Aditya Varkhedi
5/23/2016
CSC 492
Senior Project Paper - CampusBookShare

1. Introduction

Every few months, students in the US and abroad face the expensive and time consuming task of buying textbooks for the classes they are enrolled in. Despite the advent of the internet and social networking websites the ability for peer to peer textbook exchange is still limited. There are few if any dedicated websites for general purpose peer to peer textbook transfer for every university and college.

The project I have created for my Senior Project is a peer to peer solution for buying and selling textbooks on US college campuses. The underlying idea is to create an authentication system that allows users attending US based universities to create accounts with their user names being their assigned ".edu" email addresses. I have decided to name the website/mobile application I create "CampusBookShare".

To begin, students must first complete the registration process by entering their ".edu" email address and a personal password. At this point an email is sent to their ".edu" email address, and the authentication process can begin. Users must then log into their email address and then click on the email sent to them. Inside the email a link back to CampusBookShare.org (the website) will be present. By clicking this link, a cookie will be attached within the web browser, and then the user will be routed to the CampusBookShare website. At this point they have completed the authentication process and can now share books between one another within their own school. By having users confirm their accounts by logging into their .edu email address and clicking the "confirm account" link, three different categories can be reasonably satisfied. The three categories that are satisfied are locality, safety and similarity. By having users create accounts with their ".edu" email addresses, and allowing them to buy and sell textbooks between others with the same ".edu" extension (for example, everyone with a ".calpoly.edu" email account can trade books with everyone else with a ".calpoly.edu") we can ensure that:-

a) locality – the two people involved go to, or at least went to, Cal Poly and therefore are in the same locality of each other.

b) Safety and automatic vetting - Ensuring that both people involved are students/and or alumni, we can guarantee a certain amount of safety and automatic vetting of both the buyer and the seller, and

c) Books prescribed by the school curriculum - By ensuring that both the buyer and the seller attend the same school, the books and products they sell or buy on the website are more fine tuned towards certain specialized courses and professors at Cal Poly. This means that certain courses and textbooks that are only available or used at Cal Poly are more likely to be sold on a website that caters to localized peer to peer textbook sales.

2. Detailed Requirements

We began looking at what we would have to design and develop in order to provide the service we had thought about. We wanted to provide a mobile application as well as a web application to be able to provide all the features on both the platforms.

The web application and the mobile application would allow the users to create accounts within their own university or college, communicate with others through the service after searching through books that they wanted to buy, search for specific books, view all the books at a university or college, upload their own books, and logout of the service.

A special distinction between the web service and the mobile application would be the integration of the Zebra Xing 3rd party library barcode scanner which would be available on the mobile application and could be used by clicking the “Barcode Scan” button on the “Sell” tab in the Android application which will be discussed in more detail further down.. This would allow users with Android mobile phones to scan a book's barcode by simply hovering over it with their built in camera, and then querying a Google Books OATH link to receive all the information about the book simply based on the ISBN number that the barcode scanner would return. All this information would then automatically populate all of the fields related to the book, such as “Title”, “Author”, “ISBN” etc. It would even return an image of the book as a bitmap. At this point the user could simply enter in an asking price, and a book condition value and then click “Sell” which would upload the book to the database and allow other users to view it within their searches. A portion of the Google Books OATH code would be rewritten in PHP to allow for the web service to be able to autopopulate all the fields about the book by entering in the ISBN on the website as well. This feature would be labeled as an “ISBN Autofill” button on both the mobile application and the website.

3. Application Architecture and Development Environment

For the backend, I decided to use the LAMP stack. The decision to go with the LAMP stack was because it provides an easy way to develop backend servers and also provides a simple integration between the web application and the backend database. The LAMP stack also provides an easy way to scale the application. The LAMP stack has the php server that can be used and has the MySQL database where we can persist the data for the web application as well as the mobile application. The same data that is served to the web application is also served to the mobile application through http “GET and “POST” requests. The data presented to the user from the “GET” and “POST” requests is JSON data. This data is consumed both by the android and the web application.

For the mobile application I decided to build an Android Application since I had dabbled in it before and had the Adroid SDK installed with IntelliJ on my machine. The application would communicate with the php server on the LAMP stack.

Some of the work that I completed or turned in as my Senior Project was a cooperative effort with a co-developer of mine, who developed much of the Server Side scripts and Database tables. To begin with my part, I decided to start writing a basic Android application. The next few paragraphs will detail those efforts.

The following figure shows the architecture diagram for the service to be provided. The users are supported on cell phones running the android operating system and the users can also use the application from any web browser.

