

ShopEase: Design, Implementation, and Security Analysis of a PHP-Based E-Commerce Web Application with Session-Based Cart Management

Soham Wagh, Saumya Tiwari, Harsh Sharma

Department of Computer Engineering, Bharat College of Engineering, Kanhor, Badlapur (west), Mumbai 421503

Project Guide: Prof. Samir Kumar

Abstract — ShopEase is a full-stack, PHP-based e-commerce web application implemented on the LAMP (Linux, Apache, MySQL, PHP) stack as an academic mini-project demonstrating industry-relevant architectural patterns. The system implements a dual-privilege model separating Administrator and Customer roles at both the session and database layers, a session-based shopping cart that maintains cart state without requiring user authentication for browsing, a dynamic product categorisation system with real-time per-category product counts surfaced from live database queries, and a MySQLi-driven persistence layer with referential integrity enforced through InnoDB foreign key constraints. The Administrator panel provides full product lifecycle management (CRUD: Create, Read, Update, Delete) while the Customer interface supports product browsing, category and keyword-based filtering, cart operations, and a complete simulated order checkout flow with MySQL transaction-backed database persistence. This paper presents the complete architectural decisions, database schema design with full SQL DDL, backend implementation methodology including annotated PHP code segments, session management strategy, Admin-User privilege separation model, Bootstrap 5 frontend UI design patterns, a comprehensive OWASP Top 10 (2021) security evaluation with code-referenced vulnerabilities, an end-to-end data flow analysis, a structured testing strategy, and a prioritised production-hardening roadmap with concrete implementation guidance.

Key Words: E-Commerce, PHP, MySQLi, Session Management, LAMP Stack, Shopping Cart, Admin-User Privilege Separation, Product Catalogue, Web Security, OWASP Top 10, Bootstrap 5, CRUD Operations, SQL Injection Prevention, BCrypt, CSRF Protection

1. INTRODUCTION

E-commerce revenues exceeded USD 5.8 trillion globally in 2023 (Statista [2]) and continue growing at a compound annual rate exceeding 10%. The accelerating shift from physical retail to online storefronts has democratised commerce for small and medium-sized enterprises (SMEs), allowing them to reach customers far beyond geographic proximity. For these businesses, a self-hosted, custom-built web storefront provides significant cost advantages and

complete ownership of customer data, compared to third-party marketplace solutions such as Amazon, Flipkart, or Meesho, which impose listing fees, commission structures, and algorithmic disadvantages for newer sellers.

Despite the clear commercial demand, student and small-scale developer implementations of e-commerce systems frequently exhibit a consistent set of well-documented failure modes: direct SQL string concatenation creating injection vulnerabilities, plaintext or weakly-hashed password storage, absent session expiry or regeneration mechanisms, no meaningful separation between administrator and customer privilege contexts, and missing CSRF protection on state-mutating operations [11, 13]. ShopEase was designed and built specifically to address these failure modes within a practical, deployable LAMP-stack architecture suitable for institutional learning and small-scale real-world deployment.

1.1 Motivation and Problem Statement

The fundamental motivation for ShopEase arises from three intersecting needs: (i) a teaching artefact demonstrating a complete, working e-commerce system with enough architectural depth to serve as a reference implementation for computer engineering students; (ii) honest acknowledgement of limitations, not claiming production-readiness where genuine vulnerabilities exist; and (iii) implementability without paid dependencies, licensed frameworks, or cloud services, making it accessible to students in resource-constrained environments.

1.2 Principal Contributions

(a) A fully functional, multi-page e-commerce web application with dual Admin/Customer privilege separation, built entirely on free, open-source LAMP stack tooling with no paid external dependencies. (b) A session-based shopping cart mechanism that maintains cart state across page navigations and browser sessions without requiring authentication for browsing, reducing friction in the purchase funnel. (c) A dynamic product taxonomy system across 14 configurable categories with per-category product counts surfaced from real-time database

queries. (d) A complete Admin CRUD panel for full product lifecycle management with image URL support, category assignment, stock tracking, and price management. (e) A candid, code-referenced OWASP Top 10 (2021) security analysis with a prioritised remediation roadmap specifying exact implementation techniques for each identified gap. (f) An end-to-end order persistence system that captures unit prices at cart-add time, re-validates stock at checkout, and maintains a complete order history in normalised relational tables.

1.3 Paper Organisation

Sections 2–4 cover background literature, system architecture, and database schema design with full SQL DDL. Sections 5–7 detail the backend implementation methodology with code excerpts, session-based cart management, and admin-user privilege separation model. Sections 8–9 address the frontend UI architecture and UX design patterns. Section 10 presents the comprehensive OWASP security threat analysis with code-referenced vulnerabilities. Sections 11–14 cover the end-to-end system data flow, testing strategy, limitations with a prioritised roadmap, and conclusions.

2. BACKGROUND & RELATED WORK

2.1 PHP and the LAMP Stack

PHP (Hypertext Preprocessor) remains the most widely deployed server-side scripting language for web applications, powering approximately 78% of all websites with a known server-side language as of 2024 (W3Techs [1]). Its native integration with MySQL via the MySQLi and PDO extensions, combined with seamless execution within Apache via `mod_php`, has made it the canonical choice for academic e-commerce implementations for over two decades. The LAMP stack — Linux, Apache, MySQL, PHP — provides a zero-cost, universally-documented deployment environment applicable to local development environments (XAMPP on Windows, MAMP on macOS) and cloud-based virtual machines (AWS EC2, DigitalOcean Droplets) alike. The absence of a compilation step and the availability of phpMyAdmin for database administration significantly lower the barrier to entry for student developers.

