Project Owner: Eric Reynolds Team Members: Clinton Morris, Jesse Asbury, Will Westergren, Hazel Walker

PlayCart POS

Project Documentation

A Team 6   
Project

Table of Contents

Product Description………………………………………………………..…………………………2-3

MVP………………………………………………………………………………………………………..4

Database Implementation………………………………………………..………………………..5-7

ERD…………………………………………………………………………………………………………8

SQL Code……………………………………………………………………………………………..9-13

Programming Flowcharts…………………………………………………………………………….14

Project Code……………………………………………………………………………………...15-126

Network Implementation………………………………………………………………..…….127-129

User Interface and Purpose…………………………………………………………..………130-131

GUI Flowchart…………………………………………………………………………….……..132-134

Initial Cybersecurity Concerns………………………………………………………….…....134-136

Expected and Experienced Challenges………..……………………………………….…137-138

RTM……………………………………………………………………………………………..…139-149

Bug Tracking and Fixes………………………………………………………………………...150-151

Strengths, Weaknesses, and Expected Contributions……………………….……….…152-155

GUI Screenshots…………………………………………………………………………………156-164

Works Cited……………………………………………………………………………………………165

Appendix A: Issues That Could Not be Solved…………………………………………………166

Appendix B: The Use of AI as a Tool…………………………………………………………167-168

Appendix C: Great Things…………………………………………………………………….169-170

Group Signatures………………………………………………………………………………….…171

Product Description

PlayCart POS is a modern point-of-sale system built specifically for video game retail stores. Designed with everyday users in mind, it runs on a locally hosted MariaDB database using a Raspberry Pi, offering a reliable and affordable solution for managing core store operations. The system combines ease of use with powerful features to help stores run more efficiently without requiring advanced technical knowledge.

At the center of PlayCart POS is a flexible inventory management system that tracks video game titles, platforms, pricing, and stock levels. This gives employees a clear and up-to-date view of what’s available, making restocking and sales easier to manage. The system is built to reduce errors and keep inventory accurate, which helps improve both customer satisfaction and store performance.

The sales process is designed to be quick and simple. Employees can easily record purchases or rentals, with each transaction automatically linked to both the customer and the game involved. This helps speed up checkouts, reduce mistakes, and offer better customer service. By supporting both purchases and game rentals, PlayCart POS also allows stores to offer more flexible options to their customers.

The system securely stores customer information, making it easy to identify returning customers and provide faster, more personalized service. It also keeps a complete daily record of all transactions, including date, time, game and customer IDs, and payment methods. These records are important for generating reports, reviewing performance, and helping store managers make informed business decisions.

From a management standpoint, PlayCart POS makes it simple to add and remove employee accounts, ensuring that only authorized staff have access to system features. To keep data safe, the system includes built-in support for USB backups, so data can be saved externally at any time. This protects against data loss and ensures business continuity in case of technical issues.

PlayCart POS is designed to meet the essential needs of video game retailers while keeping things straightforward and dependable. Whether managing inventory, handling customer transactions, tracking rentals, or overseeing staff, the system provides all the tools needed to run a successful store. With room for future upgrades and added features, PlayCart POS is a strong foundation for growing and managing a modern game retail business. (edited)

MVP

PlayCart is a software application designed to streamline the daily operations of a video game retail store by leveraging a locally hosted database using MariaDB on a Raspberry Pi. This system features inventory tracking, maintaining a comprehensive database of video games—including titles, platforms, prices, and stock levels—allowing employees to manage entries easily. Additionally, PlayCart facilitates sales processing by enabling employees to efficiently record transactions linked to specific customers and games, enhancing customer service. The application also supports rentals, allowing customers to rent games alongside purchases, further expanding service offerings. It logs vital transaction details such as dates, customer IDs, game IDs, and tracks how a sale was completed, which are crucial for generating sales reports and providing basic reporting. By focusing on these core functionalities, PlayCart provides a robust solution that meets the basic operational needs of a retail store while laying the groundwork for future enhancements and scalability.

Database Implementation

We use MariaDB as our database system, hosted locally on the Raspberry Pi. This satisfies the requirement for a meaningful, integrated, and relational database that runs directly on the device.

Our database includes six related tables:

**Games**

This table stores all available video game inventory. Each record includes data such as the game title, platform, price, stock quantity, and availability status. It will be referenced in both purchase and rental operations.

**Customers**

This table holds information about store customers, including their name, contact details, and a unique customer ID. It links to both transactions and rentals to track a customer’s activity history.

**Transactions**

This table logs all completed sales. Each transaction entry includes the date, the customer who made the purchase, the game(s) sold, and the employee who processed the sale. It supports reporting, receipts, and customer history.

**Employees**

This table manages user access. It stores employee names, login credentials (securely hashed), and access roles (such as cashier or manager). It will help restrict access to admin-level features.

**Rentals**

This table tracks all rental activity. It includes fields for rental start and end dates, game ID, customer ID, return status, and associated employee. This allows us to simulate a rental process with check-in/check-out logic.

**EventLogs**

This table will log any issues experienced by the user and label them by event type as well as display information about when the error occurred and with which employee user. The log table will also include a “details” field that stores more information about the logged event.

**Cart**

This table holds on to specific transaction information like the game ID selected by the customer, as well as the type of transaction being completed and the session ID number specific to that transaction.

**DailySummary**

This table provided a summary of the expected vs actual sales for the day, along with the variance between the two. It also keeps track of which employee closed out the day and what time it was closed.

**BusinessDayStatus**

Keeps track of daily sales with a drop total and an expected total, and will show whether there is a variance between what was expected to be brought in by the end of the day and what was reported. This table references the “employees” table and keeps record of which employee made the drop for a specific night.

All tables are connected through foreign key relationships where appropriate. For example, transactions and rentals reference both games and customers by their IDs. The employees table is referenced to show who completed each action.

The database is accessed through backend scripts hosted by Apache on the Raspberry Pi. These scripts will handle real-time data operations like updating inventory, recording sales, managing user logins, and tracking rentals. All operations will be handled locally within the Pi environment.

This structure aligns with our project goals and supports the key functions of a video game store POS system.

ERD

A diagram of a company

AI-generated content may be incorrect.

SQL Code

**PlayCart\_Schema.sql:**

****

CREATE DATABASE PlayCart;

USE PlayCart;

CREATE USER 'playcart\_admin'@'localhost' IDENTIFIED BY 'PlayCartAdmin2025!';

GRANT SELECT, INSERT, UPDATE, DELETE ON PlayCart.\* TO 'playcart\_admin'@'localhost';

FLUSH PRIVILEGES;

CREATE TABLE Employees (

employee\_id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

username VARCHAR(50) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

role ENUM('employee', 'manager') NOT NULL,

is\_active BOOLEAN DEFAULT TRUE,

password\_set BOOLEAN DEFAULT FALSE

);

CREATE TABLE Games (

game\_id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100),

platform VARCHAR(30) NOT NULL,

sku VARCHAR(30) UNIQUE NOT NULL,

cost DECIMAL(10,2),

sale\_price DECIMAL(10,2),

rent\_price DECIMAL(10,2),

quantity\_in\_stock INT DEFAULT 0,

date\_received DATETIME,

game\_type ENUM('sale', 'rental', 'both') DEFAULT 'sale'

);

CREATE TABLE Customer (

customer\_id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

address VARCHAR(100),

email VARCHAR(50) UNIQUE NOT NULL,

phone\_number varchar(15),

customer\_number INT UNIQUE

);

CREATE TABLE Rentals (

rental\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT NOT NULL,

employee\_id INT NOT NULL,

game\_id INT NOT NULL,

rent\_out\_date DATETIME NOT NULL,

rent\_return\_date DATETIME,

returned BOOLEAN DEFAULT FALSE,

FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id),

FOREIGN KEY (employee\_id) REFERENCES Employees(employee\_id),

FOREIGN KEY (game\_id) REFERENCES Games(game\_id)

);

CREATE TABLE Transactions (

transaction\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT NOT NULL,

employee\_id INT NOT NULL,

game\_id INT NOT NULL,

order\_total DECIMAL(10,2) NOT NULL,

order\_date DATETIME NOT NULL,

transaction\_type VARCHAR(20) NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id),

FOREIGN KEY (employee\_id) REFERENCES Employees(employee\_id),

FOREIGN KEY (game\_id) REFERENCES Games(game\_id)

);

CREATE TABLE EventLogs (

event\_log\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT NOT NULL,

event\_type VARCHAR(50) NOT NULL,

event\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

details TEXT,

FOREIGN KEY (employee\_id) REFERENCES Employees(employee\_id)

);

CREATE TABLE Cart (

cart\_id INT AUTO\_INCREMENT PRIMARY KEY,

session\_id VARCHAR(255) NOT NULL,

game\_id INT NOT NULL,

transaction\_type ENUM('purchase', 'rental', 'return') NOT NULL,

FOREIGN KEY (game\_id) REFERENCES Games(game\_id)

);

CREATE TABLE DailySummary (

summary\_id INT AUTO\_INCREMENT PRIMARY KEY,

summary\_date DATE NOT NULL,

expected DECIMAL(10,2) NOT NULL,

actual DECIMAL(10,2) NOT NULL,

variance DECIMAL(10,2) NOT NULL,

employee\_id INT NOT NULL,

closed\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (employee\_id) REFERENCES Employees(employee\_id)

);

CREATE TABLE BusinessDayStatus (

id INT AUTO\_INCREMENT PRIMARY KEY,

business\_date DATE NOT NULL,

is\_closed BOOLEAN DEFAULT FALSE,

closed\_at DATETIME,

drop\_total DECIMAL(10,2),

expected\_total DECIMAL(10,2),

variance DECIMAL(10,2),

closed\_by INT,

FOREIGN KEY (closed\_by) REFERENCES Employees(employee\_id)

);

Programming Flowcharts

A screenshot of a diagram

AI-generated content may be incorrect.

Project Code

**style.css:**

****

:root {

--primary: #ad84c6;

--accent: #D4AF37;

--text: #1A1A1A;

--bg: #ad84c6;

--white: #ffffff;

--success: #5CB85C;

--error: #D9534F;

--font-main: 'Segoe UI', 'Helvetica Neue', 'Arial', sans-serif;

}

/\* === GLOBAL === \*/

body {

font-family: var(--font-main);

color: var(--text);

background-color: var(--primary);

margin: 0;

padding: 0;

}

/\* === GLOBAL MAIN CONTAINER === \*/

main {

max-width: 1200px;

margin: 3rem auto;

padding: 2rem 3rem;

background-color: var(--white);

border-radius: 12px;

box-shadow: 0 0 12px rgba(0, 0, 0, 0.05);

}

/\* === HEADERS === \*/

h1, h2, h3 {

color: var(--primary);

}

/\* === BUTTONS === \*/

.btn {

background-color: #6e3b9f;

color: #ffffff;

font-family: inherit;

font-size: 1rem;

padding: 0.75rem 1.75rem;

border-radius: 6px;

cursor: pointer;

font-weight: bold;

border: none;

width: 200px;

height: 48px;

text-align: center;

transition: background-color 0.2s ease;

white-space: nowrap;

display: inline-flex;

align-items: center;

justify-content: center;

}

.btn:hover,

.btn:focus {

background-color: #5c2f8a;

outline: none;

}

/\* === BUTTON GROUP === \*/

.button-group {

display: flex;

flex-wrap: wrap;

gap: 1rem;

margin-top: 2rem;

margin-bottom: 2rem;

justify-content: center;

}

/\* === TABLES === \*/

table {

width: 100%;

max-width: 100%;

border-collapse: collapse;

margin-top: 1rem;

table-layout: fixed;

word-wrap: break-word;

}

th, td {

padding: 10px;

text-align: left;

border-bottom: 1px solid #ccc;

}

th {

background-color: var(--primary);

color: var(--text);

}

table tr:nth-child(even) {

background-color: #f9f9f9;

}

/\* === FORMS === \*/

input, select {

padding: 10px;

border: 1px solid #ccc;

border-radius: 6px;

font-size: 1rem;

font-family: inherit;

margin: 5px 0;

}

form {

margin-bottom: 1rem;

}

.form {

display: flex;

flex-wrap: wrap;

gap: 1rem;

margin-top: 2rem;

margin-bottom: 2rem;

align-items: flex-end;

}

.form label {

display: flex;

flex-direction: column;

font-weight: bold;

font-size: 0.95rem;

}

.form label.button-row {

display: flex;

flex-direction: row;

gap: 1rem;

align-items: flex-end;

}

/\* === INLINE ROW FORM === \*/

.form-inline-row > div {

flex: 1 1 300px;

min-width: 300px;

}

/\* === SMALLER BUTTONS IN TABLES === \*/

td .btn {

height: 32px;

padding: 0.4rem 1rem;

font-size: 0.85rem;

}

/\* === EMPLOYEE TABLE FIELDS === \*/

.employee-table td input,

.employee-table td select {

padding: 6px 8px;

font-size: 0.9rem;

height: 32px;

max-width: 140px;

box-sizing: border-box;

}

.employee-table th,

.employee-table td {

padding: 6px 8px;

}

.employee-table th:first-child,

.employee-table td:first-child {

text-align: center;

width: 40px;

}

/\* === STATUS COLORS === \*/

.success {

color: var(--success);

}

.error {

color: var(--error);

}

/\* === UNIVERSAL FONT & SPACING FIX === \*/

body, input, select, textarea, button {

font-size: 1rem;

font-family: var(--font-main);

line-height: 1.6;

box-sizing: border-box;

}

\*,

\*::before,

\*::after {

box-sizing: inherit;

}

/\* === Inventory Table Layout Improvements === \*/

.inventory-table td input,

.inventory-table td select {

width: 100%;

padding: 8px;

font-size: 1rem;

border-radius: 5px;

border: 1px solid #ccc;

box-sizing: border-box;

}

.inventory-table td {

vertical-align: top;

padding: 12px 8px;

}

.inventory-table form {

display: contents;

}

.inventory-table td {

vertical-align: middle;

}

body.close-day-page .form button.btn {

align-self: flex-end;

height: 48px;

}

<style>

.close-day-table {

width: 100%;

border-collapse: collapse;

table-layout: fixed;

margin-top: 1rem;

}

.close-day-table th,

.close-day-table td {

padding: 12px 10px;

border-bottom: 1px solid #ccc;

text-align: left;

vertical-align: middle;

}

.close-day-table th {

background-color: var(--primary);

color: var(--text);

}

.close-day-table td input[type="number"] {

width: 100%;

max-width: 100px;

padding: 6px;

box-sizing: border-box;

}

.close-day-table td .btn {

width: 100%;

max-width: 120px;

padding: 6px 12px;

font-size: 0.9rem;

}

</style>

.warning {

display: block;

font-weight: bold;

font-size: 0.85rem;

margin-top: 0.25rem;

}

.warning-low {

color: #c62828;

}

.warning-out {

color: #000;

}

**usb\_command.sh:**



#!/bin/bash

LOGFILE="/var/log/usb\_operations.log"

while true; do

if [ -f /var/www/html/usb\_command ]; then

CMD=$(cat /var/www/html/usb\_command)

echo "$(date) - Received command: $CMD" >> $LOGFILE

if [ "$CMD" == "mount" ]; then

mount /media/usb >> $LOGFILE 2>&1

echo "$(date) - Mount attempted via fstab." >> $LOGFILE

elif [ "$CMD" == "unmount" ]; then

umount /media/usb >> $LOGFILE 2>&1

echo "$(date) - Unmount attempted." >> $LOGFILE

else

echo "$(date) - Unknown command: $CMD" >> $LOGFILE

fi

rm /var/www/html/usb\_command

fi

sleep 1

done

**Backup\_To\_USB.php:**

****

<?php

session\_start();

date\_default\_timezone\_set('America/New\_York');

// Session Timeout

$timeout\_duration = 900; // 15 minutes

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY']) > 1800) {

session\_unset();

session\_destroy();

header("Location: Login.php?timeout=1");

exit;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // Reset timer on activity

// Require login before allowing backup

if (!isset($\_SESSION['employee\_id'])) {

die("Unauthorized access.");

}

$usbPath = '/media/usb';

$showOutput = '';

// Handle mount request

if (isset($\_GET['mount'])) {

file\_put\_contents('/var/www/html/usb\_command', 'mount');

$showOutput .= "<p>Unmounting... Please wait a moment before unplugging the drive.</p>";

sleep(2);

}

// Handle unmount request

if (isset($\_GET['unmount'])) {

file\_put\_contents('/var/www/html/usb\_command', 'unmount');

$showOutput .= "<p>Unmounting... Please wait a moment before unplugging the drive.</p>";

sleep(2);

}

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

} catch (PDOException $e) {

// Silent fail: don't crash the page

$pdo = null;

}

// Run database backup

if (isset($\_GET['backup'])) {

$dbUser = 'playcart\_admin';

$dbPass = 'PlayCartAdmin2025!';

$dbName = 'PlayCart';

$date = date('Y-m-d\_H-i-s');

$filename = "playcart\_backup\_$date.sql";

$fullPath = "$usbPath/$filename";

// Confirm USB is mounted

$mountCheck = shell\_exec("findmnt $usbPath");

if (empty($mountCheck)) {

$showOutput .= "<p style='color:red;'>USB is not mounted at <code>$usbPath</code>. Backup aborted.</p>";

$showOutput .= "<p>Please mount the USB before running a backup.</p>";

} else {

// Confirm we can write

shell\_exec("echo 'test\_write' > $usbPath/\_php\_write\_test.txt");

// Run the backup using --skip-lock-tables to avoid permission issues

$command = "mysqldump --skip-lock-tables -u $dbUser -p$dbPass $dbName > '$fullPath' 2>&1";

$output = shell\_exec($command);

if (file\_exists($fullPath)) {

$showOutput .= "<p style='color:green;'>Backup created: <strong>$filename</strong></p>";

$showOutput .= "<p>Saved to: <code>$fullPath</code></p>";

// Log successful backup

try {

$stmt = $pdo->prepare("INSERT INTO EventLogs (employee\_id, event\_type, details) VALUES (?, 'backup\_success', ?)");

$stmt->execute([$\_SESSION['employee\_id'], "Backup saved to $fullPath"]);

