

BOOK | Spartan

Team 25

Final Report

San Jose State University

CS 157A / Section 1

Fall 2019 - Team 25

Professor Mike Wu

Created by:

- Au Tran
- Benny Ooi
- Aldrich Mangune

Date:

12/06/2019

Table of Contents

Project Requirements	2
I. Project Description	2
II. System Environment	3
III. Functional Requirements	4
IV. Non-functional Issues	6
Project Design	19
ERD Model	19
Normalization Process	21
Table Schema and Database Models	22
Database Implementation	35
LOGIN PAGE	36
LOGOUT	38
REGISTER AN ACCOUNT	39
HOMEPAGE	43
ADDING A TEXTBOOK LISTING	46
EDIT LISTING	54
Cancel Listing Functionality	64
SEARCHCHING TEXTBOOK LISTING CATALOG	73
VIEW PRODUCT PAGE:	86
ADD TO WISHLIST	91
VIEW WISHLIST:	94
REMOVE FROM WISHLIST	97
ADD TO SHOPPING CART	102
VIEW SHOPPING CART	105
REMOVE FROM SHOPPING CART	108
View Checkout	112
Checking Payment Info	116
CHECKOUT REQUEST	119
Changing Account Settings:	137
VIEW PROFILE:	144
VIEW CREDIT CARDS	147
VIEW ORDERS	150
HOW TO SETUP ON MACOS	155
PROJECT CONCLUSION	157

Project Requirements

I. Project Description

In this project, we will be working on a three-tier database application for SJSU students to sell or buy their used textbooks. We will be developing a database for an e-commerce site for online textbook shopping service. A popular way for students to save money on course materials is to purchase used textbooks. The goal is to provide a common interactive platform for students to buy new/used textbooks or listing their used textbooks for sale. In addition, we would also like to offer another alternative for SJSU students that want to reliably sell and buy used textbooks. In this application, there will be 2 majorities of users will be using this application which is the buyer and the seller of the used textbooks. The seller can sell their used textbook at any price as they wish. An e-commerce website will provide ease of access to our products/services and satisfy the demands of our customers in a timely manner.

II. System Environment

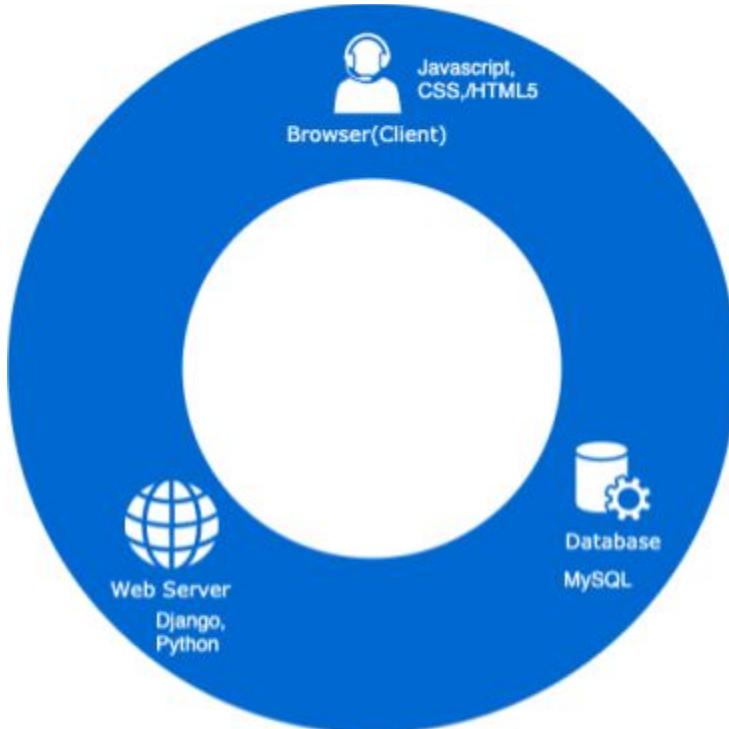


Figure 1 System Architecture

The hardware/software that was used to develop the system were Mac OSX and Windows 10 machines. The RDBMS for persistent data storage was MySQL 8.0. The structure of our system is represented by (Figure 1) above. The client interface/front-end of the system was built using JavaScript, CSS, and HTML 5. The server that we setup was built using the Django web framework for the Python language. Finally, the database tier utilized MySQL for persistent data storage. Additional application languages that was used for development were Github, [...].

III. Functional Requirements

Account Login

- Users will be prompted with a login panel and they must log into their personal account to continue with the application.

Registering a new account

- Users will also be given an option on the startup screen to register a new account. Users will submit the necessary information for account creation and use the newly created account to login to the application.
- **This will insert a new row into Account table**

Changing Account Information

- Users can change their own account information such as email, first name, last name, profile picture
- **This will update a row in Account table**

Viewing the Textbook Catalog

- Users will be able to view the entire textbook catalogue on sale. They will also have the options to order the textbooks by title, author, or price

Searching for a textbook

- Users will be able to search for a specific textbook by inputting certain information such as Title, Author, ISBN # and it will return a list view of all textbooks matching the description

Add a Textbook to Wishlist

- Users can bookmark various textbooks they are interested in for convenient access later. The list of the bookmarked textbook will appear in the Bookmark tab in their profile for them to view.
- **This will insert a new row into Wishlist table**

Remove Textbook from Wishlist

- Users can remove textbooks from their wishlist.
- **This will delete a row from Wishlist table.**

Add a Textbook to Shopping Cart

- Users can add items to their shopping cart to buy
- **This will insert a new row into Shopping Cart table**

Remove Textbook from Shopping Cart

- Users can remove textbooks from their shopping cart
- **This will delete a row from Shopping Cart table**

User profile

- Every user will have their own profile page containing their basic information and user will be able to update their own profile page. They may also be able to view other people's profile such as a seller for a textbook they are interested in buying.

Listing a Textbook for Sale

- Users can list a textbook that they have for sale. They must provide information regarding the textbook-like Title, Author, ISBN #, Price, Publisher, Year Published, a short description of the condition of the textbook, and a couple of images displaying the textbook.
- Users can also update their existing textbook listings. They can cancel these listings or update information about them such as price, conditions of the textbook, or upload new images of the textbook.
- **This will add a new row into Textbook and Listing table. Also add one or more rows into Category Has Textbook table.**

Checkout Order

- Users should be able to checkout their order and purchase the textbooks they want
- **This will add a new row into Order, Checkout, and one or more row into Order Contain Textbook table. It will also update a row in Listing table and deletes all textbooks purchased from every account's Wishlist and Shopping Cart. It may add one row to PaymentInfo and Account_Has_PaymentInfo table if user adds a new credit card.**

View Order History

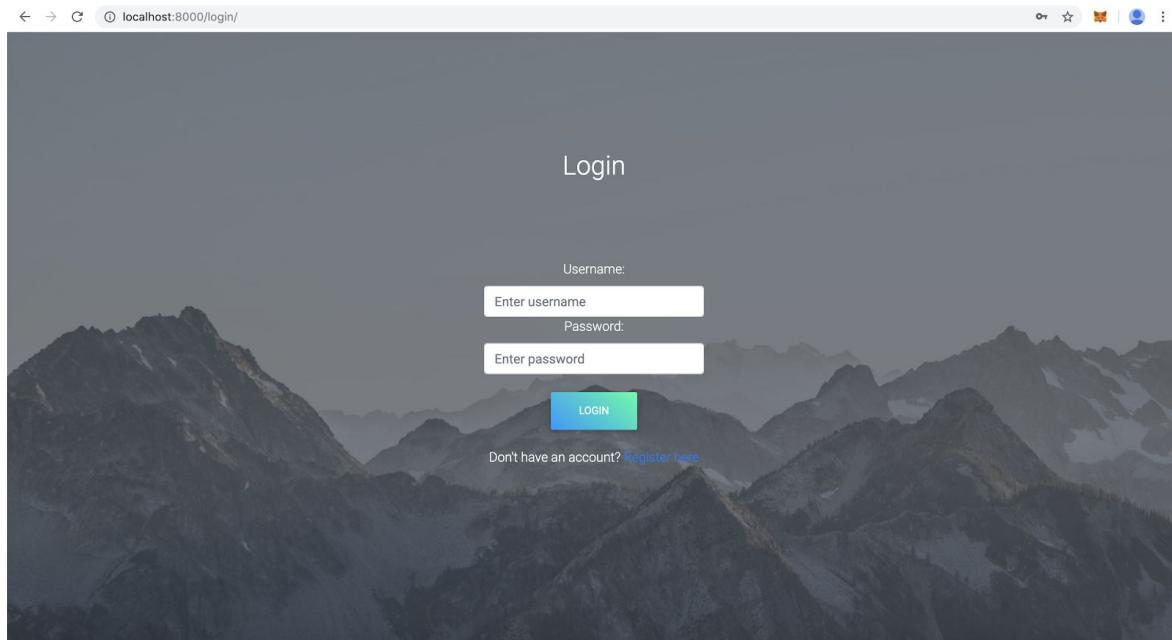
- Users can view their order history and see what they have purchased

View Credit Cards Saved

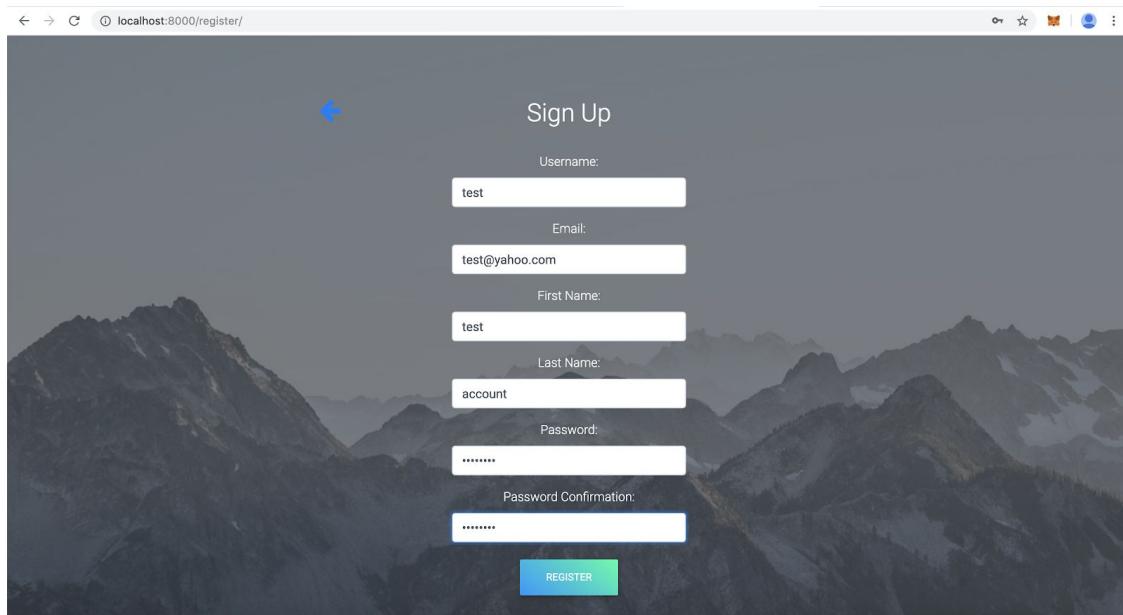
- Users can see what credit cards they have used for past orders

IV. Non-functional Issues

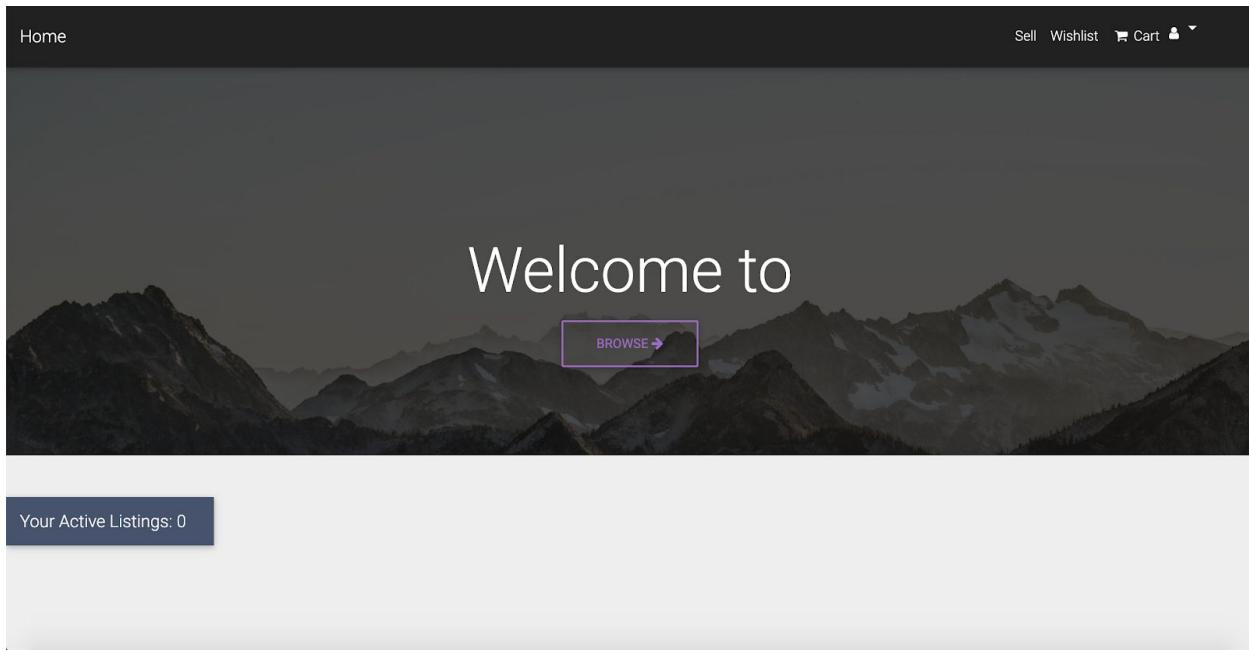
GUI



This is the default login page when you starts up the server



Registration screen for you to register a new account



The homepage screen, you can click Browse to go see the list of available textbooks for sale. If you have any active listings, they will be display on the bottom if you scroll down

The screenshot shows a user interface for adding a listing. On the left, there's a large white input form titled "Add Listing". It includes fields for ISBN (9780465060733), Title (Basic Economics), Author (Thomas Sowell), Publisher (Basic Books), Condition (BRAND NEW), Date Published (December 2014), Price (\$75), Category (with a list of options like Architecture, Communication, Computer Science, etc.), and a Description field containing "example description". On the right, there's a preview window showing the front cover of the book "Basic Economics" by Thomas Sowell, a file upload section ("Choose File Basic-Economics.jpg"), and a green "ADD LISTING" button.

On the upper right corner, you can click the Sell button to be redirected to an Add Listing form so you can input information about your textbook and list them for sale.

Home

Edit Listing

ISBN: 9780465060733 Title: Basic Economics

Author: Thomas Sowell

Publisher: Basic Books

Condition: BRAND NEW Date Published: December 2, 2014

Price: 75.0 Category:

- Architecture
- Biology
- Communication
- Computer Science
- Engineering
- Finance
- History
- Humanities
- Law
- Literature
- Mathematics
- Medicine
- Physics

Description: example description

No file chosen

If you have an active listing for sale, you can scroll down on the homepage clicks on the Edit Listing button and be redirected to a form so that you can change the information about your listing.

Home Sell Wishlist Cart

Title Search Sort by Title Desc

ARCHITECTURE COMMUNICATION COMPUTER SCIENCE ENGINEERING FINANCE HISTORY HUMANITIES LAW LITERATURE MATHEMATICS MEDICINE PHYSICS

THEORIES AND METHODS OF WRITING CENTER STUDIES
A PRACTICAL GUIDE
EDITED BY Jo Mackiewicz and Rebecca Day Babcock

By Jo Mackiewicz, Rebecca Babcock
Theories and Methods of Writing Center Studies: A Practical Guide
ISBN: 9780367188498
Condition: LIKE NEW
Listed for sale by: root
[150.00](#)

The American Sign Language Handshape Dictionary
Second Edition with New DVD
Richard A. Tennant, Marianne Gluszak Brown
Illustrated by Valerie Nelson-Metlay

By Richard A. Tennant, Marianne Gluszak Brown, Valerie Nelson-Metlay
The American Sign Language Handshape Dictionary
ISBN: 9781563684449
Condition: DECENT
Listed for sale by: millsjoseph

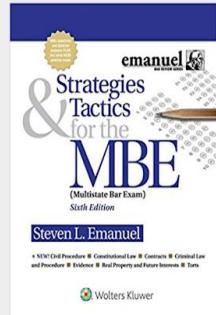
Strategies & Tactics for the MBE (Multistate Bar Exam)
Sixth Edition
Steven L. Emanuel

By Steven L. Emanuel
Strategies & Tactics for the MBE (Emanuel Bar Review) 6th Edition
ISBN: 9781543805727
Condition: DECENT
Listed for sale by: root
[165.00](#)

Modern Architecture Since 1900
PHIDON

By William J.R. Curtis
Modern Architecture Since 1900 / Edition 3
ISBN: 9780714833569
Condition: FAIR
Listed for sale by: michael24
50.00

If you click on the Browse button on the homepage, it will send you to a page where you can see and filter the textbook listings for sale. More information on the search listing functionality in the database implementation section.



Strategies & Tactics for the MBE (Emanuel Bar Review) 6th Edition

ISBN: 9781543805727

By Steven L. Emanuel

Publisher: Wolters Kluwer

Date Published: May 22, 2016

Condition: DECENT

Listed for sale by user root

Price: 165.00

[ADD TO CART](#)

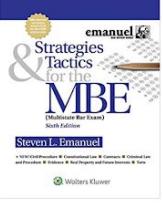


Description:

The Seventh Edition of Strategies & Tactics for the MBE has been carefully revised by Steve Emanuel and is full of up-to-date advice on how to analyze Multistate Bar Exam (MBE) questions in all MBE subject areas (Civil Procedure, Constitutional Law, Contracts, Criminal Law and Procedure, Evidence, Real Property with Future Interests, and Torts). Steve Emanuel—author of the Emanuel Law Outlines and CrunchTime books in the MBE subject areas—has passed the bar exam in several states (including New York and California) and worked with law students to prepare them for taking the MBE.

This is the view product page for a textbook listing. You can click the Add to Cart button to add the textbook to your shopping cart. You can also click the icon next to the Add to Cart button to add textbook to your wishlist

Wishlist

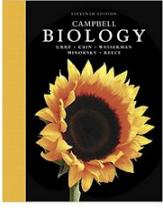
Item	Price	Total
 Strategies & Tactics for the MBE (Emanuel Bar Review) 6th Edition ISBN: 9781543805727 Condition: DECENT Descriptor: The Seventh Edition of Strategies & Tactics for the MBE has been carefully revised by Steve Emanuel and is full of up-to-date advice on how to analyze Multistate Bar Exam (MBE) questions in all MBE subject areas (Civil Procedure, Constitutional Law, Contracts, Criminal Law and Procedure, Evidence, Real Property with Future Interests, and Torts). Steve Emanuel—author of the Emanuel Law Outlines and CrunchTime books in the MBE subject areas—has passed the bar exam in several states (including New York and California) and worked with law students to prepare them for taking the MBE.	\$165.0	\$165.0 

If you click on Wishlist in the upper right corner, you will be redirected to a page where you can see the items in your wishlist. You can click the little x next to each textbook listing next to the Total column to remove the item from your wishlist.

Home

Sell Wishlist Cart

Shopping Cart

Item	Price	Total
 Campbell Biology ISBN: 9780134093413 Condition: MINT Description: The Eleventh Edition of the best-selling Campbell BIOLOGY sets students on the path to success in biology through its clear and engaging narrative, superior skills instruction, innovative use of art and photos, and fully integrated media resources to enhance teaching and learning. To engage learners in developing a deeper understanding of biology, the Eleventh Edition challenges them to apply their knowledge and skills to a variety of new hands-on activities and exercises in the text and online. Content updates throughout the text reflect rapidly evolving research, and new learning tools include Problem-Solving Exercises, Visualizing Figures, Visual Skills Questions, and more.	\$30.0	\$30.0

Total: \$30.0

[CHECKOUT](#)

On the upper right corner, clicking the Cart icon will sends you to your shopping cart. Similar to wishlist page, you can clicks the little x to the right next to the Total column to delete the item from your shopping cart. You can also clicks Checkout to begins placing the order.

Home Sell Wishlist Cart

Shipping Address

First Name:

Last Name:

Address 1:

Address 2:

Hawaii

City:

Zip Code:

Shipping Method

Standard - 5.00\$ - Ships in 3-5 Business Day

Express - 12.00\$ - Ships in 1-2 Business Day

Use as Billing Address

ORDER SUMMARY

Cracking the Coding Interview
ISBN: 9780984782857
Author: Gayle Laakmann McDowell
Publisher: CareerCup
Condition: LIKE NEW
Price: 98.0

SHIPPING: \$12.00
ORDER TOTAL: \$110

First step of the checkout form, input your shipping address and choose shipping methods

Home Sell Wishlist Cart

City:

Zip Code:

NEW CREDIT CARD

Name on Card:

example card

Card Number:

May 2025

765|

Second step of the checkout form, input your billing address and credit cards information

REVIEW YOUR ORDER

PLACE ORDER

SHIPPING

EDIT

PAYMENT INFORMATION

EDIT

CREDIT CARD INFORMATION

EDIT

SHIPPING ADDRESS

test account
850 Meadowbrook North
Honolulu, HI 96808

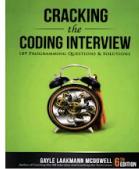
BILLING ADDRESS

test account
850 Meadowbrook North
Honolulu, HI 96808

example card
***** 9125
Exp. 05/2025
Price: \$110

SHIPPING METHOD

EXPRESS

Item	Price	Total
 Cracking the Coding Interview ISBN: 9780984782857 Condition: LIKE NEW Description: I am not a recruiter. I am a software engineer. And as such, I know what it's like to be asked to whip up brilliant algorithms on the spot and then write flawless code on a whiteboard. I've been through this as a candidate and as an interviewer. Cracking the Coding Interview, 6th Edition is here to help you through this process, teaching you what you need to know and enabling you to perform at your very best. I've coached and interviewed hundreds of software engineers. The result is this book. Learn how to uncover the hints and hidden details in a question, discover how to break down a problem into manageable chunks,	\$98.0	\$98.0

Last step of the checkout form, you can review your order summary and click the red Place Order button to submit your order.

MY ACCOUNT SETTINGS

EDIT YOUR PROFILE DETAILS

Profile Picture: No file chosen

Username: test

Email address: test@yahoo.com

First name: test

Last name: account

Password:

Password Confirmation:

SAVE

Clicking on the avatar drop down button on the upper right corner and the Settings option will send you to a page where you can modify your account information

test

test account

test@yahoo.com

Date Joined: Dec. 6, 2019, 6:25 a.m.

in

BASIC ECONOMICS
A Common Sense Guide to the Economy
Thomas Sowell

Basic Economics
Condition: MINT
Price: \$75.0

EDIT LISTING

Clicking on the avatar drop down button on the upper right corner and the Profile option will send you to a page where you can see your basic account information and your lists of active listings

Home



example card

***** 9125

Expire On: 05/2025

Billing Address: 850 Meadowbrook North Honolulu, HI 96808

Clicking on the avatar drop down button on the upper right corner and the My Credit Cards option will send you to a page where you see the credit cards saved to your account

Home Sell Wishlist Cart

Order History

Order ID: 10

CRACKING the CODING INTERVIEW
Author: Gayle Laakmann McDowell
Publisher: CareerCup
Condition: LIKE NEW
Listed for sale by: jesseking
Price: \$98.0

Shipping Method: EXPRESS
Shipping Cost: \$12.00
Time of Purchase: Dec. 8, 2019, 8:02 a.m.
Total price: 110

Clicking on the avatar drop down button on the upper right corner and the My Orders option will send you to a page where you can view your Order History.

- **SECURITY**

All of our account passwords are first hashed and then stored in our database.

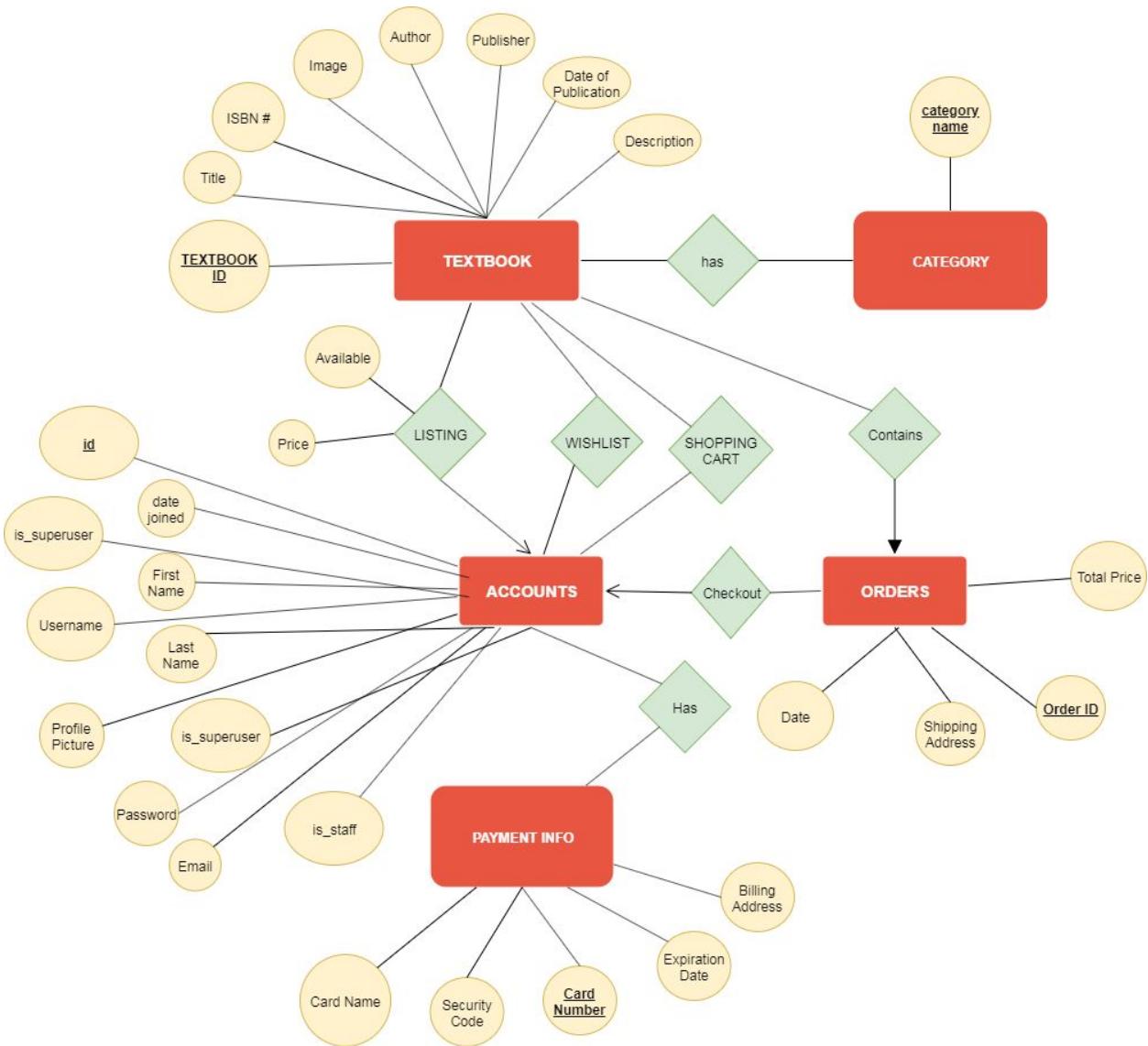
Our MySQL queries are all sanitized to prevent MySQL injection attacks

- **ACCESS CONTROL**

- We store the logged in user's information similarly to a session so that features such as editing textbook listing can only be done on listing of their own and our query make sure of that by using the session information in queries.
- Every other URL besides the login/register page requires the user to be logged in or it will return a Page Not Found error.

Project Design

a. ERD Model



Entity Sets:

- 1) Textbook - Entity set representing a textbook. It contains attributes ISBN-13 #, Title, Image, Author, Publisher, Year of Publication, Description. It has an auto-increment primary key textbook_ID
- 2) Category - An entity set representing a category that a textbook belongs to. It has an auto-increment primary key category_id and attribute category name.
- 3) Orders - an entity set representing a transaction/order made (order histories). It contains auto-increment key Order ID, Date, and Shipping Address attribute.
- 4) Accounts - An entity set representing the account of the users of our application. It contains attributes is_superuser, is_staff, is_active, Password, Email, Profile Picture, Name, Username and primary key id
- 5) Payment Info is an entity set representing a credit/debit card users used to pay for transactions. It contains attributes Card Owner(Name), Security Code, Card Type, Expiration Date, Billing Address and primary key is the 16-digit Card Number.