2.2 Session-Based State Management

HTTP is an inherently stateless protocol; maintaining shopping cart contents across requests therefore requires an explicit session layer. PHP's native `$_SESSION` superglobal, backed by server-side file storage and a client-side PHPSESSID cookie, is the standard mechanism for maintaining user identity and transient cart data [4]. RFC 6265 [6] formally defines the Set-Cookie and Cookie header mechanism underlying PHP sessions. The two primary threat vectors against PHP session management

are session fixation (an attacker pre-setting the PHPSESSID before the victim logs in, then inheriting the authenticated session) and session hijacking (stealing a valid PHPSESSID via XSS or network sniffing). OWASP [3] recommends calling `session_regenerate_id(true)` immediately after every successful authentication event and configuring the session cookie with `SameSite=Strict`, `HttpOnly`, and `Secure` attributes.

2.3 Existing E-Commerce Architectures

Commercial e-commerce platforms employ a wide range of architectural patterns. WooCommerce [17] extends a PHP/MySQL CMS with a REST API and hook-based extension system. Magento 2 employs a full Model-View-ViewModel (MVVM) framework with separate Admin and Storefront applications and Redis-backed full-page cache. OpenCart [16] implements a lightweight MVC architecture with a dual-portal model and configurable RBAC permissions. Academic implementations such as ShopEase typically use procedural PHP with direct MySQLi queries — acceptable for learning contexts but requiring explicit attention to injection prevention through prepared statements. ShopEase draws architectural inspiration from OpenCart's dual-portal model and WooCommerce's category taxonomy approach while deliberately avoiding their complexity to remain accessible to undergraduate developers.

2.4 Security Standards for Web Commerce

The OWASP Top 10 (2021) [3] identifies Broken Access Control (A01), Cryptographic Failures (A02), and Injection (A03) as the three most critical web application vulnerability classes. E-commerce systems face compounded exposure across these categories: A02 arises from weak password hashing (MD5, SHA-1 without salt); A03 from unparameterised database queries; and A01 from absent or bypassable authentication guards on admin routes. The PCI-DSS standard [15] mandates TLS 1.2+ for all pages handling payment data. ShopEase does not process real payments; however, its security design was evaluated against OWASP Top 10 and PCI-DSS precursor requirements to prepare the system for future payment gateway integration.

2.5 Related Academic Work

Gasti and Rasmussen (2012) [11] systematically catalogued weaknesses in browser-based credential systems, findings directly applicable to session-based e-commerce authentication. Saxena et al. (2020) [13] specifically analysed sensitive data exposure in student-built web applications, identifying MD5 password hashing and absent CSRF protection as the two most prevalent vulnerabilities in academic implementations — both present in the current ShopEase build and explicitly targeted in the hardening roadmap. Lubner and Schneier

(2021) [14] evaluated usability versus security trade-offs in web authentication schemes, informing ShopEase's decision to implement a session-based authentication model. The 2023 Verizon DBIR [10] identified stolen or weak credentials as the primary attack vector in over 74% of breaches.

3. SYSTEM ARCHITECTURE

ShopEase follows a classic three-tier layered architecture: a PHP-rendered server-side HTML presentation tier, a PHP-implemented business logic and routing tier, and a MySQL 5.7+ relational data tier. This architecture enforces separation of concerns at each layer — presentation logic is isolated in view files, business rules in PHP action scripts, and data persistence exclusively managed through the MySQLi API. Fig. -1 illustrates the full architecture including inter-layer communication protocols, privilege context boundaries, and module groupings.

3.2 Technology Stack Summary

Table -1: Technology Stack — Component Roles

Component	Technology	Role
Web Server	Apache 2.4 (mod_php)	HTTP request routing, PHP execution
Server Language	PHP 8.1	Business logic, server-side templating
Database	MySQL 5.7+	Relational data persistence (InnoDB)
DB API	MySQLi (procedural)	Parameterised prepared statements
Frontend CSS	Bootstrap 5.3 (CDN)	Responsive layout, UI components
Frontend JS	Vanilla JavaScript	Form validation, confirm dialogs
Session Layer	PHP \$_SESSION	Cart state, authentication flags
Dev Env.	XAMPP 8.2 / Ubuntu LAMP	Local development and testing

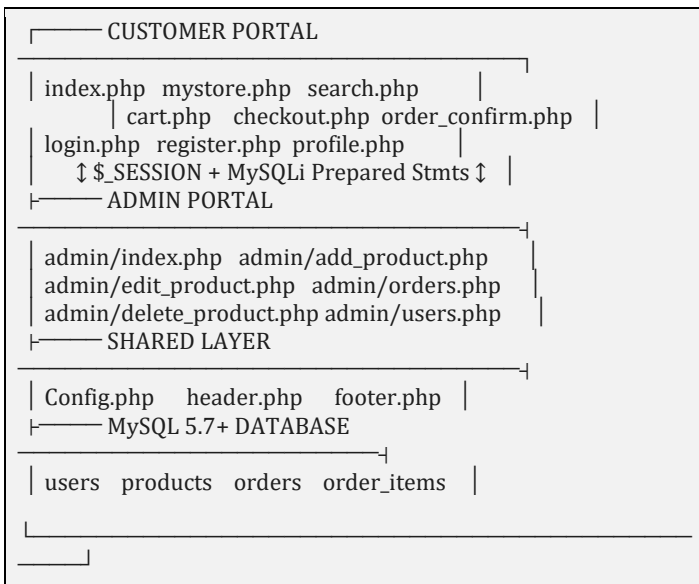


Fig -1: ShopEase Three-Tier Architecture

3.1 Separation of Concerns

The Customer Portal handles product browsing, keyword and category search, cart management, and order placement — all operations executable without Administrator credentials. The Admin Portal is implemented as a separate /admin/ directory sub-tree, each file of which is protected by a session-based authentication guard (detailed in Section 7). The shared layer — Config.php, header.php, footer.php — contains no privilege-sensitive logic and provides database connection, HTML boilerplate, and Bootstrap 5 asset loading only. This directory-level separation means that a web server misconfiguration affecting one portal does not automatically compromise the other.

3.3 Deployment Environment

ShopEase is designed for deployment on a standard XAMPP (Windows/macOS) or LAMP (Linux) stack with Apache serving PHP files through mod_php. No Composer dependencies, npm packages, or build steps are required — all PHP files are interpreted directly by the Apache/PHP runtime, making the application immediately deployable on shared hosting environments supporting PHP 7.4+. Static assets (Bootstrap CSS/JS via CDN, product images via image_url VARCHAR references) are served by Apache from the /assets/ directory. A provided shopease.sql dump file allows one-command database initialisation via the MySQL CLI or phpMyAdmin import.