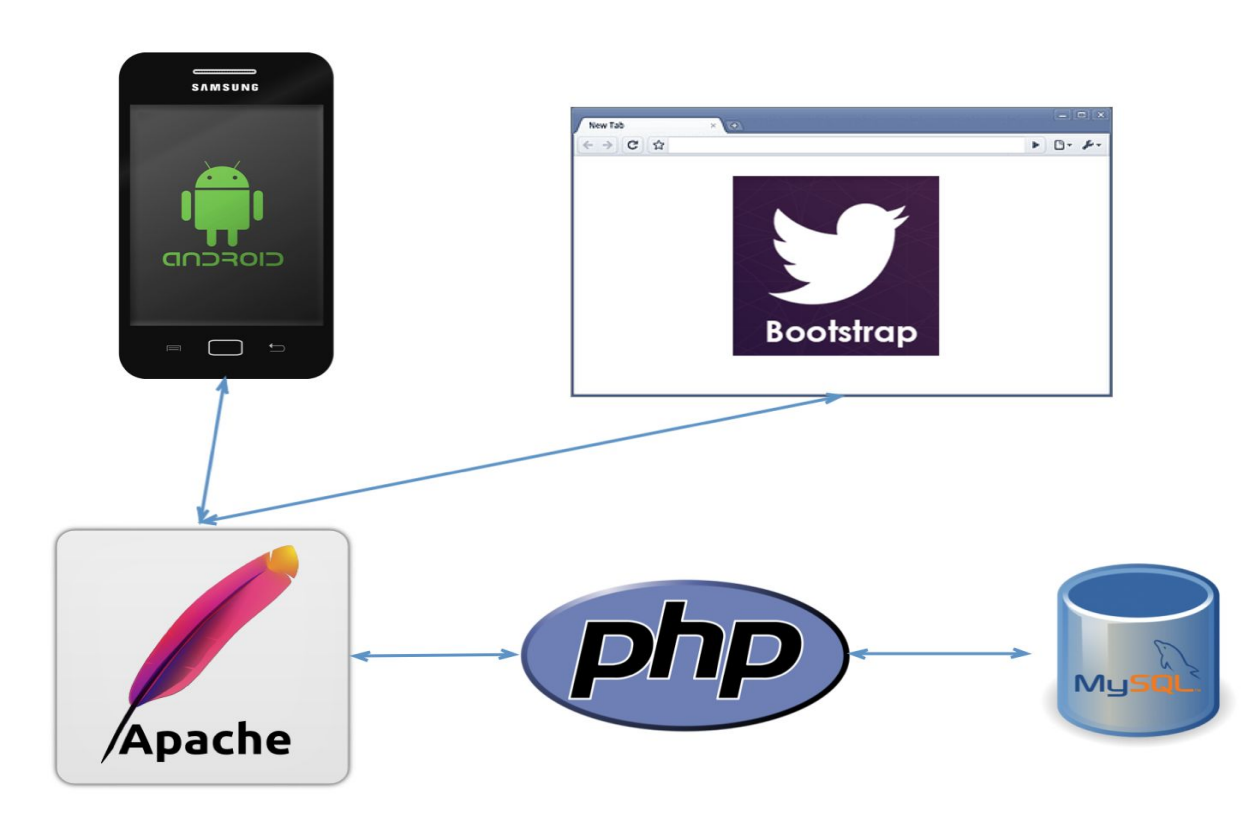


Figure 1: This is an architecture diagram of the senior project with all the elements to be created.
The top two images are the front end architecture elements which are the Android and the Bootstrap website, the bottom three are the backend servers.
References:
Android
https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcT6vEZ2F-GVFFmWWuhx8zTL5KySG0oWmhvuQYaV6KyUro1zgvrP
Bootstrap
http://syntaxxx.com/bootstrap-101/
Apache
http://www.wired.com/2012/10/apache-dell-and-calxeda/
PHP
http://opendatakosovo.org/training/php-the-right-way/
MySQL
http://imasters.expert/mysql-replication-in-5-minutes/

The backend is a apache2 server with the web application being served by php. The database is a relational database that persists all the data for the application.

For hosting the application, we used Amazon AWS and deployed a LAMP server. The application code for the mobile application was developed using intelliJ IDEA with the android SDK from Google.

We designed the database and deployed it using phpAdmin portal that comes with the LAMP stack. A detailed description of the database and phpAdmin follows in later sections. The webpages were developed using the bootstrap framework.

Manual testing was done on the pages in the web as wells as the mobile application. The emulator provided by the android SDK was used to test the android application.

