Documentação Técnica - Vitexa V1

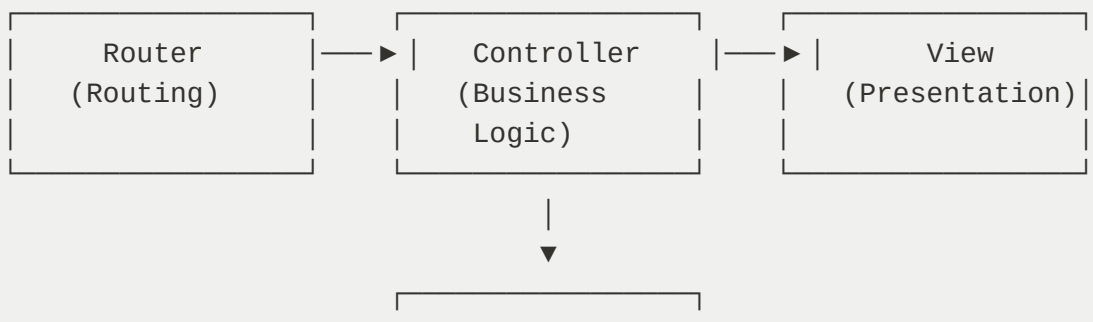
Índice

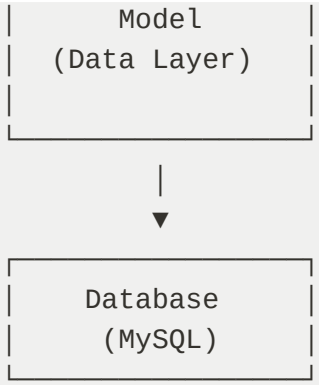
Visão Geral da Arquitetura

O Vitexa V1 é construído usando uma arquitetura MVC (Model-View-Controller) personalizada em PHP, projetada para ser escalável, segura e de fácil manutenção.

Componentes Principais

Plain Text





Fluxo de Requisição

1. **Entrada**: Todas as requisições passam por `public/index.php`
2. **Roteamento**: O `Router` analisa a URL e determina o controlador/ação
3. **Middleware**: Verificações de autenticação e segurança
4. **Controlador**: Processa a lógica de negócio
5. **Modelo**: Interage com o banco de dados
6. **View**: Renderiza a resposta HTML ou JSON
7. **Saída**: Resposta enviada ao cliente

Estrutura MVC

Model (Modelo)

Os modelos são responsáveis pela lógica de dados e interação com o banco de dados.

Localização: `app/models/`

Classe Base Model

PHP

```
// app/core/Model.php
class Model {
    protected $db;
    protected $table;

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

```
    // Métodos básicos de CRUD
    public function find($id) { /* ... */ }
    public function create($data) { /* ... */ }
    public function update($id, $data) { /* ... */ }
    public function delete($id) { /* ... */ }
}
```

Exemplo de Uso - User Model

PHP

```
// app/models/User.php
class User extends Model {
    protected $table = 'users';

    public function findByEmail($email) {
        $stmt = $this->db->prepare("SELECT * FROM users WHERE email = ?");
        $stmt->execute([$email]);
        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    public function createUser($data) {
        // Validação e sanitização
        $data['password'] = password_hash($data['password'],
PASSWORD_BCRYPT);

        return $this->create($data);
    }
}
```

View (Visualização)

As views são responsáveis pela apresentação dos dados.

Localização: `app/views/`

Sistema de Templates

PHP

```
// app/core/View.php
class View {
    public static function render($view, $data = []) {
        extract($data);

        ob_start();
```

```
        require_once "app/views/{$view}.php";
        $content = ob_get_clean();

        return $content;
    }
}
```

Exemplo de View

PHP

```
// app/views/dashboard/index.php
<div class="dashboard">
    <h1>Bem-vindo, <?= htmlspecialchars($user['name']) ?>!</h1>

    <div class="stats-grid">
        <div class="stat-card">
            <h3>IMC</h3>
            <p class="stat-value"><?= $stats['bmi'] ?></p>
        </div>
        <!-- Mais estatísticas... -->
    </div>
</div>
```

Controller (Controlador)

Os controladores gerenciam a lógica de negócio e coordenam Models e Views.

Localização: `app/controllers/`

Classe Base Controller

PHP

