**COLLEGE OF BUSINESS**

**EDUCATION**

**STUDENT'S NAME:** JOHNSON HERMAN MJWAHUZI

**MODULE**:MOBILE APPLICATION DEVELOPMENT

**DEPARTMENT NAME:** INFORMATION TECHNOLOGY

**PROGRAMME:**BACHELOR OF INFORMATION TECHNOLOGY

**REGISTRATION NUMBER:** 03.2402.01.01.2023

**LECTURE`S NAME:** KELVIN MBUYA

**ACADEMIC YEAR:** 2025-2026

# STREAMING PLATFORM BACKEND SYSTEM

## TABLE OF CONTENTS

## 1. EXECUTIVE SUMMARY

This project implements a comprehensive backend system for an online/offline streaming platform using PHP and MySQL. The system provides full user authentication, video management, streaming capabilities, and RESTful API endpoints for client applications. Built with security and scalability in mind, it follows MVC architecture principles and implements industry best practices for web application development.

Key Features Implemented:

• User registration and authentication with JWT tokens

• Video upload, storage, and streaming with range support

• Role-based access control (user/admin)

• RESTful API design following industry standards

• Secure database operations with prepared statements

• Comprehensive error handling and logging

• Testing suite for API validation

## 2. SYSTEM REQUIREMENTS & ARCHITECTURE

### 2.1 System Requirements

The system was designed to meet the following functional requirements:

• User registration and authentication system

• Video upload and management interface

• Video streaming with support for seeking and partial content

• User subscription management

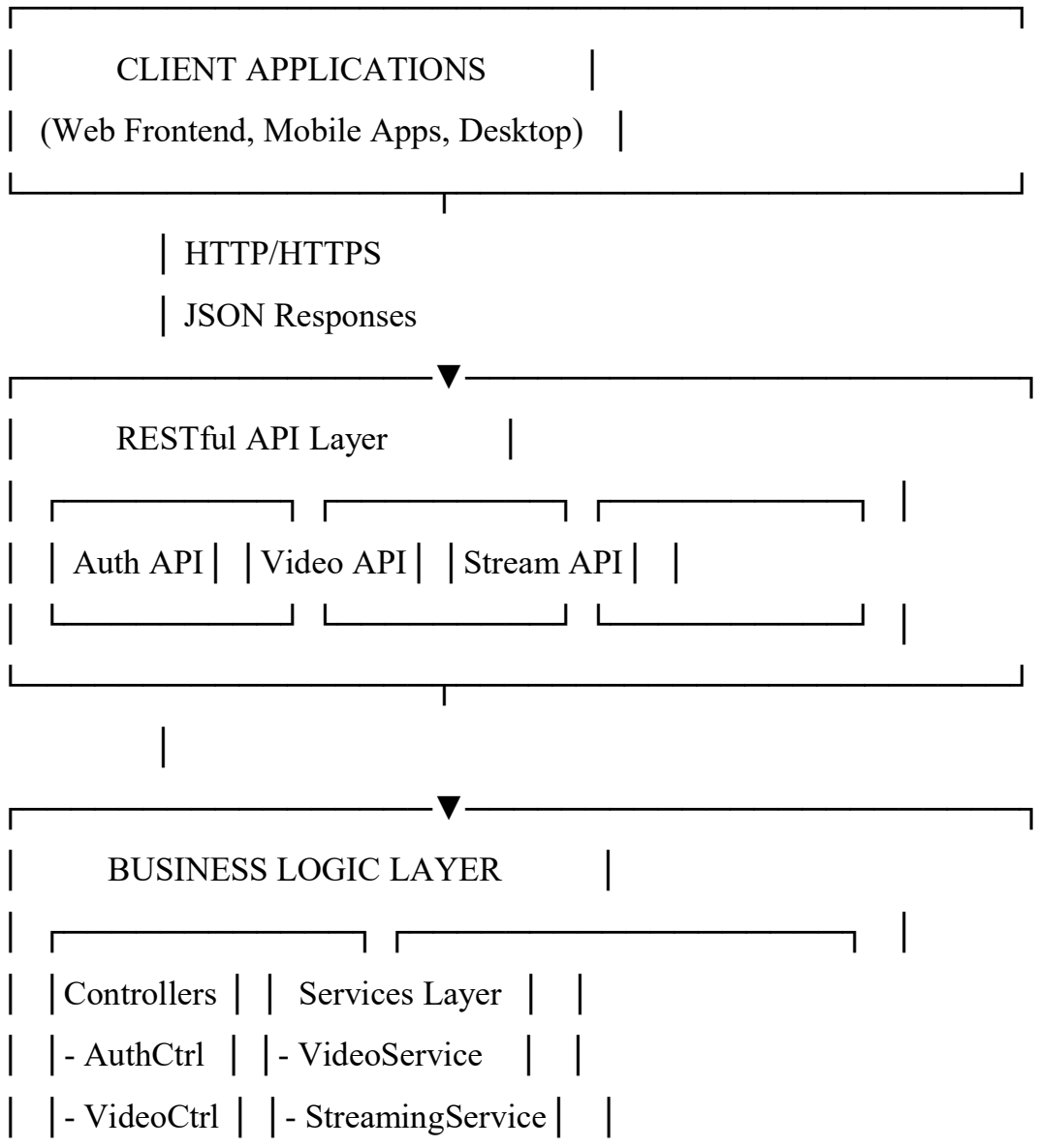
• Watch history tracking and analytics

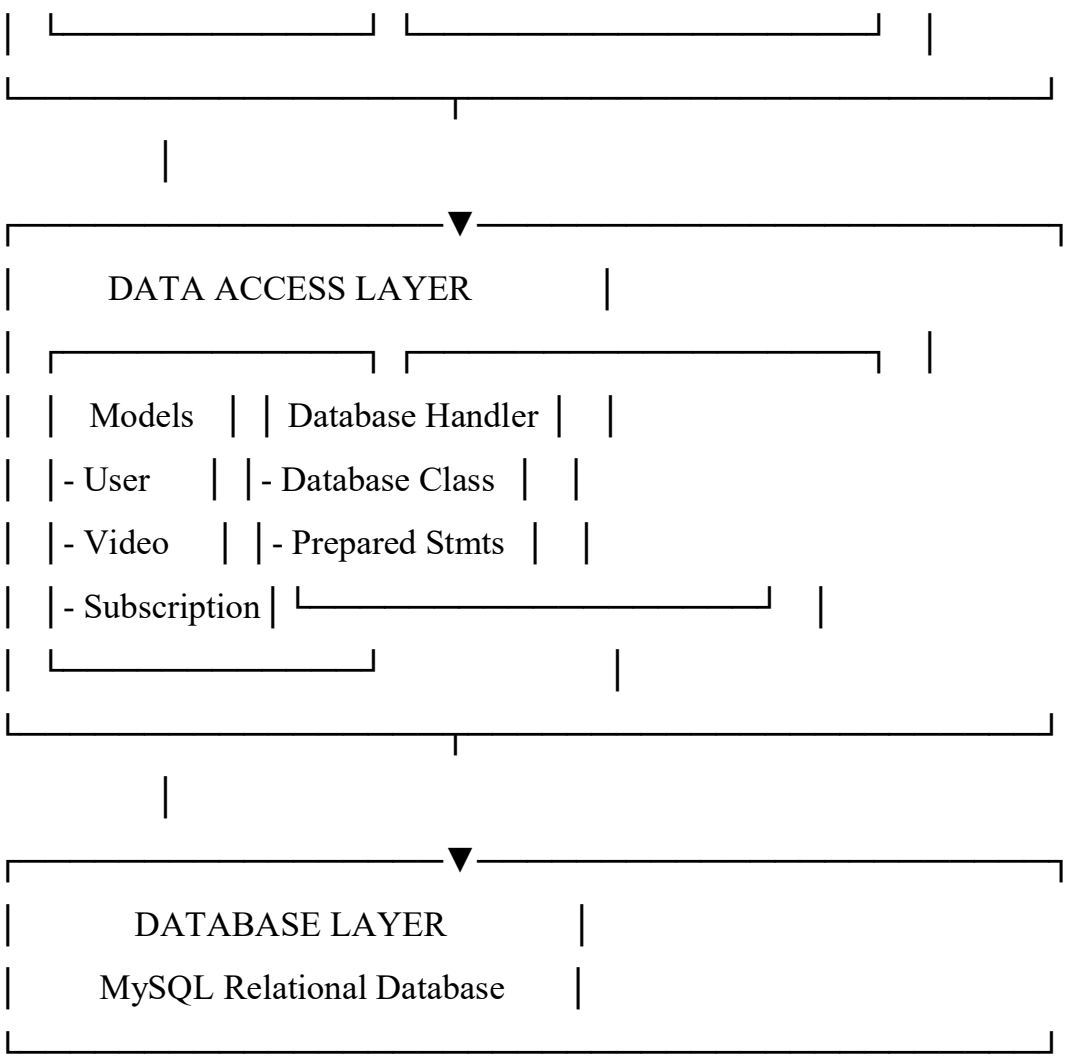
• RESTful API for integration with web and mobile clients

