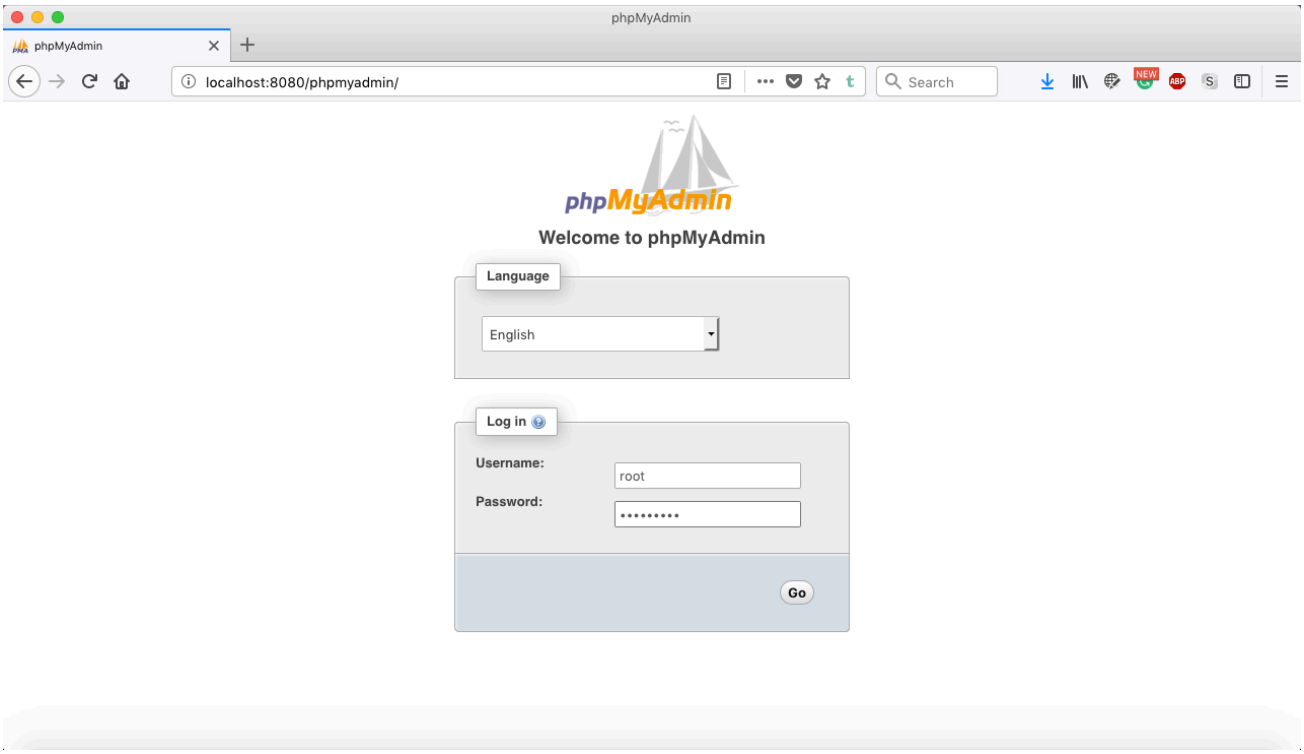
```
# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
Include /etc/phpmyadmin/apache.conf
```

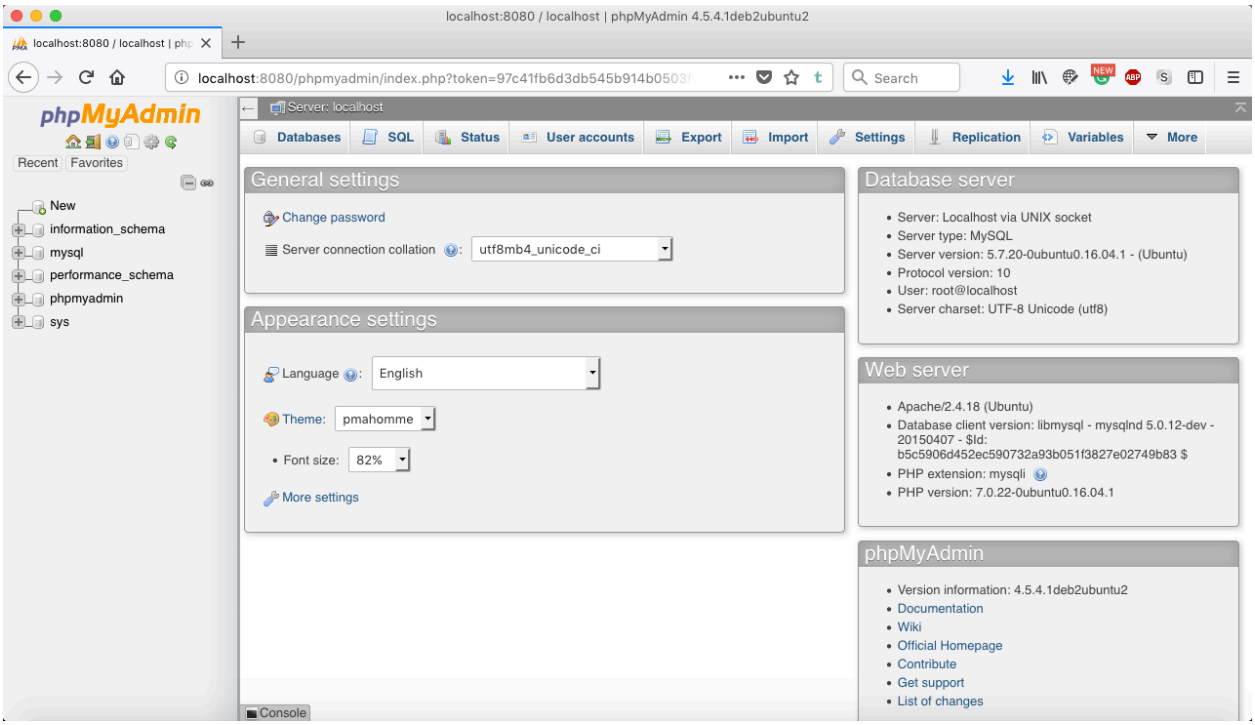
Like you did before, save and close the file. Restart Apache by going back to your root directory and entering:

`/etc/init.d/apache2 restart`

```
/etc/init.d/apache2 restart
```

Provide your password if necessary. Now, if you visit `localhost:8080/phpmyadmin`, a log-in screen for the service should appear. With a username of `root` and the password you chose beforehand while installing the software, sign-in to phpMyAdmin. Here, you can view and edit all of your databases, something we will do more in a later section.




localhost:8080 / localhost | phpMyAdmin 4.5.4.1deb2ubuntu2
localhost:8080/phpmyadmin/index.php?token=97c41fb6d3db545b914b0503
Search
phpMyAdmin
Recent
Favorites
New
information_schema
mysql
performance_schema
phpmyadmin
sys
Server: localhost
Databases
SQL
Status
User accounts
Export
Import
Settings
Replication
Variables
More
General settings
Change password
Server connection collation:
utf8mb4_unicode_ci
Appearance settings
Language:
English
Theme:
pmahomme
Font size:
82%
More settings
Database server
Server: Localhost via UNIX socket
Server type: MySQL
Server version: 5.7.20-0ubuntu0.16.04.1 - (Ubuntu)
Protocol version: 10
User: root@localhost
Server charset: UTF-8 Unicode (utf8)
Web server
Apache/2.4.18 (Ubuntu)
Database client version: libmysql - mysqlnd 5.0.12-dev - 20150407 - $Id: b5c5906d452ec590732a93b051f3827e02749b83 $
PHP extension: mysqli
PHP version: 7.0.22-0ubuntu0.16.04.1
phpMyAdmin
Version information: 4.5.4.1deb2ubuntu2
Documentation
Wiki
Official Homepage
Contribute
Get support
List of changes
Console

Configuring the SMTP Server-

As mentioned earlier, after a user enters his information, a confirmation message is sent to the email account provided. To accomplish this task, we use PHP's built-in `mail()` function. Before this function is operational, a mail server needs to be set up on the system for it to use. The purpose of this server is to handle the message and send it to the appropriate location. Mail servers communicate using the `Simple Mail Transfer Protocol`, or `SMTP`.

Instead of installing a mail server, though, we will set up Simple SMTP (sSMTP), a lightweight mail transfer agent (MTA) that will communicate and relay messages to a mail server, which will then do the work of sending the messages to the proper place. **(Note: We will be using a Gmail account to work with sSMTP. If you do not have an account, create one before moving forward.)**

To get started, enter the command:

```
sudo apt-get install ssmtp
```

Next, run:

```
sudo nano /etc/ssmtp/ssmtp.conf
```

to edit the configuration of the program.

Before we start editing the file, know that there **cannot** spaces between anything in a line. For example, writing `name= John` or `name = John` would not be interpreted correctly by your computer. You would have to say `name=`John, without a space between any of the characters. Forgetting this rule is an easy way to create an incorrect configuration file and spend hours wondering what error you could have made. I say this from personal experience.

