DATA MANAGEMENT SYSTEMS

FINAL PROJECT REPORT

GROUP 25 PRESENTS

OTU BOOKSTORE

AUTHORED BY

Mazen Massoud (100580842)

Alec De Sousa (100709863)

Aayush Parikh (100724827)

Naweed Adel (100660467)

Index

OTU Bookstore

Abstract

The OTU Bookstore website provides an efficient, user-friendly and COVID-friendly solution to the stressful rush of buying textbooks. This was done by leveraging Node.js to create a website that is intuitive, descriptive, robust and secure. OTU Bookstore also prides itself on being completely COVID-friendly. Through the precautions during the ordering and delivering process and the very fact that most of the process is online, the Bookstore strictly follows all COVID regulations. Through these factors, the act of purchasing textbooks for university has been greatly simplified. A user only needs a valid *ontariotechu.net* email account, and they can view every course they are in, and all textbooks required for a given course. At checkout, the price of the textbooks are automatically included in the user's tuition so that they need not pay immediately. Once the textbook is delivered and ready to be acquired, the user will need to go to the physical bookstore. There, following strict COVID-19 guidelines, the user will receive the book with limited contact among staff.

Introduction

For this project, we wanted to design a bookstore accessible to all users, disregarding the current situation at hand. We wanted to create something that would consider COVID-19 when being implemented. We made many design choices that revolved around COVID-19. The most important of which was the addition of in-store pick-up or online delivery; both of these options will be discussed in this report. We also wanted to make sure that the experience of buying a textbook was as seamless as possible, which is why everything has been organized perfectly in the database. Students can log in using their studentID, and browse through a collection of textbooks and filter the book to see the books associated with their currently enrolled courses. This entire project has been designed as a Node.js application.

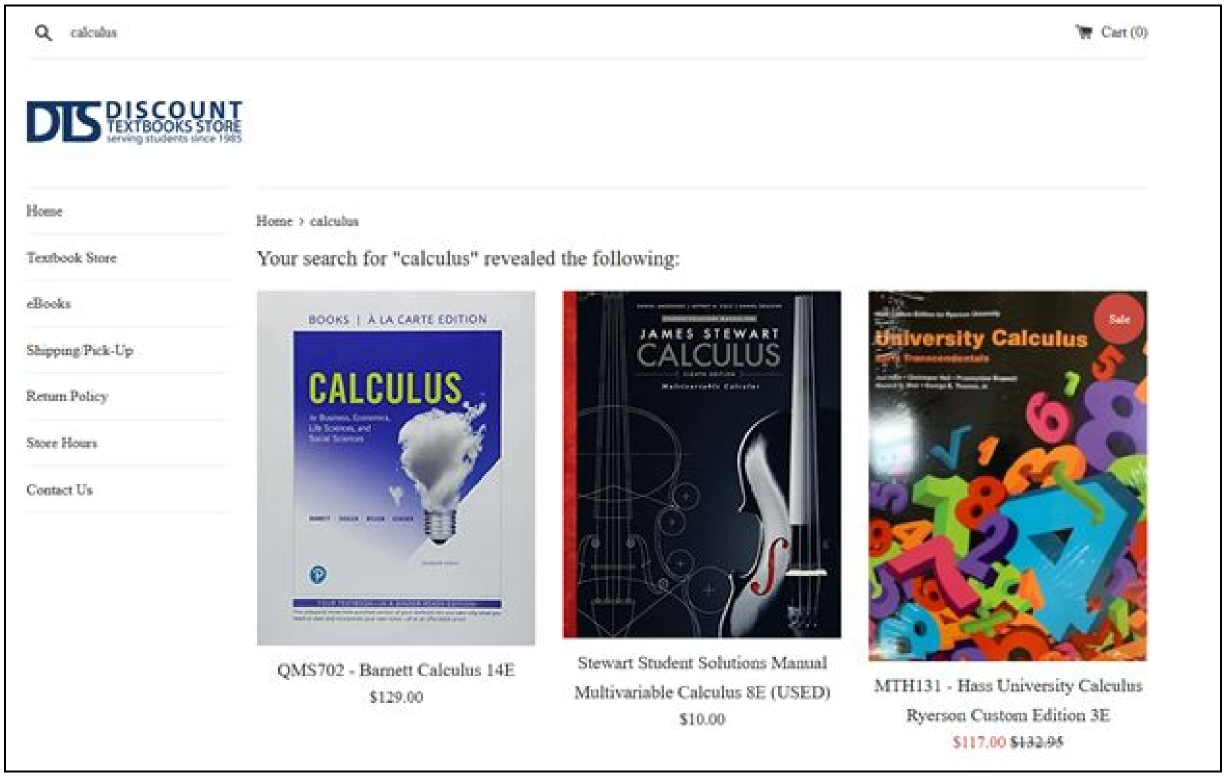
Goals of the Project

The goal of the project was to create an efficient, easy-to-use, and COVID-friendly online bookstore for the purposes of purchasing university textbooks. This was implemented by first logging into the website with an OnTechU email (Gmail, and similar email addresses are not acceptable). In doing so, the website will automatically find the user's university information (via OnTechU's database) and will display various information (name, address, student number, etc.) as well as the courses they have enrolled into. The choice of automatically retrieving a student's information was to increase the user-friendliness of the website as well as to make it more efficient than our competitors.

As stated before, the website complies with COVID-19 health and safety guidelines and strives to deliver the promise of a highly competent experience in spite of it. When the user's textbook is ordered, delivered and ready to be picked up, the website will automatically schedule a date and time for the user to go to the physical bookstore. The date and time will be unique for all users to ensure the least amount of people are gathered inside the bookstore. In addition, any employees at the bookstore will be required to strictly follow all restrictions in place due to the pandemic. With this option, the entire process of purchasing books is organized and streamlined (students do not have to search for their textbooks or wait in lines) as well as minimizing COVID risk during these uncertain times.

Relation to Other Work

Although there are similar websites in this field, we believe our website is superior to our competitors in efficiency, ease of use, and COVID mindfulness. The first difference between websites is the login feature. When a user logs in, they have their data automatically retrieved from OnTechU. They can view what program, year and courses they are in, as well as view the textbooks pertinent to them. This way the user can purchase textbooks much faster and without hassle. This is in comparison with some online textbook stores where there is no login feature (as seen in *Figure 1*) or their login feature does not automatically retrieve any information about the user (*Figure 2*).



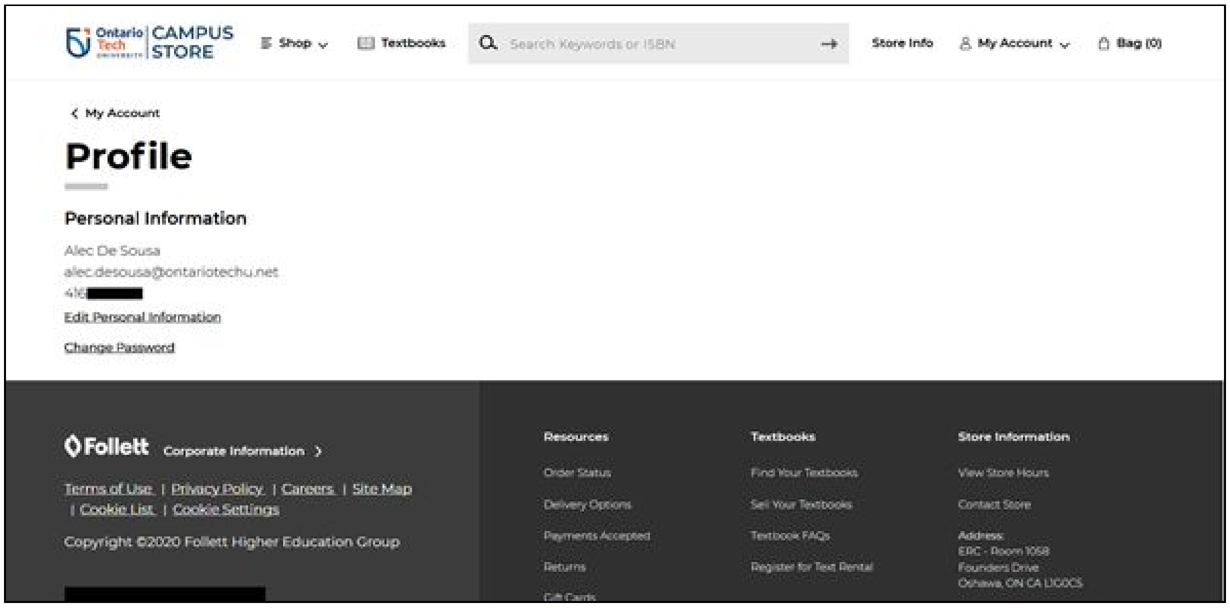
Your search for "calculus" revealed the following:

QMS702 - Barnett Calculus 14E
$129.00

Stewart Student Solutions Manual Multivariable Calculus 8E (USED)
$10.00

MTH131 - Hass University Calculus Ryerson Custom Edition 3E
$117.00 ~~$132.95~~

Figure 1: There is no login feature on this website.



