# FINAL PROJECT REPORT

Group 24

Isha Afzaal, Brooklyn Christopher and Rubaiyet Meem

Professor: Dr. Reda Elhajj

CPSC 471 - Database Management Systems - Fall 2020

December 11, 2020

# Abstract

As prices of post-secondary course textbooks rises every year, many students facing economic hardships are disadvantaged when they are unable to procure much-needed learning resources. When unable to purchase new textbooks, many students simply opt to complete courses without them or negotiate trades or purchases with peers. When investigating this issue, we found that currently, many post-secondary students in Calgary utilize a Facebook group for purchasing resources. Drawing from this model, the Textbook Exchange application (called Textbook Exchange App from hereon) created by our group aims to create a peer-to-peer network where students, alumni and retailers are able to register for buyer and/or seller accounts to facilitate textbook exchanges where all parties benefit.

This paper outlines the project's problem-scope, how it solves the problem, users of the system such as students, retailers and staff, details of implementation and a how-to run guide along with API documentation. Built using Django, Django REST framework and Python, the Textbook Exchange App did not undergo major changes from the design to implementation and testing phase. Key concepts detailed in the project's EERD, Relational and OO diagram in its planning phase were also largely retained in the final API endpoint implementation. The Textbook Exchange App largely focuses on providing functionality and views for end-users such as buyers, sellers and administrative staff. As implied, it takes into account differing levels of technological skills for its end-users. Finally, a simple how-to-run guide is provided at the end of the paper to aid any user in running the Textbook Exchange App source code.


# Introduction

The Textbook Exchange App database chiefly attempts to solve the problem of easing post-secondary students' financial burdens when purchasing course textbooks. It also is a response to annually increasing textbook prices. When researching this issue, we found that students are expected to spend between $1200 - $1400 on textbooks for the current academic year. Additionally, the inflation of textbook prices places financial burden on students, as studies have shown that students oftentimes forgo attaining learning resources and feel that their learning is impacted as a result. The Textbook Exchange App facilitates peer-to-peer textbook exchanges and enables students and retailers to buy and sell their textbooks at rates that appeal to this demographic.

# System Solution

Information/data about users, textbooks, courses, universities and interactions are kept in an SQL database (using SQL LITE in Django). End-users interact with the database via API endpoints. Different users use the endpoints in various ways. For example, administrative staff with certain permissions are able to create, modify and delete user accounts. Our system allows users to be able to search through textbooks in the database via book name and academic course. The application also allows users to add their own textbooks to the database to be sold to others. They can view all the textbooks that are being sold, who is selling them, and the initial price they are offering. In order to facilitate the textbook exchange, a chat feature will also be available, which will allow buyers and sellers to interact and decide on prices, methods of payment and how the exchange will take place.

