

Revised MizzouCheckout System  
University of Missouri  
CS3380: Database Applications and Information Systems  
Hunter Ginther, Jakob Daugherty, Zach Dolan, Kevin Free, Michael McLaughlin, and  
Alyssa Nielsen (Team Lead)  
13<sup>th</sup> May 2016

## Introduction

Mizzou Student Unions (“The Client”) has asked us to help re-work their current inventory management system to improve usability and efficiency. Kevin Free has created an inventory page for displaying all of the items, using radio buttons to filter the inventory. For the back-end, he utilizes PHP to run similar queries in a prepared statement format and display the results. One could sort the inventory by damaged, checked in, checked out, and by the different item categories, including bikes, Macs, and PCs. From this page, you are also able to update the condition of any particular item. It will take you to the edit page, only accessible by an admin user through the use of sessions, where you can change the condition to any of the options from the `item_condition` table. A drop down menu is populated with the values for every condition. Whatever the user selects is then put into the prepared statement and an update query is performed. We decided not to allow the ability to edit critical information about items and other tables because that shouldn’t be necessary once the items are in the system correctly. Kevin also created all the insert functionality for adding items and item information. This page has a drop down menu including item, item condition, waiver, item category, and location to choose where you want to insert a new entry into. Once selected, the insert page includes different PHP files to allow for the corresponding table to be added to. This operates very similar to the Spring 2016 CS 3380 course’s Lab 6. The admin user fills out the fields and the information is thrust into a prepared statement and run on the SQL server. Proper sessions and HTTPS requirements are met on all pages and it should be fairly unbreakable due to the error checking he included. These pages should meet all the requirements for adding, listing, and editing entries in the tables across the database.

Hunter Ginther has designed and created the custom database and handled inventory and home page operations, including a system of displaying items on the home page based on various item attributes, including whether they are checked out, overdue, or available. He also created a system of adding, removing, and updating items based on the different desk locations chosen by the user via a dropdown in the navigation bar on the home page, allowing for easy and efficient view transitions between desired desk locations. Using the entity relationship design given in class, he turned it into a fully operational MySQL database based on the group’s decisions for how to tweak it to fit out needs, with flexibility and durability. Using his knowledge of database operations, he was key in producing a viable inventory table. The home page is complete with color codes to easily identify attributes like availability and overdue, so that the employee checking in and out items can see at a glance what the status is of each item. Finally, he helped with traffic flow and page redirection throughout the application to ease user access to the system.

Zach Dolan started the design of the website from scratch using bootstrap and implemented an easy to use user interface. He tried to keep it pretty similar to the design and functionality of the current system to avoid a learning curve when the new system is implemented, although still changing some aspects to make the user interface easier and more intuitive to use. He also created the `style.css` file for styling the elements and keeping them the same for an even flow across pages, favoring centered content wrapped in a single `div` because it provides an easy transition to mobile devices if the client starts to use tablets in addition to, or instead of, laptops. Minor changes were made to the UI as we progressed further in the functionality, as to be sure that all of the functionality could be implemented in a way that makes sense to the user. Additionally, he worked with the authentication

system, including limiting the information being displayed to outside users. He created the log out system, which takes the user to `logout.php`, ends their session, destroys variables, and redirects them to the home page. Finally, he also saw the need to clean up the search page for the user, adding a few methods of searching for a student including by ID, pawprint, or last name. He also added an option to search directly for the item ID itself to verify who has the item checked out. The queries are completed using switch statements based on which radio is selected, which are a lot cleaner and easier to read in the code than several if statements. The queries select the student's relevant information and populate the table along with the ID of the item that matches the ID of the student.

Jakob Daugherty hosted the project on his virtual machine, helped write the project proposal and final documentation, and performed database analytics and administration. He created a GitHub repository to handle version merging and publishing stability, and he also created test data for beta testing on site functionality. Additionally, he compiled all of the information regarding each team member's contributions and facts about the application that allowed him to draft the final report for the project. In order to draft the final documentation for the application, he also analyzed the application database normal form, added indexes and reworked primary key attributes for more efficient database functionality, as well. Finally, he reviewed or revised all queries for accuracy and security and added additional search functionality to the inventory page.