Profile

Personal Information

Alec De Sousa
alec.desousa@ontariotechu.net
416[redacted]
Edit Personal Information
Change Password

Follett Corporate Information

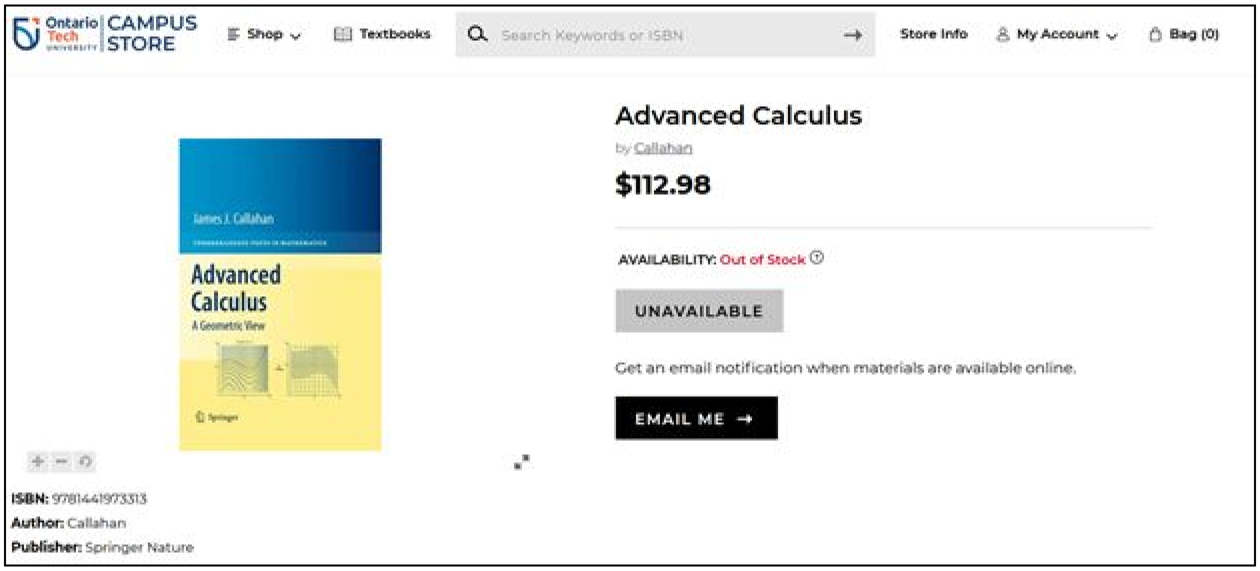
Terms of Use | Privacy Policy | Careers | Site Map | Cookie List | Cookie Settings

Copyright ©2020 Follett Higher Education Group

Resources	Textbooks	Store Information
Order Status	Find Your Textbooks	View Store Hours
Delivery Options	Sell Your Textbooks	Contact Store
Payments Accepted	Textbook FAQs	Address: ERC - Room 1058 Founders Drive Oshawa, ON CA L1G0C5
Returns	Register for Text Rental	
Gift Cards		

Figure 2: The OnTechU bookstore, which does not retrieve any information about the user.

The website also includes a feature that enhances the user's quality of life. On any given textbook page, there will be a section to display which course (or courses, if applicable) the textbook is used for. If the user is currently enrolled in one of these displayed courses, that course name will be bolded for their convenience. This is a simple and effective way of expressing to the user if the textbook is pertinent to them. As seen in *Figure 3*, this feature is not present in our competitors' bookstores.



Advanced Calculus
by Callahan
$112.98

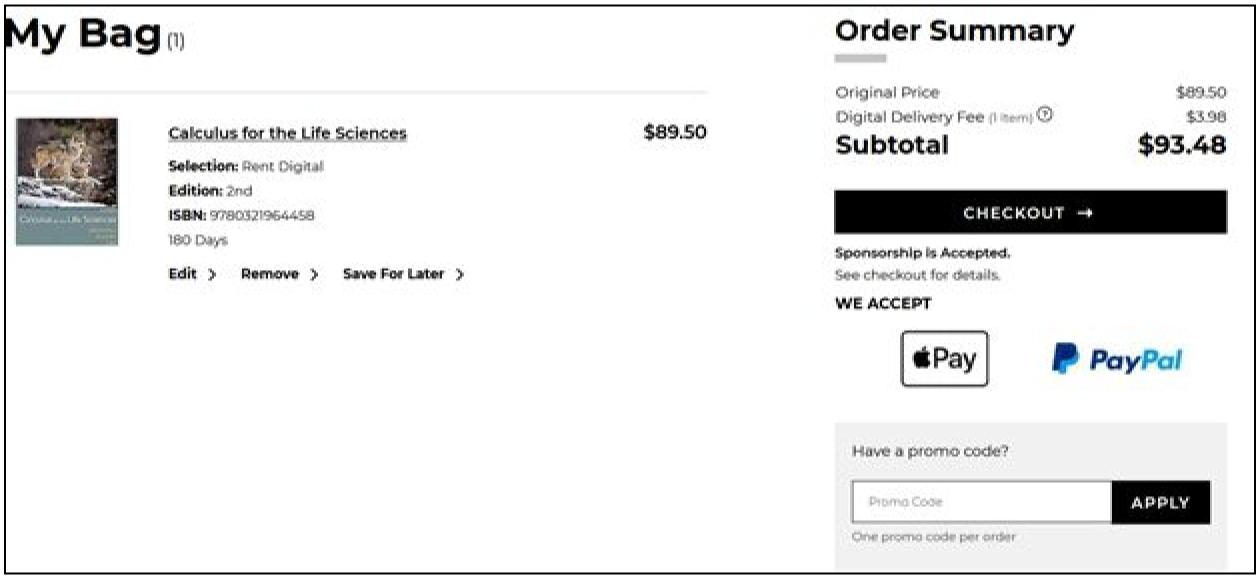
AVAILABILITY: Out of Stock

Get an email notification when materials are available online.

ISBN: 9781441973313
Author: Callahan
Publisher: Springer Nature

Figure 3: The OnTechU bookstore does not display which course(s) this calculus textbook is for.

The payment page of our website also has a unique option of pay when compared to our rivals. As seen in *Figure 4*, the OnTechU bookstore has 2 methods of purchasing the textbook. This is by using Apple Pay or by using PayPal. This is a very restrictive system for users who cannot pay straight away, so we instead opted to add any textbook fees directly into the user's tuition. This is so that the user can decide to pay for textbooks later, if money is currently hard for them to come by. Having tuition and the textbook prices added together also simplifies the process of paying, as the user only needs to pay once**.** From this point, the user must select *In-Store Pickup* as their way of obtaining their desired textbooks, and wait for the textbook to be delivered. The process of *In-Store Pickup* reserves the book for the user so that it can be picked up in-store following COVID guidelines.



My Bag (1)

Calculus for the Life Sciences — $89.50

Selection: Rent Digital
Edition: 2nd
ISBN: 9780321964458
180 Days

Order Summary

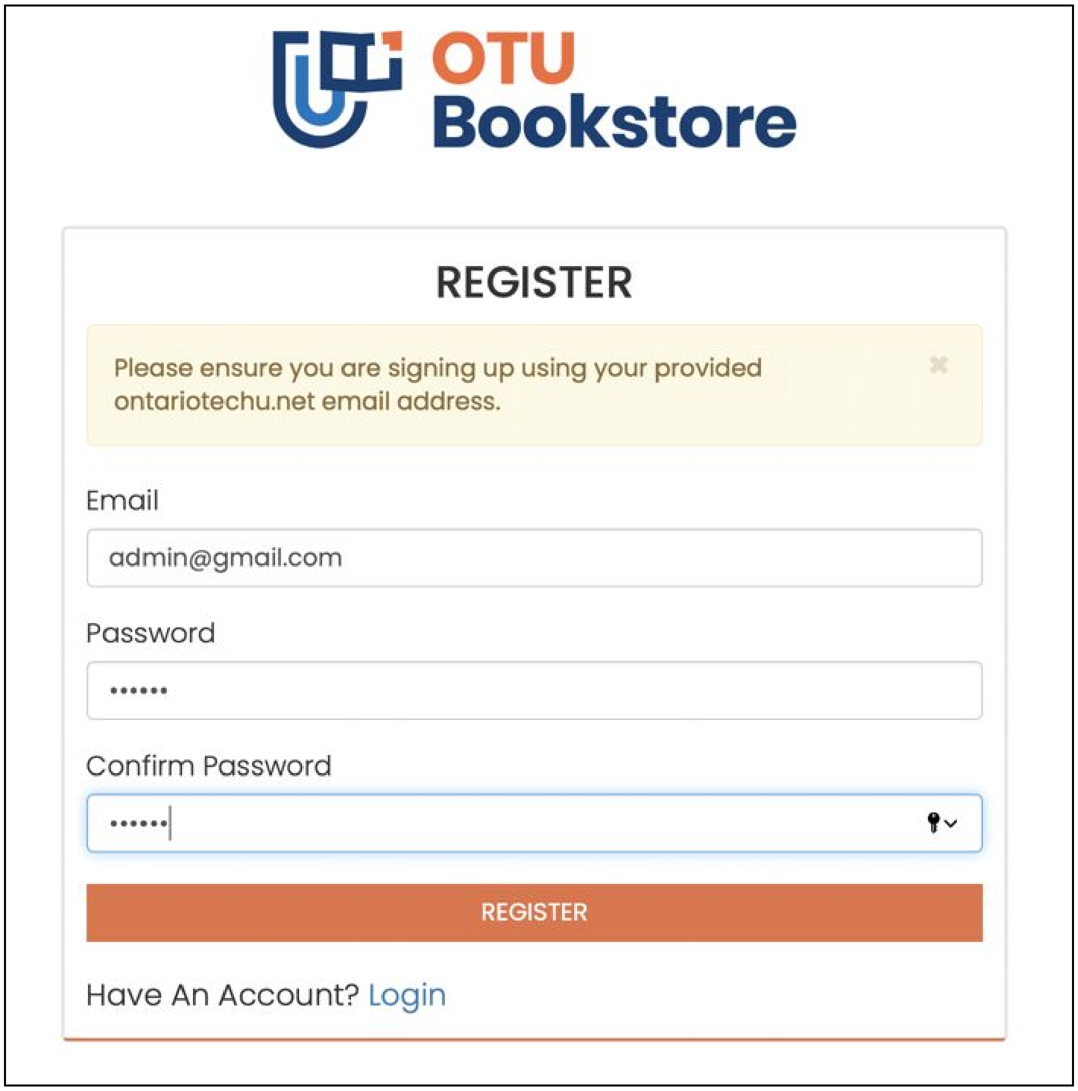
Original Price	$89.50
Digital Delivery Fee (1 item)	$3.98
Subtotal	**$93.48**

Sponsorship Is Accepted.
See checkout for details.

Figure 4: Apple Pay and PayPal are accepted at OnTechU's bookstore.

Main Body of Work

The OTU website created allows users to efficiently search for their course textbooks and order them with minimal effort. Registering for an account is restricted to only ONTU emails and does not allow other email extensions to be used (*Figure 5*). It offers a clean dashboard that contains key information regarding the account. It shows the enrolled courses, active orders, as well as the option to search for textbooks through courses. The bookstore tab shows the list of all available textbooks, their availability, and price. There is also a smart search engine that shows any results containing any of the words in the search bar, even if incomplete (*Figure 6*). The website allows previews of each textbook showing any relevant information such as author, associated courses, price, etc. From there the order process allows users to choose preferable dates for pick up. Once the order has been placed it can easily be cancelled before it is confirmed by an administrator user of the website. After the order is confirmed and completed it is moved to an archived section. The stack implements multiple resources such as Node.js, PassportJS, JavaScript etc. It uses Argon2i hashing for maximum password security, which should be expected of a university integrated website. The application uses pooled database queries in order to return or obtain any information requested. It also uses HTML and CSS for the visualization of the friendly and efficient user interface.