Relationships:

- b. Listing is a many-to-one relationship between Accounts and Textbooks. Each user would be able to add many textbook listings for sale and each textbook can only points to 1 account. It has an attribute Available which serves as a boolean field. If Available = 1, then the listing is still available for sale. If Available = 0, the listing has been sold already and is no longer available for sale.
- c. Wishlist is a many-to-many relationship between Accounts and Textbooks. Each user would be able to add many textbooks to their wishlist and each textbooks could be in many users' wishlist.
- d. Checkout is a many-to-one relationship between Accounts and Order. The user would be able to checkout many orders while each order must be bind to 1 account.
- e. Between Orders and Textbooks, there is a many-to-one relationship Contains where an order can contain multiple textbooks while each textbook can only belong to 1 order.
- f. Between Textbook and Category, the many-to-many relationship Has represents that 1 category can have many textbooks and many textbooks can belong to many different categories.
- g. Between Accounts and PaymentInfo, the many-to-many relationship Has represents that 1 account can have multiple payment methods and 1 credit card can be used for transactions in multiple accounts.

Constraints:

Django doesn't allow composite primary keys (multi-columns primary key) so instead it Adds a default auto-increment field id as primary key for the relationships table. I Adds a constraint to these relationship tables for example Listing so that Account_id And Textbook_id must be unique together in every row so they behaves the same as a Primary key. Then in my queries I never use the auto-increment field id and instead just Use textbook_id and account_id without worrying about duplicates etc.

h. Normalization Process

FDs:

Account

Id -> username, profile_picture, email, password, first name, last name, is_superuser, is_staff, is_active

Textbook

Textbook_id -> Title, ISBN #, Image, Author, Publisher, Date of Publication, Description

Category

No FDs

Orders

Order_ID -> Total Price, Shipping Address, Date Purchased

PaymentInfo

Card Number -> Card Name, Security Code, Billing Address, Expiration Date

Listing relationship

Account_id, Textbook_id -> Price, Available

Based on the FDs above, the tables all satisfy BCNF therefore it is not necessary To perform the normalization process.

i. Table Schema and Database Models

We are using Django as part of our 3 tier architecture and unfortunately Django comes with an ORM that implements a database table as an object in Python. This means that I have to specify my schemas as classes in Python and Django will automatically convert the classes into MySQL tables and each object instance I created will be converted to a new table instance in MySQL database.

- 1.) The problem is that Django does not support the use of composite primary key. It requires that only 1 column can be primary key maximum. Therefore, in the tables that represent relationships Django automatically adds a field id as the auto-increment primary key. For example, relationship Wishlist is a relationship that connects entity set Account and Textbook. The table for Wishlist will become Wishlist(id, Account_key, Textbook_key). **However, Django lets me add a constraint specifying that no instances of Wishlist can have the same Account_key and Textbook_key so it behaves the same as a primary key. The only setback is that those relationships will have a mandatory auto-increment primary key id, the behavior will be the same.**
- 2.) Django automates the conversion of my schemas into Database tables. However, the naming convention can be a little disambiguous. In Account table, id is the primary key and in other relationships that contain the primary key of Account it is stored as Account_id.
- 3.) Account table has extra fields that Django automatically adds to it because it recognizes Account table as the user model. These fields aren't listed in E/R or database queries because they aren't used at all.
- 4.) When converting schemas to database tables with Django, it also creates other tables such as Admin and admin_auth_permission that are not used at all in our application so we didn't list them.

IMPORTANT

The database queries below are not taken all at the same time. A textbook may exist in some Account's Wishlist or Shopping Cart in the below screenshots even though in the Listing table the availability equals 0 (sold already not available for sale) because I did not screenshot the Wishlist and Shopping Cart table before I checked out some orders and then I took a screenshot of Listing table.

The screenshot shows a MySQL Workbench interface with a query editor and a results grid.

```

Query 1
use store;
SELECT * FROM Account;

```

The results grid displays the following data:

	id	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined	Profile_Picture
▶	1	pbkdf2_sha256\$2019-12-08 08:28:02.696521	1	root	John Doe	xtranx2@yahoo.com	1	1	2019-11-15 01:42:11.000000	people_qNkcXcu.p		
2	k3LZTvtW*C	NULL	0	csmartinez	Destiny Reyes	martincheyenne@hotmail.com	0	1	2019-11-15 01:42:46.228551			
3	#9jAWBKT(b	NULL	0	hsharp	David Wright	samanthahogan@harrington.com	0	1	2019-11-15 01:42:46.233496			
4	OaSSPfzwI1	NULL	0	travis86	Michael Rodgers	barnettmonica@morris-carter.com	0	1	2019-11-15 01:42:46.237572			
5	lQrC^rdt7	NULL	0	reedmonica	Frank Rodriguez	sarah17@ayers-johnson.com	0	1	2019-11-15 01:42:46.242007			
6	4^R9Vxy9nK	NULL	0	jesseking	Micheal Winters	renee47@page.com	0	1	2019-11-15 01:42:46.246044			
7	SpLS@VIZ%0	NULL	0	millsjoseph	Christina Hunt	reedtclark@davis-taylor.com	0	1	2019-11-15 01:42:46.250915			
8	#4O^ZAVvJn	NULL	0	martinkyle	Lauren Cabrera	jeremiah37@gmail.com	0	1	2019-11-15 01:42:46.255478			
9	juc9Rnb8@&w	NULL	0	brady53	Deborah Robinson	ujacobs@mccullough.com	0	1	2019-11-15 01:42:46.260397			
10	h7W\$Xsbg^E	NULL	0	james21	James McDonald	patrick42@jordan.com	0	1	2019-11-15 01:42:46.265089			
11	TVhX3E^o%Y	NULL	0	awilliams	Heather Johnson	lisa79@yahoo.com	0	1	2019-11-15 01:42:46.268806			
12	32m2Fyxn\$y	NULL	0	rose97	Loretta Peterson	hscott@campos.org	0	1	2019-11-15 01:42:46.273125			
13	%JJo2f2kD)	NULL	0	christina77	James Cummings	sawyerkenneth@gmail.com	0	1	2019-11-15 01:42:46.277175			
14	9^Enkxgl+d	NULL	0	jameswilliams	Jennifer Lee	michaelchavez@gonzalez-davis...	0	1	2019-11-15 01:42:46.281633			
15	^MxcoWShq7	NULL	0	jenniferwilli...	Alison Walker	rwillis@white-palmer.biz	0	1	2019-11-15 01:42:46.286193			
16	_FKKs4U#6	NULL	0	mbryant	Christy Bennett	cynthiajohnson@hotmail.com	0	1	2019-11-15 01:42:46.290385			
17	P+b+2Re9A4	NULL	0	barberernest	Latoya Robinson	foxmark@roman.com	0	1	2019-11-15 01:42:46.295517			
18	(^1Qh4x4's	NULL	0	carladudley	Monica Bright	cruzalexander@roberts.com	0	1	2019-11-15 01:42:46.300285			
19	&58saafalz@	NULL	0	amber68	Shane Bird	reedbradley@yahoo.com	0	1	2019-11-15 01:42:46.305440			
20	J68Hbymtt(NULL	0	lindseygomez	Jasmin Frank	rodriguezlisla@hotmail.com	0	1	2019-11-15 01:42:46.310096			
21	^1kolmh-ly74	NULL	0	richard71	Lynn Burnett	james35@woods.com	0	1	2019-11-15 01:42:46.314999			
22	(P^1FCI@L	NULL	0	nilara	Angela Becker	jrivera@mcdowell-foley.com	0	1	2019-11-15 01:42:46.319426			
23	+j20WBEaoF	NULL	0	michael04	Marcia Jones	weisscheryl@gmail.com	0	1	2019-11-15 01:42:46.323660			

Figure 1

Figure 1 Accounts(**id**, Username, last_login, is_superuser, is_staff, is_active, date_joined
Profile_Picture, Password, Email, Name)

Query 1

```

1 • use store;
2 • SELECT * FROM Textbook;
3
4
5
6
7

```

Result Grid

Textbook_ID	ISBN	Title	Image	Author	Publisher	Date_Published	Cond	Description
1	9781461471370	An Introduction to Statistical Learning:... An-Introduction-To-S...	Gareth James, Dani...	Springer	2017-09-01	MINT	An Introduction to Statistical Learning provide	
2	9780525568292	Cracking the AP Physics 1 Exam 2020... AP-Physics-Exam_...	The Princeton Review	Princeton Review	2019-08-06	DECENT	PREMIUM PRACTICE FOR A PERFECT 5! A	
3	9780323393225	Atlas of Human Anatomy	Atlas-Of-Human-An...	Frank H. Netter MD	Elsevier Health S...	2018-03-13	DECENT	The only anatomy atlas illustrated by physician
7	9781454894018	Basic Legal Research: Tools and Strat...	basic-legal-research...	Amy E. Sloan	Wolters Kluwer	2018-02-28	MINT	Basic Legal Research: Tools and Strategies, 3
8	9781543805727	Strategies & Tactics for the MBE (Ema...	518HIIr9HL..._SX3...	Steven L. Emmanuel	Wolters Kluwer	2016-05-22	DECENT	The Seventh Edition of Strategies & Tactics for
9	9781469893419	Bates' Guide to Physical Examination	Bates-Guide-To-Phy...	Lynn S. Bickley	LWW	2016-09-09	LIKE NEW	Bates' Guide to Physical Examination and His
10	9780984782857	Cracking the Coding Interview	Cracking-The-Codin...	Gayle Laakmann M...	CareerCup	2015-07-01	LIKE NEW	I am not a recruiter. I am a software engineer.
11	9781479274833	Elements of Programming Interviews i...	Elements-Of-Program...	Adnan Aziz, Tsung...	CreateSpace Ind...	2016-09-15	MINT	The core of EPI is a collection of over 250 prc
12	9780133915389	Engineering Mechanics: Dynamics 14t...	Engineering_Mecha...	Russell C. Hibbeler	Pearson	2015-04-10	DECENT	A Proven Approach to Conceptual Understand
13	9780714833569	Modern Architecture Since 1900 / Editi...	modern-architecture...	William J.R. Curtis	Phaidon Press	1996-07-27	FAIR	Since its first publication in 1982, Modern Arci
14	9780071592536	Security Analysis: Sixth Edition	Security-Analysis_3...	Benjamin Graham,...	McGraw-Hill Edu...	2008-09-25	MINT	"A road map for investing that I have now bee
15	9781563684449	The American Sign Language Handsh...	51zvHUIBpL..._SX3...	Richard A. Tennant,...	Gallaudet Univers...	2010-07-31	DECENT	Now, the bestselling resource The American S
20	9780367188498	Theories and Methods of Writing Cent...	Theories_and_Meth...	Jo Mackiewicz, Reb...	Routledge	2019-11-13	LIKE NEW	This collection helps students and researcher
22	9780395872741	World History: Patterns of Interaction	World_History_Patte...	McDougal Littell	McDougal Littell	1998-02-02	DECENT	World History Textbook has cover scratches a
25	9780134093413	Campbell Biology	Campbell-Biology.jpg	Lisa A. Urry, Michael...	Pearson	2016-10-29	MINT	The Eleventh Edition of the best-selling Camp
26	9780465060733	Basic Economics	Basic-Economics_Q...	Thomas Sowell	Basic Books	2014-12-02	MINT	example description
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Textbook 3

Figure 2

Figure 2 Textbook(Textbook_ID, ISBN, Title, Image, Author, Publisher, Date_Published, Condition, Description)

Query 1

use store;
SELECT * FROM Orders;

Order_ID	Date	Shipping_Address	Total_Price	Shipping_Method
2	2019-11-29 05:59:28.188283	2275 Ellena Drive Apt 4 Santa Clara, CA 95050	361	EXPRESS
3	2019-11-30 02:00:13.960078	2275 Ellena Drive Apt 4 Santa Clara, CA 95050	92	STANDARD
8	2019-12-03 23:59:52.589983	993 River St. Santa Clara, CA 95050	80	STANDARD
9	2019-12-05 04:40:12.367638	993 River St. Santa Clara, CA 95050	83	STANDARD
10	2019-12-08 08:02:09.302901	850 Meadowbrook North Honolulu, HI 96808	110	EXPRESS
11	2019-12-10 04:17:03.000000	797 Helen Street Bolingbrook, IL 60440	30	STANDARD
12	2019-12-10 04:22:42.353704	8814 Cardinal St. Westminster, HI 21157	55	STANDARD
13	2019-12-10 04:24:27.670936	8129 Manor St. San Jose, CA 95127	312	EXPRESS
14	2019-12-10 04:25:48.958574	2 Forest Street Nashua, MN 03060	90	EXPRESS
15	2019-12-10 04:26:48.206659	546 Summer Drive Beloit, ND 53511	177	EXPRESS
16	2019-12-10 04:27:28.956637	8170 Old Argyle Road Clermont, IL 34711	65	STANDARD
17	2019-12-10 04:28:57.356775	921 North Del Monte Dr. Milford, RI 12590	55	STANDARD
18	2019-12-10 04:31:09.889706	946 Rock Maple Ave. Salem, DE 01970	145	STANDARD
19	2019-12-10 04:31:44.211601	7833 Nicolls Street Valrico, MA 33594	162	EXPRESS
20	2019-12-10 04:34:26.180546	454 Garden Street Utica, ME 59128	55	STANDARD
NULL	NULL	NULL	NULL	NULL

Orders 5

Figure 3

Figure 3 Order(Order_ID, Date, Shipping_Address, Total_Price, Shipping_Method)

Figure 4

Figure 4 PaymentInfo(Card_Number, Card_Name, Security_Code, Expiration_Date, Billing_Address)

Figure 5

Figure 5 Account Has PaymentInfo(id , Account id, Card Number)

This primary key id is an auto-increment field automatically added by Django. It is never used in my queries since I added a constraint that let Account_id and Card_Number must be unique together in every row so it behaves like a primary key.

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons for database management. Below the toolbar, a query editor window titled "Query 1" contains the following SQL code:

```

1 • use store;
2 • SELECT * FROM Category_Has_Textbook;
3
4
5
6
7

```

Below the query editor is a status bar showing "100%" and "36:2". The main area is titled "Result Grid" and displays a table with three columns: "id", "Category_Name", and "Textbook_ID". The data in the table is as follows:

	id	Category_Name	Textbook_ID
▶	34	Architecture	13
	107	Biology	25
	40	Communication	15
	98	Communication	20
	2	Computer Science	1
	29	Computer Science	10
	30	Computer Science	11
	37	Engineering	12
	36	Finance	14
	115	Finance	26
	90	History	22
	27	Law	7
	23	Law	8
	28	Literature	7
	99	Literature	20
	1	Mathematics	1
	3	Mathematics	2
	39	Mathematics	12
	5	Medicine	3
	94	Medicine	9
	4	Physics	2
	38	Physics	12
	NULL	NULL	NULL

The table has a header row with column names. The data rows show various category names like Architecture, Biology, Communication, etc., each associated with a specific textbook ID. The last row of the table is empty (NULL values). Below the table, a footer bar shows the table name "Category_Has_Textbook 7".

Figure 6

Figure 6 Category_Has_Textbook(**id**, Category_Name, Textbook_id)

This primary key id is an auto-increment field automatically added by Django. It is never used in my queries since I added a constraint that Category_Name and Textbook_id must be unique together in every row so it behaves like a primary key.

The screenshot shows the MySQL Workbench interface with a query editor and a results grid.

Query Editor:

```

1 • use store;
2 • SELECT * FROM Listing;
3
4
5
6
7

```

Result Grid:

	id	Price	Account_id	Textbook_ID	Available
▶	1	300	7	1	0
	2	50	13	2	0
	3	75	3	3	0
	7	78	1	7	0
	8	165	1	8	0
	9	60	1	9	0
	10	98	6	10	0
	11	87	8	11	0
	12	250	25	12	0
	13	50	30	13	0
	14	99	20	14	0
	16	140	7	15	0
	19	150	1	20	0
	21	78	33	22	0
	24	30	1	25	0
	25	50	1	26	0
	NULL	NULL	NULL	NULL	NULL

Listing 4

Figure 7

Figure 7 Listing(**id**, Price, Account_id, Textbook_id)

This primary key **id** is an auto-increment field automatically added by Django. It is never used in my queries since I added a constraint that **Account_id** and **Textbook_id** must be unique together in every row so it behaves like a primary key.

The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons for database management. Below the toolbar, a query editor window titled "Query 1" contains the following SQL code:

```

1 • use store;
2 • SELECT * FROM Wishlist;
3
4
5
6
7

```

Below the query editor is a results grid titled "Result Grid". The grid has three columns: "id", "Account_id", and "Textbook_ID". The data in the grid is as follows:

id	Account_id	Textbook_ID
43	1	1
42	1	2
41	1	13
40	1	15
31	33	1
32	33	9
30	33	20
44	33	25
33	51	13
34	51	20
35	51	25
► 37	52	1
36	52	7
38	52	8
39	52	15
NULL	NULL	NULL

At the bottom of the results grid, there is a footer bar with the text "Wishlist 12".

Figure 8

Figure 8 Wishlist(**id**, Account_id, Textbook_id)

This primary key id is an auto-increment field automatically added by Django. It is never used in my queries since I added a constraint that Account_id and Textbook_id must be unique together in every row so it behaves like a primary key.

⚡ Query 1

📁 📂 ⚡ ⚡ 🔎 🔍 ✎ 🔍 🔍 Limit to 1000 rows ↴ ⭐ 🐾

```
1 • use store;
2 • SELECT * FROM Shopping_Cart;
```

3
4
5
6
7

100% 28:2

Result Grid Filter Rows: Search Edit: Export/Import

id	Account_id	Textbook_ID
70	1	1
58	33	1
64	52	1
69	1	2
63	52	7
65	52	8
59	33	9
68	1	13
60	51	13
67	1	15
66	52	15
57	33	20
61	51	20
71	33	25
62	51	25
NULL	NULL	NULL

Shopping_Cart 13

Figure 9

Figure 9 Shopping_Cart(**id**, Account_id, Textbook_id)

This primary key id is an auto-increment field automatically added by Django. It is never used in my queries since I added a constraint that Account_id and Textbook_id must be unique together in every row so it behaves like a primary key.

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons and a dropdown menu set to 'Query 1'. Below the toolbar is a query editor window containing the following SQL code:

```

1 •  use store;
2 •  SELECT * FROM Checkout;
3
4
5
6
7

```

Below the query editor is a result grid titled 'Result Grid' with three columns: 'id', 'Account_id', and 'Order_ID'. The data in the grid is as follows:

id	Account_id	Order_ID
1	1	2
2	1	3
8	1	9
11	1	12
10	8	11
20	17	8
12	33	13
13	33	14
▶ 16	51	17
17	51	18
18	51	19
9	52	10
14	52	15
15	52	16
19	52	20
NULL	NULL	NULL

At the bottom of the result grid, it says 'Checkout 9'.

Figure 10

Figure 10 Checkout(**id**, Account_id, Order_id)

This primary key id is an auto-increment field automatically added by Django. It is never used in my queries since I added a constraint that Account_id and Order_id must be unique together in every row so it behaves like a primary key.

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons for file operations, search, and navigation. Below the toolbar, a query editor window titled "Query 1" contains the following SQL code:

```

1 • use store;
2 • SELECT * FROM Order_Contain_Textbook;
3
4
5
6
7

```

Below the query editor is a status bar showing "100%" and "37:2". The main area is a "Result Grid" showing the output of the query. The grid has three columns: "id", "Order_ID", and "Textbook_ID". The data consists of 20 rows, each containing a unique combination of Order_ID and Textbook_ID. The last row is a blank row with all columns set to "NULL".

	id	Order_ID	Textbook_ID
▶	13	13	1
	12	12	2
	8	8	3
	14	14	7
	15	15	8
	16	16	9
	10	10	10
	3	3	11
	1	2	12
	17	17	13
	2	2	14
	18	18	15
	19	19	20
	9	9	22
	11	11	25
	20	20	26
	NULL	NULL	NULL

At the bottom of the result grid, there is a footer bar with the text "Order_Contain_Textbook 6".

Figure 11

Figure 11 Order_CContain_Textbook([id](#), Order_id, Textbook_id)

This primary key id is an auto-increment field automatically added by Django. It is never used in my queries since I added a constraint that Order_id and Textbook_id must be unique together in every row so it behaves like a primary key.

⚡ Query 1

Limit to 1000 rows

1 • use store;
2 • SELECT * FROM Category;

3
4
5
6
7

100% 11:1

Result Grid Filter Rows: Search Edit: Export/Import:

Category_Name
Architecture
Arts
Biology
Chemistry
Communication
Computer Science
Engineering
Finance
History
Humanities
Law
Literature
Mathematics
Medicine
Physics
NUL

Category 8

Figure 12

Figure 12 Category(**Category Name**)

Database Implementation

IMPORTANT:

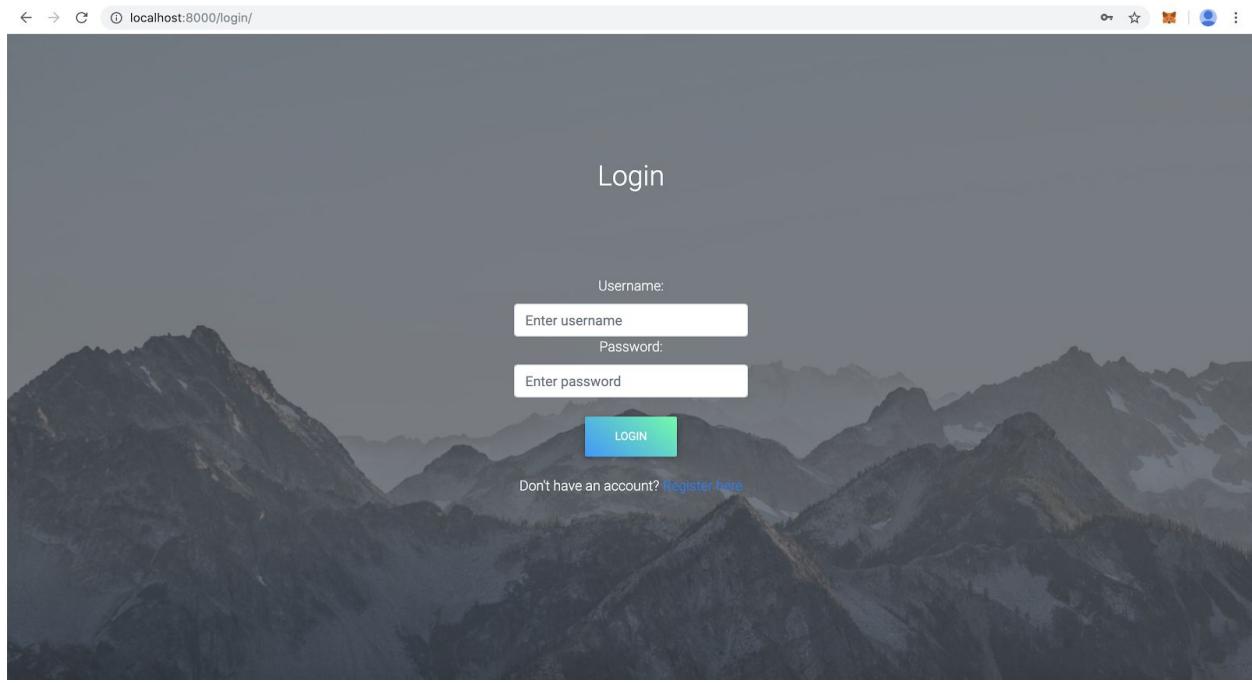
If I am doing a SELECT query statement, in Django I do not need to establish a connection with connection.cursor() and use cursor.execute() to do this query. Django ORM maps a table Account for example to a class Account in Python. Each instance of an Account class is equivalent to a row in table Account. So if I perform a select query on table Account I can invoke

```
accs = Account.objects.raw("SELECT * FROM Account")
```

This will return to me a list of Account objects corresponding to the rows returned. I can access the column value username of the first row returned by using accs[0].username for example.

Request.user is similar to a session that stores information about the logged in user. I can access information about the account such as username, id, email, password etc of the logged in user through it.

LOGIN PAGE



Upon starting our application, you will access it by going to localhost:8000 and will be directed to our login page which prompts you to enter your account information.

\

BACKEND LOGIC

```
47 @require_POST
48 @csrf_exempt
49 def login_request(request):
50     """
51         Process the login request by querying the database for an existing Account given user input
52         and returns a JsonResponse to the ajax call
53     """
54
55
56     context = {}
57     if request.is_ajax() == True:
58         username = request.POST.get("username")
59         password = request.POST.get("password")
60         if len(Account.objects.raw("SELECT * FROM Account WHERE username = %s", [username])) == 0:
61             context['username'] = 'Not found'
62             context['password'] = 'Incorrect'
63         else:
64             context['username'] = 'Found'
65             user = authenticate(username=username, password=password)
66             if user is None:
67                 context['password'] = 'Incorrect'
68             else:
69                 auth_login(request, user)
70                 context['message'] = 'Success';
71     return JsonResponse(context)
72
```

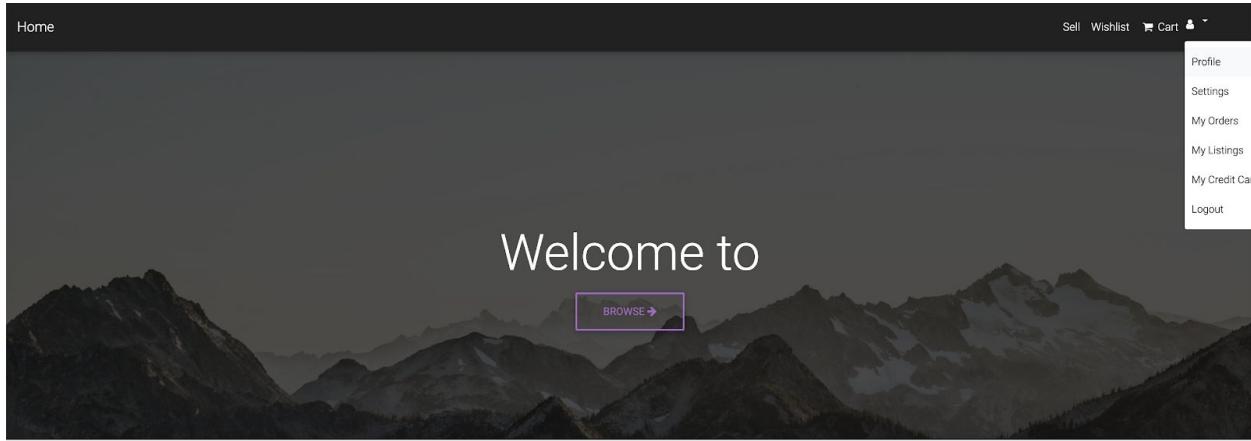
```
if len(Account.objects.raw("SELECT * FROM Account WHERE username = %s", [username])) == 0:
```

I query the Account table where username = username provided by the user and if the result returns 0 row then I return a message to the view that there exists no account with that login information.

Else if a result is found, I invoke authenticate with the username and password and if it returns me a None object that means the password is incorrect and I return a message saying the password is incorrect.

Otherwise, I logs the user in with auth_login

LOGOUT



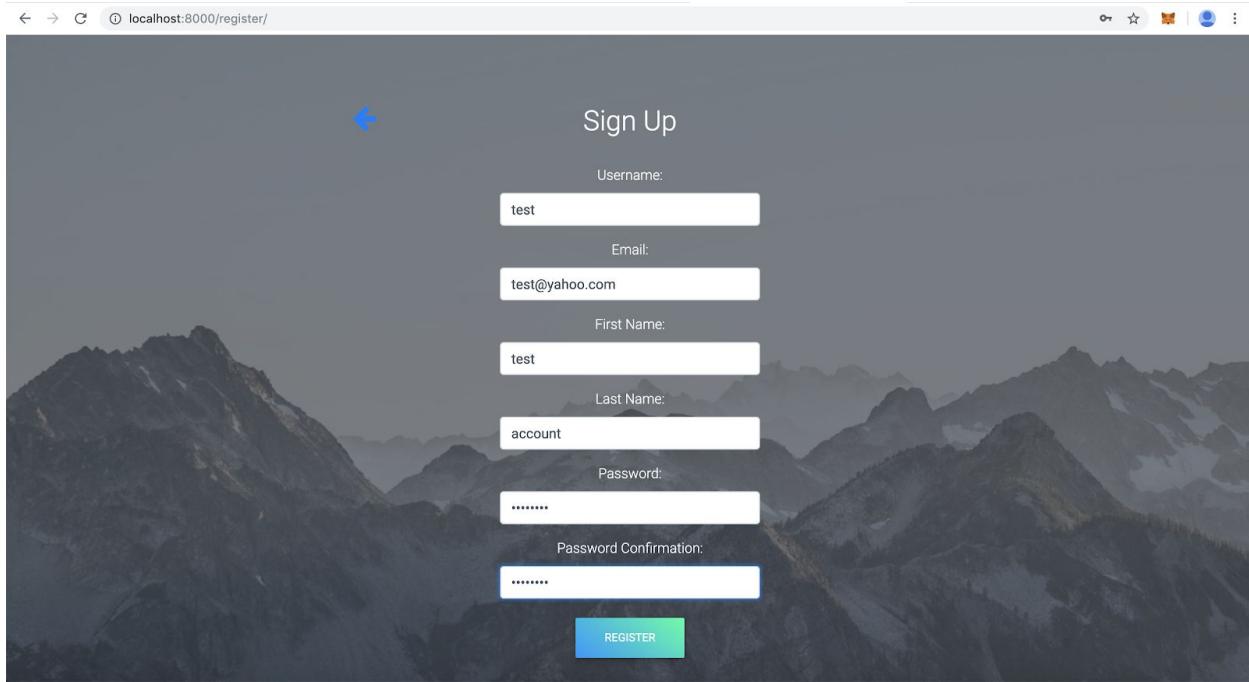
Once you are logged in, you can select the avatar dropdown button on the upper right corner and select Logout button to log out of the application. You will then be redirected to the login page.

```
72  
73 @login_required  
74 def logout(request):  
75  
76     auth_logout(request)  
77     return HttpResponseRedirect(reverse('home'))  
78
```

Just invoke auth_logout on the request and the user will be logged out.