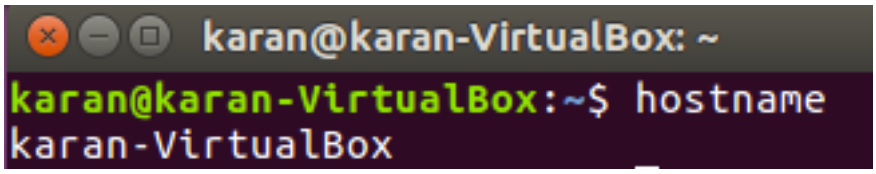
In the row that says `root`, replace `postmaster` with your email username/address. Where it says `mailhub`, replace `mail` with `smtp.gmail.com:587`, which is the address of the `SMTP` server

we will be relaying to. Now, after this line, manually type and add the fields `AuthUser`, `AuthPass`, `UseTLS`, and `UseSTARTTLS`:

```
mailhub= smtp.gmail.com:587

AuthUser=
AuthPass=
UseTLS=
UseSTARTTLS=
```

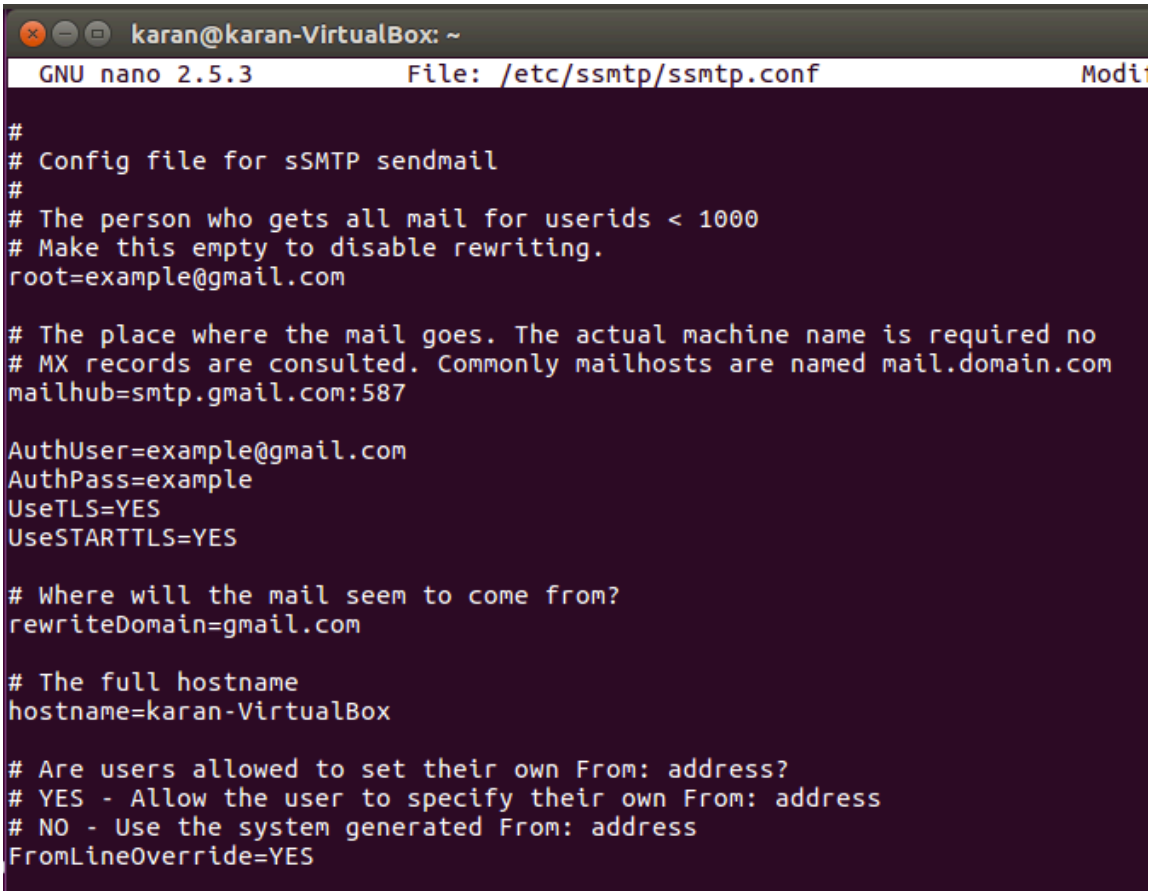
For `AuthUser`, enter your email username/address once again. The `AuthPass` field should contain this account's password. Then, enter `YES` for both `UseTLS` and `UseSTARTTLS`.

Now, fill in the `rewriteDomain` field with `gmail.com`. Make sure to delete the `#` before this field. If the character is left there, your machine will ignore this line. If your `hostname` row is already filled in, leave it as is. However, if it is not, enter the hostname of your machine. This can found by conveniently typing `hostname` into the command line.



Finally, where it says `FromLineOverride`, delete the `#` at the beginning and replace `NO` with `YES`. This allows the user to specify the `From:` field at the top of emails. Again, make sure all of the previous statements you typed have no spaces in between them.

At the end of this process, your `ssmtp.conf` file should look like so:



Save the file and exit out of it.

Now, to make PHP compatible with this program, we must edit apache's `php.ini`:

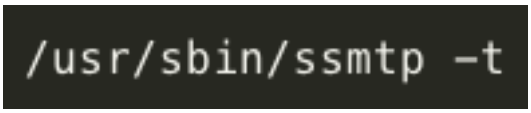
```
sudo nano /etc/php/7.0/apache2/php.ini
```

To find the line we need to edit in this large file, type `ctrl+w`, which activates a search macro within `nano`. Type the word `sendmail` in the prompted spot and then hit `enter`. The window should take you to a chunk of text that looks like this:

```
[mail function]
; For Win32 only.
; http://php.net/smtp
SMTP = localhost
; http://php.net/smtp-port
smtp_port = 25

; For Win32 only.
; http://php.net/sendmail-from
;sendmail_from = me@example.com

; For Unix only.  You may supply arguments as well (default: "sendmail -t -i").
; http://php.net/sendmail-path
;sendmail_path =

; Force the addition of the specified parameters to be passed as extra paramete$
; to the sendmail binary. These parameters will always replace the value of
; the 5th parameter to mail().
```

Where it says `For Unix only`, in the third block shown above, delete the semicolon in `;sendmail_path =`. The character makes this line ignored when the `php.ini` file is read. Now, fill in the blank field with the following path:

`/usr/sbin/ssmtp -t`



Unlike the previous file, spaces between the `=` and the statement are allowed.

Now, the block should look like so:

```
; For Unix only.  You may supply arguments as well (default: "sendmail -t -i").
; http://php.net/sendmail-path
sendmail_path = /usr/sbin/ssmtp -t
```

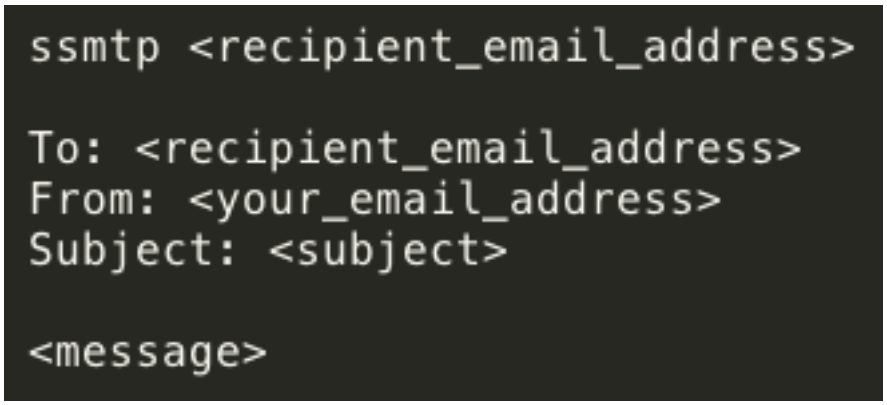
Again, save and exit the file.

To test if your MTA is set up correctly, go back to terminal and enter the command:

```
ssmtp <recipient_email_address>
```

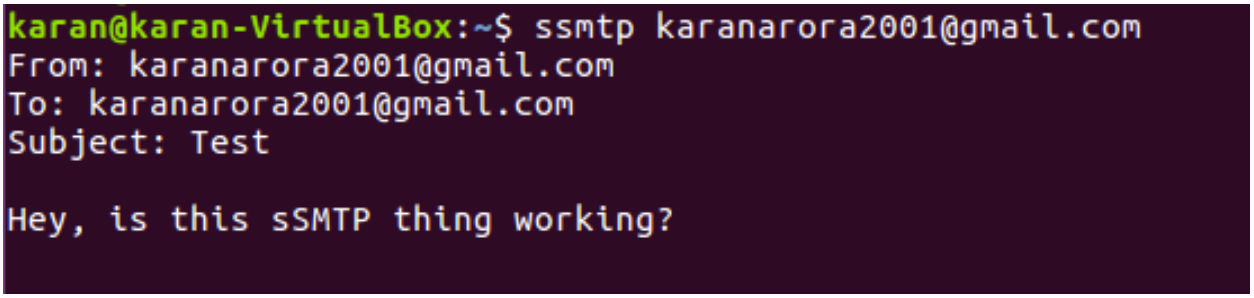
Then, hit `enter`, and type:

```
To: <recipient_email_address>
From: <your_email_address>
Subject: <subject>

<message>
```



After you are done writing your message, press `ctrl+d` to send the email.