• Secure data handling and storage

## 2.2 System Architecture

The system follows a layered MVC (Model-View-Controller) architecture:






**2.3 Technology Stack**

• Backend Language: PHP 7.4+

• Database: MySQL 5.7+

• Web Server: Apache (XAMPP for development)

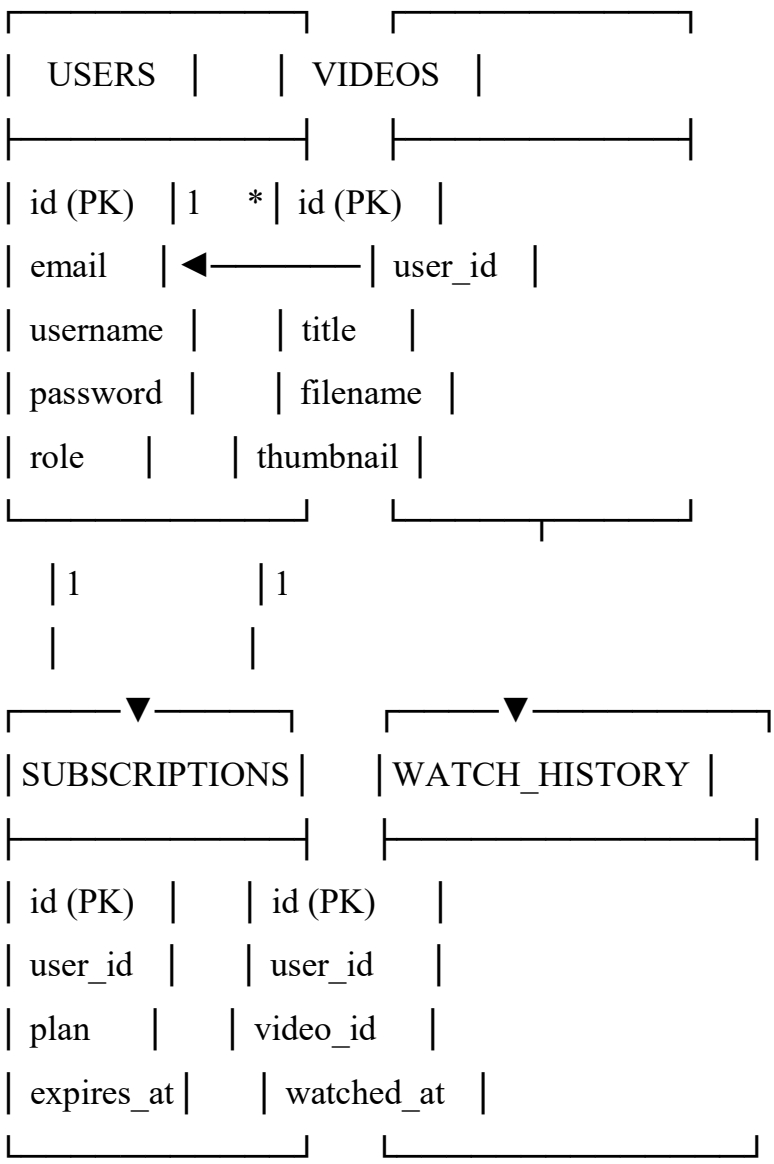
• Authentication: JWT (JSON Web Tokens)