} catch (PDOException $e) {

// Optional: silently ignore logging failure

}

} else {

$showOutput .= "<p style='color:red;'>Backup failed. Debug output:</p><pre>$output</pre>";

// Log failed backup

try {

$stmt = $pdo->prepare("INSERT INTO EventLogs (employee\_id, event\_type, details) VALUES (?, 'backup\_fail', ?)");

$stmt->execute([$\_SESSION['employee\_id'], "Backup failed. Command output: $output"]);

} catch (PDOException $e) {

// Optional: silently ignore logging failure

}

}

}

}

?>

<!DOCTYPE html>

<html>

<head>

<title>Backup to USB</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<main>

<h1 style="color: var(--primary);">Backup PlayCart to USB</h1>

<?= $showOutput ?>

<form method="get" class="button-group">

<button class="btn" name="mount" value="1">Mount USB</button>

<button class="btn" name="unmount" value="1">Unmount USB</button>

<button class="btn" name="backup" value="1">Run Backup</button>

</form>

<form method="get" action="Employee\_Dashboard.php" class="button-group">

<button class="btn" type="submit">Back to Dashboard</button>

</form>

<hr>

<div style="display: flex; justify-content: center; margin-top: 2rem;">

<div style="background: #f4f4f4; padding: 1rem 1.5rem; border-radius: 10px; width: fit-content; box-shadow: 0 0 5px rgba(0,0,0,0.1); font-size: 0.95rem;">

<h3 style="color: var(--primary); text-align: center; margin-top: 0;">Real-Time USB Debug</h3>

<pre style="margin: 0;">

<?php

echo "lsblk (filtered):\n";

echo shell\_exec("lsblk -o NAME,MOUNTPOINT,SIZE,FSTYPE -e7 | grep -v '^ram'");

?>

</pre>

</div>

</div>

</body>

</html>

**Checkout.php:**

****

<?php

session\_start();

date\_default\_timezone\_set('America/New\_York');

// Session Timeout

$timeout\_duration = 900; // 15 minutes

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY']) > 1800) {

session\_unset();

session\_destroy();

header("Location: Login.php?timeout=1");

exit;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // Reset timer on activity

// Block access if not logged in

if (!isset($\_SESSION['employee\_id'], $\_SESSION['username'], $\_SESSION['role'])) {

header("Location: Login.php");

exit;

}

// DB connection

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

} catch (PDOException $e) {

die("Database connection failed: " . $e->getMessage());

}

$taxRate = 0.043;

$confirmation = $error = '';

$customer = null;

$games = [];

// Handle cancel order

if (isset($\_GET['reset']) && $\_GET['reset'] == 1) {

unset($\_SESSION['selected\_games'], $\_SESSION['selected\_customer']);

header("Location: Transaction\_Page.php");

exit;

}

// Load session data

$customer\_id = $\_SESSION['selected\_customer'] ?? null;

$selected\_game\_quantities = $\_SESSION['selected\_games'] ?? [];

$checkout\_type = $\_POST['checkout\_type'] ?? $\_GET['checkout\_type'] ?? null;

$payment\_type = $\_POST['payment\_type'] ?? null;

$totalDue = isset($\_POST['total\_due']) ? floatval($\_POST['total\_due']) : 0;

$employee\_id = $\_SESSION['employee\_id'] ?? null;

// Fetch customer and game details

if ($customer\_id) {

$stmt = $pdo->prepare("SELECT \* FROM Customer WHERE customer\_id = ?");

$stmt->execute([$customer\_id]);

$customer = $stmt->fetch();

}

$games = [];

if (!empty($\_SESSION['selected\_games']) && is\_array($\_SESSION['selected\_games'])) {

$selected\_game\_quantities = $\_SESSION['selected\_games'];

$gameIds = array\_keys($selected\_game\_quantities);

$placeholders = implode(',', array\_fill(0, count($gameIds), '?'));

$stmt = $pdo->prepare("SELECT \* FROM Games WHERE game\_id IN ($placeholders)");

$stmt->execute($gameIds);

$fetchedGames = $stmt->fetchAll();

foreach ($fetchedGames as $game) {

$game['quantity'] = $selected\_game\_quantities[$game['game\_id']] ?? 1;

$games[] = $game;

}

}

// Calculate total

$subtotal = 0;

if ($checkout\_type && $games) {

foreach ($games as $game) {

$qty = $game['quantity'] ?? 1;

$price = ($checkout\_type === 'rental') ? $game['rent\_price'] : $game['sale\_price'];

$subtotal += $price \* $qty;

}

$salesTax = round($subtotal \* $taxRate, 2);

$calculatedTotal = round($subtotal + $salesTax, 2);

if ($totalDue === 0) $totalDue = $calculatedTotal;

}

// PREVIEW fallback for when transaction type not selected yet

if ($totalDue === 0 && $games) {

$previewSubtotal = 0;

foreach ($games as $game) {

$previewSubtotal += $game['sale\_price']; // Default to purchase preview

}

$previewTax = round($previewSubtotal \* $taxRate, 2);

$totalDue = round($previewSubtotal + $previewTax, 2);

}

// Handle return logic

if ($checkout\_type === 'return' && isset($\_POST['process\_payment'])) {

$now = date('Y-m-d H:i:s');

foreach ($games as $game) {

$stmt = $pdo->prepare("UPDATE Rentals SET returned = 1, rent\_return\_date = ? WHERE customer\_id = ? AND game\_id = ? AND returned = 0 LIMIT 1");

$stmt->execute([$now, $customer\_id, $game['game\_id']]);

}

$confirmation = "Return Completed.";

unset($\_SESSION['selected\_customer'], $\_SESSION['selected\_games']);

}

// Handle purchase/rental payment

if (isset($\_POST['process\_payment']) && $checkout\_type !== 'return') {

$now = date('Y-m-d H:i:s');

if ($payment\_type === 'card' || $payment\_type === 'split') {

$card = $\_POST['card\_number'] ?? $\_POST['split\_card\_number'] ?? '';

$expiry = $\_POST['expiry'] ?? '';

$cvv = $\_POST['cvv'] ?? '';

if (!preg\_match('/^\d{13,19}$/', $card)) {

$error = "Invalid card number. Must be 13–19 digits.";

} elseif (!preg\_match('/^(0[1-9]|1[0-2])\/(\d{2}|\d{4})$/', $expiry)) {

$error = "Invalid expiry date. Use MM/YY or MM/YYYY.";

} elseif (!preg\_match('/^\d{3,4}$/', $cvv)) {

$error = "Invalid CVV. Must be 3 or 4 digits.";

}

if (!$error) {

$last4 = substr($card, -4);

}

}

if (!$error) {

if ($payment\_type === 'cash') {

$cash = floatval($\_POST['cash\_amount']);

if ($cash < $totalDue) {

$error = "Insufficient payment. Please enter at least $" . number\_format($totalDue, 2);

} else {

$change = round($cash - $totalDue, 2);

$confirmation = "Transaction Complete - Change: $" . number\_format($change, 2);

}

} elseif ($payment\_type === 'card') {

$confirmation = "Transaction Complete - Paid with card ending in $last4";

} elseif ($payment\_type === 'split') {

$split\_cash = floatval($\_POST['split\_cash']);

if ($split\_cash >= $totalDue) {

$change = round($split\_cash - $totalDue, 2);

$confirmation = "Transaction Complete - Paid in cash, Change: $" . number\_format($change, 2);

} else {

$remaining = $totalDue - $split\_cash;

$confirmation = "Transaction Complete - Paid $" . number\_format($split\_cash, 2) . " in cash, $" . number\_format($remaining, 2) . " with card ending in $last4";

}

}

// Record in Transactions and Rentals

foreach ($games as $game) {

$price = ($checkout\_type === 'rental') ? $game['rent\_price'] : $game['sale\_price'];

$priceWithTax = round($price + ($price \* $taxRate), 2);

// Insert a transaction for each quantity of the game

for ($i = 0; $i < $game['quantity']; $i++) {

$stmt = $pdo->prepare("INSERT INTO Transactions (customer\_id, employee\_id, game\_id, order\_total, order\_date, transaction\_type)

VALUES (?, ?, ?, ?, ?, ?)");

$stmt->execute([$customer\_id, $employee\_id, $game['game\_id'], $priceWithTax, $now, $checkout\_type]);

}

// Reduce inventory by quantity

if (in\_array($checkout\_type, ['rental', 'purchase'])) {

$stmt = $pdo->prepare("UPDATE Games SET quantity\_in\_stock = quantity\_in\_stock - ? WHERE game\_id = ?");

$stmt->execute([$game['quantity'], $game['game\_id']]);

}

// Insert into Rentals table if rental

if ($checkout\_type === 'rental') {

for ($i = 0; $i < $game['quantity']; $i++) {

$stmt = $pdo->prepare("INSERT INTO Rentals (customer\_id, employee\_id, game\_id, rent\_out\_date)

VALUES (?, ?, ?, ?)");

$stmt->execute([$customer\_id, $employee\_id, $game['game\_id'], $now]);

}

}

}

unset($\_SESSION['selected\_customer'], $\_SESSION['selected\_games']);

}

}

// Prevent overwriting total if already calculated above

if ($subtotal === 0 && !empty($games) && !$checkout\_type) {

foreach ($games as $game) {

$qty = $game['quantity'] ?? 1;

$subtotal += $game['sale\_price'] \* $qty;

}

$salesTax = round($subtotal \* $taxRate, 2);

$totalDue = round($subtotal + $salesTax, 2);

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Checkout</title>

<link rel="stylesheet" href="style.css">

<style>

.checkout-section {

margin-bottom: 2rem;

}

.form-inline-row {

display: flex;

flex-wrap: wrap;

gap: 1rem;

align-items: flex-start;

margin-bottom: 1rem;

}

.form-inline-row label {

display: flex;

flex-direction: column;

font-weight: bold;

}

.form-inline-row input {

min-width: 180px;

}

.divider {

margin: 2rem 0;

border-top: 1px solid #ccc;

}

</style>

</head>

<body>

<main>

<h1>Checkout</h1>

<?php if ($error): ?>

<p class="error"><?php echo htmlspecialchars($error); ?></p>

<form method="post">

<input type="hidden" name="checkout\_type" value="<?php echo $checkout\_type; ?>">

<input type="hidden" name="payment\_type" value="<?php echo $payment\_type; ?>">

<input type="hidden" name="total\_due" value="<?php echo $totalDue; ?>">

<button type="submit">Try Again</button>

</form>

<form method="get" action="Checkout.php">

<input type="hidden" name="reset" value="1">

<button type="submit">Cancel Order</button>

</form>

<?php elseif ($confirmation): ?>

<p class="success"><?php echo htmlspecialchars($confirmation); ?></p>

<form method="get" action="Checkout.php">

<input type="hidden" name="reset" value="1">

<button type="submit">Start New Transaction</button>

</form>

<?php elseif ($customer && $games && !$payment\_type): ?>

<form method="get" action="Checkout.php" style="margin-bottom: 1em;">

<input type="hidden" name="reset" value="1">

<button type="submit">Cancel Order</button>

</form>

<div class="form-inline-row">

<div style="flex: 1 1 300px;">

<h3 style="color: var(--primary);">Customer:</h3>

<p>

<?php echo htmlspecialchars("{$customer['first\_name']} {$customer['last\_name']}"); ?><br>

Email: <?php echo htmlspecialchars($customer['email']); ?><br>

Phone: <?php echo htmlspecialchars($customer['phone\_number']); ?><br>

Address: <?php echo htmlspecialchars($customer['address']); ?>

</p>

</div>

<div style="flex: 1 1 300px;">

<h3 style="color: var(--primary);">Games in Cart:</h3>

<ul style="margin-top: 0;">

<?php foreach ($games as $game): ?>

<li>

<?php echo htmlspecialchars($game['title']); ?> (<?php echo htmlspecialchars($game['platform']); ?>) x<?= $game['quantity'] ?>

Sale: $<?php echo number\_format($game['sale\_price'], 2); ?> |

Rental: $<?php echo number\_format($game['rent\_price'], 2); ?>

</li>

<?php endforeach; ?>

</ul>

</div>

</div>

<p><strong>Subtotal: $<?php echo number\_format($subtotal, 2); ?></strong></p>

<p><strong>Sales Tax (4.3%): $<?php echo number\_format($salesTax, 2); ?></strong></p>

<p><strong>Total (with tax): $<?php echo number\_format($totalDue, 2); ?></strong></p>

<form method="post" id="checkout-form" style="margin-top: 1em;">

<label><strong>Transaction Type:</strong></label>

<select name="checkout\_type" id="checkout\_type" required onchange="handleCheckoutTypeChange()">

<option value="">-- Select --</option>

<option value="purchase" <?= $checkout\_type === 'purchase' ? 'selected' : '' ?>>Purchase</option>

<option value="rental" <?= $checkout\_type === 'rental' ? 'selected' : '' ?>>Rental</option>

<option value="return" <?= $checkout\_type === 'return' ? 'selected' : '' ?>>Return</option>

</select>

</form>

<div id="payment-section"></div>

<?php if ($checkout\_type && !$payment\_type): ?>

<form method="post" class="checkout-section" style="margin-top: 1rem;">

<input type="hidden" name="checkout\_type" value="<?php echo $checkout\_type; ?>">

<input type="hidden" name="total\_due" value="<?php echo $totalDue; ?>">

<?php if ($checkout\_type === 'return'): ?>

<input type="hidden" name="payment\_type" value="return">

<input type="hidden" name="process\_payment" value="1">

<button class="btn" type="submit">Complete Return</button>

<?php else: ?>

<label><strong>Payment Method:</strong></label>

<select name="payment\_type" required>

<option value="">-- Select --</option>

<option value="cash">Cash</option>

<option value="card">Card</option>

<option value="split">Split</option>

</select>

<div class="button-group" style="margin-top: 1em;">

<button class="btn" type="submit">Continue</button>

</div>

<?php endif; ?>

</form>

<?php endif; ?>

<?php elseif ($customer && $games && $payment\_type): ?>

<form method="post" class="checkout-section">

<input type="hidden" name="checkout\_type" value="<?php echo $checkout\_type; ?>">

<input type="hidden" name="payment\_type" value="<?php echo $payment\_type; ?>">

<input type="hidden" name="process\_payment" value="1">

<input type="hidden" name="total\_due" value="<?php echo $totalDue; ?>">

<p><strong>Total Due: $<?php echo number\_format($totalDue, 2); ?></strong></p>

<?php if ($payment\_type === 'cash'): ?>

<div class="form-inline-row">

<label>

Enter Cash Amount:

<input type="text" name="cash\_amount" required>

</label>

</div>

<?php elseif ($payment\_type === 'card'): ?>

<div class="form-inline-row">

<label>

Card Number:

<input type="text" name="card\_number" required>

</label>

<label>

Expiry:

<input type="text" name="expiry" required>

</label>

<label>

CVV:

<input type="text" name="cvv" required>

</label>

</div>

<?php elseif ($payment\_type === 'split'): ?>

<div class="form-inline-row">

<label>

Cash Amount:

<input type="text" name="split\_cash" required>

</label>

<label>

Card Number:

<input type="text" name="split\_card\_number" required>

</label>

<label>

Expiry:

<input type="text" name="expiry" required>

</label>

<label>

CVV:

<input type="text" name="cvv" required>

</label>

</div>

<?php endif; ?>

<div class="button-group">

<button class="btn" type="submit">Process Payment</button>

</div>

</form>

<div class="button-group">

<form method="post" action="Checkout.php">

<button class="btn" type="submit">Return to Payment</button>

</form>

<form method="get" action="Checkout.php">

<input type="hidden" name="reset" value="1">

<button class="btn" type="submit">Cancel</button>

</form>

</div>

<?php endif; ?>

</main>

<script>

function handleCheckoutTypeChange() {

const type = document.getElementById('checkout\_type').value;

const section = document.getElementById('payment-section');

let html = `<form method="post" class="checkout-section" style="margin-top: 1rem;">

<input type="hidden" name="checkout\_type" value="${type}">

<input type="hidden" name="total\_due" value="<?= $totalDue ?>">

`;

if (type === 'return') {

html += `

<input type="hidden" name="payment\_type" value="return">

<input type="hidden" name="process\_payment" value="1">

<button class="btn" type="submit">Complete Return</button>

`;

} else {

html += `

<label><strong>Payment Method:</strong></label>

<select name="payment\_type" required>

<option value="">-- Select --</option>

<option value="cash">Cash</option>

<option value="card">Card</option>

<option value="split">Split</option>

</select>

<div style="margin-top: 1rem;">

<button class="btn" type="submit" style="padding: 0.75rem 2rem; font-weight: bold;">Continue</button>

</div>

`;

}

html += `</form>`;

section.innerHTML = html;

}

</script>

</body>

</html>

**Close\_Day.php:**



<?php

session\_start();

date\_default\_timezone\_set('America/New\_York');

// Session Timeout

$timeout\_duration = 900; // 15 minutes

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY']) > 1800) {

session\_unset();

session\_destroy();

header("Location: Login.php?timeout=1");

exit;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // Reset timer on activity

// Block access if not manager

if (!isset($\_SESSION['employee\_id'], $\_SESSION['role']) || $\_SESSION['role'] !== 'manager') {

header("Location: Login.php");

exit;

}

// DB Setup

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

} catch (PDOException $e) {

die("DB connection failed: " . $e->getMessage());

}

// Set selected date (default to today)

$selectedDate = $\_GET['date'] ?? date('Y-m-d');

// Check if selected date is already closed

$isClosed = false;

$stmt = $pdo->prepare("SELECT \* FROM BusinessDayStatus WHERE business\_date = ?");

$stmt->execute([$selectedDate]);

$dayRecord = $stmt->fetch();

if ($dayRecord && $dayRecord['is\_closed']) {

$isClosed = true;

}

// Confirm a till

if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['confirm\_till'])) {

$employeeId = intval($\_POST['employee\_id']);

$actual = floatval($\_POST['actual']);

$stmt = $pdo->prepare("SELECT SUM(order\_total) AS expected FROM Transactions WHERE employee\_id = ? AND DATE(order\_date) = ?");

$stmt->execute([$employeeId, $selectedDate]);

$expected = round($stmt->fetchColumn(), 2);

$variance = round($actual - $expected, 2);

// Insert or update summary

$stmt = $pdo->prepare("SELECT 1 FROM DailySummary WHERE employee\_id = ? AND summary\_date = ?");

$stmt->execute([$employeeId, $selectedDate]);

if ($stmt->fetch()) {

$stmt = $pdo->prepare("UPDATE DailySummary SET actual = ?, variance = ? WHERE employee\_id = ? AND summary\_date = ?");

$stmt->execute([$actual, $variance, $employeeId, $selectedDate]);

} else {

$stmt = $pdo->prepare("INSERT INTO DailySummary (summary\_date, expected, actual, variance, employee\_id) VALUES (?, ?, ?, ?, ?)");

$stmt->execute([$selectedDate, $expected, $actual, $variance, $employeeId]);

}

}

// Close day

if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['close\_day']) && !$isClosed) {

$drop = floatval($\_POST['drop\_total']);

$stmt = $pdo->prepare("SELECT SUM(expected) AS expected\_total FROM DailySummary WHERE summary\_date = ?");

$stmt->execute([$selectedDate]);

$expected\_total = round($stmt->fetchColumn(), 2);

$variance = round($drop - $expected\_total, 2);

$stmt = $pdo->prepare("INSERT INTO BusinessDayStatus (business\_date, drop\_total, expected\_total, variance, is\_closed, closed\_by, closed\_at) VALUES (?, ?, ?, ?, 1, ?, NOW())");

$stmt->execute([$selectedDate, $drop, $expected\_total, $variance, $\_SESSION['employee\_id']]);

$isClosed = true;

// Auto-open next day

$nextDay = date('Y-m-d', strtotime($selectedDate . ' +1 day'));

$stmt = $pdo->prepare("SELECT 1 FROM BusinessDayStatus WHERE business\_date = ?");

$stmt->execute([$nextDay]);

if (!$stmt->fetch()) {

$stmt = $pdo->prepare("INSERT INTO BusinessDayStatus (business\_date, is\_closed) VALUES (?, 0)");

$stmt->execute([$nextDay]);

}

}

// Fetch employees with sales on selected date

$stmt = $pdo->prepare("

SELECT e.employee\_id, e.username, SUM(t.order\_total) AS expected

FROM Employees e

JOIN Transactions t ON e.employee\_id = t.employee\_id

WHERE DATE(t.order\_date) = ?

