Car Parking System Login Manual

Table of Contents

Overview

The Car Parking System login process is a PHP-based authentication system that handles three distinct user roles: **admin**, **owner**, and **user**. The system provides secure login functionality with role-based redirection, comprehensive error handling, and a modern Bootstrap-styled interface.

Key Features:

- Multi-role authentication (admin, owner, user)
- Session-based authentication
- Password hashing with PHP's `password_hash()` and `password_verify()`
- Role-based dashboard redirection
- Comprehensive error handling
- Modern, responsive UI with Bootstrap 5.3.0
- SQL injection prevention using prepared statements

Session Management and User Redirection

Initial Session Check

The login system begins by checking if a user is already authenticated:

```php
session_start();
require_once __DIR__ . '/db.php';

// Redirect if already logged in
if (isset($_SESSION['user_id'])) {
    switch ($_SESSION['role']) {
        case 'admin':
            header('Location: admin_dashboard.php');
            exit;
        case 'owner':
            header('Location: owner_dashboard.php');
            exit;
        default:
            header('Location: user_dashboard.php');
            exit;
    }
}
```

How It Works:

1. **Session Initialization**: `session_start()` initializes or resumes the current session
2. **Authentication Check**: The system checks for `$_SESSION['user_id']` to determine if user is logged in
3. **Role-Based Redirection**: Based on `$_SESSION['role']`, users are redirected to their appropriate dashboard:
   - **Admin**: `admin_dashboard.php` - Full system management
   - **Owner**: `owner_dashboard.php` - Parking space management
   - **User**: `user_dashboard.php` - Space reservation and viewing

Post-Login Session Setup

After successful authentication, the system establishes the user session:

```php
$_SESSION['user_id'] = $user['id'];
$_SESSION['username'] = $user['username'];
$_SESSION['role'] = $user['role'];
```

This creates a persistent session that allows the user to access role-specific features throughout their session.

Form Handling and Validation

Form Structure

The login form accepts three inputs:

- **Username**: Text field (required)
- **Password**: Password field (required)
- **Role**: Optional dropdown for role-specific login

Server-Side Processing

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = trim($_POST['username'] ?? '');
    $password = $_POST['password'] ?? '';
    $role = $_POST['role'] ?? '';

    // Validation logic follows...
}
```

Validation Steps:

1. **Input Sanitization**:
   - `trim()` removes whitespace from username
   - Null coalescing operator (`??`) provides default empty strings
2. **Required Field Check**:
   - Username and password are required
   - Role is optional but validated if provided
3. **Role Matching**:
   - If a role is specified, it must match the user's database role
   - Prevents users from accessing unauthorized dashboards

Form State Persistence

The system maintains form state during validation errors:

```php
```

```
$selected_role = $_POST['role'] ?? '';
```

This ensures the role dropdown retains its selection if login fails.

Database Interaction

Database Connection

The system uses PDO (PHP Data Objects) for secure database interaction:

```php
// From db.php
$host = 'localhost';
$db   = 'car_parking';
$user = 'root';
$pass = '';
$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,
    PDO::ATTR_EMULATE_PREPARES   => false,
];
```

User Authentication Query

```php
try {
    $stmt = $pdo->prepare("SELECT * FROM users WHERE username = ?");
    $stmt->execute([$username]);
    $user = $stmt->fetch();

    if ($user && password_verify($password, $user['password'])) {
        // Authentication successful
    }
} catch (PDOException $e) {
    $message = 'Database error: ' . $e->getMessage();
}
```

Security Features:

1. **Prepared Statements**: Prevents SQL injection attacks
2. **Password Verification**: Uses `password_verify()` for secure password checking
3. **Exception Handling**: Catches and handles database errors gracefully
4. **No Password Storage**: Raw passwords are never stored or logged

Database Schema Requirements

The system expects a `users` table with:

- `id` (Primary Key)
- `username` (Unique)
- `password` (Hashed)
- `role` (ENUM: 'admin', 'owner', 'user')
- `email`, `full_name`, `phone` (Additional fields)

Error Handling and User Feedback

Error Types and Messages

The system provides specific error messages for different failure scenarios:

```php
if ($user && password_verify($password, $user['password'])) {
    if ($role && $user['role'] !== $role) {
        $message = 'Role does not match account.';
    } else {
        // Success - proceed with login
    }
} else {
    $message = 'Invalid credentials.';
}
```

Error Categories:

1. **Invalid Credentials**: Username/password combination doesn't exist
2. **Role Mismatch**: User exists but selected role doesn't match account
3. **Database Errors**: Connection or query failures

4. **Session Errors**: Issues with session management

User Feedback Display