4. Android Application

I decided to use an older phone, as well as an older version of Android to allow me to develop easier to get a basic look and feel for the application. I started out by creating a basic login page in xml (the markup language) that Android uses. I learned how to create GUI elements on a page and hashed these elements together into one clean page. It included a username field, password field and a login button. I also took the liberty of creating a Cal Poly themed logo with Green and Gold colors. Figure 1 shows the result.

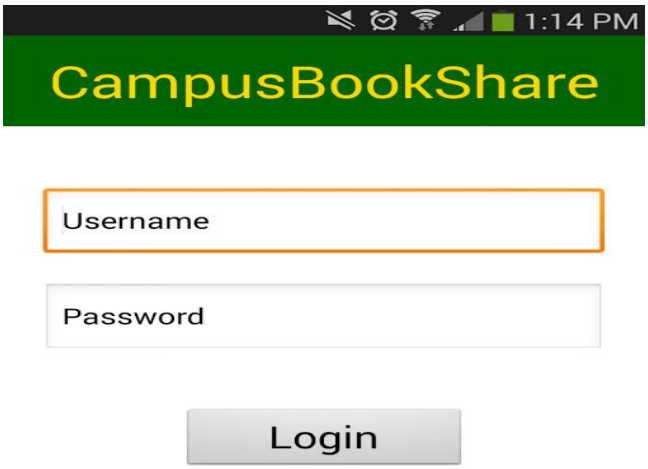


Figure 2: Basic login screen that allows the user to enter in their credentials and sign in

A common theme that was used on both the website and the application was to create a simple user interface as well as advanced features to easily upload new textbooks to your “My Bookshelf”. This bookshelf is the list of books that other people can contact you for if they are looking to buy. But first to add a book, I created a page that pulls information using a Google Books API Oath named ISBN Autofill which retrieves information about a book simply by the user typing in the books ISBN number. This immediately allows all the other fields to be filled in like so. The ISBN autofill even retrieves a Google Books image of the book. I named this page/tab the “Sell” tab. In it I allow for another feature which is the 3rd party API based “Barcode Scan” which allows the user through my application to access the phones camera, and automatically identify the book's barcode on the back and then perform the “ISBN Autofill” programmatically based on this information. The result is detailed in Figure 2.

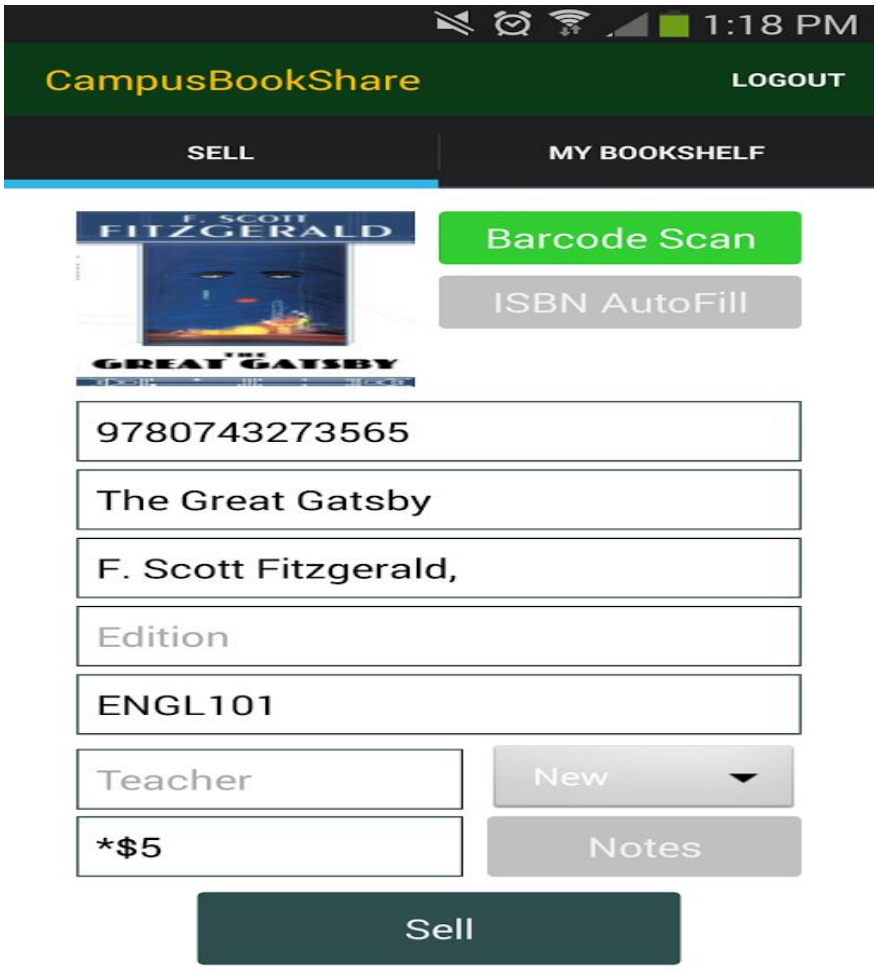
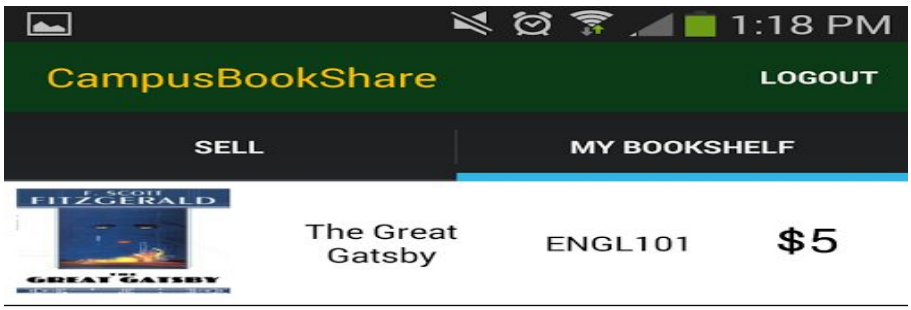


