MotoParts Manager

Technical Manual

Version: 1.4

Date: July 2025

System: MotoParts Manager

Project Code: CSS152L_AM5_11

Document Type: Technical Implementation

Guide Audience: Developers, System Administrators, and IT Personnel

Purpose: Technical specifications, architecture, and maintenance procedures

---

Table of Contents

---

1. Project Overview

MotoParts Manager is a comprehensive e-commerce platform designed for motorcycle parts retail management. The system provides multi-role access for administrators, retailers, and customers with integrated payment processing, inventory management, and business analytics.

Key Features

- Multi-role user management (Admin, Retailer, Customer)
- Product catalog with image uploads
- Shopping cart and checkout system
- PayRex payment gateway integration
- Order management and tracking
- Business analytics and reporting
- Inventory management
- Customer order history
- Export functionality for orders and reports

Technology Stack

- Backend: PHP 8.x with MySQL

- Frontend: HTML5, CSS3, JavaScript (ES6+)
- Database: MySQL 8.x
- Web Server: Apache/Nginx (localhost)
- Payment Gateway: Payrex (PayMongo alternative)
- Charts: Chart.js for data visualization
- Icons: Boxicons
- Fonts: Google Fonts (Poppins)

---

2. System Architecture

Directory Structure

```
MotoParts/
├── index.php               # Main landing page
├── customerpage.php        # Customer dashboard
├── retailerpage.php        # Retailer dashboard
├── adminpage.php           # Administrator dashboard
├── assets/
│   ├── css/                # Stylesheets
│   ├── js/                 # JavaScript files
│   └── images/             # Product images
├── includes/
│   ├── config.php          # Database configuration
│   ├── auth.php            # Authentication functions
│   └── functions.php       # Common functions
├── modules/
│   ├── search/             # Search functionality
│   ├── cart/               # Shopping cart
│   ├── orders/             # Order processing
│   ├── products/           # Product management
│   ├── users/              # User management
│   └── reports/            # Reporting system
└── database/
    └── schema.sql          # Database schema
```

System Flow Architecture

Client Browser → Web Server → PHP Application → MySQL Database ↓ Payment Gateway (Payrex) ↓ Email Notification System

Core Components

1. Authentication System: Role-based access control
2. Product Management: CRUD operations for inventory
3. Order Processing: Cart to checkout workflow
4. Payment Processing: PayRex integration
5. Analytics Engine: Sales and expense tracking
6. Export System: CSV/PDF report generation

---

3. Database Design

Entity Relationship Diagram

[Users]⟶[Orders]⟶[Order_Items]⟵[Products]

The database consists of 8 main tables with relationships:

- **Users**
- ⟶
- **Orders (One-to-Many)**
- **Users**
- ⟶
- **Products (Retailers can have multiple products)**
- **Products**
- ⟶
- **Categories (Many-to-Many via junction table)**

- Orders
- ⟶
- Order Items (One-to-Many)
- Users
- ⟶
- Shopping Cart (One-to-Many)

Tables Schema

1. users

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    username VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('admin', 'retailer', 'customer') NOT NULL,
    address TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. orders

```
CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_email VARCHAR(255) NOT NULL,
    product_name VARCHAR(255) NOT NULL,
    quantity INT NOT NULL,
    total_price DECIMAL(10, 2) NOT NULL,
    order_date DATE NOT NULL,
    order_status ENUM('Pending', 'Unshipped', 'Shipped', 'Return Request')
DEFAULT 'Pending',
    payment_status ENUM('Pending', 'Paid', 'Failed') DEFAULT 'Pending',
    shipping_date DATE,
    tracking_number VARCHAR(100),
    payment_method ENUM('Credit Card', 'Paypal', 'Bank Transfer') DEFAULT
'Credit Card',
    Order_Address TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. inventory

```
CREATE TABLE inventory (
    inventory_id INT AUTO_INCREMENT PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    product_desc TEXT,
    quantity INT NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL,
    image_path VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

4. cart

```
CREATE TABLE cart (
    cart_id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    inventory_id INT NOT NULL,
    quantity INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (inventory_id) REFERENCES inventory(inventory_id)
);
```