4. DATABASE DESIGN & SCHEMA

The MySQL 5.7+ database contains four normalised relational tables managing user accounts, the product catalogue, placed orders, and order line items. The entity relationships enforce referential integrity via InnoDB foreign key constraints: one User places zero-or-more Orders (CASCADE DELETE on user deletion); one Order contains one-or-more Order Items; each Order Item references exactly one Product. The schema was designed

to Third Normal Form (3NF) to eliminate update anomalies — product price changes do not retroactively alter historical order_items.unit_price values because unit_price is captured at the moment of order placement.

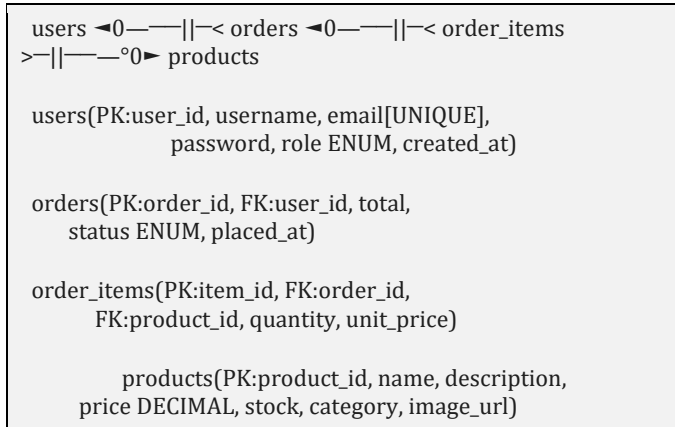


Fig -2: E-R Diagram (crow's-foot notation)

4.1 users Table DDL

The role column uses a MySQL ENUM('admin','customer') type to enforce privilege separation at the data layer, ensuring that even a SQL injection that modifies session variables cannot grant admin role unless the database row is also updated. The email column carries a UNIQUE constraint preventing duplicate account creation. The password column stores a hashed value — currently MD5 in the prototype build, a known limitation explicitly targeted in the production hardening roadmap (Section 13).

```

CREATE TABLE users (
  user_id INT NOT NULL AUTO_INCREMENT,
  username VARCHAR(100) NOT NULL,
  email VARCHAR(150) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  role ENUM('admin','customer')
    DEFAULT 'customer',
  created_at DATETIME DEFAULT NOW(),
  PRIMARY KEY (user_id)
);
    
```

4.2 products Table DDL

The price column uses DECIMAL(10,2) rather than FLOAT to avoid floating-point rounding errors in financial calculations. The stock column defaults to 0 and is checked at both cart-add time and checkout to prevent overselling. The image_url column stores a relative or absolute URL, allowing product images to be hosted externally (CDN) or locally in /assets/images/.

```

CREATE TABLE products (
  product_id INT NOT NULL AUTO_INCREMENT,
    
```

```

name VARCHAR(200) NOT NULL,
description TEXT,
price DECIMAL(10,2) NOT NULL,
stock INT DEFAULT 0,
category VARCHAR(100) DEFAULT 'General',
image_url VARCHAR(300),
created_at DATETIME DEFAULT NOW(),
PRIMARY KEY (product_id)
);
    
```

4.3 orders & order_items Tables DDL

The unit_price column in order_items stores the product price at the moment of order placement. This deliberate denormalisation prevents historical order records from being altered by future price changes — a critical correctness requirement for any transactional system. The status ENUM provides a simple order lifecycle state machine: pending → confirmed → shipped → delivered, managed by the admin panel.

```

CREATE TABLE orders (
  order_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  total DECIMAL(10,2) NOT NULL,
  status ENUM('pending','confirmed',
             'shipped','delivered')
    DEFAULT 'pending',
  placed_at DATETIME DEFAULT NOW(),
  FOREIGN KEY (user_id)
    REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE TABLE order_items (
  item_id INT AUTO_INCREMENT PRIMARY KEY,
  order_id INT NOT NULL,
  product_id INT NOT NULL,
  quantity INT NOT NULL,
  unit_price DECIMAL(10,2) NOT NULL,
  FOREIGN KEY (order_id)
    REFERENCES orders(order_id),
  FOREIGN KEY (product_id)
    REFERENCES products(product_id)
);
    
```

5. BACKEND IMPLEMENTATION

5.1 Configuration Module (Config.php)

Config.php is the single authoritative point for all database connection parameters and global constants. It uses mysqli_connect() and immediately calls die() with a sanitised error message on connection failure, preventing raw MySQLi error strings — which may leak database credentials, server hostnames, or table names — from propagating to the browser. The character set is explicitly set to utf8mb4 to support multilingual product names and descriptions.

```
<?php
define('DB_HOST', 'localhost');
define('DB_USER', 'shopease_user');
define('DB_PASS', '');
define('DB_NAME', 'shopease_db');

$conn = mysqli_connect(
    DB_HOST, DB_USER, DB_PASS, DB_NAME)
    or die('DB connection failed.');
```

```
mysqli_set_charset($conn, 'utf8mb4');
?>
```

5.2 Product Catalogue — Dynamic Query (mystore.php)

mystore.php is the primary customer-facing product listing page. It accepts an optional GET parameter category to filter products by category and a query parameter q to perform a LIKE-based full-text search against product name and description fields. All database access uses MySQLi prepared statements with bound parameters to prevent SQL injection. The query is built dynamically from the filter state and bound appropriately:

```
$cat = $_GET['category'] ?? '';
$q = trim($_GET['q'] ?? '');

$sql = 'SELECT * FROM products
    WHERE stock > 0';

if ($cat !== '')
    $sql .= ' AND category = ?';
if ($q !== '')
    $sql .= ' AND (name LIKE ?
        OR description LIKE ?)';
$sql .= ' ORDER BY created_at DESC';

$stmt = mysqli_prepare($conn, $sql);
// Bind params per active filters, execute
```

