Julia George

CSCI 499

Senior Project Implementation

January 19, 2021

Professor Henderson

Senior Documentation

Statement of Purpose: The purpose of this project is to show the skills I have learned when it comes to cybersecurity and apply them to a situation that can show I have a solid understanding of the information.

Research & Background:

Research:

- Website links used for installing and configuring Nagios core.
    - Nagios Core - Installing Nagios Core From Source
    - How To Install Nagios 4 and Monitor Your Servers on Ubuntu 18.04 | DigitalOcean
    - https://support.Nagios.com/kb/article/Nagios-core-installing-Nagios-core-from-source-96.html#Ubuntu
    - https://www.digitalocean.com/community/tutorials/how-to-install-Nagios-4-and-monitor-your-servers-on-ubuntu-18-04
    - https://support.Nagios.com/forum/viewtopic.php?f=7&t=26154

- https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3/html/console_administration_guide/configuring_Nagios_to_send_mail_notifications
- https://exchange.Nagios.org/directory/Plugins/System-Metrics/Users/Show-Users/details
- https://exchange.Nagios.org/components/com_mtree/attachment.php?link_id=1530&cf_id=24

- Website links used to configure internal files to Nagios so that if there are issues it will send use emails and alerts of any events.
  - Configure Nagios To Use Sendmail – Brandon Wamboldt
  - Nagios alerts via email
  - 9.2. Configuring Nagios Server to Send Mail Notifications Red Hat Gluster Storage 3 | Red Hat Customer Portal
  - https://brandonwamboldt.ca/configure-Nagios-to-use-Sendmail-1188/
  - https://community.spiceworks.com/topic/117801-Nagios-alerts-via-email
- Website link with information on how to test Nagios email alerts.
  - Test Email Alert - View topic • Nagios Support Forum
- The book *Effortless E-Commerce with PHP and MySQL* second edition by Larry Ullman gave information about how to secure an ecommerce website with common techniques for securing website. This includes for the database, server, client side, and common vulnerabilities. This was used in formulating my security plan for the website.
- For securing Linux Server other things were used in addition to the ecommerce book:
  - https://opensource.com/article/19/10/linux-server-security

    - https://www.acunetix.com/websitesecurity/webserver-security/
- For securing the MySQL database the additional material used:
    - https://www.acunetix.com/websitesecurity/webserver-security/
    - https://www.tecmint.com/mysql-mariadb-security-best-practices-for-linux/
    - https://dev.mysql.com/doc/mysql-security-excerpt/8.0/en/security.html
- Website used to set the setting for password expiration
    - https://dev.mysql.com/doc/refman/5.7/en/
- For creating code, the books *PHP, MySQL & JavaScript All-In-One* by Richard Blum, *Web Coding & Development All-in-One for Dummies* by Paul McFederies, and *Learning PHP, MySQL & JavaScript: With jQuery, CSS, & HTML5* by Robin Nixon were used to help with creating some of the code for securing the website.
- Website used for installing and configuring MySQL
    - https://devanswers.co/install-apache-mysql-php-lamp-stack-ubuntu-20-04/
- Website used for PHPMyAdmin installation and configurations on Ubuntu server.
    - https://www.linuxbabe.com/ubuntu/install-phpmyadmin-apache-lamp-ubuntu-20-04
- Website used for certificate creation and setup on apache server
    - https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-18-04
- Websites used for certificate creation and setup on the MySQL instance.
    - https://websiteforstudents.com/how-to-setup-self-signed-ssl-tls-on-mysql/
    - https://www.digitalocean.com/community/tutorials/how-to-configure-ssl-tls-for-mysql-on-ubuntu-18-04

- Websites used to configure and install Apache, PHP, and MySQL website
    - https://www.linode.com/docs/guides/hosting-a-website-ubuntu-18-04/
    - https://websiteforstudents.com/setup-apahce2-with-php-support-on-ubuntu-servers/
    - https://www.cloudbooklet.com/how-to-install-lamp-apache-mysql-php-in-ubuntu-20-04/
    - https://serverfault.com/questions/399487/cant-connect-to-mysql-using-self-signed-ssl-certificate
    - https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04
- Websites used to configure send mail files to work with Gmail.
    - https://tecadmin.net/Sendmail-to-relay-emails-through-gmail-stmp/
    - https://linuxconfig.org/configuring-gmail-as-Sendmail-email-relay
    - https://linuxconfig.org/configuring-gmail-as-Sendmail-email-relayhttps://linuxconfig.org/configuring-gmail-as-Sendmail-email-relay
- Installing KeePass
    - https://linuxhint.com/install_keepass_ubuntu/
- Website used to set up the bash file to send emails if there is login failure.
    - https://unix.stackexchange.com/questions/339417/running-a-shell-script-on-n-failed-login-attempts
    - https://askubuntu.com/questions/727156/email-on-failed-login-attempt
- Website used to change server name to fully qualified name.
    - https://gridscale.io/en/community/tutorials/hostname-fqdn-ubuntu

- Other parts of the code to develop the pretty front end were taken from:
    - https://www.w3schools.com/howto/howto_css_checkout_form.asp
    - https://github.com/priyesh18/book-store
    - https://www.allphptricks.com/simple-shopping-cart-using-php-and-mysql/

Background:

- Parts of this page was also created based off of Applied Networking class where I learned HTML, PHP, and CSS. This class also gave me the knowledge of setting up and connecting a database, setting up and connecting a server, and using localhost. Scripting Language class taught me how to use HTML, CSS, and JavaScript.
- Database management class taught me a more in depth look into a database and how to do more with it.
- System Analysis & Software Design I had to create a website with PHP, CSS, and HTML while also making sure it is secured. This class taught me how to do input validation and using more advanced techniques for accessing data in a database.

Project Language(s), Software, Hardware: Languages used in this project are PHP, JavaScript, HTML, jQuery, SQL, Bash Scripting Language, and CSS. The software used is Nagios core, Ubuntu Linux OS, a firewall, MySQL database, Visual Studio Code, Apache HTTP Server, KeePass, MyPHPAdmin, and PHP runtime. Hardware used is Ubuntu Virtual Machine on Oracle VM virtualbox with 1 CPU, 8GB memory, and 15GB disk.

Project Requirements: This project is for demonstrating how to secure a website by using many different security techniques.

Web Hosting Hardening:

1. The only users who need access are given physical and network access. A monitoring tool named Nagios is used to make sure that this does not change. If more than one user is detected by Nagios an email is sent to the Nagios admin making the admin aware of the breach. An Access Control policy is in place that determines the users and their roles in the organization giving the programs the user is accessing, the description of access, and when the password needs to be changed. There is a segregation of duties section, Nagios users' section, MySQL users' section, and Gmail users' sections. This determines all the users in the organization and what they are allowed to do.
2. A uwf firewall is installed and running to monitor the host traffic coming in and out of a server. This helps to prevent outside computers from accessing computers inside my network. The firewall is also responsible for making sure only necessary ports are open for network connections.
3. Apache server and MySQL are up to date on the system which makes sure known vulnerabilities are patched, decreasing chances for an attacker to exploit it.
4. Users accessing the server have unique usernames and passwords. Making it harder for malicious users to guess what the username and passwords are to make brute force attacks, dictionary attacks, and birthday attacks less likely. This is accomplished with KeePass keeping a record of the users, their passwords, and when the password expires. There is also an Access Control Policy implemented that tells what is needed for a password including length, complexity, and expiration.
5. Bash file is triggered when a person has a failed login attempt accessing the server. This helps to monitor who is accessing the server and can be an indicator if someone might be trying to get into the server who should not be.

Database Security Needs:

1. Users with access to the database are given unique usernames and passwords. Making it harder for a malicious user to guess the passwords and also decreases likelihood of brute force attacks, dictionary attacks, and birthday attacks. This is accomplished by setting complexity, password expiration, and length.
2. Database password policy is set up on users to make sure that after a certain period of time they have to change the password. If a malicious user gets their hands on the passwords it will decrease the likelihood of those passwords being usable. The Access Control Policy has a section that has the MySQL users with their allowed permissions, which user is forced to use SSL, and when their passwords need to be changed.
3. Root password on MySQL was changed, changing the password from default will decrease likelihood of a hacker getting access to MySQL root account.
4. MySQL users can only connect to MySQL from the localhost which restricts users from accessing the database externally.
5. The MySQL user that is used by php to connect to the database is granted limited permissions. The permissions granted are SELECT, INSERT, UPDATE and DELETE privileges. This prevents the user from being able to perform any administrative task on the MySQL databases, this uses the security of least privilege for hardening.

PHP and Web Security:

1. PHP can only open files from the directory where the website code is stored. This prevents users from accessing outside directories so users can not get access to those files.

2. All sensitive information like credit card numbers and passwords are hashed using md5 hashing and then stored in a database. To prevent users from easily accessing the information that is stored from the website.
3. Strict is used on the PHP pages to make sure that the files are run like they are written, to prevent hackers making problematic data.
4. Generic error statements are used to prevent malicious users from being able to see that MySQL is being used.
5. A directory was setup for the sole use of storing websites pages. E The only page a user has direct access to without signing into website is the log in page. In order to access the remaining pages a customer must be a user and have the ability to sign in to the website. This prevents unauthorized users from accessing the pages.

Common Vulnerabilities

1. User input is validated using HTML, CSS, JavaScript, jQuery, and/or PHP working together to perform validaton to prevent common attacks.
2. Error's will display saying that the information does not match email and/or password given. If an attacker attempts to login using someone else's email address. That lessens the likelihood of an attack from guessing information.
3. To help prevent DoS attacks only used ports are open using a ufw firewall and also Nagios is using a plugin named ICMP to monitor network traffic. If the traffic becomes above certain threshold, set by settings in Nagios localhost file. Then an email will be sent to the Nagios admin.
4. Through the use of mysqli_prepare( ), mysqli_stmt_bind_param( ) which sets a specific number of variables that can be passed to SQL query and final step is to execute the

query through the use of mysqli_stmt_execute;(). After all those steps of the input it run the query against the database and not allowing any SQL injection attacks.

5. Cross-Site request forgery is prevented by using authentication certs to encrypt all connections to the server.

Project Implementation Description & Explanation:

For Access to code, documentation, and disaster recovery plan access: https://github.com/juliafaye21/SeniorProject.git

This project's main purpose is to show my ability to secure a website which includes securing against known vulnerabilities and general hardening practices to take. Setting up the project the first thing to consider is what platform is the best to use. The most well-known OSs are Windows, Apple, and Linux, since more information can be gathered on these, they are the best options to consider. I chose to use Ubuntu Linux since I am most familiar with it, it is open-source, and it is considered more secure. Plus, it has many built-in security features as well as security extensions that can be installed. It is running on a virtual machine to make sure that if during the process of installing software it does not cause any possible viruses to be on my personal machine. Additionally, I can take a snapshot and easily backup from that since I do have limited resources.

The next things to consider was the webhosting server to implement. Since I am using Ubuntu Linux that narrows down the servers compatible with it to four main types: Apache, IIS, Nginx, and LiteSpeed. IIS can be run on Linux, but it is a Windows product, this could end up causing instability. It is not recommended to use IIS with a Linux

System. LiteSpeed is a good server the many big businesses use but I am unfamiliar with it, plus its free version does not give access to all of its features. Nginx and Apache are both open-source and considered to be great when it comes to security. They also are both good with Linux compatibility. I ended up choosing to use Apache because I am more familiar with that. I did not want to have to end up wasting too much time learning about a new web server since there could be/was problems along the way. I also decided to choose MySQL, and PHP because of the familiarity I have with both these platforms.

The book *Effortless E-Commerce with PHP and MySQL* second edition by Larry Ullman gave a couple point on securing a web server. Limiting the number of people with physical and network access, running a firewall, running an antivirus program, keeping all software like Apache, DNS, email systems up to date, make passwords secure and change them regularly, tracking of who accesses the server, and sending email when someone logs into the server. To accomplish these tasks, I wanted to make it as simple as possible since most individuals are not tech savvy. After some research I decided to use Nagios, it is used in many big companies and they have a free version that has a basic monitoring service. Nagios also has extensions that can be used to accomplish the task I needed to. With this software I was able to limit the number of users with access to the server to one (See image 1). Additionally, Nagios allows for different monitoring that is useful when it comes to availability of the server. It has a feature allowing a ping to the server (See image 2), determine the amount of space available on the drives (See image 3), HTTPS availability (See image 4), current load for CPU (See image 5), and seeing if the site is getting DoS (See image 6).

Service State Information

Current Status:	OK (for 37d 2h 13m 39s)
Status Information:	USERS OK - 1 users currently logged in
Performance Data:	users=1;20;50;0
Current Attempt:	1/4 (HARD state)
Last Check Time:	03-20-2021 16:04:32
Check Type:	ACTIVE
Check Latency / Duration:	0.001 / 0.002 seconds
Next Scheduled Check:	03-20-2021 16:09:32
Last State Change:	02-11-2021 12:55:23
Last Notification:	N/A (notification 0)
Is This Service Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	03-20-2021 16:08:54 (0d 0h 0m 8s ago)

Active Checks:	ENABLED
Passive Checks:	ENABLED
Obsessing:	ENABLED
Notifications:	ENABLED
Event Handler:	ENABLED
Flap Detection:	ENABLED

Image 1: Results for number of users on server.

Service State Information

Current Status:	OK (for 37d 3h 36m 15s)
Status Information:	PING OK - Packet loss = 0%, RTA = 0.09 ms
Performance Data:	rta=0.087000ms;100.000000;500.000000;0.000000 pl=0%;20;60;0
Current Attempt:	1/4 (HARD state)
Last Check Time:	03-20-2021 21:54:03
Check Type:	ACTIVE
Check Latency / Duration:	0.000 / 4.106 seconds
Next Scheduled Check:	03-20-2021 21:59:03
Last State Change:	02-11-2021 17:19:29
Last Notification:	N/A (notification 0)
Is This Service Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	03-20-2021 21:55:38 (0d 0h 0m 6s ago)

Active Checks:	ENABLED
Passive Checks:	ENABLED
Obsessing:	ENABLED
Notifications:	ENABLED
Event Handler:	ENABLED
Flap Detection:	ENABLED

Image 2: Pinging of the server results.

Service State Information

Current Status:	OK (for 37d 3h 36m 8s)
Status Information:	DISK OK - free space: / 6395 MB (33.67% inode=76%):
Performance Data:	/=12593MB;16022;18025;0;20028
Current Attempt:	1/4 (HARD state)
Last Check Time:	03-20-2021 21:54:37
Check Type:	ACTIVE
Check Latency / Duration:	0.000 / 0.008 seconds
Next Scheduled Check:	03-20-2021 21:59:37
Last State Change:	02-11-2021 17:20:06
Last Notification:	N/A (notification 0)
Is This Service Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	03-20-2021 21:56:08 (0d 0h 0m 6s ago)

Active Checks:	ENABLED
Passive Checks:	ENABLED
Obsessing:	ENABLED
Notifications:	ENABLED
Event Handler:	ENABLED
Flap Detection:	ENABLED

Image 3: Results of space available in drive

127.0.

Service State Information

Current Status:	OK (for 29d 22h 3m 29s)
Status Information:	HTTP OK: HTTP/1.1 302 Found - 562 bytes in 0.021 second response time
Performance Data:	time=0.021195s;;;0.000000 size=562B;;;0
Current Attempt:	1/4 (HARD state)
Last Check Time:	03-20-2021 21:47:57
Check Type:	ACTIVE
Check Latency / Duration:	0.001 / 0.028 seconds
Next Scheduled Check:	03-20-2021 21:52:57
Last State Change:	02-18-2021 22:45:14
Last Notification:	N/A (notification 0)
Is This Service Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	03-20-2021 21:48:38 (0d 0h 0m 5s ago)

Active Checks:	ENABLED
Passive Checks:	ENABLED
Obsessing:	ENABLED
Notifications:	ENABLED
Event Handler:	ENABLED
Flap Detection:	ENABLED

Image 4: Results of HTTPS availability

Service State Information

Current Status:	OK (for 0d 6h 24m 8s)
Status Information:	OK - load average: 0.57, 0.28, 0.18
Performance Data:	load1=0.570;5.000;10.000;0; load5=0.280;4.000;6.000;0; load15=0.180;3.000;4.000;0;
Current Attempt:	1/4 (HARD state)
Last Check Time:	03-20-2021 21:53:51
Check Type:	ACTIVE
Check Latency / Duration:	0.000 / 0.006 seconds
Next Scheduled Check:	03-20-2021 21:58:51
Last State Change:	03-20-2021 15:32:51
Last Notification:	N/A (notification 0)
Is This Service Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	03-20-2021 21:56:58 (0d 0h 0m 1s ago)

Active Checks:	ENABLED
Passive Checks:	ENABLED
Obsessing:	ENABLED
Notifications:	ENABLED
Event Handler:	ENABLED
Flap Detection:	ENABLED

Image 5: Result for current CPU load

127.0.0.1

Service State Information

Current Status:	OK (for 30d 3h 48m 48s)
Status Information:	OK - 127.0.0.1: rta 0.020ms, lost 0%
Performance Data:	rta=0.020ms;200.000;500.000;0; pl=0%;40;80;; rtmax=0.046ms;;;; rtmin=0.013ms;;;;
Current Attempt:	1/4 (HARD state)
Last Check Time:	03-20-2021 21:53:30
Check Type:	ACTIVE
Check Latency / Duration:	0.001 / 0.003 seconds
Next Scheduled Check:	03-20-2021 21:58:30
Last State Change:	02-18-2021 17:06:20
Last Notification:	N/A (notification 0)
Is This Service Flapping?	NO (0.00% state change)
In Scheduled Downtime?	NO
Last Update:	03-20-2021 21:54:58 (0d 0h 0m 10s ago)

Active Checks:	ENABLED
Passive Checks:	ENABLED
Obsessing:	ENABLED
Notifications:	ENABLED
Event Handler:	ENABLED
Flap Detection:	ENABLED

Image 6: Results to see if DoS attack occurring.

When it comes to seeing who accesses the server apache already has a built-in log that keeps track of that. In Ubuntu Linux using the root user I was able to view the log /var/log/auth.log.1. It has entries of the users who have accessed the server (See image 7). For bad logins, a Pam file is used that test the authentication of the user (See image 8) and it points to

a bash script that creates an email and sends the email to a specific email address when the user has authentication failure when logging into the server (See image 9). I set up an email account with Gmail that is used for the purpose of receiving emails for bad login attempts and Nagios notifications. The email consists of a little detail such as date and time plus the name of the user attempting to gain access (See image 10).

For the password management on the server there are two things I did to accomplish that goal. One was making an access control policy that had a detailed overview for OS, Nagios, and MySQL users. In this document it has a policy in regard to passwords including complexity, expiration, and length of the password (See image 11). This policy also has a list of the users with the roles they are assigned and when their password was changed last (See image 12). It also gives a separation of duties list; this can be helpful later on when I am not the only user (See image 13). In addition to this access control policy, I am using KeePass to store the passwords and keeping track of password expiration (See image 14).

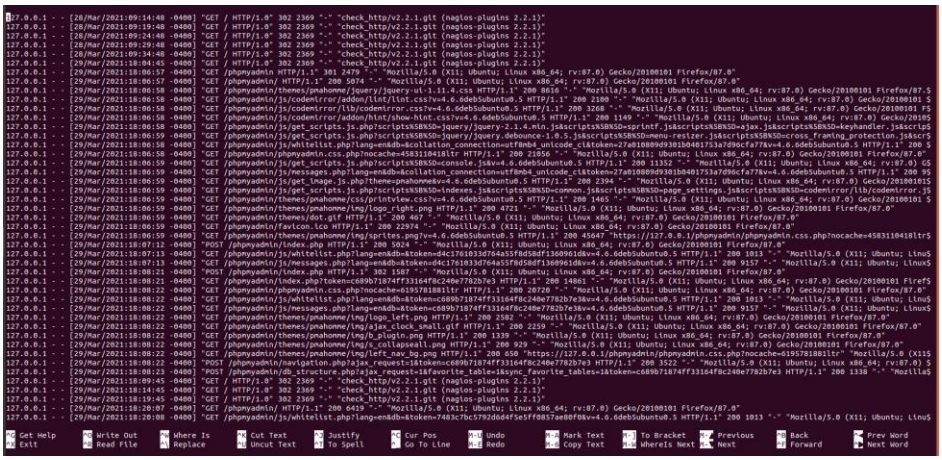


Image 7: access.log file contents.

```
# here are the per-package modules (the "Primary" block)
auth    [success=2 default=ignore]      pam_unix.so nullok_secure

# here is the script to email bad logins

auth [default=ignore] pam_exec.so seteuid /usr/bin/report_badlogin
```

Image 8: Pam file used to authenticate the user.

```
#! /bin/sh
# report_badlogin

if grep -F -x -v -f /var/log/auth.log.old /var/log/auth.log | grep -n 'authentication failure' | mail -s "Bad login attempt no
tification" "ekbaker.2553@gmail.com"; then
  cp -f /var/log/auth.log /var/log/auth.log.old
  chown root:root /var/log/auth.log.old
fi

exit 0
~
~
~
~
```

Image 9: Bash file to create email and send an email to me when user authentication fails.

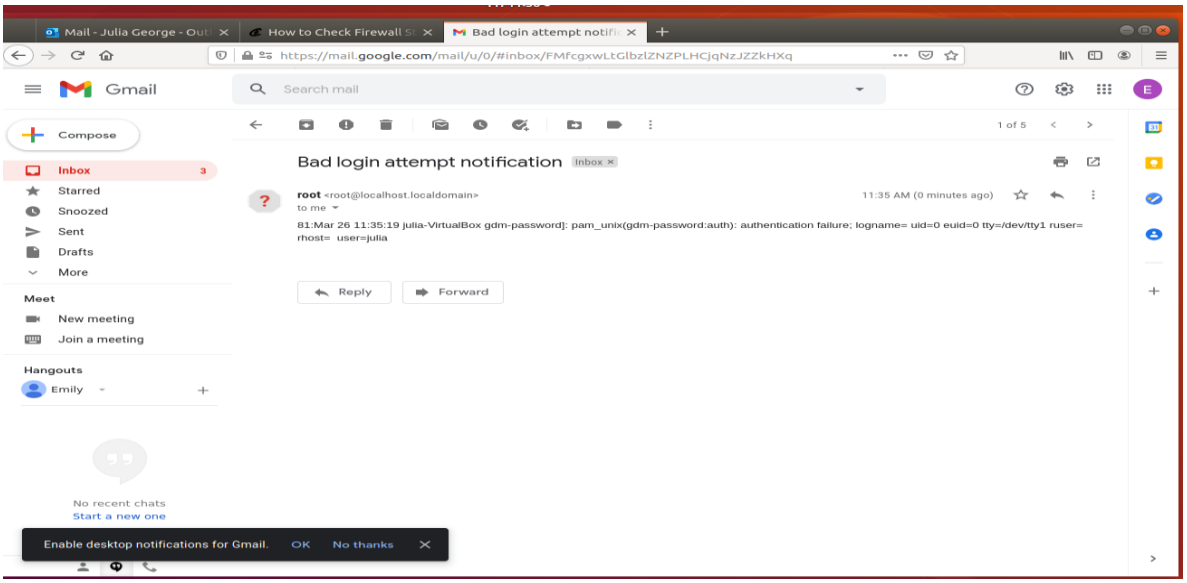


Image 10: Email sent when bad in login attempt happen.

Access Control Policy OS, Nagios, MYSQL

Policy for password change for users:

- Password must have upper- and lower-case letters.
- Passwords must be between 8 and 15 characters long.
- Passwords must contain at least one number
- Passwords must contain at least symbols
- Password must be changed every 180 days

Image 11: Access Control List Password Policy

Users and roles assigned to them:

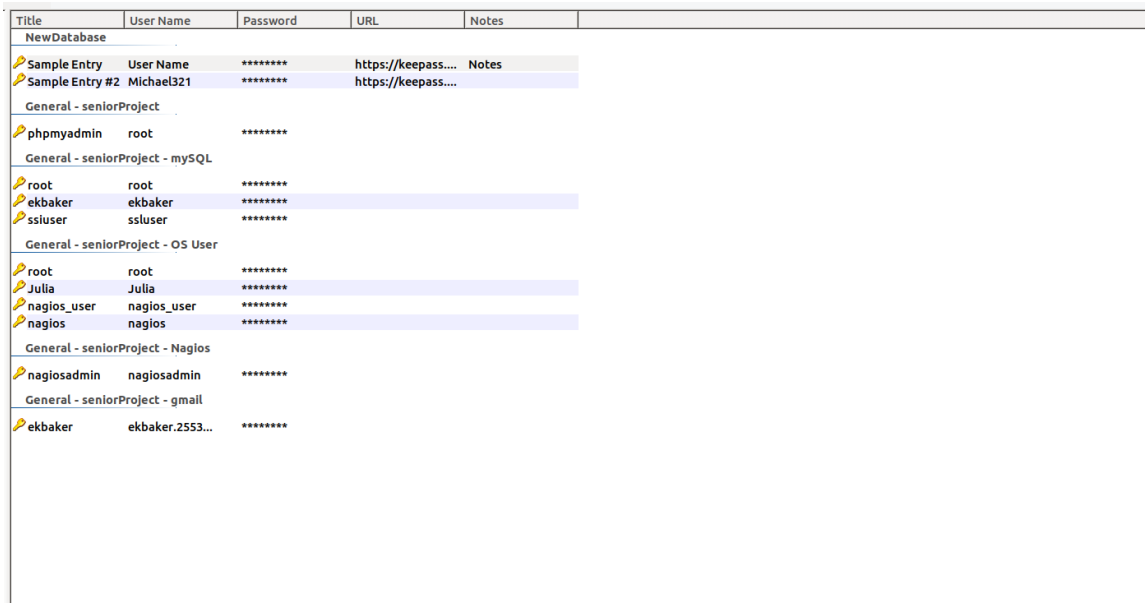
Users	Programs	Descriptions	Password Change
Root	OS	Full Access	August 17, 2020
Julia	OS	Local user	August 17, 2020
nagios_user	nagios	runs nagios service	February 11, 2021
nagios	Nagios	Used to access nagios home	February 11, 2021

Image 12: Access Control List for Users and their roles.

Segregation of Duties:

Users	Programs	Descriptions
Julia	Implementor	One man team
Julia	Requestor	One man team

Image 13: Separation of Duties ACP



Title	User Name	Password	URL	Notes
NewDatabase				
Sample Entry	User Name	********	https://keepass....	Notes
Sample Entry #2	Michael321	********	https://keepass....	
General - seniorProject				
phpmyadmin	root	********		
General - seniorProject - mySQL				
root	root	********		
ekbaker	ekbaker	********		
ssiuser	ssluser	********		
General - seniorProject - OS User				
root	root	********		
Julia	Julia	********		
nagios_user	nagios_user	********		
nagios	nagios	********		
General - seniorProject - Nagios				
nagiosadmin	nagiosadmin	********		
General - seniorProject - gmail				
ekbaker	ekbaker.2553...	********		

Image 14: Overview of the users and their passwords stored in KeePass

The last three task for security of the server was implementing a firewall, using antivirus software, and keeping software up to date. For implementing a firewall, I used ufw since it is open-source, and it is the only firewall I have ever used/installed on an Ubuntu Linux system. With the firewall implemented I created a list of allowable port numbers to restrict the ports with access (See image 15). I did not use any antivirus software because I personally do not trust free versions of the software. They usually do not cover all scopes of malware and sometimes they have malware in them. I would prefer to have a more well-known antivirus program that big businesses use to prevent possible trojan attacks. With the updating of software, I do not have any automated updated occurring but every time I log into the virtual machine, I am using the command line to make sure that if there are new updates they are being applied.

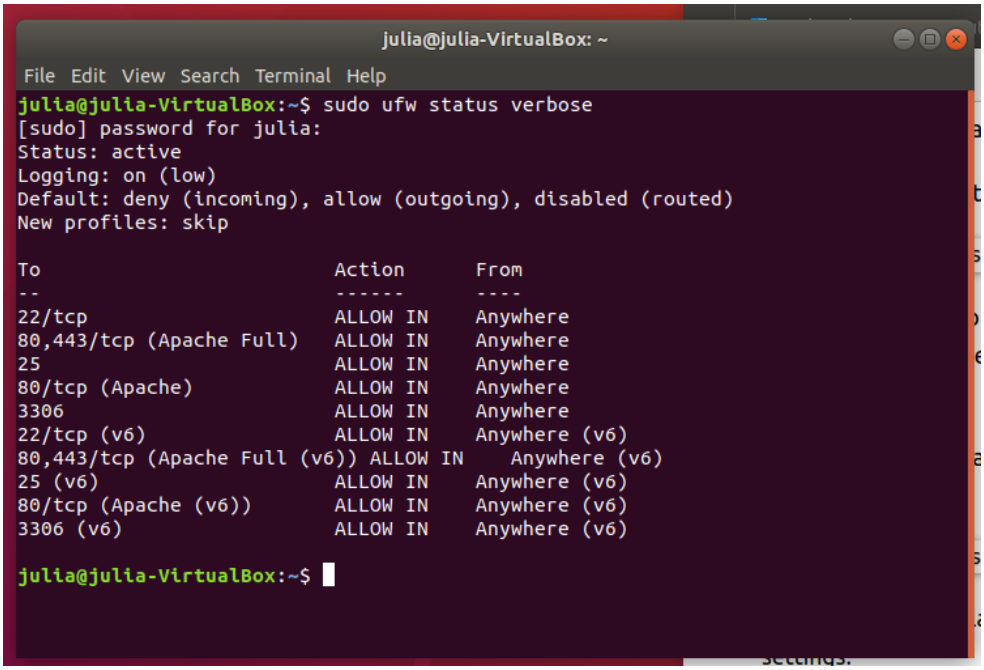


Image 15: Firewall active status and ports that are open, making sure no unnecessary ports are open.

After implementing basic hardening practices for the server, I started looking at the database. I decided to use MySQL because I am most familiar with how it works. Again, the book *Effortless E-Commerce with PHP and MySQL* second edition by Larry Ullman gave good starting points on basic hardening practices for a database. These are creating unique/secure usernames with unique secure passwords, change passwords regularly, change root user's password on new MySQL installation, all MySQL users restricted to connecting to MySQL only from localhost, and create a separate MySQL (using least privilege). During installation of the MySQL database, it prompts if you want validate password component, setting it to yes test passwords and improves security by checking strength of password. Next it asks what level of password validation policy with strong making the password have to be a length of 8 characters or longer, numeric, mixed case, special character, and checks dictionary. Choosing strong makes

sure the password is unique and the root user password will be strong (See image 16). After installation I created a user who was granted SELECT, INSERT, UPDATE, and DELETE privileges (See image 17). For this user I used the MySQL manual reference to alter the user's password to expire in 180 days (See image 18). The access control list is also implemented for the MySQL users and KeePass is used to store the passwords, plus keep track of expiration dates (See image 19).

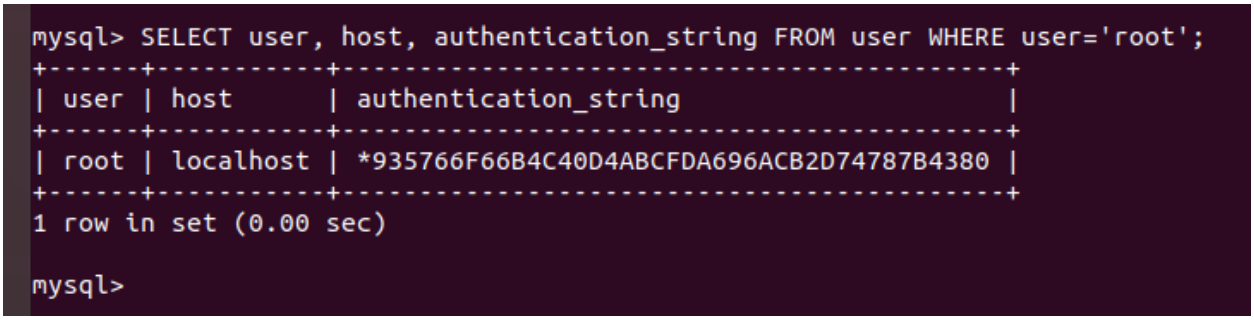


Image 16: Showing changed root password from default.

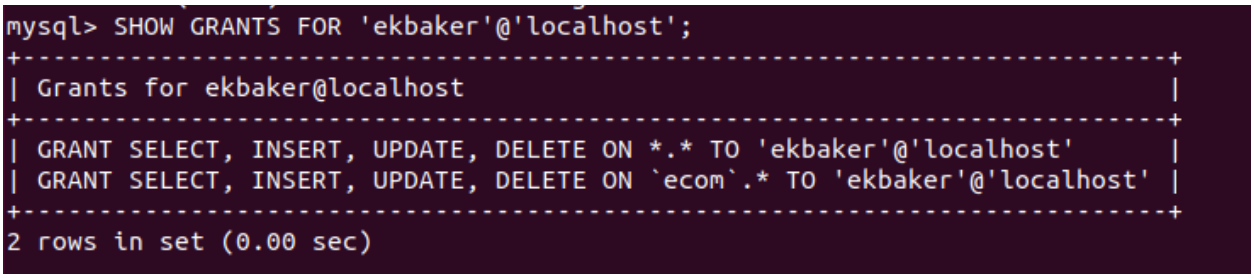


Image 17: Changing user privileges, for least privilege security.

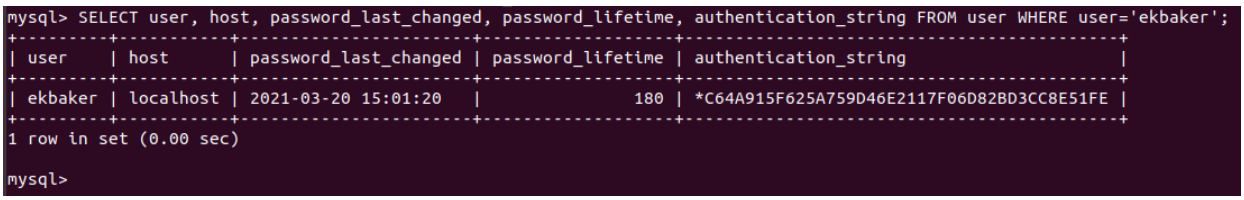


Image 18: Showing user password to expire in 180 days.

