

# Anesoft Anesthesia Simulator 7

## Complete System Documentation

---

### Executive Summary

This is a technical reference manual for the Anesthesia Simulator 7 platform as deployed and operational in January 2026. It provides specifications for 40+ API endpoints, schema definitions for 35+ database tables, session management implementation, file system organization, and operational procedures currently in production.

This serves two primary purposes: preserving system knowledge for continuity if development transitions to new developers, and providing technical specifications for maintenance, troubleshooting, or feature additions. All API endpoints are documented with actual request/response formats, error codes, and database operations as implemented in the current codebase. Database tables include field definitions, relationships, and actual query patterns used throughout the system.

The sections included will cover API endpoint specifications with complete request/response documentation, database schema with table relationships and key fields, session management lifecycle and timeout configuration, frontend JavaScript integration patterns, file system structure with backup identification, logging systems and debug procedures, and maintenance protocols with safety procedures.

Technical specs reflect the live production system. API documentation shows actual endpoint behavior, not planned features. Database schema represents current table structures and relationships. File organization describes what exists on the server, including backup files for investigation or cleanup. All specifications match the deployed system as of January 2026.

---

#### Technology Stack:

- ❖ Backend: PHP with PDO database layer
- ❖ Frontend: JavaScript with AJAX integration
- ❖ Database: MySQL with 35+ tables
- ❖ Session Management: PHP sessions with database-backed configuration
- ❖ Authentication: Session-based with MD5 password hashing
- ❖ Configuration: File-based JSON and database-stored settings
- ❖ Error Handling: Categorized error types with user-friendly messages
- ❖ Logging: Weekly rotating log files with debug mode control

### **System Capabilities:**

- ❖ 80+ case-based training scenarios
- ❖ Multi-tier user management (Demo, Standard, Pro, Admin, Super Admin)
- ❖ Group/institutional licensing with Pro feature upgrades
- ❖ Session management with configurable timeouts
- ❖ Comprehensive logging and debugging systems
- ❖ RESTful API architecture with 40+ endpoints

### **Case Management:**

- ❖ Case saving with grade recording and instructor feedback
- ❖ User case history with visibility controls
- ❖ Case grading system (High Pass, Pass, Fail, Incomplete)
- ❖ Demo user restrictions with upgrade pathways

### **User & Group Administration:**

- ❖ Group administration with member management
- ❖ Password synchronization between users and groups
- ❖ User profile management with name and email updates
- ❖ Role transitions with automatic group and password reassignment

### **Pro Features & Licensing:**

- ❖ Pro status management with expiration tracking
- ❖ Group-level and user-level Pro access
- ❖ Pro status history tracking with audit trail and reversion capability

### **Session & Authentication:**

- ❖ Real-time session validation with automatic re-authentication
- ❖ Session preference management per user
- ❖ Database-backed timeout configuration

### **System Operations:**

- ❖ Database backup and restore functionality
- ❖ Debug mode toggle for system-wide logging control

## Table of Contents

### SECTION 1: System Architecture

- 1.1 Application Structure Overview
- 1.2 Directory Organization
- 1.3 File Dependencies
- 1.4 Configuration Management
- 1.5 Complete Core Directory Reference
- 1.6 Complete API Directory Reference
- 1.7 Complete Includes Directory Reference
- 1.8 Complete Configuration Files Reference

### SECTION 2: API Reference

- 2.1 Case Data Management APIs
- 2.2 Session Management APIs
- 2.3 Pro Features & Licensing APIs
- 2.4 Case Management Operations
- 2.5 User Management APIs
- 2.6 Administrative Functions
- 2.7 Session Preference APIs
- 2.8 Pro Status History APIs
- 2.9 User Settings APIs
- 2.10 Utility & Helper APIs

### SECTION 3: Database Architecture

- 3.1 Core Tables Reference
- 3.2 Table Relationships
- 3.3 Key Fields Documentation
- 3.4 Query Patterns

## **SECTION 4: Session Management**

- 4.1 Configuration & Storage
- 4.2 Session Lifecycle
- 4.3 Timeout Calculations
- 4.4 Validation & Expiration

## **SECTION 5: Integration Guide**

- 5.1 Frontend JavaScript Patterns
- 5.2 PHP Include Standards
- 5.3 API Request/Response Formats
- 5.4 Error Handling Patterns

## **SECTION 6: Logging & Debugging**

- 6.1 Debug Mode Configuration
- 6.2 Case Save Logging
- 6.3 Session Failure Logging
- 6.4 Log File Locations

## **SECTION 7: Development Standards & Patterns**

- 7.1 Password Hashing Method
- 7.2 Session Variable Standards
- 7.3 Authentication Levels
- 7.4 Database Connection Pattern
- 7.5 Version Control Patterns
- 7.6 App Domain Structure

## **SECTION 8: Backup & Deployment Ops**

- 8.1 Backup File Locations
- 8.2 Version Control Patterns
- 8.3 Push to Live Mechanism
- 8.4 Push to Live Procedure

## SECTION 1: System Architecture

### 1.1 Application Structure Overview

The Anesthesia Simulator 7 platform follows a modular architecture with clear separation between API endpoints, administrative functions, application resources, and shared includes.

**Core Components:**

**API Layer** (/api/) 40+ RESTful endpoints handling case operations, user management, session control, and administrative functions.

**Administrative Interface** (/admin/) Full-featured admin panel for user management, group administration, case review, and system configuration.

**Application Resources** (/app/) Case scenarios, JavaScript libraries, stylesheets, images, and media files for the simulator interface.

**Shared Includes** (/includes/) Core functionality libraries including authentication, session management, logging, and database operations.

**Configuration Files** (/configs/) Database connection parameters, session settings, debug configuration, and system preferences.

### 1.2 Directory Organization

— api/	API endpoints (40+ files)
— save_case_data.php	Case saving with error handling
— get_case_data.php	Case list retrieval
— check_session.php	Session validation
— keepalive.php	Session refresh
— pro_features.php	Pro access logic
— case_management.php	Case selection handling
— ...	Additional endpoints
— admin/	Administrative interface
— users.php	User management
— groups.php	Group administration
— cases.php	Case review
— settings.php	System configuration
— ...	Additional admin pages
— app/	Application resources
— cases_updated/	Required legacy case files
— case_data/	JSON case files
— js/	Application JavaScript
— styles/	Application CSS
— images/	Application images
— hemowaveforms/	Hemodynamic waveform data
— respwaveforms/	Respiratory waveform data

└─ sounds/	Audio files
└─ classes/	PHP class definitions
└─ group/	Group management classes
└─ users/	User management classes
└─ utility/	Helper classes
└─ configs/	Configuration files
└─ db.config.php	Database credentials
└─ debug_mode.json	Debug toggle state
└─ session_settings.json	Session timeout config
└─ general.config.php	System constants
└─ includes/	Shared PHP includes
└─ include_all.inc.php	Master bootstrap file
└─ session_manager.php	Session lifecycle
└─ save_logging.php	Case save logging
└─ debug_mode.php	Debug functions
└─ ...	Additional includes
└─ js/	Global JavaScript
└─ scripts.js	Core functionality
└─ scripts_app.js	Application logic
└─ scripts_pages.js	Page-specific code
└─ settings.js	Settings interface
└─ logs/	Application logs
└─ save_logs/	Case save logs (weekly)
└─ advanced_logs/	Debug logs
└─ user_actions.log	User activity log
└─ css/	Global stylesheets
└─ style.css	Main stylesheet
└─ dev.css	Development styles
└─ loader.css	Loading animations

## 1.3 File Dependencies

### Critical System Files

**includes/include\_all.inc.php** Master bootstrap file loaded by every page. Initializes database connection, starts session, loads core functions, and sets up error handling.

**Dependencies:** configs/db.config.php, includes/session\_manager.php

**Impact:** Breaking this file breaks entire application.

**includes/session\_manager.php** Manages complete session lifecycle including initialization, validation, expiration, and license checking.

**Dependencies:** Database connection (via include\_all.inc.php)

**Impact:** Modifications affect all authenticated pages and session behavior.

---

**configs/db.config.php** Contains database connection parameters.

**Constants Defined:**

- DB\_HOST - Database server hostname
- DB\_NAME - Database name
- DB\_USER - Database username
- DB\_PASS - Database password
- PREFIX - Table prefix (wp\_)

**Impact:** Incorrect values prevent all database operations.

---

**header.php** Contains core logic for app

**Dependencies:**

- include\_all.inc.php – DB connection, etc
- includes/profile\_functions.php – for profile picture
- menu.php – for menus
- *front\_user\_id* variable – required for sessions

**Impact:** Improper modifications affect entire app and may break app.

---

**api/pro\_features.php** Contains Pro feature access logic used throughout the system.

**Key Functions:**

- isProFeatureEnabled() - Checks user/group Pro status
- upgradeToProVersion() - Upgrades user/group to Pro

**Dependencies:** Database connection, session data

**Impact:** Controls access to paid features and additional cases.

---

## 1.4 Configuration Management

**configs/debug\_mode.json**

```
{
  "save_logging_enabled": true
}
```

Toggles debug logging throughout system via toggle\_debug\_mode.php API endpoint.

#### **configs/session\_settings.json**

- Stores session timeout configuration (deprecated - now stored in database wp\_settings table).