5. expenses

```
CREATE TABLE expenses (
    expense_id INT AUTO_INCREMENT PRIMARY KEY,
    amount DECIMAL(10, 2) NOT NULL,
    description TEXT,
    expense_date DATE NOT NULL
);
```

6. sales

```
CREATE TABLE sales (
    sales_id INT AUTO_INCREMENT PRIMARY KEY,
    sales_date DATE NOT NULL,
    total_price DECIMAL(10, 2) NOT NULL
);
```

7. payment_summary

```
CREATE TABLE payment_summary (
    id INT AUTO_INCREMENT PRIMARY KEY,
    balance DECIMAL(10, 2) NOT NULL
);
```

8. system_logs

```
CREATE TABLE system_logs (
    log_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    action VARCHAR(255),
    table_affected VARCHAR(100),
    record_id VARCHAR(50),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ip_address VARCHAR(45),
    user_agent TEXT,
     FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

---

4. API Documentation

Authentication Endpoints

- POST /api/auth/login
    - Request Body:

```
        {
            "email": "user@example.com",
            "password": "userpassword",
            "user_type": "customer"
        }

```

Response:

```
        {
            "status": "success",
            "message": "Login successful",
            "user_id": 123,
            "redirect_url": "/customerpage.php"
        }
```

Product Endpoints

- GET /api/products/search
    - Parameters:
        - `q` (string): Search query
        - `category` (array): Category IDs
        - `page` (int): Page number
        - `limit` (int): Results per page
- POST /api/products/add (Retailer only)
    - Request Body:

```
   {
            "name": "Brake Pads",
            "description": "High-quality brake pads",
            "category_ids": [1, 3],
            "price": 1299.00,
            "quantity": 50,
            "image": "base64_encoded_image"
        }

```

Cart Endpoints

- POST /api/cart/add
    - Request Body:

```
{
            "product_id": 123,
            "quantity": 2
        }
```

Order Endpoints

- POST /api/orders/create
    - Request Body:

```
{
            "payment_method": "payrex",
            "shipping_date": "2025-08-01",
            "billing_address": {}
        }
```

Payment API Endpoints

Create Payment Intent

- Endpoint: `POST /create_payment_intent.php`

```
// Request
{
    "amount": 1000,
    "currency": "php",
    "metadata": {
        "order_id": "12345",
        "customer_email": "customer@example.com"
    }
}

// Response
{
    "client_secret": "pi_1234567890_secret_abcdef123456",
    "status": "success"
}
```

Payment Callback

- Endpoint: `POST /payrex_callback.php`
    - Handles payment success/failure
    - Updates order status in database
    - Sends confirmation email

Webhook Handler

- Endpoint: `POST /webhook/payrex_webhook.php`
    - Processes webhook events:
        - `payment_intent.succeeded`
        - `payment_intent.failed`
        - `payment_intent.canceled`

---

5. User Roles & Permissions

Role-Based Access Control (RBAC)

Admin

- Access: `adminpage.php`
- Permissions:
  - View all orders and users
  - Manage inventory (CRUD)
  - Add business expenses
  - Generate business reports
  - Export data (CSV/PDF)
  - View analytics dashboard
  - User Management
  - System Logs

Retailer

- Access: `retailerpage.php`
- Permissions:
  - View assigned products
  - Manage own inventory
  - View sales reports
  - Update product information
  - Add Products
  - System Reports

Customer

- Access: `customerpage.php`
- Permissions:
  - Browse products
  - Add to cart
  - Place orders
  - View order history
  - Update profile information
  - Place Orders

Session Management

```
// Authentication check
```

```
session_start();
if (!isset($_SESSION['email']) || $_SESSION['role'] !==
'desired_role') {
    header("Location: login.php");
    exit();
}
```

Session Configuration:

```
  ini_set('session.cookie_httponly', 1);
    ini_set('session.use_strict_mode', 1);
    ini_set('session.cookie_secure', 1); // HTTPS only
```

- Session Variables Stored:
  - `$_SESSION['user_id']`
  - `$_SESSION['user_type']`
  - `$_SESSION['email']`
  - `$_SESSION['login_time']`

---

6. Payment Integration

PayRex Configuration

Setup Requirements

1. API Keys: Obtain from PayRex dashboard
2. Webhook URL: Configure in PayRex settings
3. Payment Methods: Enable desired payment options

Configuration File