Figure 3: User can either scan a book using “Barcode Scan” which will automatically update all the fields on the page, OR the user can enter the ISBN number in the first field and then click “ISBN AutoFill”, OR the user can manually entire in all of the fields. The user must still enter in the “Condition” spinner, “Notes” field, and “Price” Field, as well as “Teacher” and “Class Used in”. Once the user enters in all of the relevant information the user can then click “Sell” which will communicate with the server and update the “My Bookshelf” list.

The “My Bookshelf” tab took considerable effort because a custom view adapter that allowed users to click on individual books in their bookshelf had to be created. This view adapter also had to allow the user to be able to view this information separately as a Modal Popup UI element. The “My Bookshelf” tab also took considerable effort because of the lazy loading scheme I had to implement, that allowed for constant scrolling to the bottom of the page, where more books were loaded if they were not already preloaded in the applications local cache. I set

limits to the number of books that were preloaded to reduce memory consumption and network bandwidth

Figure 4 shows the result of a basic “My Bookshelf”, it is the result of the previous Sell screen's



“Sell” button click.
Figure 4: This is a basic “My Bookshelf” with one book added to the server. The user can click on each book.

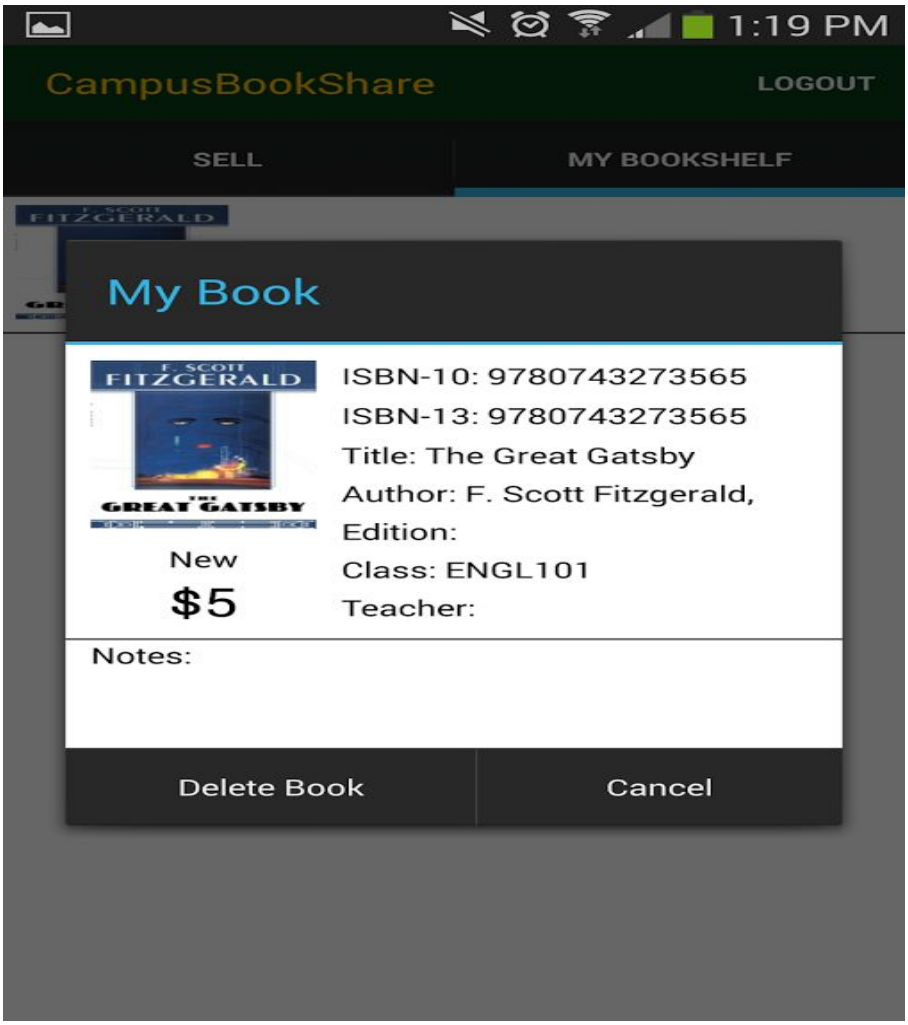


Figure 5: This is the popup modal that appears when a user clicks on a book. Notice the “Delete Book” and “Cancel” buttons that allow you to exit out of the screen.

5. Server PHP Pages

The server code was considerably more complex to write in my opinion. Although much code was already written by me and my partner in the past, I had to read up on PHP and the way it worked, and I also had to gain a basic understanding of creating MySQL queries. Luckily, LAMP provided me with an excellent interface for creating server side code, it incorporated a MySQL automation tool called phpmyadmin which allowed me to easily create tables, format them, and call SQL statements.

Below are two screenshots taken of the phpMyAdmin program that show the different databases I have created in the program, one of them which is the CampusBookShare database.

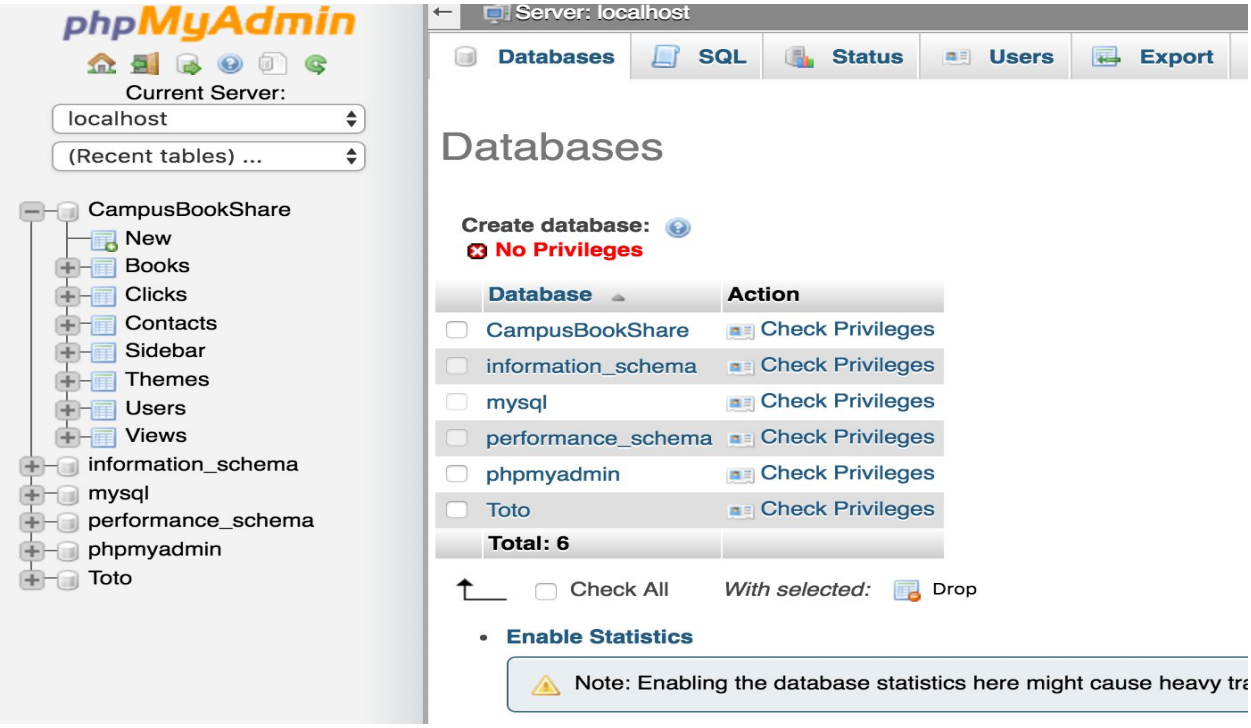


Figure 7: Shows some different test databases and the CampusBookShare database first