```
// app/core/Controller.php
class Controller {
    protected function render($view, $data = []) {
        return View::render($view, $data);
    }

    protected function json($data, $status = 200) {
        http_response_code($status);
        header('Content-Type: application/json');
        echo json_encode($data);
        exit;
    }
```

```
    protected function requireAuth() {
        if (!Session::isLoggedIn()) {
            $this->redirect(APP_URL . '/login');
        }
    }
}
```

Exemplo de Controller

PHP

```
// app/controllers/UserController.php
class UserController extends Controller {
    public function dashboard() {
        $this->requireAuth();

        $user = $this->getCurrentUser();
        $userModel = new User();
        $stats = $userModel->getUserStats($user['id']);

        echo $this->render('dashboard/index', [
            'user' => $user,
            'stats' => $stats,
            'title' => 'Dashboard - Vitexa'
        ]);
    }
}
```

Sistema de Roteamento

O sistema de roteamento personalizado mapeia URLs para controladores e ações.

Router Core

PHP

```
// app/core/Router.php
class Router {
    private $routes = [];
    private $middleware = [];

    public function addRoute($method, $path, $controller, $action,
$middleware = []) {
```

```
        $this->routes[] = [
            'method' => $method,
            'path' => $path,
            'controller' => $controller,
            'action' => $action,
            'middleware' => $middleware
        ];
    }

    public function dispatch($uri, $method) {
        foreach ($this->routes as $route) {
            if ($this->matchRoute($route, $uri, $method)) {
                $this->executeRoute($route);
                return;
            }
        }

        $this->handleNotFound();
    }
}
```

Definição de Rotas

PHP

```
// public/index.php
$router = new Router();

// Rotas públicas
$router->addRoute('GET', '/', 'HomeController', 'index');
$router->addRoute('GET', '/login', 'AuthController', 'loginForm');
$router->addRoute('POST', '/login', 'AuthController', 'login');

// Rotas protegidas
$router->addRoute('GET', '/dashboard', 'UserController', 'dashboard',
['auth']);
$router->addRoute('POST', '/plans/generate', 'PlanController', 'generate',
['auth']);
```

Middleware de Autenticação

PHP

```
// app/core/AuthMiddleware.php
class AuthMiddleware {
    public static function handle() {
```

```
        if (!Session::isLoggedIn()) {
            if (self::isApiRequest()) {
                http_response_code(401);
                echo json_encode(['error' => 'Não autorizado']);
                exit;
            } else {
                header('Location: /login');
                exit;
            }
        }
    }
}
```

Controladores (Controllers)

AuthController

Gerencia autenticação de usuários.

Métodos Principais

PHP

```
class AuthController extends Controller {
    // Exibir formulário de login
    public function loginForm() {
        if (Session::isLoggedIn()) {
            $this->redirect(APP_URL . '/dashboard');
        }

        echo $this->render('auth/login', [
            'title' => 'Login - Vitexa',
            'csrf_token' => Session::generateCsrfToken()
        ]);
    }

    // Processar login
    public function login() {
        // Verificar CSRF
        if (!$this->verifyCsrfToken($this->input('_token'))) {
            $this->json(['error' => 'Token inválido'], 403);
        }

        $email = $this->sanitize($this->input('email'));
        $password = $this->input('password');
```

```
        $userModel = new User();
        $user = $userModel->findByEmail($email);

        if ($user && password_verify($password, $user['password'])) {
            Session::login($user);
            $this->json(['success' => true, 'redirect' => '/dashboard']);
        } else {
            $this->json(['error' => 'Credenciais inválidas'], 401);
        }
    }
}
```

UserController

Gerencia o dashboard e perfil do usuário.

Métodos Principais

PHP

```
class UserController extends Controller {
    // Dashboard principal
    public function dashboard() {
        $this->requireAuth();

        $user = $this->getCurrentUser();
        $userModel = new User();

        // Obter estatísticas do usuário
        $stats = $userModel->getUserStats($user['id']);

        // Obter plano do dia
        $planModel = new Plan();
        $todayWorkout = $planModel->getTodayWorkout($user['id']);
        $todayMeals = $planModel->getTodayMeals($user['id']);