Alyssa Nielsen was the team lead, and she coordinated group meetings, submitted group project assignments and information, and kept track of deadlines. She produced the project presentation, helped write and finalize the project proposal, and revised and finalized the project documentation. In addition, she conducted database planning and composed the group ERD ideas into one submittal format. She also helped plan and construct the application user interface implementation, and did revision, organization, and formatting of all code for the application. She helped implement the login functionality, checking login credentials and setting session variables, and implemented the registration page functionality for adding new users, not allowing for duplicate usernames or student IDs. Finally, she performed most application testing for functionality, debugging, and resolved outstanding issues.

Michael McLaughlin created the check in and check out item UI and implemented the functionality on the page. He created the page to check in or out an item with a single button using item and student ID values. This page only requires the item ID to check that item back in and, if an item is available, only requires the additional student ID to check the item out. These values are sent to the database with time stamps to be inserted into the `student_item_transaction` table. He also used the time stamps to allow the system to keep track of overdue items.

### **The MIT License(MIT)**

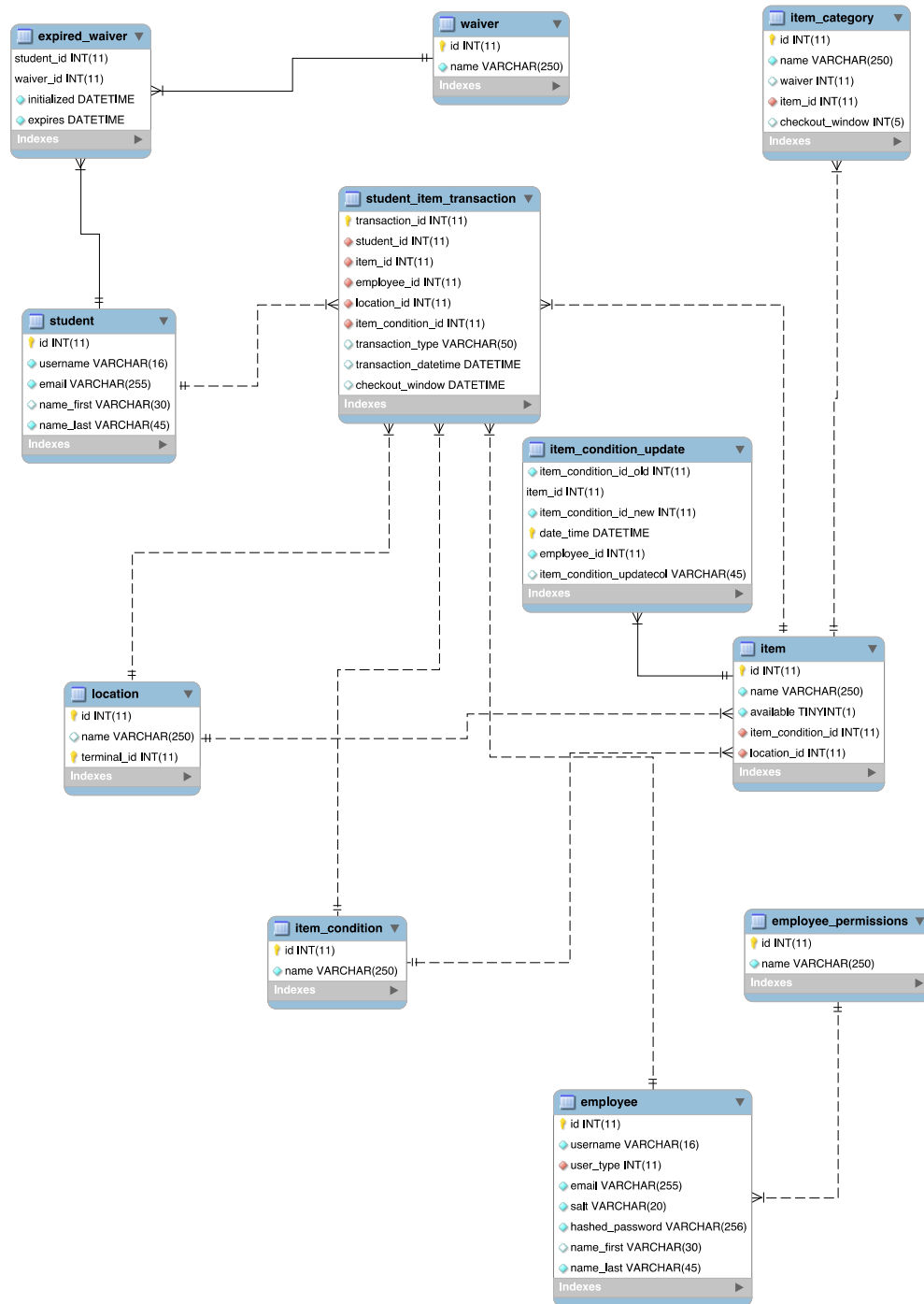
This application is licensed under the MIT license, detailed as follows.