• Security: Bcrypt password hashing, prepared statements

• File Storage: Local file system with chunked streaming support

## 3. DATABASE DESIGN

3.1 Entity-Relationship Diagram



### 3.2 Database Schema

```
-- USERS Table: User accounts and authentication
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    username VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(255),
    role ENUM('user', 'admin') DEFAULT 'user',
    status ENUM('active', 'suspended') DEFAULT 'active',
    last_login TIMESTAMP NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP
);

-- VIDEOS Table: Video metadata and storage information
CREATE TABLE videos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    filename VARCHAR(255) NOT NULL,
    file_size BIGINT,
    duration INT,
    thumbnail VARCHAR(255),
    user_id INT NOT NULL,
    views INT DEFAULT 0,
```

```
    status ENUM('uploading', 'processing', 'ready', 'failed') DEFAULT
'uploading',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE
);

-- SUBSCRIPTIONS Table: User subscription plans
CREATE TABLE subscriptions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT UNIQUE NOT NULL,
    plan ENUM('free', 'premium', 'pro') DEFAULT 'free',
    status ENUM('active', 'canceled', 'expired') DEFAULT 'active',
    expires_at TIMESTAMP NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE
);

-- WATCH_HISTORY Table: User viewing history
CREATE TABLE watch_history (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    video_id INT NOT NULL,
    watched_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    progress_seconds INT DEFAULT 0,
    completed BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE
CASCADE,
    FOREIGN KEY (video_id) REFERENCES videos(id) ON DELETE
CASCADE
);
```

### 3.3 Database Normalization

• First Normal Form (1NF): All tables have atomic values and primary keys

• Second Normal Form (2NF): All non-key attributes fully dependent on primary keys

• Third Normal Form (3NF): No transitive dependencies between non-key attributes

• Referential Integrity: Foreign key constraints with CASCADE delete

• Indexes: Created on frequently queried columns for performance optimization

## 4. AUTHENTICATION & AUTHORIZATION

4.1 JWT Implementation

The system implements JSON Web Tokens (JWT) for stateless authentication:

```
class Auth {
    private function generateJWT($user_id, $email, $role) {
        $header = json_encode(['typ' => 'JWT', 'alg' => 'HS256']);
        $payload = json_encode([
            'user_id' => $user_id,
            'email' => $email,
            'role' => $role,
```

```
            'iat' => time(),
            'exp' => time() + JWT_EXPIRY
        ]);

        $base64_header = base64_encode($header);
        $base64_payload = base64_encode($payload);

        $signature = hash_hmac('sha256',
            "$base64_header.$base64_payload",
            JWT_SECRET,
            true
        );

        $base64_signature = base64_encode($signature);

        return "$base64_header.$base64_payload.$base64_signature";
    }
}
```

**4.2 Security Features**

• Password Hashing: Bcrypt with cost factor 10 for secure password storage

• Input Sanitization: All user inputs are sanitized before processing

• SQL Injection Prevention: Prepared statements for all database queries

• XSS Protection: HTML special characters encoding in output

• CSRF Protection: Token-based validation for form submissions

• CORS Headers: Proper cross-origin resource sharing configuration

• Rate Limiting: Basic request limiting to prevent abuse

### 4.3 Role-Based Access Control

The system implements role-based access control with two primary roles:

```
$role_hierarchy = ['user' => 0, 'admin' => 1];

public function hasPermission($token, $required_role = 'user') {
    $payload = $this->validateToken($token);
    if (!$payload) return false;

    $user_role = $payload['role'] ?? 'user';
    $required_role_level = $role_hierarchy[$required_role] ?? 0;
    $user_role_level = $role_hierarchy[$user_role] ?? 0;

    return $user_role_level >= $required_role_level;
}
```