        // Obter progresso recente
        $recentProgress = $userModel->getRecentProgress($user['id'], 7);

        echo $this->render('dashboard/index', [
            'user' => $user,
            'stats' => $stats,
            'today_workout' => $todayWorkout,
            'today_meals' => $todayMeals,
            'recent_progress' => $recentProgress,
            'title' => 'Dashboard - Vitexa'
```

```
        ]);
    }

    // Registrar progresso
    public function recordProgress() {
        $this->requireAuth();

        if (!$this->verifyCsrfToken($this->input('_token'))) {
            $this->json(['error' => 'Token inválido'], 403);
        }

        $user = $this->getCurrentUser();
        $data = [
            'weight' => (float)$this->input('weight'),
            'body_fat' => $this->input('body_fat') ? (float)$this-
>input('body_fat') : null,
            'muscle_mass' => $this->input('muscle_mass') ? (float)$this-
>input('muscle_mass') : null,
            'notes' => $this->sanitize($this->input('notes'))
        ];

        $userModel = new User();
        $progressId = $userModel->recordProgress($user['id'], $data);

        $this->json([
            'success' => true,
            'progress_id' => $progressId,
            'message' => 'Progresso registrado com sucesso!'
        ]);
    }
}
```

PlanController

Gerencia geração e visualização de planos de treino e dieta.

Métodos Principais

PHP

```
class PlanController extends Controller {
    // Gerar plano via IA
    public function generate() {
        $this->requireAuth();

        $user = $this->getCurrentUser();
        $type = $this->input('type'); // 'workout' ou 'diet'
```

```
        // Verificar cache primeiro
        $cacheKey = Cache::userKey($user['id'], "plan_{$type}");
        $cachedPlan = Cache::get($cacheKey);

        if ($cachedPlan) {
            $this->json(['success' => true, 'plan' => $cachedPlan]);
            return;
        }

        try {
            $planModel = new Plan();

            if ($type === 'workout') {
                $plan = $this->generateWorkoutPlan($user);
            } elseif ($type === 'diet') {
                $plan = $this->generateDietPlan($user);
            } else {
                $this->json(['error' => 'Tipo de plano inválido'], 400);
                return;
            }

            // Salvar no banco
            $planId = $planModel->savePlan($user['id'], $type, $plan);

            // Salvar em cache por 24 horas
            Cache::set($cacheKey, $plan, 86400);

            $this->json([
                'success' => true,
                'plan_id' => $planId,
                'plan' => $plan
            ]);

        } catch (Exception $e) {
            error_log("Erro ao gerar plano: " . $e->getMessage());
            $this->json(['error' => 'Erro ao gerar plano'], 500);
        }
    }

    // Gerar plano de treino via OpenAI
    private function generateWorkoutPlan($user) {
        $prompt = $this->buildWorkoutPrompt($user);

        $data = [
            'model' => 'gpt-3.5-turbo',
            'messages' => [
                ['role' => 'system', 'content' => $this-
```

```
>getWorkoutSystemPrompt()],
                ['role' => 'user', 'content' => $prompt]
            ],
            'max_tokens' => 2000,
            'temperature' => 0.7
        ];

        $response = $this->callOpenAI($data);
        return $this->parseWorkoutResponse($response);
    }
}
```

ChatController

Gerencia o sistema de chat com IA.

Métodos Principais

PHP

```
class ChatController extends Controller {
    // Enviar mensagem para IA
    public function send() {
        $this->requireAuth();

        $user = $this->getCurrentUser();
        $message = $this->sanitize($this->input('message'));

        if (empty($message) || strlen($message) > 1000) {
            $this->json(['error' => 'Mensagem inválida'], 400);
        }

        try {
            $messageModel = new Message();

            // Salvar mensagem do usuário
            $userMessageId = $messageModel->saveMessage($user['id'],
$message, 'user');

            // Gerar resposta da IA
            $aiResponse = $this->generateAIResponse($user, $message);

            // Salvar resposta da IA
            $aiMessageId = $messageModel->saveMessage($user['id'],
$aiResponse, 'assistant');

            $this->json([
```

```
                'success' => true,
                'user_message' => [
                    'id' => $userMessageId,
                    'message' => $message,
                    'type' => 'user',
                    'created_at' => date('Y-m-d H:i:s')
                ],
                'ai_message' => [
                    'id' => $aiMessageId,
                    'message' => $aiResponse,
                    'type' => 'assistant',
                    'created_at' => date('Y-m-d H:i:s')
                ]
            ]);