---

## 1.5 Complete Core Directory Reference

### **Complete Core File Listing:**

```
/
— 404.php           - Custom 404 error page with styled template
— 500.shtml         - Server error page using SSI includes
— cases.php         - Case settings display page
— config.php        - Site configuration constants
— custom-admin.css  - Custom admin styling
— debug_log.txt     - Debug output log
— favicon.ico       - Site favicon
— footer.php        - Main site footer
— footer_basic.php  - Simplified footer template
— functions.php     - Core site functions
— header.php        - Main site header
— header_basic.php  - Simplified header template
— index.php         - Main site homepage
— login.php         - User login page
— login_footer.php  - Login page footer
— login_header.php  - Login page header
— logout.php        - Logout handler
— menu.php          - Main navigation menu
— php.ini           - PHP configuration
— pro.php           - Pro features page
— profile.php       - User profile settings page
— register.php      - User registration page
— save-groups.php   - Group data saving
— save-users.php    - User data saving
— save.php          - Data saving handler
— settings.php      - User settings page
— wp-interface.php  - WordPress authentication compatibility layer
```



## 1.6 Complete API Directory Reference

### Complete API Endpoint Listing:

/api/	
— case_data.php	- Case data retrieval endpoint
— case_data_handler.php	- Case data processing functions
— case_management.php	- Case management operations
— check_session.php	- Session validation endpoint
— create_backup.php	- File system backup using PharData (tar.gz)
— create_backup_zip.php	- File system backup using ZipArchive
— create_db_backup.php	- Database backup with progress tracking
— debug_session_time.php	- Session timing diagnostics
— demo_role.php	- Demo user role detection
— get-group-details.php	- Group information retrieval
— get-pro-history.php	- Pro status history retrieval
— get-user-details.php	- User information retrieval
— get_case_data.php	- Individual case data retrieval
— get_case_details.php	- Detailed case information
— get_pro_group.php	- Pro group status
— get_pro_status.php	- User Pro status check
— get_user_cases.php	- User's available cases list
— index.php	- API directory protection
— keepalive.php	- Session keepalive endpoint (refreshes session)
— kill_session.php	- Force session termination
— log_session_failure.php	- Client-side session failure logging
— pro_features.php	- Pro feature management
— push_live.php	- Push development to production
— reauth.php	- Re-authentication endpoint (validates credentials)
— revert-pro-status.php	- Revert Pro status changes
— save_case_data.php	- Save case completion data
— save_current_case.php	- Save current case progress
— save_global_session_setting.php	- Global session setting updates
— save_session_settings.php	- User session preferences
— set_session_preference.php	- Individual session preference
— set_session_user_flag.php	- User flag updates
— sync_passwords.php	- Password synchronization
— sync_passwords_cron.php	- Scheduled password sync
— terminate_session.php	- Session termination endpoint
— terminate_session_php.php	- PHP session termination handler
— test_save_failure.php	- Save failure testing tool
— toggle_case_visibility.php	- Case visibility control
— toggle_debug_mode.php	- Debug mode on/off
— toggle_global_session_termination.php	- Global session termination toggle
— update-group-status.php	- Group status updates
— update-pro-status.php	- Pro status updates
— update-user-pro-status.php	- Individual user Pro status
— update-user-role.php	- User role changes
— update-user-settings.php	- User settings updates
— view_case.php	- Case viewing endpoint
— view_log.php	- Log file viewing

---

```
└─ logs/ - API-specific logs directory
```

---

## 1.7 Complete Includes Directory Reference

### Complete Includes Listing:

```
/includes/
└─ admin_functions.php - Administrative utility functions
└─ admin_settings.php - Admin configuration
└─ cases_settings.php - Case display settings
└─ contact_admin.php - Admin contact functionality
└─ debug_mode.php - Debug mode controls
└─ delete_profile_picture.php - Profile picture deletion
└─ id_gen.php - Unique ID generation (generateUniqueld, ensureUniqueld)
└─ include_all.inc.php - Master bootstrap file (loads all core includes)
└─ index.php - Directory protection
└─ profile_functions.php - Profile management functions
└─ profile_picture.php - Profile picture display
└─ profile_settings.php - Profile settings page
└─ remove_profile_picture.php - Profile picture removal
└─ save_logging.php - Case save logging
└─ save_message.php - Save operation messages
└─ session.php - Session initialization
└─ session_failure_logging.php - Session failure logging
└─ session_manager.php - Session lifecycle management
└─ session_settings.php - Session configuration
└─ super_admin.php - Super admin privileges
└─ upload_profile_picture.php - Profile picture upload
└─ user_settings.php - User settings functionality
└─ user_settings_functions.php - User settings utilities
```

---

## 1.8 Complete Configuration Files Reference

### Complete Configs Directory:

```
/configs/
└─ db.config.php - Database credentials
└─ debug_mode.json - Debug toggle state
└─ general.config.php - System constants
└─ index.php - For security
└─ mysql.config.php - MySQL settings
└─ session_settings.json - Session config (deprecated)
└─ site.webmanifest - PWA manifest
```

---

## SECTION 2: API Reference

### 2.1 Case Data Management APIs

save\_case\_data.php

**Method:** POST

**Authentication:** Required (front\_user\_id session)

**Content-Type:** application/json

**Request Body:**

```
{
  "caseLog": "Complete case log text",
  "grade": "High Pass|Pass|Fail|Incomplete",
  "instructor_notes": "Optional instructor comments",
  "clientCaseScore": "Score in X/Y format",
  "clientCaseTime": "Case duration",
  "clientCaseName": "Case identifier",
  "clientUserId": "User ID override",
  "clientUserFirstName": "First name",
  "clientUserLastName": "Last name",
  "clientUserEmail": "Email address"
}
```

**Response Format:**

```
{
  "success": true,
  "message": "Case log saved successfully.",
  "insertId": 12345
}
```

**Error Response:**

```
{
  "success": false,
  "errorType": "ERROR_CODE",
  "message": "User-friendly message",
  "technicalDetails": "Technical information"
}
```

**Error Codes:**

- SESSION\_EXPIRED - User session has timed out
- VALIDATION\_ERROR - Invalid or missing data
- DATABASE\_ERROR - Database operation failed
- CONNECTION\_ERROR - Database connection issue
- TIMEOUT\_ERROR - Query execution timeout
- DUPLICATE\_ERROR - Duplicate entry detected
- GENERAL\_ERROR - Unexpected error occurred

**Processing Logic:**

1. Validates JSON input and session
2. Extracts score from caseLog using regex: `/-?\d+\s*\s*\d+\/`
3. Retrieves username and email from wp\_users table
4. Logs attempt via save\_logging.php
5. Inserts record into wp\_groupusers\_data table
6. Returns inserted record ID on success

**Database Table:** wp\_groupusers\_data

**Fields Inserted:** user\_id, user\_data, case\_grade, case\_score, datetime, instructor\_notes

---

[get\\_case\\_data.php](#)

**Method:** GET

**Authentication:** Required (front\_user\_id session)

**Response Format:**

```
[
  {
    "id": 1,
    "title": "Case Name",
    "isPro": false,
    "isDemo": false,
    "isAccessible": true,
    "requiresUpgrade": false,
    "requiresProUpgrade": false
  }
]
```

**Dependencies:**

- pro\_features.php: isProFeatureEnabled()
- case\_data\_handler.php: getAvailableCases()

**Case Accessibility Logic:**

- Demo users: Only case ID 1 accessible
- Standard users: All non-Pro cases accessible
- Pro users: All cases accessible

---

[get\\_user\\_cases.php](#)

**Method:** GET

**Authentication:** Required (front\_user\_id or user\_id session)

**Query Parameters:** show\_hidden (boolean, optional): Include hidden cases

**Response:** HTML table output

**Table Columns:**

- Case (first line of log)
- Grade
- Score
- Date (formatted as YYYY-MM-DD H:MM AM/PM)
- View Case link (data-case-id attribute)
- Visibility toggle button

**Query Structure:**

```
SELECT id,
       LEFT(user_data, INSTR(user_data, '\n'))
- 1) AS user_data_trunc,
       DATE_FORMAT(datetime, '%Y-%m-%d %l:%i %p') AS dateformatted,
       case_grade, case_score, case_hidden
FROM wp_groupusers_data
WHERE user_id = :userid
      AND (case_hidden = 0 OR case_hidden IS NULL)
ORDER BY datetime DESC
```

**Features:**

- Separate section for hidden cases when show\_hidden=true
- Toggle switch UI for visibility control
- Individual hide/show buttons per case row

[view\\_case.php](#)

**Method:** GET

**Authentication:** Required (front\_user\_id session)

**Query Parameters:** id (integer, required): Case record ID

**Response:** HTML detail view

**Output Sections:** 1. Instructor Comments section (if present) 2. Back to Cases List button 3. Instructor Notes table row 4. Case Details table:

- Date and Time
- Case Grade
- Case Score
- Case Log (multiline formatted)

**Ownership Verification:** Confirms case.user\_id matches session.front\_user\_id before displaying.

**Database Query:**

```
SELECT *  
FROM wp_groupusers_data  
WHERE id = :caseId AND user_id = :userId
```

---

[toggle\\_case\\_visibility.php](#)

**Method:** POST

**Authentication:** Required (front\_user\_id session)

**Request Body (form data):**

- id (integer): Case record ID
- status (string): 'visible' or 'hidden'