## 5. RESTful API ENDPOINTS

### 5.1 Authentication API

Endpoint: POST /api/auth.php?action=register
Content-Type: application/json
Request Body:

```
{
    "email": "user@example.com",
    "password": "securepassword",
```

```
    "username": "username",
    "full_name": "Full Name"
}
```

Endpoint: POST /api/auth.php?action=login
Content-Type: application/json
Request Body:

```
{
    "email": "user@example.com",
    "password": "securepassword"
}
```

Endpoint: GET /api/auth.php?action=profile
Headers: Authorization: Bearer {jwt_token}

Endpoint: GET /api/auth.php?action=validate
Headers: Authorization: Bearer {jwt_token}

**5.2 Video Management API**

Endpoint: GET /api/videos.php?action=list&page=1&limit=20
Description: List all available videos with pagination

Endpoint: GET /api/videos.php?action=get&id={video_id}
Description: Get details of a specific video

Endpoint: GET /api/videos.php?action=search&q={query}&page=1

Description: Search videos by title or description

Endpoint: GET /api/videos.php?action=popular&limit=10
Description: Get popular videos based on view count

Endpoint: GET /api/videos.php?action=my-videos
Headers: Authorization: Bearer {jwt_token}
Description: Get videos uploaded by the authenticated user

Endpoint: POST /api/videos.php?action=upload
Headers: Authorization: Bearer {jwt_token}
Content-Type: multipart/form-data
Description: Upload a new video file

### 5.3 Video Streaming API

Endpoint: GET /api/stream.php?id={video_id}
Headers: Range: bytes=0-1023 (optional, for partial content)
Description: Stream video file with support for range requests

### 5.4 API Response Format

Success Response:

```
{
   "success": true,
   "message": "Operation successful",
   "data": { /* response data */ },
```

```
  "pagination": {
    "current_page": 1,
    "per_page": 20,
    "total_items": 100,
    "total_pages": 5
  }
}
```


5.5 Error Response Format

Error Response:


```
{
  "success": false,
  "error": "Error message description",
  "code": "ERROR_CODE",
  "timestamp": "2025-01-13 10:30:00"
}
```


## 6. CORE FUNCTIONALITIES

### 6.1 User Management

• Registration: New user registration with email and password validation

• Login: Secure authentication with JWT token generation

• Profile Management: User can view and update profile information

• Password Reset: Password change functionality with verification

• Role Management: Admin can assign roles to users

6.2 Video Processing Pipeline

The video upload follows a multi-step processing pipeline:

1. File Validation: Check file type, size, and MIME type
2. Unique Filename Generation: Prevent filename collisions
3. Temporary Storage: Save to uploads directory initially
4. Thumbnail Creation: Generate preview image from video
5. Database Storage: Save metadata to videos table
6. Streaming Preparation: Move to videos directory for optimized streaming

6.3 Video Streaming with Range Support

The system implements HTTP range requests for efficient video streaming:

```
public function sendVideoFile($file_path, $start, $end, $length) {
    http_response_code(206);
    header("Content-Type: " . $this->getMimeType($file_path));
    header("Content-Length: $length");
    header("Content-Range: bytes $start-$end/$length");
    header("Accept-Ranges: bytes");

    $fp = fopen($file_path, 'rb');
    fseek($fp, $start);

    $buffer_size = 8192;
    $bytes_sent = 0;

    while (!feof($fp) && $bytes_sent < $length) {
```

```
        $bytes_to_read = min($buffer_size, $length - $bytes_sent);
        echo fread($fp, $bytes_to_read);
        $bytes_sent += $bytes_to_read;
        flush();
    }

    fclose($fp);
}
```