# Project Design

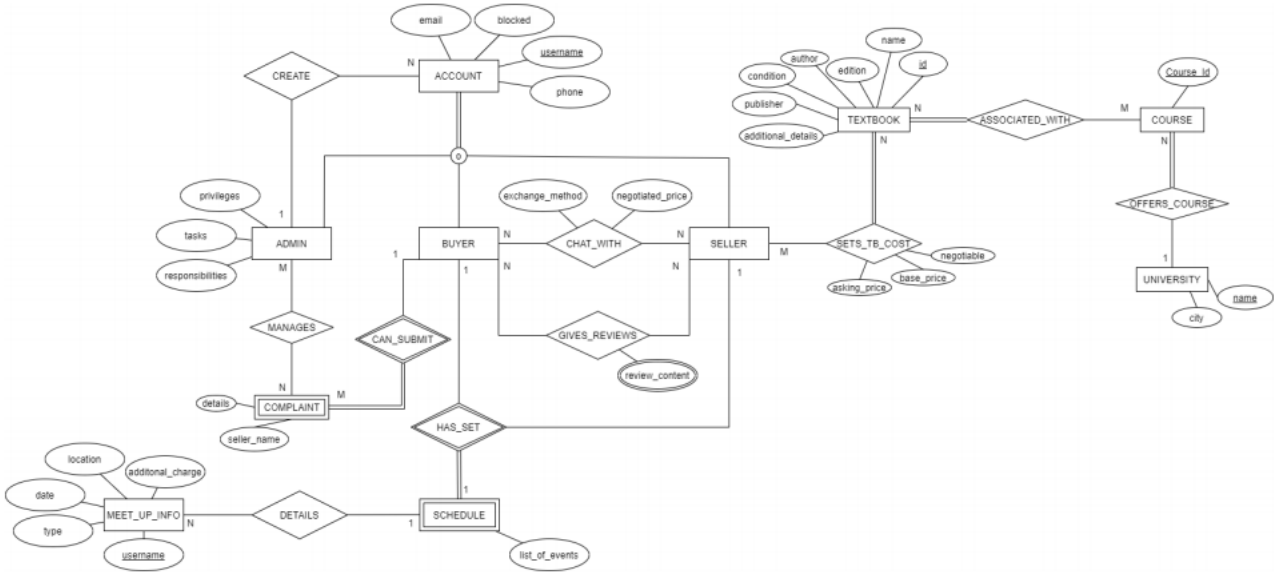
## System Users

In the Textbook Exchange App, all users are able to find textbooks using various filters such as course name, course number, university, textbook name, author, edition and year. Users are also given the ability to select a geographical area or university of their choosing for search results and possible interactions. Not only can users search for textbooks to purchase, but they can add their own textbooks to sell as well. The seller adds the textbook name, author, edition, university, edition, associated course and year for the textbook along with its condition and asking sale price. Our application features an encrypted chat feature for buyers and sellers to interact and discuss exchange options, negotiate prices, additional details, and payment methods. Buyers and sellers are able to give and receive reviews for each other after the exchange is completed. This helps to ensure buyers and sellers are acting accordingly, as well as advising others of difficult users.

This application has various users including alumni, prospective students, students who have previously completed a course, retailers, professors, self-taught individuals and administrative staff. After completing their degree, many alumni no longer have a need for some of their textbooks, and will look for ways to sell them. This is the same for students who have just completed a course which required a textbook, but have no need for them afterwards. Many of them sell textbooks from previous courses to fund the textbooks required for the new semester. Students who are about to take a class which requires a specific textbook often do not want to pay the full price for a brand new textbook, thus this application allows them to search for a used textbook to purchase. Retailers who sell used textbooks can use this platform to reach a larger customer base in their local area. Some individuals choose to self-teach a wide range of topics that many individuals go to school for. A large majority of those who self-teach either are unable to afford to go to university or have a strong interest to learn for fun. These individuals often choose to save money by purchasing used textbooks. Some professors may use this website as a method to provide their students with an affordable alternative to brand new textbooks. This application provides a place professors direct their students to. Lastly, administrative staff are constant users of the system. These include admins, the people who manage the database, and the developers, the people who maintain and update the software for the REST API.

Students, alumni and retailers would use the app in similar ways: sign-up to become a buyer and/or seller and either browse and purchase textbooks as buyers or post textbook details to sell. On the flipside, administrative staff would use the app to manage users, complaints and reviews alongside account and textbook management.