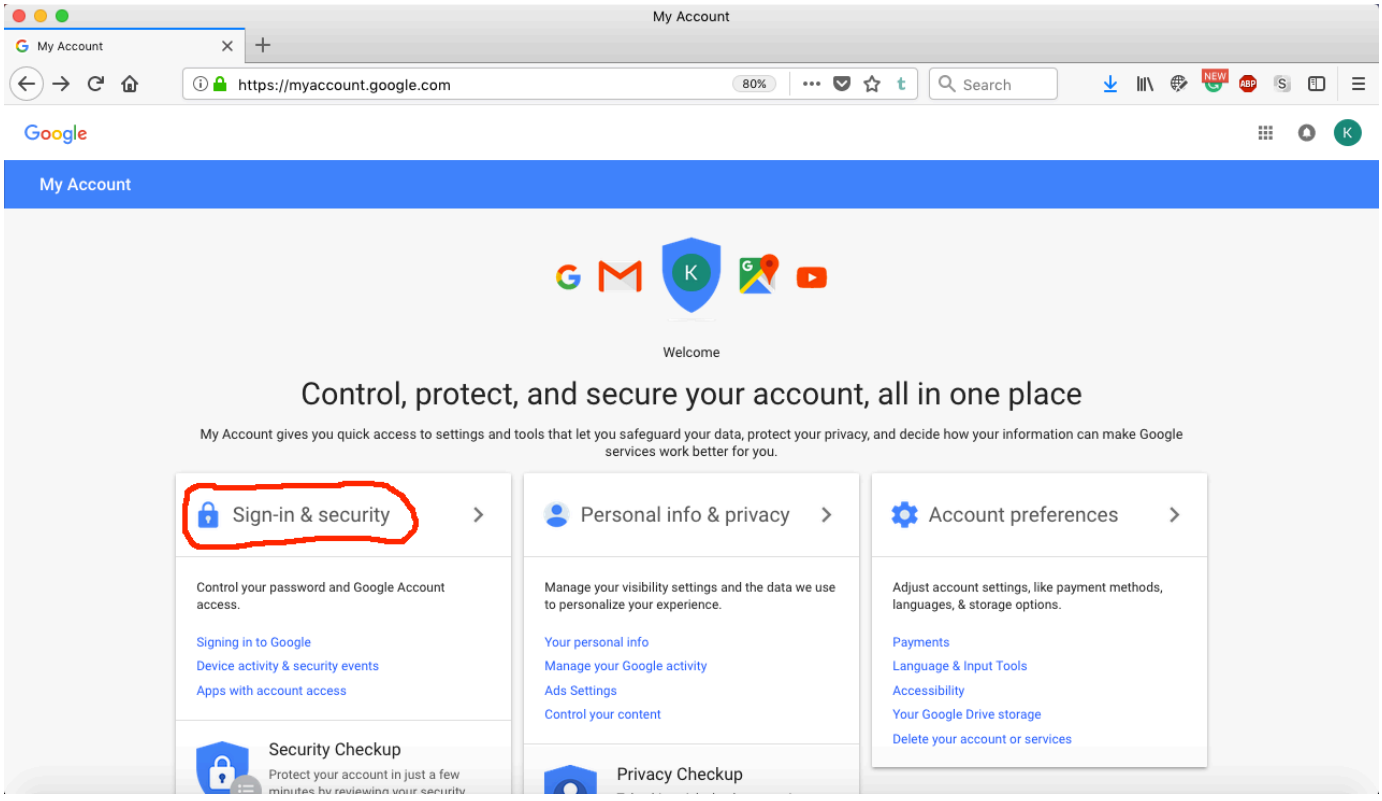
For example, if I were to send an email to myself, it would look like so:

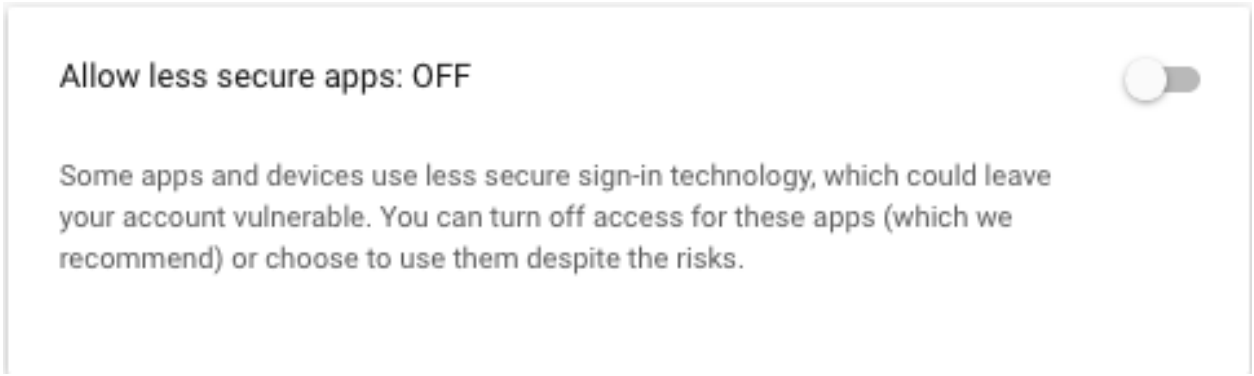


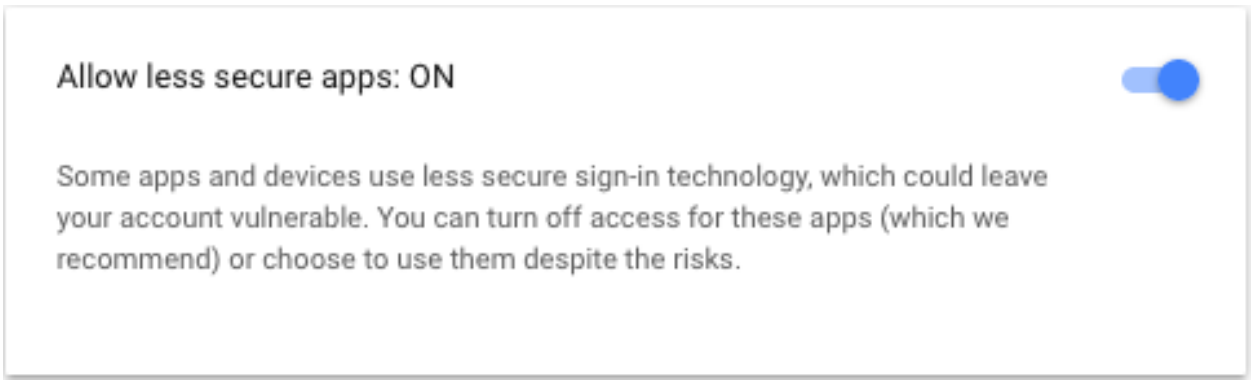
Sure enough, within a few seconds, the message appears inside my inbox:

That finally wraps up all of the configuration for your VM and MTA that is needed to make the website work. In the next part, we will look into the code that makes the site run.

(Note: If you are unable to send emails, you may have to allow less secure apps to access your account. To do so, go to `myaccount.google.com` . Then, click on `Sign-in & Security`. Scroll to the very bottom of the page, where you will find the option to allow less secure apps. Toggle the option to `ON`.)




Allow less secure apps: OFF
Some apps and devices use less secure sign-in technology, which could leave your account vulnerable. You can turn off access for these apps (which we recommend) or choose to use them despite the risks.


Allow less secure apps: ON
Some apps and devices use less secure sign-in technology, which could leave your account vulnerable. You can turn off access for these apps (which we recommend) or choose to use them despite the risks.

Code

Behind every program you interact with, there is code written by a programmer that tells the application what to do. Our website is no different. From telling the browser what content to display, to informing the server on how to handle requests from a client, to instructing the database where to store information, code written in HTML, CSS, PHP, and SQL powers our creation. In this section, we won't be exploring every line of code behind the site. Instead, we will focus on some of the more interesting features of the application:

1. Database Schema
2. Storing Data
3. Sending Emails and Email Authentication
4. Removing Inactive Accounts with Cron

Database Schema-

A database schema is the way a database is structured and how data in a system is organized. Most databases are divided into `tables`, which then contain `records`, or information, about individual items. Each column inside a record is called a `field`.

For example, in our website's database, `myDB`, there is only one table, which is named `Account`. As you can infer from the name, this table contains all of the information for a user's account. In the table, there are seven fields: `ID`, `Username`, `Password`, `Email`, `String`, `Status`, and `Time_made`. Using phpMyAdmin, we can view the database schema.

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	**ID** 🔑	mediumint(9)			No	*None*	AUTO_INCREMENT
2	**Username**	varchar(50)	latin1_swedish_ci		Yes	*NULL*	
3	**Password**	varchar(255)	latin1_swedish_ci		Yes	*NULL*	
4	**Email**	varchar(255)	latin1_swedish_ci		Yes	*NULL*	
5	**String**	varchar(255)	latin1_swedish_ci		Yes	*NULL*	
6	**Status**	varchar(50)	latin1_swedish_ci		Yes	*NULL*	
7	**Time_made**	timestamp		on update CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP

For each field, there are options you can choose to control its data. Having a `type` `mediumint()` tells the `ID` field that only integers in a certain range can be stored in it. `varchar(num)` permits a field to only hold `num` amount of characters in it. The `timestamp` `type` allows the `Time_made` field to only store timestamp objects in it, which are just objects that represent a certain date and time.