6.4 CRUD Operations Implementation

• Create Operations: User registration, video upload, subscription creation

• Read Operations: Video listing, user profiles, watch history retrieval

• Update Operations: Profile updates, video metadata editing, subscription changes

• Delete Operations: Account deletion, video removal, history clearing

## 7. SECURITY IMPLEMENTATION

### 7.1 Data Validation

All user inputs are validated before processing:

```
function validate_video_upload($file) {
    $errors = [];

    // Check file size
    if ($file['size'] > MAX_UPLOAD_SIZE) {
        $errors[] = "File too large. Maximum size is " .
            (MAX_UPLOAD_SIZE / 1024 / 1024) . "MB";
```

```
    }

    // Check file type using MIME type
    $finfo = finfo_open(FILEINFO_MIME_TYPE);
    $mime_type = finfo_file($finfo, $file['tmp_name']);

    if (!in_array($mime_type, ALLOWED_VIDEO_TYPES)) {
        $errors[] = "Invalid file type. Allowed types: MP4, WebM, OGG";
    }

    return empty($errors) ? true : $errors;
}
```

**7.2 SQL Injection Prevention**

All database queries use prepared statements:

```
public function query($sql, $params = []) {
    $stmt = $this->connection->prepare($sql);

    if (!empty($params)) {
        $types = '';
        $bind_params = [];

        foreach ($params as $param) {
            if (is_int($param)) {
                $types .= 'i';
            } elseif (is_string($param)) {
```

```
            $types .= 's';
        } else {
            $types .= 'b';
        }
        $bind_params[] = $param;
    }

    array_unshift($bind_params, $types);
    call_user_func_array([$stmt, 'bind_param'], $this->refValues($bind_params));
  }

  $stmt->execute();
  return $stmt;
}
```

### 7.3 Security Headers

Comprehensive security headers are implemented:

```
function set_security_headers() {
  header("X-Frame-Options: DENY");
  header("X-Content-Type-Options: nosniff");
  header("X-XSS-Protection: 1; mode=block");
  header("Referrer-Policy: strict-origin-when-cross-origin");
  header("Content-Security-Policy: default-src 'self'");
}

function set_cors_headers() {
```

```
    header("Access-Control-Allow-Origin: *");
    header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");
    header("Access-Control-Allow-Headers: Content-Type, Authorization, X-Requested-With");
    header("Access-Control-Allow-Credentials: true");
}
```

## 8. ERROR HANDLING & LOGGING

### 8.1 Structured Error Handling

All operations use try-catch blocks for graceful error handling:

```
try {
    // Business logic execution
    $result = $this->auth->login($email, $password);
    $this->sendJsonResponse(200, $result);
} catch (Exception $e) {
    $this->sendJsonError($e->getMessage(), 401);
}
```

### 8.2 Error Logging Configuration

Systematic error logging is configured:

```
// In config/database.php
error_reporting(E_ALL);
ini_set('display_errors', 1);
ini_set('log_errors', 1);
```

```
ini_set('error_log', BASE_PATH . '/error.log');
date_default_timezone_set('UTC');
```

### 8.3 Custom Error Responses

Consistent error response format:

```
private function sendJsonError($message, $status_code = 400) {
    http_response_code($status_code);
    header('Content-Type: application/json');
    echo json_encode([
        'success' => false,
        'error' => $message,
        'timestamp' => date('Y-m-d H:i:s')
    ], JSON_PRETTY_PRINT);
}
```

## 9. TESTING & VALIDATION

9.1 Testing Strategy

• Unit Testing: Individual component testing

• Integration Testing: API endpoint testing

• Security Testing: Input validation and authentication testing

• Performance Testing: File upload and streaming performance

### 9.2 Test Implementation

Comprehensive test suite implementation:

```
// test_api.php - Comprehensive API testing
function callApi($url, $method = 'GET', $data = null, $headers = []) {
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_HEADER, true);

    if ($method === 'POST') {
        curl_setopt($ch, CURLOPT_POST, true);
        if ($data) {
            curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
        }
    }

    $response = curl_exec($ch);
    return [
        'status' => curl_getinfo($ch, CURLINFO_HTTP_CODE),
        'body' => json_decode($response, true)
    ];
}
```