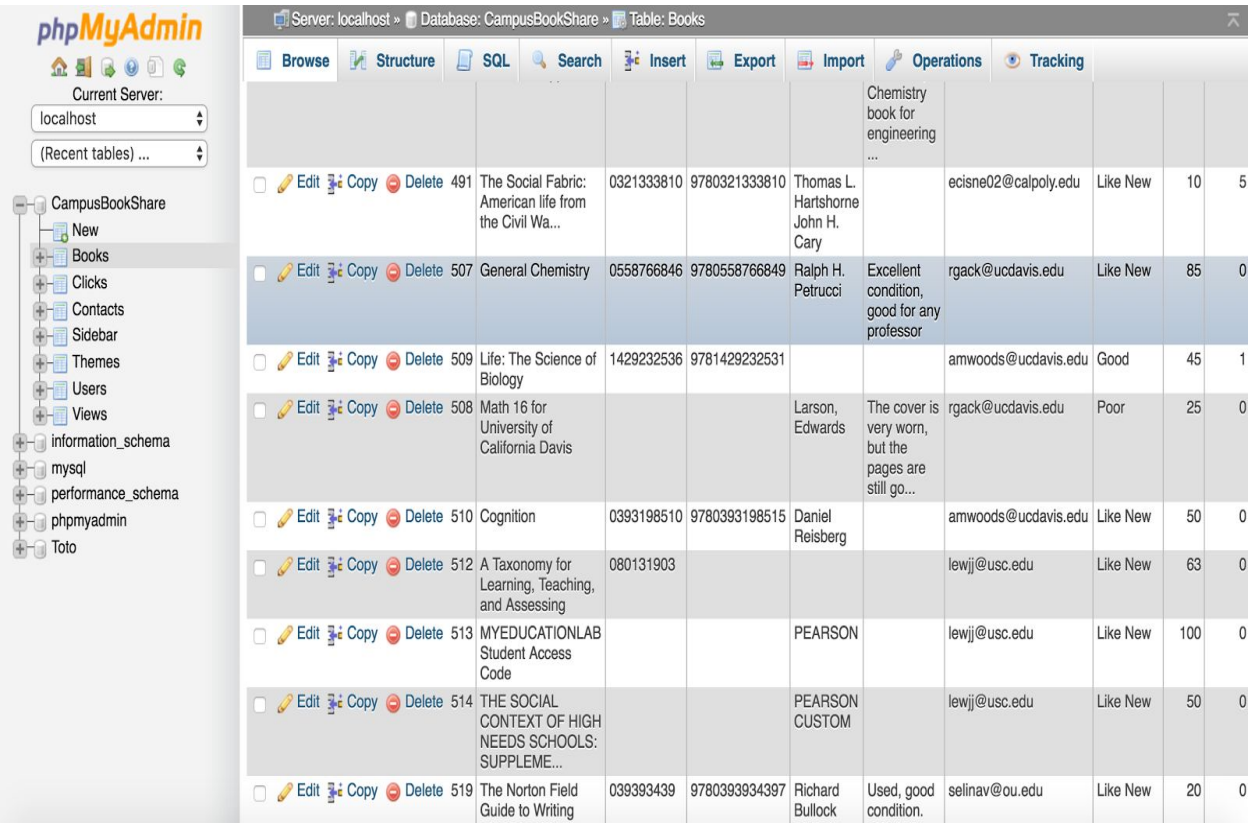


Figure 8: Shows the contents of the “Books” table in the CampusBookShare database.

The database contains the following tables which store information about the various entities needed to be persisted for the service. The tables are the following:

Table Books: Contains book attributes like “Title”, “ISBN”, “Author”, “Price”, Book Condition etc. It also has a link to the image of the book.

Table Users: This table stores the user information.

Tables Themes: This table stores the customization color schemes for each University involving that universities’ color scheme (e.g. green & gold for Cal Poly, SLO). This is for the user from a specific school. The color which includes all the layout formats I made for the webpage and application that need to be dynamically loaded.

Table Views: This table has the statistics on which record the “views per page”

Table Clicks: This table stores the stats for the number of clicks for each page

Table Sidebar: These are the images for advertisement (Not implemented).

The server side scripting code consists of PHP pages that handle different requests from both the mobile application and the website. The main PHP pages I wrote are “login.php”, “createaccount.php”, “home.php”, “viewbookshelf.php”, “contact.php”, “logout.php”, “viewallbooks.php”, “postbook.php”, and several others that were needed to handle all the requests. There are several other helper or utility pages that needed to be created.