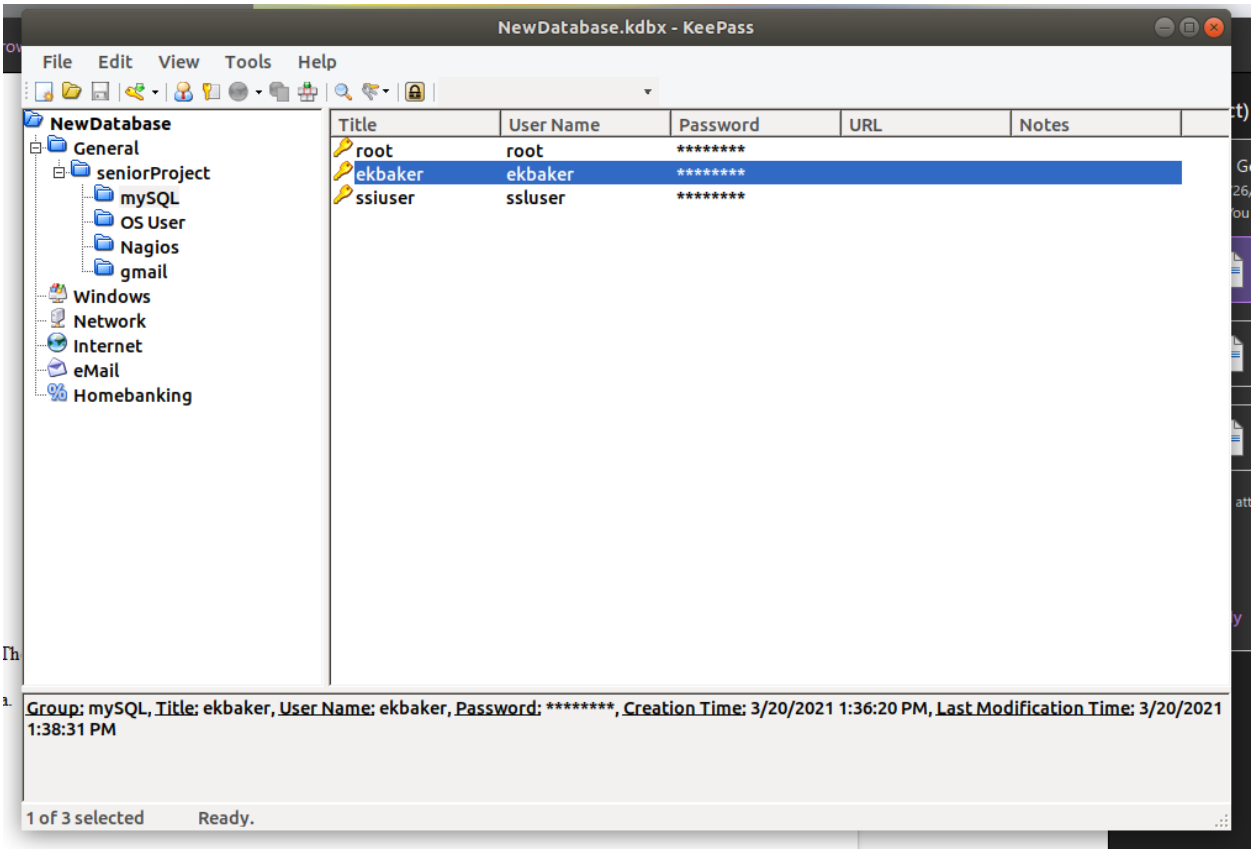


Image 19: KeePass keeping track of MySQL users.

After setting up the server and database I installed Visual Studio Code for the PHP, JavaScript, jQuery, CSS, and HTML used to create the website. A folder was created and stored in the /var/www that holds the pages used with the website. The first page a user will see when they visit the site is the login page (See image 20 & 21), login page is required to visit this site. This page uses jQuery and JavaScript to display errors if a user inserts blank information (See image 22), incorrect email format (See image 23), if the user attempts to insert a SQL injection (See image 24, 25, 26, & 27), and if the user attempts cross-site scripting attack (See image 28 & 29). When a user tries to insert malicious injections into the text fields when they press login the screen will reload, preventing information from being disclosed. The index.js file does client-side input validation by checking input and creating errors (See image 30). This checks if the

email is null and if the email is valid. If the input does exist in the database, then no errors will occur. The password needs to match the password in the database if not it will cause an error (See image 31). This file uses loginpassCheck.php, also a part of client-side validation, to make sure that the email and password being inputted exist in the database (See image 32). The file check2.php is used for the server-side validation, which checks if the input is empty and makes sure it is in the correct format (See image 33). This file also checks the password making sure it is not empty, checks to make sure the password is 6 to 14 characters in length (See image 34). If the email and password do not contain errors, then a statement is prepared to get the information from the database (See image 35).

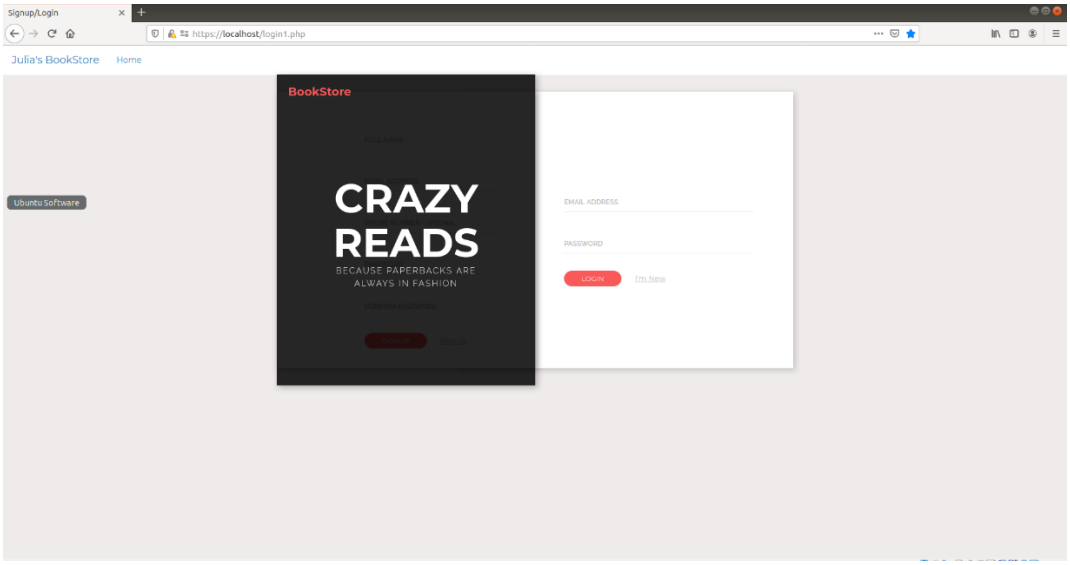


Image 20: What the user will see when they visit the site.

```
<div class="container">
   <section id="formHolder">

      <div class="row">

         <!-- Brand Box -->
         <div class="col-sm-6 brand">
            <a href="#" class="logo">BookStore</a>

            <div class="heading">
               <h2>Crazy Reads</h2>
               <p>Because paperbacks are always in fashion</p>
            </div>

            <div class="success-msg">
               <p>Great! You are one of our members now</p>
               <a href="#" class="profile">Your Profile</a>
            </div>
         </div>


         <!-- Form Box -->
         <div class="col-sm-6 form">

            <!-- Login Form -->
            <div id="login_form" name="login_form" class="login form-peice switched">
               <form class="login-form" name="login-form" id="login-form" action="login1.php" method="post">
                  <div class="form-group">
                     <label for="loginemail">Email Address</label>
                     <input type="email" name="loginemail" id="loginemail" class="loginemail">
                     <span class="errors" style="display:none" id="erroremail"></span>
                  </div>

                  <div class="form-group">
                     <label for="loginPassword">Password</label>
                     <input type="password" name="loginPassword" id="loginPassword" class="loginPassword">
```

Image 21: The code for the Login Page.

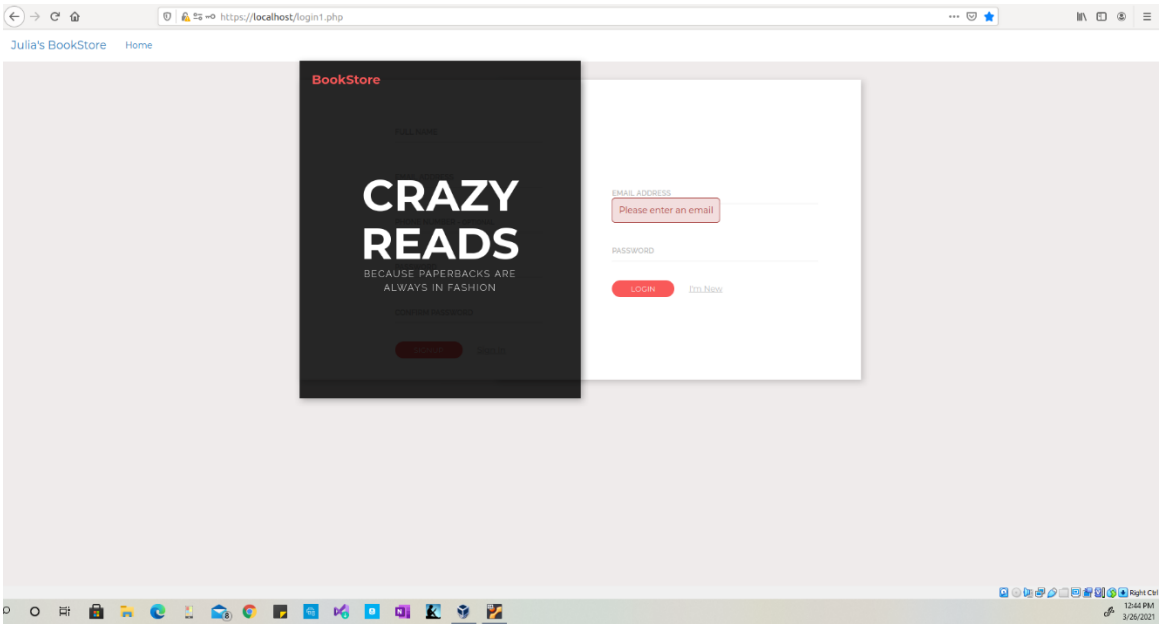


Image 22: Error message that displays when no email is entered.

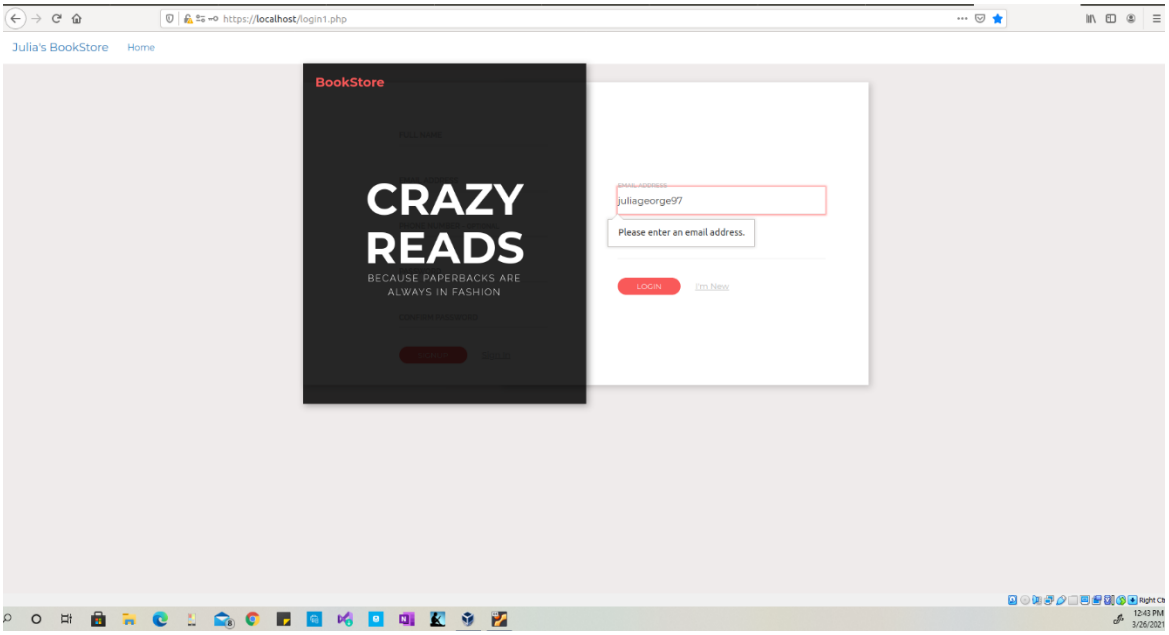


Image 23: Error message for unproperly formatted email.

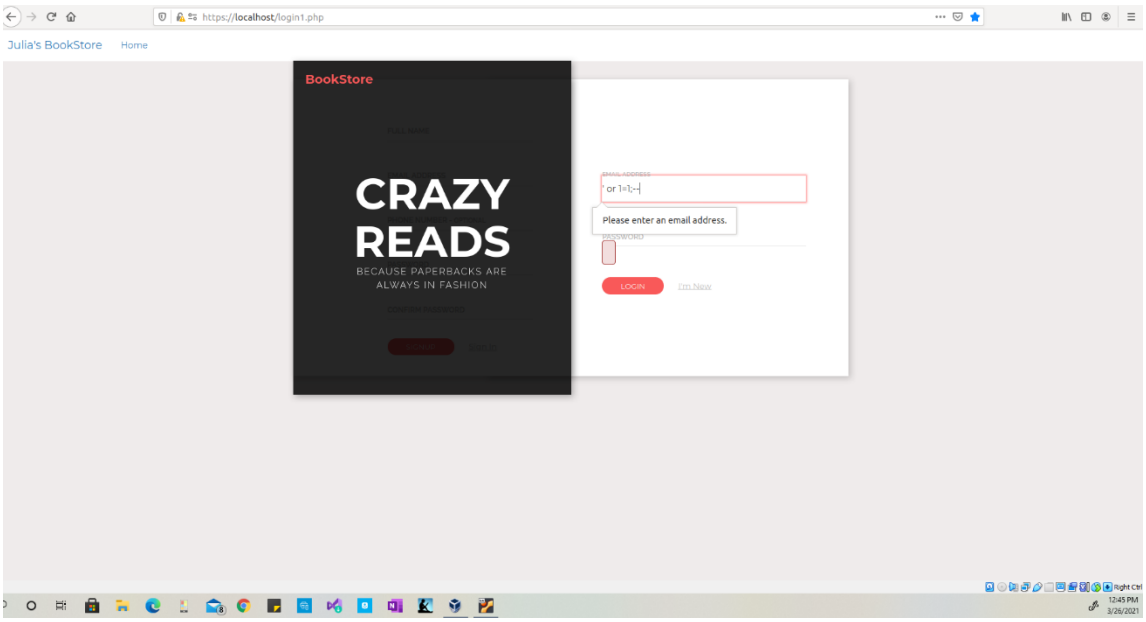


Image 24: SQL Injection attempt

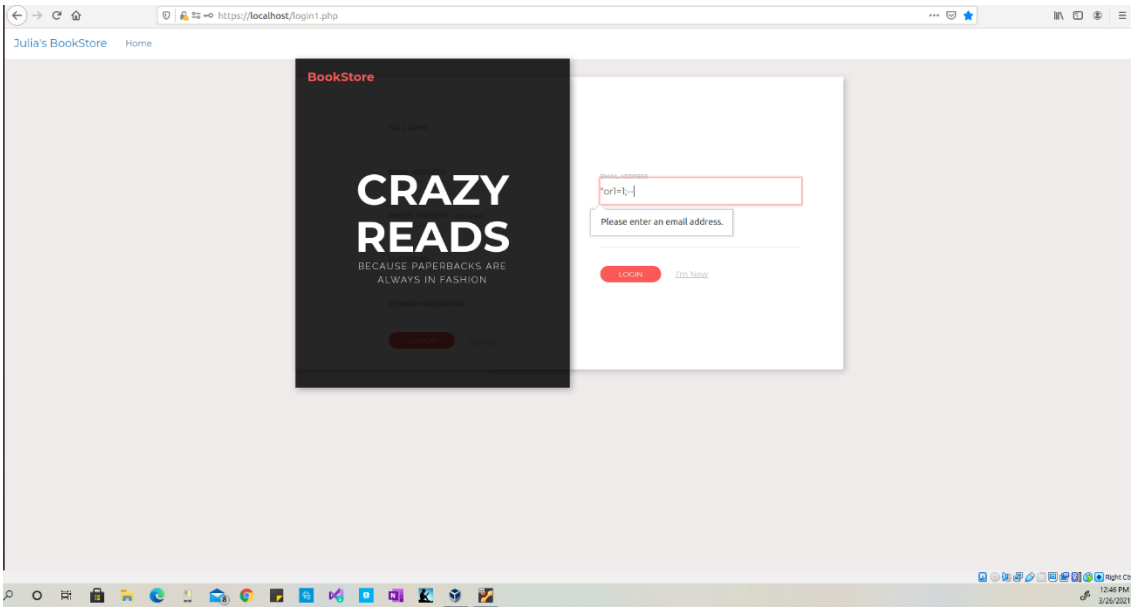


Image 25: SQL injection Attempt

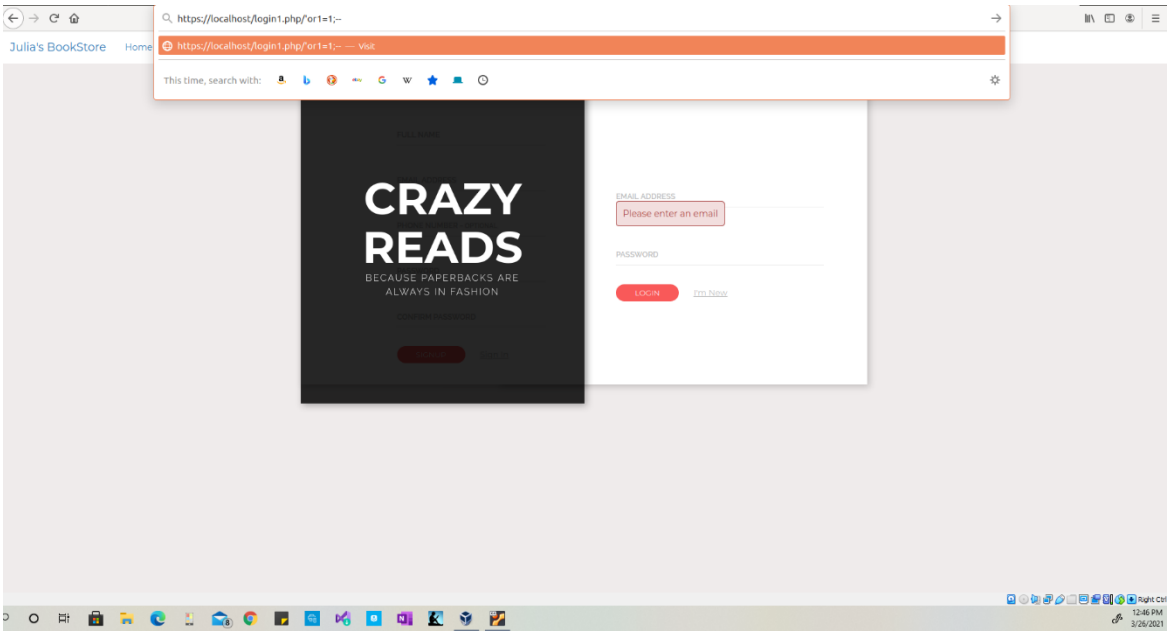


Image 26: Inserting a SQL injection into web browser

Image 27: The results of SQL injection in web browser.

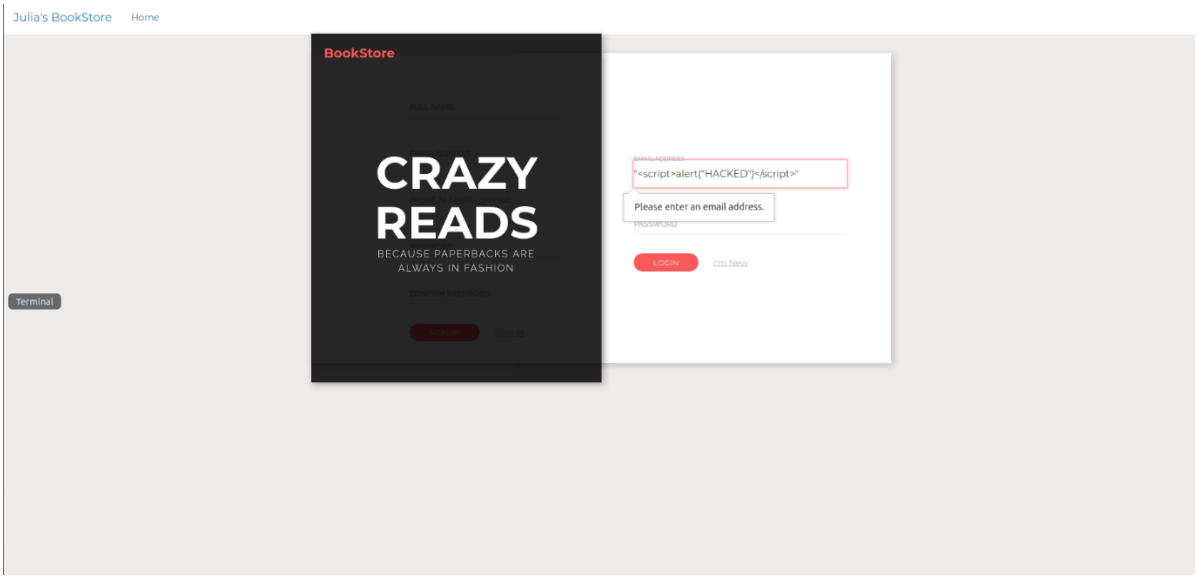


Image 28: XSS injection in text field.

Image 29: XSS injection in web browser

```
    //form validation used for form to sign in
$('input').blur(function () {
    if ($(this).hasClass('loginemail')){
    var loginEmailCk = $('#loginemail').val();
    var loginPassCheck = $('#loginPassword').val();
    //gets loginPassCheck.php page to help validate user input to see if the user exist or not.
    $.post('loginPassCheck.php' , {'loginemail' : loginEmailCk,'loginPassword' : loginPassCheck}, funct
    {
    //check if the login email is null.
    if ($('#loginemail').val().length == ""){
    $('#loginemail').siblings('span.errors').text('Please enter an email').fadeIn().parent('.form-group
    loginEmailErr = true;
    }
    //Goes to loginPassCheck.php and checks the code if the mail is valid or not..
    else if (data==4){
    $('#loginemail').siblings('span.errors').text('').fadeOut().parent('.form-groups').removeClass('has
    loginEmailErr = false;
    }
    else {
    //if the code is not 4 then it passes check and suppresses the error
    $('#loginemail').siblings('.errors').text('').fadeOut().parent('.form-groups').removeClass('hasErro
    loginEmailErr = false;
    }
    });
    }
```

Image 30: index.js checks login email then creates errors if they do not meet certain conditions.

```
//checks the user password if it is correct or not for login
if ($(this).hasClass('loginPassword')) {
var loginEmailCk2 = $('#loginemail').val();
var loginPassCheck2 = $('#loginPassword').val();
$.post('loginPassCheck.php' , {'loginemail' : loginEmailCk2,'loginPassword' : loginPassCheck2}, fur
{
if (data==13 || data==4){
$('#loginPassword').siblings('span.errors').text('Either your user name or password is not correct.
loginPassErr = true;
}
else {
$('#loginPassword').siblings('.errors').text('').fadeOut().parent('.form-groups').removeClass('hasE
loginPassErr = false;
}
});
}
```

Image 31: index.js file code that checks the password and creates errors if certain conditions are not met.

```
$email = trim($_POST["loginemail"]);
$password = trim($_POST["loginPassword"]);

// Prepare a select statement
$sql = "SELECT cust_id, email, password FROM customer WHERE email= ?";

if($stmt = mysqli_prepare($con, $sql)){
    // Bind variables to the prepared statement as parameters
    mysqli_stmt_bind_param($stmt, "s", $param_email);

    // Set parameters
    $param_email = $email;

    // Attempt to execute the prepared statement
    if(mysqli_stmt_execute($stmt)){
        // Store result
        mysqli_stmt_store_result($stmt);

        // Check if username exists, if yes then verify password
        if(mysqli_stmt_num_rows($stmt) == 1){
            echo 1;
            // Bind result variables
            mysqli_stmt_bind_result($stmt, $id, $email, $hashed_password);
            if(mysqli_stmt_fetch($stmt)){
                if(password_verify($password, $hashed_password)){
                    // Password is correct
                    echo 2;
                } else{
                    // Display an error message if password is not valid
                    echo 3;
                }
            }
        } else{
            // Display an error message if username doesn't exist
            echo 4;
        }
    } else{
        echo "Oops! Something went wrong. Please try again later.";
    }

}
```

Image 32: The loginPassCheck.php file to check database for inputted information.

```php
//Login page variables
$email = $passwd = "";
$email_err = $password_err = "";

// Processing form data when form is submitted
if(isset($_POST['login'])){

    $regLoginEmail = trim($_POST["loginemail"]);
    //Check if login value from the form is empty.
    if(empty(trim($_POST["loginemail"]))){
        $email_err = "Error";
    }
    //Check if email is in the correct format email@emailprovider.com
    elseif(!filter_var($regLoginEmail, FILTER_VALIDATE_EMAIL)){
        $email_err = "error";
    }
    else{
        //If passes the above checks the users input from the form is set to variable $email.
        $email = trim($_POST["loginemail"]);
    }

    $regPassLogin = trim($_POST["loginPassword"]);
    //Check if the login password from form is empty.
    if(empty(trim($_POST["loginPassword"]))){
        $password_err = "Error";
    }
    //Used regular expression to check if the password had letters from 6 to 14 letters in length.
    elseif(!preg_match("/^[A-Za-z]\w{6,14}$/",$regPassLogin)){
        $password_err = "error";
    }
    else{
        /**If the value from the form passes the above checks for password then the users input from the form is set
         * to the variable $passwd.
         */
        $passwd = trim($_POST["loginPassword"]);
    }
```

Image 33: check2.php file checks the email on the server side.

```php
$regPassLogin = trim($_POST["loginPassword"]);
//Check if the login password from form is empty.
if(empty(trim($_POST["loginPassword"]))){
    $password_err = "Error";
}
//Used regular expression to check if the password had letters from 6 to 14 letters in length.
elseif(!preg_match("/^[A-Za-z]\w{6,14}$/",$regPassLogin)){
    $password_err = "error";
}
else{
    /**If the value from the form passes the above checks for password then the users input from the form is set
     * to the variable $passwd.
     */
    $passwd = trim($_POST["loginPassword"]);
}
```

Image 34: check2.php file checks the password on the server side.

```php
// Prepare a select statement to bring back customer id, email, and passwords
if(empty($email_err) && (empty($password_err))){
    $sql = "SELECT cust_id, email, password FROM customer WHERE email= ?";

    if($stmt = mysqli_prepare($con, $sql)){
        // Bind variables to the prepared statement as parameters
        mysqli_stmt_bind_param($stmt, "s", $param_email);

        // Set parameters
        $param_email = $email;

        // Attempt to execute the prepared statement
        if(mysqli_stmt_execute($stmt)){
            // Store result
            mysqli_stmt_store_result($stmt);

            // Check if username exists, if yes then verify password
            if(mysqli_stmt_num_rows($stmt) == 1){
                // Bind result variables
                mysqli_stmt_bind_result($stmt, $id, $email, $hashed_password);
                if(mysqli_stmt_fetch($stmt)){
                    if(password_verify($passwd, $hashed_password)){
                        // Password is correct, so start a new session
                        session_start();

                        // Store data in session variables
                        $_SESSION["loggedin"] = true;
                        $_SESSION["cust_id"] = $id;
                        $_SESSION["email"] = $email;

                        // Redirect user to welcome page
                        header("location: index.php");


                    } else{
                        // Display an error message if password is not valid
                        $password_err = "The password you entered was not valid.";
                    }
                }
            } else{
                // Display an error message if username doesn't exist
                $username_err = "No account found with that username.";
```

Image 35: After no errors then a SELECT statement gets the users information to log them in.

If a user inputs incorrect information and tries to login they will be switched to the signup page. Additionally, this can be accessed if a user clicks the ‘I’m New’ link they will be taken to the sign-up form (See image 36). The user’s full name, email address, password, and password confirmation fields cannot be left blank. If they are an error will be displayed (See image 37). The phone number can be left blank but if there is input it has to be 10 digits (See image 38). The

name needs to be over 5 characters since it is a full name (See image 39). The email has to be in proper format and not exist in the database (See image 40). The password has to start with a letter and the confirmation password has to match (See image 41). Users also cannot insert SQL injection, if they do an error will pop up (See image 42 & 43) and after pressing signup they will be redirected to a blank sign-up page (See image 44). This page also protects against XSS scripting in text fields and in the browser bar (See image 41 & 45).

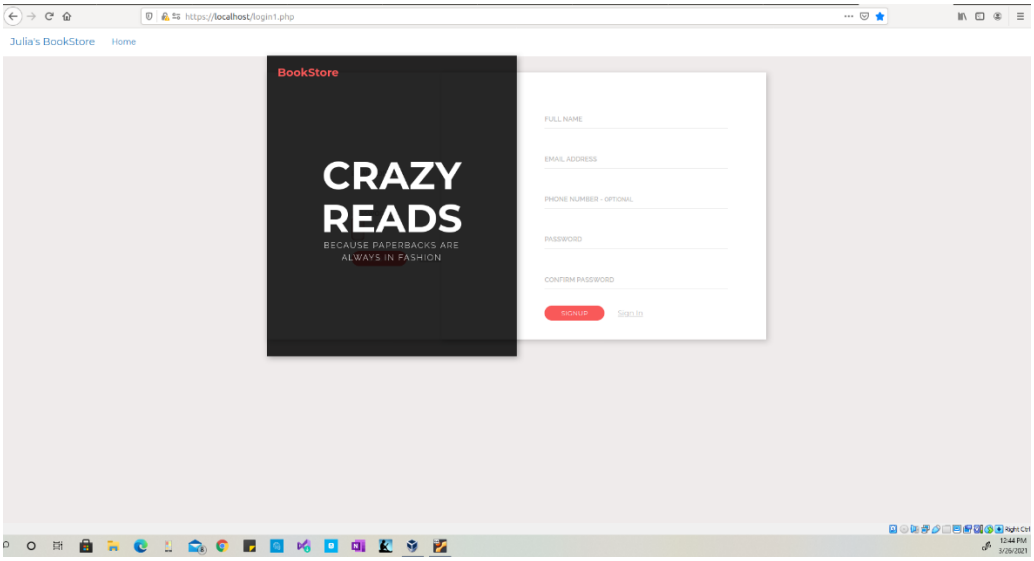


Image 36: The signup page

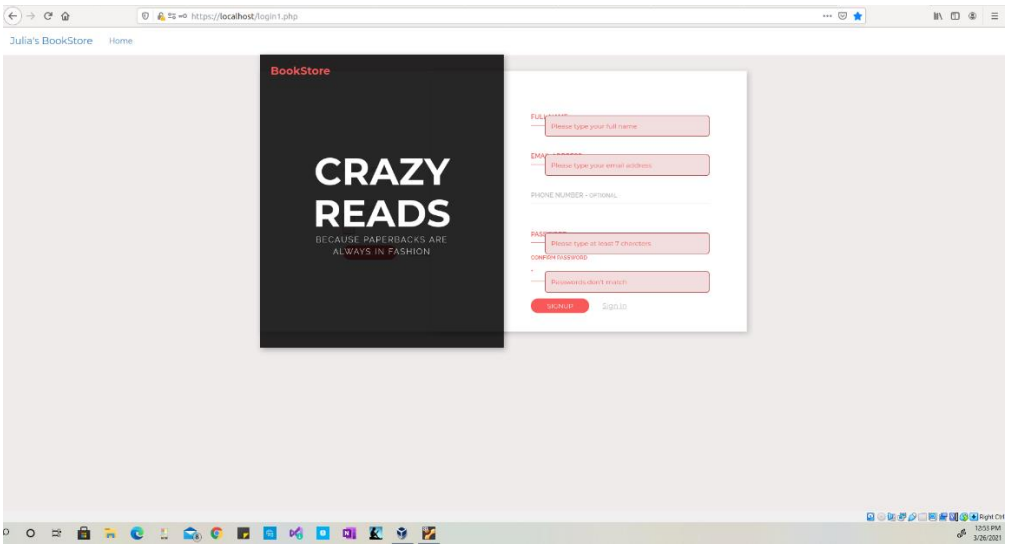


Image 37: Errors for blank input fields.

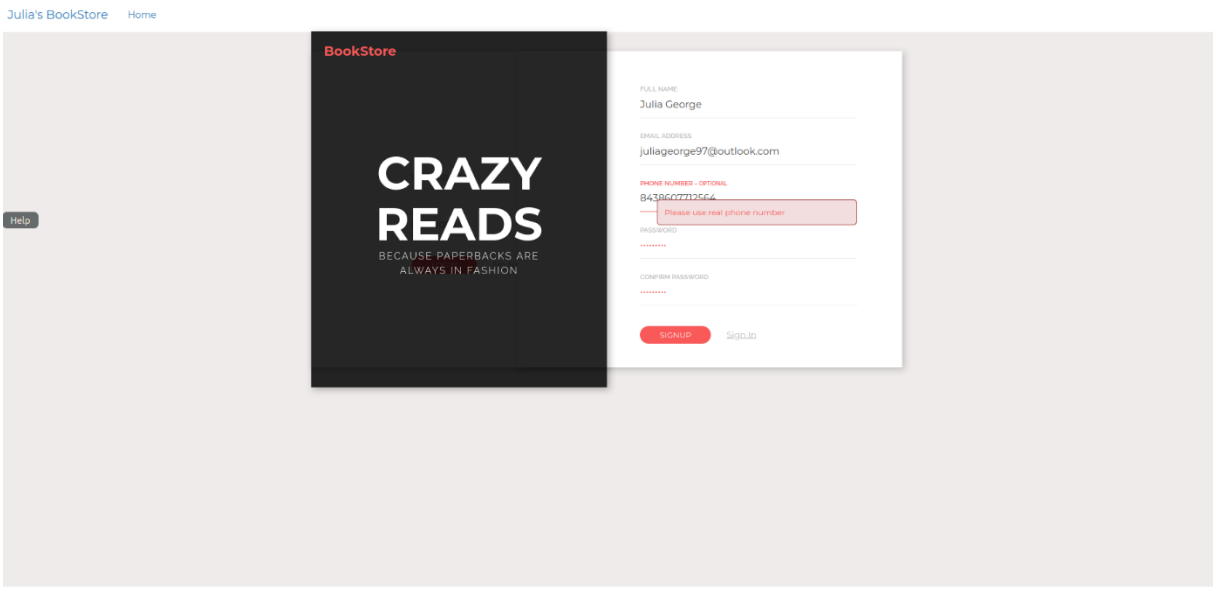


Image 38: Sign up error is the user inserts more than 10 digits.

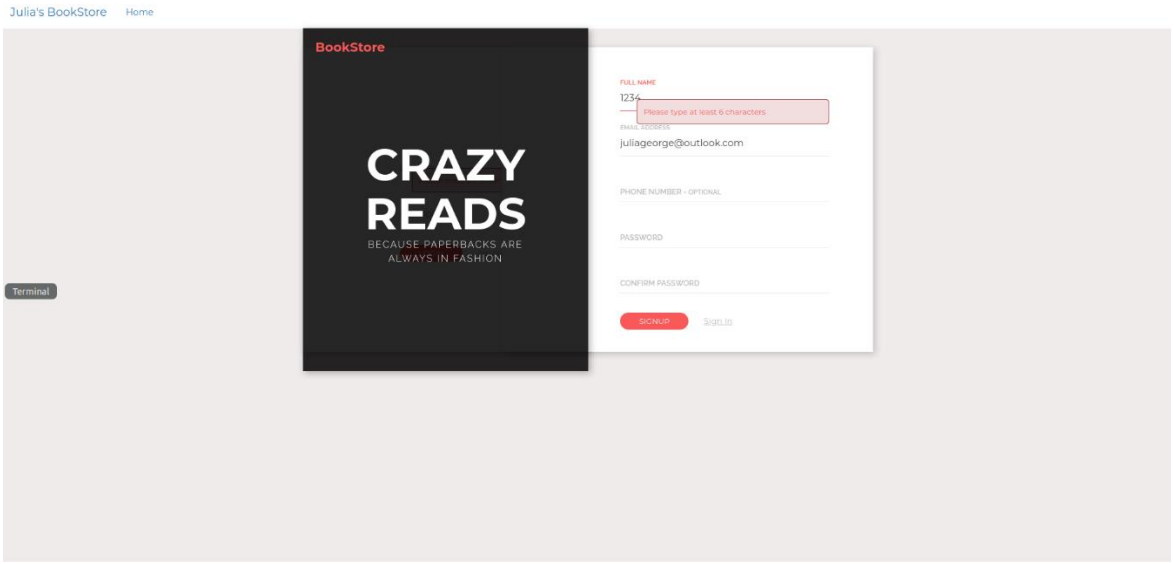


Image 39: Error displayed for less than 6 characters.