```php
<?php if ($message): ?>
    <div class="alert alert-danger">❌ <?= $message ?></div>
<?php endif; ?>
```

Success Feedback

The system shows success messages for successful registration:

```php
<?php if (isset($_GET['registered'])): ?>
    <div class="alert alert-success">✅ Registration successful! Please login.</div>
<?php endif; ?>
```

HTML Structure and Styling

Bootstrap Integration

The login form uses Bootstrap 5.3.0 for responsive design:

```html
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
```

Visual Design Elements

1. **Gradient Background**:

```css
body {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
}
```

2. **Card Layout**: Centered card with rounded corners and shadow
3. **Modern Form Controls**: Rounded input fields with smooth transitions
4. **Hover Effects**: Button animations and form field focus states

Form Structure

```html
<form method="post">
  <div class="mb-3">
    <input type="text" name="username" class="form-control"
        placeholder="👤 Username" required>
  </div>
  <div class="mb-3">
    <input type="password" name="password" class="form-control"
        placeholder="🔒 Password" required>
  </div>
  <button type="submit" class="btn btn-primary w-100 mb-3">Login</button>
</form>
```

Responsive Design

The system uses Bootstrap's grid system and responsive utilities:

- Mobile-first approach
- Centered card layout
- Responsive typography
- Touch-friendly button sizes

File Connections and Dependencies

Core Dependencies

1. **db.php**: Database connection configuration

   ```php
   require_once __DIR__ . '/db.php';
   ```

2. **Dashboard Files**: Role-specific landing pages
   - `admin_dashboard.php`
   - `owner_dashboard.php`
   - `user_dashboard.php`
3. **register.php**: User registration system
   - Linked from login form

- Redirects back to login after registration

File Interaction Flow

```
login.php
├── db.php (database connection)
├── admin_dashboard.php (admin redirect)
├── owner_dashboard.php (owner redirect)
├── user_dashboard.php (user redirect)
└── register.php (registration link)
```

Dashboard Features by Role

Admin Dashboard (`admin_dashboard.php`)

- View system statistics
- Total spaces, occupied, reserved counts
- User management
- Reservation management
- System reports

Owner Dashboard (`owner_dashboard.php`)

- Manage owned parking spaces
- Add/edit/delete spaces
- View reservations for owned spaces
- Revenue tracking

User Dashboard (`user_dashboard.php`)

- View available spaces
- Make reservations
- View personal reservation history
- Space status monitoring

Security Considerations

Password Security

1. **Password Hashing**: Uses PHP's `password_hash()` with default algorithm

```php
// Registration (register.php)
$password = password_hash($_POST['password'], PASSWORD_DEFAULT);

// Login verification
if (password_verify($password, $user['password'])) {
    // Authentication successful
}
```

2. **No Plain Text Storage**: Passwords are never stored in plain text
3. **Secure Verification**: `password_verify()` is timing-attack resistant

SQL Injection Prevention

1. **Prepared Statements**: All database queries use parameterized queries

```php
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ?");
$stmt->execute([$username]);
```

2. **Input Sanitization**: User input is sanitized before processing