**Response:**

```
{  
  "success": true  
}
```

**Toggle Logic:**

- status='visible' sets case\_hidden=1
- status='hidden' sets case\_hidden=0

**Database Operations:**

1. Verify ownership: SELECT user\_id FROM wp\_groupusers\_data WHERE id = ?
  2. Update visibility: UPDATE wp\_groupusers\_data SET case\_hidden = ? WHERE id = ?
- 

[get\\_case\\_details.php](#)

**Method:** GET

**Authentication:** None

**Query Parameters:** id (integer, required): Case ID

**Response:** JavaScript variable assignment

**Content-Type:** application/javascript

**Output Format:**

```
var CurrentCase = {  
  "id": 1,  
  "title": "Case Name",  
  "isPro": false,
```

```
...case data...
};
```

**Validation:**

- Returns console.error if case not found
- Returns console.error if Pro case accessed by non-Pro user

**save\_current\_case.php****Method:** POST**Authentication:** Required (session started)**Request Body (form data):** case (string): Case identifier**Response:**

```
{
  "success": true
}
```

**Actions:** Sets \$\_SESSION['current\_case'] = case value

## 2.2 Session Management APIs

**check\_session.php****Method:** GET**Authentication:** Optional (validates if session exists)**Response Format:**

```
{
  "valid": true,
  "userId": 123,
  "global_termination": false,
  "user_set_preference": false,
  "expired": false,
  "reason": null,
  "debug": {
    "current_time": 1703123456,
    "session_expires": 1703209856,
    "time_remaining": 86400,
    "configured_timeout": 86400
  }
}
```

### Validation Checks:

1. Time-based session expiration
2. License expiration (checked daily)

### Expiration Actions:

- Sets REDIRECT\_URL to current page
- Clears all session variables
- Destroys session

### Database Queries:

```
-- User preference
SELECT session_term_set FROM wp_users WHERE ID = ?

-- License validation
SELECT g.expiration_date
FROM wp_users u
LEFT JOIN wp_groups g ON u.group_id = g.id
WHERE u.ID = ?
```

---

## keepalive.php

**Method:** POST

**Authentication:** Required (front\_user\_id session)

### Response:

```
{
  "success": true,
  "message": "Session refreshed"
}
```

### Actions:

- Regenerates session ID via session\_regenerate\_id(true)
- Updates last\_activity timestamp

**Usage Pattern:** Call periodically (every 4-5 minutes) to maintain active sessions.

---

## terminate\_session.php

**Method:** POST

**Authentication:** None

**Response:** HTTP 200 status



**Actions:**

1. Clears session array: `$_SESSION = array()`
2. Destroys session: `session_destroy()`

---

[reauth.php](#)

**Method:** POST

**Authentication:** None (creates new session)

**Request Body (form data):**

- email (string): User email
- pass (string): Password

**Response:**

```
{
  "success": true
}
```

**Session Variables Created:**

- front\_user\_email
- front\_user\_id
- admin (1 or 0)
- user\_type
- user\_group
- user\_logged\_in (true)
- clear\_cache (true)
- SESSION\_EXPIRES (current time + timeout)

**Password Verification:**

```
$salt = sha1(md5($pass));
$hashed = md5($pass . $salt);
```

**Database Query:**

```
SELECT u.*, g.group_name as user_group,
       CASE WHEN u.user_type = 'admin' THEN 1 ELSE 0 END as is_admin
FROM wp_users u
LEFT JOIN wp_groups g ON u.group_id = g.id
WHERE u.user_email = ? AND u.user_pass = ? AND u.user_status = '1'
```

## 2.3 Pro Features & Licensing APIs

### get\_pro\_status.php

**Method:** GET

**Authentication:** Required (front\_user\_id session)

**Response:**

```
{  
  "isPro": true  
}
```

**Access Determination:** Returns result of isProFeatureEnabled('additional\_cases') function.

---

### Pro Feature Function: isProFeatureEnabled()

**Location:** pro\_features.php

**Parameter:** featureName (string)

**Returns:** boolean

**Access Logic:**

1. Verify user is logged in
2. Block access if user\_type = 'demo'
3. Check user-level Pro status:
  - pro\_status = 'Pro'
  - pro\_expiration\_date NULL or >= today
4. Check group-level Pro status:
  - group.pro\_status = 'Pro'
  - group.pro\_expiration\_date NULL or >= today
5. Return true if either user OR group has valid Pro access

**Database Query:**

```
SELECT u.pro_status as user_pro,  
       u.user_type,  
       u.pro_expiration_date as user_exp,  
       g.pro_status as group_pro,  
       g.pro_expiration_date as group_exp  
FROM wp_users u  
LEFT JOIN wp_groups g ON u.group_id = g.id  
WHERE u.ID = ?
```

---

### update-user-pro-status.php

**Method:** POST

**Authentication:** Required (user\_type must be 'super')

**Request Body (form data):**

- userId (integer): Target user ID
- newStatus (string): 'Pro' or 'No'

**Response:**

```
{
  "success": true
}
```

**Restrictions:** Cannot modify Pro status for users in Demo group

**Pro Upgrade Action:** When upgrading to Pro, sets pro\_expiration\_date to current date + 1 year.

**Database Operations:**

```
UPDATE wp_users SET pro_status = ? WHERE id = ?

-- If upgrading to Pro
UPDATE wp_users SET pro_expiration_date = DATE_ADD(CURDATE(), INTERVAL 1 YEAR) WHERE id = ?
```

## 2.4 Case Management Operations

case\_management.php

**Method:** POST

**Authentication:** Required (front\_user\_id session)

**Request Body (form data):** caseId (integer): Selected case ID

**Response:**

```
{
  "success": true
}
```

**Accessibility Validation:**

1. Pro cases require Pro status
2. Demo users limited to case ID 1
3. Standard users access non-Pro cases

**Session Update:** Sets \$\_SESSION['current\_case'] = caseId on successful selection.

## renderCaseSelector() Function

**Location:** case\_management.php

**Returns:** HTML string

**Output Structure:**

```

<div id="selectcase">
  <fieldset id="fieldsetCase">
    <legend id="legendCase">Case Library:</legend>
    <select id="caseList" name="caseList">
      <option value="defaultCase">Select a Case to Start</option>
      <option value="1">Case Name</option>
      <option value="2" disabled>Pro Case (Pro Only)</option>
      <option value="3" disabled>Demo Case (Upgrade)</option>
    </select>
  </fieldset>
</div>


```

**Label Conventions:**

- (Pro Only) appended to Pro cases for non-Pro users
- (Upgrade) appended to cases inaccessible to Demo users

## 2.5 User Management APIs

### get-user-details.php

**Method:** GET

**Authentication:** None

**Query Parameters:** id (integer): User ID

**Response:**

```

{
  "ID": 123,
  "user_login": "username",
  "user_email": "email@example.com",
  "user_type": "user",
  "pro_status": "Pro",
  "pro_expiration_date": "2025-12-31",
  "group_id": 5,
  "group_name": "Group Name",
  "previous_group": 3,
  "previous_group_name": "Previous Group"
}

```

**Database Query:**

```
SELECT u.*, g.group_name, pg.group_name as previous_group_name
FROM wp_users u
LEFT JOIN wp_groups g ON u.group_id = g.id
LEFT JOIN wp_groups pg ON u.previous_group = pg.id
WHERE u.id = ?
```

---

## update-user-role.php

**Method:** POST

**Authentication:** Required

**Request Body (form data):**

- userId (integer): Target user ID
- newRole (string): 'user', 'admin', 'demo', 'super'

**Response:**

```
{
  "success": true,
  "data": {
    "user_type": "user",
    "group_id": 5,
    "previous_group": null
  }
}
```

**Role Transition Logic:**

**Demo to User:**

- Restores previous\_group if exists
- Otherwise assigns to Individuals group
- Copies group passwords to user record
- Clears previous\_group field

**User to Demo:**

- Stores current group\_id in previous\_group (unless from Individuals)
- Assigns to Demo group
- Copies Demo group passwords to user record

**Password Synchronization:** Updates both user\_pass (hashed) and user\_pass\_clear from destination group.

**Database Transaction:**

```
BEGIN TRANSACTION;
UPDATE wp_users SET
  user_type = ?,
```

```
group_id = ?,
user_pass = ?,
user_pass_clear = ?,
previous_group = ?
WHERE id = ?;
COMMIT;
```

---

## 2.6 Administrative Functions

### toggle\_debug\_mode.php

**Method:** POST

**Authentication:** Required

**Request Body (form data):** mode (string): 'on' or 'off'

**Response:**

```
{
  "success": true,
  "mode": "on"
}
```

**Actions:**

- Calls setDebugMode() function
- Updates configs/debug\_mode.json
- Affects logMessage2() output throughout system

---

### save\_session\_settings.php

**Method:** POST

**Authentication:** Required (user\_type='super' OR userId=3)

**Request Body (JSON):**

```
{
  "session_expiry_amount": 24,
  "session_expiry_unit": "hours"
}
```

**Valid Units:** seconds, minutes, hours, days, months

**Response:**

```
{
  "success": true
}
```

**Database Operation:**

```
INSERT INTO wp_settings (setting_key, setting_value, setting_type)
VALUES (?, ?, ?)
ON DUPLICATE KEY UPDATE
  setting_value = VALUES(setting_value),
  updated_at = CURRENT_TIMESTAMP
```