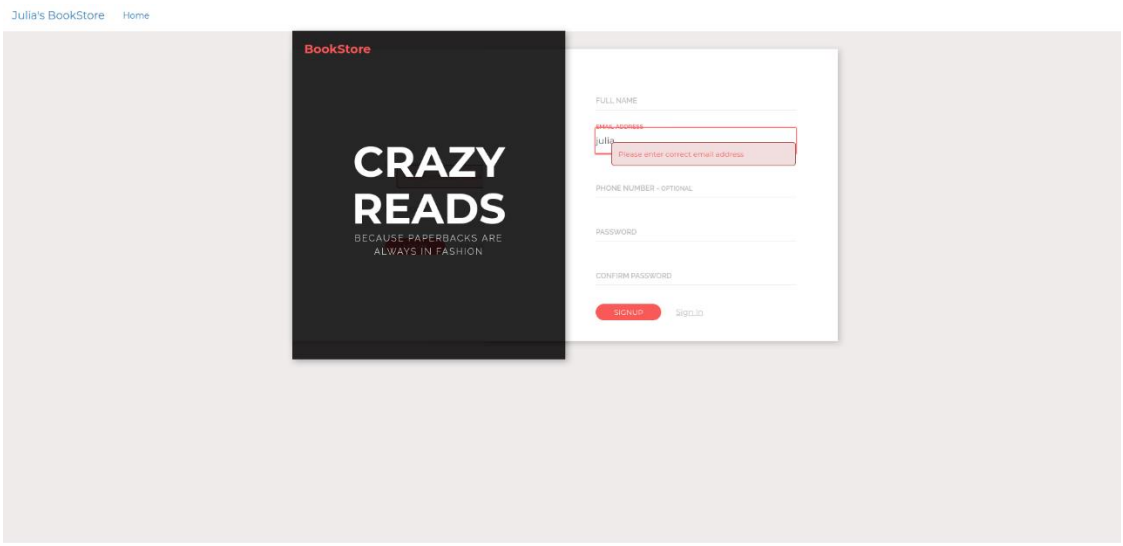


Image 40: Error displayed when the email format is not correct.

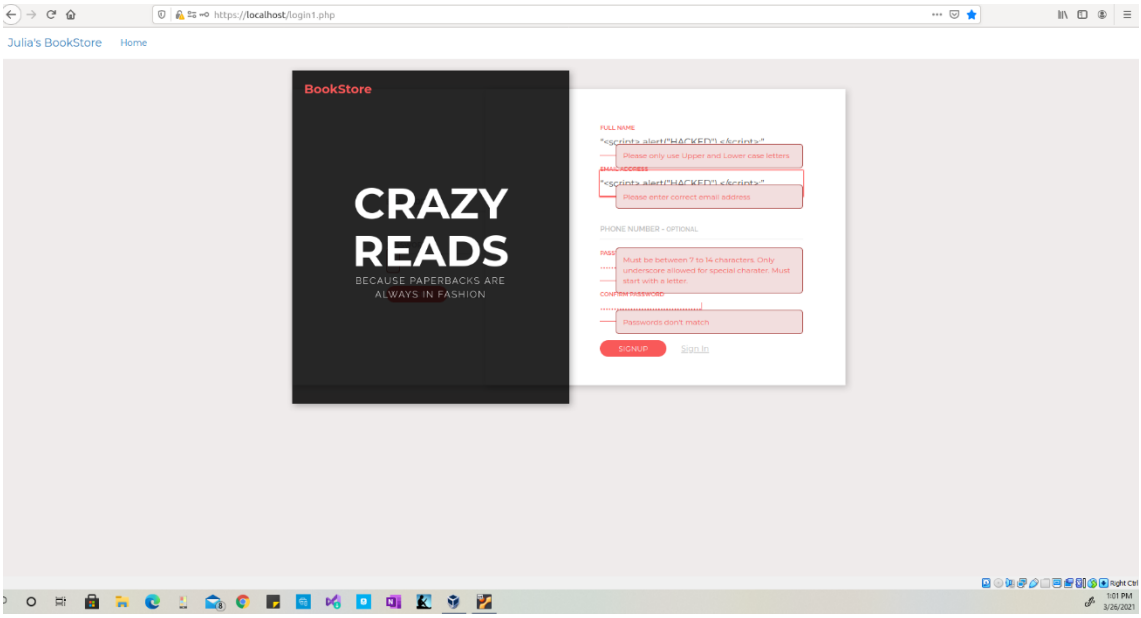


Image 41: Displays error messaged for password and errors if XSS injection inserted.

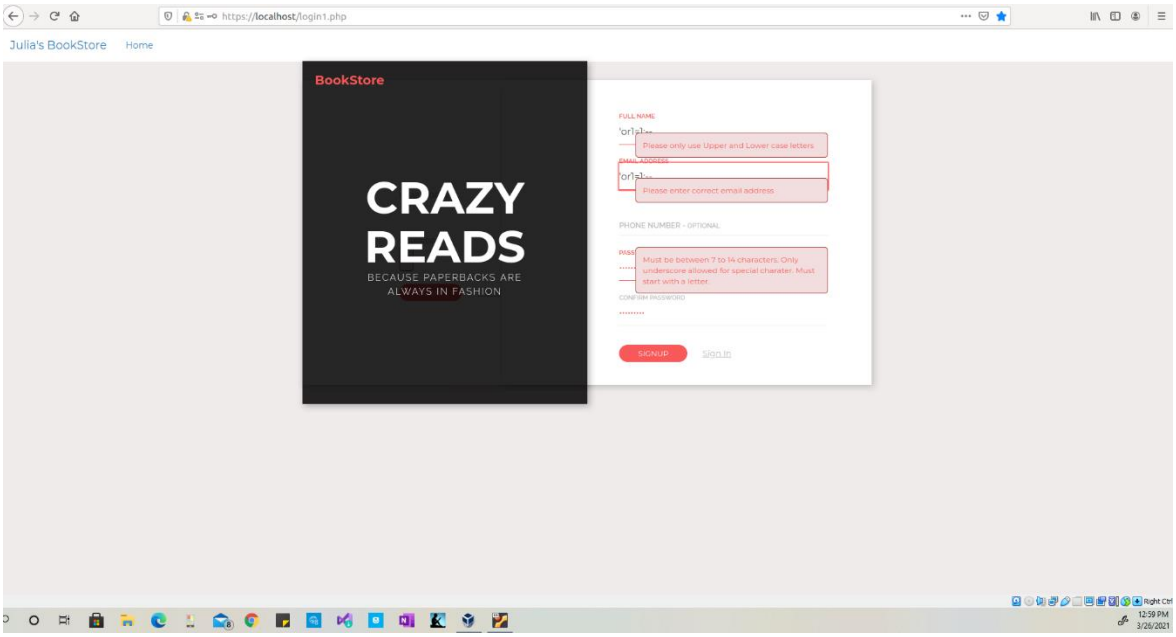


Image 42: SQL injection inserted into input fields.

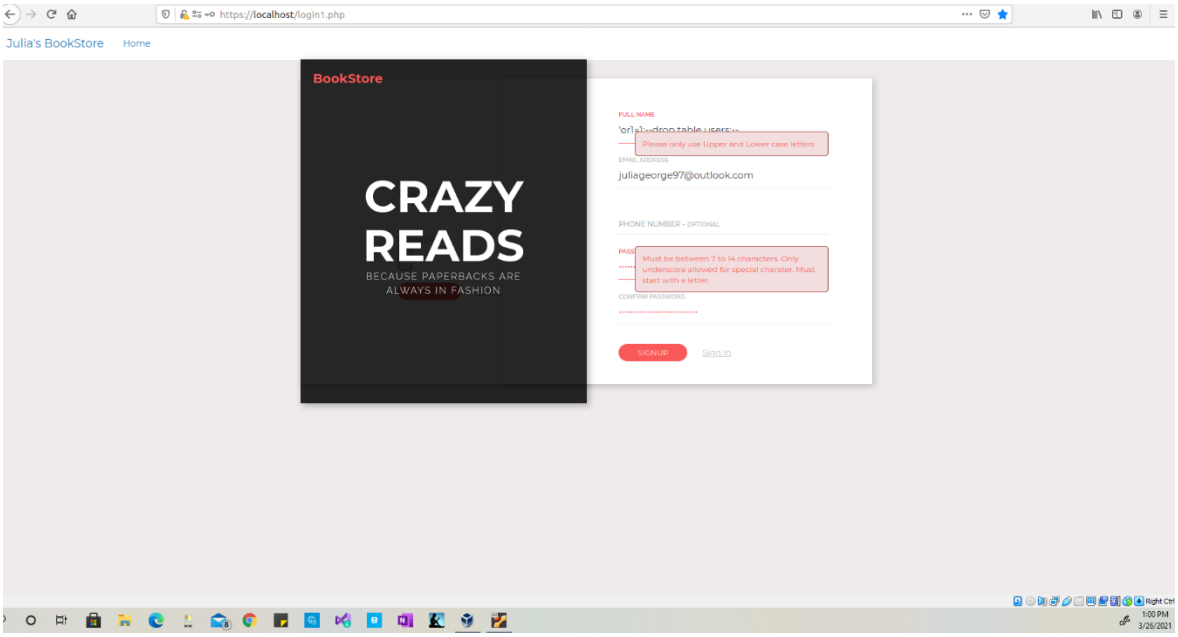


Image 43: SQL Injection inserted into input fields.

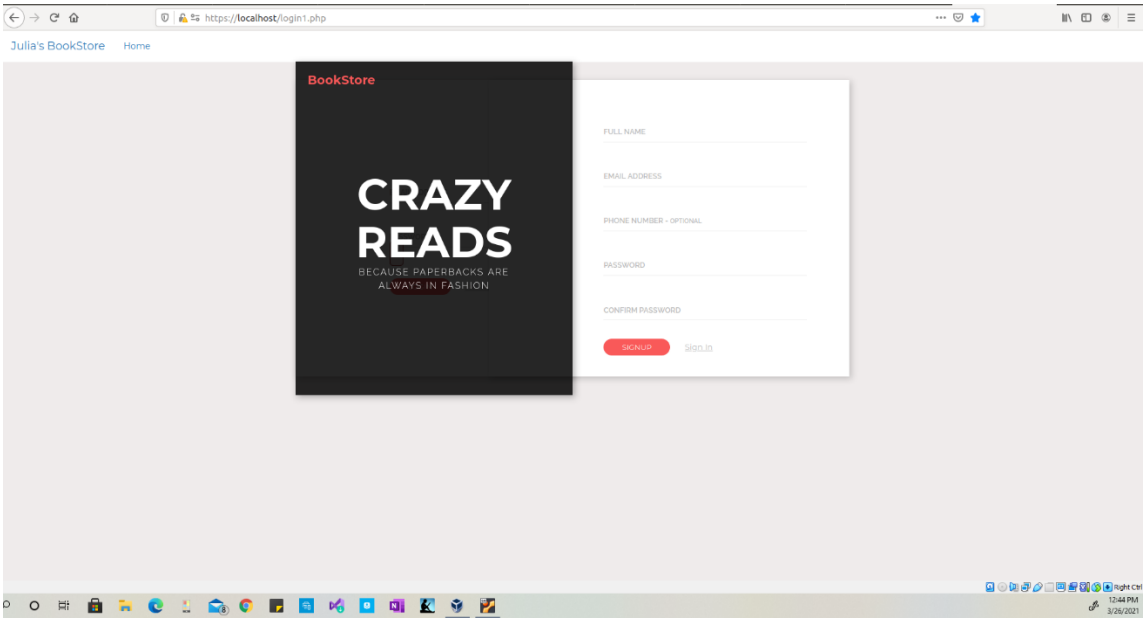


Image 44: After submitting SQL injection the user is redirected to a new signup page.

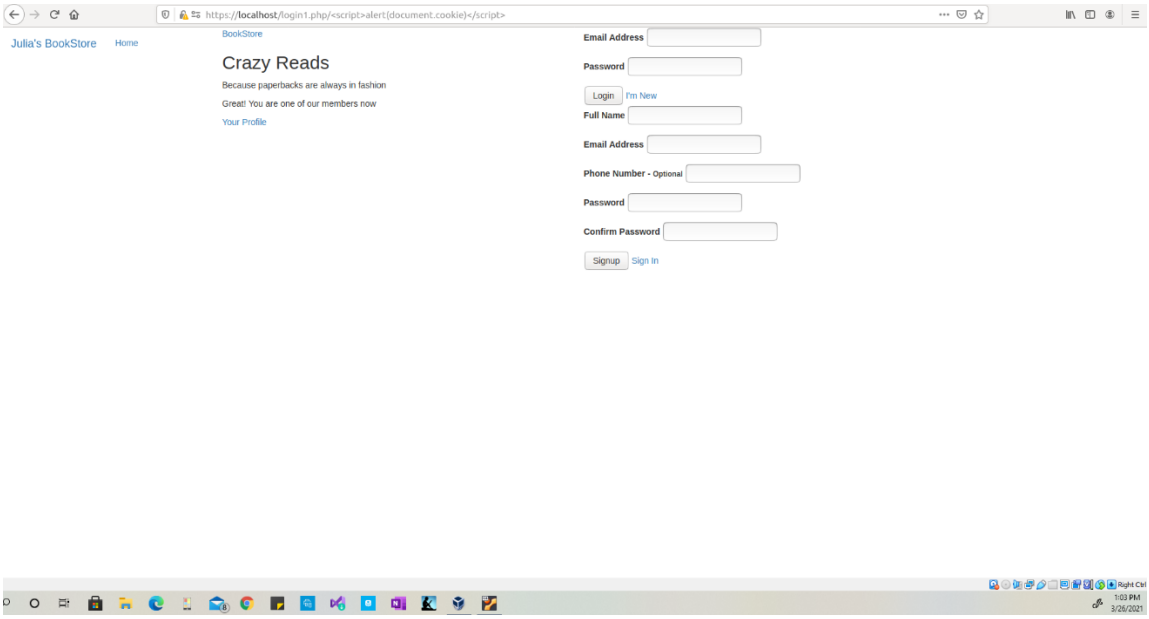


Image 45: The results of an XSS injection in the web browser.

The signup form uses the index.js file for client-side validation checking the users name, email, phone number, and the password. The username is checked to make sure the field is not blank, does not contain numbers, and is at least 6 characters (See image 46). The email is

checked to make sure the field is not blank, it is in correct format, and checks if the email is already in the database (See image 47). The file check.php is used with index.js to check the email to make sure it is not already in use (See image 48). The phone number field is tested to determine if anything is entered, if there is input then it is checked to make sure it is in proper format of 3 number, 3 numbers, and 4 numbers since dashes are accepted. It also makes sure that only numbers are being used in the text field (See image 49). The password field is checked to make sure the password is more than 7 characters but less than 14 and starts with a letter (See image 50). The last field is the password confirmation this checks to make sure it matches the input in the password field (See image 51). If any of these field produce an error, then it will display on the screen before the submit button is clicked.

```
// Form validation for the form to signup for a user account.
if($(this).closest('signup-form')){
    $('input').blur(function () {
        // User Name
        if ($(this).hasClass('name')) {
            //Uses regular expressions to have the user use upper and lower case letter. This was added by me.
            var lettersOnly = /^[a-zA-Z-' ]*$/;

            if ($(this).val().length === 0) {
                $(this).siblings('span.error').text('Please type your full name').fadeIn().parent('.form-group').addClass('hasError');
                usernameError = true;
                //Have user only allow length of the user has to be 6 or more characters.
            } else if ($(this).val().length > 1 && $(this).val().length <= 6) {
                $(this).siblings('span.error').text('Please type at least 6 characters').fadeIn().parent('.form-group').addClass('hasError');
                usernameError = true;
            }
            //Checks if the user only uses upper and lower case letters. This eas added by me.
            else if (!lettersOnly.test($(this).val())){
                $(this).siblings('span.error').text('Please only use Upper and Lower case letters').fadeIn().parent('.form-group').addClass('hasError');
                usernameError = true;
            } else {
                //If the user has the correct format in the text box then the error message is not displayed on the screen.
                $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
                usernameError = false;
            }
    }
```

Image 46: The index.js file that checks the username for errors before submitting input.

```
    // Email
    if ($(this).hasClass('email')) {
        //Uses regular expressions to check if the email is formatted correctly.
        var emailOnly = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/i;
        //Checks if the email is empty
        if ($(this).val().length == '') {
            $(this).siblings('span.error').text('Please type your email address').fadeIn().parent('.form-group').addClass('hasError');
            emailError = true;
        }
        //Checks the regular expression and test if the email is in the correct email format. Added by me.
        else if (!emailOnly.test($(this).val())){
            $(this).siblings('span.error').text('Please enter correct email address').fadeIn().parent('.form-group').addClass('hasError');
             emailError = true;
        }
        else{
            //Checks if email is in use or not.
            var emailcheck = $('#email').val();
            $('#emailcheck').html('<img src="loading.gif" width="150" />');
            //Check if email is already in the database through the code check.php page. Added by me.
            $.post('check.php' , {'email' : emailcheck}, function(data)
            {
                if(data==1){
                    $('#email').siblings('span.error').text('This email is already taken').fadeIn().parent('.form-group').addClass('hasError');
                    emailError = true;
                }
                else{
                    //If email is not in use then error is suppressed on the page.
                    $('#email').siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
                    emailError = false;
                }
            });
        }
    }
```

Image 47: The index.js file that checks the email field.

```
<?php
require_once("includes/config.php");
include("function/functions.php");

//Declare variables for email, name
$c_email2 = "";
$c_name = trim($_POST['name']);
$c_email = $_POST['email'];

//Get phone value and check if it is empty and if it is set to null else set value to form value user inputs for phone.
//if(empty(trim($_POST['phone']))){
// $c_phone = NULL;
//}else{
  //  $c_phone = trim($_POST['phone']);
//}

//Set password variable to password user sets.
$c_pass = trim($_POST['password']);

//Check if the email has already been used.
$sql2 = "SELECT email FROM customer WHERE email = ?";

    if($stmt2 = mysqli_prepare($con, $sql2)){
     // Bind variables to the prepared statement as parameters
         mysqli_stmt_bind_param($stmt2,"s",$param_email2);
    // Set parameters
         $param_email2 = $c_email;
    // Attempt to execute the prepared statement
             if(mysqli_stmt_execute($stmt2)){
        /* store result */
             mysqli_stmt_store_result($stmt2);
        if(mysqli_stmt_num_rows($stmt2) == 1){
             echo 1;

       } else{
            echo 0;
    }
}
    }
    ?>
```

Image 48: The check.php file to check the email to make sure it is not already being used.

```
//validate phone number if entered. JavaScript/JQuery added by me.
if ($(this).hasClass('phone')){
    //Use regular expressions to make sure only correct format is used for phone numbers.
    var phoneReg = /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/;
    //Checks if the length of the phone number value is not zero then it suppresses the error. Reason is the phone number is not a required field.
    if ($(this).val().length === 0){
       $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
       phoneError = false;
    }
    /**Checks the length of the value is not empty also test if the regular expression met if it is then it suppresses the error.
     * Nulls are allowed so when it test if it is empty this is allowed. */
    if(phoneReg.test($(this).val()) || $(this).val().length == ''){
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        phoneError = false;

     } else {
        //If the able is not met then it throws an error on the page.
        $(this).siblings('span.error').text('Please use real phone number').fadeIn().parent('.form-group').addClass('hasError');
        phoneError = true;
     }
}
```

Image 49: index.js file that checks the phone number field.

```
// PassWord
if ($(this).hasClass('pass')) {
    //Created regular expression to check the password is between 7 and 14 characters. Added by me.
    var passReg = /^[A-Za-z]\w{6,14}$/;
    //Checks to make sure the length us at least 7 letters long if not then it will throw an error on page.
    if ($(this).val().length < 7) {
        $(this).siblings('span.error').text('Please type at least 7 charcters').fadeIn().parent('.form-group').addClass('hasError');
        passwordError = true;
    //checks if the regular expression is met if it is suppresses the error. Added by me.
    } else if (!passReg.test($(this).val())){
        $(this).siblings('span.error').text('Must be between 7 to 14 characters. Only underscore allowed for special charater. Must start with a letter.').fadeIn
        passwordError = true;
        //Throws an error if the able conditions are not met.
    } else {
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        passwordError = false;
    }
}
```

Image 50: index.js file that checks the password field.

```
    // PassWord confirmation
    //Checks if the password confirmation text box is the same as the password text box if not throws an error.
    if ($('.pass').val() !== $('.passConfirm').val()) {
        $('.passConfirm').siblings('.error').text('Passwords don\'t match').fadeIn().parent('.form-group').addClass('hasError');
        passConfirm = true;
        //If it is then it suppresses the error
    } else {
        $('.passConfirm').siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        passConfirm = false;
    }

    // This creates the label effect of the lable showing up an disappearing.
    if ($(this).val().length > 0) {
        $(this).siblings('label').addClass('active');
    } else {
        $(this).siblings('label').removeClass('active');
    }

    });
}
```

Image 51: index.js file that checks the confirmation password field

When a user presses the submit button the input will be taken through the check2.php file and that will do server-side validation. The name will be checked if it is empty and has any numbers, if none of these are true then the value will be stored in $c_name (See image 52). The email will be checked if it is empty and not in valid format, if none of these are true then the input will be saved to $c_email (See image 53). The phone number field is checked if it is empty and checks to make sure is contains numbers and is only 10 digits, if these conditions are met it will save the input value to $c_phone (See image 54). Lastly, it checks the password field to see if it is empty, less than 6 characters or more then 14, and does not start with a letter (See image 55). If these conditions are false, then the password input will be saved to $c_pass. After checking all the inputs if they are saved to a variable then they will be sanitized before being inserted into the database to make sure no injections occur (See image 56 & 57). Once the user has signup, they will be redirected to the login page to login with their email and password.

```php
//Post form signup
if(isset($_POST['signup'])){

//Name verifaication
$regUsername = trim($_POST['name']);
//If name value from user is empty then set the username_err variable to error.
if(empty(trim($_POST['name']))){
    $username_err = "error";
}
//Use regular expression to check if the user name is in correct user input format is not it errors out.
elseif(!preg_match("/^[a-zA-Z-' ]*$/",$regUsername)){
    $username_err = "error";
}
else{
    //If the above checks pass then set the user name to variable $c_name
    $c_name = trim($_POST['name']);
}
```

Image 52: The check2.php file, checking the name input after submitting input

```php
//email verifaication
$regEmail = trim($_POST['email']);
//Check if the email value from the form is empty if so set variable to error.
if(empty(trim($_POST['email']))){
    $emailAdd_err = "error";
}
elseif(!filter_var($regEmail, FILTER_VALIDATE_EMAIL)){
    $emailAdd_err = "error";
}
else{
$c_email = trim($_POST['email']);
}
```

Image 53: check2.php file, checking the email input.

```php
//phone verification
$regPhone =  trim($_POST['phone']);
$ECPhone = str_replace([' ','.','-','(',')'],'',$regPhone);
if(empty(trim($_POST['phone']))){
    $c_phone = NULL;
}
elseif(preg_match('/^[0-9]{10}+$/',$ECPhone)){

    $c_phone = $ECPhone;
}
else{
    $phone_err = "error";
}
```

Image 54: check2.php file, checks the phone input

```php
//password verification
$regPasswd = trim($_POST['password']);
if(empty(trim($_POST['password']))){
    $pass_err = "error";
}
elseif(!preg_match("/^[A-Za-z]\w{6,14}$/",$regPasswd)){
    $pass_err = "error";
}
else{
$c_pass = trim($_POST['password']);
}
```

Image 55: check2.php file, checks the password input

```php
$sql2 = "SELECT email FROM customer WHERE email = ?";

if($stmt2 = mysqli_prepare($con, $sql2)){
   // Bind variables to the  prepared statement as parameters
  mysqli_stmt_bind_param($stmt2,"s",$param_email2);

   // Set parameters
  $param_email2 = $c_email;

   // Attempt to execute the prepared statement
   if(mysqli_stmt_execute($stmt2)){

      /* store result */
      mysqli_stmt_store_result($stmt2);

      if(mysqli_stmt_num_rows($stmt2) == 1){

        echo "1";

      }
    else {
       //delete safely
       //Create a statment that will allow user to be added to the users table.
       $query = "INSERT INTO customer (`cust_ip`, `name`, `email`, `password`, `phone`) VALUES (?,?,?,?,?)";
       //If the values are in error state than the mysqli_prepare staement will not be processed.
       if(empty($username_err) && empty($emailAdd_err) && empty($pass_err)){
           //Prepare statement is created for connection to the MySQL database.
           if($result = mysqli_prepare($con, $query)){
               //bind parameters to mysqli_stmt_bind_parameter. This is used so that MySQL injection statements cannot be used.
               mysqli_stmt_bind_param($result,"sssss",$param_ip,$param_name,$param_email,$param_password,$param_phone);
               //Set variables from bind statement to the user variables from the form.
               $param_ip = $ip;
               $param_name = $c_name;
               $param_email = $c_email;
               $param_password = password_hash($c_pass, PASSWORD_DEFAULT); // Creates a password hash
               $param_phone = $c_phone;
               echo $c_name;
               //Execute the statement
               if(mysqli_stmt_execute($result)){
                   //If the statement above is successful then the page will be redirected to the login.php page.
                   header("location: login1.php");
               }
```

Image 56: check2.php file, top portion to sanitize the input information and insert into the database.

```php
                else {
                    //If the executed statement failes then bringing back the below error on the page.
                    echo "Something went wrong. Please try again later.";
                }
                // Close statement
                mysqli_stmt_close($result);
            }
        }
    }
}
else {
    //Used for error catching
    echo "Oops! Something went wrong. Please try again later.";
}
// Close statement
mysqli_stmt_close($stmt2);
}
}
```

Image 57: check2.php file, end portion of sanitizing the input information and insert into the database.

Once logged on the user will be directed to the main page that consist of a top navigation bar with the name of the bookstore, a home tab, an order status, logout button, a greeting with the user's email, and a cart tab. There is a top portion of the page that displays a couple of books with the picture, title, author, and description. On the side there is more tabs that show all the books, adult books section, children books section, and teen books section (See image 58). The 'All Books' tab is what we will see as soon as we are logged in or click on the home tab. Clicking on the 'Adult' tab will display books that are considered to be in the adult category (See image 59). The 'Children' tab will display books that are in the children category (See image 60) and the 'Teen' tab displays books in the teen category (See image 61). Clicking on the 'Know More' button will show the more information about the book (See image 62). When a person decides they want a book they can press on the add button that will add it to the cart (See image 63). If a user has already added a particular book to the cart if they try to add it again, they will be told it is already in the cart (See image 64).

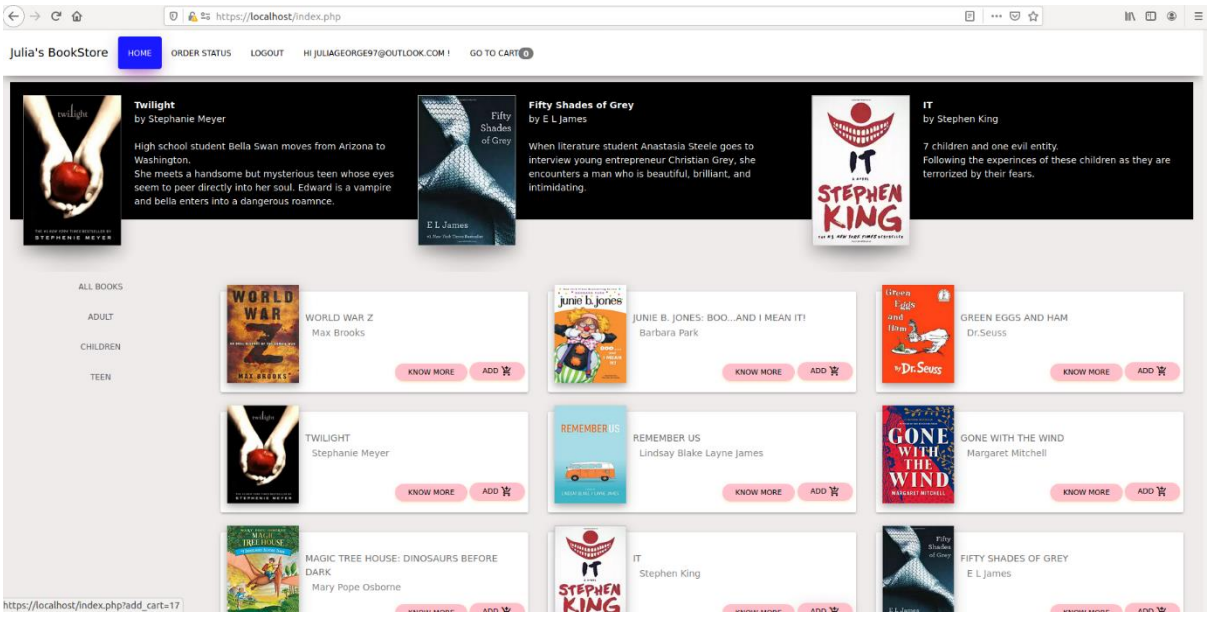


Image 58: Main page of the webpage

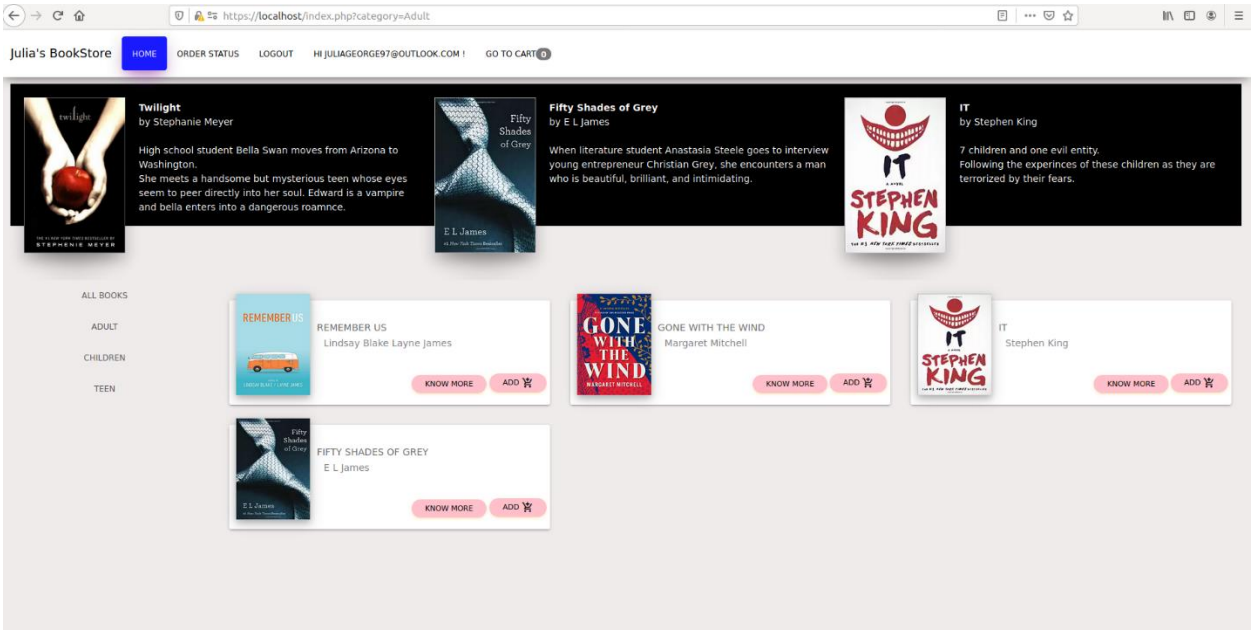


Image 59: Clicking on the adult side tab.

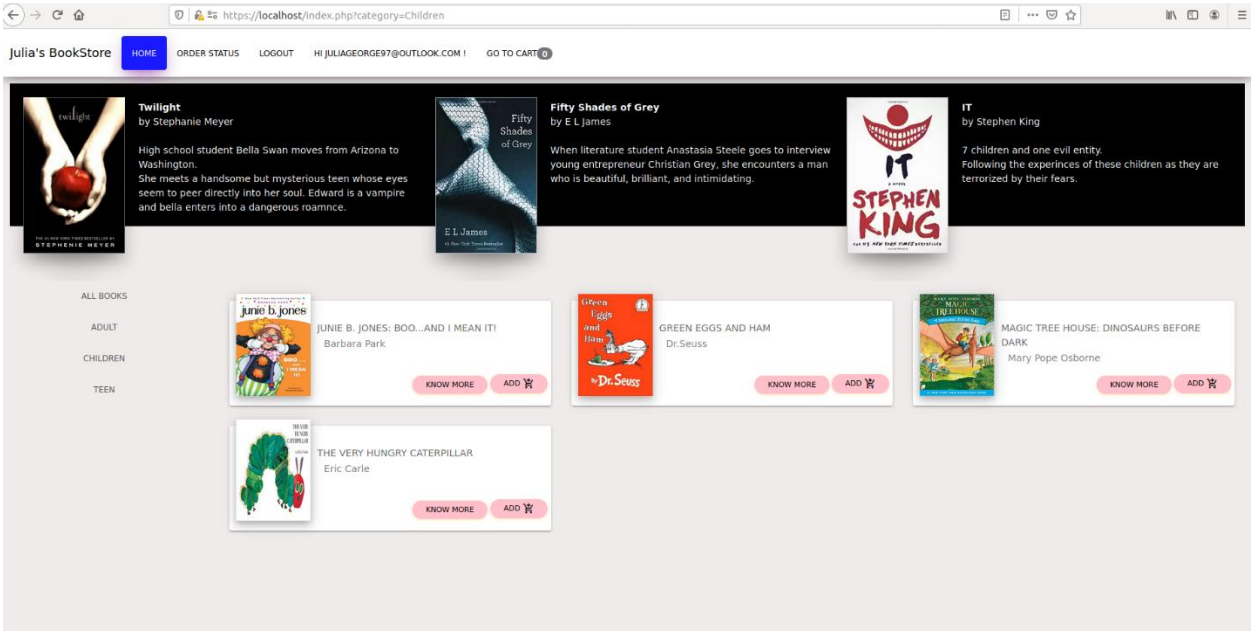


Image 60: Clicking on the children side tab.

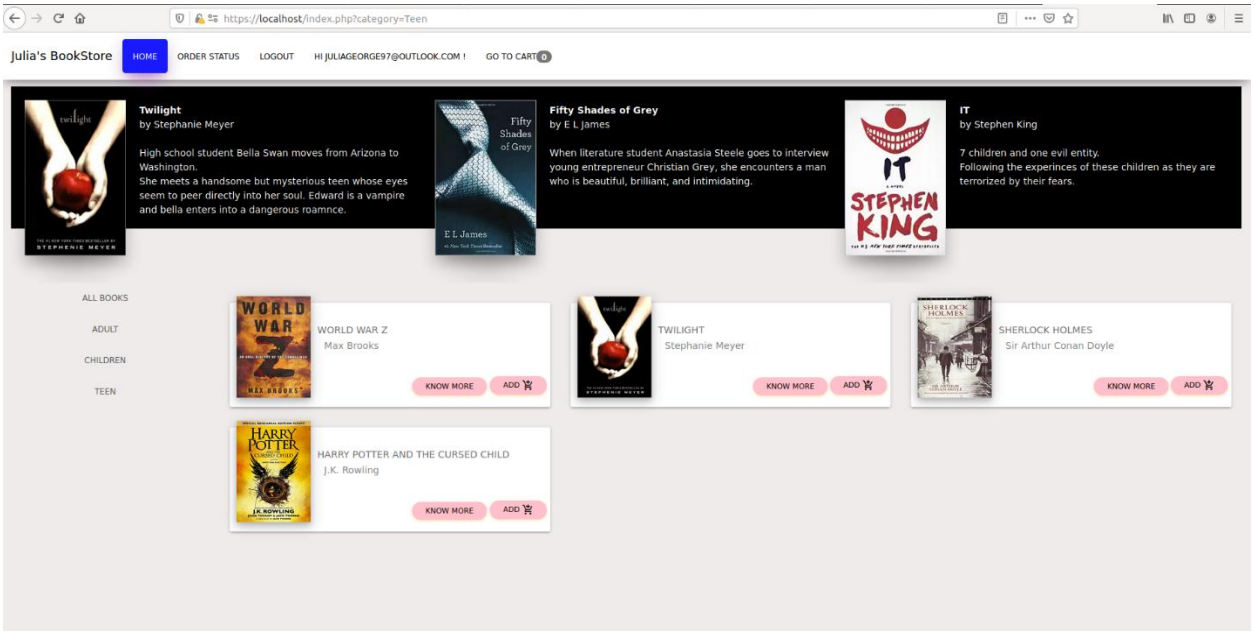


Image 61: Clicking on the teen side tab.