```php
$username = trim($_POST['username'] ?? '');
```

Session Security

1. **Session Management**: Proper session initialization and cleanup
2. **Role-Based Access Control**: Strict role checking for dashboard access
3. **Session Invalidation**: Logout functionality clears session data

Additional Security Measures

1. **Error Handling**: Generic error messages prevent information leakage
2. **HTTPS Ready**: Code structure supports SSL/TLS encryption
3. **XSS Prevention**: Output escaping with `htmlspecialchars()`

Extensibility

Adding New User Roles

To add a new role (e.g., "manager"):

1. **Update Database Schema**:

   sql

   ```
   ALTER TABLE users MODIFY COLUMN role ENUM('admin','owner','user','manager') DEFAULT 'user';
   ```

2. **Update Login Redirection**:

   php

   ```
   switch ($_SESSION['role']) {
       case 'admin':
           header('Location: admin_dashboard.php');
           break;
       case 'owner':
           header('Location: owner_dashboard.php');
           break;
       case 'manager':
           header('Location: manager_dashboard.php');
           break;
       default:
           header('Location: user_dashboard.php');
           break;
   }
   ```

3. **Create Dashboard**: Develop `manager_dashboard.php` with appropriate features

Enhanced Authentication Features

1. **Multi-Factor Authentication**:
   - Add phone/email verification
   - Implement TOTP (Time-based One-Time Password)
2. **Account Security**:
   - Password complexity requirements
   - Account lockout after failed attempts
   - Session timeout management
3. **OAuth Integration**:
   - Google/Facebook login
   - SAML integration for enterprise

Additional Validation

1. **Input Validation**:

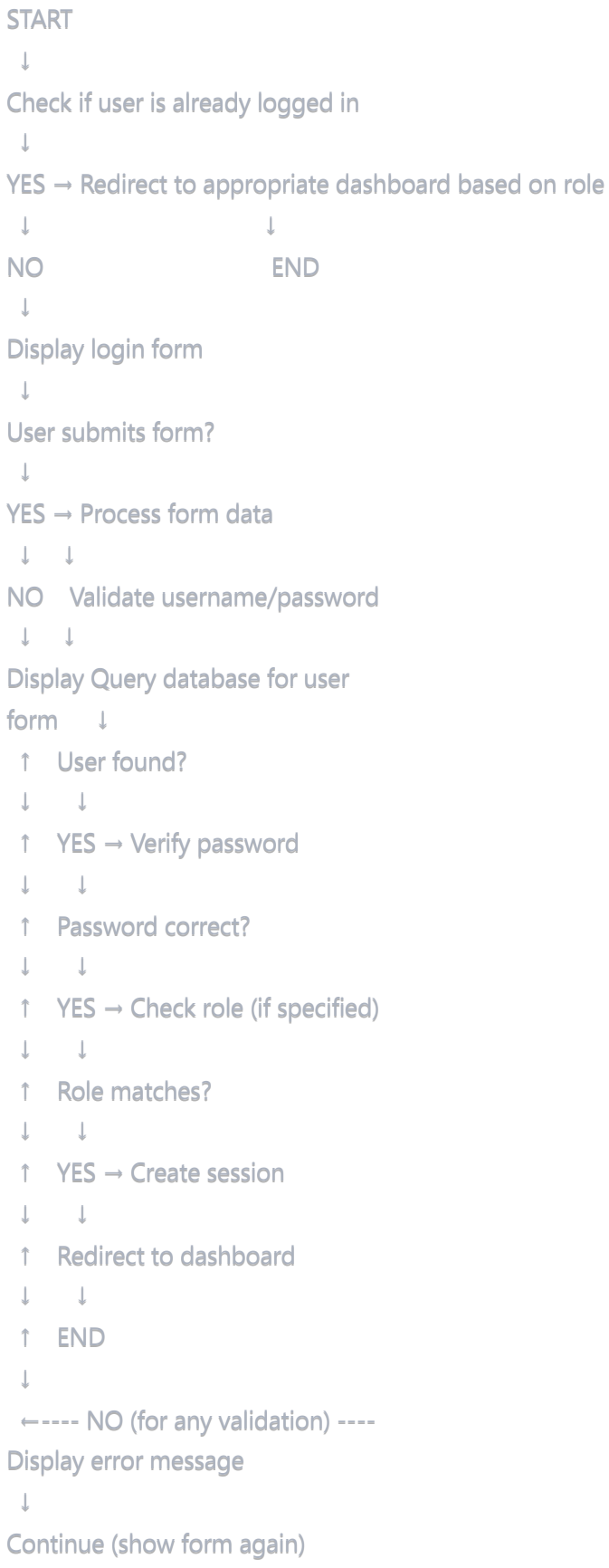
```php
// Email validation
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $message = 'Invalid email format.';
}

// Password strength
if (strlen($password) < 8) {
    $message = 'Password must be at least 8 characters.';
}
```

2. **Rate Limiting**:

```php
// Implement login attempt tracking
if ($failed_attempts >= 5) {
    $message = 'Account temporarily locked.';
}
```

Code Flow Diagram


START
Check if user is already logged in
YES → Redirect to appropriate dashboard based on role
END
NO
Display login form
User submits form?
YES → Process form data
NO
Validate username/password
Display form
Query database for user
User found?
YES → Verify password
Password correct?
YES → Check role (if specified)
Role matches?
YES → Create session
Redirect to dashboard
END
<---- NO (for any validation) ----
Display error message
Continue (show form again)

Conclusion

The Car Parking System login process provides a robust, secure, and extensible authentication system. It successfully handles multiple user roles, implements strong security practices, and provides an excellent user experience through modern UI design and comprehensive error handling.

The modular design allows for easy maintenance and future enhancements, while the security measures protect against common web vulnerabilities. The system serves as a solid foundation for a multi-tenant parking management application.