Copyright (c) 2016 Hunter Ginther, Jakob Daugherty, Zach Dolan, Kevin Free, Michael McLaughlin, and Alyssa Nielsen

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## ERD



## Queries

All queries used by our system are detailed below. We have provided the SQL syntax and a detailed description of the expected return of all queries.

- SQL (1): `SELECT i.id AS `Item ID`, i.name AS `Item Name`, (SELECT sit.student_id FROM student_item_transaction AS sit WHERE sit.item_id = i.id AND sit.transaction_datetime >= CURDATE() AND sit.transaction_type = 'Out' AND sit.checkout_window = (SELECT MAX(sit.checkout_window) FROM student_item_transaction WHERE item_id = sit.item_id)) AS `Student`, (SELECT sit.employee_id FROM student_item_transaction AS sit WHERE sit.item_id = i.id AND sit.transaction_datetime >= CURDATE() AND sit.transaction_type = 'Out' AND sit.checkout_window = (SELECT MAX(sit.checkout_window) FROM student_item_transaction WHERE item_id = sit.item_id)) AS `Employee`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location`, (SELECT sit.checkout_window FROM student_item_transaction AS sit WHERE sit.item_id = i.id AND sit.transaction_datetime >= CURDATE() AND sit.transaction_type = 'Out' AND sit.checkout_window = (SELECT MAX(sit.checkout_window) FROM student_item_transaction WHERE item_id = sit.item_id)) AS `Time Due Back` FROM item AS i, item_condition AS ic, location AS l WHERE i.item_condition_id = ic.id AND i.location_id = l.id AND i.location_id = 1 ORDER BY `Availability`, `Time Due Back` ASC, i.id;`
  - Return: All attributes for items located at the Memorial Student Union desk ordered by availability
- SQL (1.1): `SELECT i.id AS `Item ID`, i.name AS `Item Name`, (SELECT sit.student_id FROM student_item_transaction AS sit WHERE sit.item_id = i.id AND sit.transaction_datetime >= CURDATE() AND sit.transaction_type = 'Out' AND sit.checkout_window = (SELECT MAX(sit.checkout_window) FROM student_item_transaction WHERE item_id = sit.item_id)) AS `Student`, (SELECT sit.employee_id FROM student_item_transaction AS sit WHERE sit.item_id = i.id AND sit.transaction_datetime >= CURDATE() AND sit.transaction_type = 'Out' AND sit.checkout_window = (SELECT MAX(sit.checkout_window) FROM student_item_transaction WHERE item_id = sit.item_id)) AS `Employee`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location`, (SELECT sit.checkout_window FROM student_item_transaction AS sit WHERE sit.item_id = i.id AND sit.transaction_datetime >= CURDATE() AND sit.transaction_type = 'Out' AND sit.checkout_window = (SELECT MAX(sit.checkout_window) FROM student_item_transaction WHERE item_id = sit.item_id)) AS `Time Due Back` FROM item AS i, item_condition AS ic, location AS l WHERE i.item_condition_id = ic.id AND i.location_id = l.id AND i.location_id = 0 ORDER BY `Availability`, `Time Due Back` ASC, i.id;`
  - Return: All attributes for items located at the Student Center desk ordered by availability
- SQL (2): `SELECT student.name_first, student.name_last, student.username, student.email, item.name FROM student inner join item on student.id = item.id where student.id = ?;`