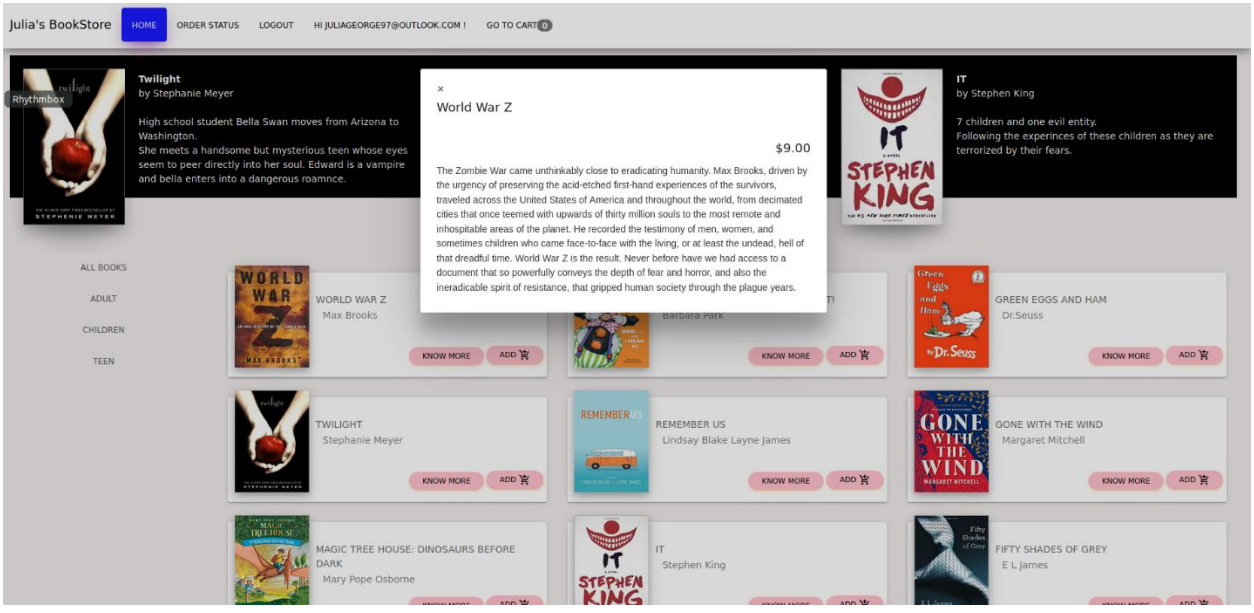


Image 62: Clicking on the know more button.

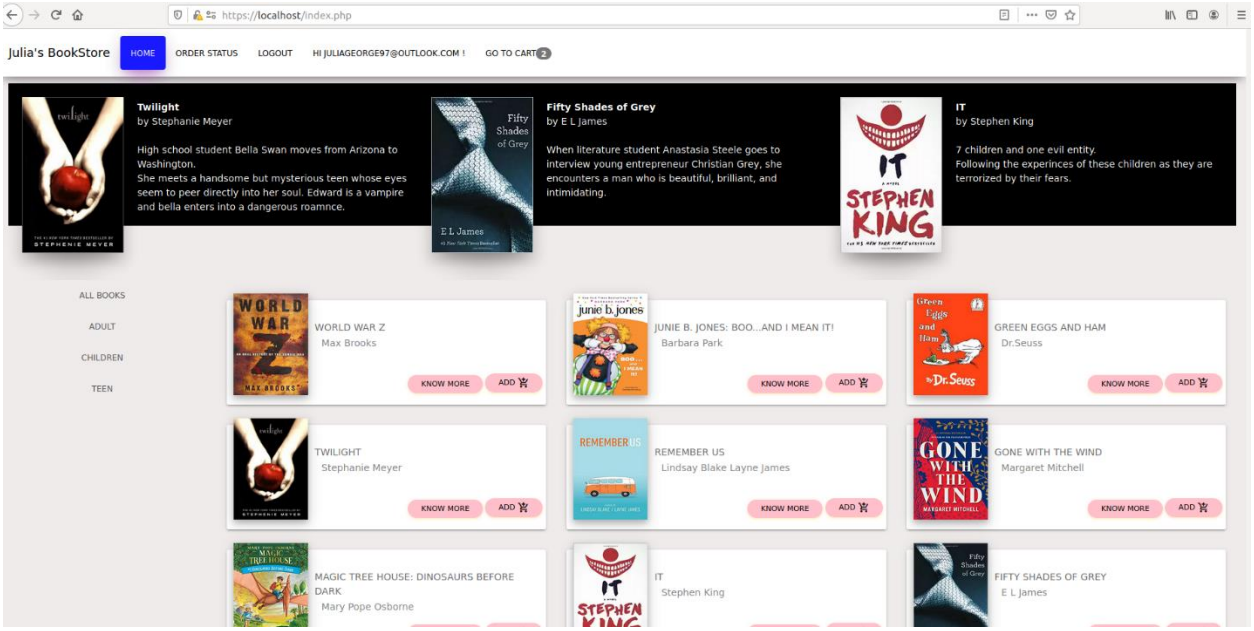


Image 63: After clicking the add button.

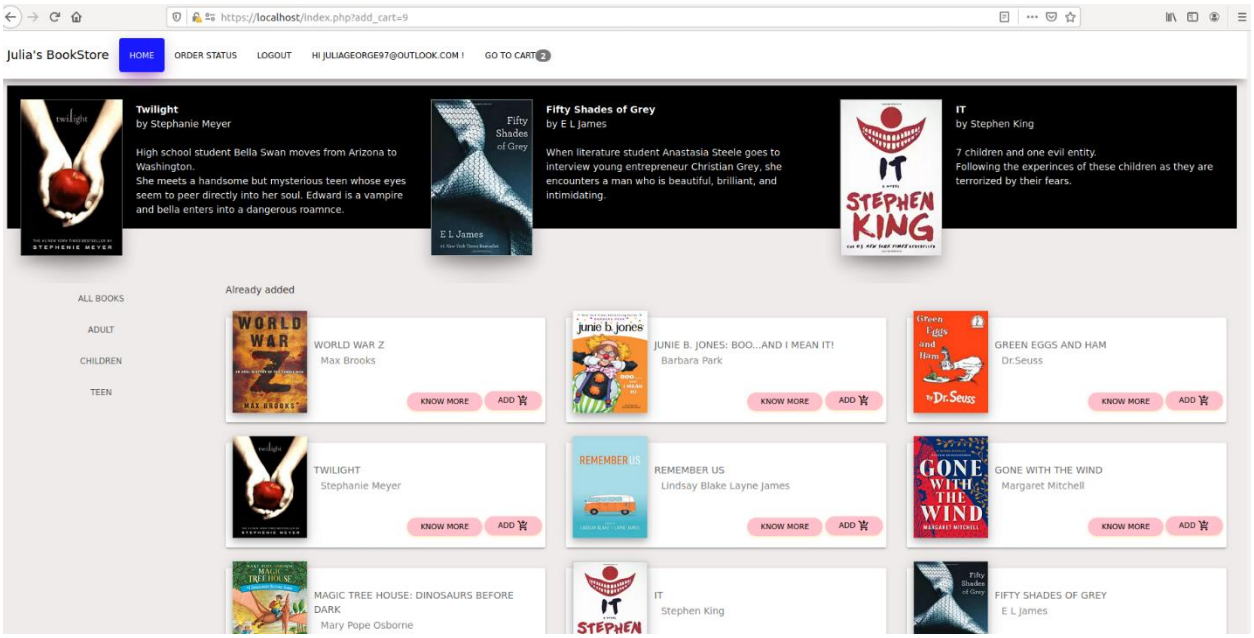
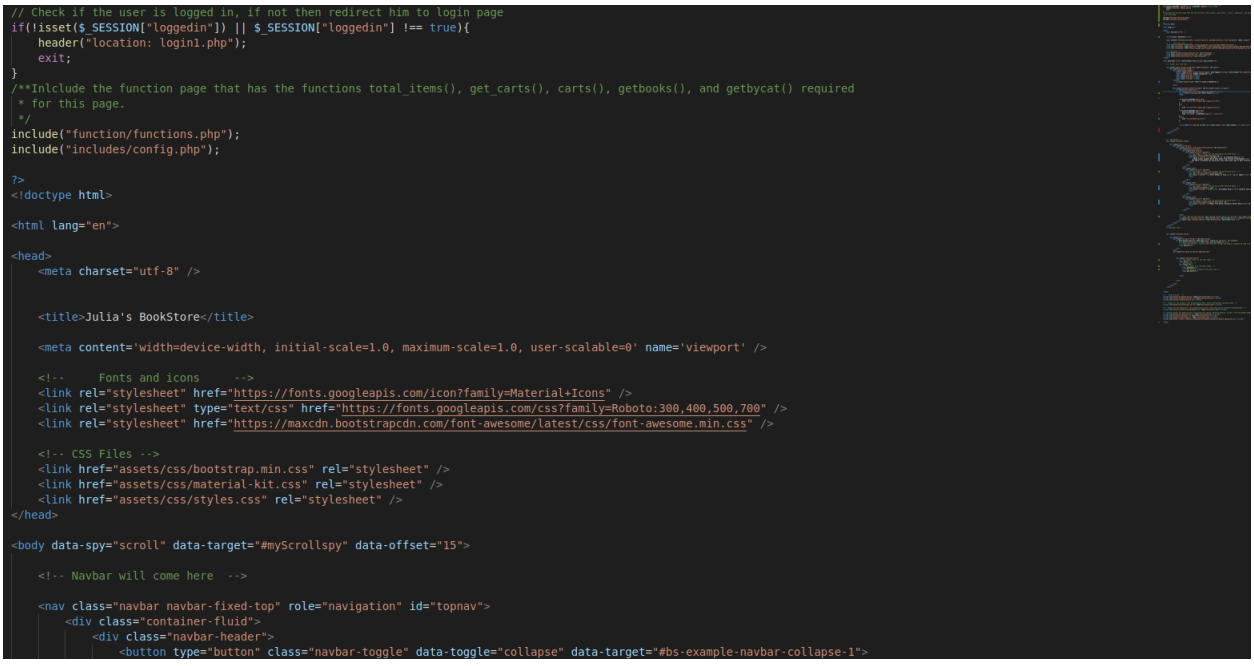


Image 64: After trying to add a book already in the cart.

Now to take a look at the code for these parts of the webpage, first looking at index.php that shows the layout of the page. The top portion of the code makes sure the user is logged in before they can visit this page and it makes sure the function.php file and config.php files are included in this page (See image 65). The second part of index.php shows the code used for the navigation bar. It has section that shows if the user is not logged in, they could be shown as a guest, but I chose to make sure they are logged in but kept this in there in case I decided to not have them logged in before viewing the main page (See image 66). This page also uses the function total_items () with the 'Go to Cart' link that connects to cart.php page (See image 66). The third section in the file is the carousel, which displays the books at the top of the page that has the title, author, and description (See image 67). The last section of code in the index.php file involves using functions in the function.php file to get the books based on category, to add books to the cart, get books from the books table, and get books by their category from the book table (See image 68).



```php
// Check if the user is logged in, if not then redirect him to login page
if(!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"] !== true){
    header("location: login1.php");
    exit;
}
/**Inlclude the function page that has the functions total_items(), get_carts(), carts(), getbooks(), and getbycat() required
 * for this page.
 */
include("function/functions.php");
include("includes/config.php");

?>
<!doctype html>

<html lang="en">

<head>
    <meta charset="utf-8" />


    <title>Julia's BookStore</title>

    <meta content='width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0' name='viewport' />

    <!--     Fonts and icons     -->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons" />
    <link rel="stylesheet" type="text/css" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700" />
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/latest/css/font-awesome.min.css" />

    <!-- CSS Files -->
    <link href="assets/css/bootstrap.min.css" rel="stylesheet" />
    <link href="assets/css/material-kit.css" rel="stylesheet" />
    <link href="assets/css/styles.css" rel="stylesheet" />
</head>

<body data-spy="scroll" data-target="#myScrollspy" data-offset="15">

    <!-- Navbar will come here  -->

    <nav class="navbar navbar-fixed-top" role="navigation" id="topnav">
        <div class="container-fluid">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">
```

Image 65: The top portion of the index.php code.

```
<nav class="navbar navbar-fixed-top" role="navigation" id="topnav">
    <div class="container-fluid">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
        </button>
            <a class="navbar-brand" href="#">Julia's BookStore</a>

        </div>

        <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
            <ul class="nav navbar-nav">
                <li class="active"><a href="index.php">Home</a></li>
                <li><a href="successpay.php">Order Status</a></li>
                <?php


                if(!isset($_SESSION['email'])){
                    echo "<li><a href='login1.php'>Login</a></li>";
                }
                else
                {
                    echo "<li><a href='logout.php'>Logout</a></li>";
                }
                 if(isset($_SESSION['email'])){
                    $sess=$_SESSION['email'];
                    echo "<li><a>Hi ".$_SESSION['email']." !</a></li>";
                }
                else {
                    echo "<li><a>Guest</a></li>";
                }
                ?>

                <li><a href="cart.php">Go to Cart<span class="badge"><?php total_items(); ?></span></a></li>

            </ul>
        </div>
    </div>
</nav>
```

Image 66: Navigation bar for the index.php file.

```
<div class="row">
    <div class="col-md-12">
        <div class="carousel slide multi-item-carousel" id="theCarousel">
            <div class="carousel-inner">
                <div class="item active">
                    <div class="col-xs-4" id="bk1">
                        <!-- jpg images, titles, and ook description declared below. -->
                        <img src="./assets/images/twilight.jpg">
                        <div class="c-content "><b>Twilight</b><br> by Stephanie Meyer<br><br>
                            <p>High school student Bella Swan moves from Arizona to Washington.</br>
                            She meets a handsome but mysterious teen whose eyes seem to peer directly into her soul. Edward is a vampire and bella enters into a
                        </div>

                    </div>
                </div>
                <div class="item">
                    <div class="col-xs-4" id="bk2">
                        <!-- jpg images, titles, and ook description declared below. -->
                        <img src="./assets/images/fiftyshades.jpg">
                        <div class="c-content "><b>Fifty Shades of Grey</b><br> by E L James<br><br> When literature student Anastasia Steele goes to interview y
                        </div>

                    </div>
                </div>
                <div class="item">
                    <div class="col-xs-4" id="bk3">
                        <!-- jpg images, titles, and ook description declared below. -->
                        <img src="assets/images/it.jpg">
                        <div class="c-content "><b>IT</b><br> by Stephen King<br><br>7 children and one evil entity. <br> Following the experinces of these child
                        </div>

                    </div>
                </div>
                <div class="item">
                    <div class="col-xs-4" id="bk4">
                        <!-- jpg images, titles, and ook description declared below. -->
                        <img src="assets/images/dinosaursbeforedark.jpg">
                        <div class="c-content "><b>Magic Tree House: Dinosaurs Before Dark</b><br> by Mary Pope Osborne<br><br> Two children and a treehouse.<br>
                        </div>

                    </div>
                </div>
```

Image 67: index.php file carousel portion of code.

```
                </div>
                <!-- This controls the left and right sliding console affect on the end of the images displayed. -->
                <a class="left carousel-control" href="#theCarousel" data-slide="prev"><i class="fa fa-chevron-circle-left" aria-hidden="true"></i></a>
                <a class="right carousel-control" href="#theCarousel" data-slide="next"></a>
            </div>
        </div>
    </div>
</div>
<!--carousel end-->



<div class="container-fluid">

    <div class="row">
        <div class="col-lg-2 col-md-2" id="myScrollspy">
            <ul data-offset-top="225" data-spy="affix" class="nav nav-pills  nav-stacked">
                <li role="presentation"><a href="index.php">All books</a></li>
                <!-- Calls the getcats() function that searches through the name of category by name from category table. -->
                <?php getcats();?>

            </ul>
        </div>
        <div class="col-lg-10 col-md-10" id="mainarea">



            <div class="container-fluid">
                <!-- This addes orders to the cart table -->
                <?php cart(); ?>
                <!-- Adding books -->
                <div class="row">
                    <!-- Gets books from the books table. -->
                    <?php getbooks();?>
                    <!-- Gets books by category from books table.-->
                    <?php get_bycat();?>


                </div>


            </div>
```

Image 68: index.php file, code showing functions to add books and get books to display.

Going to the functions page we will look at the code behind some of the functions that were seen in the index.php code. The first function we saw is the total_items() function, this is used to count the total items in the cart and displays that number by the cart tab (See image 69). This function uses the global variable $con that connects to the database. The variable $id is set to the customers id that we get from the session they started when they logged in. A variable $count is set to zero and then an if statement sees if the user clicked a book to add to the cart. A variable $query is set to a SELECT statement that gets the bookid, ip_add, cust_id from the cart table where the cart_id is equal to one unknown variable. The $stmt variable prepares for connection to the database to execute the SELECT statement assigned to $query. One unknown variable should be bound in the $id variable, this helps to prevent injections. The statement is

then executed, and the results are stored. The variable $count will count the number of rows returned from the SELECT statement. Finally, the number that was assigned to $count will be displayed. The second function is the getcats(), it defines the global variable $con that connects to the database. It then creates a query getting the name of the category then uses an if statement to make sure the statement is properly prepared to prevent an injection. The statement mysqli_prepare is connecting to database and preparing SQL statement for execution. The statement mysqli_stmt_execute is taking the value of $results and executes it. Then mysqli_stmt_bind_results will take the value of $results and binds it to $name. The while statement will take the $result value and will display the category names and then once all category names are displayed the mysqli_stmt_close will close the $result variable so the next statement can be executed (See image 70).

```php
//Function used to count total items in cart.
if(!function_exists(total_items)){
function total_items(){
    global $con;
    $id = $_SESSION["cust_id"];
    $count = 0;
    if(isset($_GET['add_cart']))
    {

        $query = "SELECT bookid, ip_add, cust_id FROM cart WHERE cust_id=?";
        if($stmt = mysqli_prepare($con,$query)){
          mysqli_stmt_bind_param($stmt,'s',$id);
          mysqli_stmt_execute($stmt);
          mysqli_stmt_store_result($stmt);
          $count=mysqli_stmt_num_rows($stmt);

        }
    }
    else {
        $query = "SELECT bookid, ip_add, cust_id FROM cart WHERE cust_id=?";
        if($stmt = mysqli_prepare($con,$query)){
          mysqli_stmt_bind_param($stmt,'s',$id);
          mysqli_stmt_execute($stmt);
          mysqli_stmt_store_result($stmt);
          $count=mysqli_stmt_num_rows($stmt);
          }
    }
    echo $count;
}
}
```

Image 69: total_items function code

```php
//Function used to display name from category table.
if(!function_exists(getcats)){
function getcats(){
  global $con;

  $query4="SELECT `name` from category";
  if($result=mysqli_prepare($con, $query4)){
    mysqli_stmt_execute($result);
    mysqli_stmt_bind_result($result,$name);
  while(mysqli_stmt_fetch($result))
  {
    echo "<li role=\"presentation\"><a href=\"index.php?category=".$name."\">".$name."</a></li>";
  }
   mysqli_stmt_close($result);
}
}
}
```

Image 70: Code for getcats() function.

Next the cart() function is used to add the book to the cart and also checks if that book is already in the carts table (See image 71). This function first brings in the global variable $con, defines the variable $ip to equal the function getIpAdd(), defines $id to the cust_id that is gotten from the logged in user, and the $book_id which gets from the add cart button. The variable $check_product is assigned the SQL statement that has a where statement gets cust_id and bookid from the website, it is unknown what those are so a question mark is put as a placeholder. Then an if else statement is used with the variable $run_check preparing the connection to the database using the $con and the query. Next the prepared statement is bound using two unknown variables that will be taken from $id variable and $book_id variable. Then the statement is executed, and the results are stored in the variable $run_check. The variable $num checks the number of rows that contain the results from the query and if there is a book already added in the table then the message ‘Already added’ is displayed. If the selected book is not in the table then the $run_check statement is closed so we can run another query. The variable $insert_cart is assigned an insert query that will insert the unknown variables for bookid, ip_add, and cust_id into the cart table. The connection will be prepared so that we can execute the query. The three

unknown values are bounded to $book_id, $ip, and $id then the statement is executed to save the information. After execution, the statement is closed.

```php
//Function used to check if the book is already in the cart table. If it isn't in the cart table then the user input
//is added to the cart table. Function used off the index page.
if(!function_exists(cart)){
function cart(){
    if(isset($_GET['add_cart']))
    {
        global $con;
        $ip=getIpAdd();
        $id = $_SESSION["cust_id"];
        $book_id=$_GET['add_cart'];
        $check_product="SELECT bookid, cust_id FROM cart WHERE  cust_id=? AND bookid=?";
        //mysqli_prepare prepared the $check_product statement for execution.
        if($run_check=mysqli_prepare($con, $check_product)){
          //Bind variables to the prepared statement as parameters
          mysqli_stmt_bind_param($run_check,'ss',$id,$book_id);
          //Attempt to execute the prepared statement
          mysqli_stmt_execute($run_check);
          //Store results
          mysqli_stmt_store_result($run_check);
          $num = mysqli_stmt_num_rows($run_check);
        if($num>0)
        {
          echo "Already added";
        }
        else {
          mysqli_stmt_close($run_check);
          $insert_cart="INSERT INTO cart(bookid, ip_add, cust_id) VALUES (?,?,?)";
          //mysqli_prepare prepares the $insert_cart statement for execution.
          if($run_cart = mysqli_prepare($con, $insert_cart)){
            //Bind variables to the prepared statement as parameters
            mysqli_stmt_bind_param($run_cart,'sss',$book_id,$ip,$id);
            //Attempt to execute the prepared statement
            mysqli_stmt_execute($run_cart);
            mysqli_stmt_close($run_cart);
            echo "added";
          echo "<script>window.open('index.php','_self')</script>";
          }
        }
      }
    }
}
}
```

Image 71: Code for cart() function.

Taking a look at the function that is used in the cart() function, the getIPAdd() function gets the IP address of the user (See image 72). It checks if the IP address is empty if not then it will be assigned to the variable $ip and then returned. The next function in the getbooks() function, which is used to display the books from the books table (See image 73). This function starts by bringing in the global variable $con, which connects to the database. A query is

determined the gets the book_id, author, keywords, title, price, image, description, and category of the book from the books table. This query is prepared with the variable $con that connects to the database. The results variable is executed then the results retrieved from the database are bind. Then a while statement is used that fetched the information and displays it. After all the information has been fetched and displayed the statement is closed.

```php
<?php

require_once("includes/config.php");

if(!function_exists(getIpAdd)){
function getIpAdd()
{
    if (!empty($_SERVER['HTTP_CLIENT_IP']))   //check ip from share internet
    {
      $ip=$_SERVER['HTTP_CLIENT_IP'];
    }
    elseif (!empty($_SERVER['HTTP_X_FORWARDED_FOR']))   //to check ip is pass from proxy
    {
      $ip=$_SERVER['HTTP_X_FORWARDED_FOR'];
    }
    else
    {
      $ip=$_SERVER['REMOTE_ADDR'];
    }
    return $ip;
}
}
```

Image 72: Code for getIpAdd() function.

```
//Function used to get books and display them from the Books table.
if(!function_exists(getbooks)){
function getbooks(){
  global $con;
    if(!isset($_GET['category'])){
  $query="SELECT `book_id`, `author`, `keywords`, `title`, `price`, `image`, `description`, `category` from `Books`";
  if($result=mysqli_prepare($con, $query)){
    mysqli_stmt_execute($result);
    mysqli_stmt_bind_result($result,$book_id,$author,$keywords,$title,$price,$image,$description,$category);

  while(mysqli_stmt_fetch($result))
  {

    echo "<div class='col-lg-4 col-md-6'>
                        <div class='card'>
                            <img class='card-img' height='200px' width='100px' src='assets/images/".$image."'>
                            <span class='content-card'>
                                <h6>".$title."</h6>
                                <h7>".$author."</h7>
                            </span>
                            <a href='index.php?add_cart=".$book_id."'><button class='buybtn btn btn-warning btn-round btn-sm'>
                  Add <i class='material-icons'>add_shopping_cart</i>
                </button></a>
                            <button class='knowbtn btn btn-warning btn-round btn-sm' data-toggle='modal' data-target='#".$book_id."'>
                  Know More
                </button>";

          //code for modal
        echo "<div class='modal fade' id='".$book_id."' tabindex='-1' role='dialog' aria-labelledby='myModalLabel' aria-hidden='true'>
                  <div class='modal-dialog'>
                    <div class='modal-content'>
                      <div class='modal-header'>
                        <button type='button' class='close' data-dismiss='modal' aria-hidden='true'>&times;</button>
                        <h4 class='modal-title' id='myModalLabel'>".$title."</h4>
                      </div>
                      <div class='modal-body'>
                      <h4><p align='right'>&#36;".$price."</p></h4>".
                          $description
                      ."</div>

                    </div>
                  </div>
                </div>
```

Image 73: Code for getBooks() function.

The get_bycat() function searches the books table by the category that the user selects and displays it on the page (See image 74 & 75). This function uses the global variable $con that connects to the database. It gets the category that the user selects and assigns it to the variable $cat_id. A SELECT statement to get the book_id, author, keywords, title, price, image, description, and category from the books table where the category is equal to the unknown category variable. An if statement is used that will prepare the connection and the query that will be executed. The $run_cat_pro will bind only allowing one variable to be used for the $cat_id variable. The statement will be executed, and the results will bind to their appropriate variables. Then the results will be stored to the variable $run_cat_pro and the variable $count_cat will go through the rows to see if the book will be found in that category. If there is nothing counted, then ‘No books found’ will be displayed. Finally, a while statement will fetch the information

that was found and stored in $run_cat_pro that will display the books based on their category.

Lastly, the function will close the statement allowing for other queries to be executed.

```php
//Function used to search the Books table by Category that user selects and display it on the page.
if(!function_exists(get_bycat)){
function get_bycat(){
  global $con;
  if(isset($_GET['category'])){
    $cat_id= $_GET['category'];
    $get_cat_pro = "SELECT `book_id`, `author`, `keywords`, `title`, `price`, `image`, `description`, `category` from `Books` WHERE `category` LIKE ?";
    if($run_cat_pro=mysqli_prepare($con,$get_cat_pro)){
      mysqli_stmt_bind_param($run_cat_pro,'s',$cat_id);
      mysqli_stmt_execute($run_cat_pro);
      mysqli_stmt_bind_result($run_cat_pro,$book_id,$author,$keywords,$title,$price,$image,$description,$category);
      mysqli_stmt_store_result($run_cat_pro);
    $count_cat = mysqli_stmt_num_rows($run_cat_pro);
    if($count_cat==0){
      echo "<h2>No books found</h2>";
    }
    while(mysqli_stmt_fetch($run_cat_pro))
  {
    echo "<div class='col-lg-4 col-md-6'>
                        <div class='card'>
                            <img class='card-img' height='200px' width='100px' src='assets/images/".$image."'>
                            <span class='content-card'>
                                <h6>".$title."</h6>
                                <h7>".$author."</h7>
                            </span>
                            <a href='index.php?add_cart=".$book_id."'><button class='buybtn btn btn-warning btn-round btn-sm'>
                  Add <i class='material-icons'>add_shopping_cart</i>
                </button></a>
                            <button class='knowbtn btn btn-warning btn-round btn-sm' data-toggle='modal' data-target='#".$book_id."'>
                  Know More
                </button>";

          //code for modal
        echo "<div class='modal fade' id='".$book_id."' tabindex='-1' role='dialog' aria-labelledby='myModalLabel' aria-hidden='true'>
                <div class='modal-dialog'>
                  <div class='modal-content'>
                    <div class='modal-header'>
                      <button type='button' class='close' data-dismiss='modal' aria-hidden='true'>&times;</button>
                      <h4 class='modal-title' id='myModalLabel'>".$title."</h4>
                    </div>
                    <div class='modal-body'>
                    <h4><p align='right'>&#36;".$price."</p></h4>".
                        $description
                    ."</div>

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

            </div>
                        </div>";    //the last two </div> are from previous echo.
```

Image 74: Code for get_bycat() function.


```php
                    <h4><p align='right'>&#36;".$price."</p></h4>".
                        $description
                    ."</div>

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

            </div>
                        </div>";    //the last two </div> are from previous echo.
```


Image 75: Code for get_bycat() function.

Going back to the webpage we will take a look at the cart tab and what happens when it is selected. Clicking on the cart tab will display the books that have been added to the cart, I have added two of them. The cart is displayed as a table with a remove column, book image column, book name column, quantity, price, and at the bottom of the table the total will be displayed (See image 76). The user can check on the checkbox (See image 77) and then press the remove button to remove the book from the cart (See image 78). When a user is satisfied with their book selection, they can check the checkout button that will take them to the payment page.

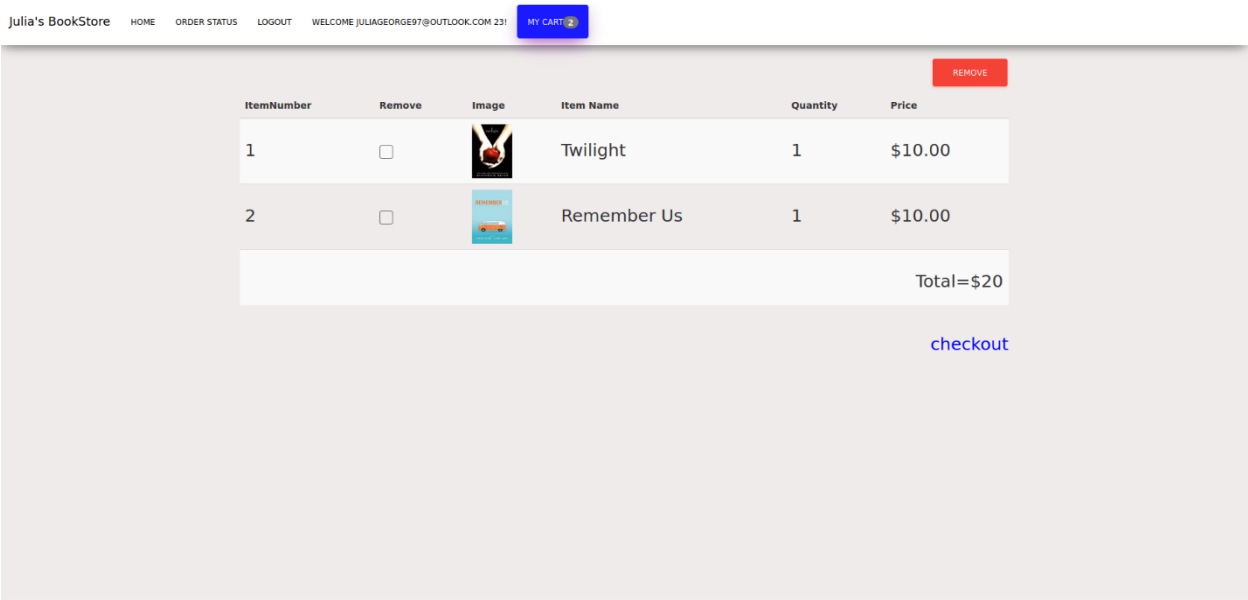


Image 76: Page displayed after clicking the cart tab.

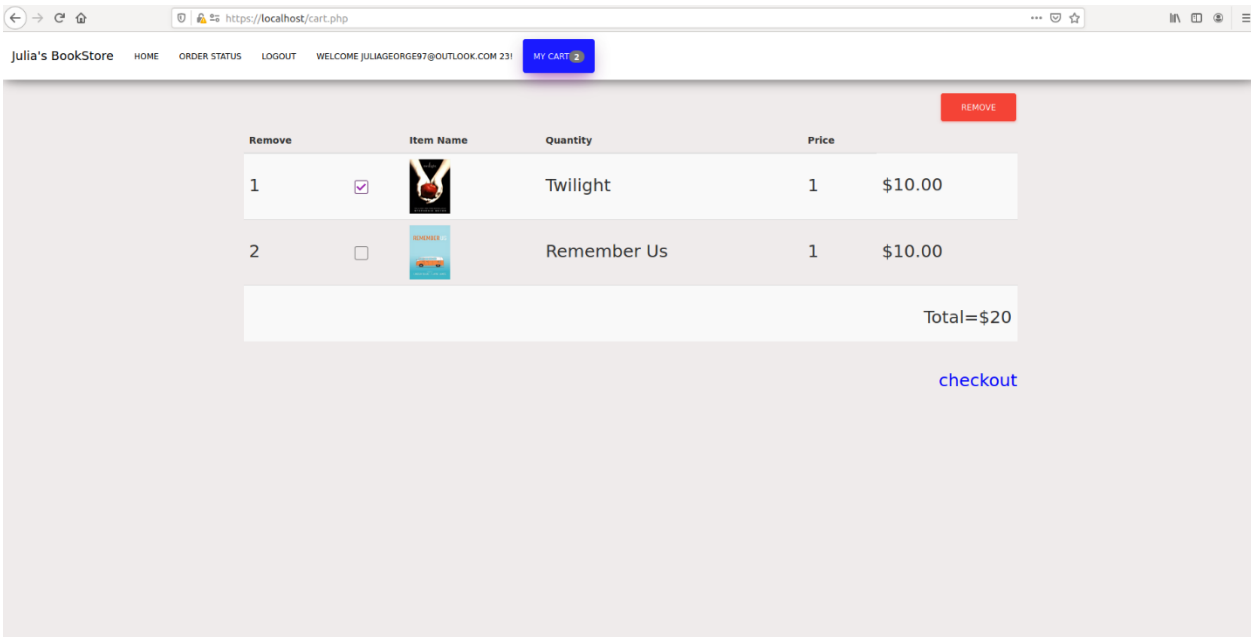


Image 77: Checking the checkbox to remove the item.