REGISTER AN ACCOUNT

For registering, I will insert a new row into Account table.



The screenshot shows a web browser window with the URL `localhost:8000/register/` in the address bar. The page title is "Sign Up". The form consists of several input fields: "Username:" with value "test", "Email:" with value "test@yahoo.com", "First Name:" with value "test", "Last Name:" with value "account", "Password:" with value "*****", and "Password Confirmation:" with value "*****". Below the form is a blue "REGISTER" button. The background of the page is a photograph of a mountain range.

If you don't have an account, clicks the Register Button beneath the Login button to be directed to a form for you to sign up.

After clicking the register button, you will automatically be logged in to the application, redirected to the landing page, and your account information will be saved in the database in the Account table. I will go ahead and fill in the information and clicks Register.

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons for database management. Below the toolbar, a query editor window displays two lines of SQL code:

```
1 • use store;  
2 • SELECT id, username, first_name, last_name, email, password, Profile_Picture FROM Account WHERE username = 'test';
```

Below the query editor is a status bar showing "100%" and "115:2". Underneath the status bar is a "Result Grid" section. It includes a "Filter Rows" search bar and buttons for "Edit" and "Export/Import". The result grid itself has columns labeled "id", "username", "first_name", "last_name", "email", "password", and "Profile_Picture". A single row is visible, corresponding to the user "test" with ID 52. The "password" column contains a hashed value, and the "Profile_Picture" column is marked as "NULL".

As you can see, a new row is inserted into the Account table with the information I just filled out. The Profile Picture column is null because we don't force you to upload a profile picture upon registration. The password is also hashed for security purposes.

BACKEND LOGIC

```
78 def register(request):
79     # If user is logged in already, redirects to homepage
80     if(request.user.is_authenticated):
81         return HttpResponseRedirect(reverse('homepage'))
82
83     # This form is used for creating registration form
84     form = forms.AccountCreationForm();
85
86
87     if(request.method == 'POST'):
88
89         form = forms.AccountCreationForm(request.POST)
90         # Hashed the user provided password
91         password = make_password(request.POST.get('password1'))
92         # Query the database to see if there is any account with this username
93         # in the database
94         get_user_query = "SELECT * FROM Account WHERE username = %s"
95         get_user_arguments = [request.POST.get('username')]
96         user = Account.objects.raw(get_user_query, get_user_arguments)[:]
97
98         # if the query results return 0 rows then go ahead and make an account
99         if len(user) == 0:
100             date_joined = datetime.now()
101             password = make_password(request.POST.get('password1'))
102             add_user_query = """INSERT INTO Account (username, password, email, first_name, last_name, date_joined,
103                                         is_staff, is_superuser, is_active) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s);"""
104             add_user_arguments = [request.POST.get('username'), password, request.POST.get('email'), request.POST.get('first_name'),
105                                   request.POST.get('last_name'), date_joined, '1', '1', '1']
106
107             with connection.cursor() as cursor:
108                 cursor.execute(add_user_query, add_user_arguments)
109             connection.close()
110
111             get_user_query = "SELECT * FROM Account WHERE username = %s AND password = %s"
112             get_user_arguments.append(password)
113             user = Account.objects.raw(get_user_query, get_user_arguments)[0]
114             |
115             auth_login(request, user);
116             request.user.is_superuser = False
117             request.user.is_staff = False
118
119             return HttpResponseRedirect(reverse('homepage'))
120         else:
121             username_exists = True;
122             return render(request, 'Register.html', context={'form':form, 'username_exists': username_exists});
123
124     return render(request, 'Register.html', context={'form':form});
```

First, this function checks if the user is already logged in. If so, it redirects the user to the homepage instead.

Then, it hashed the password provided by the user and then query

```
get_user_query = "SELECT * FROM Account WHERE username = %s"
```

This query is done to check if there already exists a user with this username already in the database. If not, go ahead and create a new account. If there already exists an account, return a message to the view and the view will update to tell the username to choose a different username.

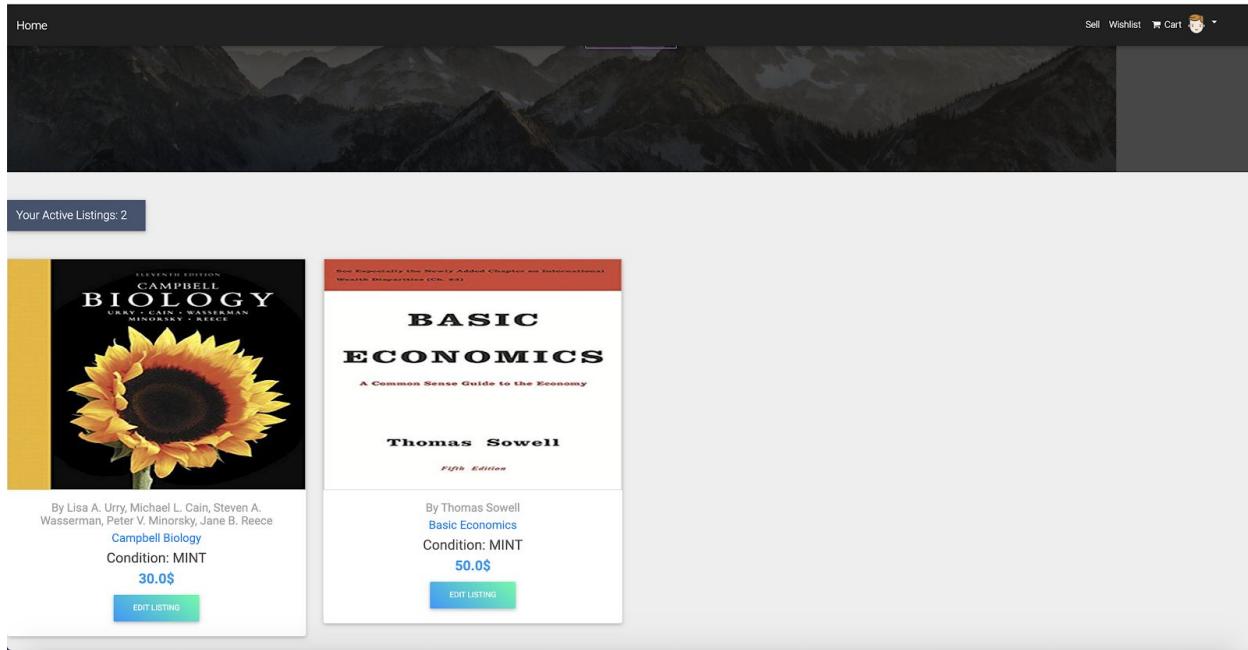
```
add_user_query =  
"""  
INSERT INTO Account (username, password, email, first_name, last_name, date_joined,  
is_staff, is_superuser, is_active)  
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s);  
"""
```

This is the query that adds a new row to the Account table with the user provided username, hashed password, email, first name, last name, date joined, is_staff, is_superuser, is_active. Is_staff, is_superuser, is_active are fields enforced on user models by Django, when creating a new account I put in values of 1 for True which means that the account is a superuser, is a staff, and is active. I put in values of 1 because for some reason if I put in 0 then it will store correctly as 0 in MySQL database but later on when I query with Django, it will shows those fields as NULL instead of 0 so for now I set them as values of 1 and set them to False later in the function.

```
get_user_query = "SELECT * FROM Account WHERE username = %s AND password = %s"  
user = Account.objects.raw(get_user_query, get_user_arguments)[0]  
  
auth_login(request, user);
```

Finally one last query is done to get the newly created row just inserted into Account table. With that row mapped to an Account object in Django, I can invoke auth_login to log the user in.

Homepage



This is the homepage which you see after logging into the application. If you are a new user, you have no active listings so this page will mostly be blank except for a navbar up top and a BROWSE button in the middle of the screen for browsing textbook listings in our database. You won't see any textbook if your database is brand new. In my case, I have 2 active listings for sell.

The screenshot shows a MySQL Workbench interface with a query editor and a result grid.

```

1 • use store;
2 •     SELECT *
3     FROM Listing A, Textbook B
4     WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID AND A.Account_id = 1;
5
6
7
8
9

```

The result grid displays the following data:

id	Price	Account_id	Textbook_ID	Available	Textbook_ID	ISBN	Title	Image	Author	Publisher	Date_Published
24	30	1	25	1	25	9780134093413	Campbell Biology	Campbell-Biology.jpg	Lisa A. Urry, Michael L. Cain, Steven A. Wasserman, Peter V. Minorsky, and Robert K. Freeman	Pearson	2016-10-29
25	50	1	26	1	26	9780465060733	Basic Economics	Basic-Economics_Q...png	Thomas Sowell	Basic Books	2014-12-02

This is the query that shows the two active listings that belong to me.

BACKEND LOGIC

```

24
25 @login_required
26 def homepage(request):
27     """
28     Retrieve the logged in user's textbook listings|
29
30     """
31
32     listing_query = """
33     SELECT *
34     FROM Listing A, Textbook B
35     WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID AND A.Account_id = %s
36     """
37     listings = Listing.objects.raw(listing_query, [str(request.user.id)])
38
39     return render(request, 'Homepage.html', context={'listings':listings})

```

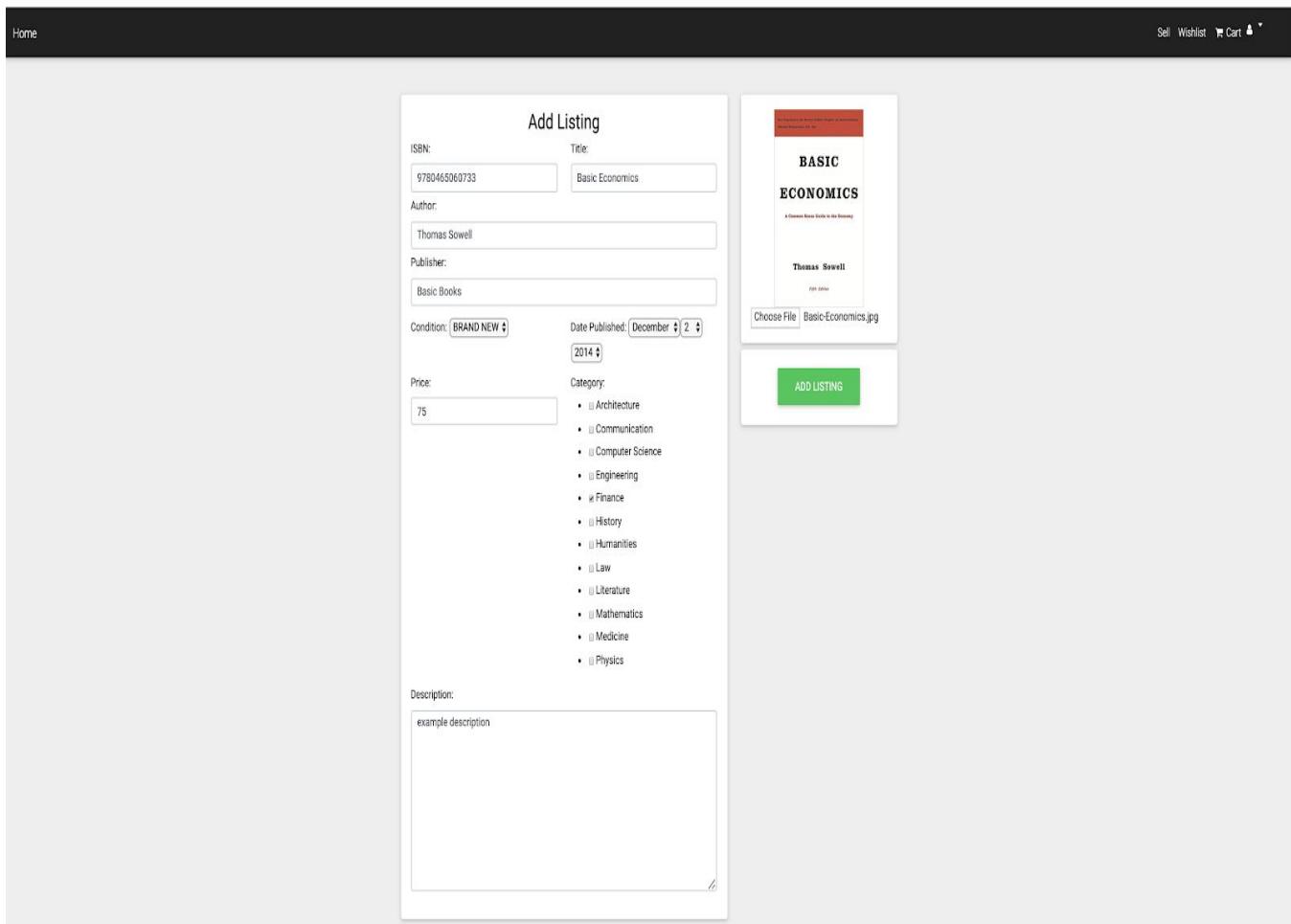
This is my select query statement for YOUR active listings to be displayed on the homepage. I join Listing and Textbook together where Listing.Available = 1 which mean that the listing is still available for sale, Listing.Textbook_ID = Textbook.Textbook_ID, and Listing.Account_id = request.user.id which means that this listing was listed by the logged in user (yourself).

Then this function returns a render of the Homepage.html file with the listings results passed in as context.

ADDING A TEXTBOOK LISTING

For this feature, I will add a new row to Textbook table, a new row to Listing table, and one or more rows to Category_Has_Textbook table.

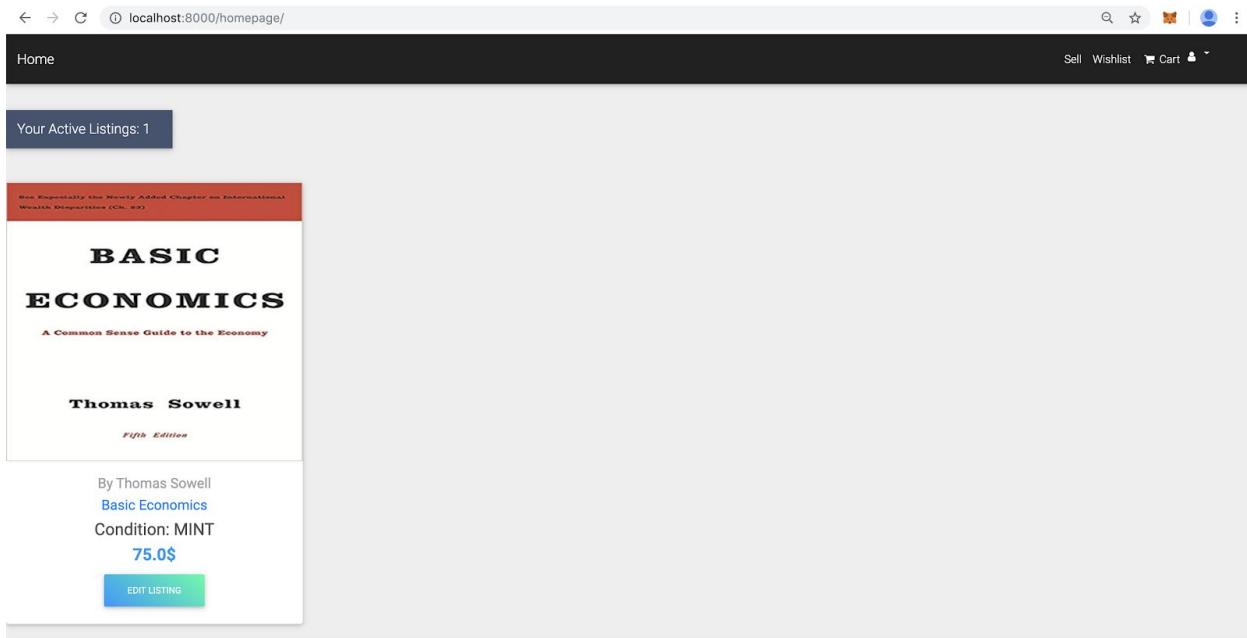
Click the SELL button on the navbar in the top right corner to begin adding a textbook listing to the database.



A form will display for you to input in the textbook's ISBN, Title, Author, Publisher, Condition, Date Published, Price, Category the textbook belongs to, Description, and an Image for the

textbook. The form is validated using javascript code in add-edit-listing.js file and basically the form will only be submitted if ISBN length is 13 and that all of the other fields must not be empty. You must also select at least 1 Category and uploads a picture for the form to submit. Nothing will be stored into our database until you hit Add Listing button, the image displayed is from using Javascript JQuery to display what you uploaded to the input field.

I will go ahead and add this textbook “Basic Economics” belonging to the category Finance.



After you press Add Listing and all of your information is valid, you will be redirected to the homepage. Scroll down and you will see your new listing presented on the bottom.

⚡ Query 1

1 • use store;
2 • SELECT * FROM Textbook;

100% 24:2

Result Grid Filter Rows: Search Edit: Export/Import:

Textbook_ID	ISBN	Title	Image	Author	Publisher	Date_Published	Cond	Description
1	9781461471370	An Introduction to Statistical Learning:...	An-Introduction-To...	Gareth James, Dani...	Springer	2017-09-01	MINT	An Introduction to Statistical Learning provides...
2	9780525568292	Cracking the AP Physics 1 Exam 2020...	AP-Physics-Exam....	The Princeton Review	Princeton Review	2019-08-06	DECENT	PREMIUM PRACTICE FOR A PERFECT 5! A...
3	9780323393225	Atlas of Human Anatomy	Atlas-Of-Human-An...	Frank H. Netter MD	Elsevier Health S...	2018-03-13	DECENT	The only anatomy atlas illustrated by physician...
7	9781454894018	Basic Legal Research: Tools and Strat...	basic-legal-researc...	Amy E. Sloan	Wolters Kluwer	2018-02-28	MINT	Basic Legal Research: Tools and Strategies for...
8	9781543805727	Strategies & Tactics for the MBE (Ema...	518hHIIr9HL..._SX3...	Steven L. Emmanuel	Wolters Kluwer	2016-05-22	DECENT	The Seventh Edition of Strategies & Tactics fo...
9	9781469893419	Bates' Guide to Physical Examination	Bates-Guide-To-Ph...	Lynn S. Bickley	LWW	2016-09-09	LIKE NEW	Bates' Guide to Physical Examination and His...
10	9780984782857	Cracking the Coding Interview	Cracking-The-Codin...	Gayle Laakmann M...	CareerCup	2015-07-01	LIKE NEW	I am not a recruiter. I am a software engineer.
11	9781479274833	Elements of Programming Interviews i...	Elements-Of-Progra...	Adnan Aziz, Tsung-...	CreateSpace Ind...	2016-09-15	MINT	The core of EPI is a collection of over 250 pro...
12	9780133915389	Engineering Mechanics: Dynamics 14t...	Engineering_Mecha...	Russell C. Hibbeler	Pearson	2015-04-10	DECENT	A Proven Approach to Conceptual Understanding...
13	9780714833569	Modern Architecture Since 1900 / Editi...	modern-architecture...	William J.R. Curtis	Phaidon Press	1996-07-27	FAIR	Since its first publication in 1982, Modern Arch...
14	9780071592536	Security Analysis, Sixth Edition	Security-Analysis_3...	Benjamin Graham,...	McGraw-Hill Edu...	2008-09-25	MINT	"A road map for investing that I have now bee...
15	9781563684449	The American Sign Language Handsh...	51z5vHUIBpL..._SX3...	Richard A. Tennant,...	Gallaudet Univers...	2010-07-31	DECENT	Now, the bestselling resource The American Sign...
20	9780367188498	Theories and Methods of Writing Cent...	Theories_and_Meth...	Jo Mackiewicz, Reb...	Routledge	2019-11-13	LIKE NEW	This collection helps students and researchers...
22	9780395872741	World History: Patterns of Interaction	World_History_Patt...	McDougal Littell	McDougal Littell	1998-02-02	DECENT	World History Textbook has cover scratches a...
24	9780465060733	Basic Economics	Basic-Economics.jpg	Thomas Sowell	Basic Books	2014-12-02	MINT	example description
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Textbook 8 Apply

As you can see in the last row, the new textbook ISBN, Title, Image, Author, Publisher, Date_Published, Cond, Description information has been inserted into the Textbook table.

The screenshot shows the MySQL Workbench interface. The query editor at the top contains the following SQL code:

```

1 • use store;
2 • SELECT * From Category_Has_Textbook;

```

The result grid below displays the data from the Category_Has_Textbook table:

id	Category_Name	Textbook_ID
34	Architecture	13
40	Communication	15
98	Communication	20
2	Computer Science	1
29	Computer Science	10
30	Computer Science	11
37	Engineering	12
36	Finance	14
100	Finance	24
90	History	22
27	Law	7
23	Law	8
28	Literature	7
99	Literature	20
1	Mathematics	1
3	Mathematics	2
39	Mathematics	12
5	Medicine	3
94	Medicine	9
4	Physics	2
38	Physics	12
NULL	NULL	NULL

The status bar at the bottom indicates "Category_Has_Textbook 9".

As you can see, a new row is inserted into the table Category_Has_Textbook. The 9th row which has id = 100, Category_Name = Finance, and Textbook_ID = 24 represents that the new textbook we just listed belong to the Category Finance. If you selected multiple categories that your textbook belongs to, each category selected will be a row to be inserted into this table.

The screenshot shows the MySQL Workbench interface. The top bar has a 'Query 1' tab, several icons for file operations, and a 'Limit to 1000 rows' button. Below the toolbar, a code editor displays the following SQL statements:

```
1 • use store;
2 • SELECT * FROM Listing;
```

The status bar at the bottom left shows '100%' and '23:2'. The main area is titled 'Result Grid' and contains a table with the following data:

id	Price	Account_id	Textbook_ID	Available
1	300	7	1	1
2	50	13	2	1
3	75	3	3	0
7	78	1	7	1
8	165	1	8	1
9	60	1	9	1
10	98	6	10	1
11	87	8	11	0
12	250	25	12	0
13	50	30	13	1
14	99	20	14	0
16	140	7	15	1
19	150	1	20	1
21	78	33	22	0
23	75	52	24	1
NULL	NULL	NULL	NULL	NULL

In the listing table, a new row is also inserted. In the very last row, a new row of Price = 75, Account_id = 52, Textbook_id = 24, and Available = 1 represents the new listing for the new textbook we just inserted. Price is the listed price of the textbook, Account_id is the foreign key of Account or the account that listed the textbook, Textbook_id is the foreign key of Textbook which is the new textbook inserted, and Available is the boolean field which indicates if this listing has been sold yet or not. If Available is 1, then the Listing is still available; if it has already sold, then Available field will be set to 0.

BACKEND LOGIC

```
views.py
318     return HttpResponseRedirect(reverse('homepage'))
319
320 @login_required
321 @require_POST
322 def add_listing_request(request):
323
324     # Getting the textbook information from the POST request
325     ISBN = request.POST.get('ISBN')
326     Title = request.POST.get('Title')
327     Author = request.POST.get('Author')
328     Publisher = request.POST.get('Publisher')
329     Cond = request.POST.get('Cond')
330     Date_Published = request.POST.get('Date_Published_year') + "-" + request.POST.get('Date_Published_month') + "-" + request.POST.get('Date_Published_day');
331     Price = request.POST.get('Price')
332     Description = request.POST.get('Description')
333     Image = request.FILES.get('Image')
334     # This just create a Django form with the data sends in by the POST request
335     form = forms.TextbookCreationForm(request.POST);
336     Categories = request.POST.getlist('Category')
337     # This checks if the data received by the POST request is valid
338     if form.is_valid():
339         Textbook_id = None;
340         with connection.cursor() as cursor:
341             insert_textbook_query = """
342                 INSERT INTO
343                     Textbook (ISBN, Title, Author, Publisher, Date_Published, Description, Cond)
344                     VALUES (%s, %s, %s, %s, %s, %s, %s)
345                 """
346
347             # Insert a new textbook into the database using the query above
348             cursor.execute(insert_textbook_query,(ISBN, Title, Author, Publisher, Date_Published, Description, Cond));
349             # Get the last row ID of cursor which is the textbook_id of the new textbook we just inserted into the database
350             Textbook_id = cursor.lastrowid
351             # For each category that the textbook belongs to, insert a corresponding Category_Has_Textbook row
352             for C in Categories:
353                 temp = Category.objects.raw("Select * From Category Where Category_Name = %s", [C])[0]
354                 cursor.execute("INSERT INTO Category_Has_Textbook (Category_Name, Textbook_ID) VALUES (%s, %s)", [temp.Category_Name, Textbook_id])
355             cursor.execute("INSERT INTO Listing (Price, Account_id, Textbook_ID, Available) VALUES (%s, %s, %s, %s)", (Price, request.user.id, Textbook_id, '1'))
356             connection.close()
357
358             book = Textbook.objects.raw("Select * From Textbook WHERE Textbook_ID= %s", [Textbook_id])[0]
359             book.Image = Image;
360             book.save();
361     else:
362         print(form.errors);
363     return HttpResponseRedirect(reverse('homepage'))
```

This is the Python code I used to handle the add listing functionality. First, I retrieve all the textbook information from the POST request and perhaps format them a bit. After that, I established a connection to the database and my SQL query to insert a new textbook into the database is

```

Insert_textbook_query =  

"""  

    INSERT INTO  

        Textbook (ISBN, Title, Author, Publisher, Date_Published, Description, Cond)  

    VALUES (%s, %s, %s, %s, %s, %s)  

"""

```

Note: Cond here stands for Condition (of the textbook)

I invoke cursor.execute(insert_textbook_query,(ISBN, Title, Author, Publisher, Date_Published, Description, Cond)); to run this query and insert in the new textbook.

The %s are just the pythonic way to prevents MySQL Injection Attacks. I passed in a list containing ISBN, Title, Author, Publisher, Date_Published, Description, Cond into the execute statement and it means that the first %s will be replaced with the value of ISBN, the 2nd %s will be replaced with the value of Title and so on.

After that, this part here handle inserting into the Category_Has_Textbook relationship that represents what Category a textbook belongs to. This for loop represents that for every category selected by the user, I will insert a new row into Category_Has_Textbook of the corresponding Category_Name and the Textbook_ID of the new textbook inserted.

```

for C in Categories:  

    temp = Category.objects.raw("Select * From Category Where Category_Name = %s",
                                [C])[0]  

    cursor.execute("INSERT INTO Category_Has_Textbook (Category_Name, Textbook_ID)  

    VALUES (%s, %s)", [temp.Category_Name, Textbook_id])

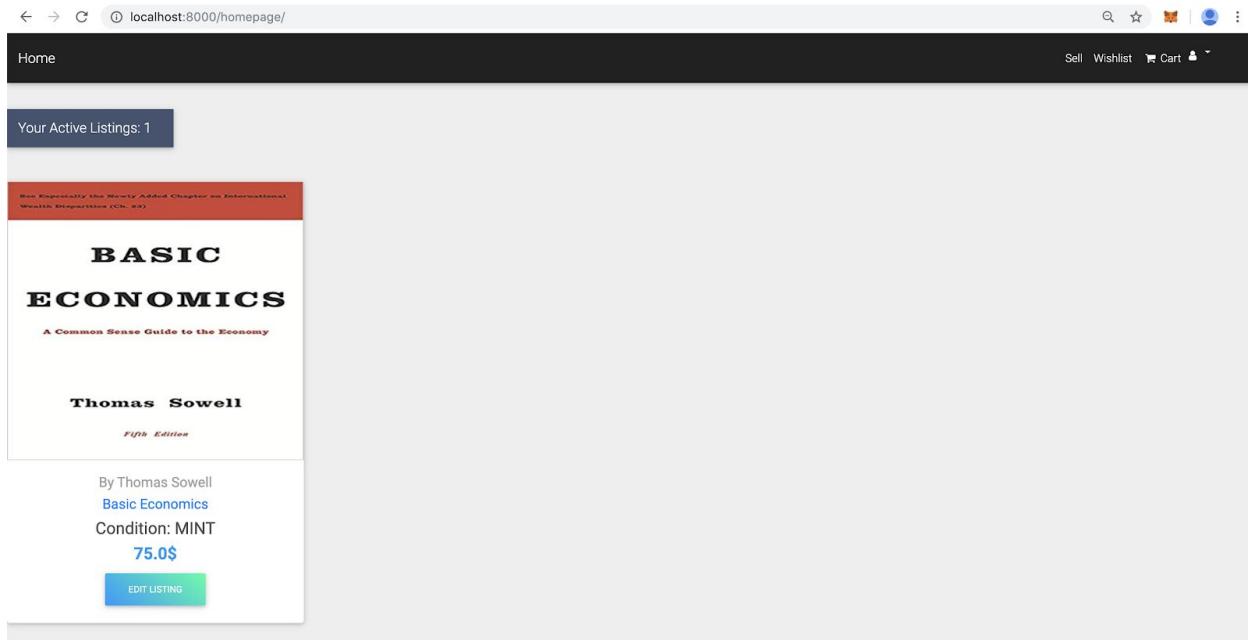
```

```
cursor.execute("INSERT INTO Listing (Price, Account_id, Textbook_ID, Available) VALUES  
(%s, %s, %s, %s)", (Price, request.user.id, Textbook_id, '1'))
```

Finally, this part inserts a new row into the Listing table that represents a new textbook listing. I inserts the Price, foreign key Account_id, foreign_key Textbook_ID, and Available as 1 to marks that it is available for sale.