GROUP BY e.employee\_id, e.username

HAVING expected > 0

ORDER BY e.employee\_id

");

$stmt->execute([$selectedDate]);

$employees = $stmt->fetchAll();

// Get summaries

$summaries = [];

$stmt = $pdo->prepare("SELECT \* FROM DailySummary WHERE summary\_date = ?");

$stmt->execute([$selectedDate]);

while ($row = $stmt->fetch()) {

$summaries[$row['employee\_id']] = $row;

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Close Day – PlayCart POS</title>

<link rel="stylesheet" href="style.css">

</head>

<body class="close-day-page">

<main>

<h1>Close Day – <?= htmlspecialchars($selectedDate) ?></h1>

<form method="GET" class="form-row-align">

<label>

View Business Day:

<input type="date" name="date" value="<?= htmlspecialchars($selectedDate) ?>">

</label>

<button type="submit" class="btn">Go</button>

</form>

<?php if ($isClosed): ?>

<p class="success"><strong>This business day has been closed.</strong></p>

<?php endif; ?>

<h3>Till Summary (Per Employee)</h3>

<table class="close-day-table">

<tr>

<th>Employee ID</th>

<th>Username</th>

<th>Expected</th>

<th>Actual</th>

<th>Variance</th>

<th>Status</th>

<th>Action</th>

</tr>

<?php foreach ($employees as $emp):

$eid = $emp['employee\_id'];

$username = htmlspecialchars($emp['username']);

$expected = round($emp['expected'], 2);

$actual = '';

$variance = '';

$status = 'Open';

$isSubmitted = false;

if (isset($summaries[$eid])) {

$actual = $summaries[$eid]['actual'];

$variance = $summaries[$eid]['variance'];

$status = 'Closed';

$isSubmitted = true;

}

$varianceClass = $variance > 0 ? 'success' : ($variance < 0 ? 'error' : '');

?>

<tr>

<td><?= $eid ?></td>

<td><?= $username ?></td>

<td>$<?= number\_format($expected, 2) ?></td>

<td>

<?php if (!$isSubmitted && !$isClosed): ?>

<form method="post" style="display: flex; align-items: center; gap: 0.5rem;">

<input type="hidden" name="employee\_id" value="<?= $eid ?>">

<input type="number" name="actual" step="0.01" required style="width: 100px;">

<?php else: ?>

$<?= number\_format($actual, 2) ?>

<?php endif; ?>

</td>

<td class="<?= $varianceClass ?>">

<?= $isSubmitted ? '$' . number\_format($variance, 2) : '-' ?>

</td>

<td>

<?= $status ?>

</td>

<td>

<?php if (!$isSubmitted && !$isClosed): ?>

<button class="btn" type="submit" name="confirm\_till">Close Till

</button>

</form>

<?php else: ?>

-

<?php endif; ?>

</td>

</tr>

<?php endforeach; ?>

</table>

<?php

$allConfirmed = true;

foreach ($employees as $emp) {

if (!isset($summaries[$emp['employee\_id']])) {

$allConfirmed = false;

break;

}

}

?>

<?php if (!$isClosed): ?>

<hr>

<h3>Finalize Drop & Close Day</h3>

<form method="post" class="form" style="display: flex; align-items: flex-end; gap: 2rem;">

<div style="display: flex; flex-direction: column;">

<label for="drop\_total" style="margin-bottom: 0.3rem;">Total Cash in Drop:</label>

<input type="number" id="drop\_total" name="drop\_total" step="0.01" required>

</div>

<button class="btn" type="submit" name="close\_day" <?= !$allConfirmed ? 'disabled' : '' ?>>

Close Day

</button>

</form>

<?php endif; ?>

<form action="Employee\_Dashboard.php" method="get" class="form">

<button class="btn" type="submit">Back to Dashboard</button>

</form>

</main>

</body>

</html>

**Employee\_Dashboard.php:**



<?php

session\_start();

date\_default\_timezone\_set('America/New\_York');

// Session Timeout

$timeout\_duration = 900; // 15 minutes

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY']) > 1800) {

session\_unset();

session\_destroy();

header("Location: Login.php?timeout=1");

exit;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // Reset timer on activity

// Handle logout

if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['logout'])) {

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

if (isset($\_SESSION['employee\_id'])) {

$log = $pdo->prepare("INSERT INTO EventLogs (employee\_id, event\_type, details) VALUES (?, 'logout', 'User manually logged out')");

$log->execute([$\_SESSION['employee\_id']]);

}

} catch (PDOException $e) {

// Silent fail for logout

}

session\_unset();

session\_destroy();

header("Location: Login.php");

exit;

}

// Block access if not logged in

if (!isset($\_SESSION['employee\_id'], $\_SESSION['username'], $\_SESSION['role'])) {

header("Location: Login.php");

exit;

}

// Connect to DB to log access

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

$log = $pdo->prepare("INSERT INTO EventLogs (employee\_id, event\_type, details) VALUES (?, 'view\_dashboard', 'Dashboard accessed')");

$log->execute([$\_SESSION['employee\_id']]);

} catch (PDOException $e) {

die("Error connecting to database: " . $e->getMessage());

}

$firstName = htmlspecialchars($\_SESSION['username']);

$role = $\_SESSION['role'];

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Employee Dashboard</title>

<link rel="stylesheet" href="style.css">

<meta http-equiv="refresh" content="900;url=Login.php">

<script>

let timeout;

function resetTimer() {

clearTimeout(timeout);

timeout = setTimeout(() => {

document.getElementById('autoLogoutForm').submit();

}, 900000); // 15 min

}

window.onload = resetTimer;

document.onmousemove = resetTimer;

document.onkeypress = resetTimer;

</script>

</head>

<body>

<main>

<img src="PlayCartPOS\_LOGO.png" alt="PlayCart POS Logo" style="max-width: 150px; display: block; margin-bottom: 1rem;">

<h1 style="color: #6e3b9f;">Welcome, <?php echo $firstName; ?>!</h1>

<p>Your role: <?php echo htmlspecialchars($role); ?></p>

<div class="button-group">

<button class="btn" onclick="location.href='Transaction\_Page.php'" aria-label="Start transaction">New Transaction</button>

<?php if ($role === 'manager'): ?>

<button class="btn" onclick="location.href='Inventory\_Management.php'" aria-label="Inventory management">Inventory Management</button>

<button class="btn" onclick="location.href='Employee\_Management.php'" aria-label="Employee management">Employee Management</button>

<button class="btn" onclick="location.href='Reports.php'" aria-label="Reports">Reports</button>

<button class="btn" onclick="location.href='Backup\_To\_USB.php'" aria-label="Backup database to USB">Backup to USB</button>

<button class="btn" onclick="location.href='Close\_Day.php'" aria-label="Close business day">Close Day</button>

<?php endif; ?>

<div>

<form method="POST">

<input type="hidden" name="logout" value="1">

<button class="btn" type="submit" aria-label="Logout">Logout</button>

</form>

</div>

</div>

<form method="POST" id="autoLogoutForm" style="display:none;">

<input type="hidden" name="logout" value="1">

</form>

</main>

</body>

</html>

**Employee\_Management.php:**



<?php

session\_start();

date\_default\_timezone\_set('America/New\_York');

// Session Timeout

$timeout\_duration = 900; // 15 minutes

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY']) > 1800) {

session\_unset();

session\_destroy();

header("Location: Login.php?timeout=1");

exit;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // Reset timer on activity

// Block access if not logged in or not manager

if (!isset($\_SESSION['employee\_id'], $\_SESSION['username'], $\_SESSION['role']) || $\_SESSION['role'] !== 'manager') {

header("Location: Login.php");

exit;

}

// DB setup

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

} catch (PDOException $e) {

die("DB error: " . $e->getMessage());

}

function toTitleCase($str) {

return ucwords(strtolower(trim($str)));

}

// Add new employee

if (isset($\_POST['add\_employee'])) {

$first = trim($\_POST['first\_name']);

$last = trim($\_POST['last\_name']);

$user = trim($\_POST['username']);

if (strlen($first) > 50 || strlen($last) > 50 || strlen($user) > 50) {

die("Error: Input too long.");

}

$pin = password\_hash($\_POST['temp\_pin'], PASSWORD\_DEFAULT);

$stmt = $pdo->prepare("INSERT INTO Employees (first\_name, last\_name, username, password\_hash, role, is\_active, password\_set) VALUES (?, ?, ?, ?, ?, 1, 0)");

$stmt->execute([

toTitleCase($first),

toTitleCase($last),

$user,

$pin,

$\_POST['role']

]);

}

// Reset password

if (isset($\_POST['reset\_password']) && $\_POST['employee\_id'] != 1) {

$newPin = password\_hash('1234', PASSWORD\_DEFAULT);

$stmt = $pdo->prepare("UPDATE Employees SET password\_hash = ?, password\_set = 0 WHERE employee\_id = ?");

$stmt->execute([$newPin, $\_POST['employee\_id']]);

}

// Update employee info

if (isset($\_POST['update\_employee']) && $\_POST['employee\_id'] != 1) {

$first = trim($\_POST['first\_name']);

$last = trim($\_POST['last\_name']);

$user = trim($\_POST['username']);

if (strlen($first) > 50 || strlen($last) > 50 || strlen($user) > 50) {

die("Error: Input too long.");

}

$stmt = $pdo->prepare("UPDATE Employees SET first\_name = ?, last\_name = ?, username = ?, role = ? WHERE employee\_id = ?");

$stmt->execute([

toTitleCase($first),

toTitleCase($last),

$user,

$\_POST['role'],

$\_POST['employee\_id']

]);

}

// Delete employee

if (isset($\_POST['delete\_employee']) && $\_POST['employee\_id'] != 1) {

// Soft delete

$stmt = $pdo->prepare("UPDATE Employees SET is\_active = 0 WHERE employee\_id = ?");

$stmt->execute([$\_POST['employee\_id']]);

// Log it to EventLogs

$logStmt = $pdo->prepare("INSERT INTO EventLogs (employee\_id, event\_type, details) VALUES (?, 'deactivate', ?)");

$logStmt->execute([

$\_SESSION['employee\_id'],

"Deactivated employee ID {$\_POST['employee\_id']}"

]);

}

// Search logic

$searchResults = [];

$filters = ['first\_name', 'last\_name', 'username', 'role'];

$search\_active = false;

if (isset($\_POST['search']) && in\_array($\_POST['filter'], $filters)) {

$filter = $\_POST['filter'];

$query = '%' . trim($\_POST['query']) . '%';

$stmt = $pdo->prepare("SELECT \* FROM Employees WHERE $filter LIKE ? AND is\_active = 1 ORDER BY employee\_id");

$stmt->execute([$query]);

$searchResults = $stmt->fetchAll();

$search\_active = true;

}

elseif (isset($\_POST['clear\_search'])) {

$search\_active = false;

}

if (!$search\_active) {

$stmt = $pdo->query("SELECT \* FROM Employees WHERE is\_active = 1 ORDER BY employee\_id");

$searchResults = $stmt->fetchAll();

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Employee Management</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<main>

<h1>Employee Management</h1>

<h3>Add New Employee</h3>

<form method="POST" class="form" style="margin-top: 2rem;">

<input type="hidden" name="add\_employee" value="1">

<!-- Grid container with two rows -->

<div style="display: grid; grid-template-columns: repeat(5, 1fr); gap: 1rem; align-items: end;">

<!-- Row 1: All fields -->

<label style="display: flex; flex-direction: column; margin: 0;">

First Name:

<input type="text" name="first\_name" required maxlength="50">

</label>

<label style="display: flex; flex-direction: column; margin: 0;">

Last Name:

<input type="text" name="last\_name" required maxlength="50">

</label>

<label style="display: flex; flex-direction: column; margin: 0;">

Username:

<input type="text" name="username" required maxlength="50">

</label>

<label style="display: flex; flex-direction: column; margin: 0;">

Temp PIN:

<input type="text" name="temp\_pin" required pattern="\d{4}" maxlength="4">

</label>

<label style="display: flex; flex-direction: column; margin: 0;">

Role:

<select name="role" required style="width: 100%;">

<option value="employee">Employee</option>

<option value="manager">Manager</option>

</select>

</label>

<!-- Row 2: 4 empty cells, then Add button under Role -->

<div></div>

<div></div>

<div></div>

<div></div>

<div>

<button class="btn" type="submit" style="width: 100%;">Add</button>

</div>

</div>

</form>

<h3>Employee List</h3>

<div style="overflow-x: hidden; width: 100%;">

<table class="employee-table">

<tr>

<th style="text-align: center;">ID</th>

<th>First</th>

<th>Last</th>

<th>Username</th>

<th>Role</th>

<th style="text-align:center;">Password Set</th>

<th style="text-align:center;">Actions</th>

</tr>

<?php foreach ($searchResults as $emp): ?>

<tr>

<form method="POST">

<td style="text-align: center;"><?= $emp['employee\_id'] ?></td>

<td><input type="text" name="first\_name" value="<?= htmlspecialchars($emp['first\_name']) ?>" <?= $emp['employee\_id'] == 1 ? 'readonly' : '' ?>></td>

<td><input type="text" name="last\_name" value="<?= htmlspecialchars($emp['last\_name']) ?>" <?= $emp['employee\_id'] == 1 ? 'readonly' : '' ?>></td>

<td><input type="text" name="username" value="<?= htmlspecialchars($emp['username']) ?>" <?= $emp['employee\_id'] == 1 ? 'readonly' : '' ?>></td>

<td>

<select name="role" <?= $emp['employee\_id'] == 1 ? 'disabled' : '' ?>>

<option value="employee" <?= $emp['role'] === 'employee' ? 'selected' : '' ?>>Employee</option>

<option value="manager" <?= $emp['role'] === 'manager' ? 'selected' : '' ?>>Manager</option>

</select>

</td>

<td style="text-align:center;">

<?= $emp['password\_set'] ? '✅' : '❌' ?>

</td>

<td style="text-align:center;">

<input type="hidden" name="employee\_id" value="<?= $emp['employee\_id'] ?>">

<?php if ($emp['employee\_id'] != 1): ?>

<div class="button-group" style="justify-content: center;">

<button class="btn" type="submit" name="update\_employee" value="1">Update</button>

<button class="btn" type="submit" name="reset\_password" value="1">Reset PIN</button>

<button class="btn" type="submit" name="delete\_employee" value="1" onclick="return confirm('Delete this employee?')">Delete</button>

</div>

<?php else: ?>

🔒

<?php endif; ?>

</td>

</form>

</tr>

<?php endforeach; ?>

</table>

</div>

<form action="Employee\_Dashboard.php" style="margin-top: 2rem;">

<button class="btn" type="submit">Back to Dashboard</button>

</form>

</main>

</body>

</html>

**Inventory\_Management.php:**

****

<?php

session\_start();

date\_default\_timezone\_set('America/New\_York');

// Session Timeout

$timeout\_duration = 900; // 15 minutes

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY']) > $timeout\_duration) {

session\_unset();

session\_destroy();

header("Location: Login.php?timeout=1");

exit;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // Reset timer on activity

// Block access if not logged in or not manager

if (!isset($\_SESSION['employee\_id'], $\_SESSION['username'], $\_SESSION['role']) || $\_SESSION['role'] !== 'manager') {

header("Location: Login.php");

exit;

}

// DB setup

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

} catch (PDOException $e) {

die("DB error: " . $e->getMessage());

}

// Add game

if (isset($\_POST['add\_game'])) {

$title = trim($\_POST['title']);

$platform = trim($\_POST['platform']);

$sku = trim($\_POST['sku']);

$type = $\_POST['game\_type'];

$sale = floatval($\_POST['sale\_price']);

$rent = floatval($\_POST['rent\_price']);

$qty = max(0, intval($\_POST['quantity\_in\_stock']));

if (strlen($title) > 100 || strlen($platform) > 30 || strlen($sku) > 30) {

die("Error: One or more inputs are too long.");

}

try {

$stmt = $pdo->prepare("INSERT INTO Games (title, platform, sku, game\_type, sale\_price, rent\_price, quantity\_in\_stock)