**Immediate Effect:** Updates current session's SESSION\_EXPIRES timestamp with new calculated timeout.

---

## get-group-details.php

**Method:** GET

**Authentication:** None

**Query Parameters:** id (integer): Group ID

**Response:**

```
{
  "id": 5,
  "group_name": "Group Name",
  "group_pass": "hashed",
  "group_pass_clear": "plaintext",
  "pro_status": "Pro",
  "pro_expiration_date": "2025-12-31",
  "expiration_date": "2026-12-31",
  "group_status": "active"
}
```

**Database Query:**

```
SELECT * FROM wp_groups WHERE id = ?
```

---

## update-group-status.php

**Method:** POST

**Authentication:** Required

**Request Body (form data):**

- groupId (integer): Target group ID
- newStatus (string): 'active' or 'inactive'

**Response:**

```
{
  "success": true
}
```

**Database Operation:**

```
UPDATE wp_groups SET group_status = ? WHERE id = ?
```

---

## 2.7 Session Preference APIs

### set\_session\_preference.php

**Method:** POST

**Authentication:** Required (front\_user\_id session)

**No Input Parameters**

**Response:**

```
{  
  "success": true  
}
```

**Actions:** Updates wp\_users table: session\_termination\_user\_set = 1

**Database Operation:**

```
UPDATE wp_users SET session_termination_user_set = 1 WHERE ID = ?
```

---

### set\_session\_user\_flag.php

**Method:** POST

**Authentication:** Required (front\_user\_id session)

**No Input Parameters**

**Response:**

```
{  
  "success": true  
}
```

**Actions:** Updates wp\_users table: session\_term\_set = 1

**Database Operation:**

```
UPDATE wp_users SET session_term_set = 1 WHERE ID = ?
```

---

### toggle\_global\_session\_termination.php

**Method:** POST

**Authentication:** Required (Super Admin only via is\_super flag)



**Request Body (JSON):**

```
{
  "enabled": true
}
```

**Response:**

```
{
  "success": true,
  "enabled": true
}
```

**Authorization:** Checks wp\_users.is\_super field before allowing operation.

**Database Operation:**

```
INSERT INTO wp_settings (setting_key, setting_value, setting_type)
VALUES ('global_session_termination_enabled', '1', 'boolean')
ON DUPLICATE KEY UPDATE
  setting_value = VALUES(setting_value),
  updated_at = CURRENT_TIMESTAMP
```

[save\\_global\\_session\\_setting.php](#)

**Method:** POST

**Authentication:** Required (user\_type must be 'super')

**Request Body (JSON):**

```
{
  "enabled": true
}
```

**Response:**

```
{
  "success": true
}
```

**Actions:** Sets global\_session\_termination\_enabled setting in wp\_settings table.

**Database Operation:**

```
INSERT INTO wp_settings (setting_key, setting_value, setting_type)
VALUES ('global_session_termination_enabled', ?, 'boolean')
ON DUPLICATE KEY UPDATE
  setting_value = VALUES(setting_value),
  updated_at = CURRENT_TIMESTAMP
```

## 2.8 Pro Status History APIs

### get-pro-history.php

**Method:** GET

**Authentication:** None

**Query Parameters:**

- group\_id (integer, required): Group ID

**Response:**

```
{
  "success": true,
  "history": [
    {
      "id": 1,
      "group_id": 5,
      "old_status": "No",
      "new_status": "Pro",
      "old_expiration_date": null,
      "new_expiration_date": "2025-12-31",
      "changed_by": 3,
      "changed_by_email": "admin@example.com",
      "changed_at": "2025-01-15 10:30:00",
      "notes": "Upgraded to Pro"
    }
  ]
}
```

**Database Query:**

```
SELECT h.*, u.user_email as changed_by_email
FROM wp_pro_status_history h
LEFT JOIN wp_users u ON h.changed_by = u.ID
WHERE h.group_id = ?
ORDER BY h.changed_at DESC
```

---

### revert-pro-status.php

**Method:** POST

**Authentication:** Required

**Request Body (form data):**

- history\_id (integer): History record ID to revert to

**Response:**

```
{
  "success": true
}
```

**Actions:** 1. Retrieves history record by ID 2. Reverts group Pro status to old values 3. Creates new history entry documenting the reversion

#### Database Operations:

```
-- Get history record
SELECT * FROM wp_pro_status_history WHERE id = ?

-- Revert status
UPDATE wp_groups
SET pro_status = ?, pro_expiration_date = ?
WHERE id = ?

-- Log reversion
INSERT INTO wp_pro_status_history
(group_id, old_status, new_status, old_expiration_date, new_expiration_date, change
d_by, notes)
VALUES (?, ?, ?, ?, ?, ?, 'Reverted to previous state')
```

## 2.9 User Settings APIs

update-user-settings.php

**Method:** POST

**Authentication:** None

**Request Body (JSON):**

```
{
  "userId": 123,
  "first_name": "John",
  "last_name": "Doe"
}
```

**Response:**

```
{
  "success": true
}
```

**Allowed Fields:**

- user\_login
- first\_name
- last\_name
- user\_email

- group\_id
- display\_name
- user\_nicename
- user\_url

**Validation:**

- First and last names cannot be empty
- Only updates fields provided in request

**Database Operation:**

```
UPDATE wp_users SET first_name = ?, last_name = ? WHERE ID = ?
```

---

## 2.10 Utility & Helper APIs

log\_session\_failure.php

**Method:** POST

**Authentication:** None

**Request Body (JSON):**

```
{
  "timestamp": "2025-12-21T10:30:00Z",
  "session_id": "abc123",
  "user_agent": "Mozilla/5.0..."
}
```

**Response:**

```
{
  "success": true
}
```

**Actions:** Calls logSessionFailure() from session\_failure\_logging.php to write to log directory.

---

### Helper Function: isDemoUser()

**Location:** demo\_role.php

**Parameter:** userId (integer, optional - defaults to session user)

**Returns:** boolean

**Detection Logic:** Returns true if user.user\_type = 'demo' OR user.user\_group = 'Demo'

**Database Query:**

```
SELECT user_type, user_group
FROM wp_users
WHERE ID = ?
```

---

### Helper Function: getAvailableCases()

**Location:** case\_data\_handler.php

**Parameter:** isPro (boolean)

**Returns:** Array of case objects

#### Case Object Structure:

```
[
  'id' => 1,
  'title' => 'Case Name',
  'isPro' => false,
  'isDemo' => false,
  'isAccessible' => true,
  'requiresUpgrade' => false,
  'requiresProUpgrade' => false
]
```

#### Accessibility Rules:

- Demo users: case ID 1 only
- Standard users: non-Pro cases
- Pro users: all cases

---

## SECTION 3: Database Architecture

**NOTE:** Please reference the separate database documentation file for full details.

### 3.1 Core Tables Reference

#### wp\_users Table

**Primary Key:** ID

#### Key Fields:

- *user\_login*, *user\_email* - Credentials
- *user\_pass*, *user\_pass\_clear* - Password storage
- *user\_type* - Role: user, admin, demo, super
- *pro\_status* - Pro, No
- *pro\_expiration\_date* - Pro access end date
- *group\_id* - Foreign key to wp\_groups
- *previous\_group* - Stored group\_id for demo transitions

- *session\_term\_set* - Session preference flag
  - *session\_termination\_user\_set* - Alternate session flag
  - *user\_status* - Active status
  - *is\_super* - Super admin flag
- 

## wp\_groups Table

**Primary Key:** id

**Key Fields:**

- *group\_name* - Group identifier
  - *group\_pass*, *group\_pass\_clear* - Group passwords
  - *pro\_status* - Pro, No
  - *pro\_expiration\_date* - Pro access end date
  - *expiration\_date* - License expiration
  - *group\_status* - active, inactive
- 

## wp\_groupusers\_data Table

**Primary Key:** id

**Key Fields:**

- *user\_id* - Foreign key to wp\_users
  - *user\_data* - Complete case log text
  - *case\_grade* - High Pass, Pass, Fail, Incomplete
  - *case\_score* - Score in X/Y format
  - *datetime* - Timestamp
  - *instructor\_notes* - Instructor comments
  - *case\_hidden* - Visibility flag
- 

## wp\_settings Table

**Primary Key:** setting\_key

**Key Fields:**

- *setting\_key* - Unique setting identifier
- *setting\_value* - Setting data
- *setting\_type* - Data type: string, integer, boolean, json
- *updated\_at* - Last modification timestamp

**Session Settings:**

- *session\_expiry\_amount*
- *session\_expiry\_unit*

- global\_session\_termination\_enabled

---

## wp\_pro\_status\_history Table

**Primary Key:** id

**Key Fields:**

- *group\_id* - Foreign key to wp\_groups
- *old\_status* - Previous Pro status
- *new\_status* - New Pro status
- *old\_expiration\_date* - Previous expiration
- *new\_expiration\_date* - New expiration
- *changed\_by* - User ID who made change
- *changed\_at* - Timestamp
- *notes* - Change description

**Purpose:** Tracks all Pro status changes for audit trail and reversion capability.

---

## 3.2 Table Relationships

**Users to Groups:**

```
wp_users.group_id → wp_groups.id
wp_users.previous_group → wp_groups.id
```

**Users to Case Data:**

```
wp_users.ID → wp_groupusers_data.user_id
```

**Groups to History:**

```
wp_groups.id → wp_pro_status_history.group_id
```

**Users to History:**

```
wp_users.ID → wp_pro_status_history.changed_by
```

---

## 3.3: Key Fields Documentation

### User Identity Fields

wp\_users.ID

**Type:** Integer, Auto-increment

**Purpose:** Primary key for user records

---

**Referenced by:** wp\_groupusers\_data.user\_id, wp\_pro\_status\_history.changed\_by

**Usage:** Session variable front\_user\_id stores this value

[wp\\_users.user\\_login](#)