EDIT LISTING

For this section, I will use UPDATE to update rows in the Listing table, Textbook table, or Category_Has_Textbook table



In the homepage, scrolling down you can see the list of your active listings and a button to Edit your Listing information. Clicking it will invoke this function:

```

188 @login_required
189 def view_product_page(request, textbook_id):
190     query = """
191     SELECT *
192     FROM Listing A, Textbook B, Account C
193     WHERE A.Available = 1 AND A.Account_id = C.id AND A.Textbook_ID = B.Textbook_ID AND A.Textbook_ID = %s
194     """
195     listing = Listing.objects.raw(query, [textbook_id])[0];
196
197     Cart = Shopping_Cart.objects.raw("SELECT * FROM Shopping_Cart WHERE Account_ID = %s AND Textbook_ID = %s", [str(request.user.id), textbook_id])
198     Wish_list = Wishlist.objects.raw("SELECT * FROM Wishlist WHERE Account_ID = %s AND Textbook_ID = %s", [str(request.user.id), textbook_id])
199     Cart = Cart[:]
200     Wish_list = Wish_list[:]
201     Listing_In_Cart = False
202     Listing_In_Wishlist = False
203     if len(Cart) == 1:
204         Listing_In_Cart = True
205     if len(Wish_list) == 1:
206         Listing_In_Wishlist = True
207     # If the listing is listed by the user, then return a view that lets the user edit the listing information instead
208     if listing.username == request.user.username:
209         Category_Query = """
210         SELECT id, A.Category_Name
211         FROM Category_Has_Textbook A, Category B
212         WHERE B.Category_Name = A.Category_Name AND A.Textbook_ID = %s
213         """
214     Categories = Category.objects.raw(Category_Query, [textbook_id])
215
216     # This part retrieves the information from the listing query at the beginning and formats it into a dictionary
217     # the dictionary is then used to initialize a TextbookChangeForm to create an edit listing information form
218     # prefilled with the textbook listing information
219     data = {'ISBN': listing.ISBN, 'Title': listing.Title, 'Author': listing.Author, 'Publisher': listing.Publisher, 'Cond': listing.Cond,
220             'Date_Published': listing.Date_Published, 'Price': listing.Price, 'Description': listing.Description, 'Image': listing.Image
221             , 'Category': Categories}
222     form = forms.TextbookChangeForm(initial=data)
223     return render(request, 'My_Product_Page.html', context={'form': form, 'listing': listing})
224
225     return render(request, 'Product_Page.html', context={'listing': listing, 'Cart': Listing_In_Cart, 'Wishlist': Listing_In_Wishlist})
226

```

This function handle returning the view when you click the edit/view button associated with each textbook listing. If the listing is not listed by you, it will return a render of Product_Page.html otherwise if listing.username == request.user.username (you listed the textbook listing) then it will return a render of My_Product_Page.html which will bring up a form for me to edit my listing.

First, this function has a passed in argument textbook_id which is the textbook_id associated with the listing you clicked on to edit.

```

query =
"""
SELECT *
FROM Listing A, Textbook B, Account C
WHERE A.Available = 1 AND A.Account_id = C.id AND A.Textbook_ID = B.Textbook_ID
AND A.Textbook_ID = %s

```

```
"""
```

I join table Listing, Textbook, Account where Listing.Available = 1 (listing for sale), Listing.Account_id = Account.Account_id (this is the seller's account id), Listing.Textbook_ID = Textbook.Textbook_ID AND Listing.Textbook_ID = textbook_id. This returns me the listing information, textbook information, and seller account id of the textbook I clicked on for edit.

Then I do another query,

```
Category_Query =
```

```
"""
```

```
SELECT id, A.Category_Name  
FROM Category_Has_Textbook A, Category B  
WHERE B.Category_Name = A.Category_Name AND A.Textbook_ID = %s
```

```
"""
```

This query joins the Category_Has_Textbook and Category table based on their Category_Name and has a specific textbook id. This finds all the category that the listed textbook belongs to.

The result of these 2 query is passed into the render of My_Product_Page.html that returns the view.

Query 1

Limit to 1000 rows

```
1 • use store;
2 • SELECT *
3 FROM Listing A, Textbook B, Account C
4 WHERE A.Available = 1 AND A.Account_id = C.id AND A.Textbook_ID = B.Textbook_ID AND A.Textbook_ID = 26;
```

100% 103:4

Result Grid Filter Rows: Search Export:

id	Price	Account_id	Textbook_ID	Available	Textbook_ID	ISBN	Title	Image	Author
25	50	1	26	1	26	9780465060733	Basic Economics	Basic-Economics_Q...	Thomas Sowell

This is the query I used to retrieve information about the textbook and use it to prefill my input fields for
Edit Listing form

Query 1

Limit to 1000 rows

```

1 •  use store;
2 •  SELECT id, A.Category_Name
3   FROM Category_Has_Textbook A, Category B
4   WHERE B.Category_Name = A.Category_Name AND A.Textbook_ID = 26;

```

100% 64:4

Result Grid Filter Rows: Search Export:

id	Category_Name
115	Finance

This is the query I used to determine which Category the textbook belonged to and used it to prefill my “Category” select option for Edit Listing form.

localhost:8000/textbook/24/

Edit Listing

ISBN: 9780465060733 Title: Basic Economics

Author: Thomas Sowell

Publisher: Basic Books

Condition: BRAND NEW Date Published: December 2 2014

Price: 75.0

Category:

- Architecture
- Biology
- Communication
- Computer Science
- Engineering
- Finance
- Humanities
- History
- Law
- Literature
- Mathematics
- Medicine
- Physics

Description: example description

BASIC ECONOMICS
A Common Sense Guide to the Economy
Thomas Sowell
Choose File No file chosen
EDIT LISTING

As you can see, it returns a form similar to the add listing form but this form is already filled with the information about the textbook listing along with the categories it belong to. You can change any of the information and clicks Edit Listing button to commit the changes to our database.

The screenshot shows a web application interface for editing a book listing. On the left, there's a sidebar with navigation links like Home, Sell, Wishlist, and Cart. The main area has a title 'Edit Listing'. It contains fields for ISBN (9780463060733), Title (Basic Economics), Author (Thomas Sowell), Publisher (Basic Books), Condition (BRAND NEW), Date Published (December 2014), Price (76.0), and a Category dropdown menu with various options like Architecture, Biology, Communication, Computer Science, Engineering, Finance, History, Humanities, Law, Literature, Mathematics, Medicine, and Physics. Below these fields is a 'Description' text area containing 'modified description'. To the right, there's a preview of the book cover for 'Basic Economics' by Thomas Sowell, showing the title, author, and a 'Choose File' button. At the bottom right is a green 'EDIT LISTING' button.

I will change the Author from Thomas Sowell to James Carter, the description from 'example description' to 'Modified Listing' and then clicks Edit Listing to commit the changes to my database.

1 • use store;
 2 • SELECT * From Textbook;

Result Grid Filter Rows: Search Edit: Export/Import:

Textbook_ID	ISBN	Title	Image	Author	Publisher	Date_Published	Cond	Description
1	9781461471370	An Introduction to Statistical Learning:...	An-Introduction-To-S...	Gareth James, Dani...	Springer	2017-09-01	MINT	An Introduction to Statistical Learning provide...
2	9780525568292	Cracking the AP Physics 1 Exam 2020...	AP-Physics-Exam_...	The Princeton Review	Princeton Review	2019-08-06	DECENT	PREMIUM PRACTICE FOR A PERFECT 5! A...
3	9780323393225	Atlas of Human Anatomy	Atlas-Of-Human-An...	Frank H. Netter MD	Elsevier Health S...	2018-03-13	DECENT	The only anatomy atlas illustrated by physician...
7	9781454894018	Basic Legal Research: Tools and Strat...	basic-legal-research...	Amy E. Sloan	Wolters Kluwer	2018-02-28	MINT	Basic Legal Research: Tools and Strategies, v...
8	9781543805727	Strategies & Tactics for the MBE (Ema...	518hHilf9HL..._SX3...	Steven L. Emmanuel	Wolters Kluwer	2016-05-22	DECENT	The Seventh Edition of Strategies & Tactics fo...
9	9781469893419	Bates' Guide to Physical Examination	Bates-Guide-To-Phy...	Lynn S. Bickley	LWW	2016-09-09	LIKE NEW	Bates' Guide to Physical Examination and His...
10	9780984782857	Cracking the Coding Interview	Cracking-The-Codin...	Gayle Laakmann M...	CareerCup	2015-07-01	LIKE NEW	I am not a recruiter. I am a software engineer...
11	9781479274833	Elements of Programming Interviews i...	Elements-Of-Progra...	Adnan Aziz, Tsung...	CreateSpace Ind...	2016-09-15	MINT	The core of EPI is a collection of over 250 pro...
12	9780133915389	Engineering Mechanics: Dynamics 14t...	Engineering_Mecha...	Russell C. Hibbeler	Pearson	2015-04-10	DECENT	A Proven Approach to Conceptual Understandin...
13	9780714833569	Modern Architecture Since 1900 / Editi...	modern-architecture...	William J.R. Curtis	Phaidon Press	1996-07-27	FAIR	Since its first publication in 1982, Modern Arc...
14	9780071592536	Security Analysis, Sixth Edition	Security-Analysis_3...	Benjamin Graham,...	McGraw-Hill Edu...	2008-09-25	MINT	"A road map for investing that I have now bee...
15	9781563668449	The American Sign Language Handsh...	51z5vHULBpL..._SX3...	Richard A. Tennant,...	Gallaudet Univers...	2010-07-31	DECENT	Now, the bestselling resource The American S...
20	9780367118849	Theories and Methods of Writing Cent...	Theories_and_Meth...	Jo Mackiewicz, Reb...	Routledge	2019-11-13	LIKE NEW	This collection helps students and researcher...
22	9780395872741	World History: Patterns of Interaction	World_History_Patte...	McDougal Littell	McDougal Littell	1998-02-02	DECENT	World History Textbook has cover scratches a...
24	9780465060733	Basic Economics	Basic-Economics.jpg	James Carter	Basic Books	2014-12-02	MINT	modified description
25	9780134093413	Campbell Biology	Campbell-Biology.jpg	Lisa A. Urry, Michael...	Pearson	2016-10-29	MINT	The Eleventh Edition of the best-selling Camp...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Textbook 1 Apply R

As you can see in the 2nd to last row, the author has been changed to James Carter and Description has been changed to modified.

```

267     @login_required
268     @require_POST
269     def edit_listing_request(request):
270
271         form = forms.TextbookChangeForm(request.POST)
272         if form.is_valid():
273
274             # Get the Date Published and list of category that the textbook belong to information
275             Date_Published = request.POST.get('Date_Published_year') + "-" + request.POST.get('Date_Published_month') + "-" + request.POST.get('Date_Published_day');
276             Category_List = request.POST.getlist('Category')
277
278             with connection.cursor() as cursor:
279
280                 # Select every row in Category_Has_Textbook table that this textbook listing belongs to
281                 Categories = Category_Has_Textbook.objects.raw("SELECT * FROM Category_Has_Textbook WHERE Textbook_ID = %s", [request.POST.get('textbook_id')])
282                 # If the textbook no longer belong to that Category, DELETE the corresponding Category_Has_Textbook row
283                 for C in Categories:
284                     if C.Category not in Category_List:
285                         cursor.execute("DELETE FROM Category_Has_Textbook WHERE Category_Name = %s AND Textbook_ID = %s", [C.Category.Category_Name, request.POST.get('textbook_id')])
286
287                 # Query to find the list of categories that the textbook still belongs to after deleting the unselected categories
288                 Categories = Category_Has_Textbook.objects.raw("SELECT * FROM Category_Has_Textbook WHERE Textbook_ID = %s", [request.POST.get('textbook_id')])
289
290                 # Slices it to unwrap the RawQueryset into a list
291                 temp = Categories[1:]
292                 Categories = []
293
294                 # Append the name of each category that the textbook belongs to into a list
295                 # This makes a LIST of all the name of the category this textbook belong to!
296                 for t in temp:
297                     Categories.append(t.Category)
298
299                 # If a category that the textbook belongs is in both Categories and Category_List
300                 # then there already exists a Category_Has_Textbook row for that Category and this Textbook so do nothing
301                 # Else insert a new Category_Has_Textbook row
302                 for C in Category_List:
303                     if C in Categories:
304                         pass
305                     else:
306                         cursor.execute("INSERT INTO Category_Has_Textbook (Category_Name, Textbook_ID) VALUES (%s, %s)", [C, request.POST.get('textbook_id')])
307
308                 Book = Textbook.objects.raw("SELECT * FROM Textbook WHERE Textbook_ID = %s", [request.POST.get('textbook_id'))][0]
309                 # If the textbook image is also edited, then make an update query including the Image column
310                 # If not then make an update query without including the Image column
311                 if request.FILES.get('Image') != None:
312                     update_textbook_query = """
313                         UPDATE Textbook
314                         SET ISBN = %s, Title = %s, Author = %s, Publisher = %s,
315                         Cond = %s, Description = %s, Date_Published = %s, Image = %s
316                         WHERE Textbook_ID = %s;
317                     """
318
319                 update_textbook_arguments = [request.POST.get('ISBN'), request.POST.get('Title'), request.POST.get('Author'), \
320                     request.POST.get('Publisher'), request.POST.get('Cond'), request.POST.get('Description'), Date_Published,\ 
321                     request.FILES.get('Image').name, request.POST.get('textbook_id')]
322
322                 cursor.execute(update_textbook_query, update_textbook_arguments)
323                 Book.Image = request.FILES.get('Image')
324                 Book.save()
325
326             else:
327                 update_textbook_query = """
328                     UPDATE Textbook
329                     SET ISBN = %s, Title = %s, Author = %s, Publisher = %s,
330                     Cond = %s, Description = %s, Date_Published = %s
331                     WHERE Textbook_ID = %s;
332                 """
333
334                 update_textbook_arguments = [request.POST.get('ISBN'), request.POST.get('Title'), request.POST.get('Author'), \
335                     request.POST.get('Publisher'), request.POST.get('Cond'), request.POST.get('Description'), Date_Published, request.POST.get('textbook_id')]
336
337                 cursor.execute(update_textbook_query, update_textbook_arguments)
338
339             # Update the price of the listing
340             update_listing_query = "UPDATE Listing SET Price = %s WHERE Textbook_ID = %s;"
341             update_listing_arguments = [request.POST.get('Price'), request.POST.get('textbook_id')]
342
343             cursor.execute(update_listing_query, update_listing_arguments)
344
345             connection.close()
346             # Redirects to homepage
347             return HttpResponseRedirect(reverse('homepage'))

```

```

307
308             Book = Textbook.objects.raw("SELECT * FROM Textbook WHERE Textbook_ID = %s", [request.POST.get('textbook_id'))][0]
309             # If the textbook image is also edited, then make an update query including the Image column
310             # If not then make an update query without including the Image column
311             if request.FILES.get('Image') != None:
312                 update_textbook_query = """
313                     UPDATE Textbook
314                     SET ISBN = %s, Title = %s, Author = %s, Publisher = %s,
315                     Cond = %s, Description = %s, Date_Published = %s, Image = %s
316                     WHERE Textbook_ID = %s;
317                 """
318
319             update_textbook_arguments = [request.POST.get('ISBN'), request.POST.get('Title'), request.POST.get('Author'), \
320                 request.POST.get('Publisher'), request.POST.get('Cond'), request.POST.get('Description'), Date_Published,\ 
321                 request.FILES.get('Image').name, request.POST.get('textbook_id')]
322
322             cursor.execute(update_textbook_query, update_textbook_arguments)
323             Book.Image = request.FILES.get('Image')
324             Book.save()
325
326         else:
327             update_textbook_query = """
328                 UPDATE Textbook
329                 SET ISBN = %s, Title = %s, Author = %s, Publisher = %s,
330                 Cond = %s, Description = %s, Date_Published = %s
331                 WHERE Textbook_ID = %s;
332             """
333
334             update_textbook_arguments = [request.POST.get('ISBN'), request.POST.get('Title'), request.POST.get('Author'), \
335                 request.POST.get('Publisher'), request.POST.get('Cond'), request.POST.get('Description'), Date_Published, request.POST.get('textbook_id')]
336
337             cursor.execute(update_textbook_query, update_textbook_arguments)
338
339             # Update the price of the listing
340             update_listing_query = "UPDATE Listing SET Price = %s WHERE Textbook_ID = %s;"
341             update_listing_arguments = [request.POST.get('Price'), request.POST.get('textbook_id')]
342
343             cursor.execute(update_listing_query, update_listing_arguments)
344
345             connection.close()
346             # Redirects to homepage
347             return HttpResponseRedirect(reverse('homepage'))

```

This edit_listing_request function handles the updating information about the textbook listing.

First, it selects every row in Category_Has_Textbook with a specific textbook_id

```
-----  
Categories = Category_Has_Textbook.objects.raw("SELECT * FROM Category_Has_Textbook  
WHERE Textbook_ID = %s", [request.POST.get('textbook_id')])  
-----
```

Then from the list of category edited by the user, it will runs

```
-----  
for C in Categories:  
  
    if C.Category not in Category_List:  
  
        cursor.execute("DELETE FROM Category_Has_Textbook WHERE Category_Name = %s AND  
Textbook_ID = %s", [C.Category.Category_Name, request.POST.get('textbook_id')])  
-----
```

This part checks if the textbook STILL belongs to those Categories. If not, then it will DELETE that row from Category_Has_Textbook

```
-----  
Categories = Category_Has_Textbook.objects.raw("SELECT * FROM Category_Has_Textbook  
WHERE Textbook_ID = %s", [request.POST.get('textbook_id')])  
  
for C in Category_List:  
  
    if C in Categories:  
  
        pass  
  
    Else:  
  
        cursor.execute("INSERT INTO Category_Has_Textbook (Category_Name, Textbook_ID)  
VALUES (%s, %s)", [C, request.POST.get('textbook_id')])  
-----
```

This part requery the Categories that the textbook still belongs to after you deleted the categories no longer selected by the user. Then it checks if the categories selected already has a row in the Category_has_Textbook table, if yes, then it does nothing, else it will INSERT a new row into Category_Has_Textbook table representing a new category that the textbook now belongs to.

Then I simply update the Textbook table with this query:

```
update_textbook_query =  
    """  
        UPDATE Textbook  
        SET ISBN = %s, Title = %s, Author = %s, Publisher = %s,  
        Cond = %s, Description = %s, Date_Published = %s, Image = %s  
        WHERE Textbook_ID = textbook_id;  
    """
```

This update the Textbook row with Textbook_ID = textbook_id. If the Image is not changed, then I used another query that is the same except it excludes the Image = %s.

If column Title is changed, then this query will change this in the database. If column ISBN was not changed, this query will recognize that the value is the same and won't update the ISBN column.

Finally, one last query is used to update the Listing table

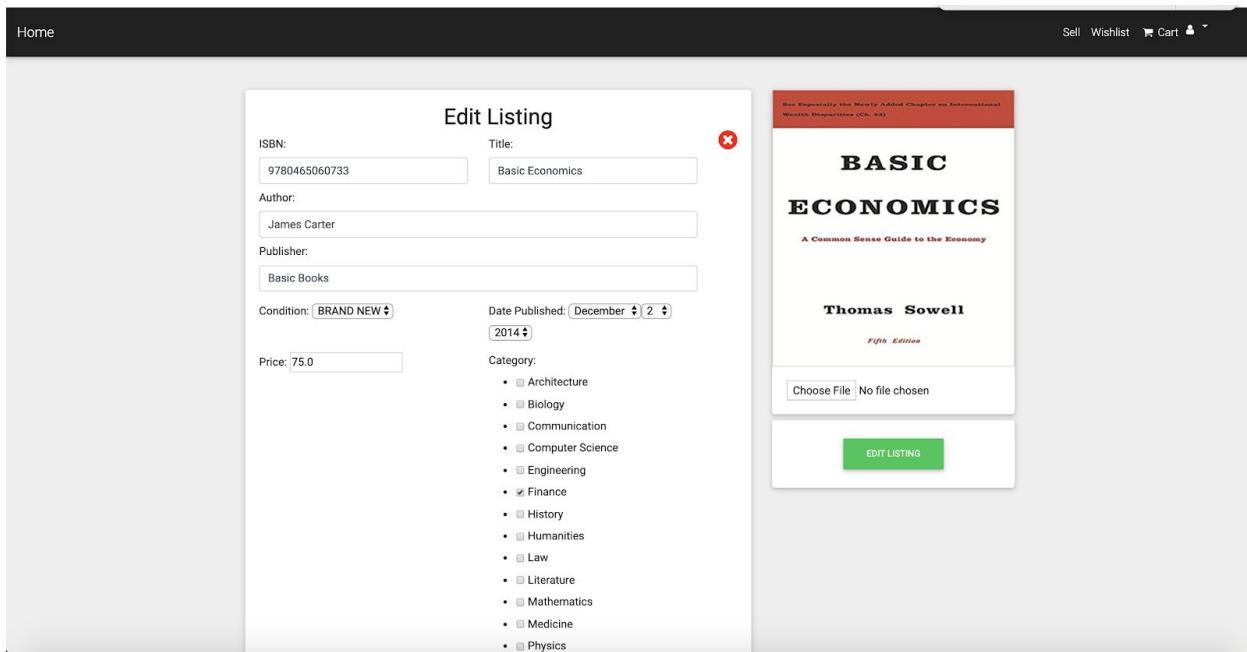
```
Update_listing_query =  
    """  
        UPDATE Listing  
        SET Price = %s  
        WHERE Textbook_ID = %s;  
    """
```

This simply updates the Price column in a row in Listing filtering by Textbook_ID

Then the function return a HttpResponseRedirect(reverse('homepage')) which redirects you to the homepage.

Cancel Listing Functionality

For this feature, I will delete a row from Textbook table, a row from Listing table, one or more rows from Category_Has_Textbook table, one or more rows from Wishlist table, and one or more rows from Shopping Cart table.

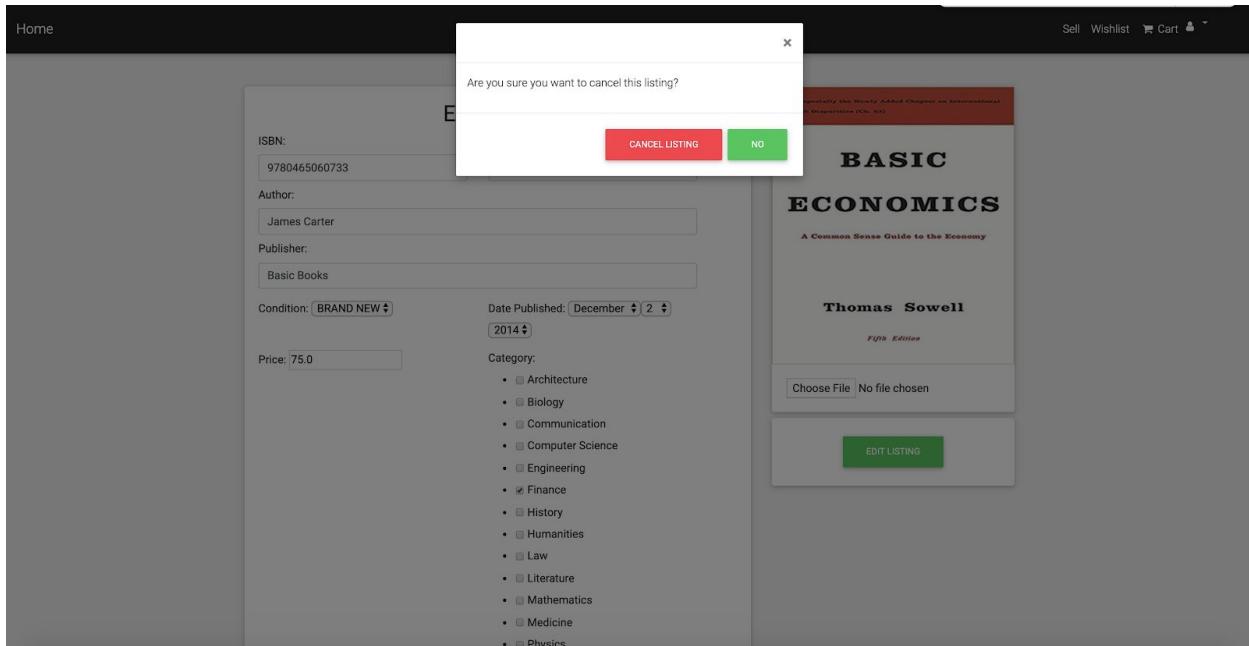


The screenshot shows a web-based application interface for managing book listings. On the left, there's a sidebar with 'Home', 'Sell', 'Wishlist', 'Cart', and a user icon. The main area has a title 'Edit Listing' and a form for updating a listing. The form fields include:

- ISBN: 9780465060733
- Title: Basic Economics
- Author: James Carter
- Publisher: Basic Books
- Condition: BRAND NEW
- Date Published: December 2, 2014
- Price: 75.0
- Category: (checkboxes) Architecture, Biology, Communication, Computer Science, Engineering, Finance (checked), History, Humanities, Law, Literature, Mathematics, Medicine, Physics

To the right of the form is a thumbnail image of the book 'Basic Economics' by Thomas Sowell, Fifth Edition. The book cover features the title 'BASIC ECONOMICS' and the subtitle 'A Common Sense Guide to the Economy'. Below the image is a file upload field labeled 'Choose File' with the message 'No file chosen' and a green 'EDIT LISTING' button.

To cancel a listing, click on Edit Listing on one of your active listings and then click on the red X on the upper right corner of the Edit Listing form.



A popup modal window will appear asking if you are sure you want to cancel the listing, click

Cancel Listing to go ahead and cancel the listing

This is the row in Textbook table with Textbook_ID = 24, the textbook that the account I am currently logged into have listed for sale

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following code:

```

1 • use store;
2 • SELECT * FROM Textbook WHERE Textbook_id = 24;
3
4
5
6
7

```

The result grid shows a single row with all columns containing NULL values, indicating the textbook has been deleted.

Textbook_ID	ISBN	Title	Image	Author	Publisher	Date_Published	Cond	Description
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

After canceling the listing, the query returns NULL as the textbook has been deleted from the database.

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following code:

```

1 • use store;
2 • SELECT * FROM Listing WHERE Account_id = 52;

```

The result grid shows one row with the following data:

id	Price	Account_id	Textbook_ID	Available
23	75	52	24	1

This is the row in Listing table with account_id = 52 (the account I am logged in to) and this shows that the account has an active listing for textbook_id 24

After canceling the listing, the query result return NULL as the listing has been deleted from the database and now the account I'm logged into has no active listings.

The screenshot shows a MySQL Workbench interface with a query editor and a results grid. The query editor contains the following code:

```
1 • use store;
2 • SELECT * FROM Listing;
```

The results grid displays the following data:

	id	Price	Account_id	Textbook_ID	Available
▶	1	300	7	1	1
	2	50	13	2	1
	3	75	3	3	0
	7	78	1	7	1
	8	165	1	8	1
	9	60	1	9	1
	10	98	6	10	0
	11	87	8	11	0
	12	250	25	12	0
	13	50	30	13	1
	14	99	20	14	0
	16	140	7	15	1
	19	150	1	20	1
	21	78	33	22	0
	23	75	52	24	1
	24	30	1	25	1
	NULL	NULL	NULL	NULL	NULL

In Listing table, the second to last row is the row that displays account_id 52 (the account I am currently logged into) has listed for sale textbook_id 24 and the status of Available is 1 (available for sale)

The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons for database management. Below the toolbar, a query editor window titled "Query 1" contains the following SQL code:

```
1 •  use store;
2 •  SELECT * FROM Shopping_Cart;
```

Below the code, a progress bar indicates "100%" completion and a timestamp "28:2". The main area displays a "Result Grid" with the following data:

id	Account_id	Textbook_ID
55	1	24
HULL	HULL	HULL

This one row in the Shopping Cart table that the account with id 1 has our textbook id 24 in their shopping cart.

⚡ Query 1

Limit to 1000 rows

1 • use store;
2 • SELECT * FROM Shopping_Cart;

3
4
5
6
7

100% 24:2

Result Grid Filter Rows: Search Edit: Export/Import:

id	Account_id	Textbook_ID
NULL	NULL	NULL

After canceling the listing, the query result returns NULL since deleting the textbook listing also delete the item from every account's shopping cart.

The screenshot shows the MySQL Workbench interface. The top bar has a 'Query 1' tab and various icons for database management. Below the toolbar, there's a dropdown for 'Limit to 1000 rows' and some status indicators. The main area contains a query editor with the following code:

```
1 • use store;  
2 • SELECT * FROM Wishlist;
```

The bottom part of the interface shows a 'Result Grid' with the following data:

id	Account_id	Textbook_ID
27	1	24
NULL	NULL	NULL

This one row in the Wishlist table shows that the account with id 1 has our textbook id 24 in their wishlist.

