PHP VA AJAX INTEGRATSIYASI:MUAMMOLAR VA YECHIMLAR

Jo'rayev To'xtasin Arabboy o'g'li

Andijon davlat universiteti, Axborot texnologiyalari kafedrasi o'qituvchisi.

olmalar1999@gmail.com

Annotatsiya

Bu maqolada PHP va AJAX texnologiyalarining integratsiyasida yuzaga keladigan asosiy muammolar va ularning yechim usullari tahlil qilingan. Zamonaviy veb-ilovalar yaratishda PHP backend va AJAX frontend texnologiyalarining samarali birgalikda ishlashi muhim ahamiyatga ega. Maqolada real vaqt rejimida ma'lumotlar almashinuvi, xavfsizlik masalalari, xatoliklarni boshqarish va ishlash samaradorligini oshirish usullari ko'rib chiqilgan. Amaliy misollarda jQuery va vanilla JavaScript yordamida AJAX so'rovlarini yuborish va PHP orqali javoblarni qayta ishlash jarayoni batafsil ko'rsatilgan.

Kalit so'zlar: PHP, AJAX, JavaScript, JSON, veb-dasturlash, asinxron so'rovlar, REST API, xavfsizlik, CSRF, CORS

Abstract

This article analyzes the main problems arising from the integration of PHP and AJAX technologies and their solutions. Effective collaboration between PHP backend and AJAX frontend technologies is crucial in creating modern web applications. The article examines real-time data exchange, security issues, error handling, and methods to improve performance efficiency. Practical examples demonstrate the detailed process of sending AJAX requests using jQuery and vanilla JavaScript and processing responses through PHP.

Keywords: PHP, AJAX, JavaScript, JSON, web development, asynchronous requests, REST API, security, CSRF, CORS

Аннотация

В данной статье анализируются основные проблемы, возникающие при интеграции технологий PHP и AJAX, и способы их решения. Эффективное

взаимодействие между серверными технологиями PHP и клиентскими технологиями AJAX имеет решающее значение при создании современных веб-приложений. В статье рассматриваются обмен данными в реальном времени, вопросы безопасности, обработка ошибок и методы повышения эффективности. Практические примеры демонстрируют подробный процесс отправки AJAX-запросов с использованием jQuery и vanilla JavaScript, а также обработки ответов через PHP.

Ключевые слова: PHP, AJAX, JavaScript, JSON, веб-разработка, асинхронные запросы, REST API, безопасность, CSRF, CORS

Kirish

Zamonaviy veb-dasturlash sohasida foydalanuvchi tajribasini yaxshilash va interaktiv interfeys yaratish uchun PHP va AJAX texnologiyalarining integratsiyasi muhim o'rin egallaydi. AJAX (Asynchronous JavaScript and XML) texnologiyasi sahifani qayta yuklash zaruriyatisiz server bilan ma'lumot almashish imkonini beradi, PHP esa kuchli backend ishlov berish va ma'lumotlar bazasi bilan ishlash imkoniyatlarini taqdim etadi.

Biroq, bu ikki texnologiyani birlashtirishda ko'plab muammolar va qiyinchiliklar yuzaga keladi. Ushbu maqolada ushbu muammolarni aniqlash va ularni yechish usullarini ko'rib chiqamiz.

AJAX va PHP haqida umumiy ma'lumot

AJAX texnologiyasining xususiyatlari

AJAX - bu veb-sahifalarni qayta yuklash zaruriyatisiz server bilan asinxron ma'lumot almashinuvini ta'minlaydigan texnologiya. AJAX quyidagi komponentlardan iborat:

- **XMLHttpRequest obyekti** - server bilan aloqa o'rnatish uchun
- **JavaScript** - foydalanuvchi interfeysi va mantiqni boshqarish uchun
- **DOM manipulyatsiyasi** - sahifani dinamik ravishda yangilash uchun
- **Ma'lumot formatlari** - JSON, XML yoki oddiy matn

PHP backend xususiyatlari

PHP (PHP: Hypertext Preprocessor) - bu server tomonida ishlaydigan skript tili bo'lib, quyidagi afzalliklarga ega:

- Ma'lumotlar bazasi bilan ishlash imkoniyatlari
- Kuchli xavfsizlik mexanizmlari
- Keng kutubxonalar to'plami
- Turli xil ma'lumot formatlarini qo'llab-quvvatlash

Asosiy muammolar va ularning tahlili

1. Xavfsizlik muammolari

CSRF (Cross-Site Request Forgery) hujumlari

CSRF hujumlari - bu noto'g'ri so'rovlar yuborilishiga olib keladigan xavfsizlik tahdididir. Bu muammo AJAX so'rovlarida token tekshiruvi yo'qligi tufayli yuzaga keladi.

Muammo misoli:

```
// Xavfsiz emas - CSRF token tekshiruvi yo'q
<?php
if ($_POST['action'] == 'delete_user') {
    $user_id = $_POST['user_id'];
    deleteUser($user_id); // Foydalanuvchini o'chirish
}
?>
```

Yechim:

```
// Xavfsiz - CSRF token bilan
<?php
session_start();
// CSRF token yaratish
function generateCSRFToken() {
    if (!isset($_SESSION['csrf_token'])) {
```

```
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
    }
    return $_SESSION['csrf_token'];
}
// CSRF token tekshiruvi
function validateCSRFToken($token) {
    return isset($_SESSION['csrf_token']) &&
           hash_equals($_SESSION['csrf_token'], $token);
}
if ($_POST['action'] == 'delete_user') {
    if (!validateCSRFToken($_POST['csrf_token'])) {
        http_response_code(403);
        echo json_encode(['error' => 'Invalid CSRF token']);
        exit;
    }

    $user_id = filter_var($_POST['user_id'], FILTER_VALIDATE_INT);
    if ($user_id) {
        deleteUser($user_id);
        echo json_encode(['success' => 'User deleted successfully']);
    }
}
?>
```

XSS (Cross-Site Scripting) hujumlari

XSS hujumlari foydalanuvchi kiritgan ma'lumotlarni to'g'ri tozalanmaganligi natijasida yuzaga keladi.

Yechim:

```
<?php
```

```
// Kiritilgan ma'lumotlarni tozalash
function sanitizeInput($data) {
    return htmlspecialchars(strip_tags(trim($data)), ENT_QUOTES, 'UTF-8');
}
// JSON javobni xavfsiz qaytarish
function sendJSONResponse($data) {
    header('Content-Type: application/json');
    echo json_encode($data, JSON_HEX_TAG | JSON_HEX_AMP | JSON_HEX_APOS | JSON_HEX_QUOT);
}
$username = sanitizeInput($_POST['username']);
$response = ['message' => 'Hello, ' . $username];
sendJSONResponse($response);
?>
```

2. Ma'lumot formatlashtirish muammolari

JSON encoding/decoding xatoliklari

PHP va JavaScript o'rtasida ma'lumot almashayotganda JSON format bilan bog'liq muammolar yuzaga kelishi mumkin.

Muammo misoli:

```
<?php
// Noto'g'ri - UTF-8 muammolari
$data = [
    'name' => 'Ахмед', // Kirill harflari
    'age' => 25
];
echo json_encode($data); // Bo'sh natija qaytarishi mumkin
?>
```

Yechim:

```
<?php
// To'g'ri - UTF-8 encoding bilan
function safeJsonEncode($data) {
    // Ma'lumotlarni UTF-8 ga aylantirish
    array_walk_recursive($data, function(&$item) {
        if (is_string($item)) {
            $item = mb_convert_encoding($item, 'UTF-8', 'auto');
        }
    });

    $json = json_encode($data, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_SLASHES);

    if (json_last_error() !== JSON_ERROR_NONE) {
        throw new Exception('JSON encoding error: ' . json_last_error_msg());
    }

    return $json;
}
try {
    $data = [
        'name' => 'Ахмед',
        'age' => 25,
        'status' => 'active'
    ];

    header('Content-Type: application/json; charset=UTF-8');
    echo safeJsonEncode($data);
```

```
} catch (Exception $e) {
    http_response_code(500);
    echo json_encode(['error' => $e->getMessage()]);
}
?>
```

3. Asinxron so'rovlar boshqaruvi

Callback hell muammosi

Ko'p miqdordagi asinxron so'rovlarni ketma-ket yuborishda kod murakkablashadi.