Please ensure you are signing up using your provided ontariotechu.net email address.

Email: admin@gmail.com

Password: ••••••

Confirm Password: ••••••

Figure 5. Shows the registration process and its security for accounts outside of OTU.

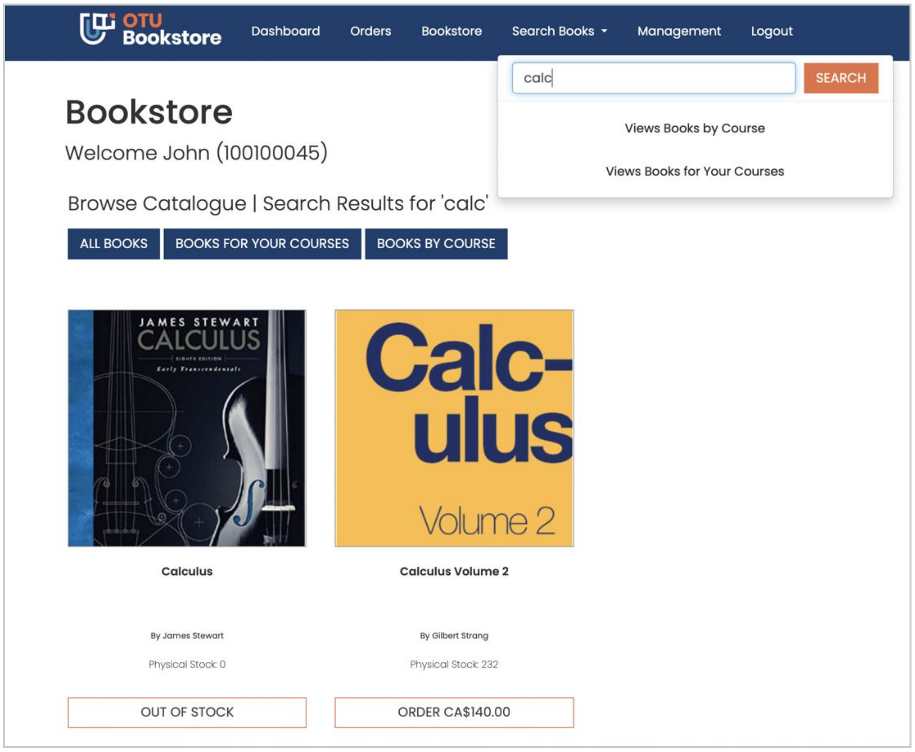


Figure 6. The search engine showing results for any textbook that contains 'calc'.

Implementation

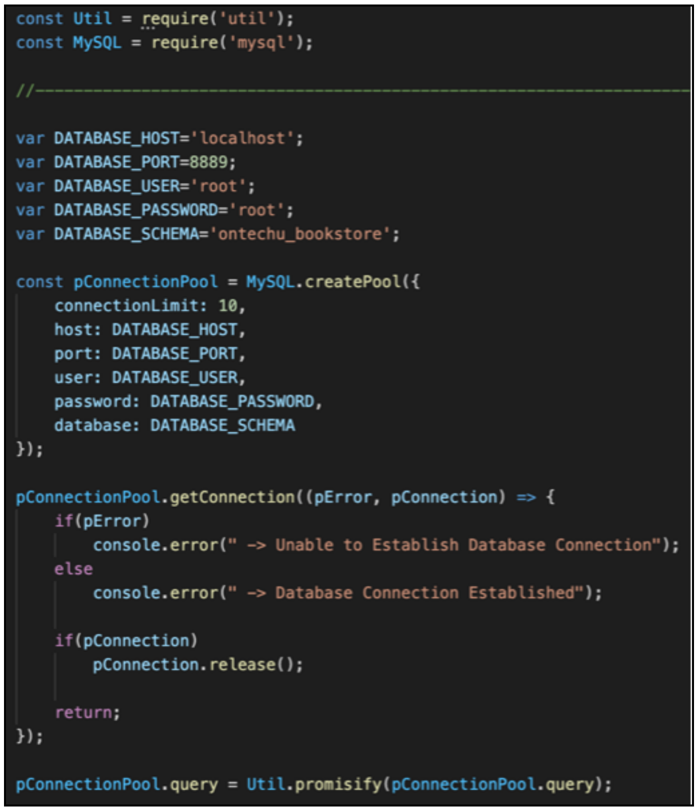
A. Project Development

The OTU Bookstore was developed as a web application that runs on a Node.js server, a very powerful and modern JavaScript runtime environment, allowing the use of a vast variety of libraries that increase the capabilities and ease of expansion for the project. Using Node, we are able to keep all important functions and sensitive information strictly on the server side and serve only the necessary information to the web client through the use of the Router from the Express library.

The Node.js application connects to the MySQL database using a MySQL library that allows us to pool queries to the local database on the server. User registration incorporates Argon2 hashing to protect the integrity of all student passwords, while user logins incorporate local strategy based PassportJS authentications which secures all sessions and pages from unauthorized access and improves the overall security of the application. PassportJS authentication methods allow us to prevent certain Routes from being accessible unless certain conditions are met. In order to improve the quality of life for students revisiting the website user sessions are stored in the database through the use of the Express MySQL Sessions library.

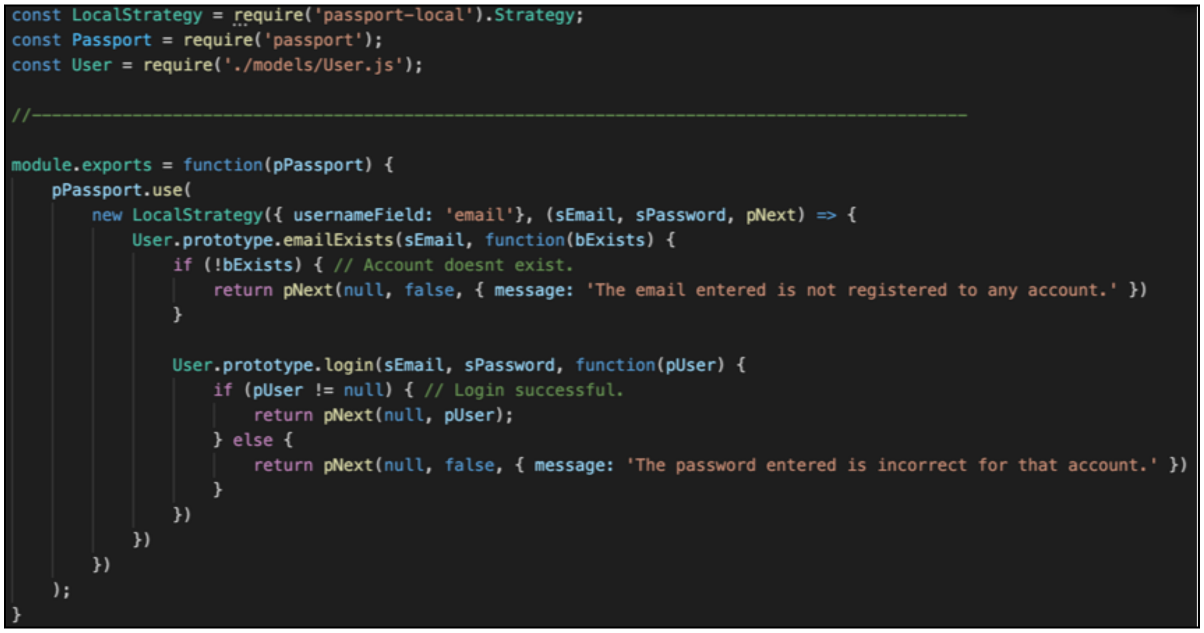
The front end of the web application is built upon the Express EJS Layouts framework library which allows us to build the website visualization as a theme based on partial EJS files. EJS file formatting allows the use of HTML and JavaScript simultaneously within the same line of codes making it a very efficient tool for developing complex systems rather quickly. Ofcourse, the EJS files are then styled through the use of CSS to make the application presentable with an elegant user interface. The theme of the web application was based on OntarioTechU branding and a custom logo and brand was created for the OTU Bookstore.

All of the data displayed on the client web page is obtained from our own integrated API system that returns a JSON object consisting of data such as table headers, table row data, and JavaScript objects. Once the data is retrieved from the server API through the use of a dataset URL, client side JavaScript files read and render the JSON data into viewable HTML elements. Asynchronous methods are also used to refresh data within certain elements without refreshing the page itself.



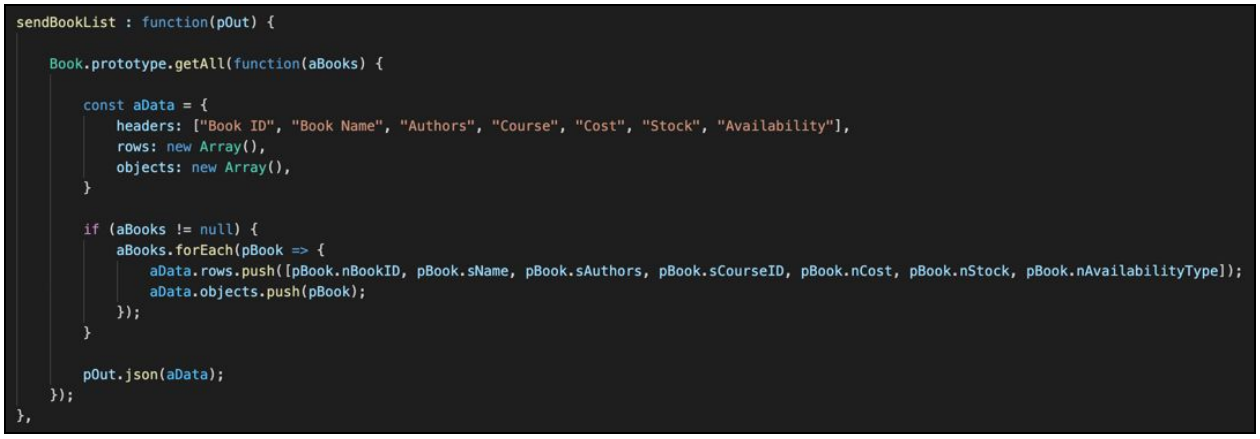
```javascript
const Util = require('util');
const MySQL = require('mysql');

//------------------------------------------------------------------------

var DATABASE_HOST='localhost';
var DATABASE_PORT=8889;
var DATABASE_USER='root';
var DATABASE_PASSWORD='root';
var DATABASE_SCHEMA='ontechu_bookstore';

const pConnectionPool = MySQL.createPool({
    connectionLimit: 10,
    host: DATABASE_HOST,
    port: DATABASE_PORT,
    user: DATABASE_USER,
    password: DATABASE_PASSWORD,
    database: DATABASE_SCHEMA
});

pConnectionPool.getConnection((pError, pConnection) => {
    if(pError)
        console.error(" -> Unable to Establish Database Connection");
    else
        console.error(" -> Database Connection Established");

    if(pConnection)
        pConnection.release();

    return;
});

pConnectionPool.query = Util.promisify(pConnectionPool.query);
```

Figure 7. MySQL database connection in JavaScript.