⚡ Query 1

Execute the selected portion of the script or everything, if there is no selection

Limit to 1000 rows

1 • use store;
2 • SELECT * FROM Wishlist;

3
4
5
6
7

100% 23:2

Result Grid Filter Rows: Search Edit: Export/Import:

id	Account_id	Textbook_ID
NULL	NULL	NULL

After canceling the listing, the query result returns NULL as deleting the textbook listing also delete it from every account's wishlist.

BACKEND LOGIC

```
251
252 @login_required
253 @require_POST
254 @csrf_exempt
255 def cancel_listing_request(request):
256     context={}
257     if request.is_ajax() == True:
258         textbook_id = request.POST.get('textbook-id')
259         with connection.cursor() as cursor:
260             # DELETE all Category_Has_Textbook, Listing, Wishlist, Shopping_Cart rows that the textbook belongs to
261             # and then delete the Textbook
262             cursor.execute("DELETE FROM Category_Has_Textbook WHERE Textbook_ID = %s;", [textbook_id])
263             cursor.execute("DELETE FROM Listing WHERE Textbook_ID = %s;", [textbook_id])
264             cursor.execute("DELETE FROM Wishlist WHERE Textbook_ID = %s;", [textbook_id])
265             cursor.execute("DELETE FROM Shopping_Cart WHERE Textbook_ID = %s;", [textbook_id])
266             cursor.execute("DELETE FROM Textbook WHERE Textbook_ID = %s;", [textbook_id])
267             connection.close()
268             context['message'] = 'Success'
269             context['redirect'] = reverse('homepage')
270     else:
271         context['message'] = 'Fail'
272     return JsonResponse(context)
273
```

To cancel a textbook listing, I need to delete all relevant information about the textbook listing from my database.

```
cursor.execute("DELETE FROM Category_Has_Textbook WHERE Textbook_ID = %s;",
[textbook_id])

cursor.execute("DELETE FROM Listing WHERE Textbook_ID = %s;", [textbook_id])

cursor.execute("DELETE FROM Wishlist WHERE Textbook_ID = %s;", [textbook_id])

cursor.execute("DELETE FROM Shopping_Cart WHERE Textbook_ID = %s;", [textbook_id])

cursor.execute("DELETE FROM Textbook WHERE Textbook_ID = %s;", [textbook_id])
```

I delete all category that contains the textbook in Category_Has_Textbook, delete the listing itself in Listing table. Delete the textbook from every account's Wishlist and also delete it from every account's Shopping Cart. Finally, I delete the Textbook entry itself.

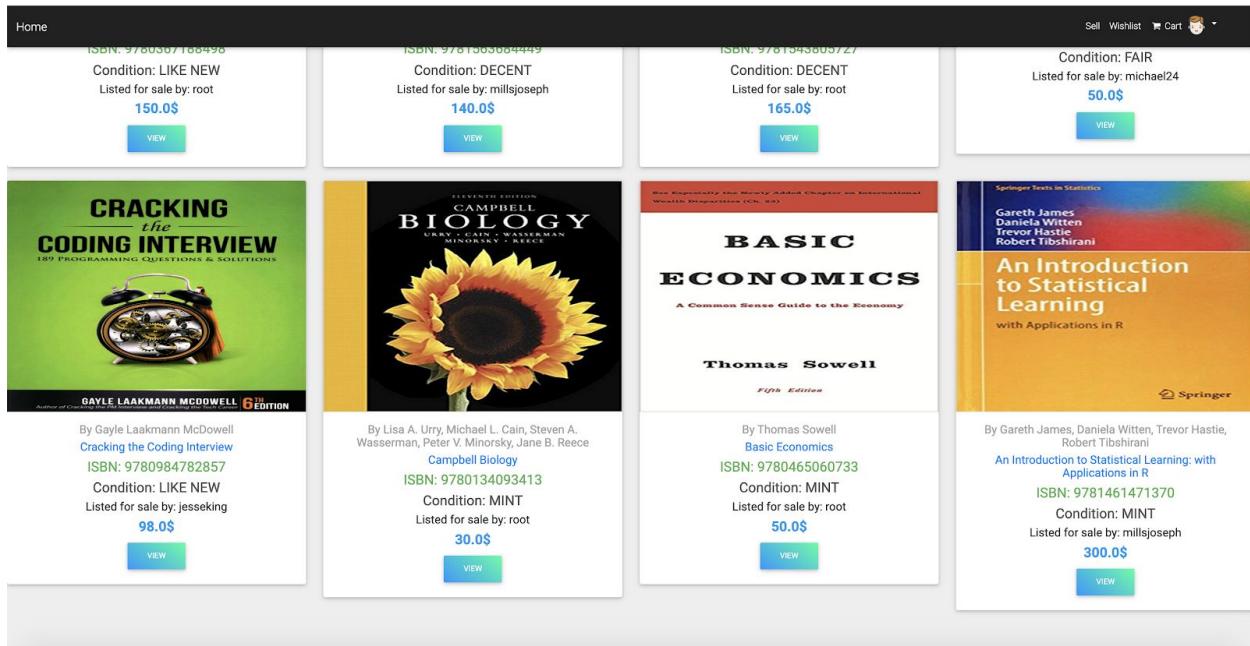
The JsonResponse(context) simply returns a message indicating whether the cancel request has gone correctly or not.

SEARCHING TEXTBOOK LISTING CATALOG

You can click on the BROWSE button on the homepage if you want to see the list of available textbook for sale (listed by other users of course). If your database is new and has no textbook listed by other accounts then you won't see anything.

The screenshot shows a website interface for a textbook catalog. At the top, there's a navigation bar with links for Home, Sell, Wishlist, Cart, and a search bar. Below the search bar is a dropdown menu set to 'Title' and a 'Sort by Title Desc' button. A horizontal menu bar contains buttons for various subjects: ARCHITECTURE, COMMUNICATION, COMPUTER SCIENCE, ENGINEERING, FINANCE, HISTORY, HUMANITIES, LAW, LITERATURE, MATHEMATICS, MEDICINE, and PHYSICS. The main content area displays four book cards:

- Theories and Methods of Writing Center Studies: A Practical Guide**
By Jo Mackiewicz, Rebecca Day Babcock
Theories and Methods of Writing Center Studies: A Practical Guide
ISBN: 9780367188498
Condition: LIKE NEW
Listed for sale by: root
[150.00](#)
- The American Sign Language Handshape Dictionary**
Second Edition with New DVD
By Richard A. Tennant, Marianne Gluszak Brown
Illustrated by Valerie Nelson-Metlay
ISBN: 9781563684449
Condition: DECENT
Listed for sale by: millsjoseph
- Strategies & Tactics for the MBE (Multistate Bar Exam) Sixth Edition**
By Steven L. Emanuel
ISBN: 9781543805727
Condition: DECENT
Listed for sale by: root
[125.00](#)
- Modern Architecture Since 1900 / Edition 3**
By William J.R. Curtis
ISBN: 9780714833569
Condition: FAIR
Listed for sale by: michael24
50.00



Query 1

use store;

SELECT * FROM Listing A, Textbook B WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID AND A.Account_ID <> 52;

Result Grid

ID	Price	Account_id	Textbook_ID	Available	Textbook_ID	ISBN	Title	Image	Author	Publisher	Date_Published
1	300	7	1	1	1	9781461471370	An Introduction to Statistical Learning:... An-Introduction-To-S...	Gareth James, Dani...	Springer	2017-09-01	
8	165	1	8	1	8	9781543805727	Strategies & Tactics for the MBE (Exam... 518hHlR9HL_SX3...	Steven L. Emanuel	Wolters Kluwer	2016-05-22	
10	98	6	10	1	10	9780984782857	Cracking the Coding Interview	Cracking-The-Codin...	Gayle Laakmann M...	CareerCup	2015-07-01
13	50	30	13	1	13	9780714833569	Modern Architecture Since 1900 / Editi...	William J.R. Curtis	Phaidon Press	1996-07-27	
16	140	7	15	1	15	9781563684449	The American Sign Language Handsh... 51zsvHUIBpl_...	Richard A. Tennant,...	Gallaudet Univers...	2010-07-31	
19	150	1	20	1	20	9780367188498	Theories and Methods of Writing Cent...	Jo Mackiewicz, Reb...	Routledge	2019-11-13	
24	30	1	25	1	25	9780134093413	Campbell Biology	Campbell-Biology.jpg	Lisa A. Urry, Michael...	Pearson	2016-10-29
25	50	1	26	1	26	9780465060733	Basic Economics	Basic-Economics_Q...	Thomas Sowell	Basic Books	2014-12-02

Result 13

Read Only

This is the query used to return the results shown above

You can press on the green button which is the list of Category and it will update the page to show only textbooks of that category. If 2 or more category is selected, it will shows all the textbook that belongs to at least 1 of those category.

The screenshot shows a website interface for buying and selling textbooks. At the top, there's a navigation bar with links for Home, Sell, Wishlist, Cart, and a user icon. Below the navigation is a search bar with a dropdown menu set to 'Title'. A dropdown menu for sorting is open, showing options like 'Sort by Title Descending' (which is checked) and 'Sort by Title Ascending'. Other options include 'Sort by Author Descending', 'Sort by Author Ascending', 'Sort by ISBN Descending', 'Sort by ISBN Ascending', 'Sort by Price Ascending', and 'Sort by Price Descending'. The main content area displays two book listings:

- An Introduction to Statistical Learning** (with Applications in R)
By Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani
ISBN: 9781461471370
Condition: MINT
Listed for sale by: millsjoseph
300.0\$
- Cracking the Coding Interview**
By Gayle Laakmann McDowell, 6th EDITION
Cracking the Coding Interview
ISBN: 9780984782857
Condition: LIKE NEW
Listed for sale by: jesseking
98.0\$

Query 1

use store;

SELECT DISTINCT A.Textbook_ID, A.id, Title, Author, ISBN, Price, Cond

FROM Listing A, Textbook B, Category_Has_Textbook C

WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID

AND A.Account_ID <> 52 AND B.Textbook_ID = C.Textbook_ID

AND C.Category_Name IN ('Computer Science');

100% 43:6

Result Grid Filter Rows: Search Export:

Textbook_ID	id	Title	Author	ISBN	Price	Cond
1	1	An Introduction to Statistical Learning:...	Gareth James, Dani...	9781461471370	300	MINT
10	10	Cracking the Coding Interview	Gayle Laakmann M...	9780984782857	98	LIKE NEW

Result 14

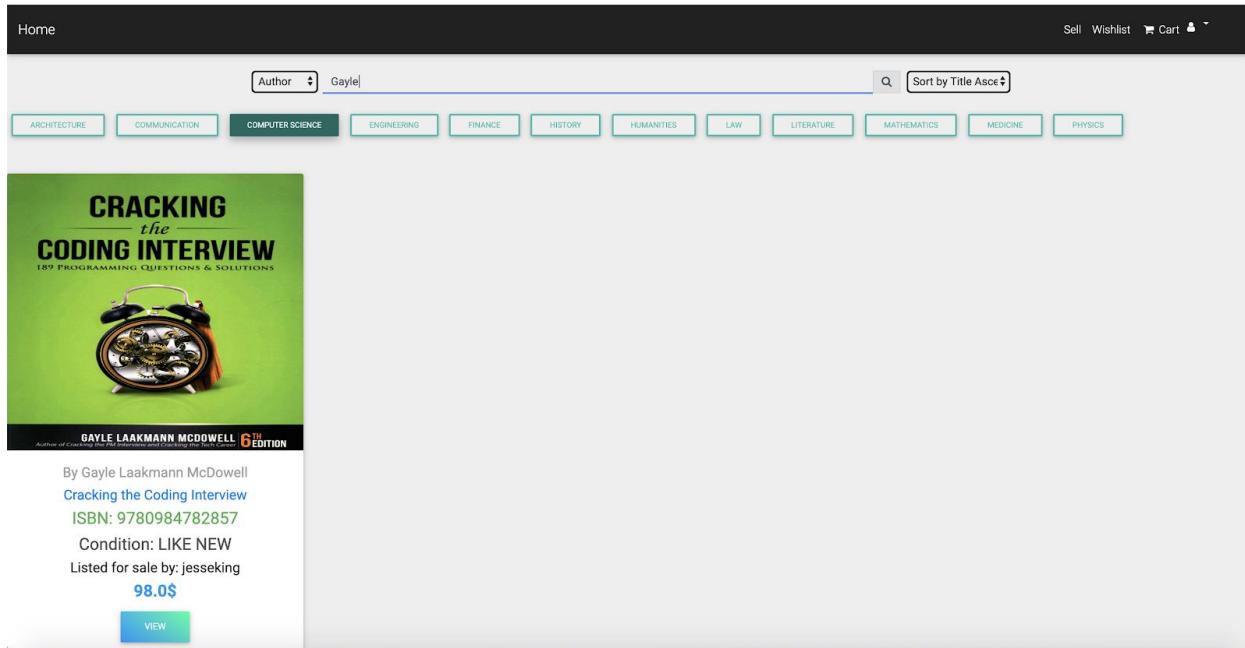
This is the query used to show the result on the last page

You can choose from a dropdown list to the right for sort by options.

The screenshot shows a web-based book search interface. At the top, there is a navigation bar with links for Home, Sell, Wishlist, Cart, and a search bar labeled "Sort by Title Asc". Below the navigation bar is a dropdown menu with options: Title (which is checked), Author, Publisher, and ISBN. A horizontal row of category buttons includes ARCHITECTURE, COMMUNICATION, ENGINEERING, FINANCE, HISTORY, HUMANITIES, LAW, LITERATURE, MATHEMATICS, MEDICINE, and PHYSICS. The main content area displays two book entries:

- An Introduction to Statistical Learning** (with Applications in R)
By Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani
ISBN: 9781461471370
Condition: MINT
Listed for sale by: millsjoseph
300.0\$
- Cracking the Coding Interview** (189 Programming Questions & Solutions)
By Gayle Laakmann McDowell, 6TH EDITION
Cracking the Coding Interview
ISBN: 9780984782857
Condition: LIKE NEW
Listed for sale by: jesseking
98.0\$
VIEW

You can also click the dropdown option to the left to choose whether you want to search by Title, Author, Publisher, or ISBN



As you can see, depending on what category pressed, what the filter is, and the text in the search bar it will return the result accordingly.

Query 1

use store;

SELECT DISTINCT A.Textbook_ID, A.id, Title, Author, ISBN, Price, Cond

FROM Listing A, Textbook B, Category_Has_Textbook C

WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID

AND A.Account_ID <> 52 AND B.Textbook_ID = C.Textbook_ID

AND C.Category_Name IN ('Computer Science') AND B.Author LIKE '%Gayle%';

Result Grid Filter Rows: Search Export:

Textbook_ID	id	Title	Author	ISBN	Price	Cond
10	10	Cracking the Coding Interview	Gayle Laakmann M...	9780984782857	98	LIKE NEW

Result 15

This is the query used to show the result on the last page

BACKEND LOGIC:

```
603
604     @login_required
605     def search(request):
606
607         # Query all the Categories and textbook Listings in the database
608         Categories = Category.objects.raw("Select * FROM Category")
609         search_query = """
610             Select *
611             FROM Listing A, Textbook B
612             WHERE A.Textbook_ID = B.Textbook_ID AND A.Available = 1
613             AND A.Account_ID <> """ + str(request.user.id) + """ ORDER BY Title DESC;
614             """
615
616         Listings = Listing.objects.raw(search_query)
617
618         return render(request, 'search.html', context={'Categories': Categories, 'listings': Listings});
```

When you first click on browse, this is the function that handles the search page view for displaying every single textbook listings

Categories = Category.objects.raw("Select * FROM Category")

Straightforward since I simply select all rows from Category table

```
search_query =  
"""  
    Select *  
    FROM Listing A, Textbook B  
    WHERE A.Textbook_ID = B.Textbook_ID AND A.Available = 1  
    AND A.Account_ID <> """ + str(request.user.id) + """ ORDER BY Title  
    DESC;  
"""
```

I join Listing table and Textbook table where Listing.Textbook_ID = Textbook.Textbook_ID and Listing.Available = 1 (available for sale) And A.Account_ID <> request.user.id which get the listings not listed by the logged in user (you), and by default it is order by Title desc This function then return a render of search.html and it passed in the two query results for rendering

```

618     @login_required
619     @csrf_exempt
620     def search_request(request):
621
622         results = None;
623         if request.is_ajax() == True:
624             Categories = json.loads(request.GET.get('Categories'))
625             Sort_ID = request.GET.get('Sort')
626             text = request.GET.get('search-text')
627             filter = request.GET.get('select-filter')
628             sort = sort_order(Sort_ID)
629
630
631         if len(Categories) == 0:
632             # If no Category option is selected and search-bar text is empty, then simply returns a
633             # search query containing every textbook listing ordered by selected sorting order
634             # Else, search for all textbook listings where the filter such as ISBN, Title, Author matches the pattern
635             # in the search bar
636             if text == '':
637                 results = Listing.objects.raw("SELECT * FROM Listing A, Textbook B WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID AND A.Account_ID <> \"\"
638                 + str(request.user.id) + " " + sort + ";")
639             else:
640
641                 results = Listing.objects.raw("SELECT * FROM Listing A, Textbook B WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID AND A.Account_ID <> \"\"
642                 + str(request.user.id) + " AND " + filter + " LIKE '%" + text + "%' " + sort + ";")
643
644         else:
645             # If a Category is selected, do the same as above but this time adds condition to ensure that
646             # the textbook belongs to at least 1 of the category selected
647             query = """
648                 SELECT DISTINCT A.Textbook_ID, A.id, Title, Author, ISBN, Price, Cond
649                 FROM Listing A, Textbook B, Category_Has_Textbook C
650                 WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID
651                 AND A.Account_ID <> "" + str(request.user.id) + "" AND B.Textbook_ID = C.Textbook_ID
652                 AND C.Category_Name IN (
653                     ...
654
655                 for category in Categories:
656                     query += "" + Categories[category] + ", "
657                 if text == '':
658                     query = query[:-2] + " ) " + sort + ";"
659                     results = Listing.objects.raw(query)
660                 else:
661                     query = query[:-2] + " ) AND " + filter + " LIKE '%" + text + "%' " + sort + ";"
662                     results = Listing.objects.raw(query)
663
664         return render(request, 'search_results.html', context={'listings': results})

```

This is the function that handles when the user for example changes the sort options, select category buttons to display only textbooks in those categories, and when the user types in the search bar.

Categories is list of categories selected by user, Sort_ID is the sort by option, text is the text in the search bar, filter is whether you search by title, author, isbn etc

FIRST CASE:

If no category option is selected and the text in the search bar is empty, then my search query is

```
-----  
SELECT * FROM Listing A, Textbook B WHERE A.Available = 1 AND A.Textbook_ID =  
B.Textbook_ID AND A.Account_ID <> request.user.id + sort  
-----
```

Join the table Listing and Textbook

WHERE Listing.Available = 1 (available for sale)

AND Listing.Textbook_ID = Textbook.Textbook_ID

AND Listing.Account_id != request.user.id (person listed isn't the logged in account) and the +
sort represents the sorting option (Order by title asc, author desc etc).

SECOND CASE:

If there is no category option selected but there is text in the search bar however then my search query become

```
-----  
SELECT * FROM Listing A, Textbook B WHERE A.Available = 1 AND A.Textbook_ID =  
B.Textbook_ID AND A.Account_ID <> request.user.id AND filter LIKE '%text%' + sort  
-----
```

The only difference is 1 more condition filter LIKE ‘%text%’ where the value of filter is (Author, ISBN, Title, Publisher) and text is the text in the search bar. Basically returns all rows where the filtered column has text like what is contained in the search bar.

LAST CASE:

If category options ARE selected, then this section of code runs

```
query =  
"""  
    SELECT DISTINCT A.Textbook_ID, A.id, Title, Author, ISBN, Price, Cond  
    FROM Listing A, Textbook B, Category_Has_Textbook C  
    WHERE A.Available = 1 AND A.Textbook_ID = B.Textbook_ID  
    AND A.Account_ID <> """ + str(request.user.id) + """ AND B.Textbook_ID  
    = C.Textbook_ID  
    AND C.Category_Name IN (  
  
"""  
for category in Categories:  
    query += "'" + Categories[category] + "', "  
    if text == '':  
        query = query[:-2] + ") " + sort + ";"  
        results = Listing.objects.raw(query)  
    else:  
        query = query[:-2] + ") AND " + filter + " LIKE '%" + text + "%' " + sort +  
";"
```

```
results = Listing.objects.raw(query)
```

I join table Listing, Textbook, and Category_Has_Textbook where Listing.Available = 1, Listing.Textbook_ID = Textbook.Textbook_ID, A.Account_ID <> request.user.id (logged in account), Textbook.Textbook_ID = Category_Has_Textbook.Textbook_ID AND C.Category_Name IN ()�.

Notice the C.Category_Name IN (). I am doing string manipulation to adds the list of category to the IN operator which will returns to me any textbooks that belong to at least 1 of the category selected.

Also, I am selecting by DISTINCT Textbook_ID since 1 textbook can belong to multiple Category, it will have multiple Category_Has_Textbook rows which joined together will give me multiple rows with the same Textbook_ID.

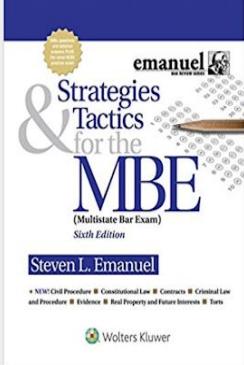
If the text is empty, then perform the query with the sort option added to the string.

If the text is NOT empty, add the filter LIKE '%text%' similar to the 2nd case and then the sort option and then perform the query

Then finally this function render search_results.html and passes the query results into the render function.

VIEW PRODUCT PAGE:

Home Sell Wishlist Cart



Strategies & Tactics for the MBE (Emanuel Bar Review) 6th Edition

ISBN: 9781543805727

By Steven L. Emmanuel

Publisher: Wolters Kluwer

Date Published: May 22, 2016

Condition: DECENT

Listed for sale by user root

Price: 165.0\$

ADD TO CART **W**

Description:

The Seventh Edition of Strategies & Tactics for the MBE has been carefully revised by Steve Emanuel and is full of up-to-date advice on how to analyze Multistate Bar Exam (MBE) questions in all MBE subject areas (Civil Procedure, Constitutional Law, Contracts, Criminal Law and Procedure, Evidence, Real Property with Future Interests, and Torts). Steve Emanuel—author of the Emanuel Law Outlines and CrunchTime books in the MBE subject areas—has passed the bar exam in several states (including New York and California) and worked with law students to prepare them for taking the MBE.

You can click on the view button of any textbook listings not your own to be shown a product page displaying all the information about the listing. If this listing is already in the user's shopping cart, then the Add to Cart button will be changed to "ALREADY IN SHOPPING CART" so that the user can't add it to their shopping cart again. Likewise, if the listing is already in the user's wish list, the icon next to the Add to Cart button will be displayed in bold.

Query 1

Execute the selected portion of the script or everything, if there is no selection

use store;

```
2 •  SELECT A.Account_id, B.Textbook_id, Price, Available, Title, ISBN
3   FROM Listing A, Textbook B, Account C
4   WHERE A.Available = 1 AND A.Account_id = C.id AND A.Textbook_ID = B.Textbook_ID
5     AND A.Textbook_ID = 8;
6
7
8
9
10
11
```

100% 66:2

Result Grid Filter Rows: Search Export:

Account_id	Textbook_id	Price	Available	Title	ISBN
1	8	165	1	Strategies & Tactics for the MBE (Ema...	9781543805727

This is the result of view product page. I purposely choose a few specific columns instead of all

* like in the actual query so that the image would fit.

```

188 @login_required
189 def view_product_page(request, textbook_id):
190     query = """
191     SELECT *
192     FROM Listing A, Textbook B, Account C
193     WHERE A.Available = 1 AND A.Account_id = C.id AND A.Textbook_ID = B.Textbook_ID AND A.Textbook_ID = %s
194     """
195     listing = Listing.objects.raw(query, [textbook_id])[0];
196
197     Cart = Shopping_Cart.objects.raw("SELECT * FROM Shopping_Cart WHERE Account_ID = %s AND Textbook_ID = %s", [str(request.user.id), textbook_id])
198     Wish_list = Wishlist.objects.raw("SELECT * FROM Wishlist WHERE Account_ID = %s AND Textbook_ID = %s", [str(request.user.id), textbook_id])
199     Cart = Cart[:]
200     Wish_list = Wish_list[:]
201     Listing_In_Cart = False
202     Listing_In_Wishlist = False
203     if len(Cart) == 1:
204         Listing_In_Cart = True
205     if len(Wish_list) == 1:
206         Listing_In_Wishlist = True
207     # If the listing is listed by the user, then return a view that lets the user edit the listing information instead
208     if listing.username == request.user.username:
209         Category_Query = """
210         SELECT id, A.Category_Name
211         FROM Category_Has_Textbook A, Category B
212         WHERE B.Category_Name = A.Category_Name AND A.Textbook_ID = %s
213         """
214     Categories = Category.objects.raw(Category_Query, [textbook_id])
215
216     # This part retrieves the information from the listing query at the beginning and formats it into a dictionary
217     # the dictionary is then used to initialize a TextbookChangeForm to create an edit listing information form
218     # prefilled with the textbook listing information
219     data = {'ISBN': listing.ISBN, 'Title': listing.Title, 'Author': listing.Author, 'Publisher': listing.Publisher, 'Cond': listing.Cond,
220             'Date_Published': listing.Date_Published, 'Price': listing.Price, 'Description': listing.Description, 'Image': listing.Image,
221             'Category': Categories}
222     form = forms.TextbookChangeForm(initial=data)
223     return render(request, 'My_Product_Page.html', context={'form': form, 'listing': listing})
224
225     return render(request, 'Product_Page.html', context={'listing': listing, 'Cart': Listing_In_Cart, 'Wishlist': Listing_In_Wishlist})
226

```

The view_product_page function takes in an argument textbook_id used to identify which textbook we want to view.

Viewing product page of a listing that is not your own is done in line 190-206 and the return on line 225

First query is

```

query =
"""

SELECT *

FROM Listing A, Textbook B, Account C

WHERE A.Available = 1 AND A.Account_id = C.id AND A.Textbook_ID =
B.Textbook_ID

AND A.Textbook_ID = %s

```

```
"""
```

```
listing = Listing.objects.raw(query, [textbook_id])[0];
```

Join Listing, Textbook, Account table based on Listing.Available = 1 (available for sale), where Account_id of Listing & Account is the same, and where Textbook_id of Listing & Textbook is the same. This returns a relation of a single row since Textbook_ID is a primary key and is thus unique.

Then,

```
Cart = Shopping_Cart.objects.raw("SELECT * FROM Shopping_Cart  
WHERE Account_ID = %s AND Textbook_ID = %s", [str(request.user.id), textbook_id])
```

```
Wish_list = Wishlist.objects.raw("SELECT * FROM Wishlist  
WHERE Account_ID = %s AND Textbook_ID = %s", [str(request.user.id), textbook_id])
```

These two query find ALL rows of Wishlist and Shopping Cart where Account_ID is equal to logged in user's account id and Textbook_ID equals to the textbook we want to view. This returns a relation of maximum 1 row since Account_id and Textbook_ID is unique for each row of Wishlist and Shopping Cart.

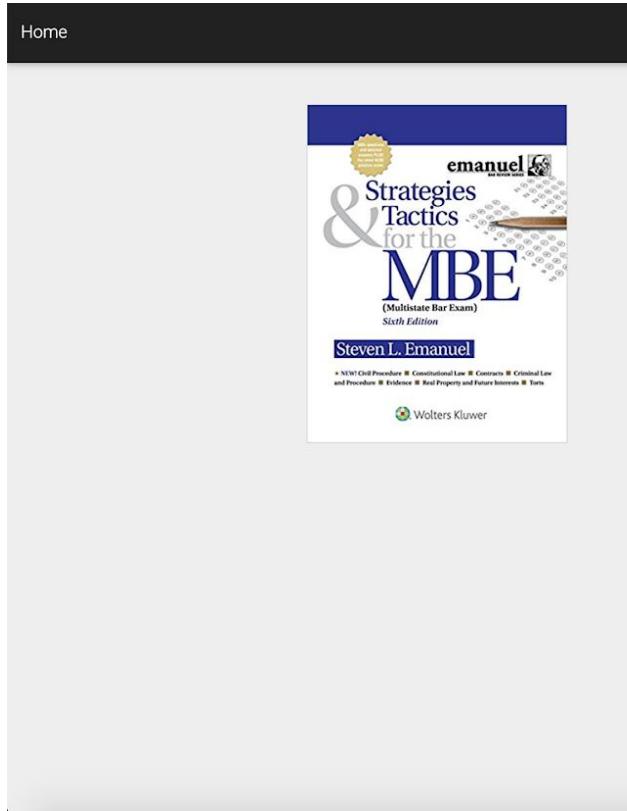
```
if len(Cart) == 1:  
    Listing_In_Cart = True  
  
if len(Wish_list) == 1:  
    Listing_In_Wishlist = True
```

This checks if the returned number of row is 1 then we know that the textbook is in the user's shopping cart or query

Then the function return a render of Product_Page.html passing in the results of listing query and two boolean variables to see whether the textbook is in the user's shopping cart and wish list.