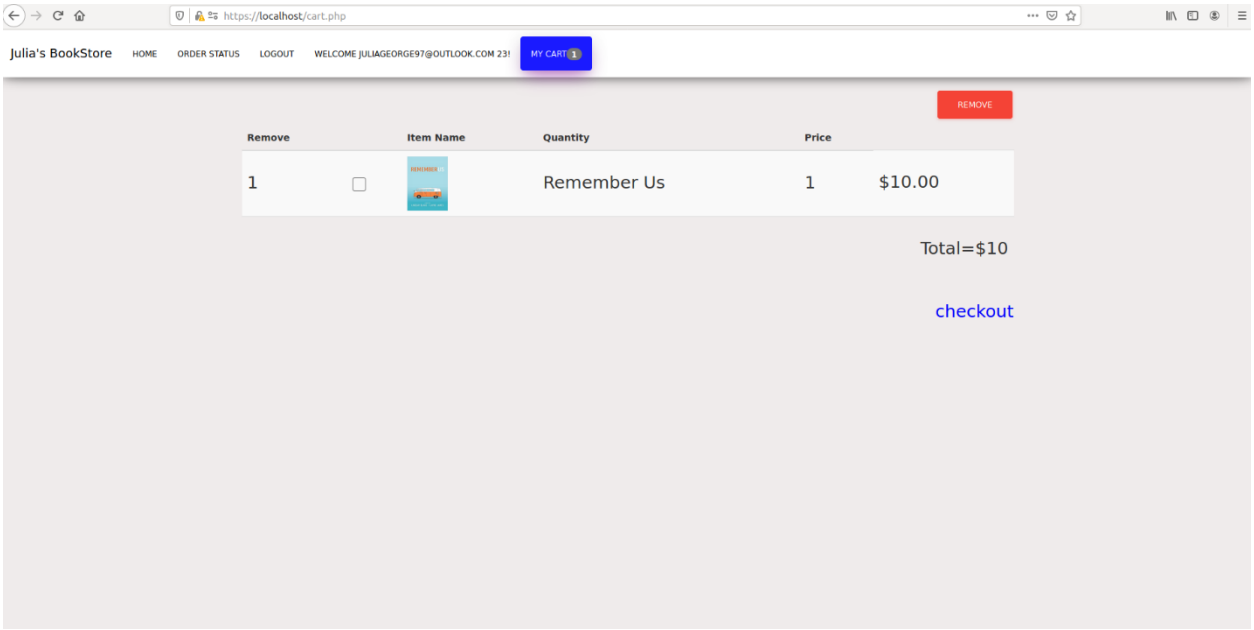
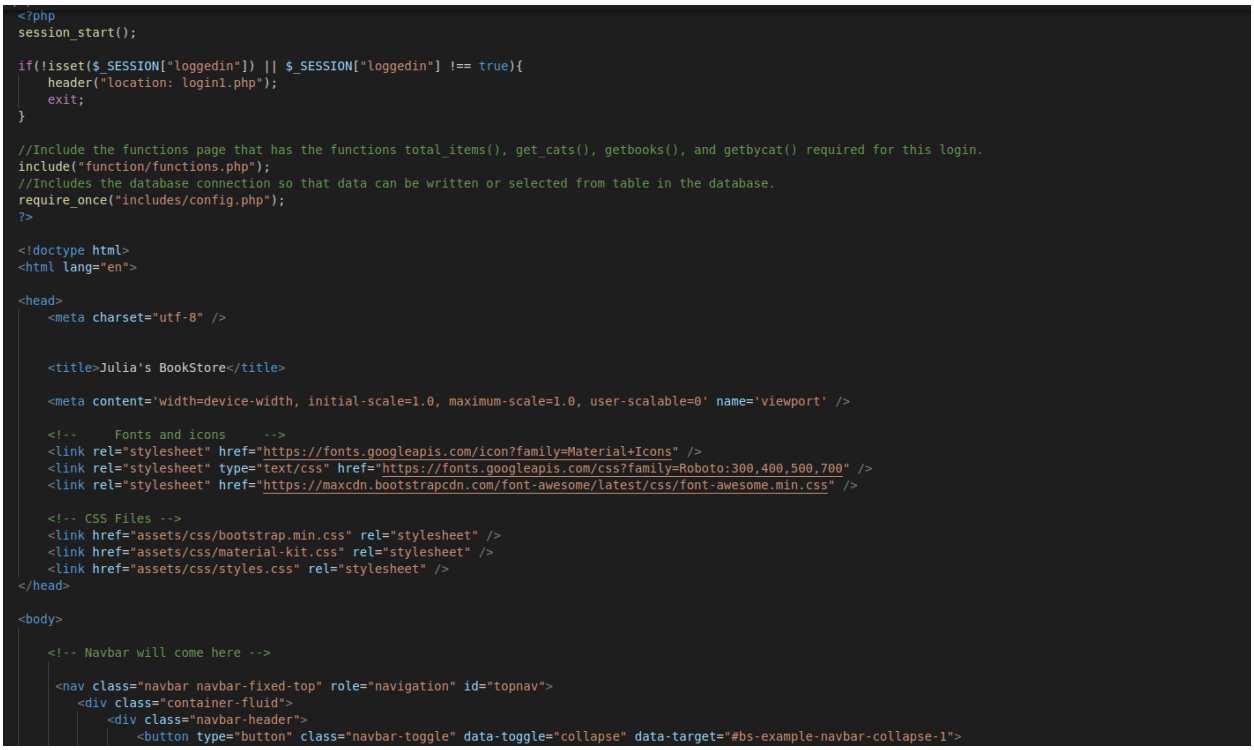


Image 78: Cart after clicking the remove button.

Now let us take a look at the code in cart.php that makes up this page and the actions that are involved in it. At the top of the code, it makes sure that a user has been logged in before they are able to access this page and it uses the functions.php page plus the config.php pages (See image 79). This page has the same navigation bar as the main page with the cart tab being highlighted in blue. The next section of the code will display the table with the function mycart() being used to get the information that was saved to the cart (See image 80). Lastly, on the cart.php file after the user presses the update cart button it will see if the user has checked any of the check boxes. If they are checked then the books checked will be deleted based on the bookid to remove and the customers id. The variable $run_delete will involve connecting to the database to run the query that was assigned to $delete_books. If the query is run, then the user will be redirected back to the cart.php page (See image 81). Finally, the checkout button will redirect the users to the checkout.php page (See image 81).



```php
<?php
session_start();

if(!isset($_SESSION["loggedin"]) || $_SESSION["loggedin"] !== true){
    header("location: login1.php");
    exit;
}

//Include the functions page that has the functions total_items(), get_cats(), getbooks(), and getbycat() required for this login.
include("function/functions.php");
//Includes the database connection so that data can be written or selected from table in the database.
require_once("includes/config.php");
?>

<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8" />


    <title>Julia's BookStore</title>

    <meta content='width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0' name='viewport' />

    <!--     Fonts and icons     -->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons" />
    <link rel="stylesheet" type="text/css" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700" />
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/latest/css/font-awesome.min.css" />

    <!-- CSS Files -->
    <link href="assets/css/bootstrap.min.css" rel="stylesheet" />
    <link href="assets/css/material-kit.css" rel="stylesheet" />
    <link href="assets/css/styles.css" rel="stylesheet" />
</head>

<body>

    <!-- Navbar will come here -->

    <nav class="navbar navbar-fixed-top" role="navigation" id="topnav">
       <div class="container-fluid">
           <div class="navbar-header">
               <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">
```

Image 79: The cart.php page, the start of the code.

```
<!-- This will display the items that are in the cart using the mycart() function. Including the name of the item, the quantity, the price,
        and a place to remove the item. -->
<div class="container">
    <table class="table-striped table">
        <thead class="thead-inverse">
            <tr>
                <th>ItemNumber</th>
                <th>Remove</th>
                <th>Image</th>
                <th>Item Name</th>
                <th>Quantity</th>
                <th>Price</th>
            </tr>
        </thead>
        <tbody>
            <form action="cart.php" method="post">
            <?php mycart(); ?>


    <!-- This will display a button that when clicked will remove the selected item by using the PHP process below. -->
    <div align="right">
        <button name="update_cart" type="submit" class="btn btn-danger">Remove</button>
    </div>
            </form>
</tbody>

    </table>
```

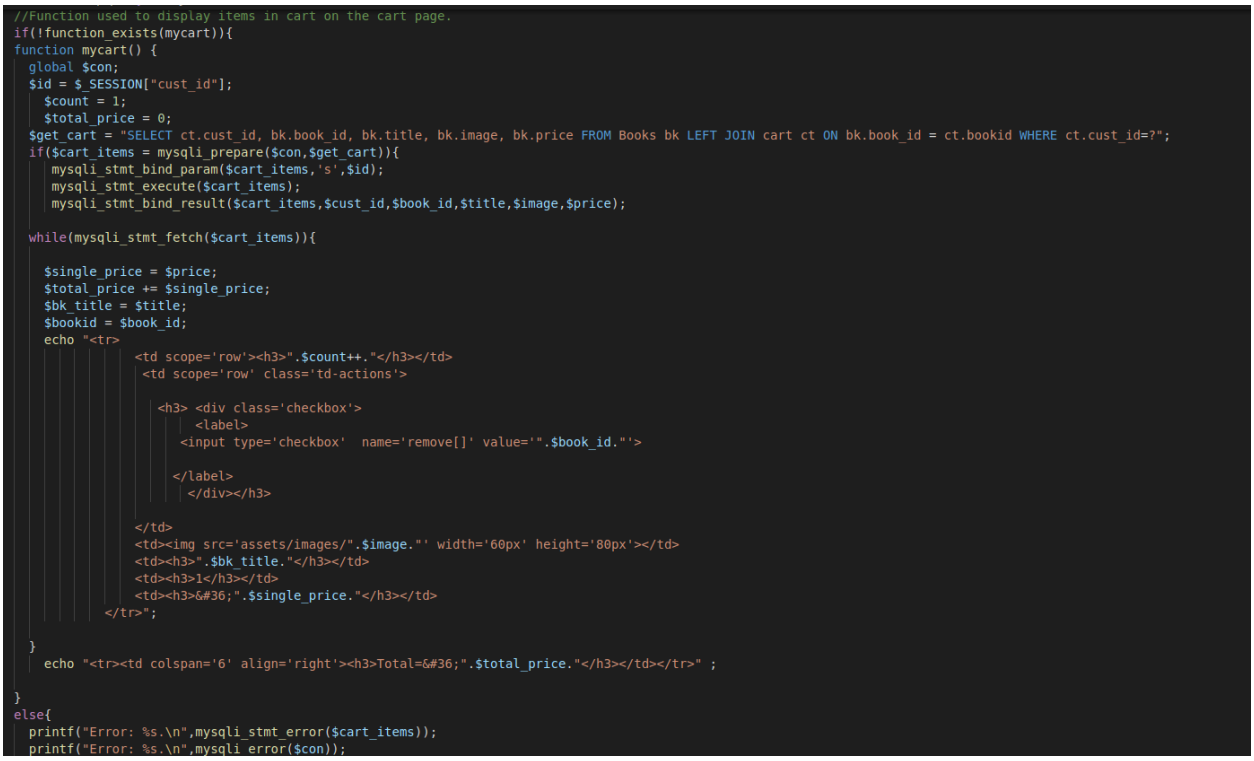
Image 80: Code of cart.php file that displays the cart table and the function used to display the items in the cart.

```
    <?php
    //Set session id cust_id to variable $id.
    $id = $_SESSION['cust_id'];
    //Once update_cart is posted
    if(isset($_POST['update_cart']))
   {
       //Once remove button is clicked the book will be removed from the list.
       foreach($_POST['remove'] as $remove_id){
           $delete_books = "delete from cart where bookid = '$remove_id' AND cust_id = '$id'";
           //To run the query to delete the book.
          $run_delete = mysqli_query($con, $delete_books);
              //When the query is run the page will update with the removed book gone.
              if($run_delete){
                  echo "<script>window.open('cart.php','_self');</script>";
              }
       }
   }
        ?>

    <div class="container" align="right" ><h3> <a  style="text-decoration:none " href="checkout.php">checkout </a></h3>
          </div>

</body>

<!--   Core JS Files   -->
<script src="assets/js/jquery.min.js" type="text/javascript"></script>
<script src="assets/js/bootstrap.min.js" type="text/javascript"></script>
<script src="assets/js/material.min.js"></script>

<!--  Plugin for the Sliders, full documentation here: http://refreshless.com/nouislider/ -->
<script src="assets/js/nouislider.min.js" type="text/javascript"></script>

<!--  Plugin for the Datepicker, full documentation here: http://www.eyecon.ro/bootstrap-datepicker/ -->
<script src="assets/js/bootstrap-datepicker.js" type="text/javascript"></script>

<!-- Control Center for Material Kit: activating the ripples, parallax effects, scripts from the example pages etc -->
<script src="assets/js/material-kit.js" type="text/javascript"></script>

</html>
```

Image 81: Code for the cart.php page that involves removing books from the cart and the checkout link.

Let us take a look at the one function that was used in this section of code, the mycart() function. This function is used to display items in the cart on the cart page (See image 82). Like every other function it uses the global variable $con to connect to the database. We declare the variable $id to be assigned to the cust_id that we established after the user logged into the website. The other variables declared are $count, which is equal to one, and $total_price, which is equal to 0. Next it assigns the variable $get_cart to the SELECT statement that joins the books table and the cart table. It selects cust_id and bookid from the cart table than selects book_id, title, image, and price from the books table. It left joins on the book_id from the books table and the bookid from the carts table. Then it selects where the cust_id is equal to an unknown variable we get from the user. It then performs an if statement declaring $cart_items that prepares for connection to the database to execute the SELECT statement. The unknown variable is bound to make sure that only one variable is gotten from the input to be used for the $id variable that was declared. The statement is then executed, and the results are bound to their corresponding variables.

A while statement is used to fetch the information from the $cart_items variable. A variable $single_price is declared which is equal to the price of the book that we got from the database. The $total_price variable is declared that equals the total plus the single price of a book. The $bk_title variable is equal to the title of the book that we got from the database and the $bookid is equal to the $book_id variable that we got from the database. Next the while statement will display the count of books which we saw in the cart page, it increments when a new book is added. Then it has the checkbox that uses the $book_id variable for when the user clicks to remove. Next is the image, then the book title, the single price, and lastly it will display the total price of the books. If the if statement can not execute then errors will occur.



```
//Function used to display items in cart on the cart page.
if(!function_exists(mycart)){
function mycart() {
  global $con;
  $id = $_SESSION["cust_id"];
    $count = 1;
    $total_price = 0;
  $get_cart = "SELECT ct.cust_id, bk.book_id, bk.title, bk.image, bk.price FROM Books bk LEFT JOIN cart ct ON bk.book_id = ct.bookid WHERE ct.cust_id=?";
  if($cart_items = mysqli_prepare($con,$get_cart)){
     mysqli_stmt_bind_param($cart_items,'s',$id);
     mysqli_stmt_execute($cart_items);
     mysqli_stmt_bind_result($cart_items,$cust_id,$book_id,$title,$image,$price);

  while(mysqli_stmt_fetch($cart_items)){

    $single_price = $price;
    $total_price += $single_price;
    $bk_title = $title;
    $bookid = $book_id;
    echo "<tr>
                <td scope='row'><h3>".$count++."</h3></td>
                 <td scope='row' class='td-actions'>

                  <h3> <div class='checkbox'>
                       <label>
                      <input type='checkbox'  name='remove[]' value='".$book_id."'>

                      </label>
                       </div></h3>

                </td>
                <td><img src='assets/images/".$image."' width='60px' height='80px'></td>
                <td><h3>".$bk_title."</h3></td>
                <td><h3>1</h3></td>
                <td><h3>&#36;".$single_price."</h3></td>
              </tr>";

  }
    echo "<tr><td colspan='6' align='right'><h3>Total=&#36;".$total_price."</h3></td></tr>" ;

}
else{
  printf("Error: %s.\n",mysqli_stmt_error($cart_items));
  printf("Error: %s.\n",mysqli_error($con));
```

Image 82: The code for the mycart() function.

When a user is ready to check out, they can click the checkout button where they will be redirected to a payment page (See image 83). On the checkout page the user will be asked to put in their billing information and payment information. The billing information ask for the address, city, state, and zip. The payment information asks for the name on the card, credit card number, expiration month, and expiration year, and the CVV. If the user does not enter any information, then there will be errors that pop up to specify it can not be left blank (See image 84). If the user tries to insert a SQL inject errors will pop up (See image 85 and 86). When they click submit with the errors, they will be redirected back to the payment page and have to insert information without errors. This also involves making sure that the user cannot insert SQL injections into the web browser (See image 87). In addition to SQL injections the site also protects against XSS injections (See image 88) and when trying to click continue to checkout the page will be

redirected to a new payment page where the user will have to reinsert information. The user can not insert an XSS injection into the web browser either (See image 89).

The individual fields also have specific rules they must meet and if they do not meet those then an error will be given. The city can only contain letter and if it has numbers an error will occur (See image 90). The state can only consist of letters and can only have two (See image 91 & 92). The zip has to be at minimum 5 numbers, a dash with four numbers, and only numbers (See image 93, 94, & 95). The name on the card has to be only letters and a maximum of 50 and minimum of 6 (See image 96 & 97). The card number can not be more then 16 characters or less then 16 characters (See image 98 & 99). The month has to be a number 1-12 and cannot contain a letter (See image 100 & 101). The expiration year has to be a valid year, cannot contain more than two characters, and can not contain letters (See image 102, 103, 104). Lastly, the CVV can not contain less than 3 characters, more than 3 characters, or contain letters (See image 105, 106, 107).

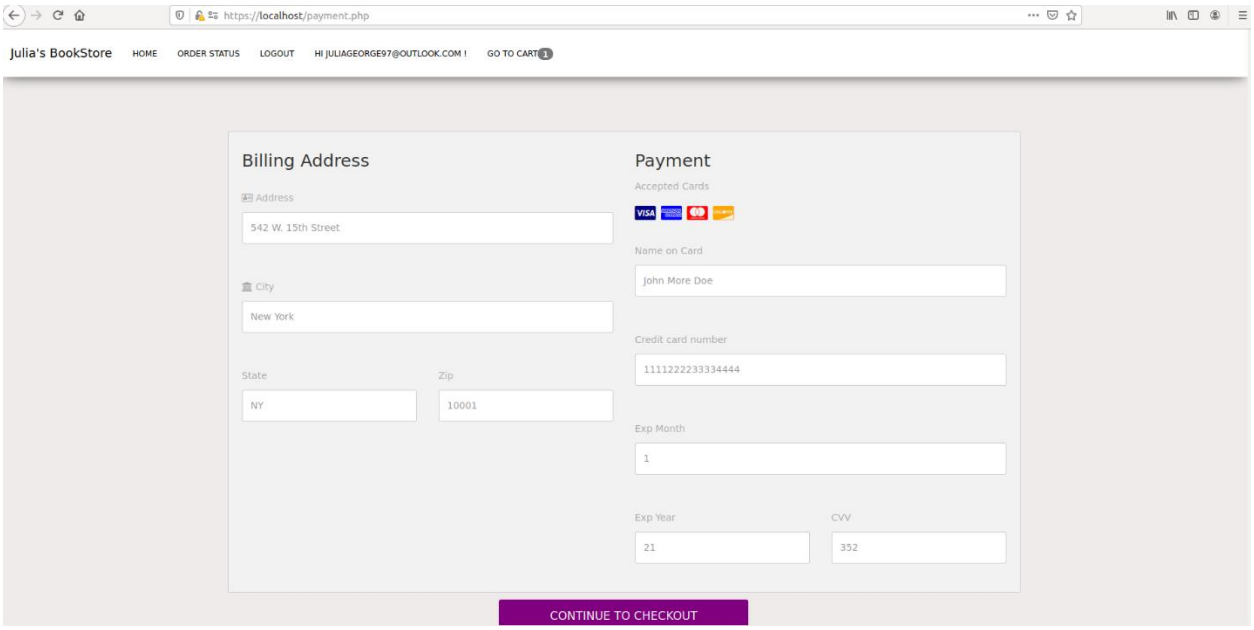


Image 83: Page displayed after clicking the checkout link.

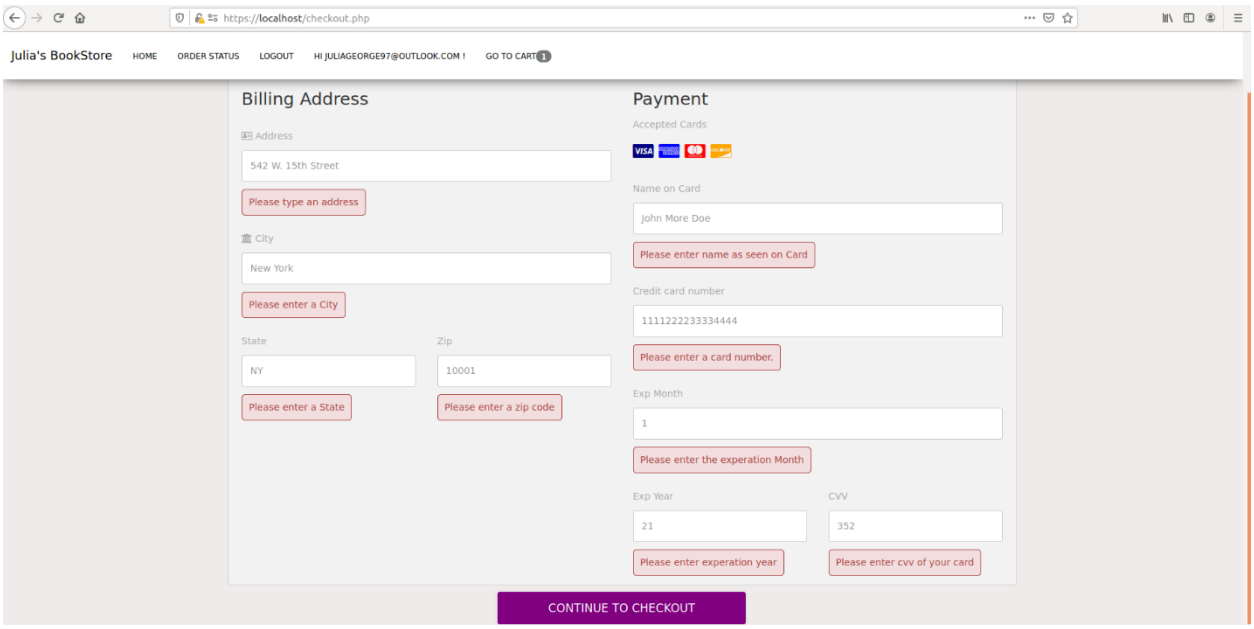


Image 84: Errors if all the fields are left blank.

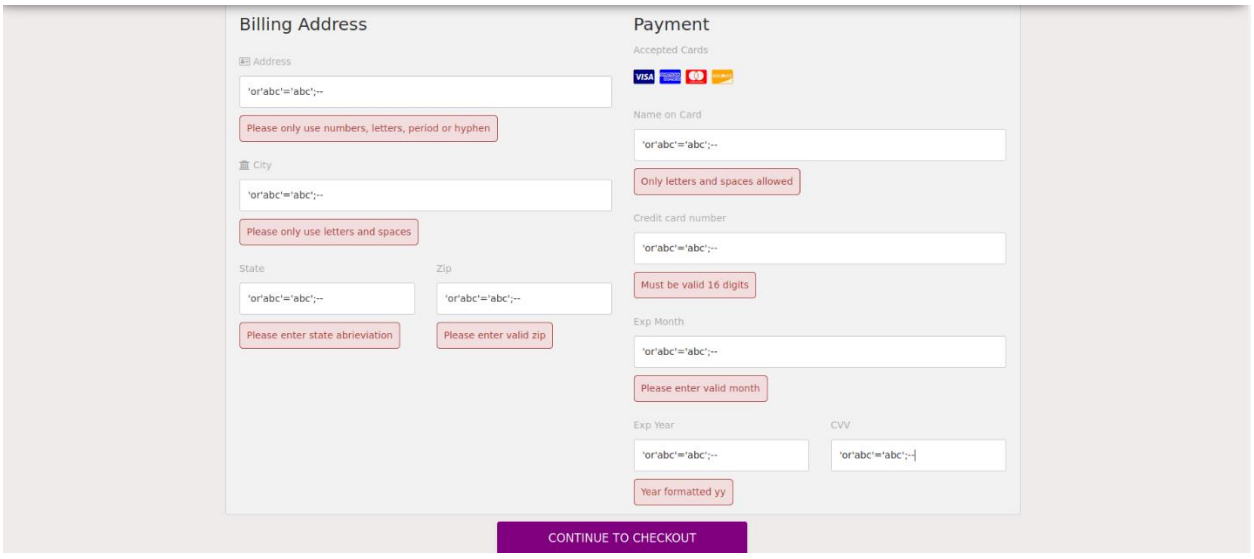


Image 85: Errors displayed if a SQL injection is typed in the text fields.

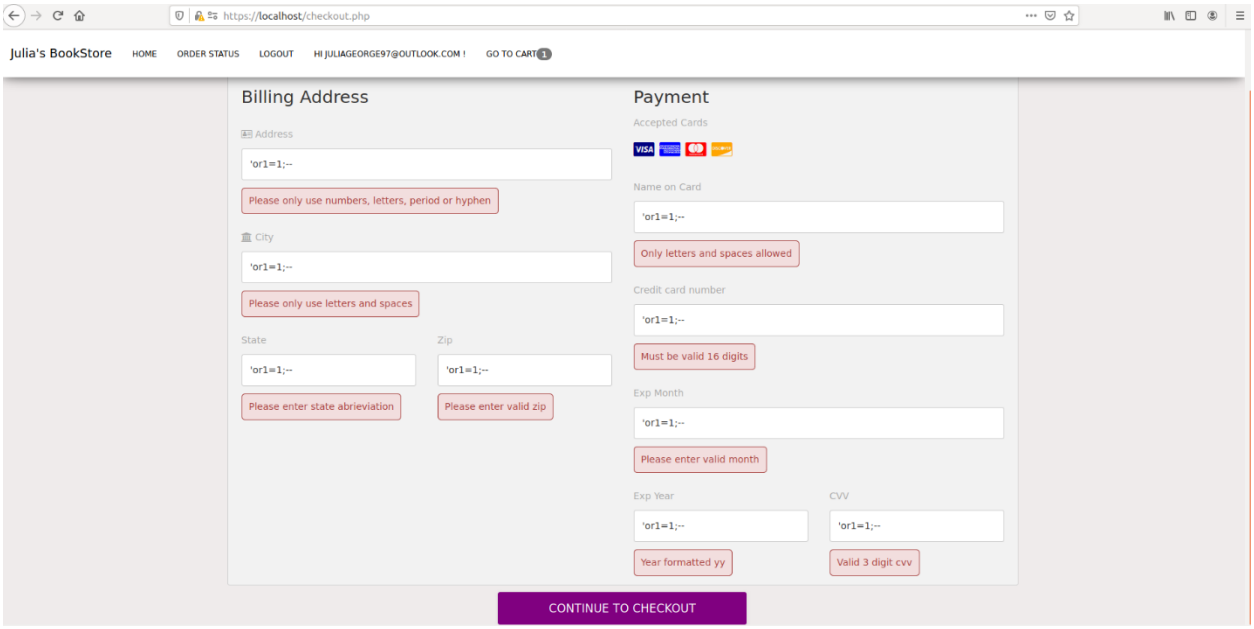


Image 86: Errors displayed if a SQL injection is typed in the text fields.

Image 87: Results after typing SQL injection into the web browser.

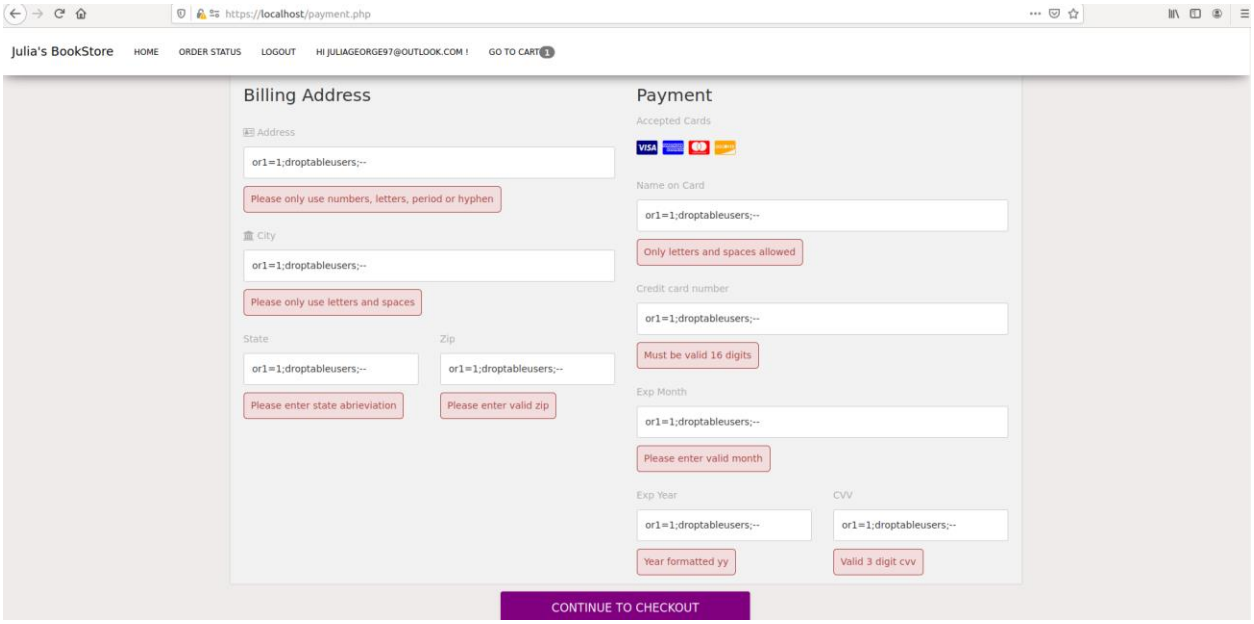


Image 88: XSS injection errors displayed when typed in fields.

Image 89: Display after typing XSS injection into the web browser.

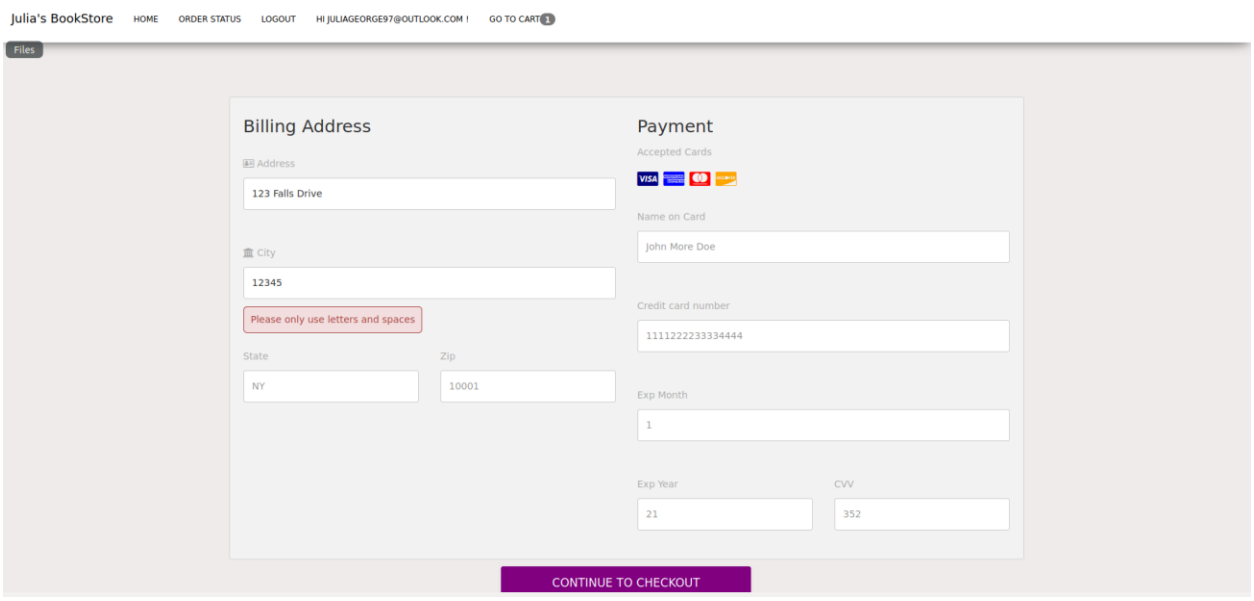


Image 90: Error if city text field has numbers.

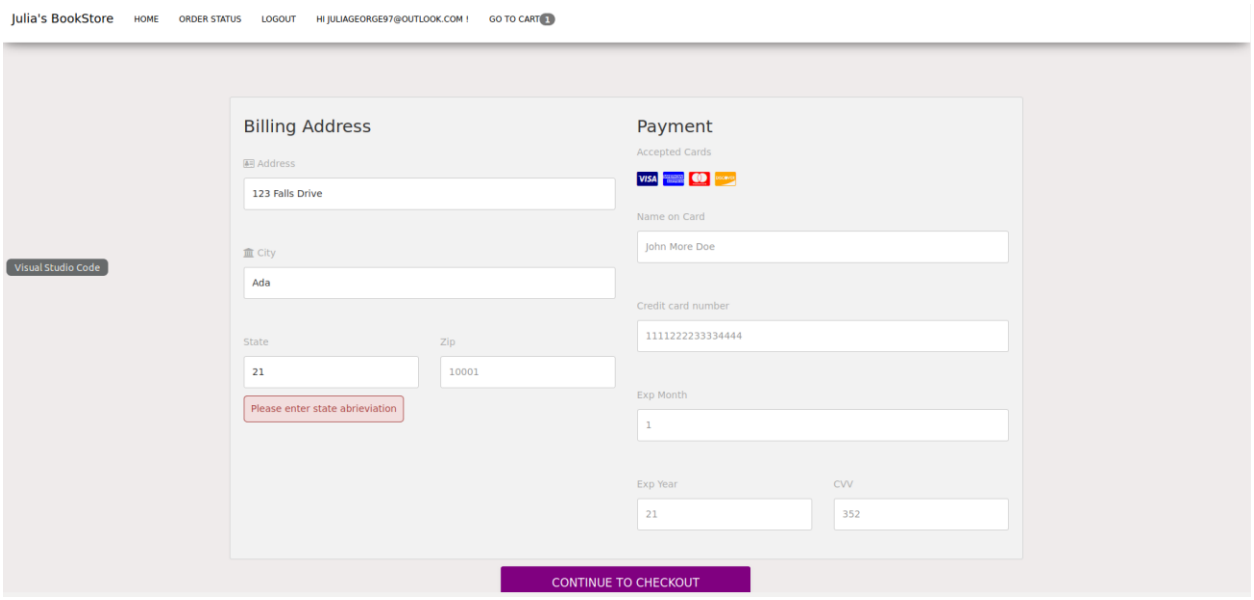


Image 91: Error is state text field contains numbers.

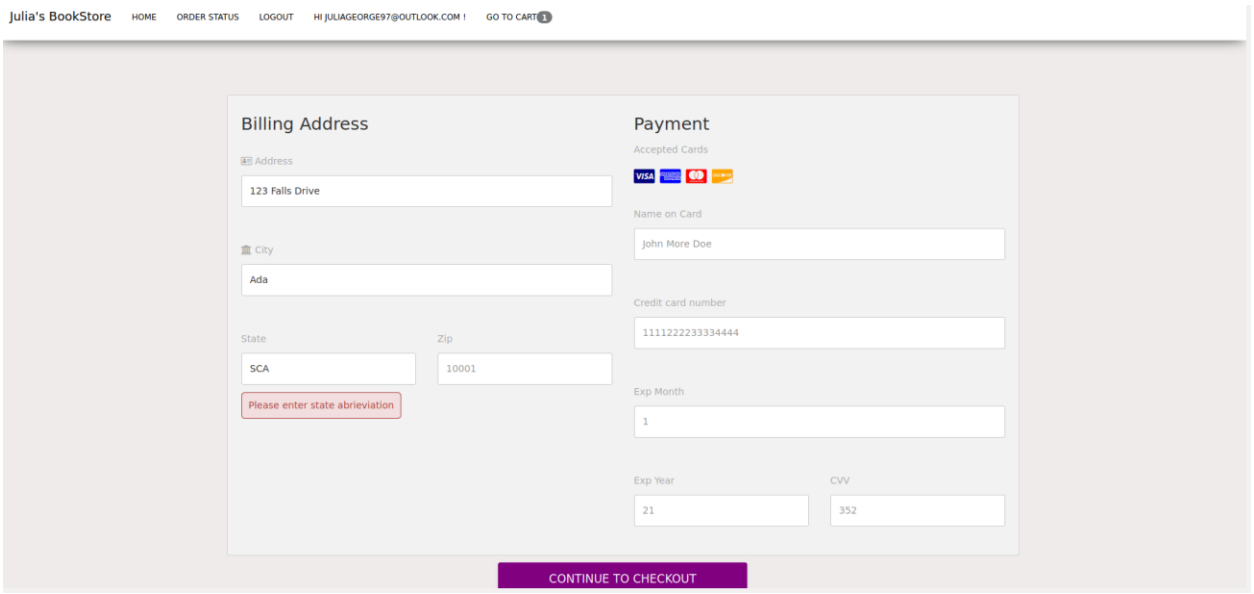


Image 92: Error if state text field contains more than 2 letters.