**Type:** Varchar

**Purpose:** Username for display and identification

**Constraints:** Unique per user

**Usage:** Displayed in admin interfaces, not used for authentication

[wp\\_users.user\\_email](#)

**Type:** Varchar

**Purpose:** Primary authentication identifier

**Constraints:** Unique, used for login

**Usage:** Converted to lowercase before authentication, stored in front\_user\_email session

[wp\\_users.user\\_pass / user\\_pass\\_clear](#)

**Type:** Varchar (both)

**Purpose:** Hashed and plaintext password storage

**Hashing:** MD5 with SHA1(MD5()) salt

**Usage:** user\_pass used for authentication, user\_pass\_clear for group password sync

---

## Role & Permission Fields

[wp\\_users.user\\_type](#)

**Type:** Enum-style varchar

**Values:** 'user', 'admin', 'demo', 'super'

**Purpose:** Role-based access control

**Hierarchy:** super > admin > user > demo

**Usage:** Checked in session for permission validation

[wp\\_users.is\\_super](#)

**Type:** Boolean/Tinyint

**Purpose:** Super admin flag for system-wide settings access

**Usage:** Required for session timeout configuration, global termination toggle

[wp\\_users.user\\_status](#)

**Type:** Varchar/Integer

**Values:** '1' (active), '0' (inactive)

**Purpose:** Account activation status

**Usage:** Must be '1' for authentication to succeed



## Pro Licensing Fields

`wp_users.pro_status`

**Type:** Varchar

**Values:** 'Pro', 'No'

**Purpose:** Individual Pro feature access

**Evaluation:** Checked alongside group Pro status (either grants access)

`wp_users.pro_expiration_date`

**Type:** Date

**Purpose:** User-level Pro access expiration

**Validation:** Compared against current date, null = no expiration

**Usage:** Set to +1 year on Pro upgrade

`wp_groups.pro_status`

**Type:** Varchar

**Values:** 'Pro', 'No'

**Purpose:** Group-level Pro feature access

**Inheritance:** All group members inherit this status

`wp_groups.pro_expiration_date`

**Type:** Date

**Purpose:** Group-level Pro access expiration

**Validation:** Checked if group Pro status is 'Pro'

---

## Group Membership Fields

`wp_users.group_id`

**Type:** Integer

**Purpose:** Current group membership

**Foreign Key:** References wp\_groups.id

**Usage:** Determines group-level permissions and Pro status

`wp_users.previous_group`

**Type:** Integer, Nullable

**Purpose:** Stores group\_id before Demo role transition

**Usage:** Enables restoration to original group when exiting Demo role

**Set when:** User transitions from non-Individuals group to Demo

**Cleared when:** User transitions from Demo back to User

## Session Control Fields

`wp_users.session_term_set`

**Type:** Boolean/Tinyint

**Purpose:** User has set session termination preference

**Usage:** Flag indicating user acknowledged session settings

`wp_users.session_termination_user_set`

**Type:** Boolean/Tinyint

**Purpose:** Alternate session preference flag

**Usage:** Similar to session\_term\_set, both exist in codebase

---

## Case Data Fields

`wp_groupusers_data.user_id`

**Type:** Integer

**Foreign Key:** References wp\_users.ID

**Purpose:** Case ownership

**Validation:** Checked before allowing case view/hide operations

`wp_groupusers_data.user_data`

**Type:** Text/Longtext

**Purpose:** Complete case log content

**Format:** Multiline text with timestamps and actions

**Parsing:** First line extracted for summary display

`wp_groupusers_data.case_grade`

**Type:** Varchar

**Values:** 'High Pass', 'Pass', 'Fail', 'Incomplete'

**Purpose:** Performance assessment

**Display:** Shown in case history and detail views

`wp_groupusers_data.case_score`

**Type:** Varchar

**Format:** "X/Y" (e.g., "95/100")

**Extraction:** Regex pattern `/- ?\d+\s*\s*\s*\d+/` from case log

**Override:** Can be provided directly in save request

`wp_groupusers_data.case_hidden`

**Type:** Boolean/Tinyint

**Values:** 0 (visible), 1 (hidden)

**Purpose:** User-controlled case visibility

**Toggle Logic:** Inverted (visible status sets to 1)

[wp\\_groupusers\\_data.instructor\\_notes](#)

**Type:** Text

**Purpose:** Instructor feedback on saved case

**Display:** Shown prominently at top of case detail view

---

## Configuration Fields

[wp\\_settings.setting\\_key](#)

**Type:** Varchar, Unique

**Purpose:** Configuration identifier

**Examples:** 'session\_expiry\_amount', 'session\_expiry\_unit',  
'global\_session\_termination\_enabled'

[wp\\_settings.setting\\_value](#)

**Type:** Text

**Purpose:** Configuration data

**Format:** String, parsed based on setting\_type

[wp\\_settings.setting\\_type](#)

**Type:** Varchar

**Values:** 'string', 'integer', 'boolean', 'json'

**Purpose:** Data type indicator for parsing

---

## Audit Trail Fields

[wp\\_pro\\_status\\_history.changed\\_by](#)

**Type:** Integer

**Foreign Key:** References wp\_users.ID

**Purpose:** Tracks who made Pro status changes

**Join:** Used with wp\_users.user\_email for display

[wp\\_pro\\_status\\_history.changed\\_at](#)

**Type:** Datetime

**Purpose:** Timestamp of Pro status change

**Default:** CURRENT\_TIMESTAMP

**Ordering:** DESC for most recent first

---

[wp\\_pro\\_status\\_history.notes](#)

**Type:** Text

**Purpose:** Reason for Pro status change

**Examples:** “Upgraded to Pro”, “Reverted to previous state”

## 3.4: Query Patterns

### Authentication Queries

#### User Login Validation

```
SELECT u.*, g.group_name as user_group,
       CASE WHEN u.user_type = 'admin' THEN 1 ELSE 0 END as is_admin
FROM wp_users u
LEFT JOIN wp_groups g ON u.group_id = g.id
WHERE u.user_email = ?
      AND u.user_pass = ?
      AND u.user_status = '1'
```

**Purpose:** Authenticate user and load session data

**Used in:** reauth.php, login system

**Returns:** Single user record with group name and admin flag

### Pro Feature Access Queries

#### Check User/Group Pro Status

```
SELECT u.pro_status as user_pro,
       u.user_type,
       u.pro_expiration_date as user_exp,
       g.pro_status as group_pro,
       g.pro_expiration_date as group_exp
FROM wp_users u
LEFT JOIN wp_groups g ON u.group_id = g.id
WHERE u.ID = ?
```

**Purpose:** Determine if user has Pro access via individual or group

**Used in:** isProFeatureEnabled() function

**Logic:** Returns true if either user\_pro OR group\_pro is valid

### Session Validation Queries

#### License Expiration Check

```
SELECT g.expiration_date
FROM wp_users u
```

```
LEFT JOIN wp_groups g ON u.group_id = g.id
WHERE u.ID = ?
```

**Purpose:** Daily license validation during session

**Used in:** session\_manager.php, check\_session.php

**Frequency:** Once per 24 hours per user

**Cached in:** \$\_SESSION['LAST\_EXPIRATION\_CHECK']

#### User Session Preference

```
SELECT session_term_set
FROM wp_users
WHERE ID = ?
```

**Purpose:** Check if user has set session preferences

**Used in:** check\_session.php

**Returns:** Boolean flag

## Case Management Queries

### User Case History (Visible Only)

```
SELECT id,
       LEFT(user_data, INSTR(user_data, '\n') - 1) AS user_data_trunc,
       DATE_FORMAT(datetime, '%Y-%m-%d %l:%i %p') AS dateformatted,
       case_grade, case_score, case_hidden
FROM wp_groupusers_data
WHERE user_id = :userid
      AND (case_hidden = 0 OR case_hidden IS NULL)
ORDER BY datetime DESC
```

**Purpose:** Display user's visible saved cases

**Used in:** get\_user\_cases.php

**Features:** Extracts first line only, formats datetime, filters hidden

### User Case History (Including Hidden)

```
SELECT id,
       LEFT(user_data, INSTR(user_data, '\n') - 1) AS user_data_trunc,
       DATE_FORMAT(datetime, '%Y-%m-%d %l:%i %p') AS dateformatted,
       case_grade, case_score, case_hidden
FROM wp_groupusers_data
WHERE user_id = :userid
      AND case_hidden = 1
ORDER BY datetime DESC
```

**Purpose:** Display user's hidden cases separately

**Used in:** get\_user\_cases.php when show\_hidden=true

### Case Detail with Ownership Verification

```
SELECT *
FROM wp_groupusers_data
WHERE id = :caseId
AND user_id = :userId
```

**Purpose:** Retrieve case details with security check

**Used in:** view\_case.php

**Security:** Returns empty if user doesn't own case

### Save New Case

```
INSERT INTO wp_groupusers_data
(user_id, user_data, case_grade, case_score, datetime, instructor_notes)
VALUES (:user_id, :case_log, :case_grade, :case_score, NOW(), :instructor_notes)
```