The `Null` option tells a field whether or not it can contain `NULL` information. The `Default` option instructs a field what to store if no information has been inserted into it yet. Finally, the `Extra` and `Attributes` options provide extra rules. `AUTO_INCREMENT` informs the `ID` field to increase its stored number by `1` each time a new record is created. `ON UPDATE CURRENT_TIMESTAMP` tells the `Time_made` field to store the current time and date each instance the column is updated.

To create a database and its associated tables and fields, you can either visit `localhost:8080/phpmyadmin` or do it manually via PHP and SQL code. The former option is much easier as it will give you a visual display.

There are many more options available when configuring your database. However, for the website we will be creating, these are all the ones that are needed.

Storing Data-

In order to store and interact with information in a database, SQL uses a set of commands called `queries`. Examples of SQL queries are the `INSERT` statement, which inserts data into a given location, the `SELECT` statement, which grabs information from the database, and the `DELETE` statement, which removes specified data.

One basic way of storing data is to use the `mysqli_query()` function, which is built into PHP. `mysqli` stands for `MySQL Improved`, which is just an extension of MySQL that PHP uses. By inserting a connection variable (`$conn`) and a variable that contains SQL code (`$sql`) as arguments, the function is able to "query" the database and perform the desired operation.

An example of this code being used would be:

```
mysqli_query($conn, $sql);
```

But, a huge flaw of using this function is that it allows for SQL Injection. SQL Injection is a malicious form of code injection where attackers insert their own SQL statements into a query, usually through some form of user input, thereby gaining the ability to alter the query in any way they wish. They would have the ability to delete data or even select and steal all of it without having to enter a password. As you can imagine, this would have disastrous effects if a database with important contents, such as credit card information, was targeted.

So, instead of communicating with raw SQL, which can be easily injected and transformed, we use prepared statements. In prepared statements, variables and parameters are sent separately to the server. As a result, they cannot maliciously interfere with a database.

Preparing SQL statements is only a bit longer of a process than using `mysqli_query()`:

```
$stmt= mysqli_stmt_init($conn);

mysqli_stmt_prepare($stmt, "INSERT INTO <Table> (<Field>) VALUES (?)");

mysqli_stmt_bind_param($stmt, "s", $example);

mysqli_stmt_execute($stmt);
mysqli_stmt_close($stmt);
```

The first line declares `$stmt` using `mysqli_stmt_init()` and a connection variable as an argument. This creates the prepared statement. Then, `mysqli_stmt_prepare()` uses this variable and an SQL string as arguments. In the example SQL, there is an `INSERT` statement that inserts `VALUES` into a desired table and field. The `(?)` after `VALUES` represents the data to be inserted. This `(?)` is filled in by the next function, `mysqli_stmt_bind_param()`. In this line, the `$stmt` variable is again used as an argument. Then, "`s`" is used to signal what data type the parameter, `$example`, is. "`s`" means that `$example` is a `string`. `$example` is the value that replaces the `(?)`. Finally, `mysqli_stmt_execute()` executes the query on the database, and `mysqli_stmt_close()` closes the statement. More information on this process can be found at http://us3.php.net/manual/en/mysqli.stmt-init.php.

To sum up this block of code, the value `$example`, which is a string, is being inserted into `<Table>` under the field `<Field>`. By using a prepared statement, we are protected from SQL Injection.

(Note: Although only one value and field were shown in the example, you can use more than one in `mysqli_stmt_prepare()`)

Now that you have all of this background information, we can begin looking at the code for the webpage. Specifically, let's focus on how accounts are created and stored. If you want to look beyond the snippets of code shown on here, full files can be found in the Appendix.

If you recall, the homepage, `index.php`, has a form where users can enter their information:

This form is created by the following block of code in `index.php`:

```
31          <form action="/createhandler.php" method= "POST">
32
33              Username:<br>
34              <input type="text" name="username" size="25"><br><br>
35
36              Password:<br>
37              <input type="password" name="password" size="25"><br>
38              <p class="charlimit"><i>
39              Password must be at least 4 characters</i></p>
40
41              Confirm Password:
42              <input type="password" name="confirm" size="25"><br><br>
43
44              Email:
45              <br><input type="text" name="email" size="25"><br>
46              <p class="charlimit"><i>
47              A message will be sent to this email for
48              authentication</i></p>
49
50              <input type="submit" value="Create Account">
51          </form>
```

The first line of the block contains the `action`, or what the form should do after being submitted. In this case, the form's data is to be submitted to `createhandler.php`, a file on the

server. The `method` tells the form how to transfer its information. There are two types of methods, `POST` and `GET`. `GET` appends its data onto the end of URL's, making it unsafe to use on a form like this, where private account data is being entered. `POST` transfers data inside of `HTTP` requests, making it the better option.

So, after the user clicks the "Create Account" button, his information is sent to `createhandler.php` via `POST`. This is where the bulk of the work is done. First, the file connects to the database using `mysqli_connect()`. The database's name, hostname, username, and password are all inserted into the function as arguments:

```
13  //Variables inserted into connection
14  $host='localhost';
15  $db_uname='root';
16  $db_pword='redrose10';
17  $dbname='myDB';
18
19  //Connection variable
20  $conn = mysqli_connect($host, $db_uname, $db_pword, $dbname);
```

Note that the connection variable mentioned in the beginning of the section, `$conn`, is assigned to this function. Whenever a query is made to a database, this variable is used.

Then, the file retrieves the data from `POST`:

```
22  /*Casts each input received by POST method
23  into a string and then assigns it a variable */
24  $username= (strval($_POST["username"]));
25  $password= (strval($_POST["password"]));
26  $confirm= (strval($_POST["confirm"]));
27  $email= (strval($_POST["email"]));
```

Next, the server performs multiple checks on the information, making sure it meets requirements. For example, the password must be at least four characters long, and the username or email cannot be the same as anyone else's. If any of the requirements are not met, the user is sent back to `index.php` with an error message. Afterwards, a random code is generated for the user, and an email is sent to him for authentication. This process is discussed more in depth in the next section.

Even though the data has passed all checks, it cannot be stored on the server in plain text. In the event of the server being hacked, it is a good practice to hash data. Hashing functions take data and convert it into an algorithmically sorted string of characters. For example, a hash of the word `password` would be:

`$2a$12$hJZK/R..d9gqqmskvsHDUeYAZiuauqz4jW.I2.CH8Ux1nRbmYGHZO`

If the database was hacked, attackers wouldn't be able to comprehend key information.

Hashing with PHP is a quick process. All that is needed is `password_hash()`. The function's first argument is the data to be hashed. Then, the function takes what hashing algorithm to use as its second argument, which in this case is `bcrypt`, expressed by `PASSWORD_DEFAULT`. The third argument is optional. By setting the `cost` to `13`, it takes longer for data to be hashed, making it more time-consuming and difficult to use a brute-force attack method against the database. Notice that the only things being hashed are the user's password, email, and special code: the important information. Everything else can be left untouched and stored as plain text.

```
138  /*Hashing user password, email, and string using password_hash
139    function. Salt is automatically generated using this function
140    and the algorithm used is the strongest one PHP has (bcrypt).
141    The cost is set to 13.*/
142  $option=array('cost' => 13);
143  $hashpass= password_hash($password, PASSWORD_DEFAULT, $option);
144  $hashmail= password_hash($email, PASSWORD_DEFAULT, $option);
145  $hashcode= password_hash($code, PASSWORD_DEFAULT, $option);
```

Now, a query can finally be executed to store the information in the database. Using the process discussed earlier, a statement is initialized, prepared, and variables are bound to it. This time, multiple pieces of data are being inserted into multiple fields.

```
164  //Variable that holds string inactive (bind_param only accepts vars)
165  $inactive="Inactive";
166
167  /*Initializes object, prepares statement, binds parameters into
168    prepared statement, executes statement and closes it */
169  $stmt= mysqli_stmt_init($conn);
170
171  mysqli_stmt_prepare($stmt, "INSERT INTO Account (Username, Password,
172  Email, Status, String) VALUES (?, ?, ?, ?, ?)");
173
174  mysqli_stmt_bind_param($stmt, "sssss", $username, $hashpass,
175  $hashmail, $inactive, $hashcode);
176
177  mysqli_stmt_execute($stmt);
178  mysqli_stmt_close($stmt);
```

The “sssss” in `mysqli_stmt_bind_param()` tells the function that all five parameters being bounded are of a `string` data type. Then, the statement is executed and closed.

(Note: In line 165, the string “`Inactive`” is assigned to a variable because `mysqli_stmt_bind_param()` only accepts variables as arguments. “`Inactive`” cannot be passed as an argument to the function, only its variable, `$inactive`.)

Afterwards, the connection to the database is closed and the user is taken to `email.php`, which informs him that an email has been sent for authentication, a process discussed in the next section.

Record in phpMyAdmin after form is submitted:

ID	Username	Password
1	Karan	$2y$13$AuO.3JF51gJTayefE6scSOLu8S1lBUMpz.N.bTmLBrG...