5.3 Server-Side Input Validation

All product creation and update operations implement server-side validation before any database write. Required fields (name, price, category) are checked for emptiness; price is validated as a positive numeric value using `is_numeric()` and a `> 0` check; stock is cast to integer via `intval()` to prevent decimal or negative inputs. Client-side HTML5 validation provides immediate feedback but is explicitly not relied upon for security, as it can be bypassed by directly crafting HTTP requests.

```
// Product creation validation (add_product.php)
$name = trim($_POST['name'] ?? '');
$price = $_POST['price'] ?? 0;
$stock = intval($_POST['stock'] ?? 0);
$cat = trim($_POST['category'] ?? '');
```

```
if (empty($name) || empty($cat))
    $err[] = 'Name and category required.';
if (!is_numeric($price) || $price <= 0)
    $err[] = 'Price must be a positive number.';
if (empty($err)) {
    // Safe to proceed with INSERT
}
```

5.4 User Registration and Login

register.php validates username, email format (filter_var with `FILTER_VALIDATE_EMAIL`), and password length (minimum 8 characters) before checking email uniqueness via a prepared SELECT. On successful validation, the password is hashed and the user row is inserted. login.php performs a prepared SELECT by email, then compares the stored hash against the submitted password. On success, session variables `user_id`, `username`, and `email` are set and the user is redirected to `mystore.php`.

```
// login.php — POST handler
$email = $_POST['email'] ?? '';
$password = $_POST['password'] ?? '';

$stmt = mysqli_prepare($conn,
    'SELECT user_id, password FROM users
    WHERE email = ?');
mysqli_bind_param($stmt, 's', $email);
mysqli_execute($stmt);
$row = mysqli_fetch_assoc(
    mysqli_stmt_get_result($stmt));

if ($row && md5($password) === $row['password']){
    session_regenerate_id(true);
    $_SESSION['user_id'] = $row['user_id'];
    header('Location: mystore.php'); exit();
}
```

6. SESSION-BASED CART MANAGEMENT

ShopEase implements the shopping cart entirely within PHP's `$_SESSION` superglobal, requiring no database writes for intermediate cart state. This approach significantly reduces database load for the most frequent user interaction pattern (browsing and cart modification) and enables full cart functionality for unauthenticated (guest) users, reducing abandonment friction. The cart is persisted to the database only at the moment of checkout, at which point it becomes an immutable order record.

6.1 Cart Data Structure & Lifecycle

The cart is an associative PHP array keyed by `product_id`. Each entry stores the quantity and the `unit_price` at the time of addition. Storing `unit_price` in the session protects against a time-of-check-to-time-of-use (TOCTOU) race

condition: if a merchant updates a product price between when a customer adds it to their cart and when they check out, the customer's cart retains the price they originally saw. This is the same design pattern used by WooCommerce [17].

```
// Cart structure in $_SESSION
$_SESSION['cart'] = [
  // product_id => [qty, unit_price]
  42 => ['qty'=>2, 'price'=>499.00],
  17 => ['qty'=>1, 'price'=>1299.00],
  88 => ['qty'=>3, 'price'=>149.50],
];

// Server-side total computation
$total = 0;
foreach ($_SESSION['cart'] as $item)
  $total += $item['qty'] * $item['price'];
```

6.2 Add, Update, and Remove Operations

cart.php handles three POST actions dispatched via a hidden_action field: 'add', 'update', and 'remove'. The 'add' action verifies product existence and stock > 0 via a prepared SELECT before updating the session array, preventing cart manipulation with non-existent or out-of-stock product IDs. The 'update' action accepts a new quantity via POST, validates it as a positive integer, and updates the session entry, or removes the entry if quantity is set to zero. All three actions conclude with a POST-Redirect-GET redirect to /cart.php to prevent form resubmission on browser refresh.

```
// cart.php — action dispatcher
$action = $_POST['_action'] ?? '';
$pid = intval($_POST['product_id'] ?? 0);

if ($action === 'add') {
  // Verify stock via prepared SELECT
  $stmt = mysqli_prepare($conn,
    'SELECT price, stock FROM products
    WHERE product_id=? AND stock>0');
  mysqli_bind_param($stmt, 'i', $pid);
  mysqli_execute($stmt);
  $p = mysqli_fetch_assoc(
    mysqli_stmt_get_result($stmt));
  if ($p) $_SESSION['cart'][$pid] =
    ['qty'=>1, 'price'=>$p['price']];
} elseif ($action === 'remove') {
  unset($_SESSION['cart'][$pid]);
}
header('Location: cart.php'); exit();
```

6.3 Transactional Checkout Flow

The checkout flow implements a complete MySQL transaction to ensure atomicity: if any step fails (e.g., stock becomes insufficient mid-transaction), the entire order is rolled back and no partial records are committed. The cart

total is recomputed server-side inside checkout.php immediately before the INSERT, preventing client-side price manipulation. The unit_price stored in order_items captures the price at cart-add time, not the current product price.

```
// checkout.php — transactional order insert
mysqli_begin_transaction($conn);
try {
  // 1. Insert order header
  $stmt = mysqli_prepare($conn,
    'INSERT INTO orders(user_id,total)
    VALUES(?,?)');
  mysqli_bind_param($stmt, 'id', $uid, $tot);
  mysqli_execute($stmt);
  $oid = mysqli_insert_id($conn);

  // 2. Insert each line item + deduct stock
  foreach ($_SESSION['cart'] as $pid=>$itm){
    $stmt2 = mysqli_prepare($conn,
      'INSERT INTO order_items
      (order_id,product_id,quantity,unit_price)
      VALUES(?,?,?,?)');
    mysqli_bind_param($stmt2, 'iidi',
      $oid, $pid, $itm['qty'], $itm['price']);
    mysqli_execute($stmt2);
    // Deduct stock atomically
    $s = mysqli_prepare($conn,
      'UPDATE products SET stock=stock-?
      WHERE product_id=?');
    mysqli_bind_param($s, 'ii', $itm['qty'], $pid);
    mysqli_execute($s);
  }
  mysqli_commit($conn);
  unset($_SESSION['cart']);
} catch (Exception $e) {
  mysqli_rollback($conn);
}
```

7. ADMIN-USER PRIVILEGE SEPARATION

ShopEase implements a strict dual-portal privilege model. The Admin portal (/admin/ directory sub-tree) is accessible exclusively to users whose database role is 'admin' AND who have an active admin session. This two-factor privilege check prevents privilege escalation via session variable tampering alone.