VALUES (?, ?, ?, ?, ?, ?, ?)");

$stmt->execute([$title, $platform, $sku, $type, $sale, $rent, $qty]);

// Redirect to avoid resubmitting on refresh

header("Location: Inventory\_Management.php?success=1");

exit;

} catch (PDOException $e) {

$errorMessage = "Error adding game: " . $e->getMessage();

}

}

// Edit Game

if (isset($\_POST['update\_game'])) {

$title = trim($\_POST['title']);

$platform = trim($\_POST['platform']);

$sku = trim($\_POST['sku']);

$type = $\_POST['game\_type'];

$sale = floatval($\_POST['sale\_price']);

$rent = floatval($\_POST['rent\_price']);

$qty = max(0, intval($\_POST['quantity\_in\_stock']));

$game\_id = $\_POST['game\_id'];

if ($title === '' || $platform === '' || $sku === '') {

$errorMessage = "Error: Title, Platform, and SKU cannot be blank.";

} elseif (strlen($title) > 100 || strlen($platform) > 30 || strlen($sku) > 30) {

$errorMessage = "Error: One or more inputs are too long.";

} else {

try {

$stmt = $pdo->prepare("UPDATE Games SET title=?, platform=?, sku=?, game\_type=?, sale\_price=?, rent\_price=?, quantity\_in\_stock=? WHERE game\_id=?");

$stmt->execute([$title, $platform, $sku, $type, $sale, $rent, $qty, $game\_id]);

$successMessage = "Game updated successfully.";

} catch (PDOException $e) {

$errorMessage = "Update failed: " . $e->getMessage();

}

}

}

// Delete game

if (isset($\_POST['delete\_game'])) {

$stmt = $pdo->prepare("DELETE FROM Games WHERE game\_id = ?");

$stmt->execute([$\_POST['game\_id']]);

}

// Filters

$search = $\_GET['search'] ?? '';

$platformFilter = $\_GET['platform'] ?? '';

$typeFilter = $\_GET['game\_type'] ?? '';

$conditions = [];

$params = [];

if ($search !== '') {

$conditions[] = "(title LIKE ? OR platform LIKE ? OR sku LIKE ?)";

$params[] = "%$search%";

$params[] = "%$search%";

$params[] = "%$search%";

}

if ($platformFilter !== '') {

$conditions[] = "platform = ?";

$params[] = $platformFilter;

}

if ($typeFilter !== '') {

$conditions[] = "game\_type = ?";

$params[] = $typeFilter;

}

$sql = "SELECT \* FROM Games";

if ($conditions) {

$sql .= " WHERE " . implode(" AND ", $conditions);

}

$sql .= " ORDER BY title";

$stmt = $pdo->prepare($sql);

$stmt->execute($params);

$games = $stmt->fetchAll();

// For filter dropdowns

$platforms = $pdo->query("SELECT DISTINCT platform FROM Games ORDER BY platform")->fetchAll(PDO::FETCH\_COLUMN);

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Inventory Management</title>

<link rel="stylesheet" href="style.css">

</head>

<body class="inventory-page" style="flex-direction: row;">

<main>

<?php if (!empty($successMessage)): ?>

<p class="success-message" style="color: green; font-weight: bold; margin-bottom: 1rem;">

<?= htmlspecialchars($successMessage) ?>

</p>

<?php endif; ?>

<h1>Inventory Management</h1>

<?php if (isset($\_GET['success']) && $\_GET['success'] == 1): ?>

<p class="success-message" style="color: green; font-weight: bold; margin-bottom: 1rem;">

Game added successfully!

</p>

<?php endif; ?>

<?php if (!empty($errorMessage)): ?>

<p class="error-message" style="color: red; font-weight: bold; margin-bottom: 1rem;">

<?= htmlspecialchars($errorMessage) ?>

</p>

<?php endif; ?>

<h3>Search Inventory</h3>

<form method="GET" class="form">

<label>

Search:

<input type="text" name="search" placeholder="Search by title/platform/SKU" value="<?= htmlspecialchars($search) ?>">

</label>

<label>

Platform:

<select name="platform">

<option value="">All Platforms</option>

<?php foreach ($platforms as $p): ?>

<option value="<?= htmlspecialchars($p) ?>" <?= $p === $platformFilter ? 'selected' : '' ?>>

<?= htmlspecialchars($p) ?>

</option>

<?php endforeach; ?>

</select>

</label>

<label>

Type:

<select name="game\_type">

<option value="">All Types</option>

<option value="sale" <?= $typeFilter === 'sale' ? 'selected' : '' ?>>Sale Only</option>

<option value="rental" <?= $typeFilter === 'rental' ? 'selected' : '' ?>>Rental Only</option>

<option value="both" <?= $typeFilter === 'both' ? 'selected' : '' ?>>Both</option>

</select>

</label>

<div class="btn-group-top">

<button type="submit" class="btn">Search</button>

<button type="button" class="btn" onclick="window.location.href='Inventory\_Management.php'">Clear</button>

</div>

</form>

<h3>Add New Game</h3>

<form method="POST" class="form">

<input type="hidden" name="add\_game" value="1">

<label>

Title:

<input type="text" name="title" required maxlength="100">

</label>

<label>

Platform:

<input type="text" name="platform" required maxlength="30">

</label>

<label>

SKU:

<input type="text" name="sku" required maxlength="30">

</label>

<label>

Type:

<select name="game\_type" required>

<option value="sale">Sale</option>

<option value="rental">Rental</option>

<option value="both">Both</option>

</select>

</label>

<label>

Sale Price:

<input type="text" name="sale\_price" required>

</label>

<label>

Rent Price:

<input type="text" name="rent\_price" required>

</label>

<label>

Quantity:

<input type="number" name="quantity\_in\_stock" min="0" required>

</label>

<div class="top-button-row">

<button type="submit" class="btn">Add Game</button>

</div>

</form>

<h3>Current Inventory</h3>

<div style="display: flex; flex-direction: row; overflow-x: auto;">

<table class="inventory-table" style="min-width: 1000px;">

<tr>

<th>Title</th>

<th>Platform</th>

<th>SKU</th>

<th>Type</th>

<th>Sale Price</th>

<th>Rent Price</th>

<th>Qty</th>

<th>Actions</th>

</tr>

<?php foreach ($games as $game): ?>

<tr>

<form method="POST">

<td><input type="text" name="title" maxlength="100" value="<?= htmlspecialchars($game['title']) ?>"></td>

<td><input type="text" name="platform" maxlength="30" value="<?= htmlspecialchars($game['platform']) ?>"></td>

<td><input type="text" name="sku" maxlength="30" value="<?= htmlspecialchars($game['sku']) ?>"></td>

<td>

<select name="game\_type">

<option value="sale" <?= $game['game\_type'] === 'sale' ? 'selected' : '' ?>>Sale</option>

<option value="rental" <?= $game['game\_type'] === 'rental' ? 'selected' : '' ?>>Rental</option>

<option value="both" <?= $game['game\_type'] === 'both' ? 'selected' : '' ?>>Both</option>

</select>

</td>

<td><input type="text" name="sale\_price" value="<?= $game['sale\_price'] ?>"></td>

<td><input type="text" name="rent\_price" value="<?= $game['rent\_price'] ?>"></td>

<td>

<input type="number" name="quantity\_in\_stock" value="<?= $game['quantity\_in\_stock'] ?>" min="0">

<?php if ($game['quantity\_in\_stock'] == 0): ?>

<span class="warning-out">OUT OF STOCK</span>

<?php elseif ($game['quantity\_in\_stock'] < 3): ?>

<span class="warning-low">LOW</span>

<?php endif; ?>

</td>

<td>

<input type="hidden" name="game\_id" value="<?= $game['game\_id'] ?>">

<div class="button-group">

<button class="btn" type="submit" name="update\_game" value="1">Update</button>

<button class="btn" type="submit" name="delete\_game" value="1" onclick="return confirm('Delete this game?')">Delete</button>

</div>

</td>

</form>

</tr>

<?php endforeach; ?>

</table>

</div>

<form action="Employee\_Dashboard.php">

<div class="top-button-row">

<button class="btn" type="submit">Back to Dashboard</button>

</div>

</form>

</main>

</body>

</html>

**Login.php:**

****

<?php

ini\_set('display\_errors', 1);

ini\_set('display\_startup\_errors', 1);

error\_reporting(E\_ALL);

session\_start();

date\_default\_timezone\_set('America/New\_York');

// DB connection

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

} catch (PDOException $e) {

die("Database connection failed: " . $e->getMessage());

}

// Logic state

$login\_error = '';

$show\_pin\_reset = false;

$employee\_id = '';

$temp\_user = null;

// Session-based brute force tracking

$max\_attempts = 5; // Max attempts before locking

$lock\_time = 900; // Lock time (15 minutes)

// Initialize or reset session tracking

if (!isset($\_SESSION['attempts'])) {

$\_SESSION['attempts'] = 0;

$\_SESSION['first\_attempt\_time'] = time();

}

// Handle POST

if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {

$employee\_id = $\_POST['employee\_id'] ?? '';

$pin = $\_POST['pin'] ?? '';

// Check if brute force protection is triggered

if ($\_SESSION['attempts'] >= $max\_attempts && (time() - $\_SESSION['first\_attempt\_time']) < $lock\_time) {

$login\_error = "Too many failed attempts. Please try again later.";

} else {

// Check if employee exists

$stmt = $pdo->prepare("SELECT \* FROM Employees WHERE employee\_id = :employee\_id AND is\_active = 1");

$stmt->execute(['employee\_id' => $employee\_id]);

$user = $stmt->fetch();

if ($user) {

if (password\_verify($pin, $user['password\_hash'])) {

// Reset failed attempts on successful login

$\_SESSION['attempts'] = 0;

$\_SESSION['employee\_id'] = $user['employee\_id'];

$\_SESSION['role'] = $user['role'];

$\_SESSION['username'] = $user['username'];

if ($user['password\_set'] == 0) {

$show\_pin\_reset = true;

} else {

// Log success

$pdo->prepare("INSERT INTO EventLogs (employee\_id, event\_type, details) VALUES (?, 'login', 'Login successful')")

->execute([$user['employee\_id']]);

header("Location: Employee\_Dashboard.php");

exit;

}

} else {

// Increment failed attempts and log the failed attempt

$\_SESSION['attempts']++;

if ($\_SESSION['attempts'] == 1) {

$\_SESSION['first\_attempt\_time'] = time(); // Start timer for first failed attempt

}

$login\_error = "Invalid credentials.";

// Log failed login attempt, using valid employee\_id even on failure

$details = "Login failed for Employee ID: " . htmlspecialchars($employee\_id);

$stmt = $pdo->prepare("INSERT INTO EventLogs (employee\_id, event\_type, details) VALUES (?, 'login\_fail', ?)");

$stmt->execute([$user['employee\_id'], $details]);

}

} else {

$login\_error = "Invalid credentials.";

}

}

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login - PlayCart POS</title>

<link rel="stylesheet" href="style.css">

</head>

<body>

<main>

<img src="PlayCartPOS\_LOGO.png" alt="PlayCart POS Logo" style="max-width: 150px; display: block; margin-bottom: 1rem;">

<h1 style="color: #6e3b9f;">Employee Login</h1>

<?php if ($login\_error): ?>

<p class="error" role="alert"><?= htmlspecialchars($login\_error) ?></p>

<?php endif; ?>

<?php if ($show\_pin\_reset): ?>

<form method="POST" action="Login.php" class="form">

<input type="hidden" name="set\_new\_pin" value="1">

<input type="hidden" name="employee\_id" value="<?= htmlspecialchars($employee\_id) ?>">

<label for="new\_pin">New 4-digit PIN:</label>

<input type="password" name="new\_pin" id="new\_pin" required maxlength="4" pattern="\d{4}">

<label for="confirm\_pin">Confirm PIN:</label>

<input type="password" name="confirm\_pin" id="confirm\_pin" required maxlength="4" pattern="\d{4}">

<button type="submit" class="btn">Set PIN and Continue</button>

</form>

<?php else: ?>

<form method="POST" action="Login.php" autocomplete="off" class="form">

<label for="employee\_id">Employee ID:</label>

<input type="text" name="employee\_id" id="employee\_id" required pattern="\d+" inputmode="numeric" maxlength="6">

<label for="pin">4-Digit PIN:</label>

<input type="password" name="pin" id="pin" required pattern="\d{4}" inputmode="numeric" maxlength="4" autocomplete="off">

<button type="submit" class="btn">Login</button>

</form>

<?php endif; ?>

</main>

</body>

</html>

**Reports.php:**



<?php

session\_start();

date\_default\_timezone\_set('America/New\_York');

// Session Timeout

$timeout\_duration = 900; // 15 minutes

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY']) > 1800) {

session\_unset();

session\_destroy();

header("Location: Login.php?timeout=1");

exit;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // Reset timer on activity

if (!isset($\_SESSION['employee\_id'], $\_SESSION['username'], $\_SESSION['role'])) {

header("Location: Login.php");

exit;

}

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

} catch (\PDOException $e) {

die("Database error: " . $e->getMessage());

}

// Filter inputs

$params = [];

$start\_date = $\_GET['start\_date'] ?? null;

$end\_date = $\_GET['end\_date'] ?? null;

$employee\_id = $\_GET['employee\_id'] ?? null;

$username = $\_GET['username'] ?? null;

$transaction\_type = $\_GET['transaction\_type'] ?? null;

$game\_title = $\_GET['game\_title'] ?? null;

$platform = $\_GET['platform'] ?? null;

// Build filters for transactions (Transactions, Game Perf, Customer Spend)

$filters\_tx = [];

if ($start\_date && $end\_date) {

$filters\_tx[] = "DATE(t.order\_date) BETWEEN :start\_date AND :end\_date";

$params['start\_date'] = $start\_date;

$params['end\_date'] = $end\_date;

}

if ($employee\_id) {

$filters\_tx[] = "t.employee\_id = :employee\_id";

$params['employee\_id'] = $employee\_id;

}

if ($username) {

$filters\_tx[] = "e.username LIKE :username";

$params['username'] = "%$username%";

}

if ($transaction\_type) {

$filters\_tx[] = "t.transaction\_type = :transaction\_type";

$params['transaction\_type'] = $transaction\_type;

}

if ($game\_title) {

$filters\_tx[] = "g.title LIKE :game\_title";

$params['game\_title'] = "%$game\_title%";

}

if ($platform) {

$filters\_tx[] = "g.platform LIKE :platform";

$params['platform'] = "%$platform%";

}

$whereSQL = $filters\_tx ? 'WHERE ' . implode(' AND ', $filters\_tx) : '';

// Filters for Daily Summary (no game filters)

$filters\_summary = [];

$summaryParams = [];

if ($start\_date && $end\_date) {

$filters\_summary[] = "ds.summary\_date BETWEEN :start\_date AND :end\_date";

$summaryParams['start\_date'] = $start\_date;

$summaryParams['end\_date'] = $end\_date;

}

if ($employee\_id) {

$filters\_summary[] = "ds.employee\_id = :employee\_id";

$summaryParams['employee\_id'] = $employee\_id;

}

$whereSummary = $filters\_summary ? 'WHERE ' . implode(' AND ', $filters\_summary) : '';

// Filters for BusinessDayStatus (only employee name)

$filters\_days = [];

$dayParams = [];

if ($start\_date && $end\_date) {

$filters\_days[] = "b.business\_date BETWEEN :start\_date AND :end\_date";

$dayParams['start\_date'] = $start\_date;

$dayParams['end\_date'] = $end\_date;

}

if ($username) {

$filters\_days[] = "e.username LIKE :day\_username";

$dayParams['day\_username'] = "%$username%";

}

$whereDay = $filters\_days ? 'WHERE ' . implode(' AND ', $filters\_days) : '';

// Transactions

$txQuery = "

SELECT t.\*, g.title AS game\_title, g.platform, e.username

FROM Transactions t

JOIN Games g ON t.game\_id = g.game\_id

JOIN Employees e ON t.employee\_id = e.employee\_id

$whereSQL

ORDER BY t.order\_date DESC

";

$stmt = $pdo->prepare($txQuery);

$stmt->execute($params);

$transactions = $stmt->fetchAll();

$totalAmount = array\_reduce($transactions, fn($carry, $row) => $carry + $row['order\_total'], 0);

// Daily Summary

$summaryFilters = [];

$summaryParams = [];

if (!empty($\_GET['start\_date']) && !empty($\_GET['end\_date'])) {

$summaryFilters[] = "ds.summary\_date BETWEEN :start\_date AND :end\_date";

$summaryParams['start\_date'] = $\_GET['start\_date'];

$summaryParams['end\_date'] = $\_GET['end\_date'];

}

if (!empty($\_GET['employee\_id'])) {

$summaryFilters[] = "ds.employee\_id = :employee\_id";

$summaryParams['employee\_id'] = $\_GET['employee\_id'];

}

$whereSummary = $summaryFilters ? 'WHERE ' . implode(' AND ', $summaryFilters) : '';

$summaryStmt = $pdo->prepare("

SELECT ds.\*, e.username

FROM DailySummary ds

JOIN Employees e ON ds.employee\_id = e.employee\_id

$whereSummary

ORDER BY ds.summary\_date DESC, ds.employee\_id

");

$summaryStmt->execute($summaryParams);

$summaries = $summaryStmt->fetchAll();

// Closed Business Days

$dayFilters = [];

$dayParams = [];

if (!empty($\_GET['start\_date']) && !empty($\_GET['end\_date'])) {

$dayFilters[] = "b.business\_date BETWEEN :start\_date AND :end\_date";

$dayParams['start\_date'] = $\_GET['start\_date'];

$dayParams['end\_date'] = $\_GET['end\_date'];

}

if (!empty($\_GET['username'])) {

$dayFilters[] = "e.username LIKE :day\_username";

$dayParams['day\_username'] = '%' . $\_GET['username'] . '%';

}

$whereDay = $dayFilters ? 'WHERE ' . implode(' AND ', $dayFilters) : '';

$dayStmt = $pdo->prepare("

SELECT b.\*, e.username

FROM BusinessDayStatus b

JOIN Employees e ON b.closed\_by = e.employee\_id

$whereDay

ORDER BY b.business\_date DESC

");

$dayStmt->execute($dayParams);