- Return: The first name, last name, username, email, and any items checked out by that student where the student ID matches that of the given search term
- SQL (2.1): `SELECT student.name_first, student.name_last, student.username, student.email, item.name FROM student inner join item on student.id = item.id where student.username = ?;`
  - Return: The first name, last name, username, email, and any times checked out by that student where the student username matches that of the given search term
- SQL (2.2): `SELECT student.name_first, student.name_last, student.username, student.email, item.name FROM student inner join item on student.id = item.id where student.name_last = ?;`
  - Return: The first name, last name, username, email, and any items checked out by that student where the student's last name matches that of the given search term
- SQL (3.0): `INSERT INTO employee (id,username, user_type, email, salt, hashed_password, name_first, name_last) VALUES (?,?,,?,,?,,?);`
  - Return: Inserts a new employee into the employee table
- SQL (4.0): `SELECT i.id AS `Item ID`, i.name AS `Item Name`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location` FROM item AS i, item_condition AS ic, location AS l WHERE i.item_condition_id = ic.id AND i.location_id = l.id ORDER BY i.id;`
  - Return: The id, name, availability, condition, and location of all items in the inventory
- SQL (4.1): `SELECT i.id AS `Item ID`, i.name AS `Item Name`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location` FROM item AS i, item_condition AS ic, location AS l WHERE i.item_condition_id = ic.id AND i.location_id = l.id AND i.item_condition_id > 2 ORDER BY i.id;`
  - Return: The id, name, availability, condition, and location of all items in the inventory that are damaged or in unworking condition
- SQL (4.2): `SELECT i.id AS `Item ID`, i.name AS `Item Name`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location` FROM item AS i, item_condition AS ic, location AS l WHERE i.item_condition_id = ic.id AND i.location_id = l.id AND i.available = 0 ORDER BY i.id;`
  - Return: The id, name, availability, condition, and location of all items in the inventory that are currently checked out
- SQL (4.3): `SELECT i.id AS `Item ID`, i.name AS `Item Name`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location` FROM item AS i, item_condition AS ic, location AS l WHERE i.item_condition_id = ic.id AND i.location_id = l.id AND i.available = 1 ORDER BY i.id;`
  - Return: The id, name, availability, condition, and location of all items in the inventory that are currently available for checkout
- SQL (4.4): `SELECT i.id AS `Item ID`, i.name AS `Item Name`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location` FROM item AS i, item_condition AS ic, location AS l WHERE i.item_condition_id = ic.id AND`