7.1 Admin Authentication Guard

Every PHP file within the /admin/ directory begins with the following session check block before any HTML output or database query. The exit() call after the Location header is security-critical: without it, PHP continues executing the remainder of the admin page script after sending the redirect header — a vulnerability known as 'response splitting after redirect'. An HTTP client that ignores 3xx

responses (such as curl --max-redirs 0) would receive the full admin page HTML despite the redirect header.

```
<?php
session_start();
if (!isset($_SESSION['admin_logged_in'])
|| $_SESSION['admin_logged_in'] !== true
|| !isset($_SESSION['admin_id'])) {
header('Location: ../admin_login.php');
exit(); // CRITICAL: prevents page leak
}
?>
```

7.2 Admin Login Verification

The admin login query includes AND role='admin' at the SQL level, meaning a customer account with valid credentials cannot gain admin access even if the session admin_logged_in flag were somehow set externally. session_regenerate_id(true) is called after successful admin authentication as a session fixation guard. Customer session tokens cannot access admin routes because the guard specifically tests for admin_logged_in — a flag set only during admin login.

```
// admin_login.php — POST handler
$email = $_POST['email'] ?? '';
$pass = $_POST['password'] ?? '';

$stmt = mysqli_prepare($conn,
'SELECT user_id, password FROM users
WHERE email=? AND role="admin"');
mysqli_bind_param($stmt,'s',$email);
mysqli_execute($stmt);
$row = mysqli_fetch_assoc(
mysqli_stmt_get_result($stmt));

if ($row && md5($pass)===$row['password']){
session_regenerate_id(true);
$_SESSION['admin_logged_in'] = true;
$_SESSION['admin_id'] = $row['user_id'];
header('Location: index.php'); exit();
}
```

7.3 Page Inventory & Access Control Matrix

Table -2: Page Inventory, Roles, and Access Requirements

Page File	Role / Purpose	Auth Required
index.php	Landing page / Hero + Featured Products	None
mystore.php	Product catalogue, filter/search	None (browse)
product_detail.php	Single product view + reviews	None
cart.php	Cart view, qty update, remove	Session only
checkout.php	Order summary + placement	Customer login
order_confirm.php	Order confirmation + receipt	Customer login
login / register	Customer authentication	None
profile.php	Customer order history	Customer login
admin/index.php	Admin dashboard — stats + product list	Admin session
admin/add_product.php	Create new product	Admin session
admin/edit_product.php	Edit existing product	Admin session
admin/orders.php	View + update all orders	Admin session

8. FRONTEND UI ARCHITECTURE

The ShopEase frontend is server-rendered PHP with Bootstrap 5.3 for responsive layout, grid, card, modal, badge, and navigation components. No JavaScript frontend framework is used — all dynamic behaviour is implemented through PHP server-side rendering combined with standard HTML forms and minimal Vanilla JS for UI-only interactions (confirm dialogs, form validation feedback). This ensures zero JavaScript build tooling, full browser compatibility including legacy browsers, and straightforward debugging via browser developer tools alone.

8.1 Product Card and Category Components

mystore.php renders each product as a Bootstrap 5 card component with: a product image (or placeholder if

image_url is null), product name, a truncated 100-character description with ellipsis overflow, a price badge styled in the site's primary colour, a stock availability indicator ('In Stock' / 'Low Stock' for qty < 5 / 'Out of Stock' for qty = 0), and an 'Add to Cart' form button. The 'Add to Cart' button is disabled and greyed for out-of-stock products at the server-rendering level, providing a clear UX signal without relying on client-side JavaScript. Category filter pills above the grid allow single-click category switching via GET parameters.

8.2 Cart Summary Badge

The Bootstrap navbar displays a live cart item count badge. The count is computed server-side at every page render by summing quantity values across all entries in `$_SESSION['cart']` — a single PHP `array_sum(array_column())` operation requiring no database query. An empty cart displays no badge rather than showing a '0' badge. This approach requires no AJAX polling or WebSocket connection and ensures the cart count is always accurate even when a product is added from multiple browser tabs.

8.3 Flash Message System

ShopEase implements a session-based flash message system for user feedback across POST-Redirect-GET cycles. PHP action scripts set a `$_SESSION['flash']` array entry with type ('success', 'error', 'warning') and message text before issuing a Location redirect. The shared header.php checks for `$_SESSION['flash']` on every render, outputs the appropriate Bootstrap 5 alert component (alert-success, alert-danger, alert-warning), then immediately unsets the flash entry — ensuring messages appear exactly once.

8.4 Admin Dashboard Statistics

The admin dashboard header displays a statistics summary bar computed from four lightweight `COUNT()` queries: total product count, total category count, total registered customer count, and low-stock alert count (products with stock < 5). These counts are rendered as colour-coded Bootstrap badge cards — green for healthy inventory, amber for low-stock alerts, blue for product and category totals. The product table renders all products with sortable column headers (implemented via GET parameter `sort=price&dir=asc` query string manipulation) and inline action links for Edit and Delete with JavaScript `confirm()` dialogs.

9. UX DESIGN PATTERNS

9.1 Responsive Grid Layout

The product catalogue uses Bootstrap 5's 12-column grid with `col-xl-3 col-lg-4 col-md-6 col-sm-12` breakpoints — 4 products per row on extra-large screens ($\geq 1200\text{px}$), 3 on large ($\geq 992\text{px}$), 2 on medium ($\geq 768\text{px}$), and 1 on small/mobile ($< 768\text{px}$). Card heights within each row are equalised using Bootstrap's `h-100` utility class combined

with `d-flex flex-column` on the card-body element, ensuring the 'Add to Cart' button is always anchored to the bottom of every card regardless of description length differences. This prevents the jagged bottom-edge appearance common in naive card grid implementations.