**Purpose:** Save completed case to database

**Used in:** save\_case\_data.php

**Returns:** Last insert ID for confirmation

## User Management Queries

### User Details with Groups

```
SELECT u.*, g.group_name, pg.group_name as previous_group_name
FROM wp_users u
LEFT JOIN wp_groups g ON u.group_id = g.id
LEFT JOIN wp_groups pg ON u.previous_group = pg.id
WHERE u.id = ?
```

**Purpose:** Complete user information including current and previous groups

**Used in:** get-user-details.php, admin interfaces

**Joins:** Both current group and previous\_group for Demo role handling

### Update User Role with Group Transition

```
-- Get current state
SELECT user_type, group_id, previous_group
FROM wp_users
WHERE id = ?

-- Update with transaction
BEGIN TRANSACTION;
UPDATE wp_users SET
  user_type = :new_role,
  group_id = :new_group,
  user_pass = :new_pass,
  user_pass_clear = :new_pass_clear,
  previous_group = :prev_group
WHERE id = :user_id;
COMMIT;
```

**Purpose:** Role transition with automatic group and password management

**Used in:** update-user-role.php

**Transaction:** Ensures atomic update of related fields

---

## Group Administration Queries

### Group Details

```
SELECT *  
FROM wp_groups  
WHERE id = ?
```

**Purpose:** Complete group information

**Used in:** get-group-details.php, admin interfaces

### Update Group Status

```
UPDATE wp_groups  
SET group_status = ?  
WHERE id = ?
```

**Purpose:** Activate or deactivate group

**Used in:** update-group-status.php

### Update Group Pro Status

```
UPDATE wp_groups  
SET pro_status = ?,  
    pro_expiration_date = ?  
WHERE id = ?
```

**Purpose:** Change group Pro licensing

**Used in:** update-group-status.php, Pro management

---

## Pro Status History Queries

### Get Group Pro History

```
SELECT h.*, u.user_email as changed_by_email  
FROM wp_pro_status_history h  
LEFT JOIN wp_users u ON h.changed_by = u.ID  
WHERE h.group_id = ?  
ORDER BY h.changed_at DESC
```

**Purpose:** Audit trail of Pro status changes

**Used in:** get-pro-history.php

**Display:** Shows who made changes and when

### Log Pro Status Change

```
INSERT INTO wp_pro_status_history
(group_id, old_status, new_status, old_expiration_date,
new_expiration_date, changed_by, notes)
VALUES (?, ?, ?, ?, ?, ?, ?)
```

**Purpose:** Record Pro status change for audit

**Used in:** update-user-pro-status.php, revert-pro-status.php

### Revert Pro Status

```
-- Get history record
SELECT * FROM wp_pro_status_history WHERE id = ?

-- Apply old values
UPDATE wp_groups
SET pro_status = :old_status,
    pro_expiration_date = :old_expiration
WHERE id = :group_id

-- Log reversion
INSERT INTO wp_pro_status_history
(group_id, old_status, new_status, old_expiration_date,
new_expiration_date, changed_by, notes)
VALUES (?, ?, ?, ?, ?, ?, 'Reverted to previous state')
```

**Purpose:** Undo Pro status change

**Used in:** revert-pro-status.php

**Creates:** New history entry documenting reversion

## Configuration Queries

### Get Session Settings

```
SELECT setting_key, setting_value
FROM wp_settings
WHERE setting_key IN ('session_expiry_amount', 'session_expiry_unit')
```

**Purpose:** Load timeout configuration

**Used in:** getSessionExpirySeconds() function

**Returns:** Key-value pairs for calculation

### Update Configuration Setting

```
INSERT INTO wp_settings (setting_key, setting_value, setting_type)
VALUES (?, ?, ?)
ON DUPLICATE KEY UPDATE
    setting_value = VALUES(setting_value),
    updated_at = CURRENT_TIMESTAMP
```



**Purpose:** Store or update configuration value

**Used in:** save\_session\_settings.php, save\_global\_session\_setting.php

**Upsert:** Creates if missing, updates if exists

---

## Common Query Patterns

### Ownership Verification Pattern

```
SELECT user_id FROM table_name WHERE id = ?  
-- Then verify: user_id == $_SESSION['front_user_id']
```

**Used in:** toggle\_case\_visibility.php, view\_case.php

**Purpose:** Security check before allowing operations

### Super Admin Check Pattern

```
SELECT is_super FROM wp_users WHERE ID = ?  
-- Then verify: is_super == 1
```

**Used in:** toggle\_global\_session\_termination.php

**Purpose:** Restrict system-wide settings to super admins

### Join Pattern for User Context

```
FROM wp_users u  
LEFT JOIN wp_groups g ON u.group_id = g.id  
WHERE ...
```

**Used throughout:** Most user-related queries

**Purpose:** Include group context with user data

---

## SECTION 4: Session Management

### 4.1 Configuration & Storage

**Storage Location:** wp\_settings database table

#### Configuration Settings:

- session\_expiry\_amount (integer): Numeric timeout value
- session\_expiry\_unit (string): Time unit
- global\_session\_termination\_enabled (boolean): Global toggle

#### Default Values:

- Amount: 24
- Unit: hours
- Global termination: false

## 4.2 Session Lifecycle

### Initialization Process

1. Load settings via getSessionExpirySeconds() from wp\_settings
2. Convert to seconds based on unit
3. Set cookie **Parameters:** session\_set\_cookie\_params(\$timeout)
4. Start session: session\_start()
5. Set expiry: \$\_SESSION['SESSION\_EXPIRES'] = time() + timeout

**Excluded Pages:** login.php, register.php, index.php

### Validation Checks

#### Time-Based Expiration:

```
if (time() > $_SESSION['SESSION_EXPIRES']) {
    // Session expired
    - clear and destroy
}
```

**License Expiration:** Checked once per 24 hours via LAST\_EXPIRATION\_CHECK session variable.

```
SELECT g.expiration_date
FROM wp_users u
LEFT JOIN wp_groups g ON u.group_id = g.id
WHERE u.ID = ?
```

### Expiration Handling

#### Actions on Expiration:

```
$_SESSION['REDIRECT_URL'] = $_SERVER['REQUEST_URI'];
session_unset();
session_destroy();
```

#### AJAX Response:

```
{
  "valid": false,
  "expired": true,
  "reason": "session_expired|license_expired"
}
```

## 4.3 Timeout Calculations

### Conversion Formula

#### getSessionExpirySeconds() Function:

```
switch($unit) {
    case 'seconds': return $amount;
    case 'minutes': return $amount * 60;
    case 'hours': return $amount * 3600;
    case 'days': return $amount * 86400;
    case 'months': return $amount * 2592000;
    Default: return 86400;
}
```

**Examples:**

- 24 hours = 86,400 seconds
- 30 minutes = 1,800 seconds
- 7 days = 604,800 seconds
- 1 month = 2,592,000 seconds

## 4.4 Validation & Expiration

### Overview

Session validation makes sure that authenticated users maintain valid access throughout their interaction with the simulator. The system balances security requirements with user experience, especially for extended case sessions that may last 30-60+ minutes.

**Key Components:**

- Real-time session status monitoring
- Configurable timeout enforcement
- License expiration verification
- Non-disruptive re-authentication
- Work preservation during session recovery

### 4.4.1 Session Status Checking

**Primary Validation Endpoint:** /api/check\_session.php

**Method:** GET

**Authentication:** Validates existing session

#### *Complete Response Structure*

```
{
  "valid": true,
  "userId": 123,
  "global_termination": false,
  "user_set_preference": false,
  "expired": false,
  "reason": null,
  "debug": {
    "current_time": 1703123456,
```

```

    "session_expires": 1703209856,
    "time_remaining": 86400,
    "configured_timeout": 86400
  }
}

```

#### Response Field Reference

Field	Type	Description
valid	boolean	Session authentication status
userId	integer/null	Authenticated user ID or null
global_termination	boolean	System-wide termination setting
user_set_preference	boolean	User has acknowledged session settings
expired	boolean	Session has expired
reason	string/null	Expiration reason code
debug	object	Diagnostic timing information

#### Reason Codes

Code	Description	Recommended Action
session_expired	Time-based timeout exceeded	Display re-authentication modal
license_expired	Group license past expiration	Redirect to login with expired flag
null	Session valid, no expiration	Continue normal operation

### 4.4.2 Expiration Types

#### Time-Based Session Expiration

Sessions expire when the current server time exceeds the stored expiration timestamp.

##### Trigger Condition:

```
current_time > SESSION_EXPIRES
```

**Configurable Parameters:** - Timeout amount (numeric value) - Timeout unit (seconds, minutes, hours, days, months) - Global termination toggle (system-wide override)

**Default Configuration:** 24 hours

**Admin Configuration Location:** Super Admin Settings Panel

#### License-Based Expiration

Group licenses have expiration dates that override session validity.

**Validation Frequency:** Once per 24-hour period per session

**Database Check:** - Retrieves group.expiration\_date for user's assigned group - Compares against current date - Caches result to prevent repeated queries

**On License Expiration:** - Session immediately invalidated - User redirected to login page with expired parameter - Re-authentication not available until license renewed

#### 4.4.3 Expiration Handling Process

When a session expires, the system executes the following sequence:

##### *Step 1: State Preservation*

Store current page URL for post-authentication redirect

##### *Step 2: Session Cleanup*

Clear all session variables  
Destroy PHP session

##### *Step 3: Response Generation*

**For AJAX/API Requests:**

```
{
  "valid": false,
  "expired": true,
  "reason": "session_expired"
}
```

**For Standard Page Requests:** - Session cleared server-side - Frontend polling detects expiration  
- Re-authentication modal displayed

#### 4.4.4 Re-Authentication System

The re-authentication system allows users to restore their session without page reload, preserving unsaved work such as in-progress case data.