Email
$2y$13$4qCK7xaHQaJ.XHoalgQr9uT87lSTUx24De4X35XYhdX...

String
$2y$13$pGulOX9nuutq.gPFjVaVteRiFnF9UYjL7rq1rAck//m...

Status	Time_made
Inactive	2017-12-25 21:27:07

Sending Emails and Authentication-

While the user's information is being stored, `createhandler.php` also sends a message to the email address entered to confirm that the address belongs to him. If you remember from earlier, when an account is first created, its status is set as "Inactive". If this status is not changed to "Active" within ten minutes, the account is automatically deleted from the database. The sent email contains a link the user must click on in order to activate his account. First, though, let's look at how an email is sent in PHP.

In `createhandler.php`, there is a block of code which runs the `mail()` function:

```
121  //URL
122  $url= "http://localhost:8080/linkhandler.php?code=$code";
123
124   /*Sends an email to the address entered. If there is an error,
125     it sets a session error*/
126   if (!(mail("$email","Account Creation",
127              "Click the link to create your account! $url",
128
129              "From: Karan Arora"))){
130        $_SESSION["error"]= "There was an error sending the email";
131        header("Location: index.php");
132        exit();
133        }
```

The first argument of `mail()` is the target address, and the next is the subject of the email. The third argument contains the contents of the message, and the fourth and final one is who the email is from. In the image above, a URL directing to `linkhandler.php` is sent to the user with a code that activates his account.

The `if` statement on line `126` tells the server to return back to `index.php` with an error message if, for some reason, the email cannot be sent.

Now, let's focus on how an account is actually activated when the link is clicked. In `createhandler.php`, a special code, `$code`, is generated using the `rand()` function. This function picks a random number between a certain range. In this case, a value between `0` and `10000000000000000` (a very large number) is chosen:

```php
//Random number is generated
$code= rand(0, 10000000000000000);
```

Next, in order to make sure this code isn't the same as any other user's code, a `while` loop is used to cycle through all of the previous keys in the database. An array containing these codes, `$list`, was made previously in the file for this loop's use. Again, a full version of `createhandler.php` can be found in the Appendix.

```php
while ($num <= $len){
    if (password_verify($code, $list[$num])) {
        $code= rand(0, 10000000000000000);
        $num=0;
        }
    else {
        $num++;
        }
    }
```

In this loop, each index of `$list` is iterated over and compared to the code generated for the user through `password_verify()`, which takes the hash of the first argument and sees if it matches up with the hash of the second. If both of the hashes are the same, a new code is generated and the loop restarts. If not, the loop moves to the next index in `$list`, eventually going through the entire array and breaking.

Finally, the email is sent to the user with the unhashed code being inserted in the URL, while a hashed version of it is inserted into the database.

Now, the user receives an email looking something like this:

Click the link to create your account! http://localhost:8080/linkhandler.php?code=6875332989729941

When the link is clicked, `linkhandler.php` first connects to the database using `mysqli_connect()`. Next, it grabs the code by using the `GET` method, which, unlike `POST`, can get information from URL's. In the URL, the phrase `?code=6875332989729941` is called a `Query String` (unrelated to an SQL query). Started by the question mark, it sets the parameter, `code`, to the value `6875332989729941`. In the image below, `$code` is assigned to the URL's `code`:

```php
/*Gets the username specified in the query from URL emailed
  to the user*/
$code=(strval($_GET['code']));
```

Next, using the SQL statement:

```php
$sql= "SELECT String, Status, Username FROM Account";
```

the function `checker()`, manually made in the file, validates the code and sees if it is the same as anything in the database.

```php
if (!(checker($sql, $conn, $code))){
   //Unsets and destroys session
   session_unset();
   session_destroy();

   //Starts session
   session_start();
   //Sets session error
   $_SESSION["error"]= "Account does not exist";
   header("Location: index.php");
   exit();
   }
```

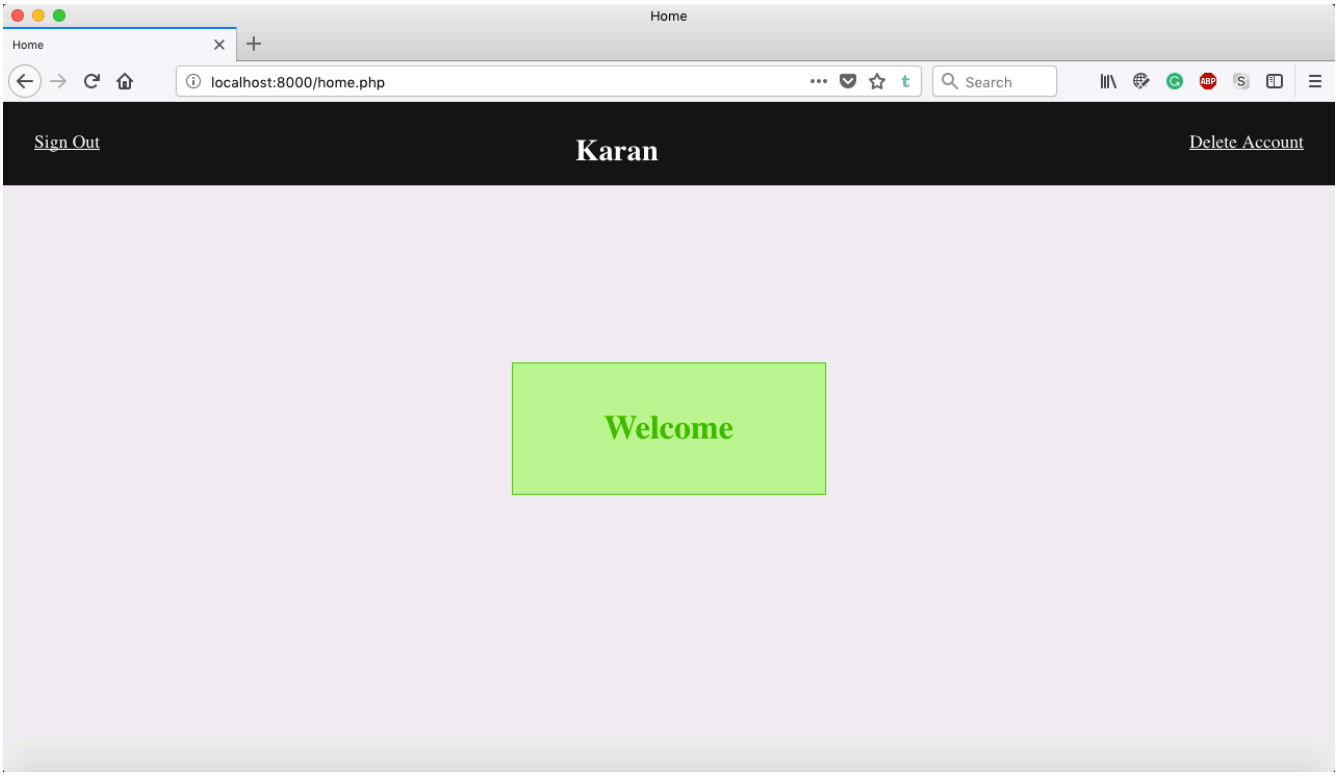
```php
/*Function that returns true if the code in the link corresponds to
  an account, and that account is inactive*/
function checker($newsql, $newconn, $newcode) {

   $result= mysqli_query($newconn, $newsql);
   if (mysqli_num_rows($result) > 0){
       while ($row=mysqli_fetch_assoc($result)){
           //Compares string in table to first parameter
            if ((password_verify($newcode, $row["String"])
                and ($row["Status"]=="Inactive"))){
                   //Set session username
                   $_SESSION["username"]=$row["Username"];
                   return true;
           }
       }
   }
}
```

In `checker()`, the SQL query is sent to server, which returns every record's `Status`, `String`, and `Username` field. If a selected record's hashed code and the code retrieved by `GET` match up using `password_verify()`, and the account is "Inactive", the function returns `true` and exits. If the code is invalid, the function does not return anything, and the user is directed to `index.php` and given an error message.

After the username associated with the code is found, the user's account is updated to "Active" with the following SQL query:

```
75  $stmt= mysqli_stmt_init($conn);
76
77  mysqli_stmt_prepare($stmt, "UPDATE Account SET Status='Active'
78  WHERE Username=?");
79
80  mysqli_stmt_bind_param($stmt, "s", $username);
81
82  mysqli_stmt_execute($stmt);
83  mysqli_stmt_close($stmt);
```

Finally, the user is taken to `home.php`:



Removing Inactive Accounts-

If a user enters an email that is not his, or for some reason does not want to activate his account, his "Inactive" record is left sitting on the database, never to be used. Leaving "Inactive" records on the database would clutter it quickly and take up space for no reason. But how would you remove them? Manually checking phpMyAdmin for these accounts would be a tedious and undesired job. Running a script that checks the database is a better idea but even with this, who would run the script? Instead, we can use a `cron`. A cron is a software program that performs a task periodically. By setting one up, we can execute a script that looks for inactive accounts whenever we want it to.