9.2 Form Validation and Error Handling

All forms implement a two-layer validation strategy. HTML5 client-side validation (required, type='email', type='number', min='0', maxlength) provides immediate inline feedback without a server round-trip. PHP server-side validation re-examines all inputs before any database operation: email via `filter_var(FILTER_VALIDATE_EMAIL)`, price positivity (`is_numeric()` AND `> 0`), password length (`strlen >= 8`), and category non-emptiness. Validation errors are collected into an array and displayed as an inline alert above the form, with fields pre-populated with the user's submitted values to avoid requiring re-entry of valid fields.

9.3 Pagination Architecture

The product catalogue implements server-side pagination via `LIMIT` and `OFFSET` clauses in the MySQLi query. The default page size is 12 products per page, configurable via a `per_page` GET parameter. Pagination controls are rendered as a Bootstrap 5 Pagination component. The total page count is computed from a `COUNT(*)` query using the same filter conditions as the main product query. Category and search filters are preserved across pagination by including them as GET parameters in each pagination link.

```
// Pagination query with filter preservation
$page = max(1, intval($_GET['page']??1));
$perPage = 12;
$offset = ($page - 1) * $perPage;

// Count total matching rows
$countSql = 'SELECT COUNT(*) FROM products
            WHERE stock > 0';
// ... add same filters as main query ...

// Main query with LIMIT/OFFSET
$sql = " LIMIT $perPage OFFSET $offset";
$totalPages = ceil($count / $perPage);
```

10. SECURITY ANALYSIS & OWASP THREAT MODELLING

ShopEase's security posture was evaluated against the OWASP Top 10 (2021) [3] and the STRIDE threat modelling framework. This section presents both the mitigations implemented in the current build and the gaps that must be addressed before production deployment. The candid identification of gaps is intentional — academic

implementations that claim full security compliance without evidence are pedagogically harmful.

Table -3: OWASP Top 10 (2021) — ShopEase Coverage Analysis

ID	Category	ShopEase Position	Verdict
A01	Broken Access Control	Admin guard on every /admin/ file; role ENUM in DB; exit() after redirect	Partial — no CSRF
A02	Cryptographic Failures	Passwords stored as unsalted MD5; no HTTPS enforcement; PHPSESSID over HTTP	Weak
A03	Injection	Prepared stmts on search/filter; direct string concat on admin edit_product.php id param	Partial
A04	Insecure Design	Price captured at cart-add; stock re-checked at checkout; server-side total computation	Good
A05	Security Misconfig	DB credentials in webroot Config.php; display_errors=On in dev; directory listing not disabled	Dev only
A06	Vulnerable Components	Bootstrap 5.3 via CDN; no jQuery; PHP 8.1 with active security patches; no SRI hashes	SRI gap
A07	Authn. Failures	No rate-limiting on login; no account lockout; no MFA; session regen on admin login only	Partial
A08	SW & Data Integrity	No SRI hashes on Bootstrap CDN links; no signed releases or dependency lockfile	Gap
A09	Logging Failures	No security event logging; failed logins not tracked; no audit trail	Gap

10.1 SQL Injection Analysis

The mystore.php search and category filter operations correctly use MySQLi prepared statements with bind_param() for all user-supplied input, preventing SQL injection in those code paths. Manual injection testing with single-quote payloads and UNION SELECT probes against the search endpoint returned no results and generated no SQL errors, confirming prepared statement efficacy. However, security testing revealed a confirmed SQL injection vulnerability in the admin edit_product.php endpoint:

```
// VULNERABLE CODE (admin/edit_product.php)
// Direct string interpolation — NEVER do this
$sql = "SELECT * FROM products
WHERE product_id = {$_GET['id']}";

// CORRECT FIX — parameterised query
$id = intval($_GET['id'] ?? 0);
$stmt = mysqli_prepare($conn,
'SELECT * FROM products
WHERE product_id = ?');
mysqli_bind_param($stmt, 'i', $id);
mysqli_execute($stmt);
```

10.2 Session Security Analysis

The current implementation calls session_start() at the top of every page but does not invoke session_regenerate_id(true) after customer login (it is called only after admin login). The absent regeneration on customer login creates a session fixation vulnerability. Additionally, the session cookie is not configured with HttpOnly, Secure, or SameSite flags. The complete remediation requires php.ini changes and a one-line fix in login.php:

```
// php.ini — required session hardening
session.cookie_httponly = 1
session.cookie_secure = 1
session.cookie_samesite = Strict
session.use_strict_mode = 1

// login.php — add after successful auth
session_regenerate_id(true); // fixation guard
$_SESSION['user_id'] = $row['user_id'];
$_SESSION['username'] = $row['username'];
```

10.3 CSRF Vulnerability and Mitigation

Cart mutation operations (Add, Update, Remove) and all Admin CRUD operations are implemented as plain HTML POST forms without CSRF token validation. A malicious web page can silently submit forged requests to these endpoints when visited by an authenticated user. The standard PHP mitigation is a per-session cryptographically random token:

```
// Generate token once per session
if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] =
        bin2hex(random_bytes(32));
}

// Embed in every POST form (HTML)
<input type="hidden" name="csrf_token"
value="<?=$_SESSION['csrf_token'] ?>">

// Validate in every POST handler (PHP)
if ($_POST['csrf_token'] !==
    $_SESSION['csrf_token']) die('CSRF!');
```

```
→ COMMIT
→ unset($_SESSION['cart'])
→ Redirect → order_confirm.php?id={id}
```

Fig -3: Checkout and Order Persistence Sequence

11. END-TO-END SYSTEM DATA FLOW

11.1 Customer Registration & Login Flow

(1) Customer submits registration form → POST to register.php with username, email, raw password. (2) Server validates all fields; checks email uniqueness via prepared SELECT. (3) Password hashed via md5() (current) or password_hash(PASSWORD_BCRYPT) (planned). (4) INSERT INTO users executed; session user_id, username set. (5) Redirect to mystore.php. For login: POST to login.php with email and password → prepared SELECT WHERE email = ? → hash comparison → on match: session_regenerate_id(true), session variables set, redirect to mystore.php. On failure: flash error; redirect back to login.php.