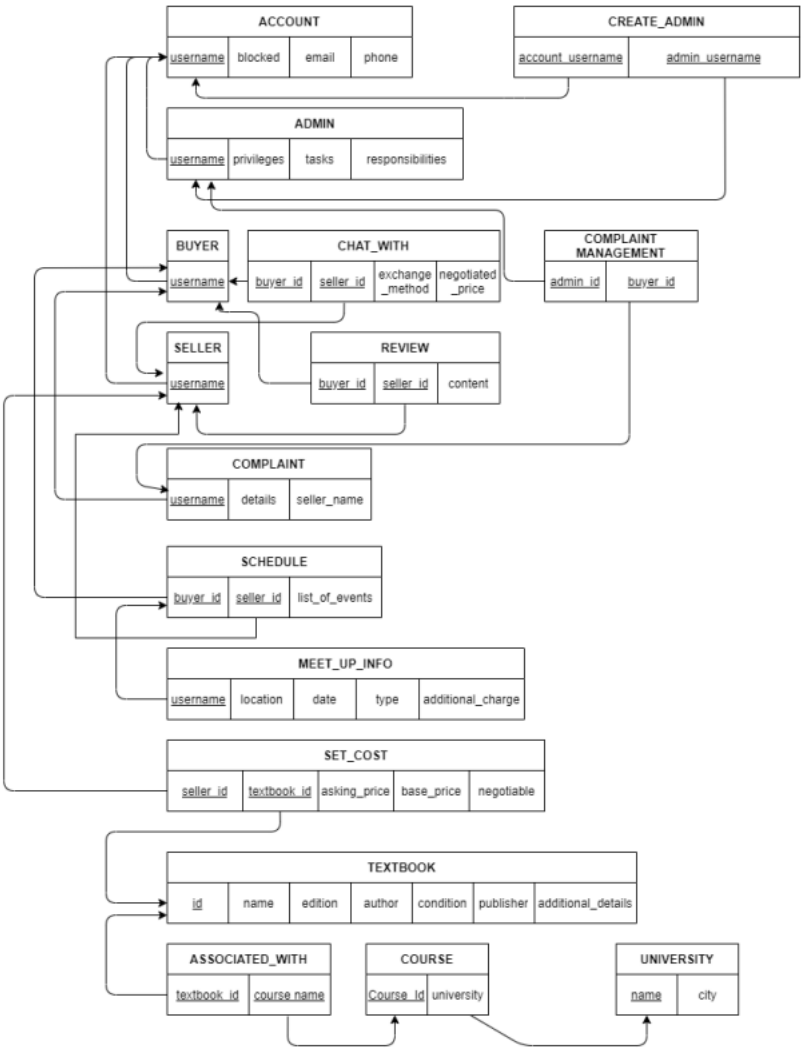
## ER Diagram




No changes have been made since the presentation on Wednesday, December 9th, 2020.

# Implementation

## Relational Model




Direct algorithm conversion was used between our ERD and the relational model. No significant changes were required during these conversions and all relationships were maintained in the creation.

## Chosen DBMS

Django was used as the DMBS for this project as it is an SQL based software (using SQL LITE). Django is a free, easy-to-use high-end framework that is commonly found in web development. It was also beneficial as the code is written in Python, a simple and easy to manage programming language that all group members were familiar with. Django is also organized into an extremely organized system that is easy to follow, which was beneficial when debugging the REST API endpoints. Lastly, Django has an extensive built in security system. Django not only hides source code, but allows for the user configuration and authentication.

## Implemented SQL Statements

The SQL statements implemented in the system generally are of the following forms:

- GET all data in a table: SELECT * FROM A;
- GET a certain data item: SELECT * FROM A WHERE id = given_id;
- POST: INSERT INTO A VALUES (i, j, k);
- PUT: UPDATE A SET variable = new WHERE id = given_id;
- DELETE: DELETE FROM A WHERE id = given_id;

Where A is the target table, given_id is the user-requested ID of the target object, (i, j, k) are table parameters and new is the new value to set the variable to.

The following is a list of the SQL implementations. For the sake of simplicity, the queries are grouped by the tables they are involved with.

## Table: Textbook

**Query:**
INSERT INTO TEXTBOOK
VALUES (id, name, edition, author, condition, publisher,
additionalDetails);

**Query:**
UPDATE TEXTBOOK
SET condition=cond, additional_details=additionalDets
WHERE id = tbID;

**Query:**
SELECT *
FROM TEXTBOOK
WHERE Name = name;

**Query:**
SELECT DISTINCT (name, edition)
FROM TEXTBOOK;

**Query:**
DELETE FROM TEXTBOOK
WHERE id = ID;

DELETE FROM ASSOCIATED_WITH
WHERE textbook_id = ID;

DELETE FROM SET_COST
WHERE textbook_id =ID;

### Table: Course

**Query:**
INSERT INTO Courses
VALUES (courseName, courseId, uni)

searchCourse(courseId:string):boolean
**Query:**
SELECT *
FROM Courses
WHERE number = courseId

**Query:**
SELECT *
FROM Courses

**Query:**
SELECT Textbook.name
FROM Textbooks, Courses, Associated_With
WHERE textbook_id = id AND course_name = Courses.name AND number = courseId