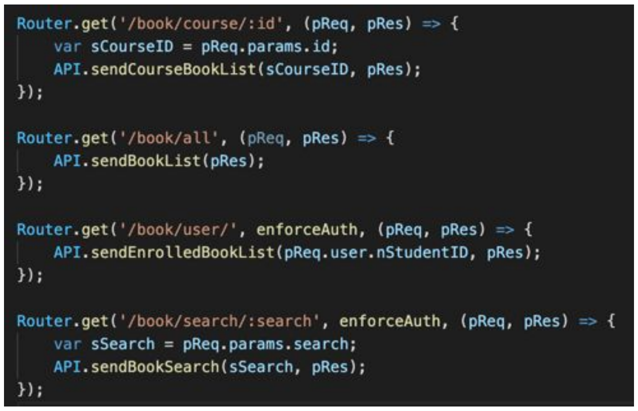


```javascript
const LocalStrategy = require('passport-local').Strategy;
const Passport = require('passport');
const User = require('./models/User.js');

//-----------------------------------------------------------------------------------------

module.exports = function(pPassport) {
    pPassport.use(
        new LocalStrategy({ usernameField: 'email'}, (sEmail, sPassword, pNext) => {
            User.prototype.emailExists(sEmail, function(bExists) {
                if (!bExists) { // Account doesnt exist.
                    return pNext(null, false, { message: 'The email entered is not registered to any account.' })
                }

                User.prototype.login(sEmail, sPassword, function(pUser) {
                    if (pUser != null) { // Login successful.
                        return pNext(null, pUser);
                    } else {
                        return pNext(null, false, { message: 'The password entered is incorrect for that account.' })
                    }
                })
            })
        })
    );
}
```

Figure 8. Sample of user logins using PassportJS authentication.



```javascript
sendBookList : function(pOut) {

    Book.prototype.getAll(function(aBooks) {

        const aData = {
            headers: ["Book ID", "Book Name", "Authors", "Course", "Cost", "Stock", "Availability"],
            rows: new Array(),
            objects: new Array(),
        }

        if (aBooks != null) {
            aBooks.forEach(pBook => {
                aData.rows.push([pBook.nBookID, pBook.sName, pBook.sAuthors, pBook.sCourseID, pBook.nCost, pBook.nStock, pBook.nAvailabilityType]);
                aData.objects.push(pBook);
            });
        }

        pOut.json(aData);
    });
},
```



```javascript
Router.get('/book/course/:id', (pReq, pRes) => {
    var sCourseID = pReq.params.id;
    API.sendCourseBookList(sCourseID, pRes);
});

Router.get('/book/all', (pReq, pRes) => {
    API.sendBookList(pRes);
});

Router.get('/book/user/', enforceAuth, (pReq, pRes) => {
    API.sendEnrolledBookList(pReq.user.nStudentID, pRes);
});

Router.get('/book/search/:search', enforceAuth, (pReq, pRes) => {
    var sSearch = pReq.params.search;
    API.sendBookSearch(sSearch, pRes);
});
```

Figure 9 and 10. Sample of API exporting headers, rows, and JavaScript objects as a JSON object.

B. Code Quality

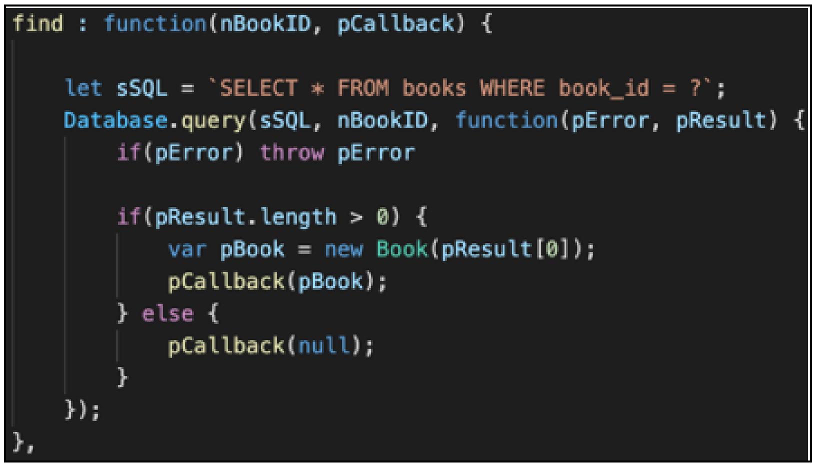
The project was developed with expandability in mind, and the best way to achieve maximum expandability is to develop the system in an object-oriented fashion. In this case, all of the information retrieved from the database tables were handled as User, Course, Book, and Order JavaScript prototype objects. This allowed us to keep all of the code organized, try and minimize coupling where possible, and expand on features of the site rapidly which led to a polished prototype that we are proud to present alongside this report.

The latest and greatest libraries and development tools for Node.js were used extensively throughout the project to ensure a product that resonates quality, performance, security and professionalism. To top things off, all applicable code was written in hungarian notation when possible. Hungarian notation is an old school coding notation that works wonders in object oriented programming as variables are named with prefixes that quickly define their types (p for pointer, s for string, n for number, etc).

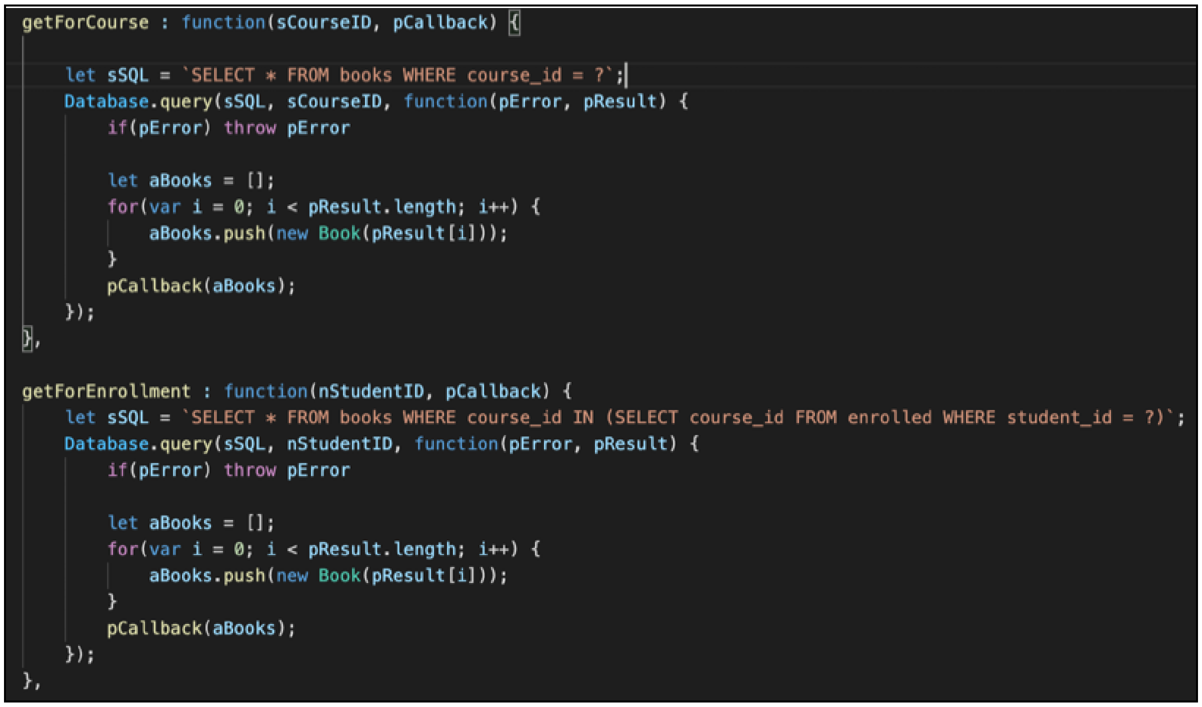
C. Database Queries

In a previous phase of this project, we had created potential views to use within the web application. However, after conducting research on the matter we have found that in professional development environments views are not always preferred as not all developers working on the project may have direct access to the MySQL system to make sufficient changes to the view and other alterations. Therefore to ensure the quick modifiability and expandability of our product, result set views were instead coded as JavaScript MySQL queries that can be modified on demand by any developer working on the project much more easily. The following samples present the queries used to develop the product to the functioning state it is presented in.