- i.location\_id = l.id AND LOWER(i.name) LIKE LOWER('bike%') ORDER BY i.id;
- Return: The id, name, availability, condition, and location of all bikes in the inventory
  - SQL (4.5): SELECT i.id AS `Item ID`, i.name AS `Item Name`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location` FROM item AS i, item\_condition AS ic, location AS l WHERE i.item\_condition\_id = ic.id AND i.location\_id = l.id AND LOWER(i.name) LIKE LOWER('mac%') ORDER BY i.id;
    - Return: The id, name, availability, condition, and location of all Macs in the inventory
  - SQL (4.6): SELECT i.id AS `Item ID`, i.name AS `Item Name`, available AS `Availability`, ic.name AS `Item Condition`, l.name AS `Location` FROM item AS i, item\_condition AS ic, location AS l WHERE i.item\_condition\_id = ic.id AND i.location\_id = l.id AND LOWER(i.name) LIKE LOWER('pc%') ORDER BY i.id;
    - Return: The id, name, availability, condition, and location of all PCs in the inventory
  - SQL (5.0): INSERT INTO item\_category (id, name, waiver, item\_id) VALUES (?, ?, ?, ?);
    - Return: Creates a new row in the item\_category table
  - SQL (6.0): INSERT INTO item\_condition (id, name) VALUES (?, ?);
    - Return: Creates a new row in the item\_condition table
  - SQL (7.0): INSERT INTO item (id, name, available, item\_condition\_id, location\_id) VALUES (?, ?, 1, 1, ?);
    - Return: Creates a new row in the item table with a default availability and condition of available and good respectfully
  - SQL (8.0): INSERT INTO location (id, name, terminal\_id) VALUES (?, ?, ?);
    - Return: Creates a new desk location in the location table
  - SQL (9.0): INSERT INTO waiver (id, name) VALUES (?, ?);
    - Return: Creates a new waiver in the waiver table
  - SQL (10.0): SELECT salt, hashed\_password, user\_type FROM employee WHERE username=?;
    - Return: The salt, hashed password, and user type from the employee table where the username matches the given search term
  - SQL (11.0): SELECT id, name FROM item\_condition ORDER BY id;
    - Return: The id, and name of all rows in the item\_condition table ordered by id
  - SQL (11.1): "UPDATE item SET item\_condition\_id = ' " . \$\_POST[ ' Item\_Condition ' ] . " ' WHERE name = ' " . \$\_POST[ 'Item\_Name' ] . " ' ";
    - Return: Update the item condition table to the new condition id, given as a post variable, where the item name matches the given post variable.

## Normalization

Every table in our database has the characteristics of a 3<sup>rd</sup> normal form database, meeting the following requirements: no repeating groups, no non-prime attribute is



functionally dependent on a proper subset of any candidate key, and no candidate key contains a transitive dependency. The expired\_waiver and item\_condition\_update tables are in 3<sup>rd</sup> normal form. This lower normal form does not require the use of a Super Key. For the expired\_waiver table, a super key could not be used because the same student may have to sign the same waiver again, if it is expired. So to ensure that we do not have large amounts of duplicate data, the candidate key {student\_id, waiver\_id} has been used. Additionally, the item\_condition\_update table contains time stamped data that may be a duplicate of previous data. For example, throughout its lifetime, a computer may go from good condition to un-working and back again. This made using a super key for the table difficult. With the time stamp element present, we have decided to make the candidate key {item\_id, datetime} in order to allow for accurate logging of condition updates. All other tables in our database have been found to be in BCNF.

## Indexing

On our database we have defined quite a few indexes to help with data retrieval and integrity. We use table ID indexes for most personal data and all BCNF tables. Now the indexing for these tables has been defined as the MySQL default because no special indexing was required. Additionally, we have created a unique index on the employee table using the username attribute. This allows for faster login and prevents duplicate username and password combinations. Finally, we have created an auto incrementing index on the student\_item\_transaction table. This auto increment allows for searching of recent transactions fast and easy.

## Security

The Revised MizzouCheckout System application uses several protective security methods, specifically many PHP functions, to prevent SQL injection, Cross-Site Scripting Attacks (XSS) attacks, the use of unsecured HTTP, and other types of attacks. To prevent SQL injection attacks on our site, we used MySQLi prepared statements for any SQL queries we ran, using mysqli\_prepare to link to the database and prepare the SQL query, mysqli\_stmt\_bind\_param to bind the values to the SQL query and create the statement, and mysqli\_stmt\_execute to execute the statements. We also used mysqli\_stmt\_bind\_result and mysqli\_stmt\_fetch when binding the statement to our encrypted data, such as passwords. To prevent XSS attacks, we have used htmlspecialchars on any sensitive user input submitted via forms on our application. We also only store a hashed and salted password in the database, using a random number given via mt\_rand for the salt, and password\_hash for the hashing of the password with PASSWORD\_BCRYPT, which creates a new 60-character password hash using a strong one-way hashing algorithm. We used password\_verify to verify credentials, comparing the user input to the hashed password in the user's row in the database for login and registration. To be sure that a secure HTTP connection is always being used, we used a redirect at the top of each page to redirect to HTTPS. We also set sessions that are checked on each page the user goes to, so that we will know which user has logged in and used their username and user type as our session variables so that only an admin user has access to the navigation bar buttons for registering a new user and adding items.