Muammo misoli:

```
// Callback hell - o'qish qiyin
$.ajax({
    url: 'get_user.php',
    success: function(user) {
        $.ajax({
            url: 'get_posts.php',
            data: {user_id: user.id},
            success: function(posts) {
                $.ajax({
                    url: 'get_comments.php',
                    data: {post_id: posts[0].id},
                    success: function(comments) {
                        // Ma'lumotlarni ko'rsatish
                        displayData(user, posts, comments);
                    }
                });
            }
        });
```

```
    }
});
```

Yechim - Async/Await bilan:

```
// Zamonaviy yondashuv - async/await
async function loadUserData(userId) {
    try {
        const user = await fetchData('get_user.php', {id: userId});
        const posts = await fetchData('get_posts.php', {user_id: user.id});
        const comments = await fetchData('get_comments.php', {post_id: posts[0].id});

        displayData(user, posts, comments);
    } catch (error) {
        console.error('Ma\'lumot yuklashda xatolik:', error);
        showErrorMessage('Ma\'lumotlarni yuklashda xatolik yuz berdi');
    }
}
// Umumiy fetch funksiyasi
async function fetchData(url, data = {}) {
    const response = await fetch(url, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
        },
        body: new URLSearchParams(data)
    });

    if (!response.ok) {
```

```
        throw new Error(`HTTP error! status: ${response.status}`);
    }

    return await response.json();
}
```

4. Ishlash samaradorligi muammolari

Database Connection Pool

Har bir AJAX so'rovi uchun alohida ma'lumotlar bazasi ulanishi ochish samaradorlikni pasaytiradi.

Yechim - Database Singleton Pattern:

```
<?php
class DatabaseConnection {
    private static $instance = null;
    private $connection;

    private function __construct() {
        try {
            $this->connection = new PDO(
                "mysql:host=localhost;dbname=mydb;charset=utf8mb4",
                $username,
                $password,
                [
                    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
                    PDO::ATTR_PERSISTENT => true, // Persistent connection
                ]
            );
```

```
            } catch (PDOException $e) {
                throw new Exception("Database connection failed: " . $e->getMessage());
            }
        }

        public static function getInstance() {
            if (self::$instance === null) {
                self::$instance = new self();
            }
            return self::$instance;
        }

        public function getConnection() {
            return $this->connection;
        }

        // Connection ni qayta ishlatishni oldini olish
        private function __clone() {}
        public function __wakeup() {
            throw new Exception("Cannot unserialize a singleton.");
        }
    }
    // Foydalanish
    function getUsers() {
        $db = DatabaseConnection::getInstance()->getConnection();
        $stmt = $db->prepare("SELECT id, name, email FROM users WHERE active = 1");
```

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

Response Caching

Tez-tez so'raladigan ma'lumotlarni kesh qilish orqali ishlash tezligini oshirish.

```
<?php
class ResponseCache {
    private $cacheDir = 'cache/';
    private $cacheTime = 300; // 5 daqiqa

    public function __construct($cacheDir = null, $cacheTime = null) {
        if ($cacheDir) $this->cacheDir = $cacheDir;
        if ($cacheTime) $this->cacheTime = $cacheTime;

        if (!is_dir($this->cacheDir)) {
            mkdir($this->cacheDir, 0755, true);
        }
    }

    public function get($key) {
        $filename = $this->cacheDir . md5($key) . '.cache';

        if (file_exists($filename) &&
            (time() - filemtime($filename)) < $this->cacheTime) {
            return json_decode(file_get_contents($filename), true);
        }
```

```
        return null;
    }

    public function set($key, $data) {
        $filename = $this->cacheDir . md5($key) . '.cache';
        file_put_contents($filename, json_encode($data));
    }

    public function delete($key) {
        $filename = $this->cacheDir . md5($key) . '.cache';
        if (file_exists($filename)) {
            unlink($filename);
        }
    }
}
// API endpoint misoli
$cache = new ResponseCache();
$cacheKey = 'users_' . md5($_GET['filter'] ?? '');
// Keshdan tekshirish
$users = $cache->get($cacheKey);
if ($users === null) {
    // Ma'lumotlar bazasidan olish
    $users = getUsersFromDatabase($_GET['filter'] ?? '');

    // Keshga saqlash
    $cache->set($cacheKey, $users);
}
header('Content-Type: application/json');
```

```
echo json_encode($users);
?>
```

Amaliy implementatsiya misollari

1. To'liq CRUD operatsiyalari

Frontend (JavaScript) qismi:

```
class UserManager {
    constructor(baseUrl) {
        this.baseUrl = baseUrl;
        this.csrfToken = this.getCSRFToken();
    }

    // CSRF token olish
    getCSRFToken() {
        return document.querySelector('meta[name="csrf-token"]')?.getAttribute('content');
    }

    // Barcha foydalanuvchilarni olish
    async getUsers() {
        try {
            const response = await fetch(`${this.baseUrl}/users.php?action=list`, {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json',
                }
            });

            const data = await this.handleResponse(response);
```

```
            this.displayUsers(data.users);
        } catch (error) {
            this.showError('Foydalanuvchilarni yuklashda xatolik: ' +
error.message);
        }
    }

    // Yangi foydalanuvchi qo'shish
    async addUser(userData) {
        try {
            const response = await fetch(`${this.baseUrl}/users.php`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body: new URLSearchParams({
                    action: 'create',
                    csrf_token: this.csrfToken,
                    ...userData
                })
            });

            const data = await this.handleResponse(response);
            this.showSuccess('Foydalanuvchi muvaffaqiyatli qo\'shildi');
            this.getUsers(); // Ro'yxatni yangilash
        } catch (error) {
            this.showError('Foydalanuvchi qo\'shishda xatolik: ' + error.message);
        }
```

```
}

// Foydalanuvchini yangilash
async updateUser(userId, userData) {
    try {
        const response = await fetch(`${this.baseUrl}/users.php`, {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRF-Token': this.csrfToken
            },
            body: JSON.stringify({
                action: 'update',
                id: userId,
                ...userData
            })
        });

        const data = await this.handleResponse(response);
        this.showSuccess('Foydalanuvchi ma\'lumotlari yangilandi');
        this.getUsers();
    } catch (error) {
        this.showError('Yangilashda xatolik: ' + error.message);
    }
}

// Foydalanuvchini o'chirish
async deleteUser(userId) {
```

```
    if (!confirm('Haqiqatan ham o\'chirmoqchimisiz?')) return;

    try {
        const response = await fetch(`${this.baseUrl}/users.php`, {
            method: 'DELETE',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRF-Token': this.csrfToken
            },
            body: JSON.stringify({
                action: 'delete',
                id: userId
            })
        });

        const data = await this.handleResponse(response);
        this.showSuccess('Foydalanuvchi o\'chirildi');
        this.getUsers();
    } catch (error) {
        this.showError('O\'chirishda xatolik: ' + error.message);
    }
}

// Response ni boshqarish
async handleResponse(response) {
    if (!response.ok) {
        throw new Error(`HTTP ${response.status}: ${response.statusText}`);
    }
```

```
    const contentType = response.headers.get('content-type');
    if (!contentType || !contentType.includes('application/json')) {
        throw new Error('Server JSON javob qaytarmadi');
    }

    const data = await response.json();

    if (data.error) {
        throw new Error(data.error);
    }

    return data;
}

// Foydalanuvchilarni ko'rsatish
displayUsers(users) {
    const container = document.getElementById('users-container');
    container.innerHTML = '';

    users.forEach(user => {
        const userDiv = document.createElement('div');
        userDiv.className = 'user-card';
        userDiv.innerHTML = `
            <h3>${this.escapeHtml(user.name)}</h3>
            <p>Email: ${this.escapeHtml(user.email)}</p>
            <p>Status: ${user.active ? 'Faol' : 'Nofaol'}</p>
```

```
                <button
onclick="userManager.editUser(${user.id})">Tahrirlash</button>
                <button
onclick="userManager.deleteUser(${user.id})">O'chirish</button>
            `;
            container.appendChild(userDiv);
        });
    }

    // XSS dan himoya
    escapeHtml(text) {
        const div = document.createElement('div');
        div.textContent = text;
        return div.innerHTML;
    }

    // Xabarlarni ko'rsatish
    showSuccess(message) {
        this.showMessage(message, 'success');
    }

    showError(message) {
        this.showMessage(message, 'error');
    }

    showMessage(message, type) {
        const messageDiv = document.createElement('div');
        messageDiv.className = `message ${type}`;
```

```
        messageDiv.textContent = message;

        document.body.appendChild(messageDiv);

        setTimeout(() => {
            messageDiv.remove();
        }, 5000);
    }
}
// Foydalanish
const userManager = new UserManager('/api');
userManager.getUsers();
```