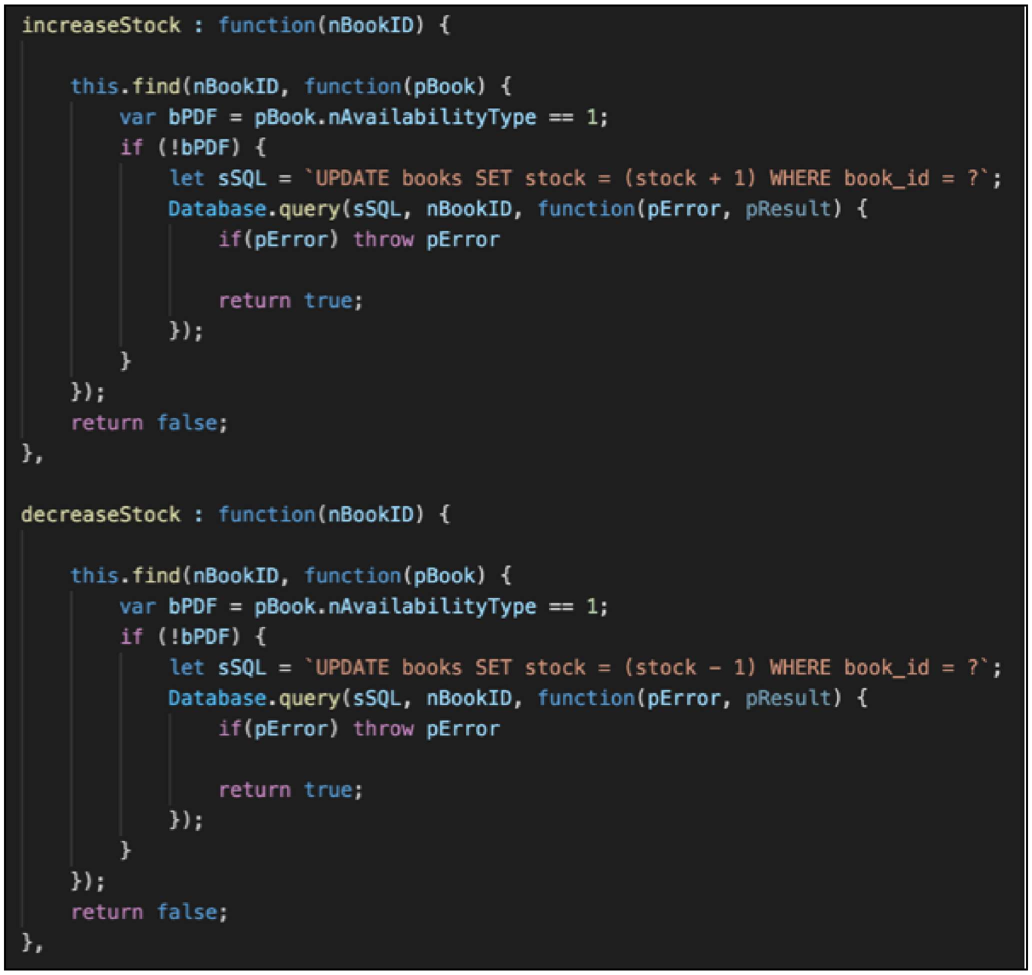
Book Related Queries



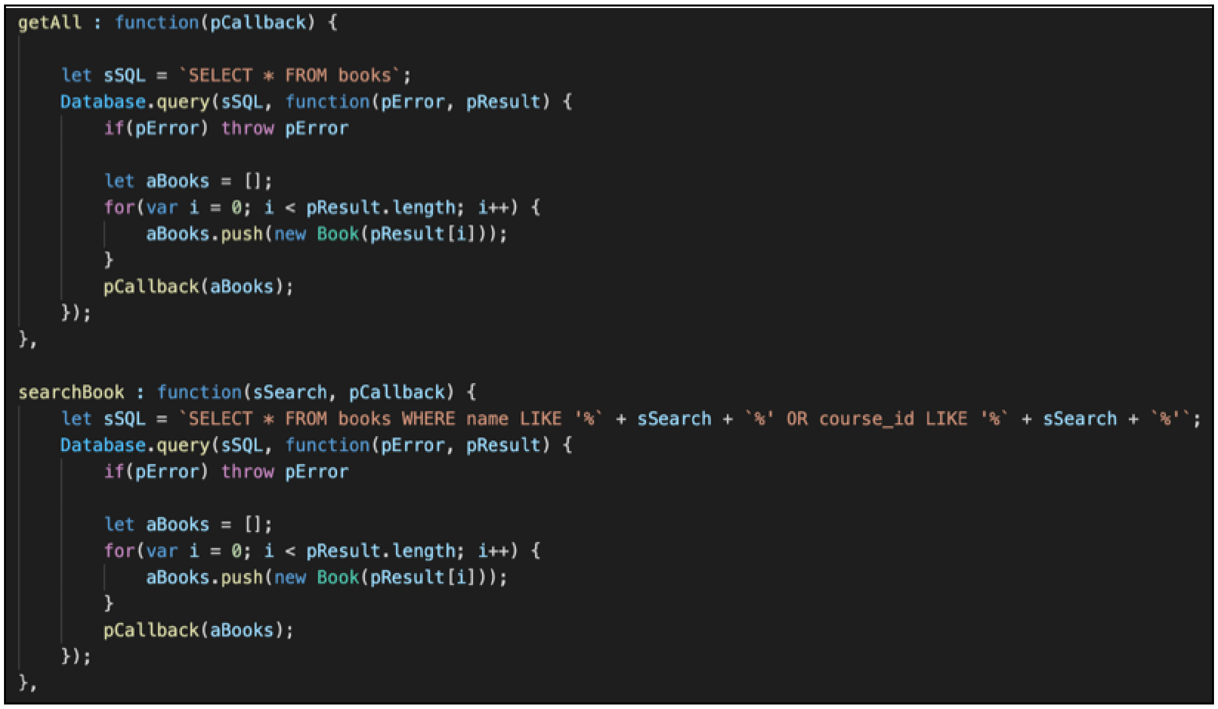
```
find : function(nBookID, pCallback) {

    let sSQL = `SELECT * FROM books WHERE book_id = ?`;
    Database.query(sSQL, nBookID, function(pError, pResult) {
        if(pError) throw pError

        if(pResult.length > 0) {
            var pBook = new Book(pResult[0]);
            pCallback(pBook);
        } else {
            pCallback(null);
        }
    });
},
```



```
getForCourse : function(sCourseID, pCallback) {

    let sSQL = `SELECT * FROM books WHERE course_id = ?`;
    Database.query(sSQL, sCourseID, function(pError, pResult) {
        if(pError) throw pError

        let aBooks = [];
        for(var i = 0; i < pResult.length; i++) {
            aBooks.push(new Book(pResult[i]));
        }
        pCallback(aBooks);
    });
},

getForEnrollment : function(nStudentID, pCallback) {
    let sSQL = `SELECT * FROM books WHERE course_id IN (SELECT course_id FROM enrolled WHERE student_id = ?)`;
    Database.query(sSQL, nStudentID, function(pError, pResult) {
        if(pError) throw pError

        let aBooks = [];
        for(var i = 0; i < pResult.length; i++) {
            aBooks.push(new Book(pResult[i]));
        }
        pCallback(aBooks);
    });
},
```

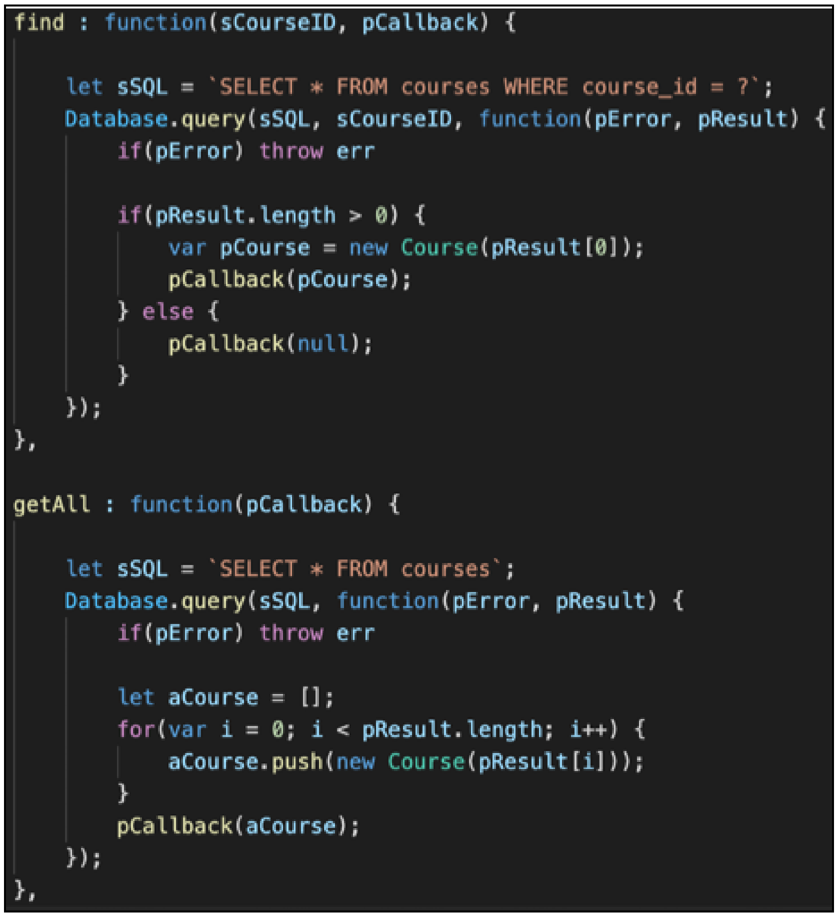


```
increaseStock : function(nBookID) {

    this.find(nBookID, function(pBook) {
        var bPDF = pBook.nAvailabilityType == 1;
        if (!bPDF) {
            let sSQL = `UPDATE books SET stock = (stock + 1) WHERE book_id = ?`;
            Database.query(sSQL, nBookID, function(pError, pResult) {
                if(pError) throw pError

                return true;
            });
        }
    });
    return false;
},

decreaseStock : function(nBookID) {

    this.find(nBookID, function(pBook) {
        var bPDF = pBook.nAvailabilityType == 1;
        if (!bPDF) {
            let sSQL = `UPDATE books SET stock = (stock - 1) WHERE book_id = ?`;
            Database.query(sSQL, nBookID, function(pError, pResult) {
                if(pError) throw pError

                return true;
            });
        }
    });
    return false;
},
```