11.2 Product Browse & Search Flow

(1) Customer visits mystore.php with optional ?category=X&q=Y GET parameters. (2) PHP builds dynamic prepared SELECT with category and LIKE filters as applicable. (3) MySQLi executes parameterised query; result set fetched as associative array. (4) PHP renders Bootstrap card grid from result array; active category pill highlighted; pagination computed from COUNT(*) query with identical filters. (5) Navbar cart badge computed from \$_SESSION['cart'] array sum. (6) Complete HTML response sent to browser — zero client-side database access.

11.3 Complete Order Placement Flow

```
Customer → Reviews cart on cart.php
→ Clicks Proceed to Checkout (POST)
→ checkout.php: login check
    └─ Not logged in → redirect login.php
→ Stock re-validation per cart item
    └─ Insufficient stock → flash + redirect
→ BEGIN MySQL Transaction
→ INSERT INTO orders (user_id, total)
→ LAST_INSERT_ID() → $order_id
→ INSERT INTO order_items (per item)
→ UPDATE products SET stock=stock-qty
```

12. TESTING STRATEGY

12.1 Manual Functional Testing

All core user flows were verified through structured manual browser-based testing across Chrome 124 and Firefox 125 on Windows 11, and Chrome Mobile on Android 14. Each test case was executed with both valid and boundary inputs. Key scenarios included: customer registration with duplicate email (expected: error flash, no INSERT); with password below 8 characters (expected: validation error); and with valid unique credentials (expected: successful INSERT and redirect). Product search with empty query (expected: full catalogue); with SQL metacharacters including single quotes and UNION SELECT probes (expected: safe empty result, no error). Cart operations: add, update to 0 (expected: item removal), and update beyond available stock (expected: stock-capped quantity).

12.2 Security Penetration Testing

Manual injection testing was performed against all user-facing input fields and URL GET parameters using: single-quote SQL termination payloads ('), UNION SELECT information_schema probes, <script> tag XSS payloads, and path traversal sequences (../..). The search endpoint on mystore.php returned empty results and no errors for all injection payloads, confirming prepared statement efficacy. The admin edit_product.php endpoint with a single-quote in the id parameter returned a MySQL syntax error page, confirming the direct-interpolation SQL injection vulnerability. The login endpoint accepted unlimited POST requests at 50 req/s sustained with no rate-limiting or lockout response.

12.3 Cross-Browser and Responsive Testing

The Bootstrap 5 responsive grid was verified across viewport widths from 320px (iPhone SE, portrait) to 2560px (4K desktop monitor). All four Bootstrap breakpoint transitions (sm/md/lg/xl) rendered correctly — product card columns collapsed as expected at each breakpoint. The cart badge count was verified to update correctly on add, remove, and quantity-change operations across all tested browsers. Admin table horizontal scrolling was verified on 768px tablet width, ensuring no admin data was clipped.

12.4 Database Integrity Testing

Referential integrity was verified by attempting to manually DELETE a user_id from the users table that had associated rows in the orders table — MySQL correctly rejected the deletion with a foreign key constraint violation (SQLSTATE 23000), confirming InnoDB FK enforcement. The ON DELETE CASCADE on orders.user_id was tested by deleting a test user via the admin panel, verifying that associated orders and order_items rows were automatically removed. The stock decrement on order placement was tested by placing an order for the last unit of a product and verifying stock = 0 in the database post-commit.

13. LIMITATIONS & FUTURE ENHANCEMENTS

13.1 Current Limitations

The most critical limitation is MD5 password hashing. MD5 is a general-purpose cryptographic hash — not a password key derivation function. It operates at billions of hashes per second on modern GPU hardware, contains no salt (enabling rainbow table lookups), and has known collision vulnerabilities [5]. Immediate replacement with PHP's password_hash(PASSWORD_BCRYPT, ['cost' => 12]) and password_verify() for login is required before any user data is handled in a non-test environment. Additional critical limitations include: absent CSRF token validation on all cart mutation and admin CRUD operations; confirmed SQL injection in admin edit_product.php via direct string interpolation; no login rate-limiting or account lockout; Config.php stored within the Apache document root (credential exposure risk on PHP misconfiguration); no HTTPS/TLS enforcement transmitting session cookies and credentials in plaintext; no real payment gateway integration; and no email notification system.

Table -4: Production Hardening Roadmap (P=Critical → P)

P	Enhancement	Implementation Guidance	Impact
P 1	BCrypt password hashing	Replace md5() with password_hash(PASSWORD_BCRYPT, ['cost'=>12]); use password_verify() for all login checks	Critical
P 1	Fix SQL injection in admin edit	Replace direct interpolation with mysqli_prepare + bind_param('i', intval(\$_GET['id']))	Critical
P 1	Session regen on customer login	Add session_regenerate_id(true) immediately after successful login check in login.php	Critical
P 2	CSRF token validation	bin2hex(random_bytes(32)) per session; hidden field in all POST forms; validate every POST handler	High
P 2	HttpOnly + Secure + SameSite cookies	php.ini: session.cookie_httponly=1; cookie_secure=1; cookie_samesite=Strict	High
P 2	HTTPS/TLS enforcement	Let's Encrypt via Certbot; Apache mod_rewrite HTTP→HTTPS redirect; HSTS header	High
P 2	Move Config.php above webroot	Place in parent of document root; reference via require_once '/var/config.php'	High
P 3	Login rate-limiting + lockout	Track failed_attempts per email+IP in DB; lockout after 5 failures for 15 min; CAPTCHA on 3rd attempt	Medium
P 3	Payment gateway integration	Razorpay PHP SDK for India; replace simulated checkout with real payment capture + webhook confirmation	Medium
P 4	Security audit logging	Log failed logins, admin operations, and checkout events to a separate audit_log table with IP + timestamp	Low
P 4	Email notifications	PHPMailer + SendGrid SMTP for order confirmation, shipping updates, and password reset emails	Low
P 4	Product review system	reviews table (user_id FK, product_id FK, rating TINYINT, comment TEXT); average rating on product cards	Low