Backend (PHP) qismi:

```
<?php
// users.php - RESTful API endpoint
require_once 'config.php';
require_once 'DatabaseConnection.php';
require_once 'ResponseCache.php';
class UserAPI {
    private $db;
    private $cache;

    public function __construct() {
        $this->db = DatabaseConnection::getInstance()->getConnection();
        $this->cache = new ResponseCache();

        // CORS sozlamalari
        $this->setCORSHeaders();
```

```
        // Content-Type ni tekshirish
        $this->validateContentType();
    }

    private function setCORSHeaders() {
        header("Access-Control-Allow-Origin: " . ALLOWED_ORIGIN);
        header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");
        header("Access-Control-Allow-Headers: Content-Type, X-CSRF-Token");
        header("Access-Control-Allow-Credentials: true");

        // Preflight OPTIONS so'rovini hal qilish
        if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
            http_response_code(200);
            exit;
        }
    }

    private function validateContentType() {
        $method = $_SERVER['REQUEST_METHOD'];
        if (in_array($method, ['POST', 'PUT']) &&
            !str_contains($_SERVER['CONTENT_TYPE'] ?? '', 'application/')) {
            $this->sendError('Noto\'g\'ri Content-Type', 400);
        }
    }
```

```
public function handleRequest() {
    try {
        session_start();

        $method = $_SERVER['REQUEST_METHOD'];

        switch ($method) {
            case 'GET':
                $this->handleGet();
                break;
            case 'POST':
                $this->handlePost();
                break;
            case 'PUT':
                $this->handlePut();
                break;
            case 'DELETE':
                $this->handleDelete();
                break;
            default:
                $this->sendError('Noma\'lum HTTP metodi', 405);
        }
    } catch (Exception $e) {
        error_log("API Error: " . $e->getMessage());
        $this->sendError('Server xatosi', 500);
    }
}
```

```
private function handleGet() {
    $action = $_GET['action'] ?? 'list';

    switch ($action) {
        case 'list':
            $this->getUsers();
            break;
        case 'get':
            $this->getUser($_GET['id'] ?? null);
            break;
        default:
            $this->sendError('Noma\'lum action', 400);
    }
}

private function handlePost() {
    $this->validateCSRFToken();

    $action = $_POST['action'] ?? null;

    switch ($action) {
        case 'create':
            $this->createUser();
            break;
        default:
            $this->sendError('Noma\'lum action', 400);
    }
}
```

```
private function handlePut() {
    $input = json_decode(file_get_contents('php://input'), true);

    if (json_last_error() !== JSON_ERROR_NONE) {
        $this->sendError('Noto\'g\'ri JSON format', 400);
    }

    $this->validateCSRFToken($input['csrf_token'] ?? '');

    $action = $input['action'] ?? null;

    switch ($action) {
        case 'update':
            $this->updateUser($input);
            break;
        default:
            $this->sendError('Noma\'lum action', 400);
    }
}

private function handleDelete() {
    $input = json_decode(file_get_contents('php://input'), true);
    $this->validateCSRFToken($input['csrf_token'] ?? '');

    $action = $input['action'] ?? null;

    switch ($action) {
```

```
            case 'delete':
                $this->deleteUser($input['id'] ?? null);
                break;
            default:
                $this->sendError('Noma\'lum action', 400);
        }
    }

    private function getUsers() {
        $cacheKey = 'users_list';

        // Keshdan tekshirish
        $users = $this->cache->get($cacheKey);

        if ($users === null) {
            $stmt = $this->db->prepare("
                SELECT id, name, email, active, created_at
                FROM users
                ORDER BY created_at DESC
            ");
            $stmt->execute();
            $users = $stmt->fetchAll();

            // Keshga saqlash
            $this->cache->set($cacheKey, $users);
        }

        $this->sendSuccess(['users' => $users]);
```

```
    }

    private function getUser($id) {
        $id = filter_var($id, FILTER_VALIDATE_INT);
        if (!$id) {
            $this->sendError('Noto\'g\'ri foydalanuvchi ID', 400);
        }

        $stmt = $this->db->prepare("
            SELECT id, name, email, active, created_at
            FROM users
            WHERE id = ?
        ");
        $stmt->execute([$id]);
        $user = $stmt->fetch();

        if (!$user) {
            $this->sendError('Foydalanuvchi topilmadi', 404);
        }

        $this->sendSuccess(['user' => $user]);
    }

    private function createUser() {
        $name = $this->sanitizeInput($_POST['name'] ?? '');
        $email = filter_var($_POST['email'] ?? '', FILTER_VALIDATE_EMAIL);
        $password = $_POST['password'] ?? '';
```

```
        // Validatsiya
        if (empty($name) || !$email || empty($password)) {
            $this->sendError('Barcha maydonlar to\'ldirilishi shart', 400);
        }

        if (strlen($password) < 8) {
            $this->sendError('Parol kamida 8 ta belgidan iborat bo\'lishi kerak', 400);
        }

        // Email mavjudligini tekshirish
        $stmt = $this->db->prepare("SELECT id FROM users WHERE email = ?");
        $stmt->execute([$email]);
        if ($stmt->fetch()) {
            $this->sendError('Bu email allaqachon ro\'yxatdan o\'tgan', 409);
        }

        // Parolni hash qilish
        $hashedPassword = password_hash($password, PASSWORD_DEFAULT);

        // Yangi foydalanuvchi yaratish
        $stmt = $this->db->prepare("
            INSERT INTO users (name, email, password, active, created_at)
            VALUES (?, ?, ?, 1, NOW())
        ");
```

```
    if ($stmt->execute([$name, $email, $hashedPassword])) {
        // Keshni tozalash
        $this->cache->delete('users_list');

        $this->sendSuccess([
            'message' => 'Foydalanuvchi muvaffaqiyatli yaratildi',
            'user_id' => $this->db->lastInsertId()
        ]);
    } else {
        $this->sendError('Foydalanuvchi yaratishda xatolik', 500);
    }
}

private function updateUser($data) {
    $id = filter_var($data['id'] ?? null, FILTER_VALIDATE_INT);
    if (!$id) {
        $this->sendError('Noto\'g\'ri foydalanuvchi ID', 400);
    }

    $name = $this->sanitizeInput($data['name'] ?? '');
    $email = filter_var($data['email'] ?? '', FILTER_VALIDATE_EMAIL);

    if (empty($name) || !$email) {
        $this->sendError('Nomi va email to\'ldirilishi shart', 400);
    }

    // Email uniqueness ni tekshirish
```

```
        $stmt = $this->db->prepare("SELECT id FROM users WHERE email = ? AND id != ?");
        $stmt->execute([$email, $id]);
        if ($stmt->fetch()) {
            $this->sendError('Bu email boshqa foydalanuvchi tomonidan ishlatilmoqda', 409);
        }

        // Yangilash
        $stmt = $this->db->prepare("
            UPDATE users
            SET name = ?, email = ?, updated_at = NOW()
            WHERE id = ?
        ");

        if ($stmt->execute([$name, $email, $id])) {
            // Keshni tozalash
            $this->cache->delete('users_list');

            $this->sendSuccess(['message' => 'Foydalanuvchi ma\'lumotlari yangilandi']);
        } else {
            $this->sendError('Yangilashda xatolik', 500);
        }
    }

    private function deleteUser($id) {
        $id = filter_var($id, FILTER_VALIDATE_INT);
```

```
        if (!$id) {
            $this->sendError('Noto\'g\'ri foydalanuvchi ID', 400);
        }

        // Foydalanuvchi mavjudligini tekshirish
        $stmt = $this->db->prepare("SELECT id FROM users WHERE id = ?");
        $stmt->execute([$id]);
        if (!$stmt->fetch()) {
            $this->sendError('Foydalanuvchi topilmadi', 404);
        }

        // Soft delete (active = 0)
        $stmt = $this->db->prepare("UPDATE users SET active = 0, deleted_at =
NOW() WHERE id = ?");

        if ($stmt->execute([$id])) {
            // Keshni tozalash
            $this->cache->delete('users_list');

            $this->sendSuccess(['message' => 'Foydalanuvchi o\'chirildi']);
        } else {
            $this->sendError('O\'chirishda xatolik', 500);
        }
    }

    private function validateCSRFToken($token = null) {
        if ($token === null) {
```

```
            $token = $_POST['csrf_token'] ?? $_SERVER['HTTP_X_CSRF_TOKEN'] ?? '';
        }

        if (!isset($_SESSION['csrf_token']) ||
            !hash_equals($_SESSION['csrf_token'], $token)) {
            $this->sendError('CSRF token noto\'g\'ri', 403);
        }
    }

    private function sanitizeInput($data) {
        return htmlspecialchars(strip_tags(trim($data)), ENT_QUOTES, 'UTF-8');
    }

    private function sendSuccess($data) {
        header('Content-Type: application/json; charset=UTF-8');
        echo json_encode(['success' => true] + $data, JSON_UNESCAPED_UNICODE);
        exit;
    }

    private function sendError($message, $code = 400) {
        http_response_code($code);
        header('Content-Type: application/json; charset=UTF-8');
        echo json_encode([
            'success' => false,
            'error' => $message
```

```
        ], JSON_UNESCAPED_UNICODE);
        exit;
    }
}
// API ni ishga tushirish
$api = new UserAPI();
$api->handleRequest();
?>
```

2. Real-time ma'lumotlar yangilanishi

Server-Sent Events (SSE) bilan:

PHP qismi (events.php):

```
<?php
// events.php - Server-Sent Events endpoint
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');
header('Connection: keep-alive');
header('Access-Control-Allow-Origin: *');
// Output buffering ni o'chirish
if (ob_get_level()) {
    ob_end_clean();
}
// Ma'lumotlar bazasiga ulanish
require_once 'DatabaseConnection.php';
$db = DatabaseConnection::getInstance()->getConnection();
// Oxirgi yangilanish vaqtini olish
$lastUpdate = $_GET['lastupdate'] ?? 0;
function sendSSEMessage($data, $event = 'message', $id = null) {
    if ($id !== null) {
```

```
            echo "id: $id\n";
        }
        echo "event: $event\n";
        echo "data: " . json_encode($data) . "\n\n";

        // Browserga darhol yuborish
        if (ob_get_level()) {
            ob_flush();
        }
        flush();
    }
    // Connection holatini tekshirish
    function isConnectionAlive() {
        return connection_status() === CONNECTION_NORMAL;
    }
    // Asosiy loop
    $startTime = time();
    $maxExecutionTime = 30; // 30 soniya
    while (isConnectionAlive() && (time() - $startTime) < $maxExecutionTime)
{
        try {
            // Yangi ma'lumotlarni tekshirish
            $stmt = $db->prepare("
                SELECT id, name, email, created_at,
                    UNIX_TIMESTAMP(created_at) as timestamp
                FROM users
                WHERE UNIX_TIMESTAMP(created_at) > ?
                ORDER BY created_at DESC
```

```
");
$stmt->execute([$lastUpdate]);
$newUsers = $stmt->fetchAll();

// Agar yangi ma'lumotlar bo'lsa
if (!empty($newUsers)) {
    sendSSEMessage([
        'type' => 'new_users',
        'users' => $newUsers,
        'count' => count($newUsers)
    ], 'user_update');

    // Oxirgi yangilanish vaqtini o'zgartirish
    $lastUpdate = max(array_column($newUsers, 'timestamp'));
}

// Online foydalanuvchilar sonini yuborish
$stmt = $db->prepare("
    SELECT COUNT(*) as online_count
    FROM user_sessions
    WHERE last_activity > DATE_SUB(NOW(), INTERVAL 5
MINUTE)
");
$stmt->execute();
$onlineData = $stmt->fetch();

sendSSEMessage([
    'online_count' => $onlineData['online_count'],
```

```
            'timestamp' => time()
        ], 'online_status');

        // Heartbeat yuborish
        sendSSEMessage(['status' => 'alive'], 'heartbeat', time());

    } catch (Exception $e) {
        error_log("SSE Error: " . $e->getMessage());
        sendSSEMessage(['error' => 'Server xatosi'], 'error');
        break;
    }

    // 2 soniya kutish
    sleep(2);
}
// Connection ni yopish
sendSSEMessage(['status' => 'disconnected'], 'disconnect');
?>
```

JavaScript qismi:

```
class RealTimeUpdater {
    constructor(eventUrl) {
        this.eventUrl = eventUrl;
        this.eventSource = null;
        this.reconnectDelay = 1000; // 1 soniya
        this.maxReconnectDelay = 30000; // 30 soniya
        this.reconnectAttempts = 0;
        this.lastEventId = 0;
    }
```

```
// SSE connection ni boshlash
start() {
    this.connect();

    // Sahifa yopilayotganda connection ni yopish
    window.addEventListener('beforeunload', () => {
        this.stop();
    });

    // Network holatini kuzatish
    window.addEventListener('online', () => {
        console.log('Network online - reconnecting...');
        this.connect();
    });

    window.addEventListener('offline', () => {
        console.log('Network offline');
        this.stop();
    });
}

// SSE connection o'rnatish
connect() {
    if (this.eventSource) {
        this.eventSource.close();
    }
```

```
const url = new URL(this.eventUrl);
if (this.lastEventId > 0) {
    url.searchParams.set('lastupdate', this.lastEventId);
}

this.eventSource = new EventSource(url.toString());

this.eventSource.onopen = (event) => {
    console.log('SSE connection opened');
    this.reconnectAttempts = 0;
    this.reconnectDelay = 1000;
    this.updateConnectionStatus('connected');
};

this.eventSource.onmessage = (event) => {
    try {
        const data = JSON.parse(event.data);
        this.handleMessage(data);
    } catch (error) {
        console.error('SSE message parsing error:', error);
    }
};

this.eventSource.onerror = (event) => {
    console.error('SSE connection error:', event);
    this.eventSource.close();
    this.updateConnectionStatus('disconnected');
    this.scheduleReconnect();
```

```
};

// Maxsus event handlerlar
this.eventSource.addEventListener('user_update', (event) => {
    try {
        const data = JSON.parse(event.data);
        this.handleUserUpdate(data);
        this.lastEventId = event.lastEventId || Date.now();
    } catch (error) {
        console.error('User update handling error:', error);
    }
});

this.eventSource.addEventListener('online_status', (event) => {
    try {
        const data = JSON.parse(event.data);
        this.updateOnlineStatus(data.online_count);
    } catch (error) {
        console.error('Online status handling error:', error);
    }
});

this.eventSource.addEventListener('heartbeat', (event) => {
    // Connection hayotiyligini tekshirish
    this.lastHeartbeat = Date.now();
});

this.eventSource.addEventListener('error', (event) => {
```

```
            try {
                const data = JSON.parse(event.data);
                this.showError('Server xatosi: ' + data.error);
            } catch (error) {
                this.showError('Noma\'lum server xatosi');
            }
        });
    }

    // Qayta ulanishni rejalashtirish
    scheduleReconnect() {
        this.reconnectAttempts++;

        // Exponential backoff
        const delay = Math.min(
            this.reconnectDelay * Math.pow(2, this.reconnectAttempts - 1),
            this.maxReconnectDelay
        );

        console.log(`Reconnecting in ${delay}ms (attempt ${this.reconnectAttempts})`);

        setTimeout(() => {
            if (!this.eventSource || this.eventSource.readyState === EventSource.CLOSED) {
                this.connect();
            }
        }, delay);
```

```
}

// Connection ni to'xtatish
stop() {
    if (this.eventSource) {
        this.eventSource.close();
        this.eventSource = null;
    }
    this.updateConnectionStatus('disconnected');
}

// Yangi foydalanuvchilar yangilanishini boshqarish
handleUserUpdate(data) {
    if (data.type === 'new_users' && data.users.length > 0) {
        // Yangi foydalanuvchilarni ro'yxatga qo'shish
        this.addNewUsersToList(data.users);

        // Notification ko'rsatish
        this.showNotification(
            `${data.count} ta yangi foydalanuvchi ro'yxatdan o'tdi`,
            'info'
        );

        // Real-time counter yangilash
        this.updateUserCount();
    }
}
```

```
// Yangi foydalanuvchilarni ro'yxatga qo'shish
addNewUsersToList(users) {
    const container = document.getElementById('users-container');
    if (!container) return;

    users.forEach(user => {
        // Mavjud foydalanuvchini tekshirish
        if (document.getElementById(`user-${user.id}`)) return;

        const userElement = document.createElement('div');
        userElement.id = `user-${user.id}`;
        userElement.className = 'user-card new-user';
        userElement.innerHTML = `
            <h3>${this.escapeHtml(user.name)}</h3>
            <p>Email: ${this.escapeHtml(user.email)}</p>
            <p>Qo'shilgan: ${this.formatDate(user.created_at)}</p>
            <span class="new-badge">Yangi</span>
        `;

        // Ro'yxat boshiga qo'shish
        container.insertBefore(userElement, container.firstChild);

        // Animation qo'shish
        setTimeout(() => {
            userElement.classList.add('fade-in');
        }, 100);

        // "Yangi" belgisini olib tashlash
```

```
            setTimeout(() => {
                userElement.classList.remove('new-user');
                const badge = userElement.querySelector('.new-badge');
                if (badge) badge.remove();
            }, 5000);
        });
    }

    // Online foydalanuvchilar sonini yangilash
    updateOnlineStatus(count) {
        const statusElement = document.getElementById('online-status');
        if (statusElement) {
            statusElement.textContent = `Online: ${count}`;
            statusElement.className = count > 0 ? 'online' : 'offline';
        }
    }

    // Connection holatini yangilash
    updateConnectionStatus(status) {
        const statusElement = document.getElementById('connection-status');
        if (statusElement) {
            statusElement.textContent = status === 'connected' ? 'Ulangan' : 'Uzilib
qolgan';
            statusElement.className = `connection-status ${status}`;
        }
    }

    // Foydalanuvchilar sonini yangilash
```

```
updateUserCount() {
    // Bu yerda foydalanuvchilar sonini hisoblash va ko'rsatish
    const userCards = document.querySelectorAll('.user-card').length;
    const countElement = document.getElementById('user-count');
    if (countElement) {
        countElement.textContent = `Jami: ${userCards}`;
    }
}

// Notification ko'rsatish
showNotification(message, type = 'info') {
    // Browser notification
    if ('Notification' in window && Notification.permission === 'granted') {
        new Notification('Yangi faoliyat', {
            body: message,
            icon: '/images/notification-icon.png'
        });
    }

    // In-app notification
    const notification = document.createElement('div');
    notification.className = `notification ${type}`;
    notification.innerHTML = `
        <span class="message">${this.escapeHtml(message)}</span>
        <button class="close-btn">&times;</button>
    `;

    document.body.appendChild(notification);
```

```
    // Close button event
    notification.querySelector('.close-btn').onclick = () => {
        notification.remove();
    };

    // Auto remove
    setTimeout(() => {
        if (notification.parentNode) {
            notification.remove();
        }
    }, 5000);
}

// Xato xabarini ko'rsatish
showError(message) {
    this.showNotification(message, 'error');
}

// HTML dan qochish
escapeHtml(text) {
    const div = document.createElement('div');
    div.textContent = text;
    return div.innerHTML;
}

// Sanani formatlash
formatDate(dateString) {
```

```
        const date = new Date(dateString);
        return date.toLocaleString('uz-UZ');
    }

    // Umumiy message handler
    handleMessage(data) {
        console.log('SSE message received:', data);
    }
}
// Foydalanish
document.addEventListener('DOMContentLoaded', () => {
    // Notification ruxsatini so'rash
    if ('Notification' in window && Notification.permission === 'default') {
        Notification.requestPermission();
    }

    // Real-time updater ni boshlash
    const updater = new RealTimeUpdater('/api/events.php');
    updater.start();

    // Global o'zgaruvchi sifatida saqlash
    window.realTimeUpdater = updater;
});
```