        } catch (Exception $e) {
            error_log("Erro ao enviar mensagem: " . $e->getMessage());
            $this->json(['error' => 'Erro ao processar mensagem'], 500);
        }
    }
}
```

ReminderController

Gerencia os lembretes dos usuários.

Métodos Principais

PHP

```
class ReminderController extends Controller {
    // Listar lembretes
    public function index() { /* ... */ }

    // Criar um novo lembrete
    public function create() { /* ... */ }

    // Atualizar um lembrete existente
    public function update() { /* ... */ }

    // Deletar um lembrete
    public function delete() { /* ... */ }

    // Ativar/desativar um lembrete
    public function toggle() { /* ... */ }

    // Processar lembretes pendentes (via cron)
```

```
    public function processPending() { /* ... */ }
}
```

Modelos (Models)

User.php

Gerencia os dados dos usuários, incluindo perfil, progresso e lembretes.

Plan.php

Responsável por salvar e recuperar os planos de treino e dieta gerados pela IA.

Message.php

Armazena o histórico de conversas do chat entre o usuário e a IA.

Sistema de Lembretes

O sistema de lembretes permite que os usuários configurem notificações para diversas atividades.

Funcionalidades

- **Tipos de Lembretes**: Treino, dieta, água, medicamentos, geral.
- **Agendamento Flexível**: Horário e dias da semana personalizáveis.
- **Notificações**: Atualmente, os lembretes são registrados no log do sistema. A estrutura está pronta para ser estendida para envio de e-mails, push notifications, etc.
- **Processamento em Background**: Um script cron (`cron/process_reminders.php`) é executado periodicamente para verificar e enviar lembretes pendentes.

Fluxo de Processamento do Cron

1. O cron job é acionado (geralmente a cada minuto).
2. O script `process_reminders.php` chama o método `ReminderController::processPending()` .
3. O método `getPendingReminders()` no `User` model busca no banco de dados todos os lembretes ativos que correspondem ao horário e dia da semana atuais.
4. Para cada lembrete encontrado, o método `sendReminderNotification()` é chamado.
5. A notificação é enviada (atualmente, registrada no log).

Segurança

Medidas Implementadas

- **Autenticação**: Hash bcrypt para senhas.
- **CSRF Protection**: Tokens em todos os formulários.
- **XSS Protection**: Sanitização de dados de entrada e uso de `htmlspecialchars()`.
- **SQL Injection**: Uso de prepared statements com PDO.
- **Headers de Segurança**: CSP, X-Frame-Options, HSTS, etc., configurados em `config/config.php`.
- **Rate Limiting**: Proteção contra ataques de força bruta (configurável em `config/config.php`).
- **Validação de Dados**: Validação rigorosa no backend para todos os dados recebidos.

Configurações Adicionais

O arquivo `config/config.php` centraliza todas as configurações da aplicação, permitindo fácil customização.

- **Push Notifications**: Integração com Firebase Cloud Messaging (FCM) para notificações push.
- **Webhooks**: Endpoint para receber notificações de serviços externos.
- **Uploads**: Limites de tamanho e tipos de arquivo permitidos.
- **Logging**: Níveis de log e diretório de armazenamento.
- **Timezone**: Configuração do fuso horário da aplicação.

HomeController

Gerencia a página inicial e rotas públicas básicas.

Métodos Principais

PHP

```
class HomeController extends Controller {
    // Exibe a página inicial da aplicação
    public function index() {
        echo $this->render(
```

```
            (Session::isLoggedIn() ? 'dashboard/index' : 'home/index'),
            ['title' => APP_NAME]
        );
    }
}
```

ReminderController

Gerencia a criação, edição, exclusão, ativação/desativação e processamento de lembretes para os usuários. Este controlador interage diretamente com o `User` Model para persistência dos dados e com o sistema de cron jobs para o envio de notificações.

Métodos Principais

PHP

```
class ReminderController extends Controller {
    /**
     * Exibe a página de gerenciamento de lembretes do usuário.
     * Requer autenticação.
     */
    public function index() {
        $this->requireAuth();
        $user = $this->getCurrentUser();
        $userModel = new User();
        $reminders = $userModel->getReminders($user['id']);
        echo $this->render('reminders/index', [
            'title' => 'Lembretes - ' . APP_NAME,
            'user' => $user,
            'reminders' => $reminders
        ]);
    }