The description of the pages are below:-

Createaccount.php: The user registers to the service by using his or her “.edu” email address. The registration page for both the Android application and the web page send a POST request to the PHP server with the email address and other user attributes including the password. On the backend, the PHP page sends an email with a link to confirm the registration. The “confirm registration” link then completes the registration and marks the user as an active user. Once the user is registered, they can list books to sell, or search specific books to buy, only from their classmates. This page writes the user information to the “users” table in the database, including a MD5 hashed password which is used to validate the user on login.

Login.php: Authenticates the user from the mobile application and the web app in this file. Before submitting the request to the server, in the android application we MD5 hashed the password field, so that the information would be secure in transit used SSL encryption. The hashed password is compared with the stored hashed password in the User table stored in the CampusBookShare database. The page returns an “Invalid Login” message to the user if the hashed passwords do not match. The MD5 hashed password is retrieved from the user’s database to validate the user.

Home.php: This is the home page for the application.

Viewallbooks.php: The user can view all the books that are listed to be sold for his college.

Viewbookshelf.php: The user can view all the books that are listed by him on the website.

Searchbooks.php: The user can search for books with a certain attributes such as ISBN, Author, Name etc.

Contact.php: Users can communicate with other users registered to exchange information through the website.

There are several other PHP files that we had to code to support the various interactions with the user.

6. The Web Application

The following are some sample pages from the web application. The pages were built using Bootstrap. The following page is the home page on which the user logs in or registers (Figure 9):

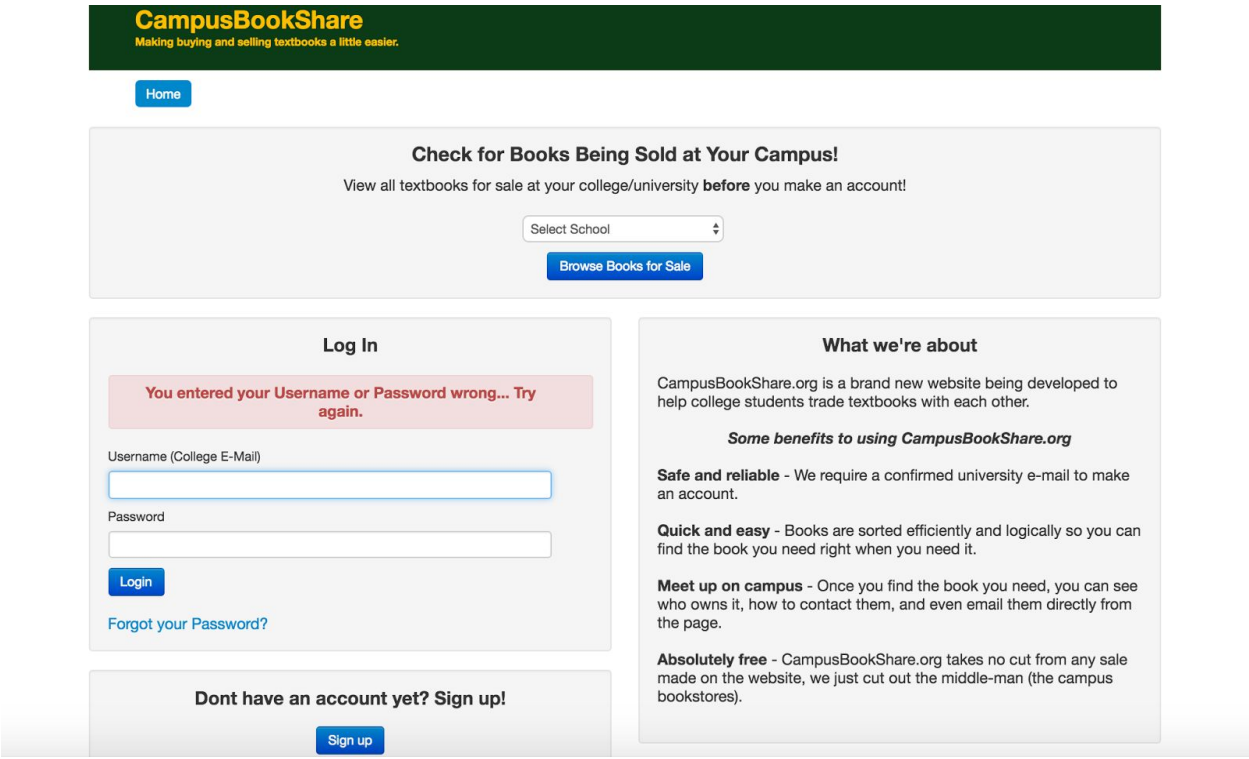


Figure 9: Shows the login page for the Bootstrap website that allows users to enter their username and password and login, or create an account by clicking “Sign Up”. The user may also “Browse Books for Sale” at each individual listed university.