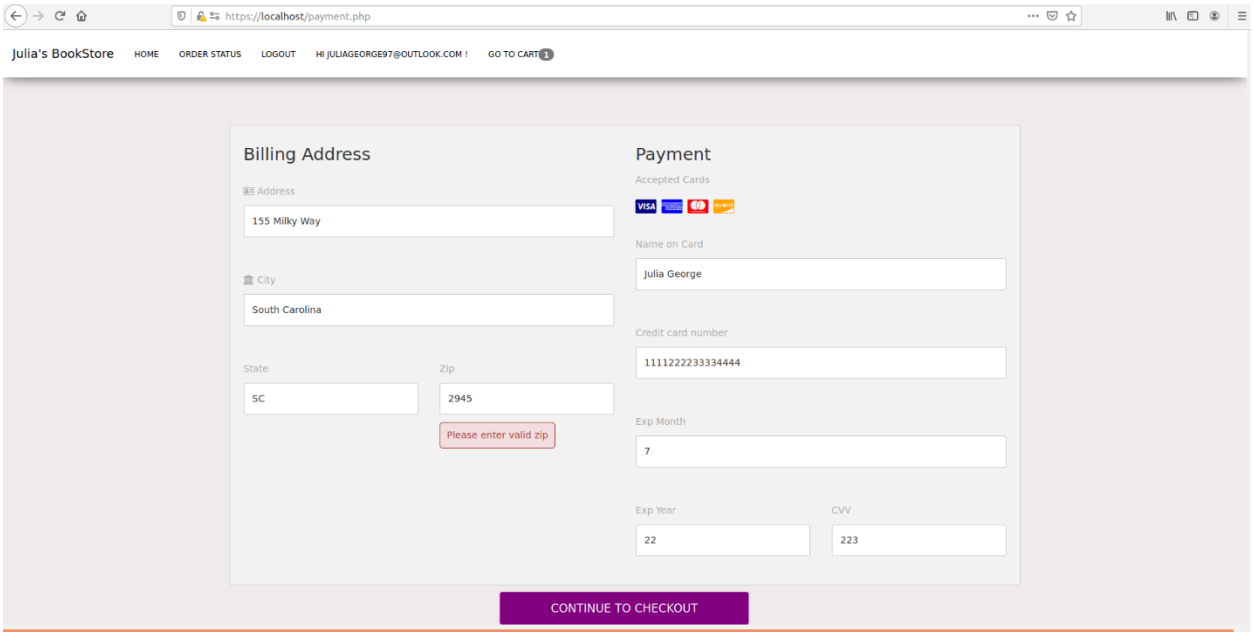


Image 93: Error if zip text field contains less than 5 numbers.

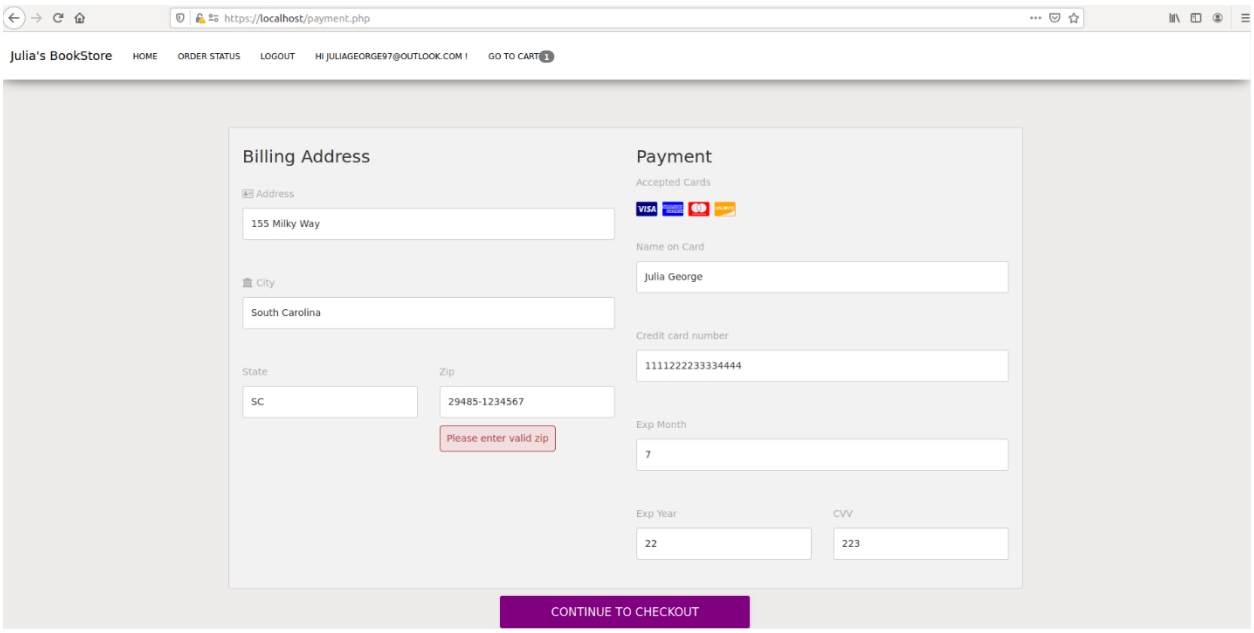


Image 94: Error if zip text field contains more than 4 numbers after the dash.

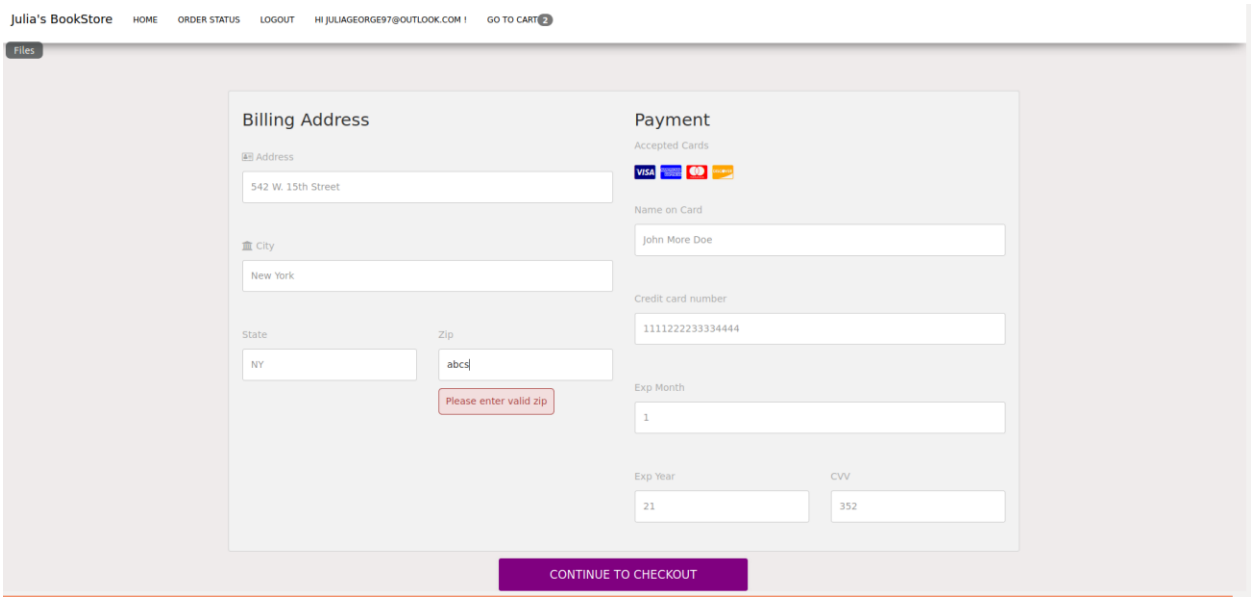


Image 95: Error if zip text field contains letters.

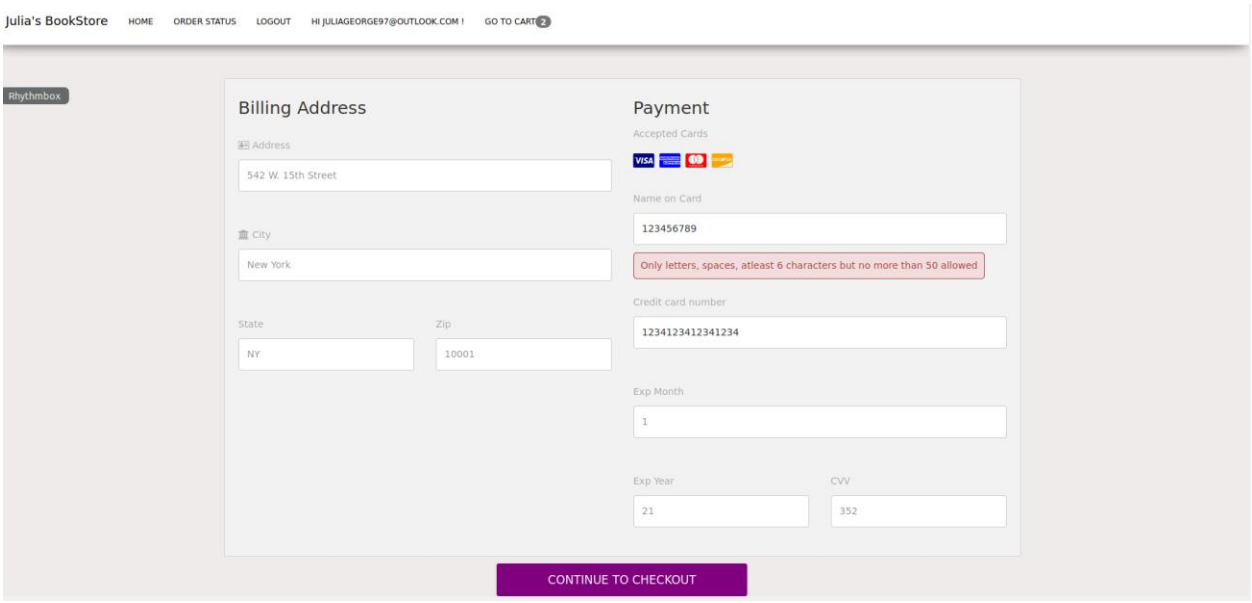


Image 96: Error if name on card text field contains numbers.

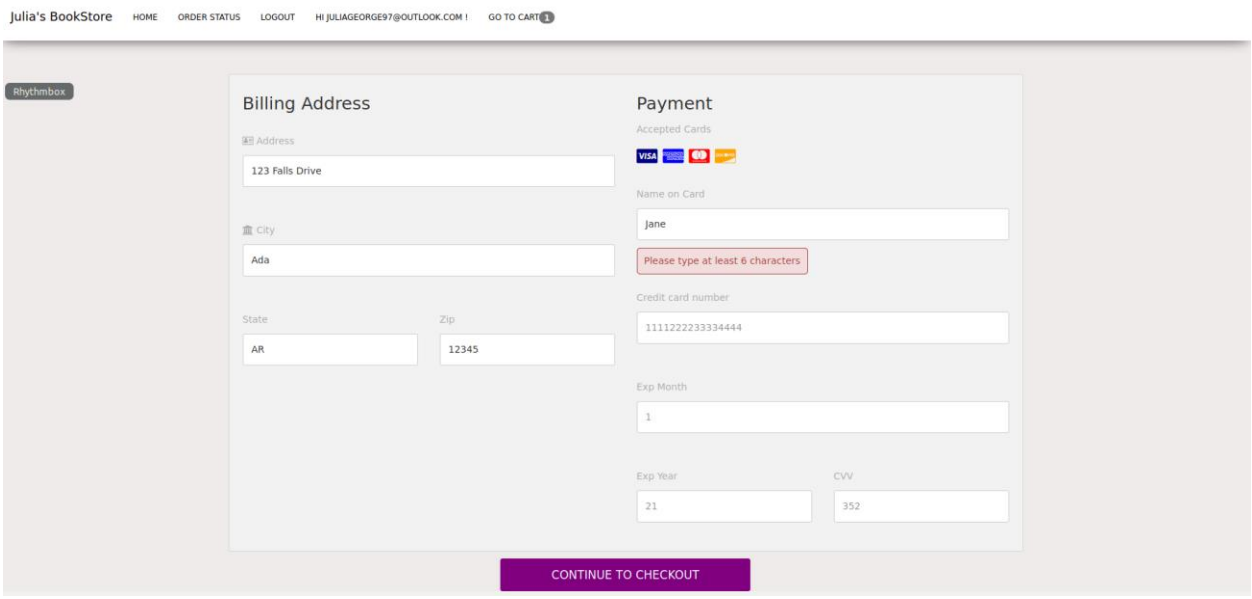


Image 97: Error if name on card text field contains less than 6 characters.

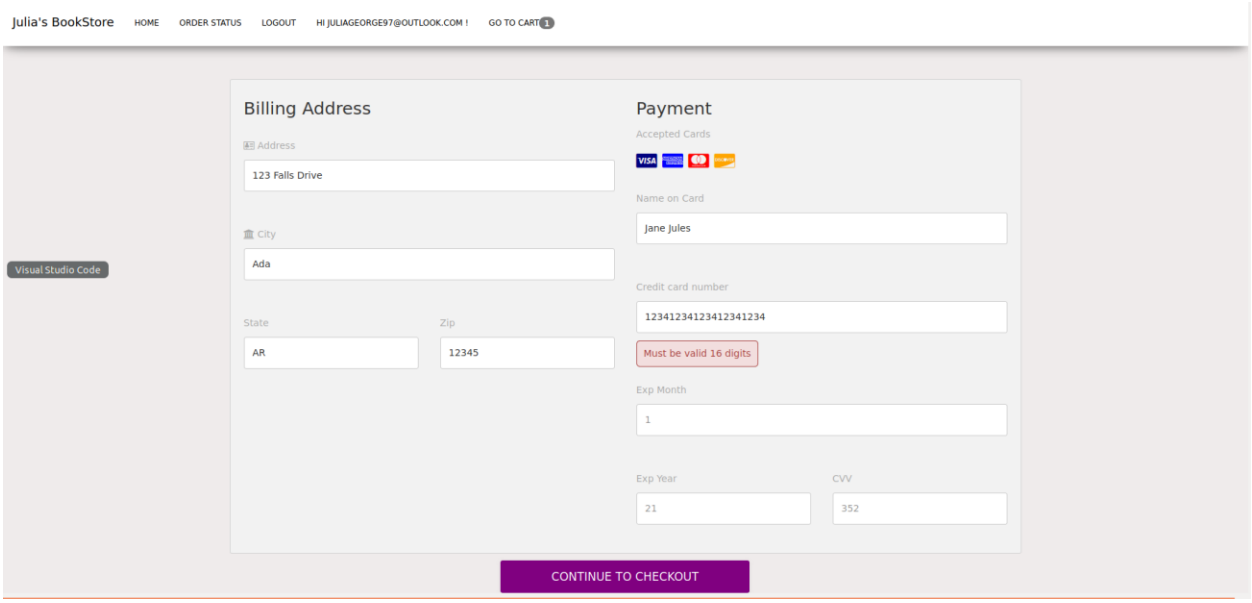


Image 98: Error if credit card number text field contains more than 16 digits.

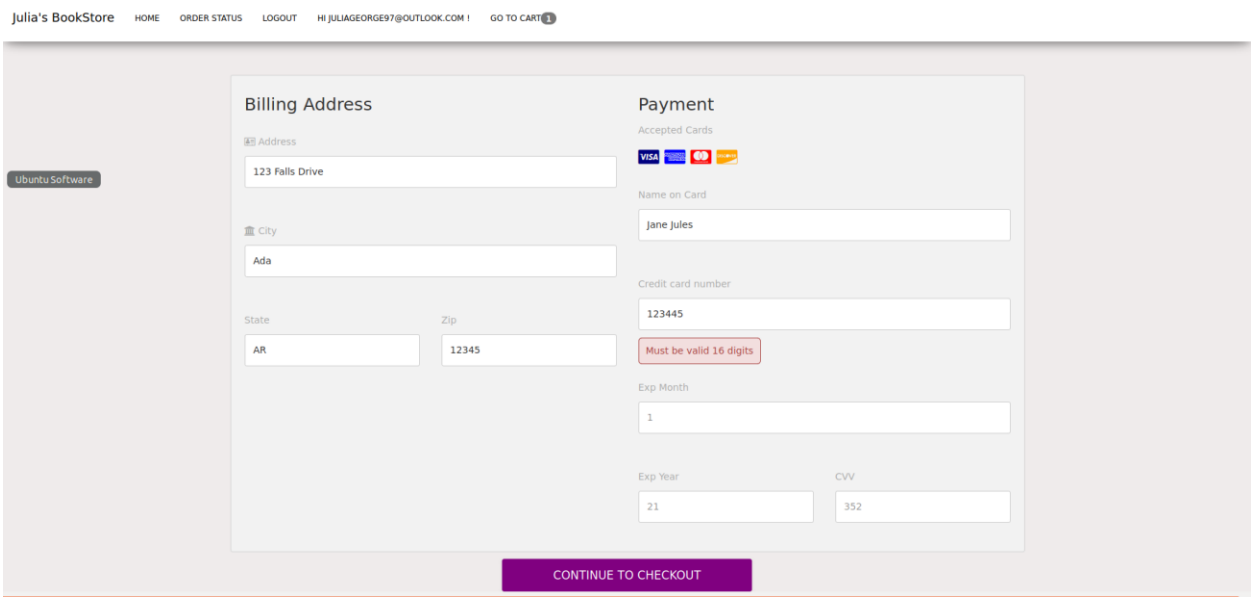


Image 99: Error if credit card number text field contains less than 16 digits.

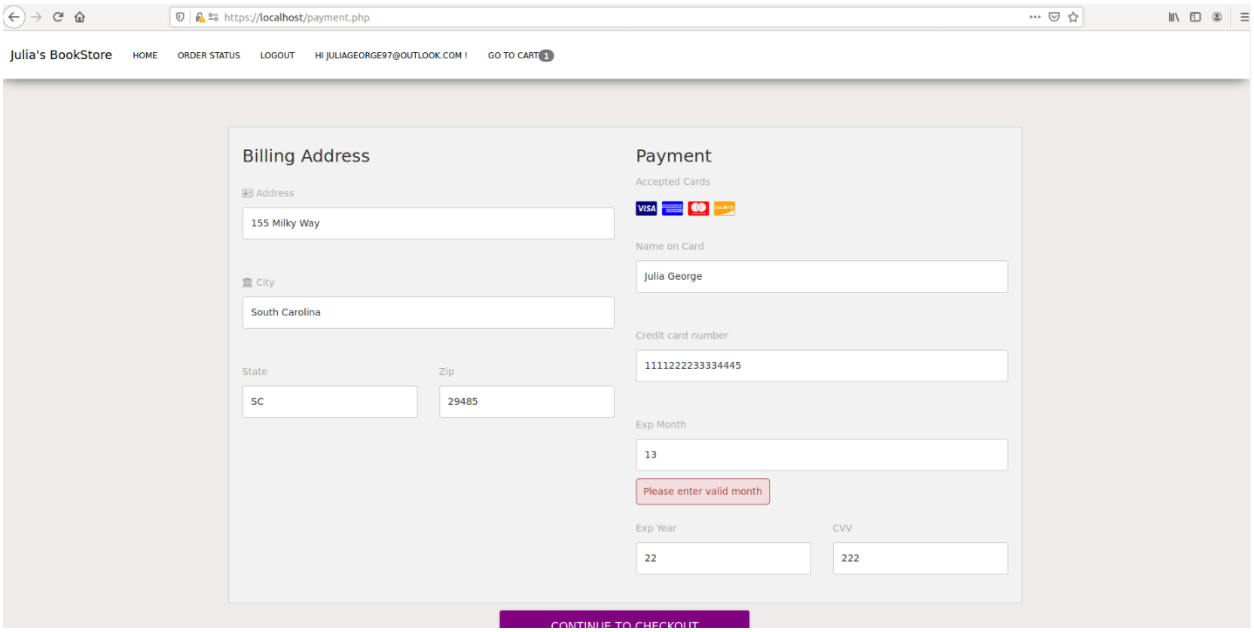


Image 100: Error if exp month text field does not contain number from 1-12.

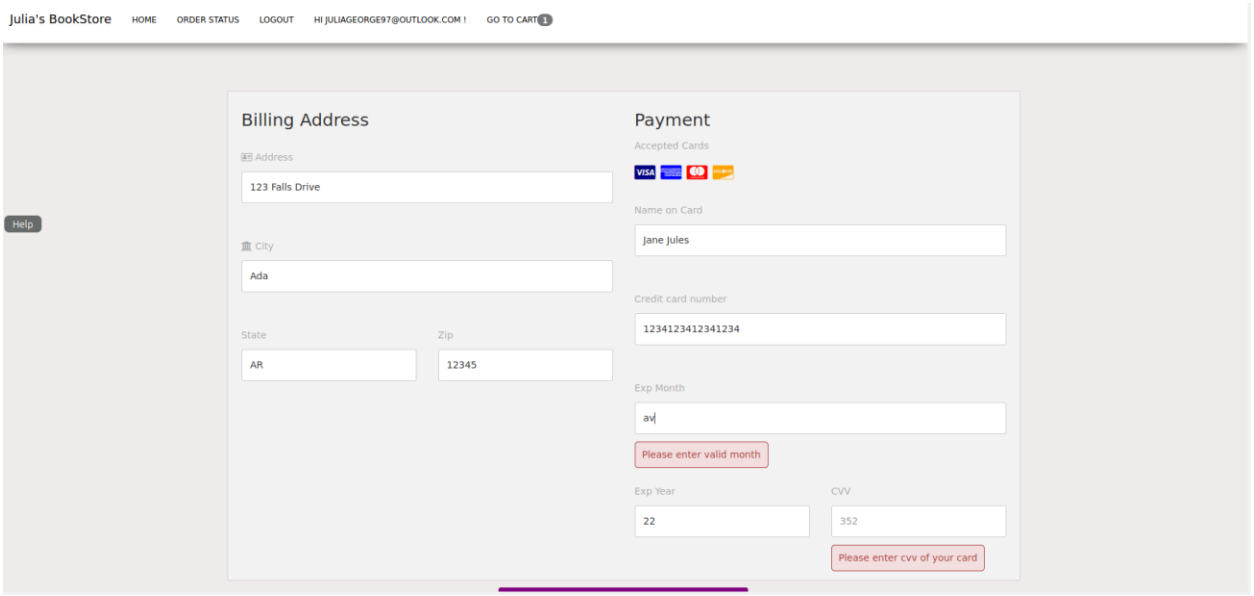


Image 101: Error if exp month text field contain letters.

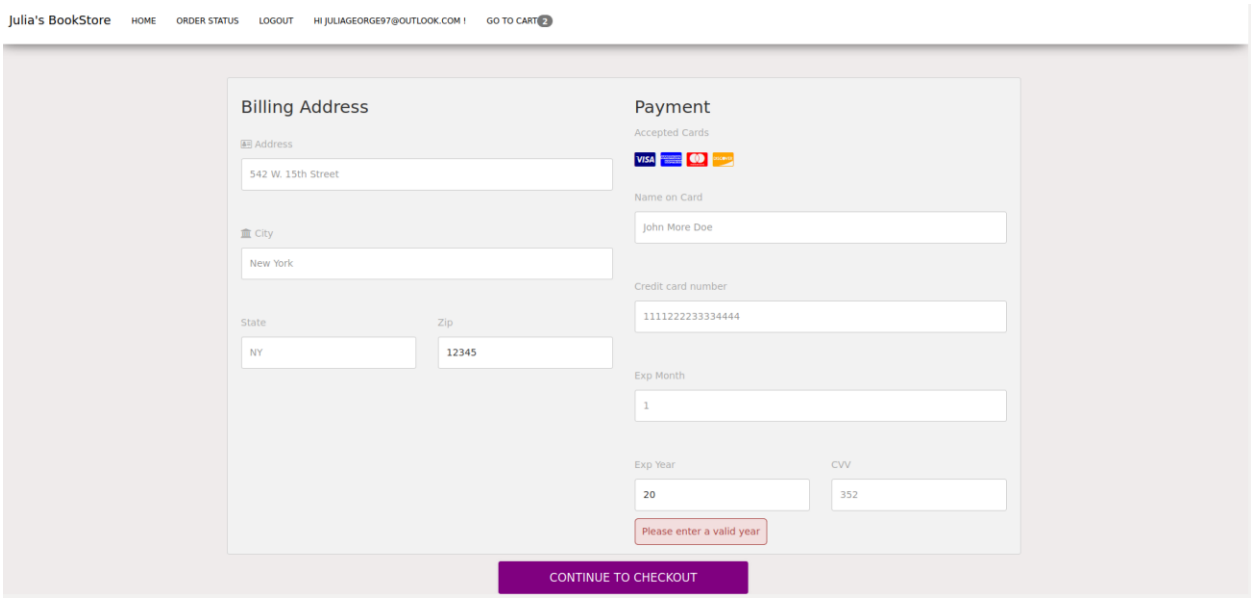


Image 102: Error if exp year text field contains a number past current year.

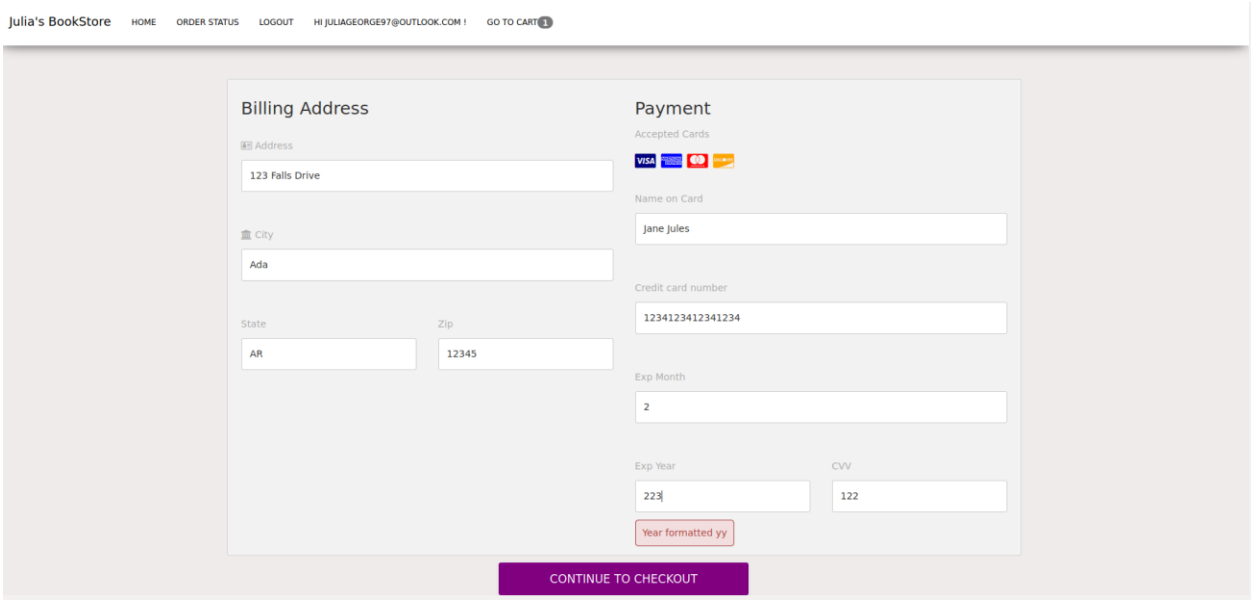


Image 103: Error if exp year text field contains more than 3 digits.

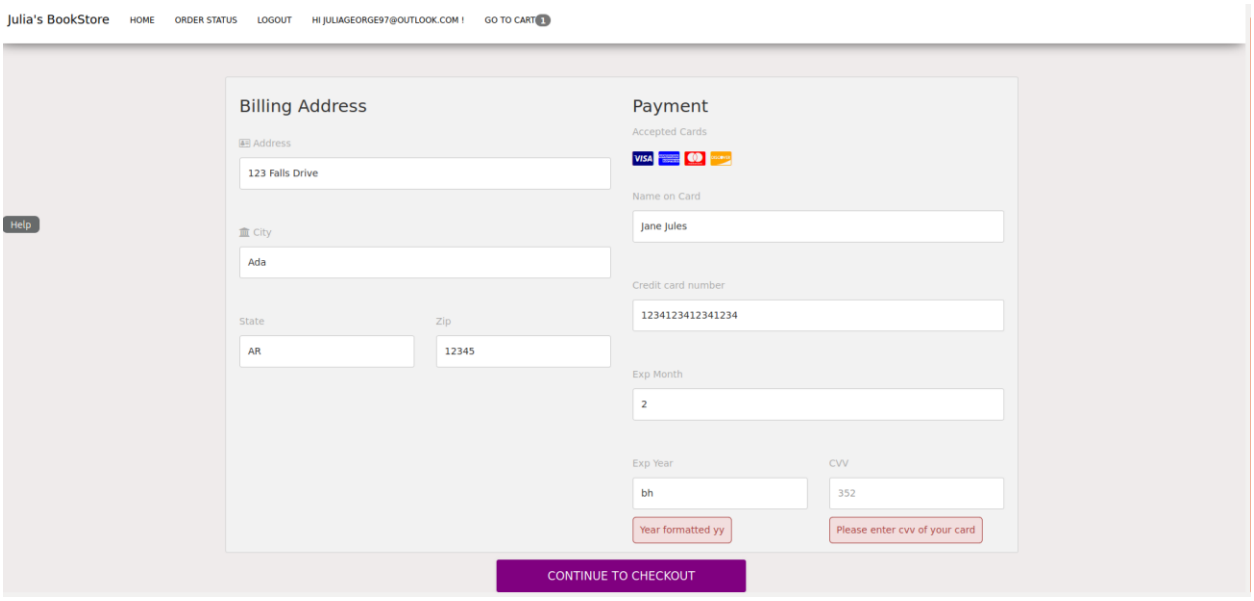


Image 104: Error if exp year text field contains letters.

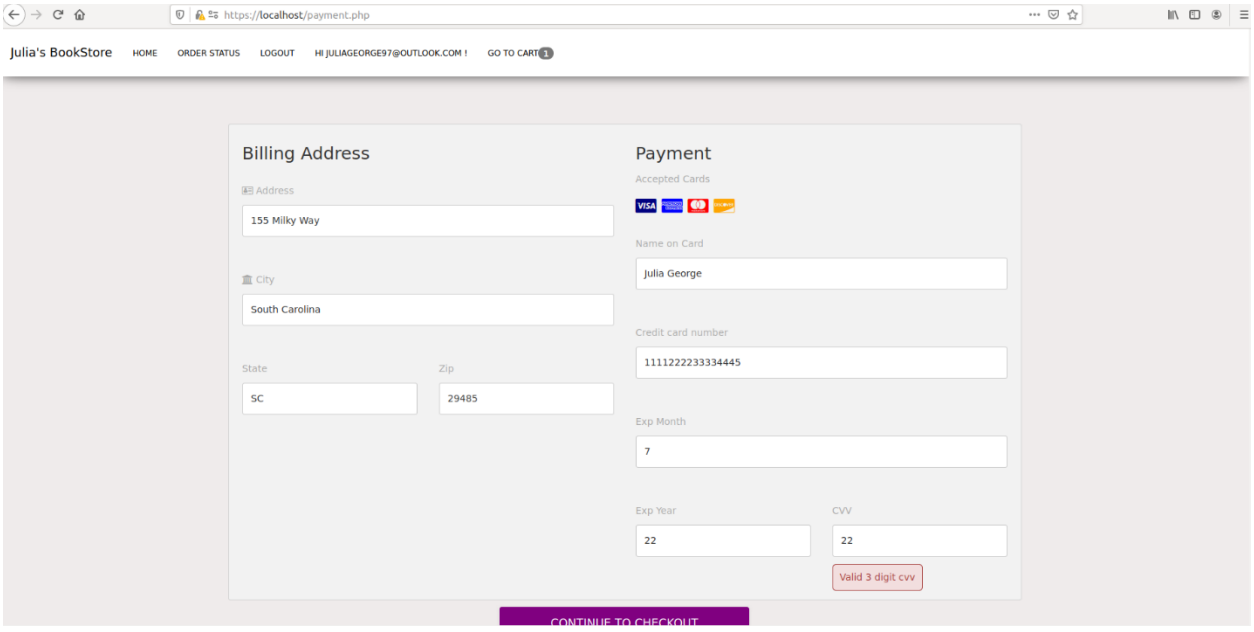


Image 105: Error if cvv text field contains less than 3 digits.

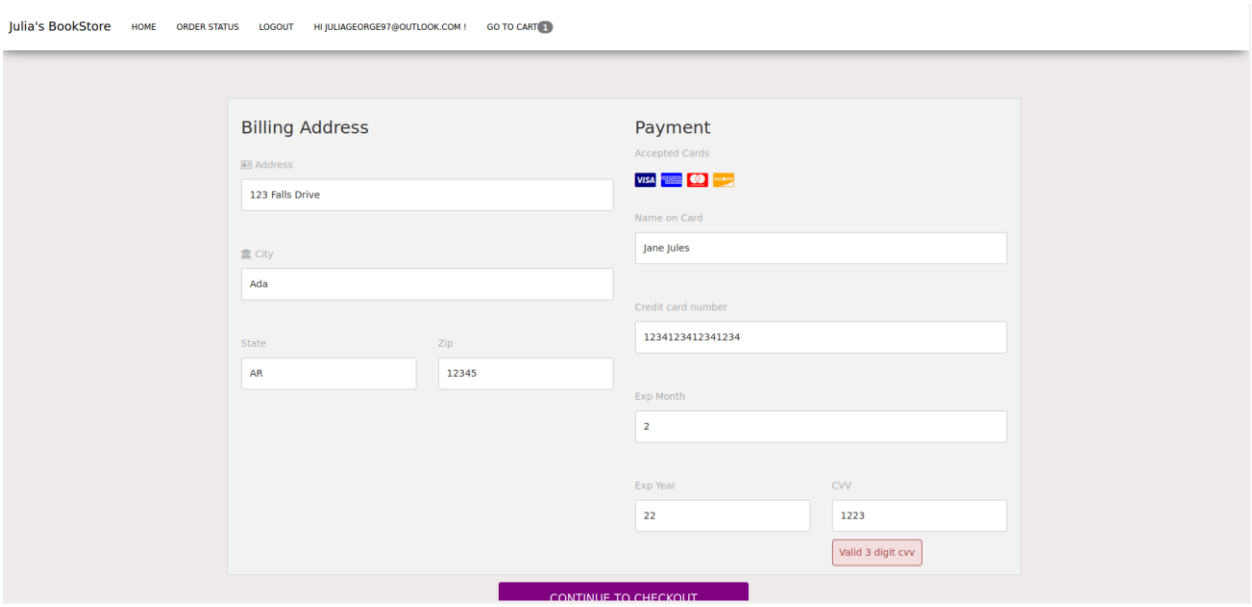


Image 106: Error if cvv text field contains more than 3 digits.