    /**
     * Cria um novo lembrete para o usuário logado.
     * Requer autenticação e validação de CSRF token.
     * Recebe dados via POST: title, message, type, time, days_of_week
(array).
     * Retorna JSON com sucesso/erro e ID do lembrete.
     */
    public function create() {
        $this->requireAuth();
        if (!$this->verifyCsrfToken($this->input('_token'))) {
            $this->json(['error' => 'Token de segurança inválido'], 403);
        }
        $user = $this->getCurrentUser();
        $userModel = new User();
```

```
        $data = [
            'title' => $this->sanitize($this->input('title')),
            'message' => $this->sanitize($this->input('message')),
            'type' => $this->sanitize($this->input('type')),
            'time' => $this->sanitize($this->input('time')),
            'days_of_week' => $this->input('days_of_week', []),
            'is_active' => 1
        ];
        // Validação de dados e dias da semana
        // ... (código de validação omitido para brevidade)
        try {
            $reminderId = $userModel->addReminder($user['id'], $data);
            $this->json(['success' => true, 'reminder_id' => $reminderId,
'message' => 'Lembrete criado com sucesso!']);
        } catch (Exception $e) {
            error_log("Erro ao criar lembrete: " . $e->getMessage());
            $this->json(['error' => 'Erro ao criar lembrete. Tente
novamente.'], 500);
        }
    }

    /**
     * Atualiza um lembrete existente do usuário logado.
     * Requer autenticação e validação de CSRF token.
     * Recebe dados via POST: reminder_id, title, message, type, time,
days_of_week (array), is_active.
     * Retorna JSON com sucesso/erro.
     */
    public function update() {
        $this->requireAuth();
        if (!$this->verifyCsrfToken($this->input('_token'))) {
            $this->json(['error' => 'Token de segurança inválido'], 403);
        }
        $user = $this->getCurrentUser();
        $userModel = new User();
        $reminderId = (int)$this->input('reminder_id');
        // Verifica se o lembrete pertence ao usuário
        // ... (código de verificação omitido para brevidade)
        $data = [
            'title' => $this->sanitize($this->input('title')),
            'message' => $this->sanitize($this->input('message')),
            'type' => $this->sanitize($this->input('type')),
            'time' => $this->sanitize($this->input('time')),
            'days_of_week' => $this->input('days_of_week', []),
            'is_active' => (int)$this->input('is_active', 1)
        ];
        // Validação de dados
        // ... (código de validação omitido para brevidade)
```

```
        try {
            $userModel->updateReminder($user['id'], $reminderId, $data);
            $this->json(['success' => true, 'message' => 'Lembrete
atualizado com sucesso!']);
        } catch (Exception $e) {
            error_log("Erro ao atualizar lembrete: " . $e->getMessage());
            $this->json(['error' => 'Erro ao atualizar lembrete. Tente
novamente.'], 500);
        }
    }

    /**
     * Exclui um lembrete do usuário logado.
     * Requer autenticação e validação de CSRF token.
     * Recebe via POST: reminder_id.
     * Retorna JSON com sucesso/erro.
     */
    public function delete() {
        $this->requireAuth();
        if (!$this->verifyCsrfToken($this->input('_token'))) {
            $this->json(['error' => 'Token de segurança inválido'], 403);
        }
        $user = $this->getCurrentUser();
        $userModel = new User();
        $reminderId = (int)$this->input('reminder_id');
        // Verifica se o lembrete pertence ao usuário
        // ... (código de verificação omitido para brevidade)
        try {
            $userModel->deleteReminder($user['id'], $reminderId);
            $this->json(['success' => true, 'message' => 'Lembrete removido
com sucesso!']);
        } catch (Exception $e) {
            error_log("Erro ao deletar lembrete: " . $e->getMessage());
            $this->json(['error' => 'Erro ao deletar lembrete. Tente
novamente.'], 500);
        }
    }

    /**
     * Alterna o status (ativo/inativo) de um lembrete.
     * Requer autenticação e validação de CSRF token.
     * Recebe via POST: reminder_id.
     * Retorna JSON com sucesso/erro e o novo status.
     */
    public function toggle() {
        $this->requireAuth();
        if (!$this->verifyCsrfToken($this->input('_token'))) {
            $this->json(['error' => 'Token de segurança inválido'], 403);
```