```
// config.php additions
define('PAYREX_SECRET_KEY', 'sk_test_your_secret_key_here');
define('PAYREX_PUBLISHABLE_KEY',
'pk_test_your_publishable_key_here');
define('PAYREX_WEBHOOK_SECRET', 'whsec_your_webhook_secret_here');
```

Payment Flow

1. Customer Checkout → `checkout.php`
2. Create Order Record → `submit_order.php`
3. Create Payment Intent → `create_payment_intent.php`
4. Process Payment → PayRex handles payment
5. Webhook Notification → `webhook/payrex_webhook.php`
6. Update Order Status → Database update
7. Customer Confirmation → Email notification

---

7. Installation Guide

System Requirements

- Web Server: Apache 2.4+ or Nginx 1.18+
- PHP: Version 8.0 or higher
- Database: MySQL 8.0 or MariaDB 10.5+
- Memory: Minimum 512MB RAM
- Storage: Minimum 1GB available space
- Composer (for dependency management)

Step-by-Step Installation

1. Clone Repository

```
git clone [repository-url]
```

```
cd MotoParts
```

2. Database Setup

```
-- Create database
CREATE DATABASE motoparts_db CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;

-- Create user and grant privileges
CREATE USER 'motoparts_user'@'localhost' IDENTIFIED BY
'secure_password';
GRANT SELECT, INSERT, UPDATE, DELETE ON motoparts_db.* TO
'motoparts_user'@'localhost';
FLUSH PRIVILEGES;

-- Import schema (assuming schema.sql is in database/ directory)
mysql -u motoparts_user -p motoparts_db < database/schema.sql
```

3. File Deployment

```
# Copy files to web server directory
sudo cp -r MotoParts/ /var/www/html/

# Set proper permissions
sudo chown -R www-data:www-data /var/www/html/MotoParts/
sudo chmod -R 755 /var/www/html/MotoParts/
sudo chmod -R 777 /var/www/html/MotoParts/assets/images/ # Write
access needed for image uploads
```

4. Configuration

Update `includes/config.php` with database and base URL settings:

```
// includes/config.php
define('DB_HOST', 'localhost');
define('DB_NAME', 'motoparts_db');
define('DB_USER', 'motoparts_user');
define('DB_PASS', 'secure_password');
define('BASE_URL', 'https://yourdomain.com/MotoParts/');
```

5. Install Dependencies

```
composer install
```

6. PayRex Setup

1. Create PayRex account
2. Get API keys from dashboard
3. Update configuration with keys (see "Configuration" section)
4. Set webhook URL: `https://yourdomain.com/webhook/payrex_webhook.php`

7. SSL Certificate Setup

```
# Let's Encrypt SSL certificate
sudo certbot --apache -d yourdomain.com
```

---

8. Configuration

Environment Variables

For production, it's recommended to use environment variables. Example `.env` file:

```
# Database
DB_HOST=localhost
DB_NAME=motoparts_prod
DB_USER=motoparts_user
DB_PASS=secure_production_password

# PayRex
```

```
PAYREX_SECRET_KEY=sk_test_xxx
PAYREX_PUBLISHABLE_KEY=pk_test_xxx
PAYREX_WEBHOOK_SECRET=whsec_xxx

# Application
APP_URL=https://yourdomain.com/MotoParts
APP_DEBUG=false
```

PHP Settings (`php.ini`)

For production environment:

```
; Security settings
expose_php = Off
display_errors = Off
log_errors = On
error_log = /var/log/php_errors.log

; Session settings
session.cookie_secure = 1
session.cookie_httponly = 1
session.use_strict_mode = 1

; Upload settings
file_uploads = On
upload_max_filesize = 5M
post_max_size = 10M
```

Apache Virtual Host Configuration

For HTTPS:

```
<VirtualHost *:443>
    ServerName yourdomain.com
    DocumentRoot /var/www/html/MotoParts

    SSLEngine on
    SSLCertificateFile /path/to/cert.pem
    SSLCertificateKeyFile /path/to/private.key

    <Directory "/var/www/html/MotoParts">
        AllowOverride All
        Require all granted
    </Directory>

    # Security headers
    Header always set Strict-Transport-Security "max-age=63072000;
includeSubDomains; preload"
    Header always set X-Content-Type-Options nosniff
    Header always set X-Frame-Options DENY
    Header always set X-XSS-Protection "1; mode=block"
</VirtualHost>
```

Email Configuration

```
// For email notifications (e.g., using PHPMailer)
define('SMTP_HOST', 'smtp.gmail.com');
define('SMTP_USER', 'your-email@gmail.com');
define('SMTP_PASS', 'your-app-password');
```

---

9. Security Features

1. Input Validation

- **All user inputs sanitized using** `filter_var()` **or** `mysqli_real_escape_string()`**.**
- **CSRF protection using tokens.**
- **SQL injection prevention via prepared statements.**
- **XSS protection through** `htmlspecialchars()`**.**

2. SQL Injection Prevention

```php
// Prepared statements
$stmt = $conn->prepare("SELECT * FROM users WHERE email = ?");
$stmt->bind_param("s", $email);
$stmt->execute();
```

3. XSS Protection

```php
// Output escaping
echo htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8');
```

4. CSRF Protection

```php
// Token generation
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));

// Token validation
if (!hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) {
    die('CSRF token mismatch');
}
```

5. Password Security

- **Minimum requirements: 8 characters, alphanumeric.**
- **Hashing algorithm:** `PASSWORD_DEFAULT` **(bcrypt).**
- **Salt automatically generated per password.**
- **No plain text password storage.**

```php
// Password hashing
$password_hash = password_hash($password, PASSWORD_BCRYPT);

// Password verification
if (password_verify($password, $stored_hash)) {
    // Login successful
}
```

6. Session Security

- **Session regeneration on login.**
- **Secure session cookies (`httpOnly`, `secure` flags).**
- **Session timeout (30 minutes idle).**
- **IP address validation.**

```
// Secure session configuration
ini_set('session.cookie_httponly', 1);
ini_set('session.use_strict_mode', 1);
ini_set('session.cookie_secure', 1); // HTTPS only
```

7. File Upload Security

```
// Validate file type
$file_info = finfo_open(FILEINFO_MIME_TYPE);
$mime_type = finfo_file($file_info, $_FILES['upload']['tmp_name']);

// Allowed MIME types
$allowed_mimes = [
    'image/jpeg',
    'image/png',
    'image/gif'
];
```

8. Database Security

- **Prepared statements for all database queries.**
- **Database user with minimal required privileges.**
- **Connection over SSL/TLS.**
- **Regular database backups with encryption.**

10. Troubleshooting

Common Issues

1. Database Connection Error

- Symptoms: "Database connection failed" errors
- Diagnostic Steps:
    - Check database server status: `systemctl status mysql`
    - **Verify connection parameters in** `config.php`
    - **Test connection manually:** `mysql -u username -p database_name`
    - **Check MySQL error log:** `tail -f /var/log/mysql/error.log`
- Solutions:
    - Restart MySQL service: `sudo systemctl restart mysql`
    - **Update connection credentials**
    - **Check firewall settings**
    - **Verify database user privileges**

2. PayRex Webhook Not Working

- Symptoms: Payments not updating order status
- Solution:
    1. Check webhook URL accessibility
    2. Verify webhook secret
    3. Check server logs

3. Image Upload Issues

- Symptoms: Images not displaying
- Solution:

```
# Check directory permissions
chmod 777 assets/images/ # Ensure write access
chown www-data:www-data assets/images/
```

- **Verify PHP upload settings (**`upload_max_filesize`**,** `post_max_size` **in**

  `php.ini`**).**
- **Check file size limits.**

4. Session Timeout

- Symptoms: Users logged out unexpectedly
- Solution:

```
// Increase session timeout
   ini_set('session.gc_maxlifetime', 3600); // 1 hour
   ini_set('session.cookie_lifetime', 3600);
```

  - **Verify session directory permissions.**
  - **Monitor session cleanup frequency.**

5. Performance Issues

- Symptoms: Slow page loading, timeouts
- Diagnostic Tools:
  1. MySQL slow query log
  2. Apache access logs
  3. PHP error logs
  4. System resource monitoring (htop, iostat)
- Optimization Strategies:
  1. Enable query caching in MySQL
  2. Implement Redis for session storage
  3. Enable Apache `mod_deflate` **for compression**
  4. **Optimize database indexes**

Debug Mode

Enable debug mode for development:

```php
// includes/config.php
if ($_SERVER['SERVER_NAME'] == 'localhost') {
    error_reporting(E_ALL);
    ini_set('display_errors', 1);
    define('DEBUG_MODE', true);
} else {
    define('DEBUG_MODE', false);
}
```

System Health Check Script

```php
<?php
// system_health.php - Basic system health checker

function checkDatabaseConnection() {
    try {
        $conn = new PDO("mysql:host=" . DB_HOST . ";dbname=" .
DB_NAME, DB_USER, DB_PASS);
        return "Database: OK";
    } catch (PDOException $e) {
        return "Database: ERROR - " . $e->getMessage();
    }
}

function checkWritePermissions() {
    $test_file = 'assets/images/test_write.txt';
    if (file_put_contents($test_file, 'test') !== false) {
        unlink($test_file);
        return "File Permissions: OK";
    }
    return "File Permissions: ERROR - Cannot write to
assets/images/";
}

function checkDiskSpace() {
    $free_bytes = disk_free_space('.');
    $free_gb = round($free_bytes / 1024 / 1024 / 1024, 2);
    return "Disk Space: {$free_gb} GB free";
}
```

```
// Run health checks
echo checkDatabaseConnection() . "\n";
echo checkWritePermissions() . "\n";
echo checkDiskSpace() . "\n";
?>
```

Error Code Reference

- AUTH_001: Invalid login credentials - Verify username/password
- AUTH_002: Account not found - Check if user exists in database
- CART_001: Product not available - Check product stock status
- ORDER_001: Payment processing failed - Contact payment provider
- UPLOAD_001: File too large - Reduce file size or increase limits
- DB_001: Database connection timeout - Check database server status

---

11. Development Guidelines

Code Standards

- PSR-12: PHP coding standards
- Semantic HTML: Proper document structure
- Responsive Design: Mobile-first approach
- Security First: Always validate and sanitize input

File Naming Convention

- Controllers: `adminpage.php`, `customerpage.php`
- API Endpoints: `create_payment_intent.php`
- Configuration: `config.php`, `db_connection.php`
- Assets: `adminstyle.css`, `script.js`

Git Workflow

```
# Feature branch
git checkout -b feature/new-feature
git add .
git commit -m "Add new feature"
git push origin feature/new-feature
```

Testing Framework

- Automated Testing Setup:

```
# PHPUnit configuration
composer require --dev phpunit/phpunit

# Run test suite
./vendor/bin/phpunit tests/
```

Testing Checklist

- Authenticator Tests (9 test cases):
    - Customer login validation
    - Retailer login validation
    - Admin login validation
    - Error handling for invalid credentials
- Search Facility Tests (6 test cases):
    - Alphanumeric search validation
    - Special character support
    - Multi-category filtering
    - Result display accuracy
- Order Processing Tests (4 test cases):
    - Product selection workflow
    - Quantity adjustment functionality
    - Payment method selection

    - Order confirmation process
- Product Management Tests (6 test cases):
    - Description field validation
    - Category selection
    - Order ID lookup
    - Quantity management
    - Price display
    - Image upload
- User Management Tests (4 test cases):
    - User creation workflow
    - Duplicate validation
    - User information updates
    - User deactivation
- Cart Management Tests (5 test cases):
    - Add to cart functionality
    - Payment method selection
    - Price formatting
    - Shipping date selection
    - Order ID generation
- Reporting Tests (5 test cases):
    - Report search functionality
    - Export format validation
    - Data archiving
    - System logs access
    - Table data accuracy

Test Coverage Results

- Total Test Cases: 41
- Pass Rate: 100%
- Fail Rate: 0%
- Pending Rate: 0%

---

12. Maintenance Procedures

Database Maintenance

Daily Tasks

- Monitor system logs for errors
- Check database connection health
- Verify backup completion

Weekly Tasks

- Clean up expired sessions
- Archive old order data
- Update system statistics

Monthly Tasks

- Optimize database tables
- Review user accounts for inactive users
- Update system documentation

Database Optimization