##### *Re-Authentication Endpoint*

**Endpoint:** /api/reauth.php

**Method:** POST

**Content-Type:** application/x-www-form-urlencoded

##### *Request Parameters*

Parameter	Type	Required	Description
email	string	Yes	User email address

Parameter	Type	Required	Description
pass	string	Yes	User password

*Success Response*

```
{
  "success": true
}
```

**Actions on Success:** - Session variables restored - User context re-established - Session expiration timestamp reset - User continues without page reload

*Failure Response*

```
{
  "success": false,
  "message": "Email or password do not match"
}
```

*Session Variables Restored*

Variable	Description
front_user_id	Primary user identifier
front_user_email	User email address
user_type	Role (user, admin, demo, super)
user_group	Assigned group name
admin	Admin privilege flag
user_logged_in	Authentication state
SESSION_EXPIRES	New expiration timestamp

#### 4.4.5 Session Maintenance (Keepalive)

For extended simulator sessions, the keepalive mechanism prevents timeout during legitimate active use.

*Keepalive Endpoint*

**Endpoint:** /api/keepalive.php

**Method:** POST

**Content-Type:** None required

*Response*

```
{
  "success": true,
  "message": "Session refreshed"
}
```

*Failure Response*

```
{
  "success": false,
  "message": "No session"
}
```

*Usage Guidelines*

- Call during active simulator use
- Prevents timeout during long case sessions
- Does not extend sessions for idle users
- Frontend handles automatic keepalive timing

#### 4.4.6 Session Termination

*Manual Termination Endpoint***Endpoint:** /api/terminate\_session.php**Method:** POST**Response:** HTTP 200 status**Actions:** - Clears session array - Destroys PHP session - No redirect performed*Kill Session Endpoint***Endpoint:** /api/kill\_session.php**Method:** POST**Purpose:** Force immediate session termination for security scenarios

#### 4.4.7 Frontend Integration Requirements

*Session Monitoring*

The frontend must implement session status monitoring to detect expiration and trigger re-authentication.

**Required Behaviors:**

1. Periodic polling of check\_session.php
2. Detection of expired status in response
3. Display of re-authentication modal on expiration
4. Prevention of data loss during re-authentication
5. Session restoration without page reload

*Re-Authentication Modal*

**Required Elements:** - Email input field - Password input field - Submit handler calling reauth.php  
- Error message display area - Non-dismissible during active expiration

**On Successful Re-Authentication:** - Close modal - Resume normal operation - Retry any pending operations (e.g., case save)

### Case Save Integration

Case save operations must handle session expiration gracefully:

1. Attempt save operation
2. If SESSION\_EXPIRED error returned:
  - i. Preserve case data in memory
  - ii. Display re-authentication modal
  - iii. After successful re-auth, retry save
3. Confirm save success to user

## 4.4.8 Troubleshooting

### Common Issues

Symptom	Likely Cause	Resolution
Immediate expiration after login	Clock synchronization issue	Verify server time accuracy
Session expires during active case	Keepalive not reaching server	Check network connectivity
Re-auth succeeds but session expires again	Database timeout settings	Verify wp_settings values
License expired message	Group expiration date passed	Renew group license

### Debug Information

The check\_session.php endpoint returns debug timing information:

```
"debug": {
  "current_time": 1703123456,
  "session_expires": 1703209856,
  "time_remaining": 86400,
  "configured_timeout": 86400
}
```

**Interpreting Debug Data:** - current\_time: Server timestamp at request - session\_expires: When session will expire - time\_remaining: Seconds until expiration - configured\_timeout: Database timeout setting in seconds

### Log Files

Session-related issues logged to:

- /logs/advanced\_logs/
- Detailed session events (when debug mode enabled)
- Server error log
- PHP session errors



#### 4.4.9 Configuration Reference

##### *Database Settings (wp\_settings table)*

setting_key	setting_type	Description
session_expiry_amount	integer	Numeric timeout value
session_expiry_unit	string	Unit: seconds, mins, hours, days, mos
global_session_termination_enabled	boolean	System-wide termination toggle

##### *User Flags (wp\_users table)*

Field	Type	Description
session_term_set	boolean	User acknowledged session settings
session_termination_user_set	boolean	User preference flag

#### 4.4.10 Security Considerations

- Sessions validated server-side on every protected request
- Re-authentication requires valid credentials
- Session IDs regenerated appropriately to prevent fixation
- License validation prevents access after subscription expiration
- Debug information available only in authenticated responses

## SECTION 5: Integration Guide

### 5.1 Frontend JavaScript Patterns

#### Session Monitoring

##### Check Session Status (Every 5 Seconds):

```
setInterval(async () => {  
  const response = await fetch('/api/check_session.php');  
  const data = await response.json();  
  
  if (!data.valid && data.expired) {  
    if (data.reason === 'session_expired') {  
      showReauthModal();  
    } else if (data.reason === 'license_expired') {  
      window.location.href = '/login.php?expired=1';  
    }  
  }  
}, 5000);
```

##### Maintain Session (Every 4 Minutes):

```
setInterval(async () => {  
  await fetch('/api/keepalive.php', { method: 'POST' });  
}, 240000);
```

---

### Case Operations

#### Save Case Data:

```
const saveCase = async (caseData) => {  
  const response = await fetch('/api/save_case_data.php', {  
    method: 'POST',  
    headers: { 'Content-Type': 'application/json' },  
    body: JSON.stringify({  
      caseLog: caseData.log,  
      grade: caseData.grade,  
      clientCaseScore: caseData.score,  
      clientCaseTime: caseData.duration,  
      clientCaseName: caseData.name  
    })  
  });  
  
  const result = await response.json();  
  
  if (!result.success) {  
    if (result.errorType === 'SESSION_EXPIRED') {  
      showReauthModal();  
    }  
  }  
}
```

```

    } else {
        displayErrorMessage(result.message);
    }
} else {
    displaySuccessMessage('Case saved successfully');
}
};

```

#### Check Pro Status:

```

const checkProAccess = async () => {
    const response = await fetch('/api/get_pro_status.php');
    const data = await response.json();

    if (data.isPro) {
        enableProFeatures();
    } else {
        displayUpgradePrompts();
    }
};

```

---

## 5.2 PHP Include Standards

### Standard API Structure

```

<?php
// Bootstrap system
include($_SERVER['DOCUMENT_ROOT'] . '/includes/include_all.inc.php');

// Set response format
header('Content-Type: application/json');

// Verify authentication
if (!isset($_SESSION['front_user_id'])) {
    echo json_encode(['error' => 'Not authenticated']);
    exit;
}

// Process request
// ... main logic ...

// Return response
echo json_encode($response);

```

---

## Common Dependencies

- **include\_all.inc.php** - Database connection, session, core functions
  - **pro\_features.php** - Pro feature access checks
  - **demo\_role.php** - Demo user detection
  - **case\_data\_handler.php** - Case data operations
  - **save\_logging.php** - Case save logging
  - **session\_failure\_logging.php** - Session failure logging
- 

## 5.3 API Request/Response Formats

### Response Format Standards

All API endpoints return consistent JSON structure:

```
{
  "success": boolean,
  "error": "string (optional)",
  "message": "string (optional)",
  "errorType": "string (optional)",
  "technicalDetails": "string (optional)"
}
```

---

## 5.4 Error Handling Patterns

### Error Type Reference

- **SESSION\_EXPIRED** - Session timeout, show re-authentication
- **LICENSE\_EXPIRED** - Group license expired, redirect to renewal
- **VALIDATION\_ERROR** - Invalid input data
- **DATABASE\_ERROR** - Database operation failed
- **CONNECTION\_ERROR** - Database connection issue
- **TIMEOUT\_ERROR** - Query execution timeout
- **DUPLICATE\_ERROR** - Duplicate data detected
- **GENERAL\_ERROR** - Unexpected error occurred
- **PERMISSION\_ERROR** - Insufficient access rights

---

## SECTION 6: Logging & Debugging

### 6.1 Debug Mode Configuration

#### Configuration File

**Location:** configs/debug\_mode.json

**Structure:**

```
{  
  "save_logging_enabled": true  
}
```

**Toggle API:** toggle\_debug\_mode.php

---

#### Debug Functions

##### isDebugModeEnabled()

- **Location:** includes/debug\_mode.php
- **Returns:** boolean
- **Default:** true if config file doesn't exist

##### setDebugMode(\$enabled)

- **Location:** includes/debug\_mode.php
- **Parameter:** enabled (boolean)
- Updates configs/debug\_mode.json
- **Returns:** boolean success status

##### logCaseSaveDebug()

- **Location:** includes/debug\_mode.php
- Only logs when debug mode enabled
- Delegates to logCaseSave() function
- **Parameters:** userId, username, email, grade, score, status, errorType, errorMsg, insertId, caseName

### 6.2 Case Save Logging

#### Log Function: logCaseSave()

- **Location:** includes/save\_logging.php
- **Output:** logs/save\_logs/saves\_log\_YYYY\_W##.txt
- **Format:** Weekly log files
- **Called by:** logCaseSaveDebug() when debug enabled