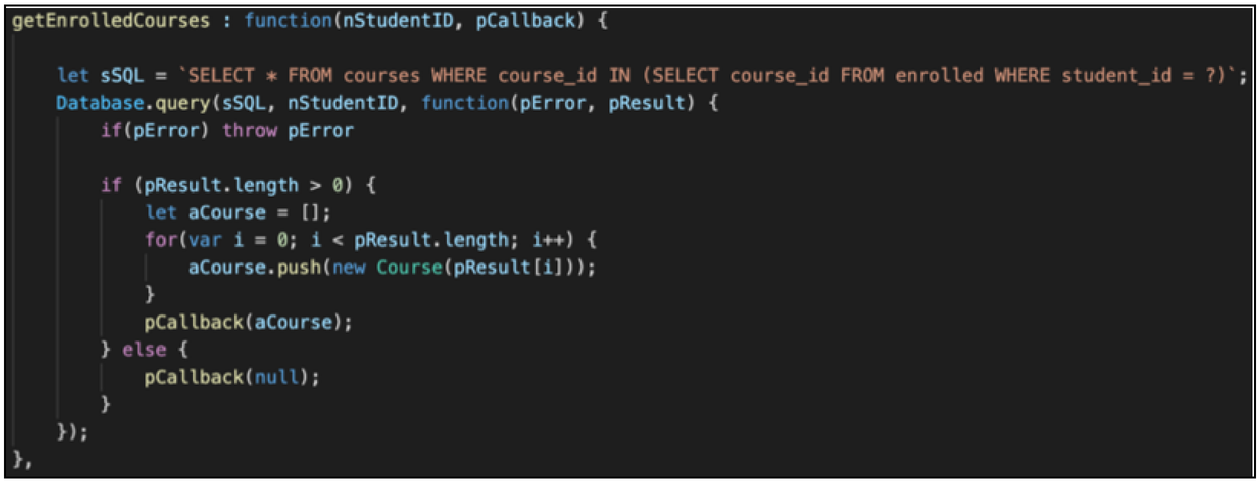


```
getAll : function(pCallback) {

    let sSQL = `SELECT * FROM books`;
    Database.query(sSQL, function(pError, pResult) {
        if(pError) throw pError

        let aBooks = [];
        for(var i = 0; i < pResult.length; i++) {
            aBooks.push(new Book(pResult[i]));
        }
        pCallback(aBooks);
    });
},

searchBook : function(sSearch, pCallback) {
    let sSQL = `SELECT * FROM books WHERE name LIKE '%` + sSearch + `%' OR course_id LIKE '%` + sSearch + `%'`;
    Database.query(sSQL, function(pError, pResult) {
        if(pError) throw pError

        let aBooks = [];
        for(var i = 0; i < pResult.length; i++) {
            aBooks.push(new Book(pResult[i]));
        }
        pCallback(aBooks);
    });
},
```

Course Related Queries

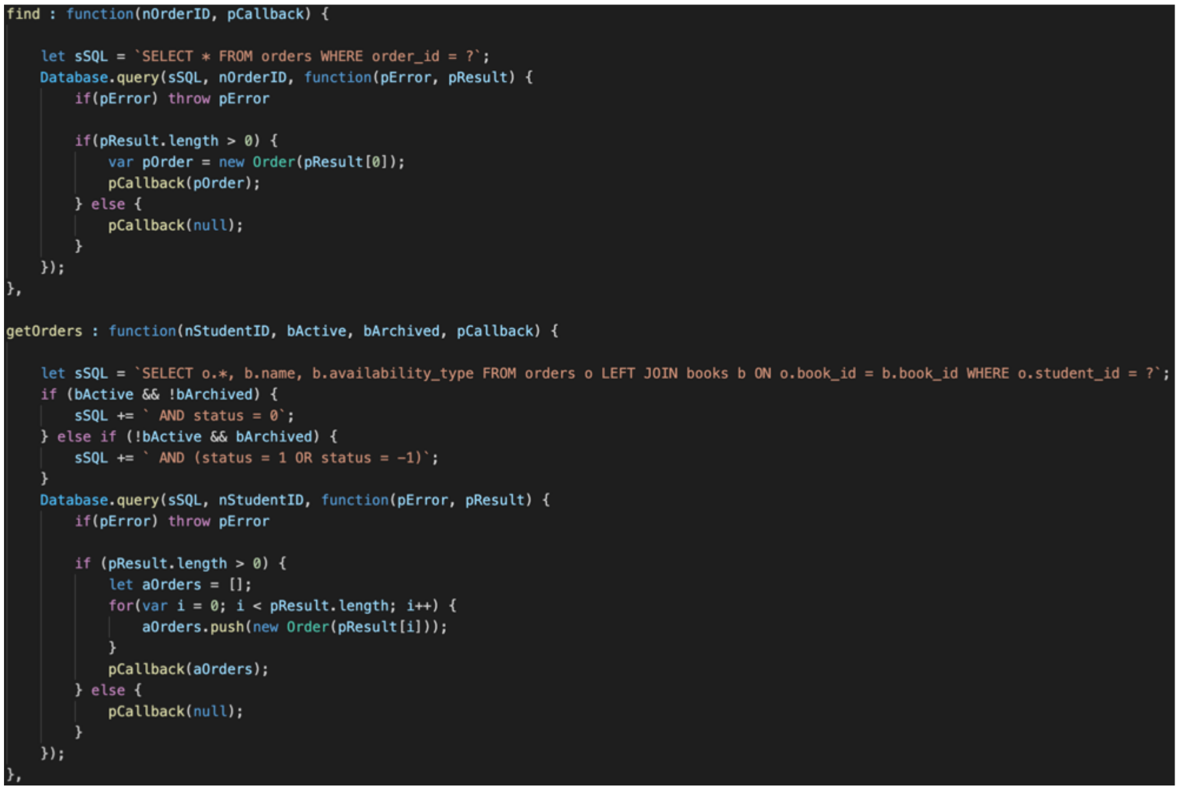


```
find : function(sCourseID, pCallback) {

    let sSQL = `SELECT * FROM courses WHERE course_id = ?`;
    Database.query(sSQL, sCourseID, function(pError, pResult) {
        if(pError) throw err

        if(pResult.length > 0) {
            var pCourse = new Course(pResult[0]);
            pCallback(pCourse);
        } else {
            pCallback(null);
        }
    });
},

getAll : function(pCallback) {

    let sSQL = `SELECT * FROM courses`;
    Database.query(sSQL, function(pError, pResult) {
        if(pError) throw err

        let aCourse = [];
        for(var i = 0; i < pResult.length; i++) {
            aCourse.push(new Course(pResult[i]));
        }
        pCallback(aCourse);
    });
},
```

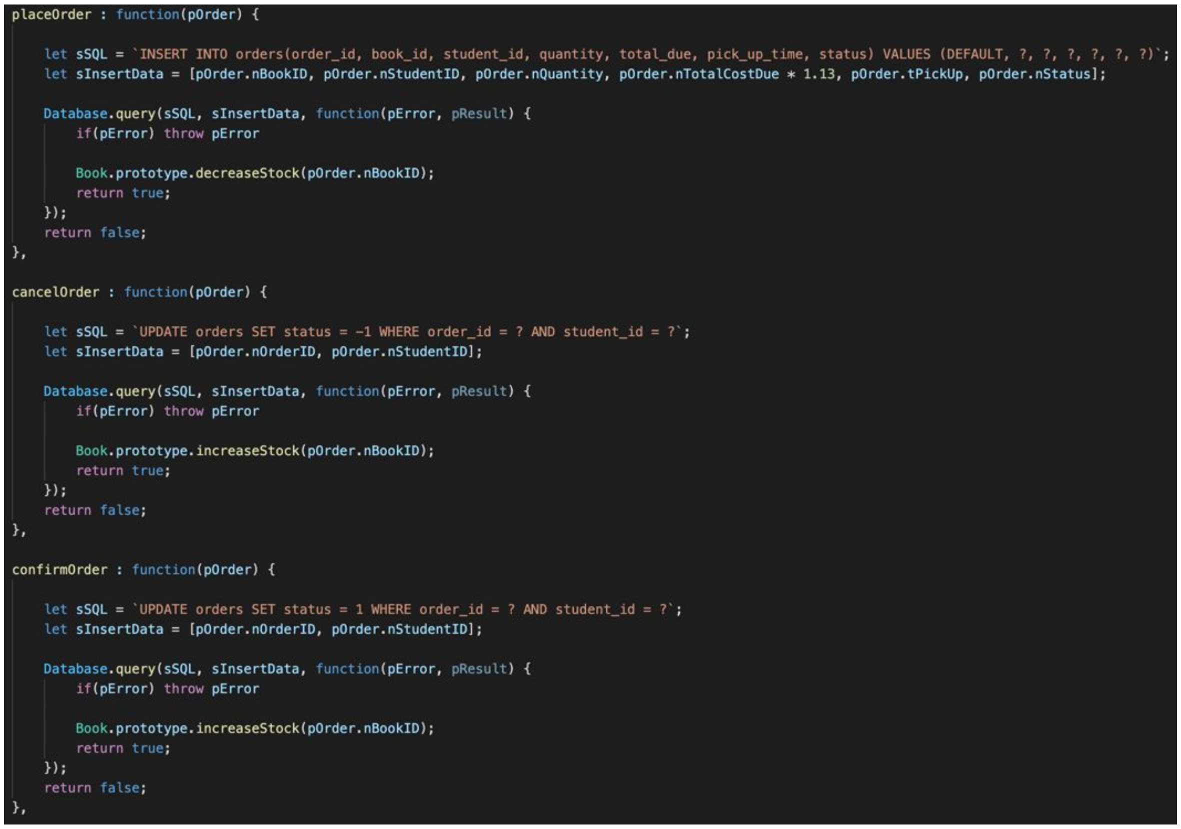


```
getEnrolledCourses : function(nStudentID, pCallback) {

    let sSQL = `SELECT * FROM courses WHERE course_id IN (SELECT course_id FROM enrolled WHERE student_id = ?)`;
    Database.query(sSQL, nStudentID, function(pError, pResult) {
        if(pError) throw pError

        if (pResult.length > 0) {
            let aCourse = [];
            for(var i = 0; i < pResult.length; i++) {
                aCourse.push(new Course(pResult[i]));
            }
            pCallback(aCourse);
        } else {
            pCallback(null);
        }
    });
},
```