```
        }
        $user = $this->getCurrentUser();
        $userModel = new User();
        $reminderId = (int)$this->input('reminder_id');
        // Verifica se o lembrete pertence ao usuário
        // ... (código de verificação omitido para brevidade)
        try {
            $newStatus = $reminder['is_active'] ? 0 : 1;
            $userModel->updateReminder($user['id'], $reminderId,
['is_active' => $newStatus]);
            $this->json(['success' => true, 'is_active' => $newStatus,
'message' => $newStatus ? 'Lembrete ativado!' : 'Lembrete desativado!']);
        } catch (Exception $e) {
            error_log("Erro ao alterar status do lembrete: " . $e-
>getMessage());
            $this->json(['error' => 'Erro ao alterar status do lembrete.'],
500);
        }
    }

    /**
     * Processa lembretes pendentes. Este método é projetado para ser
chamado via cron job.
     * Verifica lembretes ativos que correspondem ao horário e dia da semana
atuais e os processa.
     * Retorna JSON com o número de lembretes enviados e erros.
     */
    public function processPending() {
        // Verifica se a requisição é via CLI ou cron com token válido
        if (php_sapi_name() !== 'cli' && !$this->isValidCronRequest()) {
            http_response_code(403);
            echo json_encode(['error' => 'Acesso negado']);
            exit;
        }
        $userModel = new User();
        try {
            $currentTime = date('H:i');
            $currentDay = (int)date('w'); // 0=Domingo, 6=Sábado
            $pendingReminders = $userModel-
>getPendingReminders($currentTime, $currentDay);
            $sent = 0;
            $errors = 0;
            foreach ($pendingReminders as $reminder) {
                try {
                    $this->sendReminderNotification($reminder);
                    $sent++;
                } catch (Exception $e) {
                    error_log("Erro ao enviar lembrete {$reminder['id']}: "
```

```
. $e->getMessage());
                    $errors++;
                }
            }
            echo json_encode(['success' => true, 'sent' => $sent, 'errors'
=> $errors, 'total' => count($pendingReminders)]);
        } catch (Exception $e) {
            error_log("Erro ao processar lembretes: " . $e->getMessage());
            echo json_encode(['error' => 'Erro ao processar lembretes']);
        }
    }

    /**
     * Envia uma notificação de lembrete. Atualmente, apenas loga a mensagem.
     * Pode ser estendido para enviar e-mails, push notifications, SMS, etc.
     */
    private function sendReminderNotification($reminder) {
        $message = "Lembrete para {$reminder['user_name']}:
{$reminder['title']}";
        if ($reminder['message']) {
            $message .= " - " . $reminder['message'];
        }
        error_log("REMINDER: " . $message);
        return true;
    }

    /**
     * Verifica se a requisição de cron é válida através de um token.
     */
    private function isValidCronRequest() {
        $cronToken = $this->input('cron_token');
        return $cronToken === CRON_TOKEN;
    }

    // Métodos de API para gerenciamento de lembretes (CRUD)
    public function apiCreate() { $this->create(); }
    public function apiUpdate() { $this->update(); }
    public function apiDelete() { $this->delete(); }
    public function apiToggle() { $this->toggle(); }
    public function apiList() {
        $this->requireAuth();
        $user = $this->getCurrentUser();
        $userModel = new User();
        $reminders = $userModel->getReminders($user['id']);
        $this->json(['reminders' => $reminders]);
    }
}
```

User.php

O modelo `User` é central para a gestão de usuários no sistema Vitexa. Ele estende a classe `Model` base e interage diretamente com a tabela `users` no banco de dados, além de gerenciar dados relacionados como progresso e lembretes.

Propriedades Principais

- `protected $table = 'users';` : Define a tabela principal associada ao modelo.
- `protected $primaryKey = 'id';` : Define a chave primária da tabela.
- `protected $fillable` : Array de campos que podem ser preenchidos em massa (`name` , `email` , `password_hash` , `age` , `weight` , `height` , `goal`).
- `protected $timestamps = true;` : Indica que o modelo gerencia automaticamente `created_at` e `updated_at` .

Métodos Principais

PHP