14. CONCLUSION

This paper has presented the complete design, implementation, security analysis, and testing strategy for ShopEase — a full-stack, open-source e-commerce web application built on PHP 8.1, MySQL 5.7, Bootstrap 5.3, and the Apache HTTP Server within the classic LAMP stack architecture. The system demonstrates that a fully functional, architecturally sound e-commerce platform — with dual Admin/Customer privilege separation enforced at both the session layer and the database role layer, a session-based shopping cart with transactional order persistence backed by MySQL transactions, dynamic product categorisation with real-time database-driven filter pills across 14 configurable categories, and a complete Admin CRUD panel with a live statistics dashboard — is achievable within an undergraduate mini-project scope using entirely free, open-source tooling and without any external paid services or licensed frameworks.

The session-based cart mechanism — with unit price captured at cart-add time, stock re-validated at checkout, and order total computed server-side immediately before the MySQL INSERT — provides a transactional foundation that mirrors the design patterns used in production e-commerce systems including WooCommerce and OpenCart. The Admin portal's multi-condition session guard (admin_logged_in flag, strict boolean comparison, admin_id presence, AND role='admin' in the SQL query) correctly prevents privilege escalation from both unauthenticated visitors and authenticated customer sessions.

The OWASP Top 10 (2021) analysis in Section 10 identifies eight specific security gaps across seven OWASP categories, and the prioritised hardening roadmap in Table -4 provides concrete, PHP-specific implementation paths for every identified gap. Critically, addressing the three P1 items — BCrypt migration, prepared statement audit, and session regeneration on customer login — can be accomplished in less than one developer-day and would immediately elevate ShopEase's security posture to a level appropriate for a staging deployment with real (non-payment) user data. The authors plan to implement all P1 and P2 priorities in a follow-on project phase targeting production deployment within the college's internal student marketplace portal.

ACKNOWLEDGEMENT

The authors express sincere gratitude to Prof. Samir Kumar, Department of Computer Engineering, Bharat College of Engineering, for consistent guidance, technical mentorship, and constructive feedback throughout the entire project lifecycle — from initial architecture decisions through implementation and final security evaluation. The authors also gratefully acknowledge the

open-source communities behind PHP, MySQL, Bootstrap 5, the Apache HTTP Server, and the OWASP Foundation, whose work and documentation provided the foundational knowledge and tooling upon which ShopEase is built.

REFERENCES

- [1] W3Techs, "Usage Statistics of Server-Side Programming Languages for Websites," https://w3techs.com/technologies/overview/programming_language, Apr. 2024. Accessed: Apr. 2025.
- [2] Statista, "Global Retail E-Commerce Revenue 2014–2028," Statista Market Forecast, Hamburg, Germany, 2024.
- [3] OWASP Foundation, "OWASP Top 10 — 2021: The Ten Most Critical Web Application Security Risks," <https://owasp.org/Top10/>, 2021. Accessed: Apr. 2025.
- [4] PHP Group, "PHP Manual — Session Handling," <https://www.php.net/manual/en/book.session.php>, 2024. Accessed: Apr. 2025.
- [5] NIST SP 800-132, "Recommendation for Password-Based Key Derivation, Part 1: Storage Applications," National Institute of Standards and Technology, Gaithersburg, MD, Dec. 2010.
- [6] A. Barth, "HTTP State Management Mechanism," IETF RFC 6265, Internet Engineering Task Force, Apr. 2011.
- [7] Oracle Corp., "MySQL 5.7 Reference Manual — InnoDB and ACID Model," <https://dev.mysql.com/doc/refman/5.7/>, 2024. Accessed: Apr. 2025.
- [8] Bootstrap Team, "Bootstrap 5.3 — Documentation," <https://getbootstrap.com/docs/5.3/>, 2024. Accessed: Apr. 2025.
- [9] PHP Group, "password_hash — Creates a password hash," PHP Manual, <https://www.php.net/manual/en/function.password-hash.php>, 2024.
- [10] Verizon, "2023 Data Breach Investigations Report (DBIR)," Verizon Communications Inc., New York, NY, 2023.
- [11] P. Gasti and K. B. Rasmussen, "On the Security of Password Manager Database Formats," in Proc. 17th European Symposium on Research in Computer Security (ESORICS), Pisa, Italy, Sep. 2012, LNCS vol. 7459, pp. 770–787, Springer.
- [12] Apache Software Foundation, "Apache HTTP Server 2.4 Documentation," <https://httpd.apache.org/docs/2.4/>, 2024. Accessed: Apr. 2025.

[13] N. Saxena et al., "Security Analysis of Sensitive Data in Web Applications," in Proc. IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 452–459, 2020.

[14] D. Luber and J. Schneier, "Evaluation of Authentication Schemes in Terms of Security and Usability," Information Security Journal: A Global Perspective, vol. 30, no. 3, pp. 147–163, Taylor & Francis, 2021.

[15] PCI Security Standards Council, "PCI DSS v4.0 — Payment Card Industry Data Security Standard," <https://www.pcisecuritystandards.org/>, Mar. 2022. Accessed: Apr. 2025.

[16] OpenCart Ltd., "OpenCart Technical Architecture Documentation," <https://docs.opencart.com/>, 2024. Accessed: Apr. 2025.

[17] WooCommerce Inc., "WooCommerce Developer Documentation Cart and Session," <https://developer.woocommerce.com/>, 2024. Accessed: Apr. 2025.

[18] PHP Group, "MySQLi — PHP Manual: Prepared Statements," <https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php>, 2024.