ADD TO WISHLIST

For this feature, I will add one row to the Wishlist table.



The screenshot shows a product listing for the book "Strategies & Tactics for the MBE" by Steven L. Emanuel. The book cover is displayed on the left, featuring the title, author's name, and the publisher, Wolters Kluwer. To the right of the book image, there is detailed product information:

- Strategies & Tactics for the MBE (Emanuel Bar Review) 6th Edition**
- ISBN: 9781543805727**
- By Steven L. Emanuel**
- Publisher: Wolters Kluwer**
- Date Published: May 22, 2016**
- Condition: DECENT**
- Listed for sale by user root**
- Price: 165.0\$**

Below the price, there is a blue button labeled "ADD TO CART" with a shopping cart icon next to it. To the right of the button is a small icon of a person with a plus sign, which is highlighted in bold, indicating that the item has been added to the wishlist.

After clicking on the little icon next to the Add to Cart button, it will be displayed in bold which indicates that the item has been added to your wishlist

The screenshot shows the MySQL Workbench interface. The top section is a query editor titled "Query 1" containing the following SQL code:

```
1 •  use store;
2 •  SELECT * FROM Wishlist;
3
```

Below the query editor is a status bar showing "100%" and "24:2". The main area is a "Result Grid" displaying the results of the query. The grid has three columns: "id", "Account_id", and "Textbook_ID". A single row is shown with values: id=25, Account_id=52, and Textbook_ID=8. All other cells in the row are marked as "NULL".

A simple SELECT * From Wishlist query shows that a new row has been added to Wishlist table representing that account_id 52 has textbook_id 8 in its Wishlist.

The screenshot shows the MySQL Workbench interface. The top part is a query editor titled "Query 1" containing the following SQL code:

```

1 • use store;
2 • SELECT A.Account_id, D.Username, A.Textbook_ID, B.Price, B.Available, ISBN, Title,
3 Author, Publisher, Date_Published, Cond, `Description`
4 FROM Wishlist A, Listing B, Textbook C, `Account` D
5 WHERE B.Available = 1 AND A.Account_ID = 52 AND A.Textbook_ID =
6 B.Textbook_ID AND A.Textbook_ID = C.Textbook_ID AND D.ID = A.Account_ID;
7

```

The bottom part is a "Result Grid" showing the results of the query. The table has the following columns: Account_id, Username, Textbook_ID, Price, Available, ISBN, Title, Author, Publisher, Date_Published, Cond, and Description. There is one row returned:

Account_id	Username	Textbook_ID	Price	Available	ISBN	Title	Author	Publisher	Date_Published	Cond	Description
52	test	8	165	1	9781543805727	Strategies & Tactics for the MBE (Ema...	Steven L. Emmanuel	Wolters Kluwer	2016-05-22	DECENT	The Seventh Edition of...

A more complicated query clearly shows the information of textbook with id 8 and account with id 52

```

467
468 @login_required
469 @csrf_exempt
470 @require_POST
471 def add_to_wishlist(request):
472     if request.is_ajax() == True:
473         textbook_id = request.POST.get('textbook_id')
474
475     # Insert a new row into Wishlist
476     with connection.cursor() as cursor:
477         cursor.execute("INSERT INTO Wishlist(Textbook_ID, Account_ID) VALUES(%s, %s)", [textbook_id, str(request.user.id)])
478     connection.close()
479     return JsonResponse({})
480

```

```

cursor.execute("INSERT INTO Wishlist(Textbook_ID, Account_ID) VALUES(%s, %s)",
[textbook_id, str(request.user.id)])

```

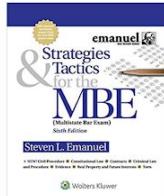
To add a new textbook to your wishlist, we only need to insert a new row into the Wishlist table containing the Account_ID and the Textbook_ID. I return an empty JsonResponse because there wasn't anything significant to return.

VIEW WISHLIST:

Click on the Wishlist option on the upper right corner to view the content of your wishlist

Home Sell Wishlist Cart

Wishlist

Item	Price	Total
 Strategies & Tactics for the MBE (Emanuel Bar Review) 6th Edition ISBN: 9781543805727 Condition: DECENT Description: The Seventh Edition of Strategies & Tactics for the MBE has been carefully revised by Steve Emanuel and is full of up-to-date advice on how to analyze Multistate Bar Exam (MBE) questions in all MBE subject areas (Civil Procedure, Constitutional Law, Contracts, Criminal Law and Procedure, Evidence, Real Property with Future Interests, and Torts). Steve Emanuel—author of the Emanuel Law Outlines and CrunchTime books in the MBE-subject areas—has passed the bar exam in several states (including New York and California) and worked with law students to prepare them for taking the MBE.	\$165.0	\$165.0 

⚡ Query 1

Limit to 1000 rows

```
1 • Execute the selected portion of the script or
2 • everything, if there is no selection
3 FROM Wishlist A, Listing B, Textbook C
4 WHERE B.Available = 1 AND A.Account_ID = 52 AND A.Textbook_ID =
5 B.Textbook_ID AND A.Textbook_ID = C.Textbook_ID;
6
7
8
9
10
```

100% 66:2

Result Grid Filter Rows: Search Export:

Account_id	Textbook_ID	Price	Available	Title	ISBN
52	8	165	1	Strategies & Tactics for the MBE (Ema...	9781543805727

This is the result of view wishlist query which matches up with what is shown in the GUI.

```

456
457 @login_required
458 def view_wishlist(request):
459     wishlist_query = """
460         SELECT *
461         FROM Wishlist A, Listing B, Textbook C
462         WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID = B.Textbook_ID
463         AND A.Textbook_ID = C.Textbook_ID
464         """
465     Wish_list= Wishlist.objects.raw(wishlist_query, [str(request.user.id)])
466     return render(request, "Wishlist.html",context={'Wishlist': Wish_list})
...

```

```

wishlist_query =
"""

SELECT *
FROM Wishlist A, Listing B, Textbook C
WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID =
B.Textbook_ID AND A.Textbook_ID = C.Textbook_ID
"""

Wish_list= Wishlist.objects.raw(wishlist_query, [str(request.user.id)])

```

Join table Wishlist, Listing, Textbook where Listing.Available = 1 (available for sale), Account_ID is equals to logged in user's account id, Textbook_ID of Wishlist & Listing & Textbook is the same. This query return a result of all textbooks in the user's wishlist. Then it returns a render of Wishlist.html and passing in the results of the Wishlist query

REMOVE FROM WISHLIST

For this feature, I will delete one row from the Wishlist table.

For each textbook listing in the Wishlist table, there is an X icon next to the Total Price column.

Clicking on it will removes the textbook from the Wishlist. So click on it, and it will displays an empty table since we removed our only wishlist item.

The screenshot shows a dark-themed web interface. At the top, there is a navigation bar with links for 'Home', 'Sell', 'Wishlist', 'Cart' (with a user icon), and a dropdown menu. Below the navigation bar, the word 'Wishlist' is displayed. A table header is visible, containing columns for 'Item', 'Price', and 'Total'. The main content area below the table is currently empty, indicating no items are present in the wishlist.

Item	Price	Total
------	-------	-------

The screenshot shows the MySQL Workbench interface. The top bar has tabs for 'Query 1' and 'SQL'. Below the tabs are various icons for database management. A toolbar includes a 'Limit to 1000 rows' button. The main area contains a query editor with the following code:

```
1 • use store;
2 • SELECT * FROM Wishlist;
3
```

Below the editor is a status bar showing '100%' and '24:2'. The central part of the interface is a 'Result Grid' showing the results of the query. The grid has columns labeled 'id', 'Account_id', and 'Textbook_ID'. One row is visible with values: id=25, Account_id=52, and Textbook_ID=8. All other cells in the row are marked as 'NULL'. The bottom of the window shows a tab labeled 'Wishlist 6' and buttons for 'Apply' and 'Revert'.

On the right side, there is a vertical toolbar with icons for different features:

- Result Grid
- Form Editor
- Field Types
- Query Stats
- Execution Plan

This is me querying all rows from Wishlist after adding an item to my wishlist earlier.

The screenshot shows the MySQL Workbench interface. The top bar has a title 'Query 1' and various toolbar icons. A dropdown menu 'Limit to 1000 rows' is open. Below the toolbar, a code editor displays the following SQL query:

```
1 •  use store;  
2 •  SELECT * FROM Wishlist;  
3
```

The bottom part of the interface shows a 'Result Grid' with the following data:

id	Account_id	Textbook_ID
NULL	NULL	NULL

This is me querying all rows from Wishlist again after clicking on the X icon to remove the item.

Now I returned 0 row since I deleted my only Wishlist entry in the database.

```

481     @login_required
482     @csrf_exempt
483     @require_POST
484     def remove_from_wishlist(request):
485         if request.is_ajax() == True:
486             textbook_id = request.POST.get('textbook_id')
487
488             # DELETE a row from Wishlist
489             with connection.cursor() as cursor:
490                 cursor.execute("DELETE FROM Wishlist WHERE Account_ID = %s and Textbook_ID = %s", [str(request.user.id), textbook_id])
491             connection.close()
492
493             # This part below requery the rest of the items in the user's wishlist
494             wish_list_query = """
495             SELECT *
496             FROM Wishlist A, Listing B, Textbook C
497             WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID = B.Textbook_ID
498             AND A.Textbook_ID = C.Textbook_ID
499             """
500
501             Wish_list = Wishlist.objects.raw(wish_list_query, [str(request.user.id)])
502
503             # If the page doesn't need to be refreshed, return False (this is when we remove from wishlist in view product page)
504             refresh = request.POST.get('refresh')
505             if refresh == 'False':
506                 return JsonResponse({})
507             else:
508                 # If we remove from wishlist when we are in the wishlist page, then we can render Wishlist_items.html to refresh
509                 # the page and shows the updated Wishlist
510                 return render(request, "Wishlist_Items.html", context={'Wishlist': Wish_list})
511

```

```

cursor.execute("DELETE FROM Wishlist WHERE Account_ID = %s and Textbook_ID = %s",
               [str(request.user.id), textbook_id])

```

To remove an item from wishlist, we delete the corresponding row in Wishlist table where account_id = your account id and the Textbook_id of the textbook we want to remove from wishlist.

```

wish_list_query =
"""
SELECT *
FROM Wishlist A, Listing B, Textbook C
WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID =
B.Textbook_ID AND A.Textbook_ID = C.Textbook_ID
"""

```

This part requery the user's remaining items in Wishlist after deleting. It join Listing and Textbook simply to get all of the information about the textbook listing. This query is only done so the results can be passed into the render function that render Wishlist_Items.html below so that the Wishlist page can be refreshed with the correct information after deleting a textbook.

ADD TO SHOPPING CART

For this feature, I will add one row to the Shopping Cart table.

Now go back to any textbook listing's product page, and click on the "Add To Shopping Cart" button to add the item to your shopping cart. After clicking on the button, it will display "ADDED To Shopping Cart" like below

The screenshot shows a product listing for 'Campbell Biology'. The left side features the book cover, which is black with a large yellow sunflower in the center. The title 'CAMPBELL BIOLOGY' is printed above the flower. The right side contains product details: 'Campbell Biology', ISBN: 9780134093413, by Lisa A. Urry, Michael L. Cain, Steven A. Wasserman, Peter V. Minorsky, and Jane B. Reece, published by Pearson on Oct. 29, 2016, in MINT condition, listed for sale by user 'root' at \$30.05. A blue button labeled 'ADDED TO SHOPPING CART' is visible. Below the button is a small icon of a person with a plus sign. The page also includes a 'Description' section with a detailed summary of the book's content and features.

The screenshot shows the MySQL Workbench interface. The top bar has a tab labeled "Query 1". Below the tab are various icons for database management. The main area contains a query editor with the following code:

```
1 •  use store;
2 •  SELECT * FROM Shopping_Cart;
```

Below the query editor is a status bar showing "100%" and "29:2". The bottom half of the window is a "Result Grid" displaying the results of the query:

id	Account_id	Textbook_ID
50	1	13
52	52	25
HULL	HULL	HULL

A simple query to select all rows from Shopping_Cart, the 2nd row in Shopping_Cart represents that I just added textbook_id 25 to account_id 52's shopping cart.

The screenshot shows the MySQL Workbench interface with a query editor titled "Query 1". The query is:

```

1 • use store;
2 • SELECT A.id, A.Account_ID, D.username, A.Textbook_ID, B.Available, B.Price,
3   C.ISBN, C.Title, C.Author, C.Cond, `Description`
4   FROM Shopping_Cart A, Listing B, Textbook C, `Account` D
5   WHERE B.Available = 1 AND A.Account_ID = 52 AND A.Textbook_ID = B.Textbook_ID
6   AND A.Textbook_ID = C.Textbook_ID AND D.ID = A.Account_ID;

```

The results grid displays one row of data:

id	Account_ID	username	Textbook_ID	Available	Price	ISBN	Title	Author	Cond	Description
52	52	test	25	1	30	9780134093413	Campbell Biology	Lisa A. Urry, Michael... MINT	The Eleventh Edition of the	

A more complicated query clearly shows the information of account id 52 and textbook id 25.

BACKEND LOGIC

```

445
446 @login_required
447 @csrf_exempt
448 @require_POST
449 def add_to_cart(request):
450     if request.is_ajax() == True:
451         textbook_id = request.POST.get('textbook_id')
452         with connection.cursor() as cursor:
453             # Insert the item into the user's shopping cart
454             cursor.execute("INSERT INTO Shopping_Cart(Textbook_ID, Account_ID) VALUES(%s, %s)", [textbook_id, str(request.user.id)])
455             connection.close()
456     return JsonResponse({})
457

```

cursor.execute("INSERT INTO Shopping_Cart(Textbook_ID, Account_ID) VALUES(%s, %s)",
[textbook_id, str(request.user.id)])

To add a new item to the user's shopping cart, just insert a new row in Shopping Cart containing the Account_ID of the user and the Textbook_ID of the textbook. Return an empty JsonResponse because there is nothing significant to return.

VIEW SHOPPING CART

You can now click on the Cart button on the upper right corner to view the items in your shopping cart. After clicking it, it will return to you a view like below:

The screenshot shows a shopping cart interface. At the top, there are navigation links: Home, Sell, Wishlist, Cart, and a user icon. Below the header, the page title is "Shopping Cart". The main content area displays a single item in a table format:

Item	Price	Total
Campbell Biology ISBN: 9780134093413 Condition: MINT Description: The Eleventh Edition of the best-selling Campbell BIOLOGY sets students on the path to success in biology through its clear and engaging narrative, superior skills instruction, innovative use of art and photos, and fully integrated media resources to enhance teaching and learning. To engage learners in developing a deeper understanding of biology, the Eleventh Edition challenges them to apply their knowledge and skills to a variety of new hands-on activities and exercises in the text and online. Content updates throughout the text reflect rapidly evolving research, and new learning tools include Problem-Solving Exercises, Visualizing Figures, Visual Skills Questions, and more.	\$30.0	\$30.0

At the bottom of the cart interface, there is a "Total: \$30.0" summary and a prominent blue "CHECKOUT" button.

The screenshot shows a database query editor window titled "Query 1". The query is as follows:

```
1 • use store;
2 • SELECT A.Account_ID, C.Textbook_ID, Price, Available, Title, ISBN
3   FROM Shopping_Cart A, Listing B, Textbook C
4   WHERE B.Available = 1 AND A.Account_ID = 52 AND A.Textbook_ID =
5     B.Textbook_ID AND A.Textbook_ID = C.Textbook_ID;
```

Below the query, the results are displayed in a "Result Grid" table:

Account_ID	Textbook_ID	Price	Available	Title	ISBN
52	25	30	1	Campbell Biology	9780134093413

This is the result of shopping cart query which matches up with what the GUI shown above.

```
401 @login_required
402 def view_shopping_cart(request):
403     # Select all the items in the user's shopping cart
404     shopping_cart_query = """
405         SELECT *
406         FROM Shopping_Cart A, Listing B, Textbook C
407         WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID = B.Textbook_ID
408         AND A.Textbook_ID = C.Textbook_ID
409     """
410     Cart = Shopping_Cart.objects.raw(shopping_cart_query, [str(request.user.id)])
411
412     # Iterate through the rows and get the total price of all the shopping cart items combined
413     total_price = 0;
414     for item in Cart:
415         total_price += item.Price
416     return render(request, "Shopping_Cart.html", context={'Cart': Cart, 'total_price': total_price})
417
shopping_cart_query =
"""
SELECT *
FROM Shopping_Cart A, Listing B, Textbook C
WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID =
B.Textbook_ID AND A.Textbook_ID = C.Textbook_ID
"""
Cart = Shopping_Cart.objects.raw(shopping_cart_query,
[str(request.user.id)])
```

The purpose of this query is to select all of the logged in user's items in their shopping cart. I join the Shopping Cart table with the Listing and Textbook table simply to get all of the textbook listing information. They are joined on the condition that Listing.Available = 1 (available for sale), Textbook_ID of Listing, Shopping Cart & Textbook are the same, and that the Account_ID is equal to the logged in user's account id.

The for loop below is iterating through the rows to sum up the price of all the items combined.

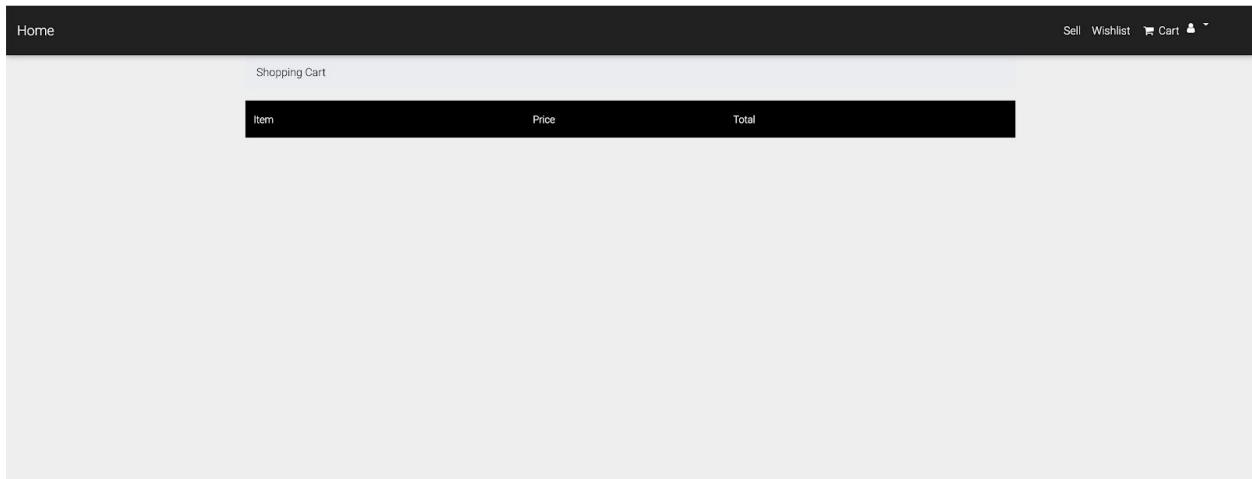
The query result and the total price is then passed into the render function for rendering

Shopping_Cart.html

REMOVE FROM SHOPPING CART

For this feature, I will remove one row from Shopping Cart Table.

Similarly to how you remove an item from Wishlist, each textbook item in your shopping cart has an X icon next to the Total Price column which you can click on to remove the item from your Shopping Cart. Clicks on it and since I only had 1 item in my shopping cart, it will display an empty table like below:



A screenshot of a web application interface showing an empty shopping cart. The top navigation bar includes links for Home, Sell, Wishlist, and Cart. The Cart link is highlighted with a dark background and white text. Below the navigation is a header labeled "Shopping Cart". A table is displayed with three columns: "Item", "Price", and "Total". There are no rows of data in the table.

Item	Price	Total

The screenshot shows the MySQL Workbench interface. The top bar has a tab labeled "Query 1". Below the bar are various icons for database management. The main area contains two lines of SQL code:

```
1 • use store;
2 • SELECT * FROM Shopping_Cart;
```

Below the code is a progress bar indicating "100%" completion and a timestamp "29:2". The bottom section is titled "Result Grid" and displays a table with three columns: "id", "Account_id", and "Textbook_ID". The data in the table is:

	id	Account_id	Textbook_ID
▶	50	1	13
	52	52	25
	HULL	HULL	HULL

This is from when I query the database after adding the textbook “Campbell Biology” to my shopping cart.

The screenshot shows the MySQL Workbench interface. The top bar has a title 'Query 1' with a lightning bolt icon. Below it is a toolbar with icons for file operations (New, Open, Save, Print, Find, Copy, Paste, Undo, Redo), a limit dropdown set to '1000 rows', and various navigation and search tools. The main area contains a query editor with the following code:

```
1 • use store;  
2 • Select * from Shopping_Cart;
```

Below the editor is a status bar showing '100%' zoom and '29:2' seconds. The bottom section is a 'Result Grid' showing the results of the query:

	id	Account_id	Textbook_ID
▶	50	1	13
	NULL	NULL	NULL

There are also 'Edit' and 'Export/Import' buttons with their respective icons.

After clicking on the X to remove the item, you can see that the Shopping_Cart row with account_id 52 and textbook_id 25 has been deleted from the database.

```

417
418     @login_required
419     @csrf_exempt
420     @require_POST
421     def shopping_cart_delete(request):
422
423         if request.is_ajax() == True:
424             textbook_id = request.POST.get('textbook_id')
425
426             with connection.cursor() as cursor:
427                 # Delete the item from the user's shopping cart
428                 cursor.execute("DELETE FROM Shopping_Cart WHERE Account_ID = %s and Textbook_ID = %s", [str(request.user.id), textbook_id])
429             connection.close()
430
431             # Query for the remaining items in the user's shopping cart after deleting
432             shopping_cart_query = """
433                 SELECT *
434                 FROM Shopping_Cart A, Listing B, Textbook C
435                 WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID = B.Textbook_ID
436                 AND A.Textbook_ID = C.Textbook_ID
437             """
438
439             Cart = Shopping_Cart.objects.raw(shopping_cart_query, [str(request.user.id)])
440             # Find the total price of all the items in the user's shopping cart
441             total_price = 0;
442             for item in Cart:
443                 total_price += item.Price
444             return render(request, "Cart_Items.html", context={'Cart': Cart, 'total_price': total_price})
445

```

```

cursor.execute("DELETE FROM Shopping_Cart WHERE Account_ID = %s and Textbook_ID = %s",
               [str(request.user.id), textbook_id])

```

To delete an item from the user's shopping cart, just delete the row where account_id is equals to the user's account id, and textbook_id is equals to the textbook's id.

The shopping cart query and total price calculations on the bottom is the same as the procedure for VIEW SHOPPING CART. I just requery all the items remaining in the shopping cart and calculates the total price so that I can use them to render Cart_Items.html and update the Shopping Cart page without reloading.

View Checkout

The screenshot shows a shopping cart interface. At the top, there's a navigation bar with 'Home', 'Sell', 'Wishlist', 'Cart', and a user icon. Below that is a 'Shopping Cart' section with a header 'Item'. A single item is listed: 'Cracking the Coding Interview' by Gayle Laakmann McDowell, 6th Edition. The image of the book shows a green cover with a clock and the title. The item details include ISBN: 9780984782857, Condition: LIKE NEW, and a detailed description about the book's purpose and content. The total price is \$98.00, and there's a 'CHECKOUT' button at the bottom.

In order to check out your items, head over to the Shopping Cart page and at the bottom right of the table you will see a Checkout button. Click it and it will redirects you to a form for you to input your information and place your order.

The screenshot shows a checkout form. On the left, there are fields for 'First Name' (containing 'test'), 'Last Name' (containing 'account'), 'Address 1 (Street Address)', 'Address 2 (Apt #, Suite, Floor etc.)', 'STATE' (a dropdown menu), 'City', and 'Zip Code'. On the right, there's a summary of the order: a book titled 'Cracking the Coding Interview' by Gayle Laakmann McDowell, 6th Edition, with ISBN 9780984782857, Author: Gayle Laakmann McDowell, Publisher: CareerCup, Condition: LIKE NEW, and Price: 98.0. Below this, it shows 'SHIPPING: \$5.00' and 'ORDER TOTAL: \$103'.

This checkout form will be displayed to you where you can input information about the order on the left and on the right will be a list of textbooks in your order.

Home

Billing Address
920 Keys St. Santa Clara, CA 95050

MY SAVED CREDIT CARDS

Au Tran
***** 3654
Expire On: 01/2020

Kevin Smith
***** 4321
Expire On: 01/2020

John Doe
***** 3456
Expire On: 01/2019

ORDER SUMMARY

Cracking the Coding Interview
ISBN: 9780984782857
Author: Gayle Laakmann McDowell
Publisher: CareerCup
Condition: LIKE NEW
Price: \$98.00

SHIPPING: \$5.00
ORDER TOTAL: \$103

On the next step of the form, if you have any credit cards saved to your account then it will be displayed as a list like on the left as shown above. You can choose any of these credit cards, scroll down, click next and move on to your order summary.

The screenshot shows the MySQL Workbench interface. The query editor at the top contains the following SQL code:

```

1 • use store;
2 • SELECT *
3   FROM PaymentInfo A, Account_Has_PaymentInfo B
4   WHERE A.Card_Number = B.Card_Number AND B.Account_ID = 1;
5

```

The result grid below displays the retrieved data:

Card_Number	Card_Name	Security_Code	Expiration_Date	Billing_Address	id	Account_id	Card_Number
1092837465654321	Kevin Smith	987	01/2020	2250 Corona Drive Santa Clara, CA 95050	4	1	1092837465654321
1234567890123456	John Doe	123	01/2019	993 River St. Santa Clara, CA 95050	2	1	1234567890123456
9871234561236543	Au Tran	123	01/2020	920 Keys St. Santa Clara, CA 95050	22	1	9871234561236543

This is the query I used to retrieve the credit cards saved to the user's account and list it out for the user to select when checking out an order.

BACKEND:

```

543 @login_required
544 def view_checkout(request):
545     # Select all textbook listings in the user's shopping cart
546     shopping_cart_query = """
547         SELECT *
548         FROM Shopping_Cart A, Listing B, Textbook C
549         WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID = B.Textbook_ID
550         AND A.Textbook_ID = C.Textbook_ID
551         """
552
553     # Select all the credit card saved to the user's account
554     credit_card_query = """
555     SELECT *
556     FROM PaymentInfo A, Account_Has_PaymentInfo B
557     WHERE A.Card_Number = B.Card_Number AND B.Account_ID = %s
558     """
559     Credit_Cards = PaymentInfo.objects.raw(credit_card_query, [str(request.user.id)])
560     Cart = Shopping_Cart.objects.raw(shopping_cart_query, [str(request.user.id)])
561     return render(request, 'Checkout.html', context={'Cart': Cart, 'Credit_Cards': Credit_Cards})
562

```

```

shopping_cart_query =
"""

SELECT *

FROM Shopping_Cart A, Listing B, Textbook C

WHERE B.Available = 1 AND A.Account_ID = %s AND A.Textbook_ID = B.Textbook_ID

```

```
AND A.Textbook_ID = C.Textbook_ID
```

```
"""
```

This query joins Shopping Cart, Listing, and Textbook table together on the condition that Listing.Available = 1 (available for sale), Textbook_ID of Shopping_Cart & Listing & Textbook are the same, and the Account_ID is equals to the logged in user's account id.

The purpose is to select all textbook listings in the user's shopping cart for checkout.

```
credit_card_query =  
"""  
SELECT *  
FROM PaymentInfo A, Account_Has_PaymentInfo B  
WHERE A.Card_Number = B.Card_Number AND B.Account_ID = %s  
"""
```

This query joins table PaymentInfo And Account_Has_PaymentInfo on the condition that their Card Number are the same and the Account_ID is equal to the logged in user's account id.

This purpose is to select all the credit cards that is saved to the user's account.

The results of the 2 query is then passed into the render function to render the Checkout.html page that returns a view of the checkout form.

Checking Payment Info

This function is invoked when the user is trying to checkout an order and opts to insert a new credit card information in the database. This function is necessary because my PaymentInfo table contains key Credit Card Number and attributes Credit Card Name, Security Code, Expiration Date, and Billing Address. If the user wants to insert a new credit card using a credit card number that already exists in the database with a different security code for example, then this function will block the request from processing since that breaks the integrity of my database.

Home

City:
Santa Clara

Zip Code:
95050

NEW CREDIT CARD

VISA MasterCard DISCOVER AMEX

Name on Card:
Au Tran

Card Number:
9871234561236543

This is an invalid credit card number according to our database.

January 2020

321

PREVIOUS NEXT

As you can see above, the user is trying to insert a new credit card with Card Number 9871234561236543 and security code 321. However it is invalid since it doesn't match up with what we have in our database.

The screenshot shows the MySQL Workbench interface. The top bar has a tab labeled "Query 1". Below the bar are various icons for database management. A toolbar with icons for file operations, search, and refresh is followed by a "Limit to 1000 rows" dropdown. The main area contains two lines of SQL code:

```
1 • use store;
2 • Select * from PaymentInfo Where Card_Number = 9871234561236543;
```

Below the code is a result grid titled "Result Grid". It has columns for "Card_Number", "Card_Name", "Security_Code", "Expiration_Date", and "Billing_Address". One row is displayed, showing:

Card_Number	Card_Name	Security_Code	Expiration_Date	Billing_Address
9871234561236543	Au Tran	123	01/2020	920 Keys St. Santa Clara, CA 95050

In our database, there already exists a row in PaymentInfo table with Card_Number = 987123456123654 and the security code is 123 not 321 so it properly marks the credit card the user is trying to use as invalid.