$dayStatus = $dayStmt->fetchAll();

// Customer Spending Summary

$customerQuery = "

SELECT c.first\_name, c.last\_name, c.email,

COUNT(t.transaction\_id) AS total\_transactions,

ROUND(SUM(t.order\_total), 2) AS total\_spent

FROM Transactions t

JOIN Customer c ON t.customer\_id = c.customer\_id

JOIN Employees e ON t.employee\_id = e.employee\_id

JOIN Games g ON t.game\_id = g.game\_id

$whereSQL

GROUP BY t.customer\_id

ORDER BY total\_spent DESC

";

$customerStmt = $pdo->prepare($customerQuery);

$customerStmt->execute($params);

$customerData = $customerStmt->fetchAll();

// Game Performance Summary

$gameQuery = "

SELECT g.title, g.platform,

COUNT(t.transaction\_id) AS total\_transactions,

ROUND(SUM(t.order\_total), 2) AS total\_earned

FROM Transactions t

JOIN Games g ON t.game\_id = g.game\_id

JOIN Employees e ON t.employee\_id = e.employee\_id

$whereSQL

GROUP BY t.game\_id

ORDER BY total\_transactions DESC

";

$gameStmt = $pdo->prepare($gameQuery);

$gameStmt->execute($params);

$gameData = $gameStmt->fetchAll();

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Reports - PlayCart POS</title>

<link rel="stylesheet" href="style.css">

<style>

body { font-family: Arial, sans-serif; padding: 1rem; }

table { border-collapse: collapse; width: 100%; margin-top: 1rem; }

th, td { border: 1px solid #ccc; padding: 8px; text-align: left; }

th { background-color: #f2f2f2; }

.section { margin-top: 2rem; }

.filters form { display: flex; flex-wrap: wrap; gap: 1rem; align-items: flex-end; }

.filters div { display: flex; flex-direction: column; }

.filters button { padding: 6px 12px; }

.red { color: red; }

.green { color: green; }

.black { color: black; }

</style>

</head>

<body>

<main>

<h1>Reports Dashboard</h1>

<!-- FILTER FORM -->

<section class="filters">

<h3>Filter Reports</h3>

<form method="GET">

<div><label>Start Date:<input type="date" name="start\_date" value="<?= htmlspecialchars($\_GET['start\_date'] ?? '') ?>"></label></div>

<div><label>End Date:<input type="date" name="end\_date" value="<?= htmlspecialchars($\_GET['end\_date'] ?? '') ?>"></label></div>

<div><label>Employee ID:<input type="text" name="employee\_id" pattern="\d\*" value="<?= htmlspecialchars($\_GET['employee\_id'] ?? '') ?>"></label></div>

<div><label>Username:<input type="text" name="username" value="<?= htmlspecialchars($\_GET['username'] ?? '') ?>"></label></div>

<div><label>Game Title:<input type="text" name="game\_title" value="<?= htmlspecialchars($\_GET['game\_title'] ?? '') ?>"></label></div>

<div><label>Platform:<input type="text" name="platform" value="<?= htmlspecialchars($\_GET['platform'] ?? '') ?>"></label></div>

<div>

<label>Transaction Type:

<select name="transaction\_type">

<option value="">All</option>

<option value="purchase" <?= ($\_GET['transaction\_type'] ?? '') === 'purchase' ? 'selected' : '' ?>>Purchase</option>

<option value="rental" <?= ($\_GET['transaction\_type'] ?? '') === 'rental' ? 'selected' : '' ?>>Rental</option>

<option value="return" <?= ($\_GET['transaction\_type'] ?? '') === 'return' ? 'selected' : '' ?>>Return</option>

</select>

</label>

</div>

<label class="button-row">

<span style="min-width: 180px;">&nbsp;</span>

<button class="btn" type="submit" name="apply\_filters">Apply Filters</button>

<button class="btn" type="submit" name="clear\_filters">Clear All Filters</button>

</label>

</form>

</section>

<!-- TRANSACTION RESULTS -->

<section class="section">

<h3>Transaction Results (<?= count($transactions) ?> entries)</h3>

<table>

<thead>

<tr>

<th>Date</th>

<th>Type</th>

<th>Employee ID</th>

<th>Username</th>

<th>Game Title</th>

<th>Platform</th>

<th>Order Total ($)</th>

</tr>

</thead>

<tbody>

<?php if (empty($transactions)): ?>

<tr><td colspan="7">No matching records found.</td></tr>

<?php else: ?>

<?php foreach ($transactions as $t): ?>

<tr>

<td><?= htmlspecialchars($t['order\_date']) ?></td>

<td><?= ucfirst($t['transaction\_type']) ?></td>

<td><?= $t['employee\_id'] ?></td>

<td><?= htmlspecialchars($t['username']) ?></td>

<td><?= htmlspecialchars($t['game\_title']) ?></td>

<td><?= htmlspecialchars($t['platform']) ?></td>

<td>$<?= number\_format($t['order\_total'], 2) ?></td>

</tr>

<?php endforeach; ?>

<tr>

<td colspan="6"><strong>Total Sales:</strong></td>

<td><strong>$<?= number\_format($totalAmount, 2) ?></strong></td>

</tr>

<?php endif; ?>

</tbody>

</table>

</section>

<!-- CUSTOMER SPENDING SUMMARY -->

<section class="section">

<h3>Customer Spending Summary</h3>

<table>

<thead>

<tr>

<th>Customer Name</th>

<th>Email</th>

<th>Total Transactions</th>

<th>Total Spent ($)</th>

</tr>

</thead>

<tbody>

<?php if (empty($customerData)): ?>

<tr><td colspan="4">No customer data found for these filters.</td></tr>

<?php else: ?>

<?php foreach ($customerData as $row): ?>

<tr>

<td><?= htmlspecialchars("{$row['first\_name']} {$row['last\_name']}") ?></td>

<td><?= htmlspecialchars($row['email']) ?></td>

<td><?= $row['total\_transactions'] ?></td>

<td>$<?= number\_format($row['total\_spent'], 2) ?></td>

</tr>

<?php endforeach; ?>

<tr>

<td colspan="3"><strong>Total Customer Spending:</strong></td>

<td><strong>$<?= number\_format(array\_sum(array\_column($customerData, 'total\_spent')), 2) ?></strong></td>

</tr>

<?php endif; ?>

</tbody>

</table>

</section>

<!-- GAME PERFORMANCE SUMMARY -->

<section class="section">

<h3>Game Performance Summary</h3>

<table>

<thead>

<tr>

<th>Game Title</th>

<th>Platform</th>

<th>Total Transactions</th>

<th>Total Revenue ($)</th>

</tr>

</thead>

<tbody>

<?php if (empty($gameData)): ?>

<tr><td colspan="4">No game data found for these filters.</td></tr>

<?php else: ?>

<?php foreach ($gameData as $row): ?>

<tr>

<td><?= htmlspecialchars($row['title']) ?></td>

<td><?= htmlspecialchars($row['platform']) ?></td>

<td><?= $row['total\_transactions'] ?></td>

<td>$<?= number\_format($row['total\_earned'], 2) ?></td>

</tr>

<?php endforeach; ?>

<tr>

<td colspan="3"><strong>Total Game Revenue:</strong></td>

<td><strong>$<?= number\_format(array\_sum(array\_column($gameData, 'total\_earned')), 2) ?></strong></td>

</tr>

<?php endif; ?>

</tbody>

</table>

</section>

<!-- DAILY SUMMARY -->

<section class="section">

<h3>Daily Summary Per Employee</h3>

<table>

<thead>

<tr>

<th>Date</th>

<th>Employee ID</th>

<th>Username</th>

<th>Expected</th>

<th>Actual</th>

<th>Variance</th>

</tr>

</thead>

<tbody>

<?php if (empty($summaries)): ?>

<tr><td colspan="6">No matching summaries found.</td></tr>

<?php else: ?>

<?php foreach ($summaries as $s):

$v = floatval($s['variance']);

$color = $v > 0 ? 'green' : ($v < 0 ? 'red' : 'black');

?>

<tr>

<td><?= $s['summary\_date'] ?></td>

<td><?= $s['employee\_id'] ?></td>

<td><?= htmlspecialchars($s['username']) ?></td>

<td>$<?= number\_format($s['expected'], 2) ?></td>

<td>$<?= number\_format($s['actual'], 2) ?></td>

<td class="<?= $color ?>">$<?= number\_format($v, 2) ?></td>

</tr>

<?php endforeach; ?>

<tr>

<td colspan="3"><strong>Total:</strong></td>

<td><strong>$<?= number\_format(array\_sum(array\_column($summaries, 'expected')), 2) ?></strong></td>

<td><strong>$<?= number\_format(array\_sum(array\_column($summaries, 'actual')), 2) ?></strong></td>

<td></td>

</tr>

<?php endif; ?>

</tbody>

</table>

</section>

<!-- CLOSED DAYS -->

<section class="section">

<h3>Closed Business Days</h3>

<table>

<thead>

<tr>

<th>Date</th>

<th>Expected Total</th>

<th>Drop Total</th>

<th>Variance</th>

<th>Closed By</th>

<th>Status</th>

</tr>

</thead>

<tbody>

<?php if (empty($dayStatus)): ?>

<tr><td colspan="6">No closed days found.</td></tr>

<?php else: ?>

<?php foreach ($dayStatus as $d):

$color = $d['variance'] > 0 ? 'green' : ($d['variance'] < 0 ? 'red' : 'black');

?>

<tr>

<td><?= $d['business\_date'] ?></td>

<td>$<?= number\_format($d['expected\_total'], 2) ?></td>

<td>$<?= number\_format($d['drop\_total'], 2) ?></td>

<td class="<?= $color ?>">$<?= number\_format($d['variance'], 2) ?></td>

<td><?= htmlspecialchars($d['username']) ?> (<?= $d['closed\_by'] ?>)</td>

<td>Closed</td>

</tr>

<?php endforeach; ?>

<tr>

<td><strong>Total:</strong></td>

<td><strong>$<?= number\_format(array\_sum(array\_column($dayStatus, 'expected\_total')), 2) ?></strong></td>

<td><strong>$<?= number\_format(array\_sum(array\_column($dayStatus, 'drop\_total')), 2) ?></strong></td>

<td><strong>$<?= number\_format(array\_sum(array\_column($dayStatus, 'variance')), 2) ?></strong></td>

<td colspan="2"></td>

</tr>

<?php endif; ?>

</tbody>

</table>

</section>

<form action="Employee\_Dashboard.php" method="get" style="text-align: center;">

<button class="btn" type="submit">Back to Dashboard</button>

</form>

</main>

</body>

</html>

**Transaction\_Page.php:**



<?php

session\_start();

date\_default\_timezone\_set('America/New\_York');

// Session Timeout

$timeout\_duration = 900; // 15 minutes

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY']) > 1800) {

session\_unset();

session\_destroy();

header("Location: Login.php?timeout=1");

exit;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // Reset timer on activity

// Block access if not logged in

if (!isset($\_SESSION['employee\_id'], $\_SESSION['username'], $\_SESSION['role'])) {

header("Location: Login.php");

exit;

}

// DB connection

$host = 'localhost';

$db = 'PlayCart';

$user = 'playcart\_admin';

$pass = 'PlayCartAdmin2025!';

$charset = 'utf8mb4';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

];

try {

$pdo = new PDO($dsn, $user, $pass, $options);

} catch (PDOException $e) {

die("Database connection failed: " . $e->getMessage());

}

// Reset transaction

if (isset($\_GET['reset'])) {

unset($\_SESSION['selected\_customer'], $\_SESSION['selected\_games']);

header("Location: Transaction\_Page.php");

exit;

}

$searchResults = [];

$gameResults = [];

$newCustomerMessage = '';

$customerType = '';

$gameSearch = false;

function toTitleCase($string) {

return ucwords(strtolower(trim($string)));

}

// Add new customer

if ($\_SERVER['REQUEST\_METHOD'] === 'POST' && isset($\_POST['add\_customer'])) {

$area = $\_POST['area\_code'];

$prefix = $\_POST['prefix'];

$line = $\_POST['line\_number'];

if (!preg\_match('/^\d{3}$/', $area) || !preg\_match('/^\d{3}$/', $prefix) || !preg\_match('/^\d{4}$/', $line)) {

$newCustomerMessage = "❌ Invalid phone number format.";

} else {

$phone = "($area) $prefix-$line";

$stmt = $pdo->prepare("INSERT INTO Customer (first\_name, last\_name, address, email, phone\_number) VALUES (?, ?, ?, ?, ?)");

$stmt->execute([

toTitleCase($\_POST['first\_name']),

toTitleCase($\_POST['last\_name']),

toTitleCase($\_POST['address']),

$\_POST['email'],

$phone

]);

$newCustomerMessage = "✅ Customer added successfully.";

$customerType = 'new';

}

}

// Customer selection and search

if (isset($\_POST['select\_customer\_id'])) {

$\_SESSION['selected\_customer'] = $\_POST['select\_customer\_id'];

$customerType = 'existing';

}

if (isset($\_POST['search\_customer'])) {

$customerType = 'existing';

$filter = $\_POST['filter'];

$query = '%' . $\_POST['query'] . '%';

if ($filter === 'name') {

$stmt = $pdo->prepare("SELECT \* FROM Customer WHERE first\_name LIKE ? OR last\_name LIKE ?");

$stmt->execute([$query, $query]);

} else {

$stmt = $pdo->prepare("SELECT \* FROM Customer WHERE $filter LIKE ?");

$stmt->execute([$query]);

}

$searchResults = $stmt->fetchAll();

}

// Game search

if (isset($\_POST['search\_game'])) {

$gameSearch = true;

$filter = $\_POST['filter'];

$query = '%' . $\_POST['query'] . '%';

$stmt = $pdo->prepare("SELECT \* FROM Games WHERE $filter LIKE ?");

$stmt->execute([$query]);

$gameResults = $stmt->fetchAll();

}

// Game selection and removal

if (isset($\_POST['select\_game\_id'])) {

$selectedId = $\_POST['select\_game\_id'];

if (!isset($\_SESSION['selected\_games']) || !is\_array($\_SESSION['selected\_games'])) {

$\_SESSION['selected\_games'] = [];

}

if (!isset($\_SESSION['selected\_games'][$selectedId])) {

$\_SESSION['selected\_games'][$selectedId] = 1;

} else {

$\_SESSION['selected\_games'][$selectedId]++;

}

}

if (isset($\_POST['remove\_game\_id'])) {

$\_SESSION['selected\_games'] = array\_diff($\_SESSION['selected\_games'], [$\_POST['remove\_game\_id']]);

}

// Fetch selected customer and game info

$selectedCustomer = null;

$selectedGames = [];

if (isset($\_SESSION['selected\_customer'])) {

$stmt = $pdo->prepare("SELECT \* FROM Customer WHERE customer\_id = ?");

$stmt->execute([$\_SESSION['selected\_customer']]);

$selectedCustomer = $stmt->fetch();

}

if (!isset($\_SESSION['selected\_games']) || !is\_array($\_SESSION['selected\_games'])) {

$\_SESSION['selected\_games'] = [];

}

$selectedGames = [];

if (!empty($\_SESSION['selected\_games']) && is\_array($\_SESSION['selected\_games'])) {

$gameIds = array\_keys($\_SESSION['selected\_games']);

$placeholders = implode(',', array\_fill(0, count($gameIds), '?'));

$stmt = $pdo->prepare("SELECT \* FROM Games WHERE game\_id IN ($placeholders)");

$stmt->execute($gameIds);

$fetchedGames = $stmt->fetchAll();

foreach ($fetchedGames as $game) {

$game['quantity'] = $\_SESSION['selected\_games'][$game['game\_id']];

$selectedGames[] = $game;

}

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Begin New Transaction</title>

<link rel="stylesheet" href="style.css">

<script>

function toggleCustomerSection() {

const type = document.getElementById('customer\_type').value;

document.getElementById('existing\_customer').style.display = (type === 'existing') ? 'block' : 'none';

document.getElementById('new\_customer').style.display = (type === 'new') ? 'block' : 'none';

}

</script>

</head>

<body onload="document.getElementById('customer\_type').value='<?php echo $customerType; ?>'; toggleCustomerSection();">

<main>

<img src="PlayCartPOS\_LOGO.png" alt="PlayCart POS Logo" style="max-width: 150px; display: block; margin-bottom: 1rem;">

<h1 style="color: #6e3b9f;">Begin New Transaction</h1>

<label for="customer\_type">Customer Type:</label>

<select id="customer\_type" name="customer\_type" onchange="toggleCustomerSection()">

<option value="">-- Select --</option>

<option value="existing">Existing Customer</option>

<option value="new">New Customer</option>

</select>

<!-- Existing Customer -->

<section id="existing\_customer" style="display:none; margin-top:1em;">

<h3>Find Existing Customer</h3>

<form method="POST">

<input type="hidden" name="search\_customer" value="1">

<label>Search by:</label>

<select name="filter">

<option value="name">Name</option>

<option value="email">Email</option>

<option value="address">Address</option>

<option value="phone\_number">Phone Number</option>

</select>

<input type="text" name="query" required>

<button class="btn" type="submit">Search</button>

</form>

<?php if (isset($\_POST['search\_customer'])): ?>

<?php if (empty($searchResults)): ?>

<p style="color:red;">Customer not found.</p>

<?php else: ?>

<ul>

<?php foreach ($searchResults as $cust): ?>

<li>

<strong><?php echo htmlspecialchars("{$cust['first\_name']} {$cust['last\_name']}"); ?></strong><br>

Email: <?php echo htmlspecialchars($cust['email']); ?><br>

Phone: <?php echo htmlspecialchars($cust['phone\_number']); ?><br>

Address: <?php echo htmlspecialchars($cust['address']); ?><br>

<form method="POST" style="display:inline;">

<input type="hidden" name="select\_customer\_id" value="<?php echo $cust['customer\_id']; ?>">