## User Manual

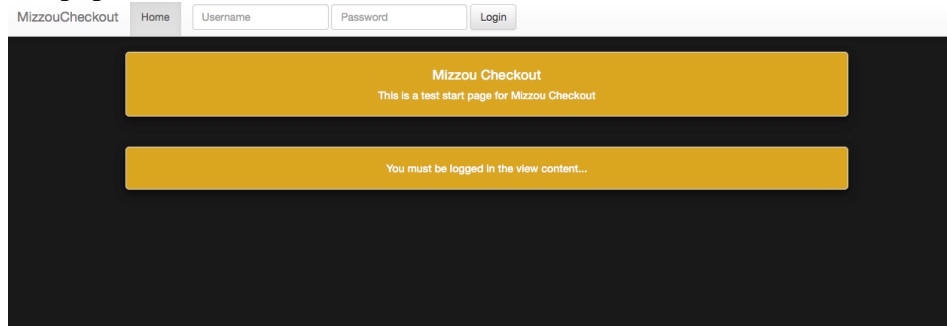
### Installation

The software dependencies related to our system are PHP and MySQL on an Apache server. These are the steps for setting up and deploying our system:

1. Setting up MySQL
  - a. Sudo apt-get install mysql-client mysql-server
    - i. check that it is working by running 'man mysql' from the command line
2. Setting up apache
  - a. Sudo apt-get install apache2
    - i. Agree or respond with 'y' to all question prompts
3. Setting up PHP
  - a. Sudo apt-get update
  - b. Sudo apt-get install php5
  - c. Sudo apt-get install libapache2-mod-php5
  - d. Sudo apt-get install php5-mysql
4. Download
  - a. Sudo mkdir /var/www/html/<foo>
    - i. Where <foo> is the desired directory name
  - b. Cd /var/www/html/<foo>
  - c. Sudo wget <http://cs3380-jadppf.centralus.cloudapp.azure.com/mucs3380spring2016>
5. Database importation
  - a. Next run the MySQL command to run the FinalProject-defs.sql file and import the tables into your database
  - b. Then update db.conf to reflect your current database username, password, and database name.
6. Verify
  - a. Direct your web browser to the destination folder and your system should display the index.php page, meaning everything is working correctly.
  - b. The link to run the site from the welcome page is <https://cs3380-jadppf.centralus.cloudapp.azure.com/mucs3380spring2016/>
  - c. The link to our Github account is <https://github.com/jadppf/mucs3380spring2016>