Xatoliklarni boshqarish va debug qilish

1. Keng tarqalgan xatoliklar va ularning yechimlari

JSON parsing xatoliklari:

```
<?php
// Xavfsiz JSON parsing
```

```
function safeJsonDecode($json, $assoc = true) {
    $data = json_decode($json, $assoc);

    if (json_last_error() !== JSON_ERROR_NONE) {
        throw new InvalidArgumentException(
            'JSON parsing xatosi: ' . json_last_error_msg()
        );
    }

    return $data;
}
// Input validatsiya
function validateInput($data, $rules) {
    $errors = [];

    foreach ($rules as $field => $rule) {
        $value = $data[$field] ?? null;

        // Required tekshiruvi
        if (isset($rule['required']) && $rule['required'] && empty($value)) {
            $errors[$field] = "$field maydoni to'ldirilishi shart";
            continue;
        }

        // Type tekshiruvi
        if (!empty($value) && isset($rule['type'])) {
            switch ($rule['type']) {
                case 'email':
```

```
                    if (!filter_var($value, FILTER_VALIDATE_EMAIL)) {
                        $errors[$field] = "$field noto'g'ri email formati";
                    }
                    break;
                case 'int':
                    if (!filter_var($value, FILTER_VALIDATE_INT)) {
                        $errors[$field] = "$field butun son bo'lishi kerak";
                    }
                    break;
                case 'string':
                    if (!is_string($value)) {
                        $errors[$field] = "$field matn bo'lishi kerak";
                    }
                    break;
            }
        }

        // Length tekshiruvi
        if (!empty($value) && isset($rule['min_length'])) {
            if (strlen($value) < $rule['min_length']) {
                $errors[$field] = "$field kamida {$rule['min_length']} ta belgidan iborat bo'lishi kerak";
            }
        }

        if (!empty($value) && isset($rule['max_length'])) {
            if (strlen($value) > $rule['max_length']) {
```

```
                $errors[$field] = "$field maksimal {$rule['max_length']} ta belgidan iborat bo'lishi kerak";
            }
        }
    }

    return $errors;
}
// Foydalanish misoli
try {
    $input = safeJsonDecode(file_get_contents('php://input'));

    $rules = [
        'name' => ['required' => true, 'type' => 'string', 'min_length' => 2, 'max_length' => 50],
        'email' => ['required' => true, 'type' => 'email'],
        'age' => ['type' => 'int', 'min' => 18, 'max' => 120]
    ];

    $errors = validateInput($input, $rules);

    if (!empty($errors)) {
        http_response_code(422);
        echo json_encode(['errors' => $errors]);
        exit;
    }

    // Ma'lumotlarni qayta ishlash...
```

```
} catch (InvalidArgumentException $e) {
    http_response_code(400);
    echo json_encode(['error' => $e->getMessage()]);
} catch (Exception $e) {
    error_log("Validation error: " . $e->getMessage());
    http_response_code(500);
    echo json_encode(['error' => 'Server xatosi']);
}
?>
```

2. Logging va monitoring tizimi

```
<?php
class APILogger {
    private $logFile;
    private $errorLogFile;

    public function __construct($logDir = 'logs/') {
        if (!is_dir($logDir)) {
            mkdir($logDir, 0755, true);
        }

        $this->logFile = $logDir . 'api_' . date('Y-m-d') . '.log';
        $this->errorLogFile = $logDir . 'errors_' . date('Y-m-d') . '.log';
    }

    public function logRequest($method, $endpoint, $data = [], $response = null,
$duration = 0) {
        $logEntry = [
```

```
            'timestamp' => date('Y-m-d H:i:s'),
            'method' => $method,
            'endpoint' => $endpoint,
            'ip' => $_SERVER['REMOTE_ADDR'] ?? 'unknown',
            'user_agent' => $_SERVER['HTTP_USER_AGENT'] ?? 'unknown',
            'data' => $data,
            'response_code' => http_response_code(),
            'duration_ms' => $duration,
            'memory_usage' => memory_get_peak_usage(true)
        ];

        if ($response) {
            $logEntry['response'] = $response;
        }

        file_put_contents($this->logFile, json_encode($logEntry) . "\n",
FILE_APPEND | LOCK_EX);
    }

    public function logError($message, $context = []) {
        $errorEntry = [
            'timestamp' => date('Y-m-d H:i:s'),
            'message' => $message,
            'context' => $context,
            'trace' =>
debug_backtrace(DEBUG_BACKTRACE_IGNORE_ARGS, 5)
        ];
```

```
        file_put_contents($this->errorLogFile, json_encode($errorEntry) . "\n",
FILE_APPEND | LOCK_EX);
    }

    public function getStatistics($date = null) {
        if (!$date) $date = date('Y-m-d');

        $logFile = str_replace(date('Y-m-d'), $date, $this->logFile);

        if (!file_exists($logFile)) {
            return null;
        }

        $logs = file($logFile, FILE_IGNORE_NEW_LINES |
FILE_SKIP_EMPTY_LINES);
        $stats = [
            'total_requests' => 0,
            'methods' => [],
            'response_codes' => [],
            'avg_duration' => 0,
            'errors' => 0
        ];

        $totalDuration = 0;

        foreach ($logs as $line) {
            $entry = json_decode($line, true);
            if (!$entry) continue;
```

```
            $stats['total_requests']++;

            // Method statistics
            $method = $entry['method'] ?? 'unknown';
            $stats['methods'][$method] = ($stats['methods'][$method] ?? 0) + 1;

            // Response code statistics
            $code = $entry['response_code'] ?? 0;
            $stats['response_codes'][$code] = ($stats['response_codes'][$code] ??
0) + 1;

            // Duration
            $duration = $entry['duration_ms'] ?? 0;
            $totalDuration += $duration;

            // Errors
            if ($code >= 400) {
                $stats['errors']++;
            }
        }

        if ($stats['total_requests'] > 0) {
            $stats['avg_duration'] = $totalDuration / $stats['total_requests'];
        }

        return $stats;
    }
```

```
}
// Middleware sifatida foydalanish
class APIMiddleware {
    private $logger;
    private $startTime;

    public function __construct() {
        $this->logger = new APILogger();
        $this->startTime = microtime(true);
    }

    public function before() {
        // Request boshlanishida
        register_shutdown_function([$this, 'after']);

        // Error handler o'rnatish
        set_error_handler([$this, 'errorHandler']);
        set_exception_handler([$this, 'exceptionHandler']);
    }

    public function after() {
        $duration = (microtime(true) - $this->startTime) * 1000; // milliseconds

        $this->logger->logRequest(
            $_SERVER['REQUEST_METHOD'],
            $_SERVER['REQUEST_URI'],
            $_REQUEST,
            null,
```

```
            $duration
        );
    }

    public function errorHandler($severity, $message, $file, $line) {
        $this->logger->logError("PHP Error: $message", [
            'severity' => $severity,
            'file' => $file,
            'line' => $line
        ]);

        return false; // PHP standart error handler ham ishlashi uchun
    }

    public function exceptionHandler($exception) {
        $this->logger->logError("Uncaught Exception: " . $exception->getMessage(), [
            'file' => $exception->getFile(),
            'line' => $exception->getLine(),
            'trace' => $exception->getTraceAsString()
        ]);

        http_response_code(500);
        echo json_encode(['error' => 'Server xatosi']);
    }
}
// API da foydalanish
$middleware = new APIMiddleware();
```

```
$middleware->before();
// API logic...
?>
```