<button class="btn" type="submit">Select</button>

</form>

<hr>

</li>

<?php endforeach; ?>

</ul>

<?php endif; ?>

<?php endif; ?>

</section>

<!-- New Customer -->

<section id="new\_customer" style="display:none; margin-top:1em;">

<h3>New Customer Info</h3>

<?php if ($newCustomerMessage): ?>

<p style="color:green;"><?php echo $newCustomerMessage; ?></p>

<?php endif; ?>

<form method="POST">

<input type="hidden" name="add\_customer" value="1">

<label>First Name: <input type="text" name="first\_name" required></label><br>

<label>Last Name: <input type="text" name="last\_name" required></label><br>

<label>Address: <input type="text" name="address"></label><br>

<label>Email: <input type="email" name="email" required></label><br>

<label>Phone Number: </label>

(<input type="text" name="area\_code" maxlength="3" pattern="\d{3}" required>)

<input type="text" name="prefix" maxlength="3" pattern="\d{3}" required> -

<input type="text" name="line\_number" maxlength="4" pattern="\d{4}" required><br>

<button class="btn" type="submit">Add Customer</button>

</form>

</section>

<!-- Game Search -->

<?php if ($selectedCustomer): ?>

<section style="margin-top:2em;">

<h3>Search for Game</h3>

<form method="POST">

<input type="hidden" name="search\_game" value="1">

<label>Search by:</label>

<select name="filter">

<option value="title">Title</option>

<option value="platform">Platform</option>

<option value="sku">SKU</option>

</select>

<input type="text" name="query" required>

<button class="btn" type="submit">Search</button>

</form>

<?php if (isset($\_POST['search\_game'])): ?>

<?php if (empty($gameResults)): ?>

<p style="color:red;">No matching games found.</p>

<?php else: ?>

<ul style="list-style-type: none; padding-left: 0;">

<?php foreach ($gameResults as $game): ?>

<li style="margin-bottom: 1rem;">

<div style="display: flex; justify-content: space-between; align-items: center; flex-wrap: wrap; gap: 1rem;">

<div style="flex-grow: 1;">

<?php echo htmlspecialchars("{$game['title']} ({$game['platform']}) - SKU: {$game['sku']} - In Stock: {$game['quantity\_in\_stock']}"); ?>

</div>

<form method="POST">

<input type="hidden" name="select\_game\_id" value="<?php echo $game['game\_id']; ?>">

<button class="btn" type="submit">Add to Cart</button>

</form>

</div>

</li>

<?php endforeach; ?>

</ul>

<?php endif; ?>

<?php endif; ?>

</section>

<?php endif; ?>

<!-- Cart & Checkout -->

<?php if (!empty($selectedGames)): ?>

<hr>

<h3>Games in Cart</h3>

<ul style="list-style-type: none; padding-left: 0;">

<?php foreach ($selectedGames as $game): ?>

<li style="margin-bottom: 1rem;">

<div style="display: flex; justify-content: space-between; align-items: center; flex-wrap: wrap;">

<span>

<?php echo htmlspecialchars($game['title']); ?>

(x<?= $game['quantity'] ?>)

</span>

<form method="POST" style="margin-top: 0.5rem;">

<input type="hidden" name="remove\_game\_id" value="<?php echo $game['game\_id']; ?>">

<button class="btn" type="submit">Remove One</button>

</form>

</div>

</li>

<?php endforeach; ?>

</ul>

<?php endif; ?>

<!-- Final Actions -->

<?php if ($selectedCustomer): ?>

<p><strong>Customer:</strong> <?php echo htmlspecialchars("{$selectedCustomer['first\_name']} {$selectedCustomer['last\_name']}"); ?></p>

<?php endif; ?>

<?php if ($selectedCustomer && !empty($selectedGames)): ?>

<form method="POST" action="Checkout.php">

<input type="hidden" name="customer\_id" value="<?php echo $selectedCustomer['customer\_id']; ?>">

<button class="btn" type="submit">Proceed to Checkout</button>

</form>

<?php endif; ?>

<form action="Transaction\_Page.php" method="get" style="display:inline;">

<button class="btn" type="submit" name="reset" value="1">Start New Transaction</button>

</form>

<form action="Employee\_Dashboard.php" method="get" style="display:inline;">

<button class="btn" type="submit">Return to Dashboard</button>

</form>

</main>

</body>

</html>

Network Implementation

PlayCart POS is a custom-built point-of-sale system designed specifically for the unique demands of video game retail environments. It streamlines essential business functions such as inventory management, transaction tracking, and real-time sales processing. At the core of the system is a reliable and efficient local network implementation, powered by a Raspberry Pi configured as the central server. This setup enables the entire POS system to run on a local web stack, providing robust performance even in offline scenarios.

The system architecture utilizes Apache as the primary web server and MariaDB as the backend database engine. This lightweight yet powerful configuration allows for seamless access to the POS interface across multiple devices on the same local network. By hosting all application components—including web assets, backend scripts, and the database—directly on the Raspberry Pi, PlayCart POS creates a collaborative in-store environment that requires no external internet connection.

The Raspberry Pi serves as the hub for all network activity. Apache delivers the web interface, built with HTML, CSS, and JavaScript. At the same time, backend logic interacts with the MariaDB database to manage critical data such as product catalogs, inventory levels, user accounts, and sales records. Any browser-enabled device connected to the same Wi-Fi network can access the POS system by entering either the Raspberry Pi’s local IP address (e.g., http://192.168.x.x/) or its mDNS hostname (e.g., http://raspberrypi.local/). This plug-and-play accessibility supports a range of hardware, from laptops and tablets to dedicated terminals, enabling multiple users to operate the system simultaneously with minimal setup.

MariaDB plays a key role in ensuring consistent and secure data operations. The database is optimized for concurrent access, allowing multiple terminals to interact with it in real time. For instance, one employee can process a customer transaction while another updates inventory levels or generates reports. This architecture effectively mirrors the functionality of an IoT network, where multiple smart endpoints connect and sync through a centralized node. In this context, the Raspberry Pi acts as the central node, managing data integrity across all connected clients. This local-first approach offers high reliability and responsiveness, which is especially valuable in retail environments where internet access may be unstable or unavailable.

While the current implementation prioritizes local functionality, the system is built with scalability in mind. Although cloud synchronization and remote access features are not yet in place—largely due to resource constraints and complexity—they remain part of the roadmap for future development. For now, data backup is handled via connected USB storage devices. Scheduled scripts can automatically copy the MariaDB database at defined intervals, providing an easy and cost-effective safeguard against data loss.

Looking ahead, additional enhancements may include features like automatic stock level alerts, sales milestone notifications, and even optional remote monitoring capabilities. These could be delivered directly to staff terminals, enhancing day-to-day responsiveness and overall operational efficiency.

The network implementation of PlayCart POS delivers a stable, scalable, and professional-grade platform tailored for video game retail. Leveraging the Raspberry Pi’s flexibility, the proven reliability of Apache, and the efficiency of MariaDB, the system enables multi-device, real-time interaction without relying on cloud infrastructure. With support for future expansion and IoT-like functionality baked into its architecture, PlayCart POS stands as a powerful in-store solution—and a strong foundation for the next generation of smart retail systems.

User Interface and Purpose

**Login:** The basic login screen will serve as the launchpad for the site. Employees can log in using their employee ID number and navigate through the site depending on their specific role.

**Dashboard:** The dashboard will be the main part of the site for the user. This is where games will be displayed to the user with prices to either rent or buy. They can scroll through the selection displayed and add items to a virtual cart.

**Checkout:** Once a user has added at least one item to their cart, they can access the checkout screen. This screen will be a secure page where the user can rent or purchase the selections they made while browsing the site.

**Employee Login:** Once the page is reached, there will be an option to login. Only employees with valid credentials can access the rest of the site, which is laid out in a very user-friendly manner.

**Employee Dashboard:** Once an employee is logged in, they are met with a screen that allows them to create a new transaction for a customer that already exists or for a new customer after adding in their information

**Management Dashboard:** If a user with management privileges signs in, they are met with a larger variety of options. They can start a new transaction of course, but they also have access to some management-only options as well.

**Inventory Management Screen:** The inventory Management screen will show the management employee user what is currently in stock and available to be sold to customers.

**Employee Management:** The Employee management screen allows user with the role of manger to add new employees to the system and set up their credentials so they can gain access to all options on the site.

**Reports:** The reports screen displays information about processed orders and the customers who placed them. It also provides a search feature which would be helpful on busier days when a lot of transactions were made.

**Backup To USB:** This screen shows which devices are connected directly to the system and allows for management personal to mount or unmount storage devices to the system. Once a device is properly mounted, a backup can be run and data on the site at that time will be saved to the storage device attached.

**Close Day:** This screen allows management personal to close out sales for the day and verify that the expected transaction totals match what is present at the time of closing.

**UI Goals:** The user interface will be built with user convenience in mind. Users accessing the site will be able to navigate through available inventory on the dashboard to quickly find items to purchase or rent. Management employees will be able to efficiently manage both inventory functions as well as all details on previously placed orders. This system allows for employees and managers to preform only the tasks that are assigned to them and allows for simple navigation that keeps customers satisfied with the multiple options provided for their gaming needs.

GUI Flowchart

A screenshot of a computer

AI-generated content may be incorrect.

Constructed with convenience in mind, the system provides a seamless experience for both customers and employees as they navigate the login screens and dashboards. Upon launching the application, users are greeted with a user-friendly interface that prompts them to enter their credentials. Employees input their unique username and password to access the employee dashboard, where they can manage inventory, process sales, and view reports.

In contrast, other users utilize a simplified version of the interface, designed specifically for their needs. If a customer attempts to access the employee login, the system immediately denies access and displays a clear message informing them that they lack the proper credentials. This ensures security while maintaining a smooth experience for all users. Similarly, if an employee attempts to access an admin account, the system denies access in the same way.

Once logged in, both employees and admins can easily browse products, view detailed descriptions, and add items to a customer’s cart without hassle. The system responds instantly to user actions, providing real-time updates on inventory and order status, ensuring a responsive and efficient shopping experience. Administrators, however, have exclusive access to employee management tools, report generation, a USB backup feature, and the ability to close the store for the day.

The ability to manage (create, edit, and remove) employees within the system is highly efficient. The "Reports" tab allows administrators to preview data collected by the system on purchases. Additionally, the USB backup feature enables the system to create a backup of itself onto a USB device, allowing for easy recovery in the event of a disaster. This streamlined approach not only reduces operational costs but also enhances user satisfaction by allowing both employees and admins to interact with the same intuitive platform. By focusing on user experience, the system effectively meets the needs of all users while maintaining both security and efficiency.

Cybersecurity Concerns and Mitigations

When creating a system hosted on a Raspberry Pi for the PlayCart POS system using Apache and MariaDB, several security concerns must be addressed to ensure the system is safe and functional. Since the system will be hosted locally, the focus will be on password protection, session management, role-based access control, SQL injection prevention, and cross-site scripting (XSS).

**Password Protection & Authentication:** We will use basic authentication with Apache’s mod\_auth\_basic to protect sensitive parts of the system. Passwords will be hashed using secure algorithms (e.g., bcrypt or SHA-256) before being stored in the MariaDB database. This ensures passwords are not stored in plain text and helps prevent unauthorized access (Orhagen, 2024).

**Session Management:** We will use secure cookies to manage user sessions. Cookies will be set with HttpOnly and Secure flags to prevent session hijacking. Additionally, session timeouts will automatically log out users after a period of inactivity, reducing the risk of unauthorized access due to forgotten or idle sessions (Ella, 2025).

**Access Control:** Role-based access control (RBAC) will be implemented to restrict access to sensitive features of the POS system. For example, cashiers will only have access to sales data, while managers can modify inventory and view reports. This will be handled with backend logic and MariaDB queries that check user roles before granting access. The roles will be stored in the MariaDB database, with each user having a unique role identifier (Orhagen, 2024).

**SQL Injection Protection:** Parameterized queries will be used to protect the system from SQL injection attacks. This will ensure that user inputs are handled securely. Additionally, user input validation will be enforced, checking for invalid characters or dangerous input before being processed by the database. PDO (PHP Data Objects) will be used for safe and efficient database interactions (Ella, 2025).

**Cross-Site Scripting (XSS) Prevention:** To prevent XSS attacks, all user inputs will be sanitized and escaped before being displayed on the web interface. PHP functions like htmlspecialchars() will be used to escape any HTML tags users might try to inject. We will also implement a Content Security Policy (CSP) header as an additional layer of defense against malicious content injection (Ella, 2025).

**Secure Data Transmission:** Since the system will be locally hosted, we will use a self-signed SSL certificate for HTTPS to secure the communication between the client and server. While self-signed certificates are sufficient for local development, in a production environment, a trusted SSL certificate from a recognized authority should be used to avoid security warnings and ensure trusted communication (Orhagen, 2024).

The PlayCart POS System also features SSH hardening, input validation, and port filtering. SSH hardening removes weaker defenses and reduces the scale of damage that could be inflicted by an attacker by reinforcing logins. Proper input validation applied to the input of users helps to reinforce the system. Port filtering to block and allow communication on various vulnerable and secure points will also encourage more reinforcement to the system (Orhagen, 2024).

For the PlayCart POS System to function properly, SSH must be enabled in order to allow the Raspberry Pi’s operating system to communicate with local devices, using the running program to allow the project to be accessed by other devices. However, running SSH creates glaring vulnerabilities that malicious outside forces can exploit (Ella, 2025). To mitigate these issues, we must first configure the Raspberry Pi to replace the unprotected default username and password to establish a basic, but secure defense against those who access the device (Orhagen, 2024).

However, the system can also be accessed through the internet via a connected device, which raises concerns about unauthorized attempts to access the admin panel. Fortunately, this risk can be easily avoided by locking the admin panel behind a login screen with a unique password and username to access admin privileges. Not only the login, but the other menus will also be designed in a visually appealing easy-on the eyes way. By implementing these measures, we safeguard user data and prevent unauthorized access. Using a self-signed SSL certificate for HTTPS ensures secure communication, even in our local environment. Together, these measures form a solid foundation for a secure and reliable POS system (Ella, 2025; Orhagen, 2024).

Expected Challenges

The challenges we may face with the PlayCart POS system encompass various critical aspects of its functionality and security. First, network connectivity and stability will be paramount; devices will rely on stable Wi-Fi when connecting via local IP or hostname (e.g., http://192.168.x.x/), and any disruption could interrupt access to the POS system during transactions. Additionally, security implementation will present its own hurdles. Setting up role-based access control (RBAC) will require meticulous session handling and user authentication, along with proper implementations of password hashing, SQL injection protection, and XSS prevention in Apache and PHP.

Collaboration among multiple devices will also pose challenges. When different staff devices connect simultaneously, it may lead to race conditions or synchronization issues, particularly in inventory management. Scalability limitations could complicate matters further, as the Raspberry Pi's limited processing power may degrade performance as more devices connect or features are added. The local server setup means that backing up and syncing data offsite will require careful planning.

Moreover, using USB drives for backups will introduce risks. If a USB drive is physically damaged or lost, it could result in irreversible data loss unless we implement redundant backups. Real-time communication features, such as alerts for low stock, will necessitate effective use of WebSocket or polling logic, which may present challenges within a local setup without relying on a cloud service.

There will also be concerns regarding role conflicts or overlap. It will be essential to ensure that only managers have access to inventory, order, and employee settings, which must be strictly enforced; any bugs in RBAC could inadvertently expose sensitive features to cashiers. Additionally, the visual layout and flow complexity of combining employee and manager processes into a single interface could confuse users, potentially leading to user experience missteps if there is no clear separation or guidance.

Finally, maintenance and updates will present logistical challenges. When the PlayCart POS system is hosted on the Raspberry Pi, updates will require manual access to the device, making it difficult to scale updates across multiple stores or devices effectively.