Order Related Queries

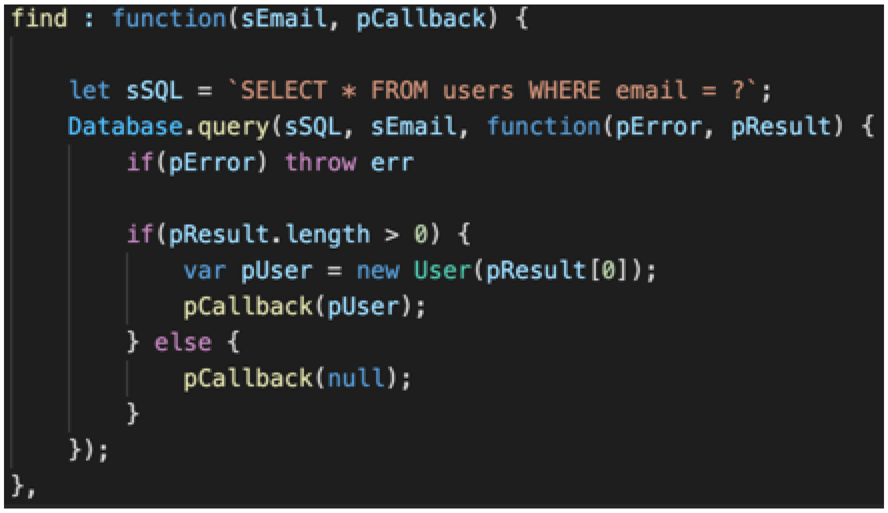


```
find : function(nOrderID, pCallback) {

    let sSQL = `SELECT * FROM orders WHERE order_id = ?`;
    Database.query(sSQL, nOrderID, function(pError, pResult) {
        if(pError) throw pError

        if(pResult.length > 0) {
            var pOrder = new Order(pResult[0]);
            pCallback(pOrder);
        } else {
            pCallback(null);
        }
    });
},

getOrders : function(nStudentID, bActive, bArchived, pCallback) {

    let sSQL = `SELECT o.*, b.name, b.availability_type FROM orders o LEFT JOIN books b ON o.book_id = b.book_id WHERE o.student_id = ?`;
    if (bActive && !bArchived) {
        sSQL += ` AND status = 0`;
    } else if (!bActive && bArchived) {
        sSQL += ` AND (status = 1 OR status = -1)`;
    }
    Database.query(sSQL, nStudentID, function(pError, pResult) {
        if(pError) throw pError

        if (pResult.length > 0) {
            let aOrders = [];
            for(var i = 0; i < pResult.length; i++) {
                aOrders.push(new Order(pResult[i]));
            }
            pCallback(aOrders);
        } else {
            pCallback(null);
        }
    });
},
```

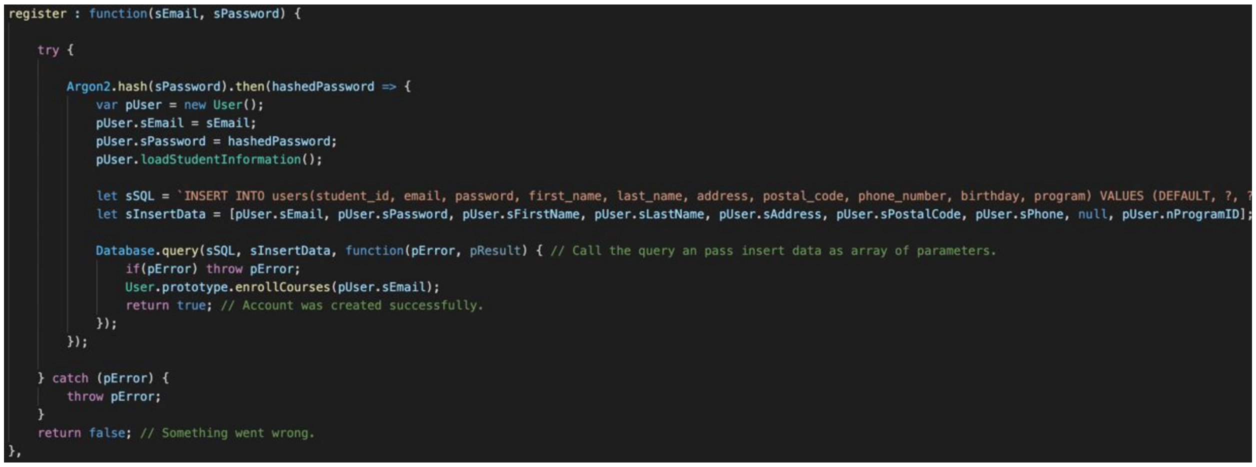


```
placeOrder : function(pOrder) {

    let sSQL = `INSERT INTO orders(order_id, book_id, student_id, quantity, total_due, pick_up_time, status) VALUES (DEFAULT, ?, ?, ?, ?, ?, ?)`;
    let sInsertData = [pOrder.nBookID, pOrder.nStudentID, pOrder.nQuantity, pOrder.nTotalCostDue * 1.13, pOrder.tPickUp, pOrder.nStatus];

    Database.query(sSQL, sInsertData, function(pError, pResult) {
        if(pError) throw pError

        Book.prototype.decreaseStock(pOrder.nBookID);
        return true;
    });
    return false;
},

cancelOrder : function(pOrder) {

    let sSQL = `UPDATE orders SET status = -1 WHERE order_id = ? AND student_id = ?`;
    let sInsertData = [pOrder.nOrderID, pOrder.nStudentID];

    Database.query(sSQL, sInsertData, function(pError, pResult) {
        if(pError) throw pError

        Book.prototype.increaseStock(pOrder.nBookID);
        return true;
    });
    return false;
},

confirmOrder : function(pOrder) {

    let sSQL = `UPDATE orders SET status = 1 WHERE order_id = ? AND student_id = ?`;
    let sInsertData = [pOrder.nOrderID, pOrder.nStudentID];

    Database.query(sSQL, sInsertData, function(pError, pResult) {
        if(pError) throw pError

        Book.prototype.increaseStock(pOrder.nBookID);
        return true;
    });
    return false;
},
```

User Related Queries

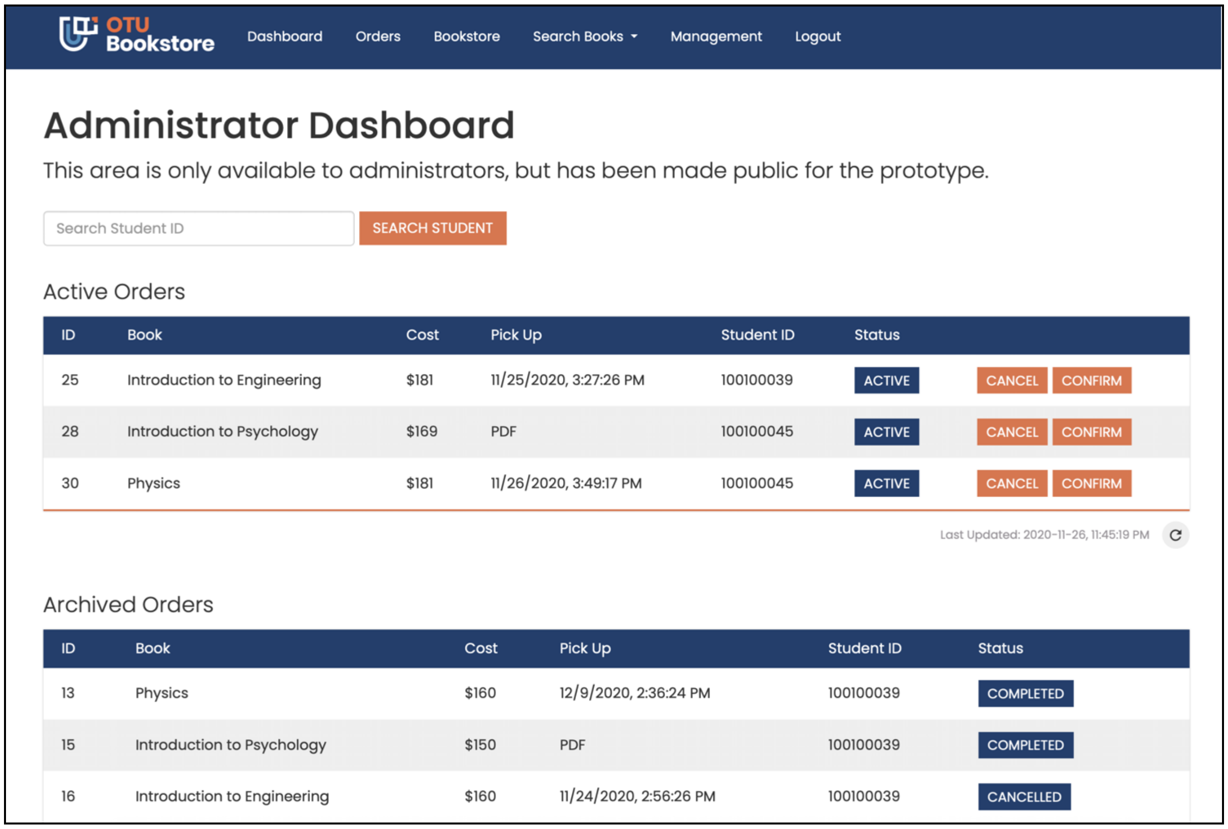


```
find : function(sEmail, pCallback) {

    let sSQL = `SELECT * FROM users WHERE email = ?`;
    Database.query(sSQL, sEmail, function(pError, pResult) {
        if(pError) throw err

        if(pResult.length > 0) {
            var pUser = new User(pResult[0]);
            pCallback(pUser);
        } else {
            pCallback(null);
        }
    });
},
```



```
register : function(sEmail, sPassword) {

    try {

        Argon2.hash(sPassword).then(hashedPassword => {
            var pUser = new User();
            pUser.sEmail = sEmail;
            pUser.sPassword = hashedPassword;
            pUser.loadStudentInformation();

            let sSQL = `INSERT INTO users(student_id, email, password, first_name, last_name, address, postal_code, phone_number, birthday, program) VALUES (DEFAULT, ?, ?
            let sInsertData = [pUser.sEmail, pUser.sPassword, pUser.sFirstName, pUser.sLastName, pUser.sAddress, pUser.sPostalCode, pUser.sPhone, null, pUser.nProgramID];

            Database.query(sSQL, sInsertData, function(pError, pResult) { // Call the query an pass insert data as array of parameters.
                if(pError) throw pError;
                User.prototype.enrollCourses(pUser.sEmail);
                return true; // Account was created successfully.
            });
        });

    } catch (pError) {
        throw pError;
    }
    return false; // Something went wrong.
},
```

Demonstration of Views on Web App



Administrator Dashboard

This area is only available to administrators, but has been made public for the prototype.