```sql
-- Optimize tables
OPTIMIZE TABLE users, products, orders, order_items;

-- Update table statistics
ANALYZE TABLE users, products, orders, order_items;

-- Clean up old sessions
DELETE FROM sessions WHERE last_activity < DATE_SUB(NOW(), INTERVAL
30 DAY);
```

Backup Procedures

```bash
#!/bin/bash
# Daily backup script
DATE=$(date +%Y%m%d_%H%M%S)
mysqldump -u backup_user -p motoparts_db >
/backups/motoparts_$DATE.sql
find /backups/ -name "motoparts_*.sql" -mtime +30 -delete
```

---

13. Appendices

A. Database Schema Diagram

```
erDiagram
    USERS ||--o{ ORDERS : places
    USERS ||--o{ PRODUCTS : sells
    USERS ||--o{ SHOPPING_CART : has
    USERS ||--o{ SYSTEM_LOGS : generates
    ORDERS ||--o{ ORDER_ITEMS : contains
    PRODUCTS ||--o{ ORDER_ITEMS : included_in
    PRODUCTS }--o{ CATEGORIES : belongs_to
```

Explanation of Relationships:

- USERS places ORDERS (One-to-Many): One user can place multiple orders.
- USERS sells PRODUCTS (One-to-Many): Retailers (a type of user) can sell multiple products.
- USERS has SHOPPING_CART (One-to-Many): A user can have multiple items in their shopping cart.
- USERS generates SYSTEM_LOGS (One-to-Many): User actions are recorded in system logs.
- ORDERS contains ORDER_ITEMS (One-to-Many): An order is composed of one or more order items.
- PRODUCTS included_in ORDER_ITEMS (One-to-Many): A product can be part of many order items.
- PRODUCTS belongs_to CATEGORIES (Many-to-Many): Products can belong to multiple categories, and categories can contain multiple products (this implies a junction table, which is a common way to implement M-N relationships in relational databases).

B. File Permission Requirements

This section outlines the recommended file permissions for the MotoParts Manager system to ensure proper functionality and security.

```
MotoParts/
├── (root directory)   755 (drwxr-xr-x)
│   ├── *.php          644 (-rw-r--r--)  # Standard PHP file permissions
│   ├── assets/        755 (drwxr-xr-x)  # Directory for static assets
│   │   ├── css/       755 (drwxr-xr-x)  # Stylesheets directory
│   │   ├── js/        755 (drwxr-xr-x)  # JavaScript files directory
│   │   └── images/    777 (drwxrwxrwx)  # Product images directory -
requires write access for uploads
│   └── includes/      755 (drwxr-xr-x)  # Directory for configuration and
common functions
│       └── config.php 600 (-rw-------)  # Database configuration -
restricted access for security
```

Key Permissions:

- 755 (drwxr-xr-x): Directories and executable files. Owner has read, write, and execute permissions. Group and others have read and execute permissions.
- 644 (-rw-r--r--): Non-executable files (like PHP scripts). Owner has read and write permissions. Group and others have read-only permissions.
- 777 (drwxrwxrwx): Directories requiring write access by the web server (e.g., for image uploads). This grants read, write, and execute permissions to all users. Note: While necessary for uploads, this is a less secure permission. Ensure only trusted processes can write to this directory.

C. Environment Variables

Environment variables are used to store sensitive configuration data and application settings, especially for production environments. This helps keep credentials out of the codebase and allows for easy configuration changes across different deployment stages.