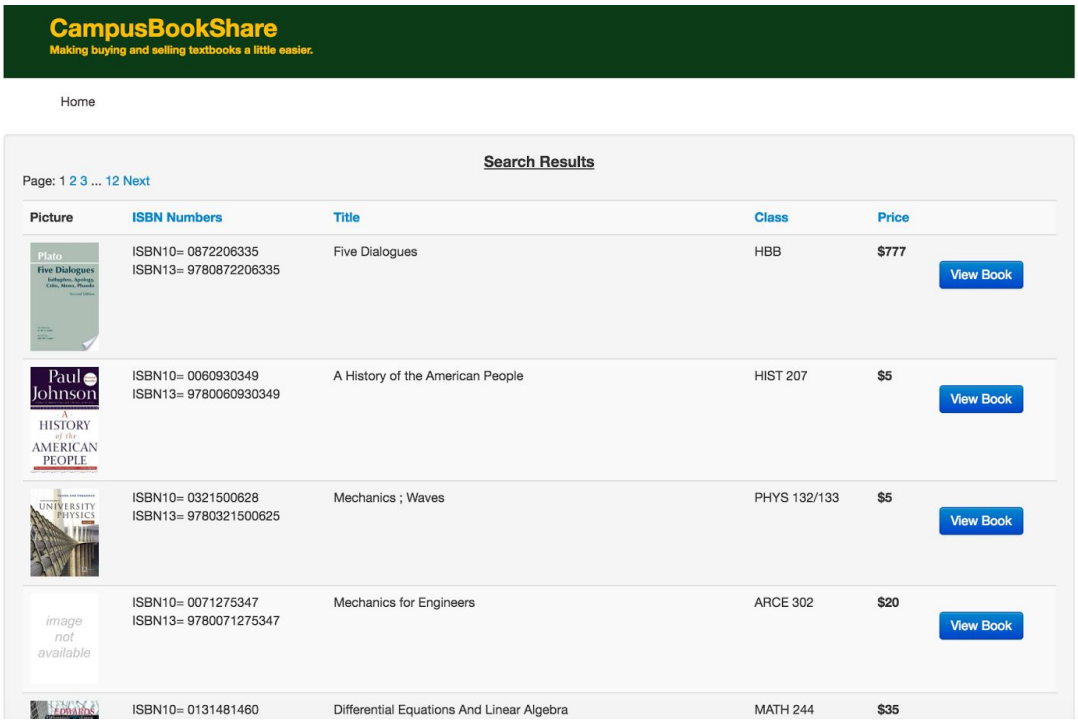


Figure 10: This shows the result of clicking “Browse Books for Sale” on the image above.

The “Themes” database allows the pages to be customized for every school that is supported by the application. This gives a look and feel of their school to the user. The above pages are with the “CalPoly SLO” theme. If a theme is setup for a school depending on the email address, the school's theme is determined and the subsequent pages all appear to be customized with the colors of that school. The differences between the application and the mobile application were listed as above with the biggest being the Zebra Xing 3rd party Barcode Scanning feature.

7. Conclusion

This senior project gave me a lot of opportunity to learn new technologies that I was not exposed to before. I was able to write web applications and mobile front ends using PHP and the Android SDK. As a part of the project, I was able to pickup SQL and database technologies. I was able to understand and use http “GET” and “POST” APIs to move data from the backend to the applications. I am still implementing some of the feature/functions that are needed to deploy this service for general use.

Using the cloud for deploying the application was also a challenge and a new technology exposure. I learned how AWS works, how to set up a Virtual Machine, and how to set up a

server on this platform, and complete the necessary networking steps to ready a webserver for use.

8. Appendix

The following shows some of the sample code for the server as well as the Android application which logs the user into the application and it communicates with the given url, returning a JSON Object that is used in the phone applications SharedPreferences memory as a cookie.

```
private String authenticate(String url)
{
    String authentication = "";
    try
    {
        HttpClient client = new DefaultHttpClient();
        HttpPost post = new HttpPost(url);
        HttpResponse response = client.execute(post);
        HttpEntity entity = response.getEntity();
        InputStream is = entity.getContent();
        String result = CommonMethods.inputStreamToString(is);
        is.close();
        JSONObject jsonObject = new JSONObject(result);

        authentication += jsonObject.getString("Success");
        authentication += jsonObject.getString("ValidUser");
        userId = jsonObject.getString("UserID");
        textHex = jsonObject.getString("TextHex");
        backgroundHex = jsonObject.getString("BackgroundHex");
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }

    return(authentication);
}
```