Active Orders

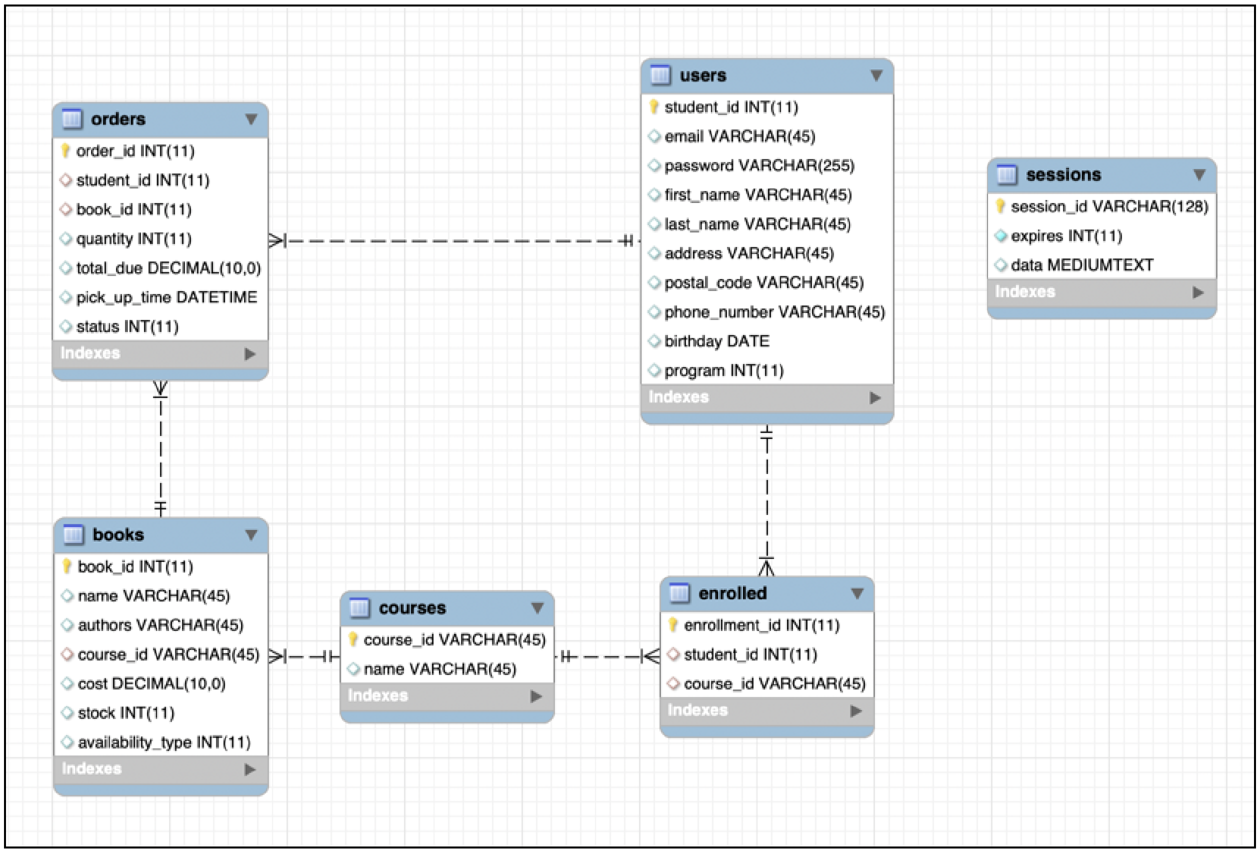
ID	Book	Cost	Pick Up	Student ID	Status	
25	Introduction to Engineering	$181	11/25/2020, 3:27:26 PM	100100039	ACTIVE	CANCEL CONFIRM
28	Introduction to Psychology	$169	PDF	100100045	ACTIVE	CANCEL CONFIRM
30	Physics	$181	11/26/2020, 3:49:17 PM	100100045	ACTIVE	CANCEL CONFIRM

Last Updated: 2020-11-26, 11:45:19 PM

Archived Orders

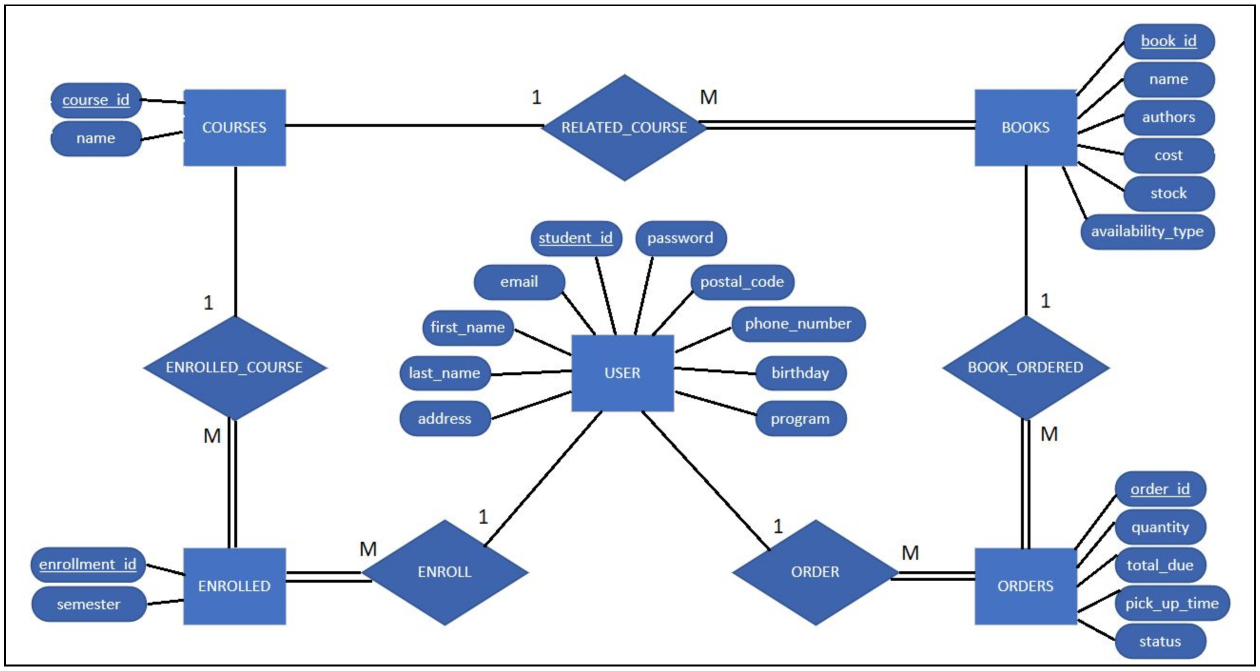
ID	Book	Cost	Pick Up	Student ID	Status
13	Physics	$160	12/9/2020, 2:36:24 PM	100100039	COMPLETED
15	Introduction to Psychology	$150	PDF	100100039	COMPLETED
16	Introduction to Engineering	$160	11/24/2020, 2:56:26 PM	100100039	CANCELLED

Schematics



The database built for this project consists of 6 tables: users, courses, enrolled, books, orders, and sessions. The first five tables listed are a part of the system core functionality, while the sessions table is automatically generated by the Node.js library known as Express MySQL Sessions which uses a MySQL connection to store authenticated login sessions with cookies and other related information.

Design Diagram



The entity-relationship diagram above represents all of the relationships between the database tables and their columns. The relationships within this project track relevant data for the bookstore system such as which users are signed up, their enrollment information as well as all courses in the system, which is all linked to the textbooks and orders stored in the database.

Project Relation to Course

Over the length of this course we are exposed to many forms of data management. We learned about machine language, and many types of storage such as RAM, registers, cache, etc. We also learned about php coding and working with servers. This project forces students to use the knowledge they've accumulated over the semester. Specifically in our project, we use hashing for our database to prevent any collisions between accounts. The database contains the information of all users and uses cookies to provide users with quick access to the website. It uses high end encryption for maximum security, which should be expected of a website that contains valuable information belonging to students. The OTU Bookstore website uses all of the things we have learned in this course and goes above the expectations for this project.

Results and Analysis

Our website for the project is above the expectations of the course while following COVID-19 safety protocols and employing the use of many real world applications/utilities, such as a planned connection to the university API for retrieving data and automatically handling textbook cost additions to student tuition. By utilizing all libraries included in the stack in a creative and effective manner, the website is able to operate at a very high standard. By using Argon2 hashing, students can be ensured their passwords are guarded with a very respectable level of security. It uses cookies to retain account related information to allow quick and easy access to the website. It rejects any email not affiliated with Ontario Tech University. It fulfils all the requirements at a very high level and is capable of being used as a legitimate website. The outcome of the website is a great success and provides results that are greater than expected. The website not only performs more efficiently than currently operating university bookstores, but is designed to be integrated as a local solution rather than third-party thus ensuring the privacy and security of student and order information.

Thoughts About Future Work

In the future we hope that we can implement our bookstore into many other universities, as well as add more administrative and management features. We would also like to add an option where the student can pay directly via credit card or PayPal. As well as some quality improvements such as adding book descriptions, or allowing the book PDFs to be viewed directly on the site once the book has been purchased.

Conclusion

As shown, we have designed a bookstore that is easily accessible to all users while taking the COVID-19 pandemic into heavy consideration when designing the project. For us to adequately accommodate both students and faculty and follow COVID-19 health and safety guidelines, we took the necessary steps such as setting up a safe contact-limited in-store pickup and online PDF delivery system. For in-store pickup, we also ensured that each customer receives a particular time not to pick up more than one student. The website was built upon a powerful stack and boasts a layout specifically designed to provide a seamless interaction between the student and the website; the students can log in using their university assigned email address and filter the textbooks to suit their needs.

References

Figure 1: https://www.discounttextbookstoronto.com/

Figure 2, 3, and 4: https://www.bkstr.com/ontariotechstore/home