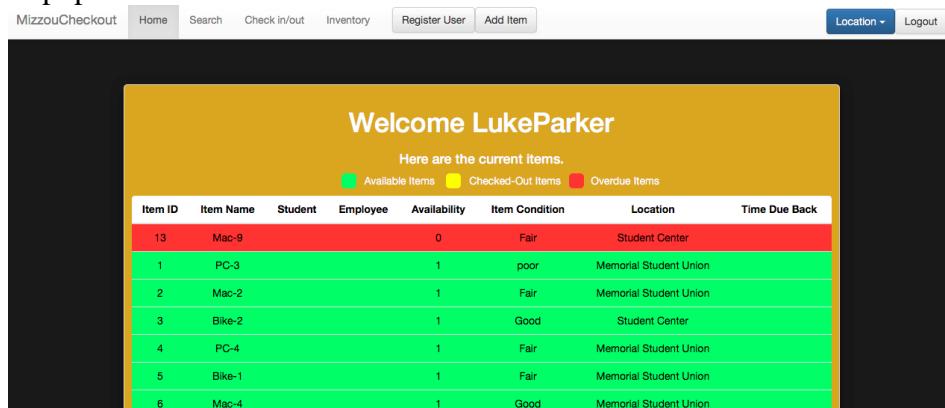
## Website Usage

### ➤ / OR index.php

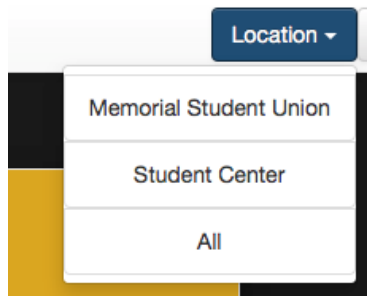


- 
- General welcome page, this should be the page present when the user first opens the website.
- From here a user must login using employee or admin credentials, redirecting them to the welcome.php page
- You may use the test login information to try this out. Use username: adminUser, password: admin to log in as an admin user, and username: regularUser, password: test to log in as a regular user.

### ➤ Welcome.php



- 
- Once logged in employees will be presented with an inventory overview. This displays Item ID, Item Name, Student and Employee ID tags associated with transaction, availability, condition, location, and return time.
- The location drop-down menu on the navigation bar can be used to sort items based on particular locations or to view all items



-

- From here an employee can access any button across the top navigation bar, including add item and register user if you are an admin user.

### ➤ Shoppingcart.php

- If the user clicks on the search tab, then they will be directed here, where they can search for currently checked out items by Item ID, Student ID, Pawprint, or Last Name using the appropriate radio button selection. The database returns the corresponding results in a table once the query is run, as seen here:

First Name	Last Name	Pawprint	Email	Items Checked Out
Zach	Dolan	zmd989	zmd989@missouri.edu	Bike-1

### ➤ Check.php

- If the user clicks on the Check in/out tab they will be directed to this page, where they will see “Welcome” and their username retrieved from the session variable set when they logged in.
- They are asked to provide an Item ID and Student ID for an item transaction. To check an item out both the item ID and student ID are required. To check an item in you only need the item ID.

Check In or Check Out an Item

Enter Item ID:

Enter Student ID:

Check In/Out

Item Was Checked Out

- 
- A confirmation is posted indicating if the item was successfully checked out

Check In or Check Out an Item

Enter Item ID:

Enter Student ID:

Check In/Out

Item Was Checked In

- 
- A confirmation is posted indication if the item was successfully checked back in

## ➤ Inventory.php

MizzouCheckout. Home Search Check In/out Inventory Register User Add Item Logout

☒ All 
 ☐ Damaged 
 ☐ Checked out 
 ☐ Checked In 
 ☐ Bikes 
 ☐ MACS 
 ☐ PCS

Go

	Item ID	Item Name	Availability	Item Condition	Location
<a href="#">Update</a>	1	PC-3	1	poor	Memorial Student Union
<a href="#">Update</a>	2	Mac-2	1	Fair	Memorial Student Union
<a href="#">Update</a>	3	Bike-2	1	Good	Student Center
<a href="#">Update</a>	4	PC-4	1	Fair	Memorial Student Union
<a href="#">Update</a>	5	Bike-1	1	Fair	Memorial Student Union
<a href="#">Update</a>	6	Mac-4	1	Good	Memorial Student Union
<a href="#">Update</a>	10	Mac-8	1	Good	Memorial Student Union

- If the user selects the inventory tab, they will be directed to this page, presenting a table of all items listed in the inventory.
- The radio buttons, located top center, can be used to change the view displayed, filtering the results in the table based on various search terms, such as damaged, checked in, checked out, bikes, Macs, or PCs.
- Clicking the update button associated with an item will direct you to the edit.php page.