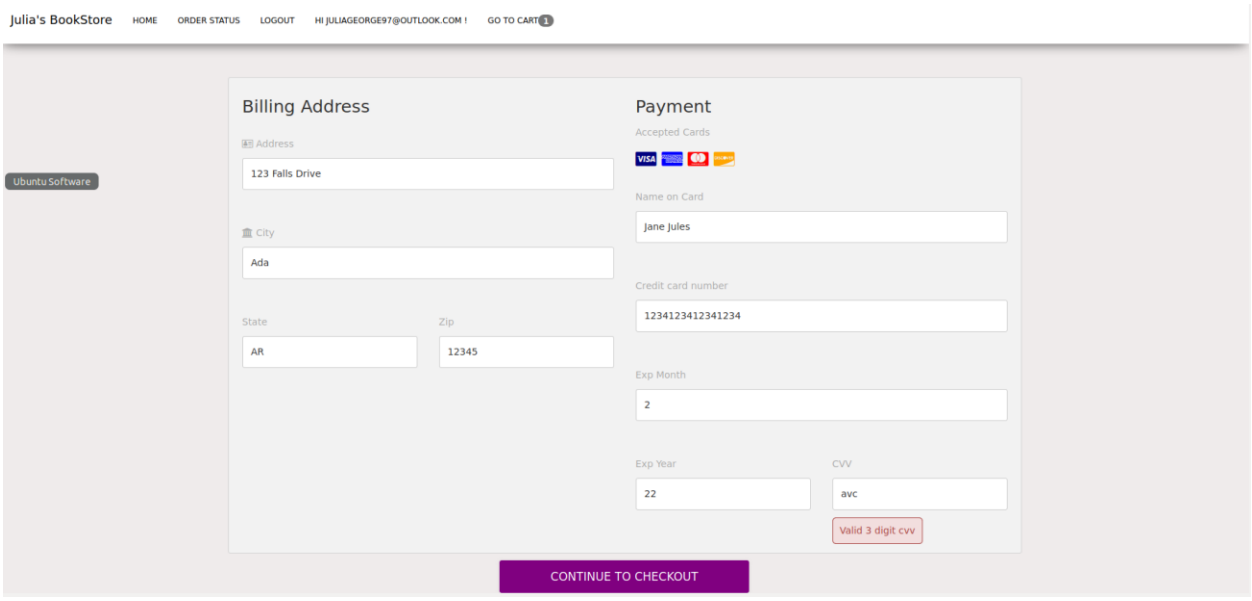


Image 107: Error if cvv text field contains letters.

The payment.js file is used for client-side validation of the payment page it creates errors before the user submits the form, making sure they do not insert XSS or SQL injections. First the code checks the billing information checking the address, making sure it is not empty and makes sure that the user only uses letters, numbers, periods, or hyphens (See image 108). Next it checks city making sure it is not empty and only uses letters and spaces (See image 109). After that it checks the state making sure that it is not empty, there are only two characters, and no numbers are present (See image 110). Lastly, for the billing information it checks the zip code making sure that it is not empty, contains 5 numbers or contains 5 numbers then a hyphen with 4 numbers, and only contains numbers (See image 111).

The second part it checks input validation for the credit card information making sure those inputs do not contain any SQL injections or XSS injections. First it checks the card name making sure that is not empty, contains only letters and spaces, and that it is a minimum of 6 characters but no longer than 50 characters (See image 112). The next thing checked is the card number making sure it contains 16 numbers and it is not blank (See image 113). Then it checks the expiration month making sure it is not blank, contains only numbers, and the month is from 1-12 (See image 114). After that the expiration year is checked which involves making sure that the user is not inserting a date that has already passed, it is not empty, and the format is two digits (See image 115). Lastly, this file checks the CVV making sure it is not empty, does not contain letters, and is 3 digits (See image 116). If there are errors this page will display it before pressing the submit button. If there are no errors, then the errors will not be activated to be displayed (See image 117).

```
//On input that has class address perform these if statements on it.
if ($(this).hasClass('address')) {
    //Set variable addressChk to regular expressions that needs used to test user input against.
    var addressChk = /^[a-zA-Z0-9-' .\-]+$/i;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please type an address').fadeIn().parent('.form-group').addClass('hasError');
        address_err = true;
    }
    //Test the regular expression assigned to value addressChk against value user input in the form.
    else if (!addressChk.test($(this).val())) {
        $(this).siblings('span.error').text('Please only use numbers, letters, period or hyphen').fadeIn().parent('.form-group').addClass('hasError');
        address_err = true;
    }
    //Else set error to false and remove class hasError
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        address_err = false;
    }
}
```

Image 108: The payment.js code to check address text field for errors

```
//On input that has class city perform these if statements on it.
if ($(this).hasClass('city')) {
    //Set variable cityChk to regular expression that needs used to test user input against.
    var cityChk=/^[a-zA-Z-' ]*$/;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please enter a City').fadeIn().parent('.form-group').addClass('hasError');
        city_err = true;
    }
    //Test the regular express assigned to value cityChk against value user input in the form.
    else if (!cityChk.test($(this).val())) {
        $(this).siblings('span.error').text('Please only use letters and spaces').fadeIn().parent('.form-group').addClass('hasError');
        city_err = true;
    }
    //Else set error to false and remove class hasError.
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        city_err = false;
    }
}
```

Image 109: The payment.js code to check city text field for errors.

```
//On input that has class state perform these if statements on it.
if ($(this).hasClass('state')) {
    //Set variable stateChk to regular expression that needs used to test user input against.
    var stateChk=/^[A-Z]{2}$/;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please enter a State').fadeIn().parent('.form-group').addClass('hasError');
        state_err = true;
    }
    //Test the regular express assigned to value stateChk against value user input in the form.
    else if (!stateChk.test($(this).val())) {
        $(this).siblings('span.error').text('Please enter state abrieviation').fadeIn().parent('.form-group').addClass('hasError');
        state_err = true;
    }
    //Else set error to false and remove class hasError
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        state_err = false;
    }
}
```

Image 110: The payment.js code to check state text field for errors.

```
//On input that has class zip perform these if statements on it.
if ($(this).hasClass('zip')) {
    //Set variable zipChk to regular expression that needs used to test user input against.
    var zipChk = /^[0-9]{5}(-[0-9]{4})?$/;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please enter a zip code').fadeIn().parent('.form-group').addClass('hasError');
        zip_err = true;
    }
    //Test the regular express assigned to value zipChk against value user input in the form.
    else if (!zipChk.test($(this).val())) {
        $(this).siblings('span.error').text('Please enter valid state').fadeIn().parent('.form-group').addClass('hasError');
        zip_err = true;
    }
    //Else set error to false and remove class hasError
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        zip_err = false;
    }
}
```

Image 111: The payment.js code to check zip text field for errors.

```
//On input that has class cardname perform these if statements on it.
if ($(this).hasClass('cardname')) {
    //Set variable cardChk to regular expression that needs used to test user input against.
    var cardchk = /^[a-zA-Z-' ]*$/;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please enter name as seen on Card').fadeIn().parent('.form-group').addClass('hasError');
        cardname_err = true;
    }
    //Have user only allow length of the cardname to be 6 or more characters.
    else if ($(this).val().length > 1 && $(this).val().length <= 6) {
        $(this).siblings('span.error').text('Please type at least 6 characters').fadeIn().parent('.form-group').addClass('hasError');
        cardname_err = true;
    }
    //Test the regular express assigned to value cardChk against value user input in the form.
    else if (!cardchk.test($(this).val())) {
        $(this).siblings('span.error').text('Only letters, spaces, atleast 6 characters but no more than 50 allowed').fadeIn().parent('.form-group').addClass('has
        cardname_err = true;
    }
    //Else set error to false and remove class hasError
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        cardname_err = false;
    }
}
```

Image 112: The payment.js code to check card name text field for errors.

```
//On input that has class cardnumber perform these if statements on it.
if ($(this).hasClass('cardnumber')) {
    //Set variable cardNumChk to regular expression that needs used to test user input against.
    var cardNumChk = /^[0-9]{16}$/;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please enter a card number.').fadeIn().parent('.form-group').addClass('hasError');
        cardnumber_err = true;
    }
    //Test the regular express assigned to value cardNumChk against value user input in the form.
    else if (!cardNumChk.test($(this).val())) {
        $(this).siblings('span.error').text('Must be valid 16 digits').fadeIn().parent('.form-group').addClass('hasError');
        cardname_err = true;
    }
    //Else set error to false and remove class hasError
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        cardnumber_err = false;
    }
}
```

Image 113: The payment.js code to check card number text field for errors.

```
//On input that has class expmonth perform these if statements on it.
if ($(this).hasClass('expmonth')) {
    //Set variable expmonthChk to regular expression that needs used to test user input against.
    var expmonthChk = /^([1-9]|1[012])$/;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please enter the experation Month').fadeIn().parent('.form-group').addClass('hasError');
        expmonth_err = true;
    }
    //Test the regular express assigned to value expmonthChk against value user input in the form.
    else if(!expmonthChk.test($(this).val())){
        $(this).siblings('span.error').text('Please enter valid month').fadeIn().parent('.form-group').addClass('hasError');
        expmonth_err = true;
    }
    //Else set error to false and remove class hasError
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        expmonth_err = false;
    }
}
```

Image 114: The payment.js code to check exp month text field for errors.

```
//On input that has class expyear perform these if statements on it.
if ($(this).hasClass('expyear')) {
    //Set variable expyeaerChk to regular expression that needs used to test user input against.
        var expyearChk = /^[0-9]{2}$/;
        //Set d to date
        var d = new Date();
        //Set n to date and year.
        var n = d.getFullYear();
        //Set getYear to string and get last 2 number in string.
        var getYear = n.toString().substr(2);
        //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
        if ($(this).val().length === 0) {
            $(this).siblings('span.error').text('Please enter experation year').fadeIn().parent('.form-group').addClass('hasError');
            expyear_err = true;
        }
        //Test the regular express assigned to value expyearChk against value user input in the form.
        else if (!expyearChk.test($(this).val())) {
            $(this).siblings('span.error').text('Year formatted yy').fadeIn().parent('.form-group').addClass('hasError');
            expyear_err = true;
        }
        //Year needs to be this year or later.
       else if($(this).val() <= getYear){
          $(this).siblings('span.error').text('Please enter a valid year').fadeIn().parent('.form-group').addClass('hasError');
            expyear_err = true;
          }
        //Else set error to false and remove class hasError
        else{
           $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
           expyear_err = false;
        }
    }
```

Image 115: The payment.js code to check exp year text field for errors.

```
//On input that has class cvv perform these if statements on it.
if ($(this).hasClass('cvv')) {
    //Set variable cvvChk to regular expression that needs used to test user input against.
    var cvvChk = /^[0-9]{3}$/;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please enter cvv of your card').fadeIn().parent('.form-group').addClass('hasError');
        cvv_err = true;
    }
    //Test the regular express assigned to value cvvChk against value user input in the form.
    else if (!cvvChk.test($(this).val())) {
        $(this).siblings('span.error').text('Valid 3 digit cvv').fadeIn().parent('.form-group').addClass('hasError');
        cvv_err = true;
    }
    //Else set error to false and remove class hasError
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        cvv_err = false;
    }
}
```

Image 116: The payment.js code to check cvv text field for errors.

```
    //Check if the value in the input box has length to it more than 0 then adds css class active if the value is zero then it removes class.
    if ($(this).val().length > 0) {
        $(this).siblings('label').addClass('active');
    } else {
        $(this).siblings('label').removeClass('active');
    }

});
```

Image 117: The payment.js code used to activate and deactivate error popups.

The other file that helps with the payment page is the payment.php file that does server-side validation after the user presses the submit button. This file starts with the php validation for the users input by looking at the address to make sure that it is not empty and only contains letters, numbers, period, and dash (See image 118). If there is no error, then the address that the user inputted will be stored in the variable $address. Next it looks at the city field to make sure it is not empty and only contains letters (See image 119). If this has no errors, then the city will be stored in the variable $city. After the city it looks at the state input making sure it is only made of two letters and it is not empty (See image 120). If there are no errors, then the variable $state will be assigned to the inputted information. Lastly, for the billing information the zip code is

assessed making sure it is not empty, has 5 numbers, or has 5 numbers with a dash then 4 numbers (See image 121). If there are no errors, then the variable $zip will be assigned to the inputted information.

Next the code looks over the information for the credit card information to make sure that the code is accurate and does not contain any injections. The card name is looked at to make sure it is not empty, does not contain numbers, and is 6 to 50 characters (See image 122). If there are no errors, then the variable $cardname will store the inputted information. Next the card number is looked at to make sure it is not empty, contains only numbers, and is 16 digits (See image 123). If there are no errors, then the variable $cardnumber stored the inputted value. After that the expiration month is assessed to make sure it is not empty, numbers only, and is a valid month using 1-12 (See image 124). If no errors are given, then the variable $expmonth is assigned the inputted value. Then the year is checked to make sure that the field is not empty, the year is not past, it is only numbers, and only two digits (See image 125). If there are not errors, then the variable $expyear is assigned the value the user inputted. Lastly, the cvv is checked to make sure it is not empty, there are only 3 digits, and only numbers are present (See image 126). If there are no errors, then the variable $cvv is assigned the inputted value.

```php
$regAddress = $_POST['address'];
//Checks to see if the address field is empty in the form.
if(empty($_POST['address'])){
    $address_err = "Address is required";
}
/**Uses regular expressions to check that upper case, lower case letters, and period or dash can be used.
 * This checks if the address meets a certain criteria lower case letters, upper case letters, period or dash, minumum length 10 characters.
 */
elseif(!preg_match("/^[a-zA-Z0-9' .\-]{10,100}+$/i", $regAddress)){
    $address_err = "error";
}
else{
    //If above passes if statements then address the user inputted into the form is set to the $address variable.
    $address = $_POST['address'];
    $address_err = "";
}
```

Image 118: The payment.php code used to check the address after pressing continue checkout button

```php
$regCity = $_POST['city'];
//Check if city field is empty
if(empty($_POST['city'])){
    $city_err = "City is required";
}
//Check if the city has lower case and upper case letter.
elseif(!preg_match("/^[a-zA-Z-' ]*$/", $regCity)){
    $city_err = "error";
}
//Else assign the city value to $city variable
else{
    //If variable passes the above two if checks then the value is assigned to a variable.
    $city = $_POST['city'];
    $city_err = "";
}
```

Image 119: The payment.php code used to check the city after pressing continue checkout button

```php
$regState = $_POST['state'];
//Check if the value is empty
if(empty($_POST['state'])){
    $state_err = "State is requred";
}
//Use regular expression to see if the value has only upper and lower case letters.
elseif(!preg_match("/^[a-zA-Z-' ]{2}*$/", $regState)){
    $state_err = "error";
}
//If value passes the above two if checks then the value is assigned to a variable.
else{
    $state = $_POST['state'];
    $state_err = "";
}
```

Image 120: The payment.php code used to check the state after pressing continue checkout button

```php
$regZip = $_POST['zip'];
//Check if the value is empty.
if(empty($_POST['zip'])){
    $zip_err = "Zip is requred";
}
//Use regular expression to see of the value has 5 numbers and optional 4 numbers.
elseif(!preg_match("/^[0-9{5}(-[0-9]{4}]?$/", $regZip)){
    $zip = $_POST['zip'];
    $zip_err = "";
}
//If value passes the above if checks then the value is assigned to a variable.
else{
    $zip_err = "error";
}
```

Image 121: The payment.php code used to check the zip after pressing continue checkout button

```php
$regCardName = $_POST['cardname'];
//Check if the value is empty.
if(empty($_POST['cardname'])){
    $cardname_err = "Name for card is requred";
}
//Use upper case and lower case letters only.
elseif(!preg_match("/^[a-zA-Z-' ]{6,25}*$/", $regCardName)){
    $cardname_err = "error";
}
//If value passes the above two if checks then the value is assigned to a variable.
else{
    $cardname = $_POST['cardname'];
    $cardname_err = "";
}
```

Image 122: The payment.php code used to check the card name after pressing continue checkout button

```javascript
//On input that has class cardnumber perform these if statements on it.
if ($(this).hasClass('cardnumber')) {
    //Set variable cardNumChk to regular expression that needs used to test user input against.
    var cardNumChk = /^[0-9]{16}$/;
    //Check if the length value is equal to zero if so then show error and add class hasError to the span html tag.
    if ($(this).val().length === 0) {
        $(this).siblings('span.error').text('Please enter a card number.').fadeIn().parent('.form-group').addClass('hasError');
        cardnumber_err = true;
    }
    //Test the regular express assigned to value cardNumChk against value user input in the form.
    else if (!cardNumChk.test($(this).val())) {
        $(this).siblings('span.error').text('Must be valid 16 digits').fadeIn().parent('.form-group').addClass('hasError');
        cardnumber_err = true;
    }
    //Else set error to false and remove class hasError
    else{
        $(this).siblings('.error').text('').fadeOut().parent('.form-group').removeClass('hasError');
        cardnumber_err = false;
    }
}
```

Image 123: The payment.php code used to check the card number after pressing continue checkout button

```php
$regExpMonth = $_POST['expmonth'];
//Check if the value it empty.
if(empty($_POST['expmonth'])){
    $expmonth_err = "Expiration month is requred";
}
//Only has numbers 1 through 12
elseif(!preg_match("/^([0-9]|01[012])$/", $regExpMonth)){
    $expmonth_err = "error";
}
//If value passes the above two if checks then the value is assigned to a variable.
else{
    $expmonth = $_POST['expmonth'];
    $expmonth_err = "";
}
```

Image 124: The payment.php code used to check the exp month after pressing continue checkout button

```php
$regExpyear = $_POST['expyear'];
//Set getYear to string and get last two number in string
$getYear = date("y");
//Check if the value is empty.
if(empty($_POST['expyear'])){
    $expyear_err = "Expiration year is requred";
}
//Has only 2 numbers in length/
elseif(!preg_match("/^[0-9]{2}$/", $regExpyear)){
    $expyear_err = "error";
}
elseif($regExpyear <= $getYear){
    $expyear_err = "error";
    //echo "<script>alert('". $getYear ."')</script>";
}
//If value passes the above two if checks then the value is assigned to a variable.
else{
    $expyear = $_POST['expyear'];
    $expyear_err = "";
}
```

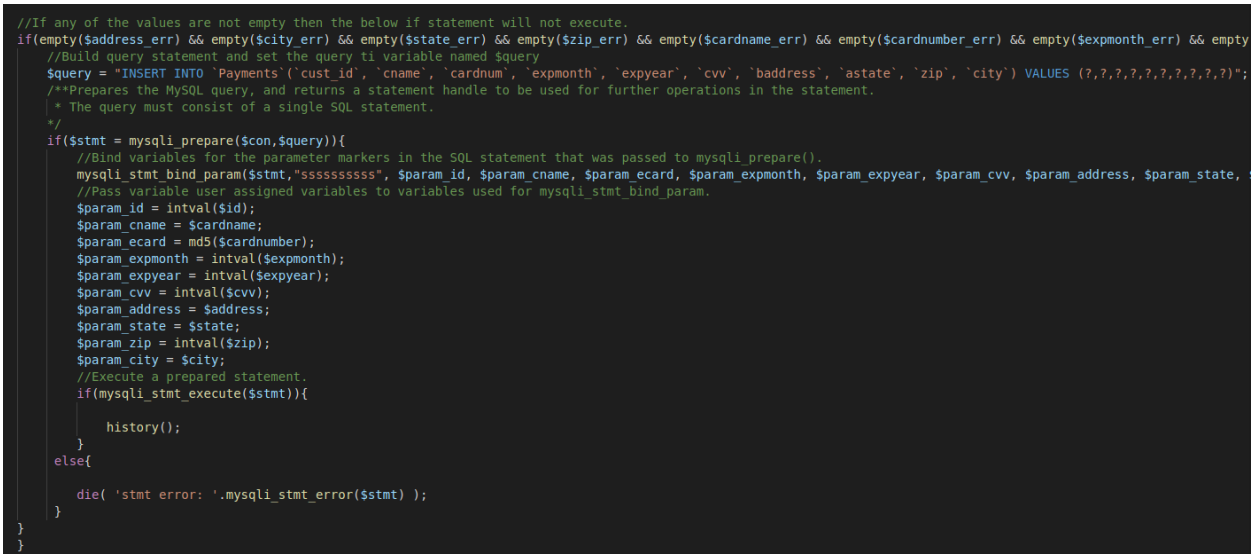
Image 125: The payment.php code used to check the exp year after pressing continue checkout button

```php
$regCvv = $cvv = $_POST['cvv'];
//Check if the value is empty.
if(empty($_POST['cvv'])){
    $cvv_err = "CVV is requred";
}
//Only use 3 length in numbers.
elseif(!preg_match("/^[0-9]{3}$/", $regCvv)){
    $cvv_err = "error";
}
//If value passes the above two if checks then the value is assigned a variable.
else{
    $cvv = $_POST['cvv'];
    $cvv_err = "";
}
```

Image 126: The payment.php code used to check the cvv after pressing continue checkout button

After checking for validation of user input the payment.php file checks to see if there were any errors. If there are no errors, then the variable $query is assigned an INSERT statement to insert into the payment table the cust_id, card name, card number, expiration month,

expiration year, cvv, address, state, zip, and city unknown values we get from the input. Next the statement is prepared so we can connect to the database and use the insert statement. Next, we bind unknown variables to the variables we need to make sure that only 10 items are used. This is used to help prevent XSS and SQL injections. Then the variables that were defined earlier are brought down to be assigned to another variable, making sure the integer values are integer and the string values are string values. Additionally, it makes sure the $cardnumber variable is hashed. After this is done the statement is executed and the function history() is used. If there are errors, then an error message will display (See image 127).



```
//If any of the values are not empty then the below if statement will not execute.
if(empty($address_err) && empty($city_err) && empty($state_err) && empty($zip_err) && empty($cardname_err) && empty($cardnumber_err) && empty($expmonth_err) && empty
    //Build query statement and set the query ti variable named $query
    $query = "INSERT INTO `Payments`(`cust_id`, `cname`, `cardnum`, `expmonth`, `expyear`, `cvv`, `baddress`, `astate`, `zip`, `city`) VALUES (?,?,?,?,?,?,?,?,?,?)";
    /**Prepares the MySQL query, and returns a statement handle to be used for further operations in the statement.
     * The query must consist of a single SQL statement.
    */
    if($stmt = mysqli_prepare($con,$query)){
        //Bind variables for the parameter markers in the SQL statement that was passed to mysqli_prepare().
        mysqli_stmt_bind_param($stmt,"sssssssss", $param_id, $param_cname, $param_ecard, $param_expmonth, $param_expyear, $param_cvv, $param_address, $param_state,
        //Pass variable user assigned variables to variables used for mysqli_stmt_bind_param.
        $param_id = intval($id);
        $param_cname = $cardname;
        $param_ecard = md5($cardnumber);
        $param_expmonth = intval($expmonth);
        $param_expyear = intval($expyear);
        $param_cvv = intval($cvv);
        $param_address = $address;
        $param_state = $state;
        $param_zip = intval($zip);
        $param_city = $city;
        //Execute a prepared statement.
        if(mysqli_stmt_execute($stmt)){

            history();
        }
     else{

        die( 'stmt error: '.mysqli_stmt_error($stmt) );
     }
}
}
```

Image 127: The payment.php code that sanitizes the input to be inserted into the database.

The history() function is only brought in on the payment.php page, so I am going to talk about that one right now before we move on to the rest of the payment.php page. This function is used to get the values from the cart and insert it into the history table when moving from the payment.php page to the success.php page (See image 128). This function uses the global $con

variable that is used to connect to the database. The $id variable is assigned the cust_id that was acquired when the user logged into the website. The variable $conf_number is assigned to the confirmationNum() function that I will go in more detail about later. Next an array is declared and the variable $count is assigned to 0. A select statement is assigned to the $get_history variable that gets the bookid, ip_add, cust_id from the cart matching the unknown cust_id.

$get info is a prepared statement that is used to connect to the database to execute the SELECT statement assigned to $get_history. Next the statement is bound to make sure only one variable is used for the $id variable, to prevent SQL injections and XSS injections. Then the statement is executed, and the results are bound to the appropriate variables. A while statement is used to fetch the results and store then in an array then the $count variable is increased by 1. The statement is then closed so the next query can be executed. A for loop is used to increment through the array and place the status as ordered with the information from the array. An INSERT statement is used to insert the values from the input into the history table. The statement is prepared for connecting to the database to execute the INSERT statement. Then the parameters are bound to make sure we only have five unknown variables to insert into the table. If we cannot bind the parameters, then an error will occur (See image 129). If the history function does not work, then errors will be displayed. Lastly, this method will delete the books from the cart table by making a DELETE statement that deletes the information depending on the customers ID. The statement will be prepared to connect to the database and execute the prepared statement. The statement is then bound and executed to delete the information from the table (See image 129).

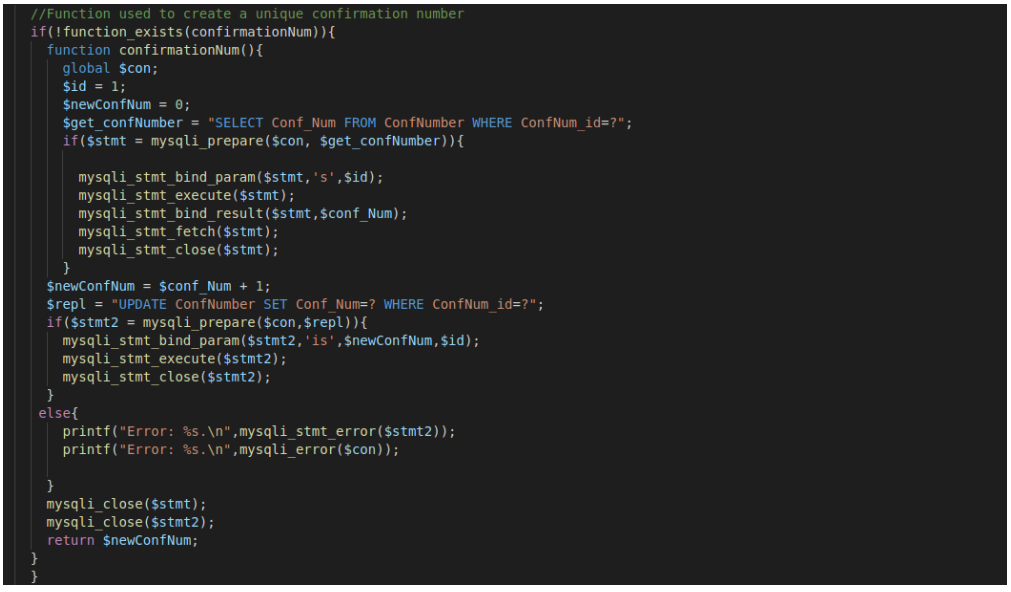
```php
/** Function used to get values from cart and insert it into the history table.
 * When moving from the payment page to successpay page.
 */
if(!function_exists(history)){
  function history(){

        global $con;
        $id = $_SESSION["cust_id"];
        $conf_number = confirmationNum();
        $myarray = array();
        $count = 0;
        $get_history = "SELECT bookid, ip_add, cust_id FROM cart WHERE  cust_id=?";
        //mysqli_prepare prepares the $get_history statement for execution.
        if($get_info = mysqli_prepare($con, $get_history)){
          //Bind variables to the prepared statement as parameters
          mysqli_stmt_bind_param($get_info,'s',$id);
          //Attempt to execute the prepared statement
          mysqli_stmt_execute($get_info);
          mysqli_stmt_bind_result($get_info,$book_id,$ip_add,$cust_id);
          $count=0;
          while(mysqli_stmt_fetch($get_info)){
            $myarray[] = array($book_id, $ip_add, $cust_id);
            $count = $count + 1;
          }
          mysqli_stmt_close($get_info);
        }

        for($x = 0; $x < $count; $x++){
        $ip = $myarray[$x][1];
        $bookid = $myarray[$x][0];
        $status = "ordered";
        $array_id = $myarray[$x][2];
        $insert_history="INSERT INTO history(confirmation_number, bookid, ip_add, cust_id, order_status) VALUES (?,?,?,?,?)";
         //mysqli_prepare prepares the $insert_history statement for execution.
         if($run_history = mysqli_prepare($con, $insert_history)){
          //Bind variables to the prepared statement as parameters
          if(mysqli_stmt_bind_param($run_history,'iisss',$prep_confNum,$prep_bookid,$prep_ip,$prep_id,$prep_status)){
         }
         else{
           printf("Error: %s.\n",mysqli_stmt_error($run_history),E_USER_ERROR);
```

Image 128: The code for the history() function.

```php
  $prep_confNum = intval($conf_number);
   $prep_bookid = intval($bookid);
   $prep_ip = strval($ip);
   $prep_id = strval($id);
   $prep_status = strval($status);
 if(mysqli_stmt_execute($run_history)){
   echo "yay";
 }
 else{

   printf("Error: %s.\n",mysqli_stmt_error($run_history),E_USER_ERROR);
 }
}
else{
 printf("Error: %s.\n",mysqli_stmt_error($get_info));
 printf("Error: %s.\n",mysqli_stmt_error($run_history));
 printf("Error: %s.\n",mysqli_error($con));
 echo("Error description: " . mysqli_error($con));


$delete_books = "DELETE FROM cart WHERE cust_id = ?";
if($run_delete = mysqli_prepare($con, $delete_books)){
 mysqli_stmt_bind_param($run_delete,'s',$id);
 mysqli_stmt_execute($run_delete);
}

if($run_delete){
   echo "<script>window.open('successpay.php','_self');</script>";
}
```

Image 129: The code for the history() function, when errors occur.

The function confirmationNum() used in the history() function creates a unique confirmation number (See image 130) so every order has a different number. The global variable $con is used to connect to the database. The $id variable is assigned the number one and the $newConfNum variable is assigned zero. A SELECT statement is used to get the conf_num from the ConfNum table where the ConfNum_id is equal to the unknown variable. The statement is then prepared by connecting to the database to execute the SELECT statement. Then the statement is binding the parameter to make sure that only one variable can be passed in. This is to prevent SQL and XSS injections. The statement is then executed, and the results are bound to the appropriate variable. Then the statement fetches the results and closes the statement for the next query to be executed. The newConfNum variable will be assigned the conf_num + 1 and then an UPDATE statement is used to set the unknown Conf_Num variable depending on the unknown ConfNum_id. The statement is prepared, bound using two unknown variables, executed, and then closed for the next query to be executed. If an update cannot occur, then errors will display. Lastly, all is closed and the variable $newConfNum is returned.



```
//Function used to create a unique confirmation number
if(!function_exists(confirmationNum)){
  function confirmationNum(){
    global $con;
    $id = 1;
    $newConfNum = 0;
    $get_confNumber = "SELECT Conf_Num FROM ConfNumber WHERE ConfNum_id=?";
    if($stmt = mysqli_prepare($con, $get_confNumber)){

      mysqli_stmt_bind_param($stmt,'s',$id);
      mysqli_stmt_execute($stmt);
      mysqli_stmt_bind_result($stmt,$conf_Num);
      mysqli_stmt_fetch($stmt);
      mysqli_stmt_close($stmt);
    }
  $newConfNum = $conf_Num + 1;
  $repl = "UPDATE ConfNumber SET Conf_Num=? WHERE ConfNum_id=?";
  if($stmt2 = mysqli_prepare($con,$repl)){
    mysqli_stmt_bind_param($stmt2,'is',$newConfNum,$id);
    mysqli_stmt_execute($stmt2);
    mysqli_stmt_close($stmt2);
  }
 else{
    printf("Error: %s.\n",mysqli_stmt_error($stmt2));
    printf("Error: %s.\n",mysqli_error($con));

  }
  mysqli_close($stmt);
  mysqli_close($stmt2);
  return $newConfNum;
}
}
```

Image 130: The code for the confirmationNum() function.

The last part of the payment.php page is the layout of the payment page. This shows the layout for the billing address which includes the address, city, state, and zip (See image 131). It shows the payment portion that has the accepted cards section then goes down the text fields that have the name on the card, credit card number, expiration month, expiration year, and CVV (See image 132). Then at the very end of the page there is the submit button which checks the information to make sure that is valid before being inserted into the database table of payments (See image 133). After the user has submitted their information, they are redirected to the order status page.

```
<break>
    <div class="row">
        <div class="col-75">
            <div class="container">
                <form method="post" action="payment.php">

                    <div class="row">
                        <div class="col-50">
                            <h3>Billing Address</h3>
                            <div class="form-group">
                            <label for="adr"><i class="fa fa-address-card-o"></i> Address</label>
                            <input type="text" id="adr" name="address" class="address" placeholder="542 W. 15th Street">
                            <span class="error" style="display:none"></span>
                             </div>

                             <div class="form-group">
                            <label for="city"><i class="fa fa-institution"></i> City</label>
                            <input type="text" id="city" name="city" class="city" placeholder="New York">
                            <span class="error" style="display:none"></span>
                            </div>

                            <div class="row">
                                <div class="col-50">
                                <div class="form-group">
                                    <label for="state">State</label>
                                    <input type="text" id="state" name="state" class="state" placeholder="NY">
                                    <span class="error" style="display:none"></span>
                                 </div>
                                </div>
                                <div class="col-50">
                                <div class="form-group">
                                    <label for="zip">Zip</label>
                                    <input type="text" id="zip" name="zip" class="zip" placeholder="10001">
                                    <span class="error" style="display:none"></span>
                                 </div>
                                    </div>
                            </div>
                        </div>

                        <div class="col-50">
                            <h3>Payment</h3>
                            <label for="fname">Accepted Cards</label>
```

Image 131: The payment.php code displaying the layout for the billing information.

```
<div class="col-50">
    <h3>Payment</h3>
    <label for="fname">Accepted Cards</label>
    <div class="icon-container">
        <i class="fa fa-cc-visa" style="color:navy;"></i>
        <i class="fa fa-cc-amex" style="color:blue;"></i>
        <i class="fa fa-cc-mastercard" style="color:red;"></i>
        <i class="fa fa-cc-discover" style="color:orange;"></i>
    </div>
    <div class="form-group">
    <label for="cname">Name on Card</label>
    <input type="text" id="cname" name="cardname" class="cardname" placeholder="John More Doe">
    <span class="error"  style="display:none"></span>
     </div>

    <div class="form-group">
    <label for="ccnum">Credit card number</label>
    <input type="text" id="ccnum" name="cardnumber" class="cardnumber" placeholder="1111222233334444">
    <span class="error"  style="display:none"></span>
    </div>

    <div class="form-group">
    <label for="expmonth">Exp Month</label>
    <input type="text" id="expmonth" name="expmonth" class="expmonth" placeholder="1">
    <span class="error"  style="display:none"></span>
    </div>

    <div class="row">
        <div class="col-50">
        <div class="form-group">
            <label for="expyear">Exp Year</label>
           <input type="text" id="expyear" name="expyear" class="expyear" placeholder="21">
           <span class="error" style="display:none"></span>
            </div>
            </div>
        <div class="col-50">
            <div class="form-group">
            <label for="cvv">CVV</label>
            <input type="text" id="cvv" name="cvv" class="cvv" placeholder="352">
            <span class="error"  style="display:none"></span>
```

Image 132: The payment.php code displaying the layout for the card information.

```
                <div class="form-group">
                <label for="ccnum">Credit card number</label>
                <input type="text" id="ccnum" name="cardnumber" class="cardnumber" placeholder="1111222233334444">
                <span class="error"  style="display:none"></span>
                </div>

                <div class="form-group">
                <label for="expmonth">Exp Month</label>
                <input type="text" id="expmonth" name="expmonth" class="expmonth" placeholder="1">
                <span class="error"  style="display:none"></span>
                </div>

                <div class="row">
                    <div class="col-50">
                    <div class="form-group">
                        <label for="expyear">Exp Year</label>
                       <input type="text" id="expyear" name="expyear" class="expyear" placeholder="21">
                       <span class="error" style="display:none"></span>
                        </div>
                        </div>
                    <div class="col-50">
                        <div class="form-group">
                        <label for="cvv">CVV</label>
                        <input type="text" id="cvv" name="cvv" class="cvv" placeholder="352">
                        <span class="error"  style="display:none"></span>
                        </div>
                    </div>
                </div>
            </div>
            </div>
            <input type="submit" name = "submit" value="Continue to checkout" class="btn">
        </form>
    </div>
</div>
</div>
</body>
</html>
```

Image 133: The payment.php code displaying the submit button that will cause the server-side validation to occur

Going back to the web page we will look at the layout of the page then look at the code that corresponds with this page (See image 134). This page consists of two tables one that has the current orders, meaning their status is ordered and then the previous orders, meaning the status is completed. The current order table consist of the confirmation number, book title, cost, status, and total price. The previous order table consist of the confirmation number, book title, cost, and status. Now to look at the code for this page. The successpay.php code has the same code at the beginning as the payment page and cart page. It has the navigation bar with the home tab, logout, email address, and cart tabs. Next it goes into the layout of the tables (See image 135), the current order table uses the getOrders() function and the previous orders table uses the getHistory() function.