RTM

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Req. No. | Req. ID | Category | Requirement Description | Status | Notes |
| 1 | 1.1 | System Setup | Download and install Raspberry Pi OS Imager | Passed | Used the official Raspberry Pi Imager tool to select and install Raspberry Pi OS (Lite). This provides a clean, reliable foundation for running the headless PlayCart POS system. |
| 1.2 | System Setup | Flash the Raspberry Pi OS to the SD card | Passed | Flashed the OS to an SD card using the imager tool. This is a required step to boot the operating system and begin device setup. |
| 1.3 | System Setup | Boot the Raspberry Pi and complete initial setup | Passed | Inserted the SD card into the Raspberry Pi and powered on the device to complete first-time configuration. Initial setup included creating a default user, connecting to Wi-Fi, and configuring essential networking to enable remote management of the PlayCart POS unit. |
| 2 | 2.1 | Configuration | Enable SSH for remote access ('sudo raspi-config'). | Passed | Enabled SSH to allow remote terminal access. This is essential for headless operation and ongoing PlayCart POS maintenance without needing a monitor or keyboard. |
| 2.2 | Configuration | Configure localization settings (timezone, keyboard, Wi-Fi country), ('sudo raspi-config'). | Passed | Used raspi-config to set the correct timezone, keyboard layout, and Wi-Fi country. This ensures system logs, data entries, and network behavior are properly localized for the target deployment region. |
| 2.3 | Configuration | Disable auto-login and ensure GUI does not load on boot ('sudo raspi-config') | Passed | Disabled auto-login and the desktop GUI on boot to conserve system resources and improve device security in a headless setup. |
| 2.4 | Configuration | Update system (`sudo apt update && sudo apt upgrade -y`). | Passed | Ran sudo apt update && sudo apt upgrade -y to update all packages. This ensures the system starts with the latest security patches and software versions, reducing vulnerabilities before deploying the PlayCart POS application. |
| 3 | 3.1 | Database | Update package lists for MariaDB installation ('sudo apt update') | Passed | Refreshed package lists to prepare for installing MariaDB. This ensures the latest available version is pulled from the repository. |
| 3.2 | Database | Install MariaDB server ('sudo apt install mariadb-server -y') | Passed | Installed the MariaDB server using apt. This provides the SQL backend needed to store and manage all PlayCart POS data. |
| 3.3 | Database | Start MariaDB service ('sudo systemctl start mariadb') | Passed | Started the MariaDB service and verified that it launched correctly. Confirmed the database engine was ready for use. |
| 3.4 | Database | Enable MariaDB to start on boot ('sudo systemctl enable mariadb') | Passed | Enabled MariaDB to auto-start on boot. This guarantees the database service is always running when the Pi powers on. |
| 3.5 | Database | Check MariaDB service status ('sudo systemctl status mariadb') | Passed | Checked MariaDB status using sudo systemctl status mariadb to confirm a successful launch and identify any startup issues. |
| 3.6 | Database | Secure the Installation (First-Time Setup) ('sudo mysql\_secure\_installation') > (Press Enter > type 'y' x2 > Set Password as: MySQLRoot2025! > type 'y' x4) | Passed | Ran mysql\_secure\_installation to harden the default MariaDB configuration. Set a secure root password and disabled insecure settings like anonymous users and remote root login. |
| 3.7 | Database | Enter MariaDB shell ('sudo mysql -u root -p') > (Use password: MySQLRoot2025!) | Passed | Logged into MariaDB using the new root credentials. Verified successful authentication and readiness to build the database structure. |
| 3.8 | Database | Create the PlayCart database ("CREATE DATABASE PlayCart;") | Passed | Created the PlayCart database schema. All application tables for employees, games, customers, and transactions are stored within this database. |
| 3.9 | Database | Select the PlayCart database for use ("USE PlayCart;") | Passed | Selected PlayCart as the active database to prepare for table creation. |
| 3.10 | Database | Create database user playcart\_admin | Passed | Created a dedicated playcart\_admin database user to separate system-level and app-level access. |
| 3.11 | Database | Grant permissions to playcart\_admin (Set password: PlayCartAdmin2025!) | Passed | Granted full privileges to playcart\_admin and set a secure password. This user is used by the POS web application to manage data securely. |
| 3.12 | Database | Apply privilege changes with ('FLUSH PRIVILEGES;') | Passed | Flushed privileges to apply the new user’s access rights immediately. |
| 3.13 | Database | Create Employees table | Passed | Created the Employees table to store staff accounts, roles, and login credentials. |
| 3.14 | Database | Create Games table | Passed | Created the Games table to track inventory, pricing, platform, and title data. |
| 3.15 | Database | Create Customer table | Passed | Created the Customer table to store renter and buyer details for transaction records. |
| 3.16 | Database | Create Rentals table | Passed | Created the Rentals table to log outgoing and returned rentals, including timestamps. |
| 3.17 | Database | Create Transactions table | Passed | Created the Transactions table to record purchases, rentals, and returns. |
| 3.18 | Database | Create Logs table | Passed | Created the Logs table to capture events and changes for audit and tracking purposes. |
| 3.19 | Database | Create Cart tabe | Passed | Created the Cart table to track in-progress customer selections during checkout. |
| 3.20 | Database | Create DailySummary table | Passed | Created the DailySummary table to log end-of-day totals and reconciliation data. |
| 3.21 | Database | Create BusinessDayStatus table | Passed | Created the BusinessDayStatus table to monitor the current open or closed state of the business day. |
| 3.22 | Database | Exit MariaDB shell ('EXIT;') | Passed | Exited the MariaDB shell after completing all database setup steps. The backend is now ready to support the PlayCart POS web application. |
| 4 | 4.1 | Web Server | Update and upgrade package lists ('sudo apt update && sudo apt upgrade -y'). | Passed | Ran a package update to ensure compatibility before installing web services. |
| 4.2 | Web Server | Install Apache2, PHP, and PHP-MySQL ('sudo apt install apache2 php php-mysql -y') | Passed | Installed Apache2, PHP, and the PHP-MySQL extension. These are required to host the PlayCart POS system and allow it to connect to the MariaDB backend. |
| 4.3 | Web Server | Start Apache2 service ('sudo systemctl start apache2') | Passed | Started the Apache2 service and confirmed it launched without errors. |
| 4.4 | Web Server | Enable Apache2 service to start on boot ('sudo systemctl enable apache2') | Passed | Enabled Apache2 to start automatically on boot so the POS web interface is always available after reboot. |
| 4.5 | Web Server | Check Apache2 service status ('sudo systemctl status apache2') | Passed | Verified Apache2 was running properly using systemctl status apache2. |
| 4.6 | Web Server | Remove Apache default index.html and confirm permissions | Passed | Removed the default index.html file from /var/www/html to prevent conflicts with the PlayCart PHP application. |
| 4.7 | Web Server | Disable directory indexing in Apache configuration | Passed | Disabled directory listing in the Apache configuration to reduce potential information exposure. |
| 4.8 | Web Server | Retrieve Raspberry Pi's IP address ('hostname -I') | Passed | Retrieved the Pi’s local IP address using hostname -I for testing the web interface from other devices on the network. |
| 4.9 | Web Server | Open test page in browser to confirm Apache2 is running (http://(YOUR\_PI\_IP)) | Passed | Loaded a test page from a browser to confirm Apache was serving files correctly. |
| 4.10 | Web Server | Open PHP info page in browser and verify correct output ('echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php') > (web browser > (e.g. http://192.168.0.81/info.php) | Passed | Created a temporary info.php page to verify that PHP was correctly installed and integrated with Apache. Checked version and module support. |
| 4.11 | Web Server | Delete info.php after confirming output ('sudo rm /var/www/html/info.php') | Passed | Deleted the info.php file after testing to avoid exposing PHP configuration details. |
| 4.12 | Web Server | Build Login.php | Passed | Created Login.php to handle user authentication and session control. |
| 4.13 | Web Server | Build Employee\_Dashboard.php | Passed | Built Employee\_Dashboard.php to display staff options and links, dynamically tailored based on user role. |
| 4.14 | Web Server | Build Transaction\_Page.php | Passed | Developed Transaction\_Page.php to allow staff to select or register customers, add games to a virtual cart, and prepare for checkout. |
| 4.15 | Web Server | Build Checkout.php | Passed | Created Checkout.php to handle payment, calculate totals, update inventory, and log completed transactions. |
| 4.16 | Web Server | Build Inventory\_Management.php | Passed | Built Inventory\_Management.php to allow authorized employees to manage game listings and stock levels. |
| 4.17 | Web Server | Build Reports.php | Passed | Built Reports.php to generate filtered summaries of past transactions, rentals, and returns for analysis. |
| 4.18 | Web Server | Build Employee\_Management.php | Passed | Created Employee\_Management.php to manage user accounts, roles, and permissions from within the web interface. |
| 4.19 | Web Server | Build Close\_Day.php | Passed | Built Close\_Day.php to finalize sales, log the daily summary, and lock transactions until the next open business day. |
| 4.20 | Web Server | Build Backup\_To\_USB.php | Passed | Created Backup\_To\_USB.php to trigger automatic database and log backups to an attached USB drive. |
| 4.21 | Web Server | Restart Apache2 to apply changes ('sudo systemctl restart apache2') | Passed | Restarted Apache2 to apply changes to configuration and file permissions. |
| 4.22 | Web Server | Verify session settings using grep command (' grep -n "session.cookie\_lifetime" /etc/php/8.2/apache2/php.ini') | Passed | Checked session cookie settings in php.ini using grep to confirm secure handling of user sessions. |
| 5 | 5.1 | Visual Elements | Move PlayCartPOS\_LOGO.png to /home/playcart | Passed | The logo file PlayCartPOS\_LOGO.png was successfully moved to /home/playcart to ensure consistent access across all devices. This location aligns with the standardized file structure used throughout the project. |
| 5.2 | Visual Elements | Run ('ls -l "/home/playcart/PlayCartPOS\_LOGO.png"') to confirm asset visibility | Passed | The command ls -l "/home/playcart/PlayCartPOS\_LOGO.png" was executed to confirm that the logo file exists and is accessible. The file was verified to have appropriate ownership and permissions. |
| 5.3 | Visual Elements | Move all .png files from Visual\_Elements to /var/www/html/ ('sudo mv /home/playcart/PlayCartPOS\_LOGO.png\*.png /var/www/html/') | Passed | All .png files, including PlayCartPOS\_LOGO.png, were moved from the Visual\_Elements directory to /var/www/html/ using the specified command. This ensures the assets are accessible to the web server for integration into the interface. |
| 5.4 | Visual Elements | Run ('ls -l /var/www/html/) to confirm font files were transferred | Passed | The command ls -l /var/www/html/ was run to verify that all font and image assets were successfully transferred. The presence of required .png files was confirmed, ensuring they are available for use in the interface. |
| 5.5 | Visual Elements | Create style.css file. | Passed | A style.css file was created to manage all custom visual styling for the PlayCart POS interface. This file serves as the central stylesheet for layout, spacing, and branding elements across the system. |
| 6 | 6.1 | Device Security | Enable and configure ufw to allow only ports 80, 443, and SSH (22) (Post-Build / Deployment Phase: Once everything is tested and stable, block SSH via UFW to reduce risk) | Passed | Enabled the UFW firewall and allowed only necessary ports: 80 (HTTP), 443 (HTTPS), and 22 (SSH). SSH will be disabled after deployment to reduce the attack surface. |
| 6.2 | Device Security | Enforce correct file permissions in /var/www/html | Passed | File permissions in /var/www/html were reviewed and restricted to limit access to only necessary users. The Apache user (www-data) was granted minimal access, while ownership of critical files was assigned to root to prevent unauthorized modifications. |
| 6.3 | Device Security | Generate self-signed SSL certificate and private key using openssl | Passed | Generated a self-signed SSL certificate using OpenSSL to provide encrypted HTTPS access for secure web sessions. |
| 6.4 | Device Security | Copy certificate and key to Apache SSL directory (e.g., /etc/ssl/certs, /etc/ssl/private) | Passed | Copied the certificate and private key to Apache’s SSL directories in preparation for serving secure content. |
| 6.5 | Device Security | Configure Apache to serve the site over HTTPS using the generated certificate | Passed | Enabled the Apache ssl module and activated the default SSL configuration to begin HTTPS support. |
| 6.6 | Device Security | Enable SSL module and the default-ssl.conf virtual host in Apache | Passed | Edited the Apache configuration to load the new SSL certificate and ensure the server delivers encrypted traffic. |
| 6.7 | Device Security | Redirect all HTTP traffic to HTTPS in Apache config | Passed | Redirected all incoming HTTP traffic to HTTPS to enforce secure connections by default. |
| 6.8 | Device Security | Restart Apache and verify the site loads over https://<pi-ip> | Passed | Restarted Apache and tested the connection via browser using the Pi’s IP address. Confirmed that HTTPS was working as expected. |
| 6.9 | Device Security | Mount external USB storage with www-data ownership for secure web-only access | Passed | Mounted a USB drive with ownership and permissions correctly set for the www-data user to allow secure, automated backups. This step required significant extra work compared to other setup tasks. The drive had to be properly recognized, formatted, and mounted with persistence across reboots, and special care was taken to ensure Apache could access it despite running in a restricted environment. Several test iterations and permission adjustments were needed before the backup script could interact with the USB mount successfully. |
| 6.10 | Device Security | Place Raspberry Pi in a secure, tamper-resistant location | Passed | Placed the Raspberry Pi in a secure, tamper-resistant physical location to reduce the risk of unauthorized access or manipulation. |
| 7 | 7.1 | Verify and Ensure Functionality | Verify that employees can log in with valid credentials | Passed | Was able to successfully log in with a valid employee login and the correct dashboard was displayed. No issues were discovered during this test! |
| 7.2 | Verify and Ensure Functionality | Ensure role-based dashboard loads correctly after login | Passed | Both the manager and employee dashboards display the correct information. No issues were discovered during this test! |
| 7.3 | Verify and Ensure Functionality | Verify that products can be searched by SKU or title | Passed | After testing the exact SKU, partial SKU, exact title, and partial title, all tests gave correct feedback given the input. No issues were discovered during this test! |
| 7.4 | Verify and Ensure Functionality | Verify that items can be added to the cart successfully | Passed | Adding an item to the cart works as intended with the title and price correctly displayed. No issues were discovered during this test! |
| 7.5 | Verify and Ensure Functionality | Ensure toggle for purchase/rental mode updates cart behavior | Passed | The label between "Rental" and "Purchase" is displayed accordingly based on the user's choice. No issues were discovered during this step! |
| 7.6 | Verify and Ensure Functionality | Verify that the return mode functions as expected | Passed | After selecting "Return" at the checkout, the correct info as well as the rate owed for the rental. No issues were discovered during this test! |
| 7.7 | Verify and Ensure Functionality | Ensure that cart can be cleared before checkout | Passed | After multiple games were added, the option to remove the games was available and worked with ease. No errors or issues were discovered during this test! |
| 7.8 | Verify and Ensure Functionality | Verify that checkout processes the transaction and updates the database | Passed | After completing a purchase or rental, the receipt is correctly displayed in the Report. No issues were discovered during this test! |
| 7.9 | Verify and Ensure Functionality | Verify inventory updates correctly after a transaction is completed | Passed | After completing a purchase or rental, the inventory is able to show the updated quantity of the item purchased! No issues were discovered during this test! |
| 7.10 | Verify and Ensure Functionality | Ensure that order history displays past transactions accurately | Passed | After checking the transaction history in the "Report" tab, all information such as the game title, amount, and date is displayed correctly. No issues were discovered during this test! |
| 8 | 8.1 | Database & Query Protection | Ensure that SQL queries use prepared statements or input sanitization | Passed | Tried entering a variety of data to multiple types of fields and was unable to expose any restricted information. |
| 8.2 | Database & Query Protection | Attempt to submit a form field with SQL injection (e.g., ' OR '1'='1) | Passed | Tried entering SQL code into search functions, no input or errors were returned. |
| 8.3 | Database & Query Protection | Submit malformed data into search field (e.g., special characters) | Passed | Special characters are processed like any other data and long input no longer crashes the site. |
| 8.4 | Database & Query Protection | Confirm invalid or empty customer data does not trigger a database error | Passed | Fields can not be left blank without a popup message appearing and telling the user to fill in those blanks. |
| 8.5 | Database & Query Protection | Verify that restricted pages do not execute any query if accessed without permission | Passed | Users are not able to access management pages without the proper credentials. If a URL exploit is attempted to gain access, the user gets redirected to the login page. |
| 8.6 | Database & Query Protection | Confirm that database does not expose error messages to users in the front end | Passed | No database errors or messages with sensitive information are ever displayed to the user. |
| 8.7 | Database & Query Protection | Ensure no direct access to database tables via browser (e.g., navigating to .php files) | Passed | If a user attempts to reach a table using what would be it's URL, they will only get a generic "page not found" screen. |
| 9 | 9.1 | System Integrity | Attempt to access a page that requires login without being logged in | Passed | Attempting to access a page that requires login without being logged in redrects to the login page. |
| 9.2 | System Integrity | Attempt to perform an action outside of assigned role (e.g., Cashier edits inventory) | Passed | Attempting to perform an action out of an employee's assigned role prevents them from doing so, and takes them right back to the login page. |
| 9.3 | System Integrity | Test form submission with session expired (simulate timeout or logout) | Passed | Attempting to submit a new form (in this case, both a new employee and checkout) does not work when the authentication session is removed or expired. |
| 9.4 | System Integrity | Attempt to checkout with an empty cart | Passed | Attempting to checkout with an empty cart will cause an error message to appear asking for input in the game selection field. |
| 9.5 | System Integrity | Add duplicate item to cart and confirm proper behavior (e.g., increase qty, no crash) | Passed | Adding duplicate items to the cart will properly add them to the cart as intended. |
| 9.6 | System Integrity | Attempt to submit a transaction without selecting purchase or rental mode | Passed | Submitting a transaction without selecting a purchase or rental mode will not process a transaction and returns an empty page. |
| 9.7 | System Integrity | Simulate power cycle or forced shutdown — ensure database remains intact | Passed | When the pi system is shut down, the database remains intact. |
| 10 | 10.1 | Access Control & Permissions | Log in as a Cashier and attempt to access Manager-only pages via URL | Passed | After logging in as an employee, the attempt to access the manager's dashboard via URL was unsuccessful. No issues were discovered during this test. |
| 10.2 | Access Control & Permissions | Log in as a Manager and verify all manager pages are visible and accessible | Passed | Once logged in, all of the prompts to each page worked as intended. No issues or errors were discovered during this test! |
| 10.3 | Access Control & Permissions | Log in as a Cashier and verify inventory and employee settings are hidden/blocked | Passed | After logging in as an employee, there are no other prompts that are restricted to the employee dashboard appearing. No issues were discovered during this test! |
| 10.4 | Access Control & Permissions | Attempt to POST data to a restricted page as Cashier (e.g., submit inventory change) | Passed | When attempting to bypass UI and POST-restricted changes as a cashier, the page gets blocked and is unable to be accessed. No issues were discovered during this test! |
| 10.5 | Access Control & Permissions | Create a new employee and assign them a role — verify correct permissions apply | Passed | After adding a new employee from the manager's screen, the correct restrictions still apply when logged in as the newly created employee. No issues were discovered during this test! |
| 10.6 | Access Control & Permissions | Log out and confirm restricted pages are no longer accessible | Passed | After logging out of an account and revisiting the site by using the previously visited option on the browser, the permission reroutes back to the login screen without accessing the account again. No issues were discovered during this test! |
| 10.7 | Access Control & Permissions | Verify employee cannot change their own role or edit their own account (unless Manager) | Passed | Employees are not given an option and have no way to change their roles or permissions, only the manager's account can access the roles of all the employees. No issues were discovered during this test! |
| 10.8 | Access Control & Permissions | Verify that only Managers can see and access the Backup to USB feature | Passed | Employees are not given an option to back up the system, only the manager's account can access the back to USB. No issues were discovered during this test! |
| 11 | 11.1 | Security | Verify login form rejects incorrect usernames and passwords | Passed | Delivers a "Invalid credentials." message and rejects incorrect usernames and/or passwords when they are entered. |
| 11.2 | Security | Attempt multiple failed logins and observe response | Passed | Delivers a "Too many failed attempts. Please try again later." message, and rejects further inputs for a while. |
| 11.3 | Security | Check for autocomplete disabled on password fields | Passed | Password are not filled in or completed when their fields are selected. |
| 11.4 | Security | Submit special characters and scripts into all form fields | Passed | When special characters and scripts are used, the "Please match the requested format" message is displayed. |
| 11.5 | Security | Confirm password fields are not visible when typed | Passed | Entered PINs do not flash or reveal any input, remaining masked and private as dots. |
| 11.6 | Security | Verify session expires or logs out user after period of inactivity | Passed | After around 10 minutes of inactivity, user sessions expire, and that user is logged out and returned to the login menu. |
| 11.7 | Security | Attempt direct URL access to system files (e.g., config files, database exports) | Passed | Attempting to access a page that requires authorization will take you to an "Unauthorized Access" webpage. |
| 11.8 | Security | Confirm logout button ends session and prevents reuse of session with back button | Passed | While logging out does end the session, you are still able to access the page the dashboard. However, when attempting to click anything that needs authorization, you are immediately booted back to the login menu. |
| 11.9 | Security | Confirm backup command does not expose credentials or raw shell output | Passed | Running the backup command in the USB backup function does not expose any credentials or raw shell output. |
| 12 | 12.1 | System Logging | Confirm that successful employee logins are logged | Passed | The logins were logged successfully. |
| 12.2 | System Logging | Confirm that failed login attempts are logged | Passed | The logins are logged correctly with no issues. |
| 12.3 | System Logging | Verify that all successful checkout transactions are logged | Passed | Every transaction is logged. |
| 12.4 | System Logging | Confirm that inventory changes are logged with employee ID | Passed | All inventory changes were logged without any issues |
| 12.5 | System Logging | Verify that employee account creation or edits are logged | Passed | All edits and creations are logged. |
| 12.6 | System Logging | Attempt a restricted action (e.g., cashier editing inventory) and confirm it is logged | Passed | All restricted actions are logged. |
| 12.7 | System Logging | Confirm that manual logouts are logged | Passed | All manual logouts were logged |
| 13 | 13.1 | User Experience | Verify that all button labels clearly describe their action | Passed | All buttons are descriptive and easy to read and understand. |
| 13.2 | User Experience | Confirm all required form fields display error messages when left blank | Passed | When a user leaves a field blank, they will receive a message to fill in that field with valid data. |
| 13.3 | User Experience | Test that success messages appear after key actions (e.g., transaction complete) | Passed | When games are added into inveotry and when a transaction is completed, the user will receive a message letting them know the process was successful. We might want to include this feature in the employee management page as well. |
| 13.4 | User Experience | Verify that cart totals and item types (purchase vs rental) are clearly shown | Passed | All transaction information is included and broken down into easily readable subcategories. The user is shown the subtotal, tax amount, and overall calculated total. They can also see the prices for both renting and purchasing the slected title. |
| 13.5 | User Experience | Ensure navigation is consistent across all pages (e.g., return to dashboard) | Passed | The site is easy to navigate and the user can return to the dashboard at any time while perusing. |
| 13.6 | User Experience | Confirm page layout does not break when resized to smaller screens (if applicable) | Passed | Site remains functional when page is zoomed in or out as well as if the window is resized. |
| 13.7 | User Experience | Verify text readability (font size, contrast, no overlapping elements) | Passed | Text is easy to read and understand. Colors used are distinctive enough to provide contrast between backgrounds, buttons, and text. |
| 13.8 | User Experience | Test that employees can complete a full transaction without needing external help | Passed | Completing a transaction is simple and straighforward. The process can be done quickly and with little ability to make a mistake. No help should be needed. |
| 13.9 | User Experience | Use a screen reader (e.g., NVDA) to verify that buttons and page structure are readable | Passed | All titles, buttons, and fields are read aloud by the screen reader. |
| 14 | 14.1 | Mobile Functionality | Access the POS system from a phone or tablet on the same Wi-Fi network | Passed | I tested the POS on my phone and the system is accessible. |
| 14.2 | Mobile Functionality | Log in and navigate through major pages on a mobile device | Passed | I tested every major page on the site and they all work on mobile. |
| 14.3 | Mobile Functionality | Complete a full transaction (cart > checkout) from a mobile browser | Passed | I completed a transaction on my phone and it completely checked out with no issues. |
| 14.4 | Mobile Functionality | Check for any broken layouts that prevent using buttons or forms | Passed | I couldn't find any broken layouts on mobile. |
| 15 | 15.1 | Sensitive Data Exposure | Verify that passwords are stored securely in the database (e.g., hashed) | Passed | All passwords are securely stored in the database as a protected hash. |
| 15.2 | Sensitive Data Exposure | Confirm that password fields are not visible in browser autofill tools | Passed | In browser tools, PIN fields are not autofilled and the password reset form doesn’t show saved entries. |
| 15.3 | Sensitive Data Exposure | Check that error messages do not expose SQL queries, filenames, or stack traces | Passed | Error messages are given by the system directly and do not expose sensitive information or SQL queries. |
| 15.4 | Sensitive Data Exposure | Confirm that user sessions cannot be hijacked by using the back button after logout | Passed | While logging out does end the session, you are still able to access the page the dashboard. However, when attempting to click anything that needs authorization, you are immediately booted back to the login menu. |
| 15.5 | Sensitive Data Exposure | Ensure URLs do not expose sensitive data (e.g., ?password=...) | Passed | URLs do not contain sensitive information or data. |
| 16 | 16.1 | Maintenance & Backups | Manager initiates database backup to USB via dashboard button | Passed | After initiating a backup the correct message is displayed. No issues were discovered during this test! |
| 16.2 | Maintenance & Backups | Verify backup file is created with correct naming format and stored properly | Passed | After inspecting the USB, the backup was created with the correct name and time displayed! No issues were discovered during this test! |
| 16.3 | Maintenance & Backups | Attempt to run backup with no USB mounted — show appropriate failure message | Passed | After attempting to run a USB backup with no USB inserted, the correct error is displayed informing us that there is no USB connected. No issues were discovered during this test! |
| 16.4 | Maintenance & Backups | Confirm backup action is logged (optional if using event\_logs) | Passed | When checking for backup logs it appears in the terminal, however, there is no prompt to see a record of backups on the manager's dashboard. No other issues were discovered during this test! |