Ishlash samaradorligini optimallashtirish

1. Database query optimizatsiyasi

```
<?php
class OptimizedUserAPI {
    private $db;

    public function __construct() {
        $this->db = DatabaseConnection::getInstance()->getConnection();
    }

    // Pagination bilan samarali data loading
    public function getUsersPaginated($page = 1, $limit = 20, $filters = []) {
        $offset = ($page - 1) * $limit;

        // Base query
        $whereConditions = ['active = 1'];
        $params = [];

        // Filters qo'shish
        if (!empty($filters['search'])) {
            $whereConditions[] = '(name LIKE ? OR email LIKE ?)';
            $searchTerm = '%' . $filters['search'] . '%';
            $params[] = $searchTerm;
            $params[] = $searchTerm;
        }
```

```
        if (!empty($filters['created_after'])) {
            $whereConditions[] = 'created_at >= ?';
            $params[] = $filters['created_after'];
        }

        $whereClause = implode(' AND ', $whereConditions);

        // Count query (pagination uchun)
        $countSql = "SELECT COUNT(*) as total FROM users WHERE $whereClause";
        $countStmt = $this->db->prepare($countSql);
        $countStmt->execute($params);
        $totalUsers = $countStmt->fetch()['total'];

        // Main query
        $sql = "
            SELECT id, name, email, created_at,
                CASE WHEN last_login > DATE_SUB(NOW(), INTERVAL 5 MINUTE)
                    THEN 1 ELSE 0 END as is_online
            FROM users
            WHERE $whereClause
            ORDER BY created_at DESC
            LIMIT ? OFFSET ?
        ";

        $params[] = $limit;
```

```
    $params[] = $offset;

    $stmt = $this->db->prepare($sql);
    $stmt->execute($params);
    $users = $stmt->fetchAll();

    return [
        'users' => $users,
        'pagination' => [
            'current_page' => $page,
            'per_page' => $limit,
            'total' => $totalUsers,
            'total_pages' => ceil($totalUsers / $limit),
            'has_next' => $page < ceil($totalUsers / $limit),
            'has_prev' => $page > 1
        ]
    ];
}

// Batch operations uchun optimizatsiya
public function bulkUpdateUsers($updates) {
    $this->db->beginTransaction();

    try {
        $stmt = $this->db->prepare("
            UPDATE users
            SET name = ?, email = ?, updated_at = NOW()
            WHERE id = ?
```

```
        ");

        foreach ($updates as $update) {
            $stmt->execute([
                $update['name'],
                $update['email'],
                $update['id']
            ]);
        }

        $this->db->commit();
        return ['updated' => count($updates)];

    } catch (Exception $e) {
        $this->db->rollback();
        throw $e;
    }
}

// Lazy loading bilan relation data
public function getUserWithPosts($userId, $loadPosts = false) {
    // User ma'lumotlarini olish
    $stmt = $this->db->prepare("
        SELECT id, name, email, created_at
        FROM users
        WHERE id = ? AND active = 1
    ");
    $stmt->execute([$userId]);
```

```
        $user = $stmt->fetch();

        if (!$user) {
            return null;
        }

        // Posts faqat kerak bo'lganda yuklash
        if ($loadPosts) {
            $stmt = $this->db->prepare("
                SELECT id, title, content, created_at
                FROM posts
                WHERE user_id = ? AND published = 1
                ORDER BY created_at DESC
                LIMIT 10
            ");
            $stmt->execute([$userId]);
            $user['posts'] = $stmt->fetchAll();
        }

        return $user;
    }
}
?>
```

2. Frontend optimizatsiyasi

```
// Debouncing bilan search optimization
class OptimizedSearch {
    constructor(searchInput, resultsContainer, apiUrl) {
        this.searchInput = searchInput;
```

```
    this.resultsContainer = resultsContainer;
    this.apiUrl = apiUrl;
    this.searchCache = new Map();
    this.debounceTimer = null;
    this.abortController = null;

    this.init();
}

init() {
    this.searchInput.addEventListener('input', (e) => {
        this.debouncedSearch(e.target.value);
    });

    // Keyboard navigation
    this.searchInput.addEventListener('keydown', (e) => {
        this.handleKeyNavigation(e);
    });
}

// Debounced search - ko'p so'rovlarni oldini olish
debouncedSearch(query) {
    clearTimeout(this.debounceTimer);

    this.debounceTimer = setTimeout(() => {
        this.performSearch(query);
    }, 300); // 300ms kechikish
}
```

```
async performSearch(query) {
    // Bo'sh query uchun
    if (!query.trim()) {
        this.clearResults();
        return;
    }

    // Keshdan tekshirish
    if (this.searchCache.has(query)) {
        this.displayResults(this.searchCache.get(query));
        return;
    }

    // Oldingi so'rovni bekor qilish
    if (this.abortController) {
        this.abortController.abort();
    }

    this.abortController = new AbortController();

    try {
        this.showLoadingState();

        const response = await fetch(`${this.apiUrl}?search=${encodeURIComponent(query)}`, {
            signal: this.abortController.signal,
            headers: {
```

```
            'Content-Type': 'application/json'
        }
    });

    if (!response.ok) {
        throw new Error(`HTTP ${response.status}`);
    }

    const data = await response.json();

    // Keshga saqlash
    this.searchCache.set(query, data);

    // Kesh hajmini cheklash (maksimal 50 ta query)
    if (this.searchCache.size > 50) {
        const firstKey = this.searchCache.keys().next().value;
        this.searchCache.delete(firstKey);
    }

    this.displayResults(data);

} catch (error) {
    if (error.name !== 'AbortError') {
        console.error('Search error:', error);
        this.showError('Qidirishda xatolik yuz berdi');
    }
} finally {
    this.hideLoadingState();
```

```
    }
}

displayResults(data) {
    this.resultsContainer.innerHTML = '';

    if (!data.users || data.users.length === 0) {
        this.showNoResults();
        return;
    }

    // Virtual scrolling uchun container
    const virtualContainer = document.createElement('div');
    virtualContainer.className = 'virtual-scroll-container';

    data.users.forEach((user, index) => {
        const resultItem = this.createResultItem(user, index);
        virtualContainer.appendChild(resultItem);
    });

    this.resultsContainer.appendChild(virtualContainer);

    // Birinchi elementni highlight qilish
    this.highlightResult(0);
}

createResultItem(user, index) {
    const item = document.createElement('div');
```

```
            item.className = 'search-result-item';
            item.setAttribute('data-index', index);
            item.setAttribute('data-user-id', user.id);

            item.innerHTML = `
                <div class="user-avatar">
                    <img src="${user.avatar || '/images/default-avatar.png'}"
                         alt="${this.escapeHtml(user.name)}" loading="lazy">
                </div>
                <div class="user-info">
                    <h4>${this.highlightMatch(user.name,
this.searchInput.value)}</h4>
                    <p>${this.highlightMatch(user.email, this.searchInput.value)}</p>
                    <span class="user-status ${user.is_online ? 'online' : 'offline'}">
                        ${user.is_online ? 'Online' : 'Offline'}
                    </span>
                </div>
            `;

            // Click event
            item.addEventListener('click', () => {
                this.selectUser(user);
            });

            return item;
        }

        // Query ni highlight qilish
```

```
    highlightMatch(text, query) {
        if (!query) return this.escapeHtml(text);

        const escapedText = this.escapeHtml(text);
        const escapedQuery = this.escapeHtml(query);
        const regex = new RegExp(`(${escapedQuery})`, 'gi');

        return escapedText.replace(regex, '<mark>$1</mark>');
    }

    // Keyboard navigation
    handleKeyNavigation(e) {
        const items = this.resultsContainer.querySelectorAll('.search-result-item');
        if (items.length === 0) return;

        const currentIndex = this.getCurrentHighlightIndex();

        switch (e.key) {
            case 'ArrowDown':
                e.preventDefault();
                this.highlightResult(Math.min(currentIndex + 1, items.length - 1));
                break;

            case 'ArrowUp':
                e.preventDefault();
                this.highlightResult(Math.max(currentIndex - 1, 0));
                break;
```

```
            case 'Enter':
                e.preventDefault();
                const highlightedItem = items[currentIndex];
                if (highlightedItem) {
                    const userId = highlightedItem.getAttribute('data-user-id');
                    this.selectUserById(userId);
                }
                break;

            case 'Escape':
                this.clearResults();
                this.searchInput.blur();
                break;
        }
    }

    highlightResult(index) {
        const items = this.resultsContainer.querySelectorAll('.search-result-item');

        // Oldingi highlight ni olib tashlash
        items.forEach(item => item.classList.remove('highlighted'));

        // Yangi highlight
        if (items[index]) {
            items[index].classList.add('highlighted');
            items[index].scrollIntoView({ block: 'nearest' });
```

```
        }
    }

    getCurrentHighlightIndex() {
        const highlightedItem = this.resultsContainer.querySelector('.search-result-item.highlighted');
        return highlightedItem ? parseInt(highlightedItem.getAttribute('data-index')) : -1;
    }

    selectUser(user) {
        // Custom event fire qilish
        const event = new CustomEvent('userSelected', {
            detail: { user }
        });
        document.dispatchEvent(event);

        this.clearResults();
        this.searchInput.value = user.name;
    }

    selectUserById(userId) {
        // Keshdan foydalanuvchini topish
        for (const [query, data] of this.searchCache) {
            const user = data.users.find(u => u.id === parseInt(userId));
            if (user) {
                this.selectUser(user);
                return;
```

```
            }
        }
    }

    showLoadingState() {
        this.resultsContainer.innerHTML = '<div class="search-loading">Qidirilmoqda...</div>';
    }

    hideLoadingState() {
        const loadingEl = this.resultsContainer.querySelector('.search-loading');
        if (loadingEl) loadingEl.remove();
    }

    showNoResults() {
        this.resultsContainer.innerHTML = '<div class="no-results">Hech narsa topilmadi</div>';
    }

    showError(message) {
        this.resultsContainer.innerHTML = `<div class="search-error">${this.escapeHtml(message)}</div>`;
    }

    clearResults() {
        this.resultsContainer.innerHTML = '';
    }
```

```
        escapeHtml(text) {
            const div = document.createElement('div');
            div.textContent = text;
            return div.innerHTML;
        }
    }
    // Lazy loading bilan image optimization
    class LazyImageLoader {
        constructor() {
            this.imageObserver = null;
            this.init();
        }

        init() {
            // Intersection Observer support tekshiruvi
            if ('IntersectionObserver' in window) {
                this.imageObserver = new IntersectionObserver((entries, observer) =>
{
                    entries.forEach(entry => {
                        if (entry.isIntersecting) {
                            this.loadImage(entry.target);
                            observer.unobserve(entry.target);
                        }
                    });
                }, {
                    rootMargin: '50px 0px', // 50px oldindan yuklash
                    threshold: 0.1
                });
```

```
    }

    // Mavjud lazy imagelarni kuzatish
    this.observeImages();

    // Yangi imagelar qo'shilganda ularni kuzatish
    this.observeNewImages();
}

observeImages() {
    const lazyImages = document.querySelectorAll('img[data-src]');

    if (this.imageObserver) {
        lazyImages.forEach(img => this.imageObserver.observe(img));
    } else {
        // Fallback - barcha imagelarni darhol yuklash
        lazyImages.forEach(img => this.loadImage(img));
    }
}

observeNewImages() {
    // MutationObserver bilan yangi imagelarni kuzatish
    const observer = new MutationObserver((mutations) => {
        mutations.forEach((mutation) => {
            mutation.addedNodes.forEach((node) => {
                if (node.nodeType === 1) { // Element node
                    const lazyImages = node.querySelectorAll('img[data-src]');
                    lazyImages.forEach(img => {
```

```
                    if (this.imageObserver) {
                        this.imageObserver.observe(img);
                    } else {
                        this.loadImage(img);
                    }
                });
            }
        });
    });
});

observer.observe(document.body, {
    childList: true,
    subtree: true
});
}

loadImage(img) {
    const src = img.getAttribute('data-src');
    if (!src) return;

    // Loading placeholder ko'rsatish
    img.classList.add('loading');

    // Yangi image obyekti yaratish
    const imageLoader = new Image();

    imageLoader.onload = () => {
```

```
            img.src = src;
            img.classList.remove('loading');
            img.classList.add('loaded');
            img.removeAttribute('data-src');
        };

        imageLoader.onerror = () => {
            img.classList.remove('loading');
            img.classList.add('error');
            img.src = '/images/placeholder-error.png'; // Error placeholder
        };

        imageLoader.src = src;
    }
}
// Request batching - ko'p so'rovlarni birlashtirish
class RequestBatcher {
    constructor(batchSize = 10, flushInterval = 100) {
        this.batchSize = batchSize;
        this.flushInterval = flushInterval;
        this.queue = [];
        this.timer = null;
    }

    add(request) {
        return new Promise((resolve, reject) => {
            this.queue.push({
                request,
```

```
            resolve,
            reject
        });

        // Batch to'ldi yoki timer tugadi
        if (this.queue.length >= this.batchSize) {
            this.flush();
        } else if (!this.timer) {
            this.timer = setTimeout(() => this.flush(), this.flushInterval);
        }
    });
}

async flush() {
    if (this.queue.length === 0) return;

    clearTimeout(this.timer);
    this.timer = null;

    const batch = this.queue.splice(0, this.batchSize);
    const requests = batch.map(item => item.request);

    try {
        const response = await fetch('/api/batch', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
```

```
            body: JSON.stringify({ requests })
        });

        const results = await response.json();

        // Har bir request uchun javobni qaytarish
        batch.forEach((item, index) => {
            const result = results[index];
            if (result.success) {
                item.resolve(result.data);
            } else {
                item.reject(new Error(result.error));
            }
        });

    } catch (error) {
        // Barcha requestlar uchun xatolik
        batch.forEach(item => item.reject(error));
    }
  }
}
// Service Worker bilan caching
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js')
    .then(registration => {
        console.log('Service Worker registered:', registration);
    })
    .catch(error => {
```

```
            console.log('Service Worker registration failed:', error);
        });
}
```