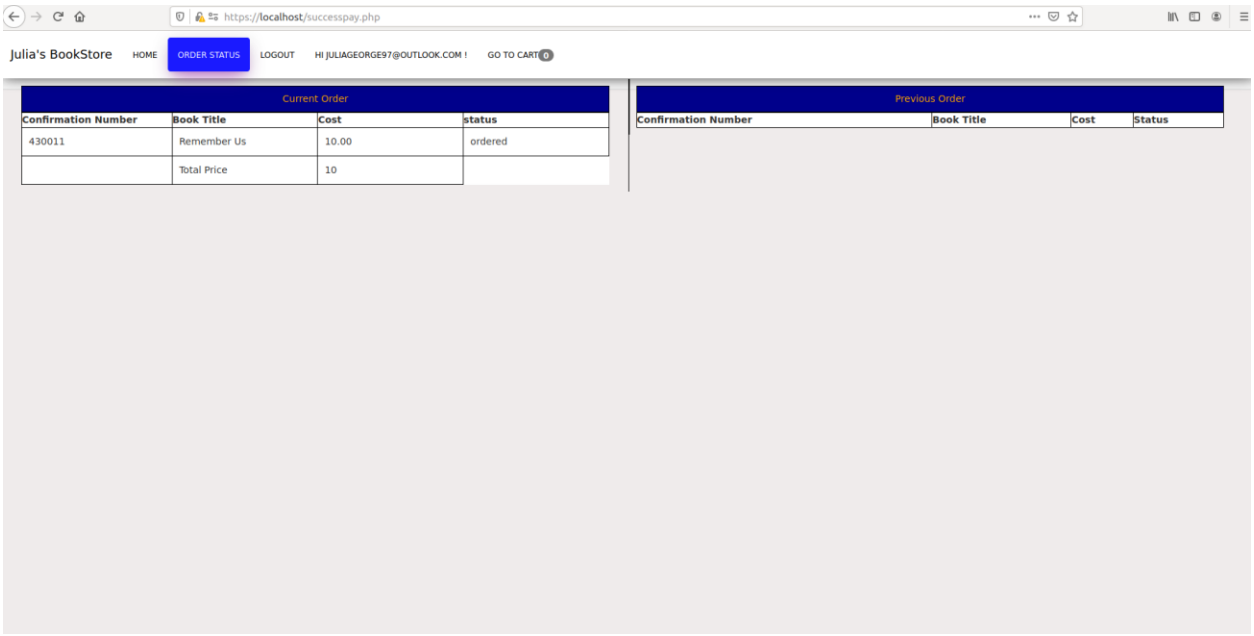


Image 134: The order status page, and the display of previous/current orders.

```
<div class="container2">
    <div class='left'>
    <table class='payment'>
        <caption class='payment'> Current Order</caption>
        <tr class="payment">
            <th class='payment'>Confirmation Number</th>
            <th class='payment'>Book Title</th>
            <th class='payment'>Cost</th>
            <th class='payment'>status</th>
        </tr>
            <?php getOrders(); ?>
    </table>
    </div>
    <div class='right'>
        <table class='payment'>
            <caption class='payment'> Previous Order</caption>
            <tr class="payment">
                <th class='payment'>Confirmation Number</th>
                <th class='payment'>Book Title</th>
                <th class='payment'>Cost</th>
                <th class='payment'>Status</th>
            </tr>
                <?php getHistory(); ?>
        </table>
    </div>
</div>
```

Image 135: The code for the layout of the history page, successpay.php file.

The getOrders() function is used to display the orders where the status is equal to ordered (See image 136). This file also uses the global $con variable to connect to the database. It gets the session id and assigns it to the variable $id. A $total_price variable is declared and assigned the value zero. Next the variable $stat is assigned to the word ordered. A SELECT statement is used and JOINED on bookid. This gets the values of title and price from the books table then gets the values of bookid, confirmation_number, and order_status from the history table. Then two unknown variables are assigned. A prepare statement is used to connect to the database so we can execute the prepared statement. Parameters are bound to make sure that only two

variables are accepted to prevent possible injections. The statement is then executed, and the results are bound to their appropriate variables. A while loop is used to fetch the information and display it onto the table. Then the total price is calculated based on the price we got from the book and the total is displayed.

```php
// Function used on the seccesspay page to display orders where status is equal to ordered.
if(!function_exists(getOrders)){
  function getOrders(){

        global $con;

        $id = $_SESSION["cust_id"];
        $total_price = 0;
        $stat = 'ordered';
        $get_history = "SELECT hs.bookid, bk.title, hs.confirmation_number, bk.price, hs.order_status FROM history as hs JOIN Books as bk ON hs.bookid = bk.book_id
        if($get_info = mysqli_prepare($con, $get_history)){
            mysqli_stmt_bind_param($get_info,'ss',$id,$stat);
            mysqli_stmt_execute($get_info);
            mysqli_stmt_bind_result($get_info,$book_id,$bk_title,$conf_num,$price,$status);

        while(mysqli_stmt_fetch($get_info)){
       $total_price += $price;
       echo "
       <tr class='payment'>
       <td class='payment'>".$conf_num."</td>
       <td class='payment'>".$bk_title."</td>
       <td class='payment'>".$price."</td>
        <td class='payment'>".$status."</td>
        </tr>
        ";
      }
      echo "
      <tr class='payment'>
      <td class='payment'></td>
      <td class='payment'>Total Price</td>
      <td class='payment'>".$total_price."</td>
      </tr>";
    }
  }
}
```

Image 136: The code for the getOrders() function.

The next function used in the history page is the getHistory() function which is used to display orders where the status is equal to completed (See image 137). This file also uses the global $con variable to connect to the database. It gets the session id and assigns it to the variable $id. Next the variable $stat is assigned to the word completed. A SELECT statement is used and JOINED on bookid. This gets the values of title and price from the books table then gets the values of bookid, order_status, and confirmation_number from the history table. Then two unknown variables are assigned. A prepare statement is used to connect to the database so we can execute the prepared statement. Parameters are bound to make sure that only two variables are accepted to prevent possible injections. The statement is then executed, and the results are

bound to their appropriate variables. A while loop is used to fetch the information and display it onto the table. Then the statement is closed so another query can be executed.

```php
//Function used on the successpay page to display orders where status is equal to completed.
if(!function_exists(getHistory)){
  function getHistory(){
        global $con;

        $id = $_SESSION["cust_id"];
        $status_bk = 'completed';
        $get_hs = "SELECT hs.bookid, hs.order_status, hs.confirmation_number, bk.title, bk.price FROM history as hs JOIN Books as bk ON hs.bookid = bk.book_id WHERE

        if($get_hsinfo = mysqli_prepare($con, $get_hs)){
          mysqli_stmt_bind_param($get_hsinfo,'ss',$id,$status_bk);
          mysqli_stmt_execute($get_hsinfo);
          mysqli_stmt_bind_result($get_hsinfo,$bookid,$status,$conf_numHS,$bk_title,$price);
          mysqli_stmt_fetch($get_hsinfo);
        }
        while(mysqli_stmt_fetch($get_hsinfo)){
      echo "<tr class='payment'>
      <td class='payment'>".$conf_numHS."</td>
      <td class='payment'>".$bk_title."</td>
      <td class='payment'>".$price."</td>
      <td class='payment'>".$status."</td>
      </tr>";
    }
    mysqli_stmt_close($get_hsinfo);
  }
}
```

Image 137: The code for the getHistory() function.

To change the status from ordered to completed we will need to go to the history table in the database (See image 138). Clicking on the edit link will bring the employee or admin to where they can change from ordered to completed (See image 139). The status has to be spelled correctly and specifically say completed, if that is not done then it will not who up as a previous order for the user. Clicking the go button will save the edited order status and will display completed instead of ordered (See image 140). Going back to the website and refreshing the page to show the new order status will now show the order under the previous order table (See image 141). The last thing to discuss is the logout tab with the code that is used to log out the user. When the user presses the logout link the user will be logged out and the sign in page will be displayed. When the user logs out the logout.php file is used (See image 142). This file will initialize the session, unset all of the session variables, destroy the session, and lastly will redirect the user to the login1.php page.

	book_index	confirmation_number	bookid	ip_add	cust_id	order_status
Edit Copy Delete	1	430002	1	127.0.0.1	17	completed
Edit Copy Delete	2	430002	3	127.0.0.1	17	completed
Edit Copy Delete	3	430003	9	127.0.0.1	17	completed
Edit Copy Delete	4	430003	10	127.0.0.1	17	completed
Edit Copy Delete	5	430004	10	127.0.0.1	17	ordered
Edit Copy Delete	6	430004	15	127.0.0.1	17	ordered
Edit Copy Delete	7	430004	21	127.0.0.1	17	ordered
Edit Copy Delete	8	430005	9	127.0.0.1	17	ordered
Edit Copy Delete	9	430005	15	127.0.0.1	17	ordered
Edit Copy Delete	10	430006	1	127.0.0.1	17	ordered
Edit Copy Delete	11	430006	18	127.0.0.1	17	ordered
Edit Copy Delete	12	430007	1	127.0.0.1	17	ordered
Edit Copy Delete	13	430008	17	127.0.0.1	17	ordered
Edit Copy Delete	14	430009	10	127.0.0.1	17	ordered
Edit Copy Delete	15	430010	3	127.0.0.1	17	ordered
Edit Copy Delete	16	430011	10	127.0.0.1	23	ordered

Image 138: The History table in the database.

Column	Type	Function	Null	Value
book_index	int(11)			16
confirmation_number	int(50)			430011
bookid	int(50)			10
ip_add	varchar(255)			127.0.0.1
cust_id	varchar(1000)			23
order_status	varchar(50)			completed

Go

Image 139: Editing the order status in the database.

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Ope

Edit Copy Delete	1	430002	1	127.0.0.1	17	completed
Edit Copy Delete	2	430002	3	127.0.0.1	17	completed
Edit Copy Delete	3	430003	9	127.0.0.1	17	completed
Edit Copy Delete	4	430003	10	127.0.0.1	17	completed
Edit Copy Delete	5	430004	10	127.0.0.1	17	ordered
Edit Copy Delete	6	430004	15	127.0.0.1	17	ordered
Edit Copy Delete	7	430004	21	127.0.0.1	17	ordered
Edit Copy Delete	8	430005	9	127.0.0.1	17	ordered
Edit Copy Delete	9	430005	15	127.0.0.1	17	ordered
Edit Copy Delete	10	430006	1	127.0.0.1	17	ordered
Edit Copy Delete	11	430006	18	127.0.0.1	17	ordered
Edit Copy Delete	12	430007	1	127.0.0.1	17	ordered
Edit Copy Delete	13	430008	17	127.0.0.1	17	ordered
Edit Copy Delete	14	430009	10	127.0.0.1	17	ordered
Edit Copy Delete	15	430010	3	127.0.0.1	17	ordered
Edit Copy Delete	16	430011	10	127.0.0.1	23	completed

Check all *With selected:* Edit Copy Delete Export

Image 140: The history table after the order_status has been updated.

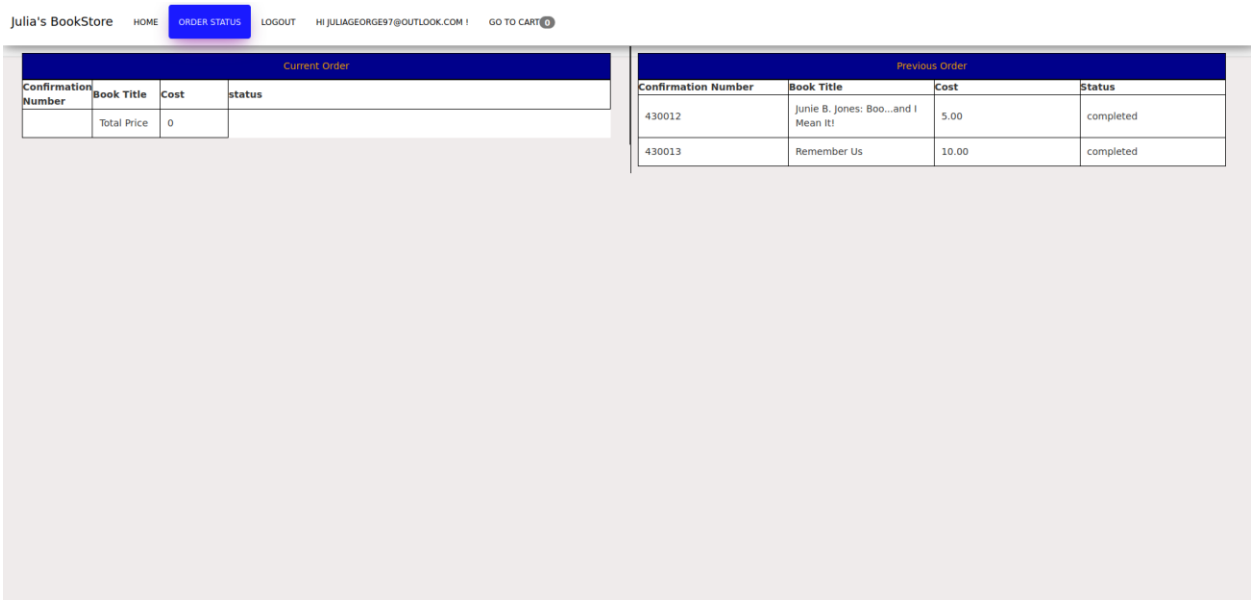


Image 141: The order status page after changing the order status in the database

```php
<?php
// Initialize the session
session_start();

// Unset all of the session variables
$_SESSION = array();

// Destroy the session.
session_destroy();

// Redirect to login page
header("location: login1.php");
exit;
?>
```

Image 142: The code used to log out, logout.php

The last thing I did with this website was to create a self-signed certificate to setup https on the apache server. The self-signed certificates were created with built in linux utility called openssl and other values. OpenSSL is the basic command line tool for creating and managing OpenSSL certificates, keys, and other files. The OpenSLL comes with options that we used in order to make our self-signed certficates. These options are: Req, this subcommand specifies that we want to use X.509 certificate signing request (CSR) management. The “X.509” is a public key infrastructure standard that SSL and TLS adhere to for its key and certificate management. We want to create a new X.509 cert, so we are using this subcommand. X.509, this further modifies the previous subcommand by telling the utility that we want to make a self-signed certificate instead of generating a certificate signing request, as would normally happen. Nodes,

this tells OpenSSL to skip the option to secure out certificate with a passphrase. We need Apache to be able to read the file, without user intervention, when the server starts up. A passphrase would prevent this from happening because we would have to enter it after every restart. Day 365, this option sets the length of time that the certificate will be considered valid. We set it for one year here. New key rsa:2048, this specifies that we want to generate a new certificate and a new key at the same time. We did not create the key that is required to sign the certificate in a previous step, so we need to create it along with the certificate. The rsa:2028 portion tells it to make an RSA key that is 2048 bits long. Keyout, this line tells OpenSSL where to place the generate private key file that we are creating. Lastly, out, this tells OpenSSL where to place the certificate that we are creating.

The certs are then saved in directories that have restricted permissions. The SSLCertificate file is saved to /etc/ssl/certs/apache-selfsigned.crt and SSLCertificateKeyFile is saved to /etc/ssl/private/apache-selfsigned.key is saved on /etc/ssl/certs/dhparam.pem. Once all the certs have been created, the CA, public, and server certs, these certs can be used to validate that the website is a trusted site. Apache then needs few files changed so that the website can use https instead of http. These files include /etc/apache2/sites-available/default-ssl.conf , /etc/apache2/conf-available/ssl-params.conf, and /etc/apache2/sites-available/000-default.conf. Then the firewall needs to be adjusted so the https traffic can access the website: 22/tcp ALLOW Anywhere, Apache Full ALLOW Anywhere, 25 ALLOW Anywhere, Apache ALLOW Anywhere, 3306 ALLOW Anywhere, 22/tcp (v6) ALLOW Anywhere (v6), Apache Full (v6) ALLOW Anywhere (v6), 25 (v6) ALLOW Anywhere (v6), Apache (v6) ALLOW Anywhere (v6), and 3306 (v6) ALLOW Anywhere (v6). After all this is done then apache server needs to be reset to get it to work.

Next there needs to be encryption to the database. Certificates used for php connection to the database is ca.pem. Three certificates setup for use on mysql ca.pem, server-cert.pem and server-key.pem to be used for authentication from php to database. The problem I had with the encryption to the database is that php and MySQL do not like authenticating on a localhost with certificates. So, I had to perform a work around to get the certs to work between the two. Using self-signed client key and server key and ca.cert file. I merged the client key and server key file with the ca.cert file. That way the client will trust the server certificate and the server will trust the client certificate.

In the config php file uses code to connect using encryption to the database (See image 143 & 144). The code uses the global variable that connects to the database to set the cert to be used because it is a localhost, we are using CA cert that has the client and server certificate include in it. Next, it uses a real connect to connect to the database and force SSL connection on port 3306. Lastly, on the database we will select to require SSL for user authentication (See image 145). Certs were set on MySQL and here are the certs that php authenticates against to connect to MySQL (See image 146).

```php
//Set variables to values for connection to database.
$Servername = '127.0.0.1';
$userName = 'ekbaker';
$pass = 'X5j13$#eCM1cG@Kdc';
$db2 = 'ecom';

/* Attempt to connect to MySQL database */
$con = mysqli_init();

/*verify server cert is set to true*/
mysqli_options($con,MYSQLI_OPT_SSL_VERIFY_SERVER_CERT,true);
```

Image 143: config.php code for connecting to the database.

```php
/*set the cert to be used because this is local host we are using ca cert*/
$con->ssl_set(NULL,NULL,'/etc/mysql/ca.pem',NULL,NULL);
/*Uses real connect to connect to the database and forcing SSL connection on port 3306*/
$db = mysqli_real_connect($con,$Servername,$userName,$pass,$db2,3306,NULL,MYSQLI_CLIENT_SSL);

/*checks if the db is connected or not.. if it cannot connect it will die and return an error*/
if(!$db)
{
    die ("Could not connect");
}
?>
```

Image 144: config.php code for connecting to the database.

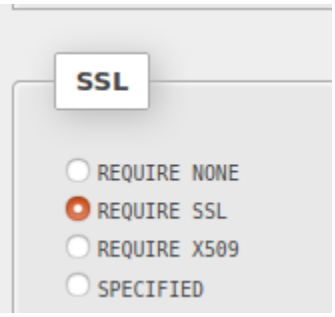


Image 145: Selecting to require SSL for user authentication

```
mysql> SHOW GLOBAL VARIABLES LIKE '%ssl%';
+---------------+----------------------------+
| Variable_name | Value                      |
+---------------+----------------------------+
| have_openssl  | YES                        |
| have_ssl      | YES                        |
| ssl_ca        | /etc/mysql/ca.pem          |
| ssl_capath    |                            |
| ssl_cert      | /etc/mysql/server-cert.pem |
| ssl_cipher    |                            |
| ssl_crl       |                            |
| ssl_crlpath   |                            |
| ssl_key       | /etc/mysql/server-key.pem  |
+---------------+----------------------------+
9 rows in set (0.04 sec)
```

Image 146: Certs set on MySQL

Test Plan:

This audit will make sure that all security features defined in security plan are implemented and that the website is secure against common vulnerabilities found in web applications.

I. The first security measure to test is for web hosting.
   a. This section will include making a checklist and manually checking these configurations to make sure they are implemented.
      - Manually checking the number of users with access to the server. Nagios was set up to make sure only one user is able to access the server. If a user other than me tries to get access, then Nagios will indicate it on the Nagios GUI by displaying yellow warning. I plan to test this by setting Nagios to allow no one access and checking the interface. I cannot test if it will email because it messes it up if I put the threshold at zero.

- Manually checking firewall by using the command line and typing ‘sudo ufw status’ will show if the firewall is in active mode or not.
- Running ‘sudo apt-get update’ on the command line to make sure all repositories are up to date. This will determine if apache server, MySQL, and PHP are up to date, if they are then there would be nothing to worry about.
- Manually check to make sure there is an access control policy that states password expiration and password complexity. Use KeePass to generate complex passwords and store them in a secure place.

II. The second security measure to test is database.
    a. This section will involve manually checking configurations to make sure they satisfy security needs.
        - Manually checking users’ passwords, permissions, password expiration, and host access to make sure it is secure.
            ⇒ Logging into MySQL database ‘mysql -u *username* -p’
            ⇒ Once logged in ‘SELECT User, Host, Password, password_expired, password_last_changed, password_lifetime FROM mysql.user;’ This will show the user with their passwords, the host they have access (should be local host), if the password has expired yet, when it will expire, and when the password was last changed.
            ⇒ ‘SHOW GRANTS FOR ‘user’@’localhost’; This will show the permissions that the user has.

III. Checking the security for PHP and webs security
    a. This section also involves manually checking to make sure it is configured properly.

- Checking to make sure that errors are properly handled to prevent information from being exposed.

  ⇒ Insert incorrect data into the different input fields and making sure that the errors that come back will not reveal information.

- Manually checking files that the website uses and making sure they are all in the same directory (bookstore) and not in different directories.
- Manually checking the code to make sure that input fields are taking that information and storing it to the database instead of a web directory.

IV. Checking the security of the site against common vulnerabilities.

a. This section involves checking to make sure that the site is secure against cross-site scripting.

- Once logged into the site insert into the street address and any other place that accepts character values

  ⇒ “<script> alert("HACKED") </script>;”

  ⇒ <body onload=alert(‘something’)>;

  ⇒ <b onmouseover=alert(‘XSS testing!‘)>

  ⇒ <script type=”text/javascript”> var test=’../example.php?cookie_data=’+escape(document.cookie); </script>

  ⇒ http://testing.com/book.html?default=<script>alert(document.cookie)</script>

  ⇒ %3cscript%3ealert(document.cookie)%3c/script%3e

  ⇒ http://www.testing.com/test.asp?pageid=2&title=Testing%20Title

b. Unfortunately, there is no way to test against a DoS attack, but Nagios is set up to detect it and send out alerts.

- A couple things to look for is to make sure the firewall only allows necessary ports. This can be done with ‘sudo ufw status’ to determine only necessary ports are opened.

c. This section checks to make sure the website it not vulnerable to SQL injections.

- Manually put in injections into input fields.

  ⇒ ‘ or 1 = 1;--

  ⇒ “ or 1 = 1;--

  ⇒ ‘ or ‘abc’=’abc’;--

  ⇒ ‘ or ‘ ‘=’ ‘;--

  ⇒ ‘ or 1=1; drop table Users;--

  ⇒ ‘

  ⇒ “

  ⇒ www.localhost.com/bookstore/...=’

  ⇒ www.localhost.com/bookstore/...=’ or 1=1;--

d. Files are not able to be uploaded so malicious file execution, remote file inclusion, and local file inclusion are not a vulnerability to this site.

Test Results:

I. The results of web hosting security testing:

a. Upon manually checking the users who have access, only one is allowed at a time on the server. Upon changing the allowed users from one to none, Nagios put in a

warning changing from green to yellow. To indicate that the number of users on the server is over the defined number allowed.

b. Typing in the command to check the firewall it states the status is active. The default conditions to deny incoming, allow outgoing, and disabled routed. The allowed ports are:

    22/tcp

    80,443/tcp (Apache Full)

    25, 80/tcp (Apache)

    3306, 22/tcp (v6)

    80,443/tcp (Apache Full (v6))

    25 (v6)

    80/tcp (Apache (v6))

    3306 (v6)

c. After running the update command everything needing updated was updated.

d. The access control policy is present, and KeePass is active.

II. The results of database security testing:

a. Logged in to the MySQL database and pulled up the users for the database. The user ‘ekbaker’ only has delete, update, select, and insert permissions. In addition to that it says the password_lifetime column is set to 180 days, the password was last changed March 20, 2021, and the host is localhost. When looking at the root user they have a changed default password, and the host says localhost.

III. The results of php and web security testing:

a. When inserting incorrect information into the text field a set error occurs.

i. Signup Errors for full name text field:
    1. “Please type at least 6 characters.”
    2. “Please type your full name”
    3. “Please only use Upper and Lower case letters”
ii. Signup Errors for email text field:
    1. “Please type your email address”
    2. “Please enter correct email address”
iii. Signup Errors for phone number:
    1. “Please user real phone number”
iv. Signup Errors for Password:
    1. “Please type at least 7 characters”
    2. “Must be between 7 to 14 characters. Only underscore allowed for special character. Must start with a letter.”
v. Signup Errors for Password Confirmation:
    1. Passwords do not match
vi. Login Errors for email text field:
    1. “Please enter an email”
    2. “Please enter an email address”
vii. Payment Errors for Address text field:
    1. “Please type an address”
    2. “Please only use numbers, letters, period, or hyphen”
    3. “Please type at least 10 characters.”
viii. Payment Errors for city text field:

1. “Please enter a city”
2. “Please only use letters and spaces”

ix. Payment Errors for state text field:

1. “Please enter a State.”
2. “Please enter state abbreviation”

x. Payment Errors for zip text field:

1. “Please enter a zip code”
2. “Please enter a valid zip”

xi. Payment Errors for name on card text field:

1. “Please enter name as seen on card”
2. “Please type at least 6 characters”
3. “Only letters and spaces allowed”
4. “Only letters, spaces, atleast 6 characters but no more than 50 allowed”

xii. Payment Errors for credit card number text field:

1. “Please enter a card number”
2. “Must be valid 16 digits”

xiii. Payment Errors for exp month text field:

1. “Please enter the expiration month”
2. “Please enter valid month”

xiv. Payment Errors for exp year text field:

1. “Please enter expiration year”
2. “Year formatted yy”

3. “Please enter a valid year”

xv. Payment Errors for cvv text field:

1. “Please enter cvv of your card.”
2. “Valid 3-digit cvv”

b. Upon manually checking where the files are located, they are all placed in a folder in a single directory that a user needs administration access to get into the /var/www/ directory.

c. All input that a user inserts gets client-side validation to check for errors then it is taken to server-side to get sanitized before being put into a database that uses a user with limited privileges.

IV. Results after testing for common vulnerabilities:

a. On the sign in page if a SQL injection is inserted into the email an error will occur and when they press submit the user will be redirected to a new login page to enter information. If a SQL injection is inserted into the password and login is attempted, it switches to the signup page.

b. On the sign in page is a XSS injection is inserted into the email an error occurs and when the login button is submitted it redirects to a new login page to enter information. If this is inserted into the password field and login is clicked the form switches to the sign-up page.

c. On the sign-up page if a SQL injection is inserted into the username field they will be given and error and if they try to sign up, they will be redirected to a new signup page. This occurs in the email field, phone number field, password field, and password confirmation field.

d. On the sign-up page if a XSS injection is inserted into the username field they will be given and error and if they try to sign up, they will be redirected to a new signup page. This occurs in the email field, phone number field, password field, and password confirmation field.

e. When attempting to put in a SQL injection in the web browser when I put https://localhost/login1.php= the SQL injection and presses the enter button the page displays “Not Found: The requested URL was not found on this server. Apache/2.4.29 (Ubuntu) Server at localhost Port 443.’ When attempting to put in a SQL injection in the web browser when I put https://localhost/login1.php/theSQLinjection the sign in and sign-up pages are displayed without the css and javascript present. It does not allow the user to click on to any buttons that are shown. It is like a bare minimum page that does not have any function.

f. When attempting to put in an XSS injection in the web browser when I put https://localhost/login1.php= the XSS injection and presses the enter button the page displays “Not Found: The requested URL was not found on this server. Apache/2.4.29 (Ubuntu) Server at localhost Port 443.’

g. On the payment page if a SLQ injection is inserted into the address field an error occurs and when pressing the link to continue to check out the page will be redirected to a new payment page. This occurs with the city, state, zip, card name, card number, expiration month, expiration year, and cvv also.

h. When a SQL injection is inserted into the web browser on the payment page the same “Not Found: The requested URL was not found on this server. Apache/2.4.29 (Ubuntu) Server at localhost Port 443” page occurs.
i. On the payment page if a XSS injection is inserted into the address field an error occurs and when pressing the link to continue to check out the page will be redirected to a new payment page. This occurs with the city, state, zip, card name, card number, expiration month, expiration year, and cvv also.
j. When attempting to put in a XSS injection in the web browser when I put https://localhost/login1.php= the XSS injection and presses the enter button the page displays “Not Found: The requested URL was not found on this server. Apache/2.4.29 (Ubuntu) Server at localhost Port 443.’
k. Just like I stated previously with the firewall, the ports open are those that are needed for this site to operate:
    i. 22/tcp: Used for SSH to operate securely.
    ii. 80, 443/tcp, to access http/https
    iii. 25: used to be able to use email.
    iv. 3306: Used for MySQL

Challenges Overcome:

Certificate creation was one of the biggest problems I had. Since this server was a VM setup for this project. The VM was not setup on a domain and there was only one VM used.. The majority of the tutorials I found online were not intended to work with this particular configuration in mind. Most of the tutorials were meant to have at least two servers one for apache web service

and with php website running on it. The other server was meant to have MySQL database platform setup on it.

PHP I discovered does not have the capabilities to use certificates for authentication from php website to MySQL database. Here is some of the workarounds I had to implement to get this to work:

- Use 127.0.0.1 instead of localhost when calling server from either code or configuration in a file.
- Since self-signed cert was created through openssl (it is free and I am on limited budget). Then the self-signed client and server certificates have to be merged with the ca.cert. That way the client will trust the server certificate and the server will trust the client certificate.
- In the php database connection file since this was localhost configuration I had to use only the ca.cert to authenticate to MySQL for it to work. I figured this out after many hours of trial and error with different configurations and reading allot of website post that talk about php and MySQL on localhost not working and reason why.

Sendmail was the next problematic problem to get working. I have never worked with sending email from a server before and had no clue where to even begin. After quite a bit of research on the different types of email setups that can be configured on a server. I decided to go with Sendmail, because it is light weight and doesn't require allot of setup. Most of the other relay email setups required that I have outlook exchange setup or setup an email provider on the server. Sendmail can use many popular email providers such as Hotmail, Gmail, or yahoo. I found several post on how to set this up but ran into few e problems as I worked through the setup.

- Errors were generated when I tried to send out emails because server was not in domain. Implemented a workaround that allows the Sendmail to think the server is on a domain when it is not. Adding entries to the /etc/hosts with loop back IP address and with a made up domain name. Then changing the name of the server to include a domain that was made up and added to the hosts file by editing the /etc/hostname.
- Another error that was generated stated either server or email is not authorized to send emails. To correct this error /etc/mail/local-host-names file will need to be modified to add entries for php, emails used to send email to, localhost, the server's name and made-up domain used on this server to be allowed to send emails from mail.
- If you send to another email address while using Sendmail other than the one specified in the gmailinfo file that used to send from. Gmail will block this email from being sent. The reason for this is Gmail security settings does not allow third party apps from sending emails through the Gmail relay server. The work around for this is either turn off the security on Gmail that blocks the use of third-party apps which, will open up security problem on your Gmail account. The other you use one email account to send and receive email that you specified in the gmailinfo file. For this project the workaround was a good compromise since it did not compromise the security of my email account.

Nagios was huge learning curve. It was a new fun experience learning how to setup or use this monitoring program. Online I found quite a few websites that documented how to set it up and use many of the plugins that comes with Nagios. The actual setup was quite easy but the configuration Nagios config files to use plugins, allow Sendmail to send emails, and monitor the local host was new challenge in itself.

- Learning how all the different underlining files work together and what needs to be modified in the files to get the plugins to work on the Ubuntu Linux VM.
- Altering files to allow Sendmail to work with Nagios.
- Setting up email within Nagios so the alerts could be sent to admin email.
- Altering files for plugins to customize monitoring for the local host and the settings I wanted to monitor for.
- Learning how to use Nagios GUI to monitor server

I put in many hours on research of how to secure php website through code. In the process I learned quite a bit about how the site just does not need secured through use of certificates or encryption from php to MySQL but through code itself. Through the user of server's side and client side validation. How you make your connection through the code helps avoid many malleolus attacks used by unscrupulous individuals.

- Using regular expressions to validate what the user can input into the text field in the form. Allot of trial-and-error to get the code to validate exactly what I was wanting the field to be able to post to the database and block any other input the user might decide to input into the field.
- Using mysqli_stmt_bind_param limits the of values that can be put into your query statement. Learning the use of this and different options that go with this type of connection.
- Cleaning up old code and pages no longer needed to lessen the foot print the attacker has to gain information about the code or database. This always makes me nervous when deleting code or files. I am always paranoid I will break code I worked so hard get working.

- Hashing values in the database such as passwords and credit card numbers. Learning how use the hash function and how to pass the variable holding the hashed value into the query without getting an error.

Future Enhancements:

Since this project is about securing a website there needs to be security audits not just after setting up the website but afterwards. Automating that process would be a better option then manually checking since that would take a lot of time, especially the bigger the company gets. Additionally, using a vulnerability scanner that can detect places that are vulnerable would be a good tool to put in place. In this project I used md5 hashing and it would be better to switch from md5 to sha26 hashing. Making all data whether in transit or at rest is encrypted in case of a possible breach.

Since this website collects sensitive information like credit card information the site would need to use a card processing business, this can transfer risk with credit card information. This would need to be researched more to make sure whatever business that would be used is reliable. Since this project was more about securing a website then making sure it is compliant with regulations, it would be a good idea to go over the PCI DSS to make sure this site complies with those standards.

When it comes to monitoring the network and servers it would be a good idea to use an enterprise version of Nagios instead of the free version since they cover more monitoring services. Additionally, I did not install Anti-Malware since I do not trust the free version since they could have hidden malware in them. It would be a good to purchase antimalware from a reliable company that could help with monitoring. Also,

setting up an intrusion prevention system along with my firewall would give more protection.

This site uses self-signed certificated but it is not normally used in a real working environment. Due to limitations that was the best I could do to get as much security for the site as possible. In a normal situation a CA would need to be acquired to get certificates instead of using self-signed certificated. Another change that could be made is using the IP address that is taken from the user to verify if this is an IP address they have used before to sign in. If not, then sending an email to the user to determine if it is them signing in or not.

For password expiration of users on the database I changed their expiration individually but if the number of employees increases this can become cumbersome. Making a global rule for the password expiration would be more ideal so that when users are added this is automatically assigned to them. Additionally, implementing password reuse policies to prevent users from being able to use their last 5 passwords. Another change to the database users is creating failed login attempt logs and password lockout after so many times of trying.

On the website a couple things to change that would make the experience for the user better is creating a dropdown menu to select state instead of inserting abbreviations. Creating an administration page for admins that would allow them to change the order status from ordered to completed, making them not have to go into the database to change it. The user being able to put a quantity if they decide to order two of the same books. Inserting a search bar for when more books are added to make it easier for users to search for their books. Then separating the payment server-side validation from the main

payment.php display page and the navigation bar being in its own file to decrease the amount of code in the files index.php, cart.php, payment.php, and successpay.php.