First, lets look at the file that checks the database: `remove.php`. Like the other PHP files, when the code is run, it first connects to the database using `mysqli_connect()`. Then, it creates a variable that contains the current date, called `$nowdate`. This variable is assigned to `time()`, a function that returns the number of seconds since Unix was created.

```
23  //Current time (when cron checks)
24  $nowdate = time();
```

Now, using the SQL query:

```
32  $sql="SELECT Status, Username, Time_made FROM Account";
```

the following block of code gets the `Status`, `Username`, and `Time_made` fields from all of the records in the database:

```
33  $result=mysqli_query($conn, $sql);
34
35  if (mysqli_num_rows($result) > 0){
36     while ($row=mysqli_fetch_assoc($result)){
37          if ($row["Status"]== "Inactive"){
38              $username=$row["Username"];
39              $date=$row["Time_made"];
40
41              //Casts to time variable
42              $pastdate=strtotime($date);
43              $diff=$nowdate-$pastdate;
44
45              if ($diff > 600){
46                  array_push($list, $username);
47                  }
48              }
49          }
50      }
```

If an account is “Inactive”, the file converts the timestamp held in `Time_made` to a number like the one returned by `time()` by using the `strtotime()` function and assigns this number to `$pastdate`. This is done so `$nowdate` and `$pastdate` are of the same type and can interact with each other. If you remember from earlier, the field `Time_made`, on being updated or created, defaults to a value of the current timestamp. So, when `createhandler.php` is run and an account is created, `Time_made` already holds the date of when it was made.

Next, the file subtracts `$pastdate` from `$nowdate` and assigns the resulting number to `$diff`. This gives us the amount time, in seconds, that has passed between when the account was created and the present. Now, if this value is above 600 seconds, or ten minutes, the username of the record is added to `$list`, an empty array created earlier in the file. After the `while` loop terminates, there will be a complete list of all accounts that need to be deleted.

A `for` loop then iterates through each username in `$list` and executes an SQL query deleting records that have a matching username.

```
55  //Iterates over each username in $list and deletes it from the table
56  for($x=0;$x<$len;$x++)
57    {
58    $username= $list[$x];
59
60   $stmt= mysqli_stmt_init($conn);
61
62   mysqli_stmt_prepare($stmt, "DELETE FROM Account WHERE
63   Username=?");
64
65   mysqli_stmt_bind_param($stmt, "s", $username);
66
67   mysqli_stmt_execute($stmt);
68   mysqli_stmt_close($stmt);
69    }
```

Finally, the file closes its connection to the database using `mysqli_close()`.

Now, let’s focus on cron.txt, which tells the server to run `remove.php` every minute. The file is short, only having two lines of text:

```
2  MAILTO=karanarora2001@gmail.com
3  * * * * * /usr/bin/php7.0 -q -f /var/www/html/remove.php
```

The first line of the file tells the server to email me if something has gone wrong. For example, if `remove.php` cannot be accessed, I will immediately receive an alert in my inbox:

The second line contains the commands the server is told to follow. The first five characters, `* * * * *` , indicate that the command is to be run every minute. The first character represents the minute, the second represents the hour, the third represents the day of the month, the fourth represents the month, and finally the fifth represents the day of the week. By putting `*` into each of these positions, it is interpreted that the command should be run every minute of every hour of every day of every month. To set cron schedules, use https://crontab.guru/.

Next, `/usr/bin/php7.0 -q –f` tells the cron in what environment to run the code. In order for PHP 7 to be executed, it must be run using PHP 7's binaries. `–q` and `–f` are options for the command. Finally, `/var/www/html/remove.php` is the pathway to the file.

(Note: When making the `.txt` file for the cron, make sure you use a simple text editor: one that does not have too many features. For example, I used TextWrangler, an app on Macs. You could even create it in Terminal using the `nano` editor. The reason for this warning is that some text editors include extra information in their files that is not shown to the user but could end up messing up cron when it comes time to execute it.)

In order to set up the cron on the VM, first go to the file where all of the server's webpages are stored:

```
cd /var/www/html
```

Then, enter:
`crontab <CronFileName>`

In my case, I would have typed:
`crontab cron.txt`

After this, you are done! To make sure the cron was added properly, you can type:
`crontab -l`

And with that, the manual on this website is wrapped up. Hopefully you have learned about how a webserver is set up, how databases are structured, how backends for a website work, and the techniques and concepts for how to store data! Thanks for reading!

Appendix

Collection of all of the HTML, CSS, and PHP scripts that run the website.

PHP-

1. `createhandler.php`- creates accounts
2. `signhandler.php`- signs users in
3. `linkhandler.php`- performs user authentication via code found in email
4. `delete.php`- deletes accounts
5. `remove.php`- removes accounts inactive for longer than ten minutes
6. `cron.txt`- goes along with `remove.php` and tells file to execute every minute
7. `signout.php`- signs users out of account

HTML-

8. `index.php`- main page for the site where users create accounts
9. `email.php`- where users are taken once account creation email is sent
10. `signin.php`- sign-in page
11. `home.php`- page for users once signed-in

CSS-

12. `main.css`- CSS file for all HTML documents

PHP-

1. `createhandler.php`- creates accounts

```php
<!-- Creates accounts -->

<?php
 //Starts session
 session_start();

 //If session username is set, it takes you to homepage
 if (isset($_SESSION["username"])){
    header("Location: home.php");
    exit();
    }

 //Variables inserted into connection
 $host='localhost';
 $db_uname='root';
 $db_pword='redrose10';
 $dbname='myDB';

 //Connection variable
 $conn = mysqli_connect($host, $db_uname, $db_pword, $dbname);

 /*Casts each input received by POST method
 into a string and then assigns it a variable */
 $username= (strval($_POST["username"]));
 $password= (strval($_POST["password"]));
 $confirm= (strval($_POST["confirm"]));
 $email= (strval($_POST["email"]));

 //Gets length of username, password, re-entered password, and email
 $passlen=strlen($password);
 $uselen=strlen($username);
 $conlen=strlen($confirm);
 $emaillen=strlen($email);

 /*Directs you back to index.php if any of the lengths are zero
 with an error message stating to fill in all fields */
 if (($uselen==0) or ($passlen==0) or ($conlen==0) or ($emaillen==0)){
    $_SESSION["error"]="Please fill in all fields";
    header("Location: index.php");
    exit();
    }

 /*Directs you back to index.php if password length is less
 than 4 characters and gives an error stating the password
```

```php
 must be longer */
 elseif ($passlen< 4){
    $_SESSION["error"]="Password must be at least 4 characters";
    header("Location: index.php");
    exit();
    }

 /*Directs you back to index.php if the password and the
 re-entered password do not match, stating an error that
 says to make sure they match */
 elseif ($password != $confirm){
    $_SESSION["error"]="The password you re-entered did not match";
    header("Location: index.php");
    exit();
    }

 /*Checks all of the elements in the Username and Email columns
 of Account */
 $sql= "SELECT Username, Email, String FROM Account";
 $result= mysqli_query($conn, $sql);

//Array where used codes are sent
$list= array();

 /*Checks if there are more than 0 rows. If this is true, it
 gets the username and email from each row and compares it
 to the ones being entered. If they're the same, it takes you
 back to index.php because there can't be two of the same
 username or email. Also, it goes through all of the codes
 in the database, appending them to $list.*/
 if (mysqli_num_rows($result) > 0){
    while ($row=mysqli_fetch_assoc($result)){
        if ($row["Username"]== $username){
            //Sets error stating username is already taken
            $_SESSION["error"]="Username is already in use";
            header("Location: index.php");
            exit();
            }

        elseif (password_verify($email, $row["Email"])){
            //Sets error stating email is already taken
            $_SESSION["error"]="Email is already in use";
            header("Location: index.php");
```

```
            exit();
            }

        $usedcode= $row["String"];
        array_push($list, $usedcode);

            }
        }

//Random number is generated
$code= rand(0, 10000000000000000);
//Length of $list
$len= count($list);
$num=0;

/*While $num is less than the length of $list, this will repeat.
  If $code is equal to any string in $list, it is given another
  random string, and num is reset to 0 so everything in $list is
  iterated through again. If $code does not equal the string given
  to it, $num is increased by one. This is done to ensure no two
  codes are the same. If two codes were the same, on the off chance
  that two accounts were inactive and had the same code, someone
  could be directed to another person's account. */
while ($num <= $len){
    if (password_verify($code, $list[$num])) {
        $code= rand(0, 10000000000000000);
        $num=0;
        }
    else {
        $num++;
        }
    }

//URL
$url= "http://localhost:8080/linkhandler.php?code=$code";

 /*Sends an email to the address entered. If there is an error,
   it sets a session error*/
 if (!(mail("$email","Account Creation",
            "Click the link to create your account! $url",

            "From: Karan Arora"))){
```

```
        $_SESSION["error"]= "There was an error sending the email";
        header("Location: index.php");
        exit();
        }

    //Sets session email
    $_SESSION["email"]=strval($email);

    /*Hashing user password, email, and string using password_hash
      function. Salt is automatically generated using this function
      and the algorithm used is the strongest one PHP has (bcrypt
      I believe). The cost is set to 13.*/
    $option=array('cost' => 13);
    $hashpass= password_hash($password, PASSWORD_DEFAULT, $option);
    $hashmail= password_hash($email, PASSWORD_DEFAULT, $option);
    $hashcode= password_hash($code, PASSWORD_DEFAULT, $option);

    /*ORIGINAL CODE

    //Sets date to be inserted
    $date = date('Y-m-d H:i:s');

    $sql= "INSERT INTO Account (
    Username,Password,Email,Status, String, Time_made)
    VALUES ('$username',
    '$hashpass',
    '$hashmail',
    'Inactive',
    '$hashcode',
    '$date')";

    mysqli_query($conn, $sql);    */

    //Statements below prevent SQL injection
    //Variable that holds string inactive (bind_param only accepts vars)
    $inactive="Inactive";

    /*Initializes object, prepares statement, binds parameters into
      prepared statement, executes statement and closes it */
    $stmt= mysqli_stmt_init($conn);

    mysqli_stmt_prepare($stmt, "INSERT INTO Account (Username, Password,
    Email, Status, String) VALUES (?, ?, ?, ?, ?)");
```

```
173
174     mysqli_stmt_bind_param($stmt, "sssss", $username, $hashpass,
175     $hashmail, $inactive, $hashcode);
176
177     mysqli_stmt_execute($stmt);
178     mysqli_stmt_close($stmt);
179
180     //Closes mysql connection to database
181     mysqli_close($conn);
182
183     //Directs you to email.php
184     header("Location: email.php");
185   ?>
```