BACKEND LOGIC:

```
512 @login_required
513 @csrf_exempt
514 def check_payment_info(request):
515     if request.is_ajax() == True:
516         Billing_Address = request.GET.get('billing_address')
517         Credit_Card_Number = request.GET.get('credit_card_number')
518         Credit_Card_Name = request.GET.get('credit_card_name')
519         Credit_Card_Security_Code = request.GET.get('credit_card_cvv')
520         Credit_Card_Expire_Date = request.GET.get('credit_card_expiration')
521
522         # Find a row in the PaymentInfo using the Card Number
523         query = "SELECT * FROM PaymentInfo WHERE Card_Number = %s"
524         result = PaymentInfo.objects.raw(query, [Credit_Card_Number])[:]
525
526         # If the length of the rows returned is 0, then it means this credit card is not contained in the
527         # database so the insertion of this new credit card is valid
528         if(len(result) == 0):
529             return JsonResponse({'message': 'Valid'})
530         else:
531             # Else, if this credit card number already exists in the database
532             # checks if the billing address, card name, security code, and expiration date matches the
533             # information in our database records. If yes, then the user can use this credit card to makes an order
534             # If not, return a message indicating that the credit card information is invalid
535             result = result[0]
536             if result.Card_Name != Credit_Card_Name or str(result.Security_Code) != Credit_Card_Security_Code \
537             or result.Expiration_Date != Credit_Card_Expire_Date or result.Billing_Address != Billing_Address:
538                 return JsonResponse({'message': 'Invalid'})
539             else:
540                 return JsonResponse({'message': 'Valid'})
```

First, it will do a query to see if the Credit Card Number already exists in the database.

```
query = "SELECT * FROM PaymentInfo WHERE Card_Number = %s"
```

This selects the row in PaymentInfo where Card Number matches the specified number

Then I check len(result) == 0 to see if the length of row returns is 0 then the credit card number is not yet in the database and return a JsonResponse message indicating the credit card information is valid.

Else, if the credit card number already exists in the database, then I check if the Card Name, Security Code, Expiration Date, and Billing Address matches what is provided by the user. If yes, then it returns a JsonResponse valid message or invalid if not.

CHECKOUT REQUEST

This feature will add a new row to Order table, add a row to Checkout table, add one or more row into Order_Contain_Textbook table. It will use update to update one or more row in Listing and set Available to 0. It also delete one or more rows from both Wishlist and Shopping Carts table. Finally, it may add one rows to both PaymentInfo and Account_Has_PaymentInfo table.

Home Sell Wishlist Cart

Shipping Address

First Name: test

Last Name: account

Address 1: 850 Meadowbrook North

Address 2: Address 2 (Apt #, Suite, Floor etc)

Hawaii

City: Honolulu

Zip Code: 96808

Shipping Method

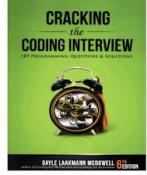
Standard - 5.00\$ - Ships in 3-5 Business Day

Express - 12.00\$ - Ships in 1-2 Business Day

Use as Billing Address

NEXT

ORDER SUMMARY



Cracking the Coding Interview
ISBN: 9780984782857
Author: Gayle Laakmann McDowell
Publisher: CareerCup
Condition: LIKE NEW
Price: 98.0

SHIPPING: \$12.00

ORDER TOTAL: \$110

Go to checkout and fill in the shipping address and click Next

Home

Sell Wishlist Cart

City:
Honolulu

Zip Code:
96808

NEW CREDIT CARD



Name on Card:

example card

Card Number:
8576123451629125

May 2025

765|

PREVIOUS NEXT

Fill in your billing address and new credit information and click Next

Home

Sell Wishlist Cart

REVIEW YOUR ORDER

PLACE ORDER

SHIPPING EDIT PAYMENT INFORMATION EDIT CREDIT CARD INFORMATION EDIT

SHIPPING ADDRESS

test account
850 Meadowbrook North
Honolulu, HI 96808

BILLING ADDRESS

test account
850 Meadowbrook North
Honolulu, HI 96808

example card
***** * 9125
Exp. 05/2025
Price: \$110

SHIPPING METHOD

EXPRESS

Item	Price	Total
Cracking the Coding Interview ISBN: 9780984782857 Condition: LIKE NEW Description: I am not a recruiter. I am a software engineer. And as such, I know what it's like to be asked to whip up brilliant algorithms on the spot and then write flawless code on a whiteboard. I've been through this as a candidate and as an interviewer. Cracking the Coding Interview, 6th Edition is here to help you through this process, teaching you what you need to know and enabling you to perform at your very best. I've coached and interviewed hundreds of software engineers. The result is this book. Learn how to uncover the hints and hidden details in a question, discover how to break down a problem into manageable chunks,	\$98.0	\$98.0

If you are happy with your order information, click the red Place Order or the top or bottom of the page to place your order.

Ch goin120

Query 1

1 • use store;
 2 • SELECT * FROM Account WHERE ID = 52;

100% 36:2

Result Grid Filter Rows: Search Edit: Export/Import:

ID	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined
52	pbkdf2_sha256\$150000\$EvE8Kyv6B1AXSFrrq...	2019-12-08 01:56:11.987550	1	test	test	account	test@yahoo.com	1	1	2019-12-06 06:25:10.175230
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

This is account with id 52, the account that I will be using to checkout this order

Query 1

1 • use store;
 2 • SELECT * FROM Textbook WHERE Textbook_ID = 10;

100% 46:2

Result Grid Filter Rows: Search Edit: Export/Import:

Textbook_ID	ISBN	Title	Image	Author	Publisher	Date_Published	Cond	Description
10	9780984782857	Cracking the Coding Interview	Cracking-The-Codin...	Gayle Laakmann M...	CareerCup	2015-07-01	LIKE NEW	I am not a recruiter. I am a...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

This is textbook id 10, the textbook that I will buy in this order

STATE OF THE DATABASE PRECHECKOUT:

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query editor contains the following SQL code:

```
1 • use store;
2 • SELECT * FROM Account_Has_PaymentInfo;
```

The results grid displays the following data:

id	Account_id	Card_Number
4	1	1092837465654321
2	1	1234567890123456
NULL	NULL	NULL

You can see here that Account_Has_PaymentInfo precheckout has no row associated with account_id 52 indicating that there is no saved credit cards for that account yet

The screenshot shows the MySQL Workbench interface with a query editor and a results grid. The query editor contains the following SQL code:

```
1 • use store;
2 • SELECT * FROM PaymentInfo;
```

The results grid displays the following data:

Card_Number	Card_Name	Security_Code	Expiration_Date	Billing_Address
1092837465654321	Kevin Smith	987	01/2020	2250 Corona Drive Santa Clara, CA 95050
1234567890123456	John Doe	123	01/2019	993 River St. Santa Clara, CA 95050
9871234561236543	Au Tran	123	01/2020	920 Keys St. Santa Clara, CA 95050
NULL	NULL	NULL	NULL	NULL

This is the list of credit cards existing in the PaymentInfo table precheckout

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following SQL code:

```

1 • use store;
2 • SELECT * FROM Listing;

```

The result grid displays the following data from the Listing table:

id	Price	Account_Id	Textbook_ID	Available
1	300	7	1	1
2	50	13	2	1
3	75	3	3	0
7	78	1	7	1
8	165	1	8	1
9	60	1	9	1
10	98	6	10	1
11	87	8	11	0
12	250	25	12	0
13	50	30	13	1
14	99	20	14	0
16	140	7	15	1
19	150	1	20	1
21	78	33	22	0
23	75	52	24	1
24	30	1	25	1
NULL	NULL	NULL	NULL	NULL

A vertical sidebar on the right contains icons for Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.

The rows in Listing table represents all of the Textbook listings. Notice that the textbook we are about to checkout (textbook_id 10) is in row 10 of the Listing table and Available is set to 1 (indicating it is available for sale).

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following SQL code:

```

1 • use store;
2 • SELECT * FROM Checkout;

```

The result grid displays the following data from the Checkout table:

id	Account_id	Order_ID
1	1	2
2	1	3
8	1	9
NULL	NULL	NULL

This is the number of checkouts existing in Checkout table. There is no row associated with account id 52 indicating that that account has not made any checkout yet

This screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following SQL code:

```

1 • use store;
2 • SELECT * FROM Orders;

```

The result grid displays the following data from the Orders table:

Order_ID	Date	Shipping_Address	Total_Price	Shipping_Method
2	2019-11-29 05:59:28.188283	2275 Ellena Drive Apt 4 Santa Clara, CA 95050	361	EXPRESS
3	2019-11-30 02:00:13.960078	2275 Ellena Drive Apt 4 Santa Clara, CA 95050	92	STANDARD
8	2019-12-03 23:59:52.589983	993 River St. Santa Clara, CA 95050	80	STANDARD
9	2019-12-05 04:40:12.367638	993 River St. Santa Clara, CA 95050	83	STANDARD
NULL	NULL	NULL	NULL	NULL

This is the number of orders existing in the Orders table before I place my order

This screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following SQL code:

```

1 • use store;
2 • SELECT * FROM Order_Contain_Textbook;

```

The result grid displays the following data from the Order_Contain_Textbook table:

id	Order_ID	Textbook_ID
8	8	3
3	3	11
1	2	12
2	2	14
9	9	22
NULL	NULL	NULL

This is the rows in Order_Contain_Textbook that shows which textbook was sold in which order.

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following SQL code:

```
1 • use store;
2 • SELECT * FROM Shopping_Cart;
```

The result grid displays the following data:

id	Account_id	Textbook_ID
54	1	10
53	52	10
NULL	NULL	NULL

This is the rows in Shopping Cart representing the textbooks in each account's shopping cart. As you can see the textbook I'm about to checkout (id 10) is in the shopping cart of two accounts.

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query editor contains the following SQL code:

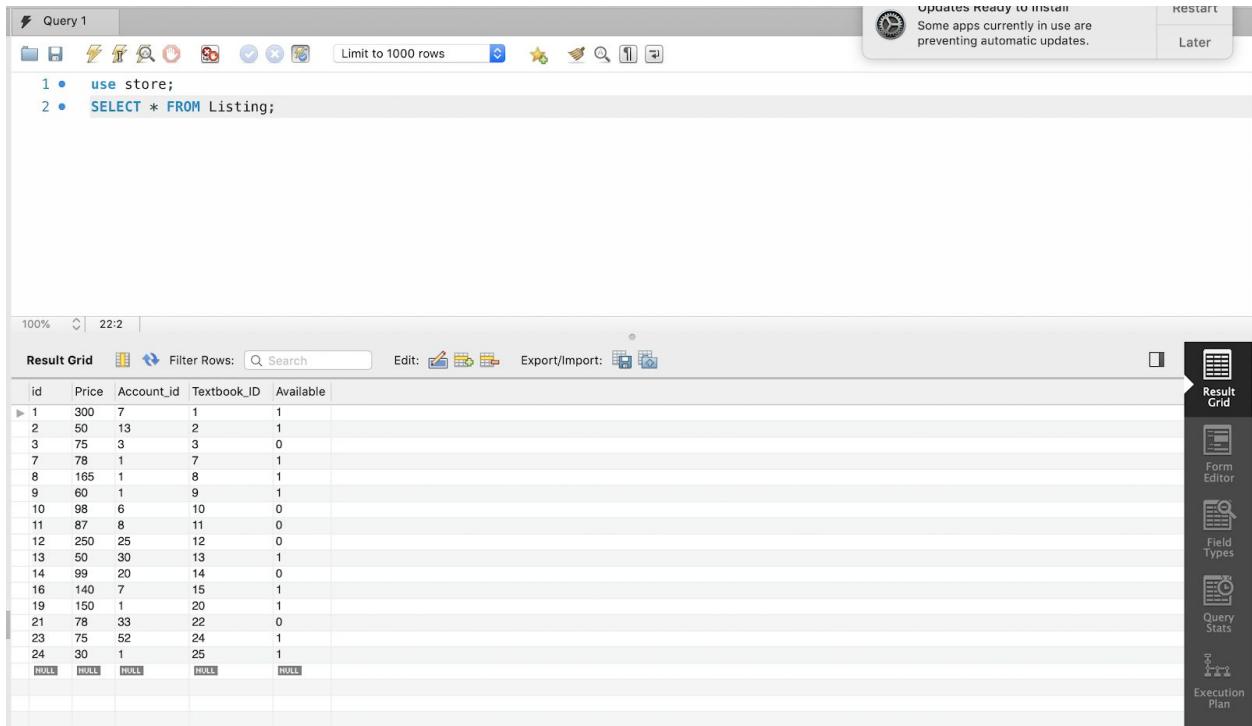
```
1 • use store;
2 • SELECT * FROM Wishlist;
```

The result grid displays the following data:

id	Account_id	Textbook_ID
26	52	10
NULL	NULL	NULL

This is the rows in Wishlist table representing the textbooks in each account's wish list. As you can see the textbook I'm about to checkout (id 10) is in the wish list of account_id 52

STATE OF THE DATABASE POSTCHECKOUT



The screenshot shows a database query interface with the following details:

- Query 1:** The current query is "SELECT * FROM Listing;".
- Updates Ready to Install:** A message indicates that some apps are currently in use, preventing automatic updates. Options to "Restart" or "Later" are available.
- Result Grid:** The results of the query are displayed in a grid format. The columns are labeled: id, Price, Account_id, Textbook_ID, and Available.
- Data:** The grid contains 24 rows of data. Row 10 has an id of 10, a Price of 98, an Account_id of 6, a Textbook_ID of 10, and an Available value of 0.
- Toolbar:** Includes standard database icons for file operations, search, and export.
- Filter Rows:** A search bar and filter options are present.
- Edit and Export/Import:** Buttons for modifying data and exporting/importing files.
- Right Panel:** A sidebar provides links to other tools: Result Grid, Form Editor, Field Types, Query Stats, and Execution Plan.

Notice in row 10 of the Listing table, after placing an order on the textbook with id 10, the row is updated with Available = 0 indicating that the textbook is sold and no longer available for sale.

The screenshot shows the MySQL Workbench interface. In the top-left corner, there's a tab labeled "Query 1". Below it is a toolbar with various icons for database management. The main area contains two numbered steps: 1. "Execute the selected portion of the script or everything, if there is no selection" and 2. "SELECT * FROM Shopping_Cart;". A tooltip for step 2 explains the command. At the bottom, there's a "Result Grid" section with columns "id", "Account_id", and "Textbook_ID". The first row shows all three columns as "NULL".

```
1 • Execute the selected portion of the script or everything, if there is no selection
2 • SELECT * FROM Shopping_Cart;
```

id	Account_id	Textbook_ID
NULL	NULL	NULL

There exists no remaining row in Shopping Cart table because after buying the textbook the textbook is then removed from every account's shopping cart so the corresponding rows are deleted from the database.

This screenshot is similar to the one above, showing the MySQL Workbench interface with a "Query 1" tab. It displays the same two-step query and the resulting "Result Grid" with all three columns as "NULL". A "Later" button is visible in the top right corner of the interface.

```
1 • use store;
2 • SELECT * FROM Wishlist;
```

id	Account_id	Textbook_ID
NULL	NULL	NULL

There exists no remaining row in Wishlist table because after buying the textbook the textbook is then removed from every account's wishlist so the corresponding rows are deleted from the database.

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons for file operations, search, and navigation. A message in the top right corner says "Updates Ready" with a note about preventing automatic updates. Below the toolbar, a query editor window titled "Query 1" contains the following SQL code:

```
1 • use store;
2 • SELECT * FROM PaymentInfo;
```

Below the query editor is a status bar showing "100% 26:2". The main area is a "Result Grid" showing the results of the SELECT query. The grid has columns: Card_Number, Card_Name, Security_Code, Expiration_Date, and Billing_Address. There are five rows of data, with the fifth row being the newly inserted row (Au Tran). The grid includes standard toolbar buttons for filtering, editing, and exporting.

Card_Number	Card_Name	Security_Code	Expiration_Date	Billing_Address
1092837465654321	Kevin Smith	987	01/2020	2250 Corona Drive Santa Clara, CA 95050
1234567890123456	John Doe	123	01/2019	993 River St. Santa Clara, CA 95050
8576123451629125	example card	765	05/2025	850 Meadowbrook North Honolulu, HI 96808
9871234561236543	Au Tran	123	01/2020	920 Keys St. Santa Clara, CA 95050
NULL	NULL	NULL	NULL	NULL

After placing the order, the new credit card information is inserted into PaymentInfo table. The row inserted is row 3.

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons for database management. Below the toolbar, a message box says: "Execute the selected portion of the script or everything, if there is no selection". The main area contains a query editor with the following content:

```
2 •  SELECT * FROM Account_Has_PaymentInfo;
```

Below the query editor is a status bar showing "100%" and "34:2". The main window title is "Result Grid". It includes a "Filter Rows" search bar and "Edit" and "Export/Import" buttons. The result grid displays the following data:

	id	Account_id	Card_Number
▶	4	1	1092837465654321
	2	1	1234567890123456
	7	52	8576123451629125
	HULL	HULL	HULL

A new row in `Account_Has_PaymentInfo` table is inserted with `account_id = 52` and `Card_Number` of the credit card we just inserted. This represents that the new credit card is now saved to the user's account.

⚡ Query 1

1 • `use store;`
2 • `SELECT * FROM Orders;`

100% 21:2

Result Grid Filter Rows: Search Edit: Export/Import:

Order_ID	Date	Shipping_Address	Total_Price	Shipping_Method
2	2019-11-29 05:59:28.188283	2275 Ellena Drive Apt 4 Santa Clara, CA 95050	361	EXPRESS
3	2019-11-30 02:00:13.960078	2275 Ellena Drive Apt 4 Santa Clara, CA 95050	92	STANDARD
8	2019-12-03 23:59:52.589983	993 River St. Santa Clara, CA 95050	80	STANDARD
9	2019-12-05 04:40:12.367638	993 River St. Santa Clara, CA 95050	83	STANDARD
10	2019-12-08 08:02:09.302901	850 Meadowbrook North Honolulu, HI 96808	110	EXPRESS
NULL	NULL	NULL	NULL	NULL

A new row is inserted into Orders table of Order_ID 10 and containing the Date, Shipping_Address, Total_Price, Shipping_Method of the order we just placed.

The screenshot shows the MySQL Workbench interface. The top bar has a title 'Query 1' and various icons for database management. A dropdown menu 'Limit to 1000 rows' is open. Below the editor area, there are zoom controls (100%, 37:2) and a toolbar with 'Result Grid', 'Filter Rows', 'Search', 'Edit', and 'Export/Import' options.

```
1 • use store;
2 • SELECT * FROM Order_Contain_Textbook;
```

id	Order_ID	Textbook_ID
8	8	3
10	10	10
3	3	11
1	2	12
2	2	14
9	9	22
NULL	NULL	NULL

In the 2nd row of `Order_Contain_Textbook`, that is the new row just inserted containing `Order_ID 10` and `Textbook_ID 10` representing that the order we just placed contain the textbook with id 10.

The screenshot shows a MySQL Workbench interface. At the top, there's a toolbar with various icons for file operations, search, and navigation. Below the toolbar, a query editor window titled "Query 1" contains the following SQL code:

```
1 • use store;
2 • SELECT * FROM Checkout;
```

Below the query editor, the status bar shows "100%" and "23:2". The main area is labeled "Result Grid" and displays a table with the following data:

	id	Account_id	Order_ID
▶	1	1	2
	2	1	3
	8	1	9
	9	52	10
	NULL	NULL	NULL

The table has four columns: id, Account_id, and Order_ID. The fourth row (id 9) has non-null values for all three columns, specifically Account_id 52 and Order_ID 10.

In the 4th row with id 9 of table Checkout, this new row was inserted with Account_id 52 and Order_ID 10 indicating that the account with id 52 just checkout the order with id 10.

BACKEND LOGIC

```
575     @login_required
576     @require_POST
577     @csrf_exempt
578     def checkout_request(self):
579         if request.is_ajax() == True:
580             Textbooks = json.loads(request.POST.get('textbooks'))
581             Shipping_Address = request.POST.get('shipping_address')
582             Shipping_Method = request.POST.get('Shipping_Method')
583             Billing_Address = request.POST.get('billing_address')
584             Total_Price = request.POST.get('order_total_price')
585             Credit_Card_Number = request.POST.get('credit_card_number')
586             Credit_Card_Name = request.POST.get('credit_card_name')
587             Credit_Card_Security_Code = request.POST.get('credit_card_cvv')
588             Credit_Card_Expire_Date = request.POST.get('credit_card_expiration')
589             Present_Time = datetime.now()
590
591         with connection.cursor() as cursor:
592             # Query to see if there is any credit card with this credit card number in the database
593             payment_info_query = "SELECT * FROM PaymentInfo WHERE Card_Number = %s"
594             Credit_Card = PaymentInfo.objects.raw(payment_info_query, [Credit_Card_Number])[:]
595
596             # If this credit card number doesn't exist in the database, insert a new row
597             # into PaymentInfo table representing a new credit card
598             if len(Credit_Card) == 0:
599                 insert_payment_info_query = """
600                 INSERT INTO PaymentInfo(Card_Number, Card_Name, Security_Code, Expiration_Date, Billing_Address)
601                 VALUES(%s, %s, %s, %s, %s)
602                 """
603                 cursor.execute(insert_payment_info_query, [Credit_Card_Number, Credit_Card_Name, Credit_Card_Security_Code, Credit_Card_Expire_Date, Billing_Address])
604
605             # Query to see if this credit card is already saved to the user's account
606             account_has_payment_info_query = """
607             SELECT * FROM Account_Has_PaymentInfo
608             WHERE Account_id = %s AND Card_Number = %s
609             """
610             result = Account_Has_PaymentInfo.objects.raw(account_has_payment_info_query, [str(request.user.id), Credit_Card_Number])[:]
611
612             # If it is not saved to the user's account, insert a new row into Account_has_paymentinfo
613             # to save the credit card to the user's account
614             if len(result) == 0:
615                 insert_account_has_payment_info_query = """
616                 INSERT INTO Account_Has_PaymentInfo(Account_id, Card_Number)
617                 VALUES (%s, %s)
618                 """
619                 cursor.execute(insert_account_has_payment_info_query, [str(request.user.id), Credit_Card_Number])
```

```
620
621         # Insert a new row into Order table representing that a new order
622         # has been made
623         insert_order_query = """
624             INSERT INTO Orders(Date, Shipping_Address, Shipping_Method, Total_Price)
625             VALUES(%s, %s, %s, %s)
626             """
627         cursor.execute(insert_order_query, [Present_Time, Shipping_Address, Shipping_Method, Total_Price])
628         Order_ID = cursor.lastrowid
629
630         # Insert a new row into Checkout table representing that the user account
631         # has checkout a new order
632         insert_checkout_query = "INSERT INTO Checkout(Account_id, Order_ID) VALUES(%s, %s)"
633         cursor.execute(insert_checkout_query, [str(request.user.id), Order_ID])
634
635         # For each item in the list of textbooks bought
636         for item in Textbooks:
637             # Delete from every account's shopping carts that contains this textbook
638             delete_from_shopping_carts_query = "DELETE FROM Shopping_Cart WHERE Textbook_ID = %s"
639             # Delete from every account's wishlist that contains this textbook
640             delete_from_wishlists_query = "DELETE FROM Wishlist WHERE Textbook_ID = %s"
641             # Update the Listing table by setting Available to 0, representing that it is not available for sale
642             update_listing_query = "UPDATE Listing SET Available = 0 WHERE Textbook_ID = %s"
643
644             # Insert a new row into Order_Contain_Textbook representing that this textbook
645             # belongs to this order (sold)
646             insert_order_contains_textbook_query = """
647                 INSERT INTO Order_CContain_Textbook(Textbook_ID, Order_ID)
648                 VALUES(%s, %s)
649                 """
650             cursor.execute(delete_from_shopping_carts_query, [item])
651             cursor.execute(delete_from_wishlists_query, [item])
652             cursor.execute(update_listing_query, [item])
653             cursor.execute(insert_order_contains_textbook_query, [item, Order_ID])
654
655         connection.close()
656         return JsonResponse({'redirect': reverse('homepage')})
```

```

payment_info_query = "SELECT * FROM PaymentInfo WHERE Card_Number = %s"

Credit_Card = PaymentInfo.objects.raw(payment_info_query, [Credit_Card_Number])[:]

if len(Credit_Card) == 0:

    insert_payment_info_query =
    """
    INSERT INTO PaymentInfo(Card_Number, Card_Name, Security_Code,
                           Expiration_Date, Billing_Address)
    VALUES(%s, %s, %s, %s, %s)
    """

    cursor.execute(insert_payment_info_query, [Credit_Card_Number, Credit_Card_Name,
                                              Credit_Card_Security_Code, Credit_Card_Expire_Date, Billing_Address])

```

This section of code first query the PaymentInfo table for a row with the specified Card Number. If the result return 0 row, that mean the credit card is not yet in the database then we will insert a new row into PaymentInfo containing the credit card information representing the new credit card inserted.

```

account_has_payment_info_query =
"""

SELECT * FROM Account_Has_PaymentInfo
WHERE Account_id = %s AND Card_Number = %s
"""

result = Account_Has_PaymentInfo.objects.raw(account_has_payment_info_query,
                                             [str(request.user.id), Credit_Card_Number])[:]

if len(result) == 0:

    insert_account_has_payment_info_query = """
    INSERT INTO Account_Has_PaymentInfo(Account_id, Card_Number)
    VALUES (%s, %s)
    """

    cursor.execute(insert_account_has_payment_info_query,
                  [str(request.user.id), Credit_Card_Number])

```

Similarly, this section of code query the Account_has_PaymentInfo table and checks if there is a row that shows that this credit card is saved to the user's account. If not, then insert a new row into Account_Has_PaymentInfo to represents that this credit card is now saved to the user's account.

```
insert_order_query =  
"""  
INSERT INTO Orders(Date, Shipping_Address, Shipping_Method, Total_Price)  
VALUES(%s, %s, %s, %s)  
"""  
cursor.execute(insert_order_query, [Present_Time, Shipping_Address, Shipping_Method,  
Total_Price])
```

This section of code insert a new row into Orders table representing a new Order has been made.

```
insert_checkout_query = "INSERT INTO Checkout(Account_id, Order_ID) VALUES(%s, %s)"  
cursor.execute(insert_checkout_query, [str(request.user.id), Order_ID])
```

This part insert a new row into Checkout table representing that the user account has just checkout the newly created order.

```
for item in Textbooks:  
  
    delete_from_shopping_carts_query = "DELETE FROM Shopping_Cart WHERE  
    Textbook_ID = %s"  
  
    delete_from_wishlists_query = "DELETE FROM Wishlist WHERE Textbook_ID =  
    %s"  
  
    update_listing_query = "UPDATE Listing SET Available = 0 WHERE  
    Textbook_ID = %s"  
  
    insert_order_contains_textbook_query =  
    """
```

```

INSERT INTO Order_Contain_Textbook(Textbook_ID, Order_ID)
VALUES (%s, %s)

"""

cursor.execute(delete_from_shopping_carts_query, [item])
cursor.execute(delete_from_wishlists_query, [item])
cursor.execute(update_listing_query, [item])
cursor.execute(insert_order_contains_textbook_query, [item, Order_ID])

```

Note that item here is a textbook_id and Textbooks is just a dictionary that contains Textbook_IDs.

For each item in Textbooks, delete from Shopping_Cart every row with Textbook_ID = item which represents removing the item from every account's shopping cart.

Delete from Wishlist every row with Textbook_ID = item which represents removing the item from every account's wish list.

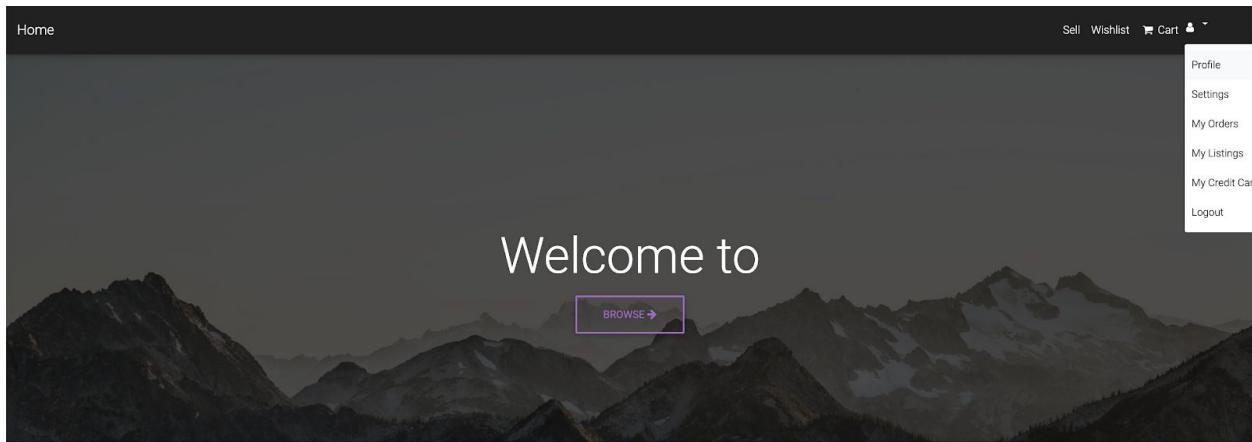
Update the row in Listing table where Textbook_id = item by setting Available to 0 represents updating the listing that it is no longer available for sale.