### Table: University

**Query:**
INSERT INTO Universities
VALUES (name, city)

**Query:**
UPDATE Universities
SET name = newName, city = city
WHERE name = originalName

**Query:**
SELECT *
FROM Universities
WHERE name = name AND city = city

**Query:**
SELECT *
FROM Universities

### Table: Buyer

**Query:**
SELECT *
FROM Sellers
WHERE username = sellerId

**Query:**
INSERT INTO Complaints
VALUES (userId, content, againstUserId)

**Query:**
INSERT INTO Schedule
VALUES (buyerId, event)

### Table: Account

**Query:**
SELECT *
FROM Account
WHERE userID = username

### Table: Admin

**Query:**
INSERT INTO Admin
VALUES (username, privileges)

**Query:**
DELETE FROM Complaint
WHERE complaintID = username

**Query:**
INSERT INTO Complaint
FROM Account
WHERE complaintID = username

**Query:**
SELECT *
FROM Complaint
WHERE complaintID = username

**Query:**
INSERT INTO Admin
VALUES (accountID = account_username)

**Query:**
DELETE FROM Admin
WHERE accountID = account_username

### Table: Seller

**Query:**
SELECT username
FROM Seller

UPDATE Set_Cost
SET price = asking_price
WHERE username = seller_id

### Table: Meeting

**Query:**
DELETE FROM Schedule
WHERE meetingID = meetingId

**Query:**
INSERT INTO Schedule
WHERE location

### Table: Schedule

**Query:**
SELECT *
FROM Schedule
WHERE user_id = seller_id

# API Documentation

The API documentation for the Textbook Exchange App was created using Postman.
It is accessable at the following link: https://documenter.getpostman.com/view/13691161/TVmTcEpY

# User Guide

## Technology Required

Python 3, Django and Django REST Framework
If you are not able to create virtual environments, install virtual env using Python: pip install virtualenv

## Instructions

1. Download Textbook Exchange App source code
2. Open command line or terminal > Navigate to the directory containing the project's source code folder
3. Create a virtual environment using the following command: virtualenv nameofenvhere
4. Activate the virtual environmen: nameofenvhere
Scripts
activate
5. Set-up Django in the virtual environment: pip install Django
6. Set-up Django REST Framework in the virtual environment: pip install djangorestframework
7. Ensure that all migrations are applied as needed: py manage.py makemigrations
8. py manage.py migrate
9. To start the database server on the local machine: py manage.py runserver
10. You are now able to send requests to and receive responses from the Textbook Exchange App database using applications such as Postman

Every model in the database has a designated GET, POST, PUT and DELETE function. They are all used similarly - for example, a GET request for all users is similar to GET requests for all textbooks. To find what endpoints are available and what fields are required, please view the urls.py and models.py files in the
api directory found within the project source code. The following are examples of each kind of operation.

The pre-requisite for each operation is that the Textbook Exchange App server is running locally.

## Example: Get a List of All Textbooks

1. Open API-testing software such as Postman or Insomnia
2. If running the server locally, set the request URL to http://127.0.0.1:8000/textbooks
3. Set the request type to 'GET' > Send
4. Done: Should now receive a JSON containing all textbooks within the database

## Example: How to Create a New Account

1. Open Postman
2. Set the request URL to http://127.0.0.1:8000/users
• List of API endpoints are found in the urls.py file within the api directory in the project files
3. Set request type to 'POST'
4. Form the JSON request body. An example is:

{“username”:”8”,
“email”:”exampleuser@eg.com”,
“phone”:”3001214444”
}

In the case of creating new accounts, only the username field is mandatory.
5. Click Send
6. Done: Assuming that you picked a username not already existing in the database, a new account has now been created

### Example: How to Update an Account

1. Open Postman
2. Note which account username you want to update. For example, we will use username “8”
3. Set the request URL to http://127.0.0.1:8000/users/8
4. Set request type to ‘PUT’
5. Form the JSON request body:

{“username”:”8”,
“email”:”alteredemail@example.com”,
“phone”:”1110004444”
}
6. Click Send
7. Done: Should now have altered the account with username “8”. To check your results, change the request type to ‘GET’ > Send

### Example: How to Delete an Account

1. Open Postman
2. Note which username you want to delete. For example, we will use username “8”
3. Set the request URL to http://127.0.0.1:8000/users/8
4. Set request type to ‘DELETE’
5. Click Send
6. Done: Account with username “8” should now be deleted. To verify, set the request type to ‘GET’ > Send. The result should be an empty list as the account does not exist anymore