2. `signhandler.php`- signs users in

```php
<!-- Signs in users -->

<?php
 //Starts session
 session_start();

 //If the session username is set it takes you to the homepage
 if (isset($_SESSION["username"])){
    header("Location: home.php");
    exit();
    }

 //Variables for connection to database
 $host='localhost';
 $db_uname='root';
 $db_pword='redrose10';
 $dbname= 'myDB';

 //Function that connects to database
 $conn = mysqli_connect($host, $db_uname, $db_pword, $dbname);

 //Casts input to strings and gets them with POST
 $username= (strval($_POST["username"]));
 $password= (strval($_POST["password"]));

 //Gets length of username and password
 $uselen=(strlen($username));
 $passlen=(strlen($password));

 /*If length of username or password is 0, it redirects you
 to signin.php and sets an error stating to fill all fields */
 if (($uselen==0) or ($passlen==0)){
    $_SESSION["error"]="Please fill in all fields";
    header("Location: signin.php");
    exit();
    }

 /*Selects the Username, Password, and Status columns from
 the Account table*/
 $sql="SELECT Username, Password, Status FROM Account";

 $result=mysqli_query($conn, $sql);

 /*Checks to see if there are more than 0 rows in the table.
```

```php
    If there are, it goes through each row and compares the
    username and password entered by the user to the usernames
    and passwords in the columns. If it finds a match, it
    directs the user to home.php. If not, or if there's 0
    rows, it directs the user back to the signin.php. The code
    also checks to make sure the account is active.*/
if (mysqli_num_rows($result)> 0) {
    //I could have used bind_result for this, but I didn't see a point

    //SQL injection isn't possible with the way the code is formatted?
    while ($row=mysqli_fetch_assoc($result)){
        if (($row["Username"]==$username) and
        //Unhashes string in table and compares it to first parameter
        (password_verify($password, $row["Password"])
        and ($row["Status"]=="Active"))){
            header("Location: home.php");

            //Sets session username
            $_SESSION["username"]= $username;

            //Closes mysql connection to database
            mysqli_close($conn);
            exit();
            }
        else{
            /*Sets session error stating there is an incorrect
            username or password */
            $_SESSION["error"]="Incorrect username or password";
            header("Location: signin.php");
            }
        }
    }
else{
    /*Sets session error stating there is an incorrect
      username or password */
    $_SESSION["error"]="Incorrect username or password";
    //Takes you to signin.php if the first statement is false
    header("Location: signin.php");
    }
?>
```

3. `linkhandler.php`- performs user authentication via code found in email

```php
<?php
 //Starts session
 session_start();

 //Sends you to your homepage if username is set
 if (isset($_SESSION["username"])){
        header("Location:home.php");
        exit();
        }

 /*Function that returns true if the code in the link corresponds to
   an account, and that account is inactive*/
 function checker($newsql, $newconn, $newcode) {

    $result= mysqli_query($newconn, $newsql);
    if (mysqli_num_rows($result) > 0){
        while ($row=mysqli_fetch_assoc($result)){
            //Compares string in table to first parameter
             if ((password_verify($newcode, $row["String"])
                 and ($row["Status"]=="Inactive"))){
                    //Set session username
                    $_SESSION["username"]=$row["Username"];
                    return true;
            }
        }
    }
 }

 //Variables inserted into connection
 $host='localhost';
 $db_uname='root';
 $db_pword='redrose10';
 $dbname='myDB';

 //Connection variable
 $conn = mysqli_connect($host, $db_uname, $db_pword, $dbname);

 /*Gets the username specified in the query from URL emailed
   to the user*/
 $code=(strval($_GET['code']));

 /*If the function is false it must mean someone tried typing
   the URL in to gain access to an account (the account wasn't
   real or is already activated). As a result you're taken to
```

```
    index.php, and the session is unset and destroyed.*/
 $sql= "SELECT String, Status, Username FROM Account";

 if (!(checker($sql, $conn, $code))){
    //Unsets and destroys session
    session_unset();
    session_destroy();

    //Starts session
    session_start();
    //Sets session error
    $_SESSION["error"]= "Account does not exist";
    header("Location: index.php");
    exit();
    }

 $username= $_SESSION["username"];

/* ORIGINAL CODE
 $sql= "UPDATE Account
        SET Status='Active'
        WHERE Username= '$username';";

 mysqli_query($conn, $sql);*/

 /*Updates account column to make user's account active and prevents
   SQL injection */

  /*Initializes object, prepares statement, binds parameters into
   prepared statement, executes statement and closes it */
 $stmt= mysqli_stmt_init($conn);

 mysqli_stmt_prepare($stmt, "UPDATE Account SET Status='Active'
 WHERE Username=?");

 mysqli_stmt_bind_param($stmt, "s", $username);

 mysqli_stmt_execute($stmt);
 mysqli_stmt_close($stmt);
 //Takes you to home page
 header("Location: home.php");
 //Closes SQL Connection
 mysqli_close($conn);
 ?>
```

4. `delete.php`- deletes accounts

```php
<!-- Deletes user's account -->

<?php
 //Starts session
 session_start();

 //Variables inserted into connection
 $host='localhost';
 $db_pword='redrose10';
 $db_uname='root';
 $dbname='myDB';

 //Connection variable
 $conn=mysqli_connect($host, $db_uname, $db_pword, $dbname);

 /*Gets username from session variable and casts it to
 a string */
 $username=(strval($_SESSION["username"]));

 /*ORIGINAL CODE
 //Deletes user's account
 $sql="DELETE FROM Account WHERE Username='$username'";
 mysqli_query($conn, $sql);
 */

 //Statements below prevent SQL injection

 /*Initializes object, prepares statement, binds parameters into
   prepared statement, executes statement and closes it */
 $stmt= mysqli_stmt_init($conn);

 mysqli_stmt_prepare($stmt, "DELETE FROM Account WHERE Username=?");
 mysqli_stmt_bind_param($stmt, "s", $username);

 mysqli_stmt_execute($stmt);
 mysqli_stmt_close($stmt);

 //Unsets and destroys session
 session_unset();
 session_destroy();

 //Directs you to index.php
 header("Location: index.php");
?>
```

5. `remove.php`- removes accounts inactive for longer than ten minutes

```php
<?php
 //Runs every minute due to cron.txt's *****

 /*If this script is accessed through a browser by a user, then he
   user will redirected to index.php*/
 if (!isset($GLOBALS['argv'])) {
    header("Location: index.php");
    exit();
    }

 //Variables inserted into connection
 $host='localhost';
 $db_pword='redrose10';
 $db_uname='root';
 $dbname='myDB';

 //Connection variable
 $conn=mysqli_connect($host, $db_uname, $db_pword, $dbname);

 //Empty array
 $list=array();

 //Current time (when cron checks)
 $nowdate = time();

 /*Get the Status, Username, and Time_made strings from each account.
   If the status is inactive, it compares the timestamp from Time_made
   to the $nowdate variable and subtracts the two. If the difference
   is more than 600 seconds (10 minutes), the username is pushed to
   the $list array.
 */
 $sql="SELECT Status, Username, Time_made FROM Account";
 $result=mysqli_query($conn, $sql);

 if (mysqli_num_rows($result) > 0){
    while ($row=mysqli_fetch_assoc($result)){
        if ($row["Status"]== "Inactive"){
            $username=$row["Username"];
            $date=$row["Time_made"];

            //Casts to time variable
            $pastdate=strtotime($date);
            $diff=$nowdate-$pastdate;

```

```
45              if ($diff > 600){
46                  array_push($list, $username);
47                  }
48              }
49          }
50      }
51
52  //Gets length of $list
53  $len=count($list);
54
55  //Iterates over each username in $list and deletes it from the table
56  for($x=0;$x<$len;$x++)
57    {
58    $username= $list[$x];
59   /* ORIGINAL CODE
60    $sql="DELETE FROM Account WHERE Username='$username'";
61    mysqli_query($conn, $sql);
62    */
63
64   //Statements below prevent SQL injection
65   /*Initializes object, prepares statement, binds parameters into
66     prepared statement, executes statement and closes it */
67   $stmt= mysqli_stmt_init($conn);
68
69   mysqli_stmt_prepare($stmt, "DELETE FROM Account WHERE
70   Username=?");
71
72   mysqli_stmt_bind_param($stmt, "s", $username);
73
74   mysqli_stmt_execute($stmt);
75   mysqli_stmt_close($stmt);
76    }
77
78   //Closes connection to database
79   mysqli_close($conn);
80  ?>
```