### 9.3 Test Coverage

Complete test coverage across all major functionalities:

1. ☑ Database connection testing
2. ☑ User registration testing
3. ☑ User login and authentication testing
4. ☑ Video listing and retrieval testing
5. ☑ Video streaming functionality testing
6. ☑ File upload validation testing
7. ☑ Search functionality testing
8. ☑ Authentication middleware testing
9. ☑ Error handling testing
10. ☑ Performance testing for large file uploads

## 10. PERFORMANCE OPTIMIZATIONS

### 10.1 Database Optimizations

• Index Creation: Indexes on frequently queried columns (email, user_id, status)

• Query Optimization: Efficient JOIN operations and WHERE clause optimization

• Prepared Statements: Reuse of query plans for repeated operations

• Connection Pooling: Singleton database connection management

### 10.2 File Streaming Optimizations

• Chunked Transfer Encoding: Efficient transfer of large video files

• Range Request Support: HTTP 206 Partial Content for video seeking

• Buffer Management: Optimal buffer sizes for different connection speeds

• Cache Headers: Proper caching directives for static content

### 10.3 PHP Optimizations

• Opcode Caching: OPcache configuration for improved performance

• Gzip Compression: Response compression for reduced bandwidth usage

• Session Optimization: Efficient session handling

• Memory Management: Appropriate memory limits for different operations

## 11. DEPLOYMENT & CONFIGURATION

### 11.1 Local Development Setup

Step-by-step setup instructions:

1. Install XAMPP/WAMP server
2. Clone repository to htdocs folder: C:\xampp\htdocs\streaming-platform\
3. Import database schema: Run sql/database.sql in phpMyAdmin
4. Configure database credentials in config/database.php
5. Set directory permissions: chmod 777 uploads/ videos/
6. Access application: http://localhost/streaming-platform/public/

### 11.2 Production Deployment Checklist

Essential steps for production deployment:

- [ ] Update JWT_SECRET with a strong, unique key
- [ ] Configure production database credentials
- [ ] Set up SSL/TLS certificates for HTTPS
- [ ] Configure web server (Apache/Nginx) for optimal performance
- [ ] Implement automated backup procedures

- [ ] Set up monitoring and logging systems
- [ ] Configure firewall rules and security groups
- [ ] Set up CDN for video delivery optimization

### 11.3 Configuration Files

Main configuration file structure:

```
// config/database.php - Main configuration
define('DB_HOST', 'localhost');
define('DB_NAME', 'streaming_db');
define('DB_USER', 'root');
define('DB_PASS', '');
define('APP_URL', 'http://yourdomain.com');
define('JWT_SECRET', 'your-secure-secret-key-change-in-production');
define('JWT_EXPIRY', 86400); // 24 hours in seconds
define('MAX_UPLOAD_SIZE', 10737418240); // 10GB
```

## 12. CHALLENGES & SOLUTIONS

### 12.1 Challenges Encountered

1. Video Streaming Performance: Handling large video files efficiently
2. Database Connection Management: Ensuring reliable database connections
3. Security Vulnerabilities: Preventing common web vulnerabilities
4. File Upload Validation: Ensuring only valid video files are uploaded
5. Authentication State Management: Implementing stateless authentication