Insert a new row into Order_Contains_Textbook representing that the new order we created contain this textbook.

Finally, I just return a JsonResponse with a message containing the url to redirect the user after finish checking out the order.

Changing Account Settings:

For this feature, I will use update to update a row from Account table



Press the avatar drop down button on the upper right corner and select Settings in order to change your account information

A form similar to this should appear with your account information filled out

Home Sell Wishlist Cart

EDIT YOUR PROFILE DETAILS



Profile Picture: people.png

Username: test

Email address: newemail@yahoo.com

First name: first

Last name: name

Password:

Password Confirmation:

SAVE

This is the account information after I changed the input fields and click save. I added a new profile picture, changed the email address, first_name, and last_name

The screenshot shows the MySQL Workbench interface. The top bar has tabs for 'Query 1' and other database connections. Below the tabs are various icons for managing databases and queries. A toolbar with icons for file operations, search, and navigation is located above the main query area. The query editor contains the following code:

```
1 • use store;
2 • SELECT id, username, first_name, last_name, email, Profile_Picture FROM Account WHERE id = 52
```

The result grid below shows one row of data:

	id	username	first_name	last_name	email	Profile_Picture
▶	52	test	test	account	test@yahoo.com	NULL
	NULL	NULL	NULL	NULL	NULL	NULL

This query shows the information of the Account before I change the email, first name, last name, and profile picture

⚡ Query 1

1 • `use store;`
2 • `SELECT id, username, first_name, last_name, email, Profile_Picture FROM Account WHERE id = 52;`

3
4
5
6
7

100% 67:2

Result Grid Filter Rows: Search Edit: Export/Import:

	id	username	first_name	last_name	email	Profile_Picture
▶	52	test	first	name	newemail@yahoo.com	people_DWJsyG4.png
	NULL	NULL	NULL	NULL	NULL	NULL

After changing the information, you can see that row in Account table has updated the first_name, last_name, email column and Profile Picture is no longer null.

Backend Logic:

```
161 @login_required
162 def settings(request):
163
164     user_query = "SELECT * FROM Account WHERE username = %s;"
165     user_query_arguments = [request.user.username]
166
167     user = Account.objects.raw(user_query, user_query_arguments)[0]
168     data = {'email': user.email, 'first_name': user.first_name, 'last_name': user.last_name}
169
170     # Populate the form with the user's email, first name, last name
171     form = forms.AccountChangeForm(initial=data)
172     if request.method == 'POST':
173         form = forms.AccountChangeForm(request.POST)
174         with connection.cursor() as cursor:
175             if(form.is_valid()):
176                 modified_data = {};
177                 password = make_password(request.POST.get('password'))
178                 if request.user.check_password(request.POST.get('password')):
179                     pass;
180                 else:
181                     cursor.execute("UPDATE Account SET password = %s WHERE username = %s", [password, request.user.username])
182                     user = Account.objects.raw('SELECT * FROM Account WHERE username = %s AND password = %s', [request.user.username, password])[0]
183                     auth_login(request, user);
184
185                 if request.FILES.get('Profile_Picture') is not None:
186                     request.user.Profile_Picture = request.FILES.get('Profile_Picture');
187                     request.user.save()
188
189                 cursor.execute("UPDATE Account SET email = %s, first_name = %s, last_name = %s WHERE username = %s", [request.POST.get('email'),
190                                         request.POST.get('first_name'), request.POST.get('last_name'), request.user.username])
191
192                 modified_data['email'] = request.POST.get('email')
193                 modified_data['first_name'] = request.POST.get('first_name')
194                 modified_data['last_name'] = request.POST.get('last_name')
195                 new_form = forms.AccountChangeForm(initial=modified_data)
196                 auth_login(request, request.user)
197                 return render(request, 'Settings.html', context={'form': new_form})
198             else:
199                 print(form.errors)
200
201         connection.close()
202     return render(request, 'Settings.html', context={'form': form});
```

Line 147-154 is for querying user account information and constructing an account settings change form using the data we query

```
user_query = "SELECT * FROM Account WHERE username = %s;"

user_query_arguments = [request.user.username]

user = Account.objects.raw(user_query, user_query_arguments)[0]

data = {'email': user.email, 'first_name': user.first_name, 'last_name':
user.last_name}

form = forms.AccountChangeForm(initial=data)
```

This retrieves the email, first name, last name info stored in your account and formatted into a dictionary. Then a form is initialized using that dictionary so that the form is pre-filled with the

account information such as the email input field being filled with your email address stored in the database.

LINE 155 - 181 is where the logic for updating account information is done:

```
password = make_password(request.POST.get('password'))  
  
if request.user.check_password(request.POST.get('password')):  
  
    Pass;  
  
Else:  
  
    cursor.execute("UPDATE Account SET password = %s WHERE username = %s;",  
    [password, request.user.username])  
  
    user = Account.objects.raw('SELECT * FROM Account WHERE username = %s AND  
    password = %s', [request.user.username, password])[0]  
  
    auth_login(request, user);
```

First, the password is hashed using the `make_password` function. Then the `check_password` function is used to see if the hashed password matches the current password. If it does, then it does nothing. Else, I update the `Account` table with the new hashed password by filtering with `username`

Since I just modified the logged in user's information, Django requires me to relogin so I query the row in `Account` that matches the `username` and `password` and use `auth_login` to logs in.

```
if request.FILES.get('Profile_Picture') is not None:  
  
    request.user.Profile_Picture = request.FILES.get('Profile_Picture');  
  
    request.user.save()
```

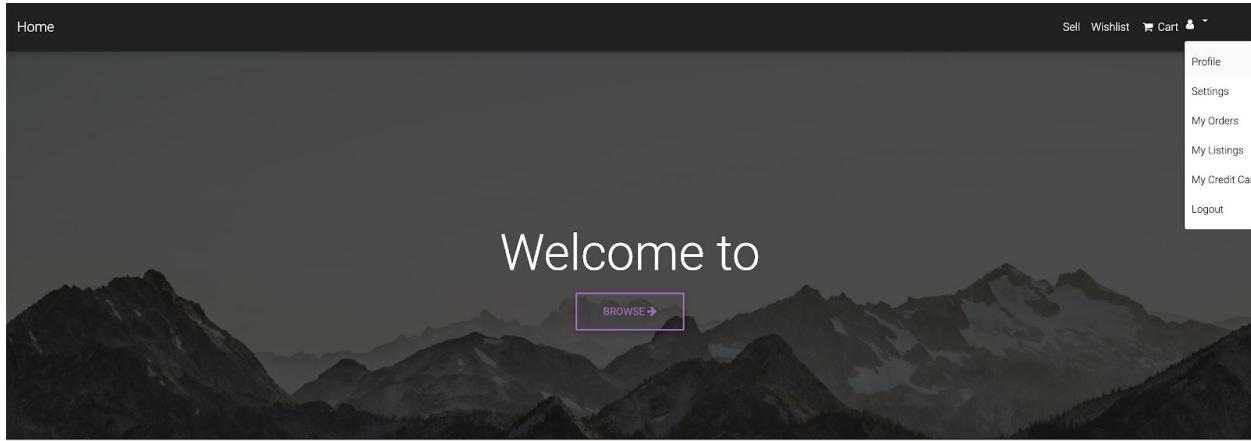
This section checks if the Profile Picture was changed. If yes, then uploads the Profile Picture into the media folder.

Finally in line 172,

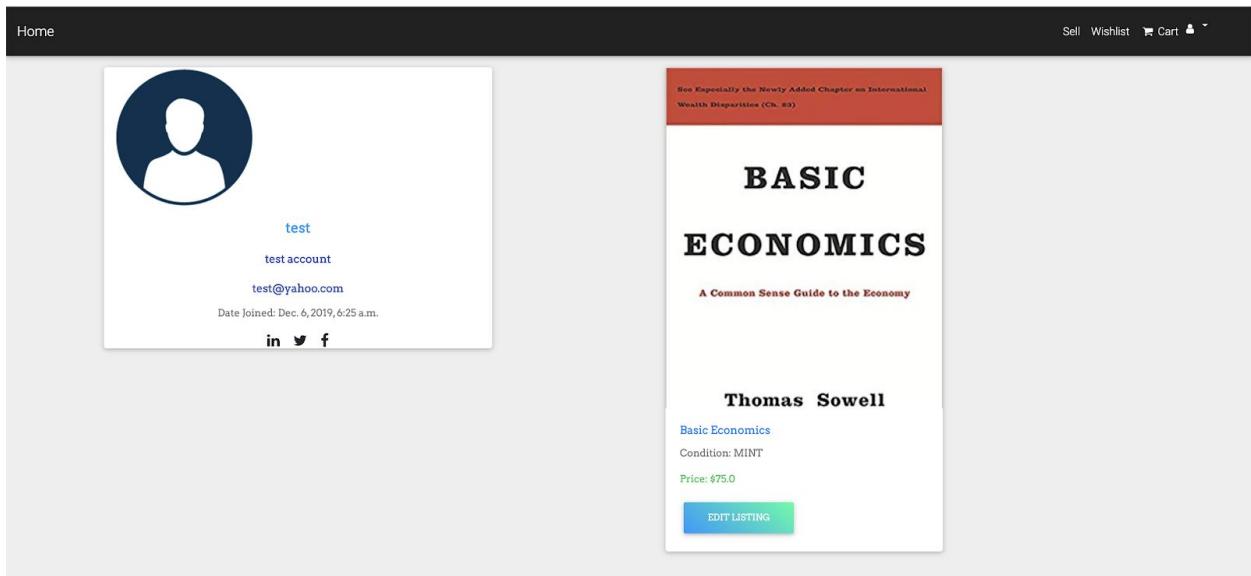
```
cursor.execute("UPDATE Account SET email = %s, first_name = %s, last_name = %s WHERE  
username = %s;", [request.POST.get('email'), request.POST.get('first_name'),  
request.POST.get('last_name'), request.user.username])
```

This query simply update the email, first_name, last_name by filtering based on the session username.

VIEW PROFILE:



As shown above, you can click on the avatar dropdown button on the upper right corner and clicks on Profile option to view your profile. It will return to you a view similar to below:



The left side contains your profile picture (if you don't have one then it shows a default picture) and on the right side will be a list of your active listings for sale.

The screenshot shows the MySQL Workbench interface. The top bar has tabs for 'Query 1' and other database connections. Below the tabs is a toolbar with icons for file operations, search, and connection management. A dropdown menu says 'Limit to 1000 rows'. The main area contains a numbered SQL query:

```

1 • use store;
2 • SELECT A.Account_ID, C.username, B.Textbook_ID, Price, A.Available, Title, ISBN
3   FROM Listing A, Textbook B, Account C
4   WHERE C.username = 'test' AND A.Account_id = C.id AND B.Textbook_ID =
5     A.Textbook_ID AND A.Available = 1;
6
7
8

```

Below the query is a status bar showing '100%' and '80:2'. The bottom section is a 'Result Grid' showing the query results:

	Account_ID	username	Textbook_ID	Price	Available	Title	ISBN
▶	52	test	24	75	1	Basic Economics	9780465060733

This is the view active listings query result which matches up with the GUI above.

```

226
227 @login_required
228 def view_profile(request, account_username):
229     account = Account.objects.raw("SELECT * FROM Account WHERE username = %s", [account_username])[0];
230     listing_query = """
231     SELECT *
232     FROM Listing A, Textbook B, Account C
233     WHERE C.username = %s AND A.Account_id = C.id AND B.Textbook_ID = A.Textbook_ID
234     AND A.Available = 1
235     """
236     listings = Listing.objects.raw(listing_query, [account_username])
237     return render(request, 'Other_Users_Profile.html', context={'account':account, 'Listings':listings})
238

```

This function takes in an argument `account_username` used to identify which account profile will be displayed.

First query:

```

account = Account.objects.raw("SELECT * FROM Account WHERE username = %s",
[account_username])[0];

```

Straight forward, simply select the row in `Account` that matches the argument `account_username`

```

listing_query =

```

```
"""
SELECT *
FROM Listing A, Textbook B, Account C
WHERE C.username = %s AND A.Account_id = C.id AND B.Textbook_ID =
A.Textbook_ID
AND A.Available = 1
```

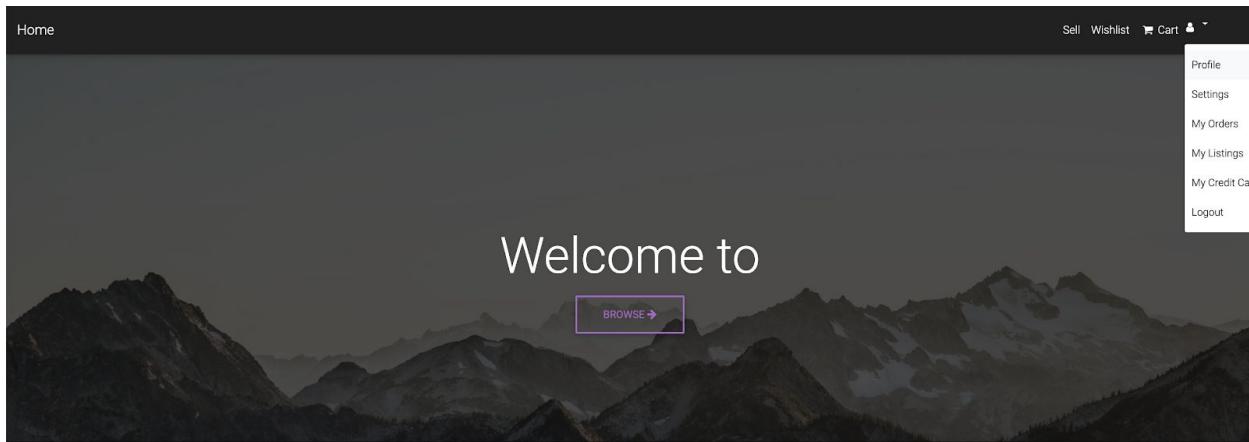
```
"""
```

```
listings = Listing.objects.raw(listing_query, [account_username])
```

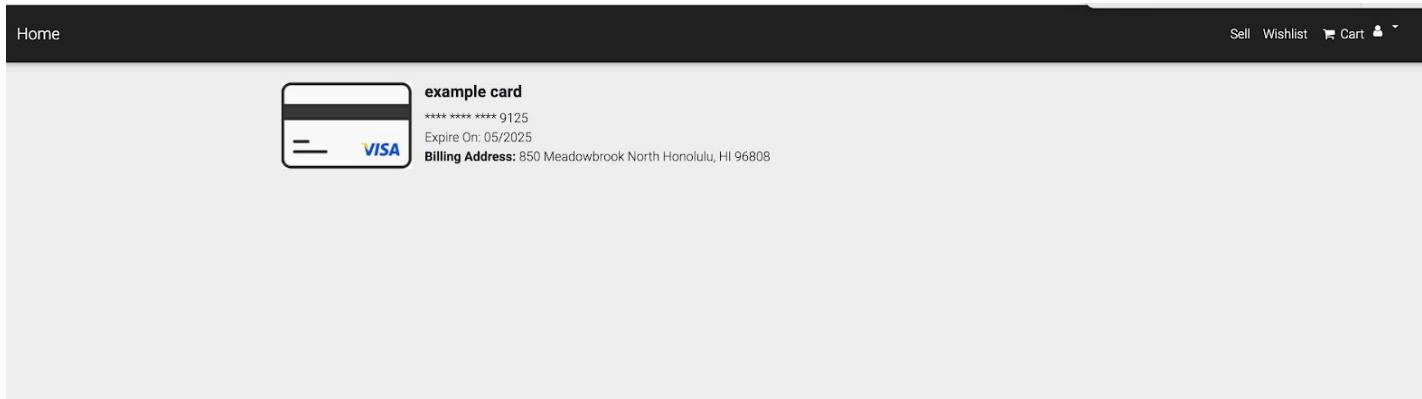
This join table Listing, Textbook, Account where the Account.username = account_username, where account_id of Listing & Textbook are the same, and where Textbook_id of Textbook & Listing are the same.

Then this function passes in the results of account query and listing query into render function to render Other_Users_Profile.html

VIEW CREDIT CARDS



As shown above, click on the avatar picture in the upper right corner and you can click on the My Credit Cards option to view the credit cards saved to your account.



The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons like file, save, and search. Below it is a query editor window titled "Query 1" containing the following SQL code:

```

1 • use store;
2 • SELECT *
3   FROM PaymentInfo A, Account_Has_PaymentInfo B
4   WHERE A.Card_Number = B.Card_Number AND B.Account_ID = 52;
5
6

```

Below the code is a progress bar at 100% and a timestamp of 59:4. The main area shows a "Result Grid" with the following data:

Card_Number	Card_Name	Security_Code	Expiration_Date	Billing_Address	id	Account_id	Card_Number
8576123451629125	example card	765	05/2025	850 Meadowbrook North Honolulu, HI 96808	7	52	8576123451629125

This is result of view credit card query in MySQL workbench which matches the results shown in the GUI.

BACKEND LOGIC

```

562
563     @login_required
564     def view_credit_cards(request):
565
566         # Selects all the credit cards saved to the user's account
567         Credit_Card_Query = """
568             SELECT *
569             FROM PaymentInfo A, Account_Has_PaymentInfo B
570             WHERE A.Card_Number = B.Card_Number AND B.Account_ID = %s
571             """
572         Credit_Cards = PaymentInfo.objects.raw(Credit_Card_Query, [str(request.user.id)])
573         return render(request, 'My_Credit_Cards.html', context={'Credit_Cards': Credit_Cards})
574

```

Credit_Card_Query =

```

"""
SELECT *

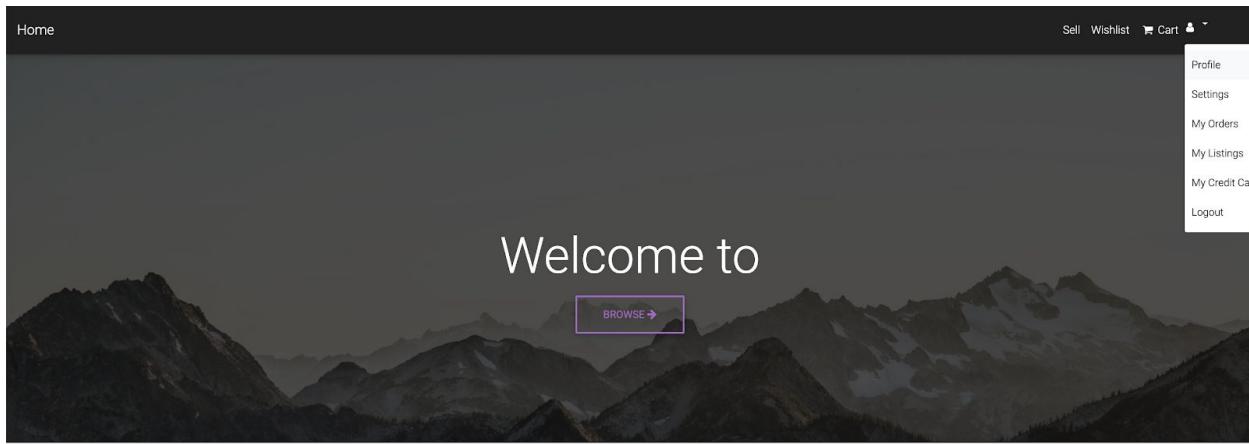
```

```
FROM PaymentInfo A, Account_Has_PaymentInfo B  
WHERE A.Card_Number = B.Card_Number AND B.Account_ID = %s  
"""
```

This query joins PaymentInfo and AccountHasPaymentInfo table on the condition that their Card Number are the same and the Account_ID is equals to the logged in user's account id.

The purpose is to selects all the credit cards saved to that user's account and pass the result into the render function for rendering My_Credit_Cards.html view.

VIEW ORDERS



You can click on the avatar dropdown button in the upper right corner and select My Orders to view your order history.

Order History

Order ID: 10

Cracking the Coding Interview

ISBN: 9780984782857

Author: Gayle Lakhman McDowell

Publisher: CareerCup

Condition: LIKE NEW

Listed for sale by: jesseking

Price: \$98.0

Shipping Method: EXPRESS

Shipping Cost: \$12.00

Time of Purchase: Dec. 8, 2019, 8:02 a.m.

Total price: 110

Your new order will appear in your order history

The screenshot shows the MySQL Workbench interface. The top bar has a tab labeled "Query 1". Below the tab are various icons for file operations, search, and help. A dropdown menu says "Limit to 1000 rows". The main area contains a numbered SQL query:

```
1 • use store;
2 • SELECT A.Account_ID, B.Order_ID, Date, Shipping_Address, Total_Price, D.Textbook_ID, Title, ISBN
3   FROM Checkout A, Orders B, Order_CContain_Textbook C, Textbook D, Listing E
4  WHERE A.Account_id = 52 AND A.Order_ID = B.Order_ID AND A.Order_ID = C.Order_ID
5    AND D.Textbook_ID = C.Textbook_ID AND E.Textbook_ID = D.Textbook_ID
6    AND E.Available = 0 AND A.Order_ID = 10;
7
```

The status bar at the bottom shows "100%" and "97:2". Below the status bar is a toolbar with "Result Grid" selected, a search bar, and an export button.

Account_ID	Order_ID	Date	Shipping_Address	Total_Price	Textbook_ID	Title	ISBN
52	10	2019-12-08 08:02:09.302901	850 Meadowbrook North Honolulu, HI 96808	110	10	Cracking the Coding Interview	9780984782857

This is the view order query results from MySQL workbench for Order_ID 10 and Account_ID 52 which matches with the results returned on the GUI.

BACKEND LOGIC

```
637 @login_required
638 def view_orders(request):
639
640     # Select all of the order checkouts that the user has done
641     Checkouts = Checkout.objects.raw("SELECT * FROM Checkout WHERE Account_id = %s ORDER BY Order_ID DESC;", [str(request.user.id)])
642     Checkouts = Checkouts[::]
643     Orders = {}
644
645     # For each checkout the user has done, perform a few query to find the list of textbooks
646     # and lsit of sellers for those textbooks for that checkout order.
647     for checkout in Checkouts:
648
649         # Select the list of textbooks that was bought in this checkout
650         textbooks_query = """
651             SELECT *
652             FROM Checkout A, Orders B, Order_CContain_Textbook C, Textbook D, Listing E
653             WHERE A.Account_id = %s AND A.Order_ID = B.Order_ID AND A.Order_ID = C.Order_ID
654             AND D.Textbook_ID = C.Textbook_ID AND E.Textbook_ID = D.Textbook_ID
655             AND E.Available = 0 AND A.Order_ID = %s
656             """
657         list_of_textbooks = Order_CContain_Textbook.objects.raw(textbooks_query, [str(request.user.id), str(checkout.Order.Order_ID)])
658         list_of_textbooks = list_of_textbooks[::]
659
660         # Select the list of sellers that sold the textbooks bought in this checkout
661         sellers_query = """
662             SELECT *
663             FROM Account A, Listing B, Order_CContain_Textbook C
664             WHERE A.id = B.Account_ID AND B.Available = 0 AND B.Textbook_ID = C.Textbook_ID AND C.Order_ID = %s
665             """
666         list_of_sellers = Account.objects.raw(sellers_query, [str(checkout.Order.Order_ID)])
667         list_of_sellers = list_of_sellers[::]
668
669         # This dictionary Orders contains the information of all orders the user has checkout
670         # Each element in this dictionary is of the format [Order ID] : [list of textbook, list of sellers]
671         # I use the zip function of list_of_textbooks and list_of_sellers because it let me iterate
672         # through both list at once and that is useful since the first textbook in list of textbook
673         # is sold by the first seller in list of sellers and so on
674         Orders[checkout.Order.Order_ID] = zip(list_of_textbooks, list_of_sellers)
675
676     return render(request, 'Orders.html', context={'Orders': Orders})
677
```

```
Checkout.objects.raw("SELECT * FROM Checkout WHERE Account_id = %s ORDER BY Order_ID
DESC;", [str(request.user.id)])
```

This query selects all the rows in Checkout with Account_id equals to the logged in user's account id which is all of the checkout orders that the account has done.

Then for each row returned in the above checkout query, two query is performed:

```
textbooks_query =
```

```
"""
SELECT *
FROM Checkout A, Orders B, Order_CContain_Textbook C, Textbook D, Listing E
WHERE A.Account_id = %s AND A.Order_ID = B.Order_ID AND A.Order_ID = C.Order_ID AND
D.Textbook_ID = C.Textbook_ID AND E.Textbook_ID = D.Textbook_ID AND E.Available = 0
AND A.Order_ID = %s
```

```
"""

```

This query join Checkout, Order, Order_CContain_Textbook, Textbook, and Listing table on the condition that Order_ID of Orders & Checkout & Order_CContain_Textbook are the same, Textbook_ID of Textbook & Order_CContain_Textbook & Listing are the same, Listing.Available = 0 (not available for sale), and Order_ID equals to the specific order ID of that checkout.

The purpose is to find the specific list of textbooks sold for that checkout and all of the relevant information about that textbook listing.

```
sellers_query =
"""
SELECT *
FROM Account A, Listing B, Order_CContain_Textbook C
WHERE A.id = B.Account_ID AND B.Available = 0 AND B.Textbook_ID = C.Textbook_ID AND
C.Order_ID = %s
"""

```

This query join Account, Listing, and Order_CContain_Textbook on the condition that Account_ID of Account & Listing are the same, Textbook_ID of Listing and Order_CContain_Textbook are the same, Listing.Available = 0 (not available for sale) and Order_ID is equals to the order ID of that checkout.

The purpose is to find the list of sellers that sold the textbooks bought for that checkout.

The first textbook returned in the list of textbook query is sold by the first seller return in the list of sellers query and so on.

The results is then formattted into an Orders dictionary which contains the information about all the orders that the user checkout and then passed into the render function for rendering Orders.html to return a view of all orders purchased.

HOW TO SETUP ON MACOS

1. Install python3 by following this [link](#) and downloading the package and install
2. Download get-pip.py running the following command in terminal

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

3. Now run

```
python3 get-pip.py
```

4. Pull our git repo using git pull to your machine

<https://github.com/aldrichmangune/CS157A-Team25>

5. Install virtualenv with

```
pip3 install virtualenv
```

6. Set up a directory for your virtual environment with

```
python3 -m virtualenv dirname (choose a dirname)
```

7. Run source dirname/bin/activate, where dirname is the directory you just set up for your virtual environment

8. Make sure your virtual environment is activated and run

```
pip3 install django=='2.2.6'
```

9. Now run

```
pip3 install mysql-connector-python
```

10. Now run

```
pip3 install Pillow
```

11. Now head over to where you pulled our git repo and move into the project folder where our code is contained. Should be in CS157A-TEAM25/project folder

12. Go to project/settings.py, open it and changes the database credentials to match yours.

(Note: this is located in CS157A-TEAM25/project/project/settings.py)

```
81
82 DATABASES = {
83     'default': {
84         'ENGINE': 'mysql.connector.django',
85         'NAME': 'store',
86         'USER': 'root',
87         'PASSWORD': 'password',
88     }
89 }
90
```

This part in the settings.py is the database settings. Name is name of the database, user is username and password is password used to login to MySQL.

13. Run python3 manage.py makemigrations

It should say no changes detected

14. Run python3 manage.py migrate

15. Now go to MySQL workbench or login to MySQL with your terminal and write

```
INSERT INTO Category VALUES ('Computer Science'), ('Chemistry'),
('Mathematics'), ('History'), ('Law'), ('Medicine'), ('Physics'),
('Engineering'), ('Arts'), ('Literature'), ('Finance');
```

Our webapp doesn't let the user add in Categories so we add the Category ourselves with MySQL workbench.

16. Go CS157A-TEAM25/project and to starts the server run

python3 manage.py runserver

(Note: When you want to run the application, make sure your virtualenv you created earlier is activated using source virtualenvdirectory/bin/activate like before)

17. Go to localhost:8000 to run our web app.

PROJECT CONCLUSION

Improvements for Future Version:

In the future version, we would like to add some credit card verification API to check if the credit card is valid also implements Paypal API for users to pay with Paypal. We would also like a review feature for the buyers to add reviews and rate sellers. Instead of just one picture for each textbook listing, we ideally would like a panels of pictures that would best display the condition of the textbook. A chat feature or some kind of message system between users would help facilitate the purchasing process between buyer and seller. Finally, we would also probably redesign the UI to makes it look more professional.

Lessons Learned:

I have never worked on a 3 tier architecture prior to doing this project and it wasn't easy to finish this project in the small timeframe we had. When I started this project, all I knew was Python and HTML5/CSS3 so I had to learn an entirely new web framework (Django), MySQL, Bootstrap, Javascript and JQuery in order to build this project. Overall, this project has propelled me to learn a lot of new things beyond just databases that will be helpful for me in the future. I regret skimping on the UI since I had neither the time nor the expertise to polish it as I had hoped. Regardless, I am happy with the backend features we have done and what I have learned from this experience.

- Au Tran