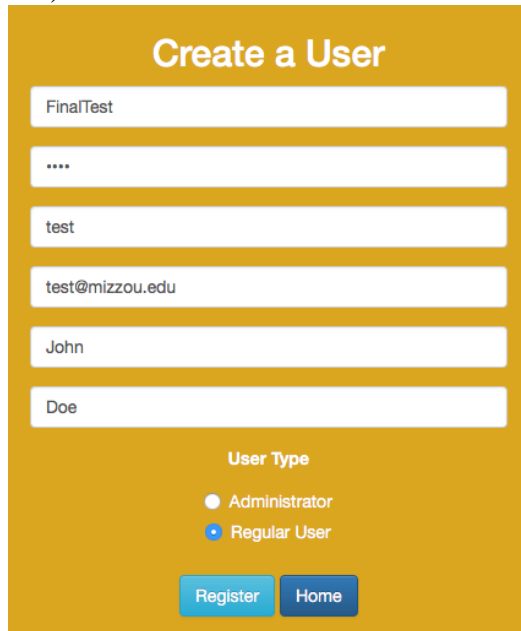
#### ➤ Edit.php

- From here you can update item conditions from their current condition to a new condition, and the top of the page tells you which item you are updating.
- Once the desired new condition has been selected using the drop down menu, clicking the Go button will update the condition of that item.

- This message will confirm the condition update, and gives you the option to go back to the inventory.php page for more viewing or updating.

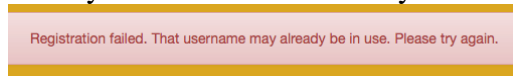
#### ➤ Register.php

- When an admin user clicks the Register User button only available to admin users in the navigation bar, they are taken to the registration form page.
- Here the session is checked to be a logged in admin user, and if they are not they are sent back to the home page. If they are a logged in, registered user, they enter in the desired username, password, student ID, email, first and last names (there will be an error notice if this ID or username is already in use).



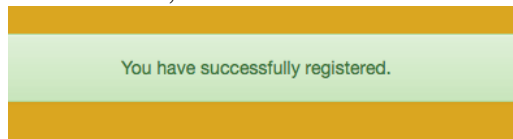
The image shows a registration form titled "Create a User" on an orange background. It contains several input fields: "FinalTest" (username), a password field with four asterisks, "test" (student ID), "test@mizzou.edu" (email), "John" (first name), and "Doe" (last name). Below these fields is a "User Type" section with two radio buttons: "Administrator" (unselected) and "Regular User" (selected). At the bottom are two buttons: "Register" (orange) and "Home" (blue).

- They then click the radio button for admin or regular user for the desired user privileges (admin allows access to creating items and users).
- Once they click register, the code saves the user input for username, password, student ID, email, first and last names to the database - using salt and hash to encrypt the password and htmlspecialchars to convert special characters to HTML entities, and further uses mysqli prepared statements for added security.
- This will either produce a failure, if for example the username or ID has already been entered into the system.



The image shows a red error message box with the text: "Registration failed. That username may already be in use. Please try again."

- Or a success, if the new user was successfully added to the database.



The image shows a green success message box with the text: "You have successfully registered."

- They can then go back to the home page by clicking the 'Home' button, to welcome.php, or log out to be sent to the welcome page, index.php, to log in as the new user, if desired.

## ➤ Insert.php

MizzouCheckout Home Search Check in/out Inventory Register User Add Item Logout

Select What to Insert

Item

go

- 
- Clicking the Add Item button, also only available to admin users, will direct the user to the insert.php page where the user can add new items into the inventory.
- Use the drop down menu to select the object that you would like to create. Options include creating items, item categories, item conditions, waivers, and desk locations.
- Hitting 'Go' will bring up the appropriate form for what you would like to create. The following example uses item, but all options follow the same work flow.

Select What to Insert

Item

go

Create an Item

Barcode

Name

Location Id

Create Home

- 
- Once filled out hit 'Create' to submit the form

Select What to Insert

Item

go

Create an Item

7

Mac-1

0

Create Home

Successfully Created Item

-



- You will receive a confirmation stating if the item was successfully created.
- Clicking the 'Home' button will direct you back to the welcome.php page.

➤ Logout.php

- Selecting 'Logout' in the top right corner will direct the user to the logout.php page.
- The user's session variables will be destroyed, and then the user is redirected to the index.php page for login.