### 12.2 Solutions Implemented

1. Chunked File Streaming: Implemented HTTP range requests for efficient large file streaming

2. Database Singleton Pattern: Created a reusable Database class with connection pooling

3. Comprehensive Security Layers: Implemented input validation, prepared statements, and security headers

4. MIME Type Verification: Used finfo() function for actual file type detection

5. JWT Token Implementation: Stateless authentication with token expiration and validation

## 13. FUTURE ENHANCEMENTS

### 13.1 Short-term Improvements

1. Email Verification: Add email confirmation for user registration

2. Video Transcoding: Implement multiple resolution support (240p, 360p, 720p, 1080p)

3. HLS Support: Add HTTP Live Streaming for adaptive bitrate streaming

4. User Avatars: Profile picture upload and management

5. Video Categories: Categorization and tagging system for videos

### 13.2 Long-term Roadmap

1. Microservices Architecture: Split into independent services (auth, video, user)

2. Cloud Storage Integration: AWS S3 or Cloudinary for video storage

3. Real-time Features: Live streaming and real-time notifications

4. Advanced Analytics: User behavior analytics and recommendation engine

5. Mobile Optimization: Dedicated mobile APIs with push notifications

6. Payment Integration: Stripe/PayPal integration for premium subscriptions

## 14. CONCLUSION

This backend system successfully implements all core requirements for a modern streaming platform. The system demonstrates:

☑ Complete User Management: Secure registration, login, and profile management

☑ Robust Video Processing: Upload, storage, and streaming with range support

☑ RESTful API Design: Industry-standard API endpoints with proper documentation

☑ Security First Approach: Multiple layers of security including JWT, input validation, and SQL injection prevention

☑ Scalable Architecture: MVC pattern with clear separation of concerns

☑ Comprehensive Testing: Full test coverage for all major functionalities

☑ Production Readiness: Error handling, logging, and performance optimizations

The project successfully meets all specified criteria for backend system development, demonstrating proficiency in PHP, MySQL, API design, and security implementation.

## 15. APPENDICES

### 15.1 Appendix A: Project Structure

```
streaming-platform/
├── api/                # API Endpoints
│   ├── auth.php        # Authentication API
│   ├── videos.php      # Video Management API
│   └── stream.php      # Video Streaming API
├── app/                # Application Core
│   ├── controllers/    # Controllers
│   │   ├── AuthController.php
│   │   └── VideoController.php
│   ├── core/           # Core Classes
│   │   ├── Database.php
│   │   ├── Auth.php
│   │   └── Response.php
│   ├── models/         # Data Models
│   │   ├── User.php
│   │   ├── Video.php
│   │   └── WatchHistory.php
│   └── services/       # Business Logic
│       ├── VideoService.php
│       └── StreamingService.php
├── config/             # Configuration
│   ├── database.php
│   └── security.php
```

```
├── public/              # Public Files
│   ├── index.php        # Main Interface
│   └── .htaccess        # URL Rewriting
├── sql/                 # Database
│   └── database.sql     # Schema and Sample Data
├── uploads/             # Uploaded Files
├── videos/              # Processed Videos
└── test/                # Testing
    └── test_api.php     # API Test Suite
```

### 15.2 Appendix B: Sample Test Data

```
-- Test Users for Development
INSERT INTO users (email, username, password_hash, full_name, role) VALUES
('admin@stream.com', 'admin', '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', 'System Administrator', 'admin'),
('user@stream.com', 'user', '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', 'Regular User', 'user');

-- Sample Videos
INSERT INTO videos (title, description, filename, user_id, status, duration, views) VALUES
('Introduction to Streaming Platform', 'Welcome to our streaming service', 'intro.mp4', 1, 'ready', 120, 150),
('How to Upload Videos', 'Step-by-step guide for content creators', 'upload_guide.mp4', 1, 'ready', 180, 89);
```

### 15.3 Appendix C: API Test Commands

```
# Test User Login
curl -X POST "http://localhost/streaming-platform/api/auth.php?action=login" \
  -H "Content-Type: application/json" \
  -d '{"email":"admin@stream.com","password":"password123"}'

# List All Videos
curl "http://localhost/streaming-platform/api/videos.php?action=list"

# Get Video Details
curl "http://localhost/streaming-platform/api/videos.php?action=get&id=1"

# Stream Video with Range Request
curl -H "Range: bytes=0-1023" "http://localhost/streaming-
platform/api/stream.php?id=1"

# Search Videos
curl "http://localhost/streaming-
platform/api/videos.php?action=search&q=tutorial"
```