Service Worker (sw.js):

```
// sw.js - Service Worker for caching
const CACHE_NAME = 'php-ajax-app-v1';
const urlsToCache = [
    '/',
    '/css/app.css',
    '/js/app.js',
    '/images/default-avatar.png',
    '/images/placeholder-error.png'
];
// Install event
self.addEventListener('install', event => {
    event.waitUntil(
        caches.open(CACHE_NAME)
            .then(cache => {
                return cache.addAll(urlsToCache);
            })
    );
});
// Fetch event - network-first strategy for API, cache-first for assets
self.addEventListener('fetch', event => {
    const url = new URL(event.request.url);

    // API requestlari uchun network-first
    if (url.pathname.startsWith('/api/')) {
```

```
        event.respondWith(
            fetch(event.request)
                .then(response => {
                    // Faqat muvaffaqiyatli javoblarni keshlash
                    if (response.status === 200) {
                        const responseClone = response.clone();
                        caches.open(CACHE_NAME).then(cache => {
                            cache.put(event.request, responseClone);
                        });
                    }
                    return response;
                })
                .catch(() => {
                    // Network yo'q bo'lsa keshdan qaytarish
                    return caches.match(event.request);
                })
        );
    } else {
        // Static fayl uchun cache-first
        event.respondWith(
            caches.match(event.request)
                .then(response => {
                    return response || fetch(event.request);
                })
        );
    }
});
```