```
# Database Connection Settings
export MOTOPARTS_DB_HOST="localhost"            # Database host (e.g.,
localhost, IP address)
export MOTOPARTS_DB_NAME="motoparts_prod"        # Production database name
export MOTOPARTS_DB_USER="motoparts_user"        # Database username
export MOTOPARTS_DB_PASS="secure_production_password" # Database password

# Application Environment Settings
export MOTOPARTS_ENV="production"                # Application environment
(e.g., production, development, staging)
export MOTOPARTS_DEBUG="false"                   # Debug mode flag
(true/false) - set to false in production

# PayRex API Credentials
export PAYREX_SECRET_KEY="sk_live_your_secret_key" # PayRex secret key for
live transactions
export PAYREX_PUBLISHABLE_KEY="pk_live_your_publishable_key" # PayRex
publishable key for client-side integration
export PAYREX_WEBHOOK_SECRET="whsec_your_webhook_secret" # Webhook secret
for verifying PayRex webhook events
```

Usage: **These variables should be loaded into the application's environment at runtime. For PHP applications, this can be done using libraries like `phpdotenv` or by configuring the web server (e.g., Apache's `SetEnv` directive or Nginx's `fastcgi_param` with `include /etc/environment`).**

---

14. Frequently Asked Questions (FAQ)

General Questions

- Q: What is MotoParts Manager?
    - A: MotoParts Manager is a comprehensive e-commerce platform designed for managing motorcycle parts retail, offering multi-role access for administrators, retailers, and customers, along with integrated payment processing, inventory management, and business analytics.
- Q: Who is the target audience for this technical manual?
    - A: This manual is intended for developers, system administrators, and IT personnel who will be working with the MotoParts Manager system.

- Q: What technologies are used in MotoParts Manager?
  - A: The system uses PHP for the backend, HTML5, CSS3, and JavaScript for the frontend, MySQL for the database, and Apache/Nginx as the web server. It also integrates with PayRex for payment processing.

Installation & Deployment

- Q: What are the minimum system requirements for installation?
  - A: You need PHP 8.0+, MySQL 8.0+ or MariaDB 10.5+, Apache 2.4+ or Nginx 1.18+, at least 512MB RAM, and 1GB storage. Composer is also required for dependency management.
- Q: How do I set up the database?
  - A: You need to create a MySQL database, a dedicated user, grant necessary privileges, and then import the `schema.sql` file. Detailed SQL commands are provided in the Installation Guide.
- Q: What file permissions are necessary after deployment?
  - A: Most files should be 644, directories 755. The `assets/images/` directory specifically needs 777 permissions to allow image uploads by the web server. The `includes/config.php` file should be 600 for restricted access.
- Q: How do I configure SSL for the application?
  - A: The manual suggests using Certbot with Apache to obtain and configure a Let's Encrypt SSL certificate.

Security

- Q: How does MotoParts Manager prevent SQL Injection?
  - A: The system uses prepared statements for all database queries to prevent SQL injection attacks.
- Q: What measures are in place for password security?
  - A: Passwords are hashed using `PASSWORD_BCRYPT` (bcrypt) with automatically generated salts. They are never stored in plain text, and minimum requirements (8 characters, alphanumeric) are enforced.
- Q: How is Cross-Site Scripting (XSS) prevented?

    - A: All user-supplied output is sanitized using `htmlspecialchars()` to prevent XSS attacks.
- Q: Are there any protections against CSRF attacks?
    - A: Yes, the system implements CSRF protection using tokens that are generated and validated for sensitive operations.

Troubleshooting

- Q: I'm getting a "Database connection failed" error. What should I do?
    - A: Check if your MySQL service is running, verify the database credentials in `config.php`, test the connection manually, and check the MySQL error logs. Solutions include restarting MySQL or updating credentials.
- Q: My product images are not uploading or displaying. What's wrong?
    - A: This is often a file permission issue. Ensure the `assets/images/` directory has 777 permissions and is owned by the web server user (`www-data`). Also, check PHP's `upload_max_filesize` and `post_max_size` settings in `php.ini`.
- Q: Users are getting logged out unexpectedly. How can I fix this?
    - A: This indicates a session management issue. You might need to increase the `session.gc_maxlifetime` and `session.cookie_lifetime` in `php.ini`. Also, verify session directory permissions and monitor session cleanup frequency.
- Q: How can I enable debug mode for development?
    - A: In `includes/config.php`, set `DEBUG_MODE` to `true` and configure `error_reporting(E_ALL)` and `ini_set('display_errors', 1)` for your local environment. Remember to disable this in production.

Development & Maintenance

- Q: What are the coding standards for this project?
    - A: The project adheres to PSR-12 for PHP coding standards, uses Semantic HTML, and follows a mobile-first approach for responsive design. Security is a primary concern in all development.
- Q: How often are database backups performed?

  - A: Automated database backups are performed weekly.
- Q: What is the process for monitoring system performance?
  - A: Performance monitoring involves checking MySQL slow query logs, Apache access logs, PHP error logs, and using system resource monitoring tools like `htop` or `iostat`. Optimization strategies include query caching, Redis for sessions, and database indexing.

---

15. GitHub Repository

The complete source code for the MotoParts Manager system is hosted on GitHub. This repository serves as the central hub for development, version control, issue tracking, and collaboration.

Repository Access

- GitHub URL: https://github.com/zmpld/motoparts

Repository Contents

The GitHub repository contains all the necessary files and directories for the MotoParts Manager application, mirroring the directory structure outlined in Section 2. System Architecture. Key contents include:

- `MotoParts/`: The main application directory.
- `database/`: Contains the `schema.sql` for database setup.
- `includes/`: Core configuration and utility functions.
- `modules/`: Modular components for various functionalities (search, cart, orders, etc.).
- `assets/`: Static files like CSS, JavaScript, and images.
- `README.md`: General information about the project, quick setup instructions, and contribution guidelines.

Contributing to the Project

Developers interested in contributing to the MotoParts Manager project can do so by following the standard GitHub workflow:

- Fork the Repository: Create a personal copy of the repository.
- Clone Your Fork: Download your forked repository to your local machine.