Bug Tracking and Fixes

|  |  |
| --- | --- |
| Bug Description | Resolution |
| After a rental purchase was made, there is no return feature option available. | This issue turned out to be a misunderstanding of system behavior rather than a true bug. The return feature is only applicable to rental transactions, not purchases, which is by design. Once this distinction was clarified during development, no code changes were necessary, and the feature was confirmed to be functioning as intended. |
| Entering long inputs to add new games or employees breaks the site. | Input validation was added to prevent excessively long entries when creating new games or employees, which previously caused the site to break or throw database errors. In addition, we resolved a critical issue related to employee deletion, where removing an employee from the database could disrupt system behavior and break log references. To fix this, we implemented a soft delete system for employees, marking them as inactive instead of removing them entirely. This change not only prevents site errors but also ensures that employee records remain associated with historical logs and transactions — an important safety consideration in any POS system, especially in the event of disputes or audits. |
| You are unable to add a game more than once to the cart, the system does not recognize adding it more than once and will not add more than one no matter how many times you input it. | This issue required a deeper fix, as the system was not originally designed to handle the same game being added to the cart multiple times. Attempting to modify this behavior initially caused conflicts in the Checkout logic, where the code did not support quantity tracking or repeated entries. We resolved it by restructuring the cart handling logic to support multiple quantities of the same game, and then updating the Checkout page to process and display those quantities correctly. After testing, the issue appears to be fully resolved and the system now supports multiple-item transactions as expected. |
| "I Tried checking event logs but the failed logins didn't show up." | Initially, failed login attempts were not being captured in the event logging system, which made it difficult to audit unauthorized access attempts. Although this specific issue wasn’t addressed directly at the time of testing, we did spend time improving the logging system overall. Enhancements were made to capture key system events such as successful logins, user actions, and administrative changes. If this issue persists, we may need to revisit the login handler to ensure failed attempts are explicitly logged. |
| "I Tried as employee and it didn't let me change anything so the security is good." | This is not a bug. The system correctly enforced role-based access control by preventing a standard employee account from making administrative changes. This confirms that the security restrictions are functioning as intended, ensuring that only users with manager privileges can access or modify sensitive areas of the system. |
| After adding an inventory item, the item was added to the list, but no success message appeared. This could cause confusion as the database stores more data over time. | A success message was missing after adding a new inventory item, even though the item was correctly inserted into the database. Without this feedback, users could be uncertain whether the action was completed successfully — especially as the inventory list grows and new entries become harder to spot. To address this, we added a clear success message to the Inventory Management page confirming when a new game is added. This improves user confidence and clarity during repeated entry tasks. |
| While most items are read aloud by the screen reader without issue. Most subheadings are not read out loud, which could cause confusion for visually impaired users. | After testing with the NVDA screen reader, I confirmed that most sections of the site are being read aloud correctly. However, subheadings and certain structural elements were inconsistently recognized, which may lead to navigation challenges for visually impaired users. Additional clarification is needed from the original tester to understand exactly which elements were missed and under what conditions. Follow-up testing or adjustments to heading tags and ARIA roles may be required to ensure full accessibility compliance. |
| After performing a successful or unsuccessful backup onto a USB, there is no log to reflect that a backup has been made. | Originally, the system did not record any logging information after initiating a backup to USB, whether the backup succeeded or failed. This made it difficult to verify if the operation had been attempted or completed, which is a concern for both auditing and recovery reliability. We addressed this by adding logging functionality to the backup process, ensuring that each backup attempt — successful or not — is recorded in the EventLogs table. This improvement provides clearer visibility into maintenance actions and helps ensure system accountability over time. |
| An SQL error occurs when an incorrect password/pin is entered in the Employee Login. | Patch addresses three main issues in the login functionality. First, it fixes incorrect error handling by ensuring that invalid employee IDs and passwords result in a generic error message, preventing attackers from knowing which part of the login is incorrect. Second, brute force protection was added to lock user accounts after 5 failed login attempts within a 15-minute period, improving security. Finally, the patch ensures that failed login attempts are correctly logged in the EventLogs table, including the employee ID and failure details for better tracking and accountability. |
| The site crashes if an existing game is updated with an empty SKU in the Inventory Management screen. | Patch was implemented that added a defensive measure to 3 fields in the Games table. Now, if a user attempts to update a game entry with the title, platform, or SKU missing, the site will display an error at the top of the page informing the user that those fields cannot be left blank. |
|  |  |
|  |  |
|  |  |
|  |  |

Team Strengths, Weaknesses, and Expected Contributions

#### Eric Reynolds – Group Lead

**Primary Role:** Project Lead (Sprint 3), Backend Developer, Planner, and Documentation Finisher

**Key Contributions:**

* Led project planning using ClickUp, finalized the Requirements Traceability Matrix (RTM), and completed documentation submissions.
* Provided backend development support, focusing on database logic and structural design.
* Ensured consistency and quality in the formatting of deliverables.

**Strengths:**

* Effective leadership and organizational skills.
* Strong backend programming and database design capabilities.
* Excellent planning and attention to formatting standards.

**Areas for Growth:**

* Limited involvement in manual QA and user-side security testing.

**Support Needs:**

* Relied on Jesse and other team members for coverage in security testing and abnormal user test cases.

#### Clinton Morris – Backend Logic Lead

**Primary Role:** Backend Logic Developer, Tester, and Documentation Contributor

**Key Contributions:**

* Developed backend logic, created system flowcharts, and authored verification steps.
* Validated transaction handling, session control, and system response accuracy.
* Led error testing and access control validations.

**Strengths:**

* Proficient in backend programming, system architecture, and debugging.
* Strong logical thinking and error identification.

**Areas for Growth:**

* Occasionally required assistance with visual formatting and presentation polish.

**Support Needs:**

* Received formatting support from Will, Hazel, and Eric.

#### Will Westergren – Backend Tester

**Primary Role:** Backend QA Specialist for Permissions, Roles, Backups, and UI Validation

**Key Contributions:**

* Tested role-based permissions, backup workflows, dashboard access, and session activity.
* Verified UI interactions and access control implementations.
* Contributed to documentation and formatting of testing reports.

**Strengths:**

* Skilled in structured testing, particularly role-based logic and backend QA.
* Analytical and methodical in problem-solving.

**Areas for Growth:**

* Occasionally needed additional time or clarification for backend logic understanding.

**Support Needs:**

* Sought support from Clinton and Eric for technical breakdowns and backend scripting.

#### Jesse Asbury – QA and Security Tester

**Primary Role:** Quality Assurance and System Security Tester

**Key Contributions:**

* Conducted comprehensive security testing, including session expiry, invalid login attempts, injection testing, and backup validation.
* Emphasized input validation and system behavior under failure conditions.

**Strengths:**

* Exceptional attention to detail in QA and security testing.
* Fast-paced tester with broad coverage of test cases.

**Areas for Growth:**

* Some test passes were brief and missed deeper backend analysis or impact assessments.

**Support Needs:**

* Frequently confirmed backend outcomes and secure behaviors with Eric.

#### Hazel Walker – Database and Documentation Specialist

**Primary Role:** Database Designer and Documentation Formatter

**Key Contributions:**

* Designed and refined the Entity-Relationship Diagram (ERD) and maintained database structure documentation.
* Played a key role in formatting and polishing all reports and visual submissions.

**Strengths:**

* Excellent at formatting, visual organization, and documentation clarity.
* Strong skills in ERD and database relationship modeling.

**Areas for Growth:**

* Less hands-on involvement in system development and test case implementation.

**Support Needs:**

* Collaborated with Clinton, Will, and Jesse for flowchart development and data validation input.

GUI Screenshots

A screenshot of a login box

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A purple rectangle with white background

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Works Cited

Mender. <https://mender.io/blog/remote-inspect-your-device-without-using-traditional-raspberry-pi-port-forwarding#:~:text=You%20need%20a%20plan%20to,vulnerability%20and%20can%20create%20risk>.

Pidora. https://pidora.ca/secure-your-raspberry-pi-strategies-to-harden-your-device-today/#:~:text=Common%20Vulnerabilities,Impact%20of%20Insecurity.

Appendix A: Issues That Could Not Be Solved

Although the bugs we were tracking were ultimately resolved, not all fixes were straightforward. Throughout the development of the PlayCart POS system, our team collaborated closely to address and resolve every issue that arose.

The system's reliance on SQL structuring as its foundation led to errors during critical processes. In particular, entering an incorrect PIN or password triggered an SQL error that prevented further input. What initially appeared to be a simple issue was, in fact, the result of three underlying problems: inadequate error handling, improperly implemented brute-force protection, and a failure to properly track failed login attempts within the EventLogs table. While users could bypass the error on the login screen by using the browser's back button and attempting to log in again, a more severe problem existed on the Inventory Management screen. Leaving the SKU form empty would cause the page to crash entirely, with no workaround available.

After identifying and addressing the root causes by fixing three key fields the team successfully patched these issues. With a rigorous battery of limited tests performed post-fix, the PlayCart POS system has been thoroughly refined and reinforced, resulting in a secure, user-friendly, and affordable solution for managing retail operations.

Appendix B: The Use of AI as a Tool

Throughout the project, our team used AI tools—particularly ChatGPT—as supplemental resources to assist with various stages of development, especially diagram generation, dynamic content design, and scope management.

AI was initially useful for generating rough drafts of diagrams. Though the early outputs required refinement, they provided a helpful starting point and enabled faster iteration. With persistence and repeated prompting, we coaxed AI into generating increasingly accurate versions of what we needed.

We also used AI extensively to ask technical and design-related questions. It often helped us think through implementation strategies, such as making our pages more dynamic while avoiding the complexity of managing too many individual views. When used interactively and for specific tasks, AI provided solid guidance and occasionally offered new approaches we hadn’t considered.

However, there were several drawbacks. One of the main challenges was the inconsistency of AI across sessions. Because free versions didn’t retain context, if we closed a tab or session, the continuity of the conversation was lost. This meant we often had to re-explain our project or repeat previous prompts to regain momentum. In some cases, AI would fixate on the wrong interpretation of a request and loop through unhelpful responses. Some team members also ran into paywalls mid-use or had the model unexpectedly change, leading to confusion or interruptions in workflow.

Despite these hurdles, we were able to make good use of the free AI tools available. We recognized that while AI was not a replacement for hands-on work or team collaboration, it served as a helpful brainstorming and support tool when used strategically and with patience.

Appendix C: Great Things

Throughout this project, our team experienced many high points that made the process both productive and rewarding. These positive experiences came from several key areas, including exceptional teamwork, individual contributions, and the strategic use of tools and resources that supported our success. Reflecting on our work together, it’s clear that these elements played a central role in shaping the outcome of our final product.

One of the standout aspects of the project was the strength of our collaboration. From the beginning, we committed to meeting weekly, typically on Mondays or Tuesdays, to focus on completing the more complex and time-sensitive tasks as a team. These work sessions became a cornerstone of our workflow. They allowed us to build out major components of each sprint while supporting one another in real-time. Whether the meetings lasted only a short while or extended into long, focused work periods, the time spent together was consistently productive and set a strong foundation for the rest of the week. Over time, we came to truly value these sessions, recognizing how critical they were for maintaining our momentum and ensuring no one was left behind.

Another important factor in our success was the recognition and application of individual strengths within the group. While every member contributed meaningfully, it’s important to highlight Eric’s role in particular. Eric consistently helped keep the group focused and on track, ensuring that tasks were not only completed but also documented and submitted accurately. His work ethic, attention to detail, and calm leadership made a noticeable difference each week. He was the reliable presence that grounded our group, and his efforts played a large part in our ability to meet our sprint goals and stay organized. Without his persistence and steady guidance, our project would have faced significantly more hurdles.

In addition to strong teamwork and individual contributions, we benefited greatly from the use of digital tools and resources. AI technologies, while not always perfect, helped support various planning and design stages. Several team members used AI tools to brainstorm ideas, structure diagrams, and refine page layouts. These tools helped accelerate decision-making and supported a clearer overall vision of the final product. Alongside this, ClickUp provided us with an efficient way to manage and delegate tasks, track progress, and stay aligned across sprints. Meanwhile, Discord served as our central communication hub, making it easy to share updates, ask questions, and coordinate meetings quickly.

Looking back, one of the most rewarding aspects of this project was seeing how well we adapted and supported one another. As deadlines approached and challenges emerged, the team responded with cooperation, determination, and resourcefulness. These experiences not only made the project more manageable but also made it more meaningful. The sense of shared responsibility and mutual respect was evident throughout, and it was encouraging to see each member of the group step up when needed.

In summary, our project was marked by strong collaboration, effective communication, and a willingness to lean on each other’s strengths. These "great things" defined the spirit of our work and played a crucial role in achieving our goals. They also reflect the type of teamwork and perseverance that we hope to carry into future projects.

Group Signatures