Xavfsizlik choralari va yaxshi amaliyotlar

1. Keng qamrovli xavfsizlik tizimi

```
<?php
// SecurityManager.php
class SecurityManager {
    private $config;

    public function __construct($config = []) {
        $this->config = array_merge([
            'rate_limit_requests' => 60,
            'rate_limit_window' => 60, // seconds
            'max_login_attempts' => 5,
            'login_lockout_time' => 900, // 15 minutes
            'session_lifetime' => 3600, // 1 hour
            'csrf_token_lifetime' => 1800, // 30 minutes
        ], $config);
    }

    // Rate limiting
    public function checkRateLimit($identifier, $action = 'default') {
        $key = "rate_limit:{$action}:{$identifier}";
        $requests = $this->getFromCache($key, 0);

        if ($requests >= $this->config['rate_limit_requests']) {
            http_response_code(429);
            echo json_encode([
                'error' => 'Juda ko\'p so\'rov yuborildi. Keyinroq urinib ko\'ring.',
                'retry_after' => $this->config['rate_limit_window']
            ]);
```

```
            exit;
        }

        $this->setCache($key, $requests + 1, $this->config['rate_limit_window']);
        return true;
    }

    // IP blacklist tekshiruvi
    public function checkIPBlacklist($ip) {
        $blacklistedIPs = $this->getBlacklistedIPs();

        if (in_array($ip, $blacklistedIPs)) {
            http_response_code(403);
            echo json_encode(['error' => 'Kirish taqiqlangan']);
            exit;
        }

        return true;
    }

    // Brute force hujumlardan himoya
    public function checkLoginAttempts($identifier) {
        $key = "login_attempts:{$identifier}";
        $attempts = $this->getFromCache($key, 0);

        if ($attempts >= $this->config['max_login_attempts']) {
            $lockoutKey = "login_lockout:{$identifier}";
```

```
            $lockoutTime = $this->getFromCache($lockoutKey);

            if ($lockoutTime) {
                http_response_code(423);
                echo json_encode([
                    'error' => 'Hisobingiz vaqtincha bloklangan. Keyinroq urinib ko\'ring.',
                    'locked_until' => date('Y-m-d H:i:s', $lockoutTime)
                ]);
                exit;
            }
        }

        return true;
    }

    // Login muvaffaqiyatsiz bo'lganda
    public function recordFailedLogin($identifier) {
        $key = "login_attempts:{$identifier}";
        $attempts = $this->getFromCache($key, 0) + 1;

        $this->setCache($key, $attempts, $this->config['login_lockout_time']);

        if ($attempts >= $this->config['max_login_attempts']) {
            $lockoutKey = "login_lockout:{$identifier}";
            $lockoutTime = time() + $this->config['login_lockout_time'];
            $this->setCache($lockoutKey, $lockoutTime, $this->config['login_lockout_time']);
```

```
        // Admin ga xabar yuborish
        $this->notifyAdmin("Brute force attack detected for: {$identifier}");
    }
}

// Muvaffaqiyatli login
public function recordSuccessfulLogin($identifier) {
    $this->deleteFromCache("login_attempts:{$identifier}");
    $this->deleteFromCache("login_lockout:{$identifier}");
}

// Advanced CSRF protection
public function generateCSRFToken($action = 'default') {
    session_start();

    $token = bin2hex(random_bytes(32));
    $timestamp = time();

    $_SESSION['csrf_tokens'][$action] = [
        'token' => $token,
        'timestamp' => $timestamp
    ];

    // Eski tokenlarni tozalash
    $this->cleanupExpiredTokens();

    return $token;
```

```
    }

    public function validateCSRFToken($token, $action = 'default') {
        session_start();

        if (!isset($_SESSION['csrf_tokens'][$action])) {
            return false;
        }

        $storedData = $_SESSION['csrf_tokens'][$action];

        // Token yaroqlilik muddatini tekshirish
        if ((time() - $storedData['timestamp']) > $this->config['csrf_token_lifetime']) {
            unset($_SESSION['csrf_tokens'][$action]);
            return false;
        }

        // Token mosligini tekshirish
        if (!hash_equals($storedData['token'], $token)) {
            return false;
        }

        // One-time token - ishlatilgandan keyin o'chirish
        unset($_SESSION['csrf_tokens'][$action]);
        return true;
    }
```

```
    // SQL Injection dan himoya
    public function sanitizeSQL($input) {
        if (is_array($input)) {
            return array_map([$this, 'sanitizeSQL'], $input);
        }

        // PDO prepared statements ishlatilganligi uchun qo'shimcha sanitization
        return trim($input);
    }

    // XSS dan himoya
    public function sanitizeOutput($data) {
        if (is_array($data)) {
            return array_map([$this, 'sanitizeOutput'], $data);
        }

        return htmlspecialchars($data, ENT_QUOTES | ENT_HTML5, 'UTF-8');
    }

    // File upload xavfsizligi
    public function validateFileUpload($file, $allowedTypes = [], $maxSize =
5242880) { // 5MB
        $errors = [];

        // Fayl mavjudligini tekshirish
        if (!isset($file['tmp_name']) || !is_uploaded_file($file['tmp_name'])) {
            $errors[] = 'Fayl yuklanmagan';
            return $errors;
```

```
    }

    // Fayl hajmini tekshirish
    if ($file['size'] > $maxSize) {
        $errors[] = 'Fayl hajmi juda katta (' . ($maxSize / 1024 / 1024) . 'MB dan oshmasligi kerak)';
    }

    // MIME type tekshiruvi
    $finfo = finfo_open(FILEINFO_MIME_TYPE);
    $mimeType = finfo_file($finfo, $file['tmp_name']);
    finfo_close($finfo);

    if (!empty($allowedTypes) && !in_array($mimeType, $allowedTypes)) {
        $errors[] = 'Fayl turi ruxsat etilmagan';
    }

    // Fayl nomi xavfsizligi
    $filename = $file['name'];
    if (preg_match('/[^a-zA-Z0-9._-]/', $filename)) {
        $errors[] = 'Fayl nomida noto\'g\'ri belgilar mavjud';
    }

    // Executable fayllarni tekshirish
    $dangerousExtensions = ['php', 'phtml', 'php3', 'php4', 'php5', 'php7', 'phps', 'js', 'exe', 'bat', 'sh'];
```

```
            $extension = strtolower(pathinfo($filename,
PATHINFO_EXTENSION));

            if (in_array($extension, $dangerousExtensions)) {
                $errors[] = 'Fayl kengaytmasi xavfli';
            }

            return $errors;
        }

        private function cleanupExpiredTokens() {
            if (!isset($_SESSION['csrf_tokens'])) return;

            $currentTime = time();
            foreach ($_SESSION['csrf_tokens'] as $action => $data) {
                if (($currentTime - $data['timestamp']) > $this-
>config['csrf_token_lifetime']) {
                    unset($_SESSION['csrf_tokens'][$action]);
                }
            }
        }

        private function getFromCache($key, $default = null) {
            // Redis yoki memcache ishlatish tavsiya etiladi
            // Bu yerda oddiy file cache
            $cacheFile = "cache/" . md5($key) . ".cache";

            if (file_exists($cacheFile)) {
```

```
        $data = json_decode(file_get_contents($cacheFile), true);
        if ($data && $data['expires'] > time()) {
            return $data['value'];
        } else {
            unlink($cacheFile);
        }
    }

    return $default;
}

private function setCache($key, $value, $ttl) {
    $cacheDir = "cache";
    if (!is_dir($cacheDir)) {
        mkdir($cacheDir, 0755, true);
    }

    $cacheFile = "$cacheDir/" . md5($key) . ".cache";
    $data = [
        'value' => $value,
        'expires' => time() + $ttl
    ];

    file_put_contents($cacheFile, json_encode($data));
}

private function deleteFromCache($key) {
    $cacheFile = "cache/" . md5($key) . ".cache";
```

```
        if (file_exists($cacheFile)) {
            unlink($cacheFile);
        }
    }

    private function getBlacklistedIPs() {
        // Ma'lumotlar bazasidan yoki fayldan o'qish
        return [
            // Blacklisted IP addresses
        ];
    }

    private function notifyAdmin($message) {
        // Email, SMS yoki boshqa notification usuli
        error_log("Security Alert: " . $message);
    }
}
?>
```

PHP va AJAX integratsiyasi zamonaviy veb-dasturlashning muhim qismi bo'lib, to'g'ri amalga oshirilganda kuchli va samarali ilovalar yaratish imkonini beradi. Ushbu maqolada ko'rib chiqilgan asosiy nuqtalar:

Asosiy yutuqlar:

- **Xavfsizlik:** CSRF, XSS va boshqa hujumlardan himoya mexanizmlari
- **Ishlash samaradorligi:** Caching, connection pooling va optimizatsiya strategiyalari
- **Xatoliklarni boshqarish:** To'liq logging va debugging tizimi
- **Real-time aloqa:** Server-Sent Events va WebSocket texnologiyalari
- **Foydalanuvchi tajribasi:** Lazy loading, debouncing va responsive design

Tavsiyalar:

1. **Xavfsizlikni birinchi o'ringa qo'ying** - barcha input va outputlarni tekshiring
2. **Asinxron operatsiyalarni to'g'ri boshqaring** - async/await dan foydalaning
3. **Ma'lumotlarni keshlang** - tez-tez ishlatiladigan ma'lumotlarni kesh qiling
4. **Xatoliklarni to'liq logga yozing** - debugging uchun muhim
5. **API versiyalashni joriy eting** - backward compatibility uchun
6. **Testlar yozing** - unit va integration testlar
7. **Monitoring o'rnating** - ishlash ko'rsatkichlarini kuzating

Kelajakdagi yo'nalishlar:

- GraphQL bilan integratsiya
- Microservices arxitekturasi
- Progressive Web Apps (PWA)
- AI/ML integration
- Blockchain texnologiyalari

PHP va AJAX texnologiyalarining to'g'ri integratsiyasi orqali zamonaviy, xavfsiz va samarali veb-ilovalar yaratish mumkin. Muhimi - doimiy ravishda yangi texnologiyalar va best practicelarni o'rganib borish.

Foydalanilgan adabiyotlar ro'yxati

1. Flanagan, D. (2020). "JavaScript: The Definitive Guide, 7th Edition." O'Reilly Media.

2. Tatroe, K., MacIntyre, P., & Lerdorf, R. (2019). "Programming PHP, 4th Edition." O'Reilly Media.

3. Lockhart, J. (2017). "Modern PHP: New Features and Good Practices." O'Reilly Media.

4. Grinberg, M. (2018). "Flask Web Development: Developing Web Applications with Python, 2nd Edition." O'Reilly Media.

5. Mozilla Developer Network. (2023). "AJAX - Getting Started." https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

6. OWASP Foundation. (2023). "OWASP Top Ten Web Application Security Risks." https://owasp.org/www-project-top-ten/

7. Richardson, L., & Ruby, S. (2013). "RESTful Web APIs: Services for a Changing World." O'Reilly Media.

8. Hunt, A., & Thomas, D. (2019). "The Pragmatic Programmer: Your Journey to Mastery, 20th Anniversary Edition." Addison-Wesley Professional.

9. Martin, R. C. (2017). "Clean Code: A Handbook of Agile Software Craftsmanship." Prentice Hall.

10. Evans, E. (2003). "Domain-Driven Design: Tackling Complexity in the Heart of Software." Addison-Wesley Professional.