### Log Entry Format:

```
[2025-12-21 10:30:45] User: john.doe (john@example.com) | Grade: A | Score: 95/100  
| Status: SUCCESS | Case: Cardiac Arrest | ID: 12345
```

## 6.3 Session Failure Logging

**Log Function:** logSessionFailure(\$data)

- **Location:** includes/session\_failure\_logging.php
- **Output:** logs/ directory
- **Purpose:** Client-side session failure tracking
- **API Endpoint:** log\_session\_failure.php

### Logged Information:

- Timestamp
- Session ID
- User agent
- Client-provided debug data

## 6.4 Log File Locations

### Save Logs:

- **Path:** logs/save\_logs/
- **Pattern:** saves\_log\_YYYY\_W###.txt
- **Rotation:** Weekly
- **Content:** Case save attempts and results

### Advanced Logs:

- **Path:** logs/advanced\_logs/
- **Pattern:** Various debug outputs
- **Content:** System debug information when debug mode enabled

### User Actions:

- **Path:** logs/user\_actions.log
- **Content:** User activity tracking

## SECTION 7: File Organization

### 7.1 Password Hashing Method

#### Current Implementation

```
$salt = sha1(md5($password));  
$hashed = md5($password . $salt);
```

#### Usage:

- User authentication (reauth.php)
- Password synchronization (update-user-role.php)
- Group password management

#### Storage:

- *user\_pass* - Hashed password
  - *user\_pass\_clear* - Plain text password
  - *group\_pass* - Hashed group password
  - *group\_pass\_clear* - Plain text group password
- 

### 7.2 Session Variable Standards

#### User Identity Variables

##### Primary Identifiers:

- *front\_user\_id* - Main user identifier
- *front\_user\_email* - User email address
- *user\_id* - Alternate user identifier

##### Authentication State:

- *user\_logged\_in* - Login status boolean
- *admin* - Admin privilege flag (1/0)
- *user\_type* - Role: user, admin, demo, super
- *user\_group* - Group name

##### Session Control:

- **SESSION\_EXPIRES** - Expiration timestamp
- **LAST\_EXPIRATION\_CHECK** - License check timestamp
- **REDIRECT\_URL** - Post-login redirect path
- *last\_activity* - Activity timestamp
- *current\_case* - Currently selected case ID
- *clear\_cache* - Cache clearing flag

## 7.3 Authentication Levels

### Access Control Matrix

**Public Access:** None recommended

- all endpoints should verify authentication

**User Level:**

- Case operations (save, view, list)
- Session management (check, keepalive)
- Pro status checking
- Case selection

**Admin Level:**

- User role management
- Group status management
- Debug mode toggle
- Pro status updates

**Super Admin Level:**

- Session timeout configuration
- Global session termination toggle
- System-wide settings

**Special Access:**

- userId=3
- Alternate super admin access for session settings

---

## 7.4 Database Connection Pattern

### Standard Connection Code

```
if (!isset($pdo)) {  
    $pdo = new PDO(  
        "mysql:host=" . DB_HOST . ";dbname=" . DB_NAME,  
        DB_USER,  
        DB_PASS  
    );  
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
}
```

**Connection Lifecycle:**

- Established in includes/include\_all.inc.php
- Reused throughout request lifecycle
- PDO with exception error mode
- Prepared statements used throughout for SQL injection prevention



## 7.5 Naming Conventions

### 7.5.1 PHP Files

#### API Endpoints:

- Mixed conventions: underscores and hyphens both used
- Underscores: case\_data.php, check\_session.php, save\_case\_data.php
- Hyphens: get-user-details.php, update-pro-status.php, revert-pro-status.php

#### Includes:

- Underscore separation: admin\_functions.php, session\_manager.php, user\_settings\_functions.php
- Special case: include\_all.inc.php uses .inc.php extension

#### Admin Files:

- Lowercase with no separators: cases.php, groups.php, users.php
- Compound names: casesdetail.php, casessummary.php

#### JavaScript Files

- No separators: caseswitch.js, flowmeter.js, loader.js
- Hyphens: screen-warning.js
- Underscores for variants: scripts\_app.js, scripts\_pages.js, settings\_cases.js

### 7.5.2 Directories

#### camelCase:

- hemoWaveforms/
- respWaveforms/

#### Underscores:

- case\_data/
- cases\_updated/
- modal\_data/
- save\_logs/
- advanced\_logs/

### 7.5.3 Configuration Files

#### Pattern: name.config.php

- db.config.php
- general.config.php
- mysql.config.php

#### JSON configs:

- debug\_mode.json
- session\_settings.json

## 7.5.4 Backup Files

### Archive backups:

- With date: directoryname\_YYYY-MM-DD.tgz
- With descriptor: case\_data.old\_YYYY-MM-DD.tgz

### Automated backups:

- Pattern: backup\_app.anesoft.com\_type\_YYYY-MM-DD\_HH-MM.extension
- Files: backup\_app.anesoft.com\_files\_2025-01-15\_14-30.tar.gz
- Database: backup\_app.anesoft.com\_database\_2025-01-15\_14-30.sql.gz

### Manual backups file (often in .bak directories):

- Backup file before edited: filename.php.bak
- Old file before edited: filename.php.old
- Restoration marker: filename.php.from\_bak
- New file backed up: filename.php.new
- Developer version before admin edited: filename.php.evx

---

## 7.6 App Directory Structure

- [anesoft.com](#) – the wordpress site (for marketing)
- [anesoftcaselogs.com](#) – deprecated old directory
- [app.anesoft.com](#) – the live main app (not wordpress)
- [demo.anesoft.com](#) – reserved for a future demo version
- [dev.anesoft.com](#) – the dev app - for development of main app
- [demos/baxter.anesoft.com](#) - has special features for the Baxter group
- [dev.vet.anesoft.com](#) – the vet dev app - for development of the veterinarian site
- [vet.anesoft.com](#) – the live veterinarian app (not wordpress)

### Important Notes:

- Veterinarian app is a separate app. Anesoft chose to make a separate app rather than a branch.
- **Veterinarian app will NOT be updated when dev.anesoft.com is pushed live.**
- **Vet to Live** requires additional fees due to separate structure. It **IS possible** to make a branched version in future development which will eliminate duplicate fees. See proposal sent via email.

## SECTION 8: Backup & Deployment Ops

### 8.1 Backup File Locations

#### Manual Backups (Through Simulator Admin Interface)

**Primary Backup Directory:** /home/anesoft/public\_html/backups/

#### File System Backups:

- **Patterns:**
  - backup\_app.anesoft.com\_files\_YYYY-MM-DD\_HH-MM.tar.gz
  - backup\_app.anesoft.com\_files\_YYYY-MM-DD\_HH-MM.zip
- **Created by:** /api/create\_backup.php and /api/create\_backup\_zip.php

#### Database Backups:

- **Pattern:** backup\_app.anesoft.com\_database\_YYYY-MM-DD\_HH-MM.sql.gz
- **Created by:** /api/create\_db\_backup.php

#### Progress Tracking:

- **Location:** /home/anesoft/public\_html/backups/db\_backup\_progress.json
- Contains real-time backup progress during database backup operations

#### Automatic Backups (Through cPanel Admin Interface):

- Physically located on a separate drive only accessible by web host (size constraints)
- Restore automatic backups through the cPanel interface

---

### 8.2 Version Control Patterns

#### Manual Backup System

The system creates timestamped backups through API endpoints in /api/ directory.

**File System Backups:** - **create\_backup.php** - Creates tar.gz compressed archives -  
**create\_backup\_zip.php** - Creates ZIP compressed archives

**File naming:** backup\_app.anesoft.com\_files\_YYYY-MM-DD\_HH-MM.tar.gz

**Source:** /home/anesoft/public\_html/app.anesoft.com

**Destination:** /home/anesoft/public\_html/backups

- Uses PHP PharData class for tar.gz compression
- Uses PHP ZipArchive for zip compression
- Processes files recursively with progress logging

### Database Backups: - create\_db\_backup.php

- Creates compressed SQL dumps

**File naming:** backup\_app.anesoft.com\_database\_YYYY-MM-DD\_HH-MM.sql.gz

**Destination:** /home/anesoft/public\_html/backups

- Creates uncompressed SQL dump first
- Compresses using gzopen/gzwrite
- Deletes uncompressed file after compression
- Processes tables in chunks of 1000 rows
- Includes progress tracking with percentage updates

### Timestamp Format

All manual backups use consistent timestamp format:

```
YYYY-MM-DD_HH-MM
```

Example filenames:

```
backup_app.anesoft.com_files_2026-01-15_14-30.tar.gz  
backup_app.anesoft.com_database_2026-01-15_14-30.sql.gz
```

---

## 8.3 Push to Live Mechanism

- Presently only possible through web host (Everplex)
- Web host (Everplex) created a custom system procedure for pushing live without data loss
- This procedure will also make a manual back up of live app before push
- Option for creating an interface for super admin to push to live is possible in future development

---

## 8.4 Push to Live Procedure

1. Send a push request to web host and web host will push to live (via email or ticket)
2. This will deploy the dev app to the production app
3. Please allow up to 24 hours for push completion

### Important Notes:

- Veterinarian app will remain unaffected, requires a separate push request to the web host
- Baxter app will remain unaffected, requires a separate push request to the web host