6. `cron.txt`- goes along with `remove.php` and tells file to execute every minute

```
1
2   MAILTO=karanarora2001@gmail.com
3   * * * * * /usr/bin/php7.0 -q -f /var/www/html/remove.php
```

7. `signout.php`- signs users out of account

```php
<!-- Signs users out -->

<?php
/*Starts session, unsets variables, and then destroys it
and redirects you to index.php*/
session_start();
session_unset();
session_destroy();

header("Location: index.php");
?>
```

HTML-

8. `index.php`- main page for the site where users create accounts

```php
<!DOCTYPE html>

<!-- Main page of website. Allows you to create an account -->

<?php
 //Starts session
 session_start();
 //If the session username is set, it takes you to home.php
 if (isset($_SESSION["username"])){
    header("Location: home.php");
    }
?>

<html>

    <head>
        <title>Sign Up</title>
        <link rel="stylesheet" type="text/css" href="main.css">
    </head>

    <body>
        <div class="head">
         <a href="signin.php">Sign In</a>
        </div>

        <!-- Div for form to create a new account -->
        <div class= "create" id="create-index">

        <!-- Directs form to a php file that sends email
            using POST method -->
        <form action="/createhandler.php" method= "POST">

            Username:<br>
            <input type="text" name="username" size="25"><br><br>

            Password:<br>
            <input type="password" name="password" size="25"><br>
            <p class="charlimit"><i>
            Password must be at least 4 characters</i></p>

            Confirm Password:
            <input type="password" name="confirm" size="25"><br><br>

            Email:
```

```
            <br><input type="text" name="email" size="25"><br>
            <p class="charlimit"><i>
            A message will be sent to this email for
            authentication</i></p>

            <input type="submit" value="Create Account">
        </form>
        </div>

        <div class="title" id="title-index">
        <h2>Create an Account</h2>
        </div>

        <!-- If a session error is set it displays it at the top of
        the screen -->
        <?php
         if (isset($_SESSION["error"])){
            $error=$_SESSION["error"];

            echo '<div';

            echo ' style="background-color: rgba(255, 86, 86, 0.5);';
            echo ' height:50px; width: 325px;';
            echo ' border: 1px solid #fc4444;';
            echo ' margin-left:auto; margin-right: auto;';
            echo ' margin-top: -115px;">';

            echo '<p';
            echo ' style="width: 290px; color: #cc2b22;';
            echo ' padding-left:13px; padding-top:1px;';
            echo ' margin-left: auto; margin-right: auto;">';

            echo "$error</p>";

            echo "</div>";

            /*Unsets and destroys session so the error message
            goes away on refresh */
            session_unset();
            session_destroy();
                  }
        ?>
    </body>
</html>
```

9. `email.php`- where users are taken once account creation email is sent

```php
<!DOCTYPE html>

<!-- Where you're directed after creating an account -->

<?php
    //Starts session
    session_start();

    //If the session username is set, it takes you to the homepage
    if (isset($_SESSION["username"])){
        header("Location:home.php");
        exit();
        }
    //If the session email isn't set, it takes you to index.php
    elseif (!(isset($_SESSION["email"]))){
        header("Location: index.php");
        exit();
        }
?>

<html>
<head>
    <title>Email</title>
    <link rel="stylesheet" type="text/css" href="main.css">
</head>
<body>
    <div class="head">
    <a href="index.php">Home</a>
    </div>

    <div class="welcome">
        <?php
              /*Prints a message in a green box stating the email
                was sent*/
              $email=(strval($_SESSION["email"]));
              echo '<p style= "width: 240px;';
              echo 'margin-left: auto; margin-right: auto;';
              echo 'margin-top: 20px; color: #21ba00;">';
              echo "An email has been sent to $email for
                    authentication. The account will be
                    removed if it is not confirmed within
                    10 minutes.</p>";

              /*Unsets and destroys session so on refresh user is taken
                to index.php*/
              session_unset();
              session_destroy();
              ?>
    </div>

</body>
</html>
```

10. `signin.php`- sign-in page

```php
<!DOCTYPE html>

<!-- Sign-in page -->

<?php
 //Starts session
 session_start();
 //If session username is set, it takes you to home.php
 if (isset($_SESSION["username"])){
    header("Location: home.php");
    }
?>

<html>
<head>
 <title>Sign in</title>
 <link rel="stylesheet" type="text/css" href="main.css">
</head>

<body>

<div class="title" id="title-sign">
 <h2>Sign-in with an Existing Account</h2>
</div>

<div class="head">
 <a href="index.php">Create an Account</a>
</div>

<!-- Where you enter information to sign in -->
<div class="create" id="create-sign">

<!-- Takes input to signhandler.php with POST method -->
 <form action="/signhandler.php" method="POST">
    Username:<br>
    <input type="text" name="username"><br><br>
    Password:<br>
    <input type="password" name="password">
    <input type="submit" value="Sign in">
 </form>
</div>

<!-- If there is a session error then it displays it at the
top of the screen -->
```

```php
<?php
if (isset($_SESSION["error"])){
    $error=$_SESSION["error"];

    echo '<div';

    echo ' style="background-color: rgba(255, 86, 86, 0.5);';
    echo ' height:50px; width: 325px;';
    echo ' border: 1px solid #fc4444;';
    echo ' margin-left:auto; margin-right: auto;';
    echo ' margin-top: -275px;">';

    echo '<p';
    echo ' style="width: 290px; color: #cc2b22;';
    echo ' padding-left:13px; padding-top:1px;';
    echo ' margin-left: auto; margin-right: auto;">';

    echo "$error</p>";

    echo "</div>";

    /*Unsets and destroys session so error message goes
    away on refresh */
    session_unset();
    session_destroy();
    }
?>

</body>
</html>
```

11. `home.php`- page for users once signed-in

```php
<!DOCTYPE html>

 <!-- Where you are directed if account creation or
 sign-in is a success -->

<?php
  //Starts session
  session_start();

  /*If the session username isn't set, you're
  taken to signin.php */
  if (!isset($_SESSION["username"])){
    header("Location: signin.php");
    }

  //Sets name variable
  $name= $_SESSION["username"];
  ?>

<html>

 <head>
  <title>Home</title>
  <link rel="stylesheet" type="text/css" href="main.css">
 </head>

 <body>
 <div class="head">
 <a href="signout.php">Sign Out</a>
 <a id="delete" href="delete.php">Delete Account</a>

 <!-- Displays your username in the header -->
 <?php
  echo("<h1>".$name."</h1>");
 ?>
 </div>
 <!-- Welcome message -->
 <div class= "welcome">
 <h1>Welcome</h1>
 </div>

 </body>

</html>
```

CSS-

12. `main.css`- CSS file for all HTML documents

```
/*Stylesheet for website */

/*Formats body */
body {
    margin: 0px;
    background-color: #f2eff1;
    }

/*Formats h2 headings */
h2 {
    font-family: Helvetica;
    }

/*Formats header at top */
.head {
    position: fixed;
    top: 0px;
    background-color: #141414;
    width: 100%;
    height: 80px;
    }

/*Formats link in the header */
.head a {
    padding-top: 30px;
    padding-left: 30px;
    color: white;
    float: left;
    font-size: 110%;
    height: 50px;
        }

/*Formats heading that displays the username in home.php */
.head h1 {
    color: white;
    width: 180px;
    margin-left: auto;
    margin-right: auto;
    padding-top: 15px;
    font-size: 180%;
    }

/*Formats the box where you enter information */
.create {
```

```
   margin-left: auto;
   margin-right: auto;
   width: 200px;
    }

/*Formats the title on top of the boxes */
.title {
    margin-left: auto;
    margin-right: auto;
    }

/*Formats messages beneath form inputs */
.charlimit {
    font-size: 75%;
    }

/*Formats green box in home.php and email.php */
.welcome {
    width: 300px;
    height: 125px;
    background-color: rgba(125, 252, 70, 0.5);
    margin: auto;
    margin-top: 250px;
    border: 1px solid #26d336;
    }

/*Formats text in green sign in home.php */
.welcome h1 {
    width: 125px;
    margin-left: auto;
    margin-right: auto;
    margin-top: 45px;
    color: #21ba00;
    }

/*Makes links in header turn gold when cursor is on top of them */
a:hover{
    color: gold;
    }

/*Formats the delete link in home.php */
#delete {
    float: right;
    padding-top: 30px;
```

```
    padding-right: 30px;
    }

/*Each heading and box needed an ID for formatting as well
because some things weren't just working */

/*Formats title in index.php */
#title-index{
    width:225px;
    margin-top: -385px;
    }

/*Formats title in sigin.php */
#title-sign{
    width:400px;
    padding-top:250px;
    }

/*Formats information-enter box in index.php */
#create-index{
    padding-top: 225px;
    }

/*Formats information-enter box in signin.php */
#create-sign {
    padding-top: 10px;
    }
```