```
git clone https://github.com/YOUR_USERNAME/motoparts.git
cd motoparts
```

- Create a New Branch: For each new feature or bug fix, create a dedicated branch.

```
git checkout -b feature/your-new-feature
```

- Make Changes: Implement your changes, adhering to the Code Standards (Section 11).
- Commit Changes: Commit your changes with clear and concise messages.

```
git add .
git commit -m "feat: Add new feature for X"
```

- Push to Your Fork: Upload your changes to your GitHub fork.

```
git push origin feature/your-new-feature
```

- Open a Pull Request (PR): Submit a pull request from your branch to the `main` branch of the original repository. Provide a detailed description of your changes.

Issue Tracking

The GitHub repository's "Issues" section is used for:

- Bug Reports: Reporting any defects or unexpected behavior.
- Feature Requests: Suggesting new functionalities or enhancements.
- Discussions: Engaging in conversations about project direction, technical challenges, or potential improvements.

Users and contributors are encouraged to utilize the Issues section for all project-related discussions and problem reporting.

Document Information

- Version: 1.4
- Last Updated: July 20, 2025
- Document Type: Technical Manual
- Classification: Internal Use
- Review Cycle: Quarterly

This technical manual provides comprehensive information for developers and system administrators working with the MotoParts Manager system. For user-facing documentation, refer to the User Manual.

Support & Maintenance

Regular Tasks

1. Database Backups: Weekly automated backups
2. Security Updates: Monthly dependency updates
3. Performance Monitoring: Query optimization
4. Log Analysis: Error tracking and resolution

Contact Information

- Developer Team: MT MotoParts Development
- Email: mtmotoparts@gmail.com
- Phone: (+63) 9298642708
- Business Hours: 9:00 AM - 6:00 PM (Mon-Sat)

Version History

- v1.0.0: Initial release with core features
- v1.1.0: Added PayRex integration
- v1.2.0: Enhanced analytics dashboard
- v1.3.0: Mobile responsive design
- v1.4.0: Updated technical manual with detailed sections and improved clarity (July 2025)

This technical manual is maintained by the MotoParts development team. Last updated: July 2025