```
class User extends Model {
    /**
     * Autentica um usuário com base no email e senha.
     * Utiliza password_verify para checar o hash da senha.
     * @param string $email
     * @param string $password
     * @return array|false Retorna os dados do usuário (sem o hash da senha)
em caso de sucesso, ou false.
     */
    public function authenticate($email, $password) { /* ... */ }

    /**
     * Cria um novo usuário no sistema.
     * Realiza a validação para evitar emails duplicados e faz o hash da
senha.
     * @param array $data Dados do usuário (incluindo 'email' e 'password').
     * @return int O ID do novo usuário.
     * @throws Exception Se o email já estiver cadastrado.
     */
    public function createUser($data) { /* ... */ }

    /**
     * Atualiza o perfil de um usuário existente.
     * Impede a atualização de campos sensíveis como ID, email e hash da
senha.
     * @param int $userId ID do usuário a ser atualizado.
```

```
     * @param array $data Dados do perfil a serem atualizados.
     * @return bool True em caso de sucesso, false caso contrário.
     */
    public function updateProfile($userId, $data) { /* ... */ }

    /**
     * Altera a senha de um usuário após verificar a senha atual.
     * @param int $userId ID do usuário.
     * @param string $currentPassword Senha atual do usuário.
     * @param string $newPassword Nova senha.
     * @return bool True em caso de sucesso.
     * @throws Exception Se o usuário não for encontrado ou a senha atual
estiver incorreta.
     */
    public function changePassword($userId, $currentPassword, $newPassword)
{ /* ... */ }

    /**
     * Obtém estatísticas detalhadas do usuário, incluindo IMC, categoria de
IMC, peso atual,
     * mudança de peso, planos ativos, dias desde o cadastro e objetivo.
     * @param int $userId ID do usuário.
     * @return array|null Array associativo com as estatísticas, ou null se
o usuário não for encontrado.
     */
    public function getStats($userId) { /* ... */ }

    /**
     * Obtém o histórico de progresso do usuário por um número específico de
dias.
     * @param int $userId ID do usuário.
     * @param int $days Número de dias para buscar o histórico (padrão: 30).
     * @return array Array de registros de progresso.
     */
    public function getProgress($userId, $days = 30) { /* ... */ }

    /**
     * Adiciona ou atualiza um registro de progresso para o usuário em uma
data específica.
     * Se já existir um registro para a data, ele é atualizado; caso
contrário, um novo é criado.
     * @param int $userId ID do usuário.
     * @param array $data Dados do progresso (peso, percentual de gordura,
massa muscular, notas).
     * @return int O ID do registro de progresso (novo ou atualizado).
     */
    public function addProgress($userId, $data) { /* ... */ }
```

```
    /**
     * Obtém todos os lembretes ativos configurados para um usuário.
     * @param int $userId ID do usuário.
     * @return array Array de lembretes.
     */
    public function getReminders($userId) { /* ... */ }

    /**
     * Adiciona um novo lembrete para o usuário.
     * Os dias da semana são codificados em JSON antes de serem salvos.
     * @param int $userId ID do usuário.
     * @param array $data Dados do lembrete (título, mensagem, tipo, hora,
dias da semana).
     * @return int O ID do novo lembrete.
     */
    public function addReminder($userId, $data) { /* ... */ }

    /**
     * Atualiza um lembrete existente.
     * @param int $reminderId ID do lembrete a ser atualizado.
     * @param array $data Dados a serem atualizados.
     * @return bool True em caso de sucesso.
     */
    public function updateReminder($reminderId, $data) { /* ... */ }

    /**
     * Exclui um lembrete do sistema.
     * @param int $reminderId ID do lembrete a ser excluído.
     * @return bool True em caso de sucesso.
     */
    public function deleteReminder($reminderId) { /* ... */ }

    /**
     * Salva um token de redefinição de senha para um usuário.
     * Remove tokens antigos para o mesmo usuário antes de inserir um novo.
     * @param int $userId ID do usuário.
     * @param string $token O token de redefinição.
     * @param string $expiresAt Data e hora de expiração do token.
     * @return int O ID do token salvo.
     */
    public function savePasswordResetToken($userId, $token, $expiresAt) { /*
... */ }

    /**
     * Obtém um token de redefinição de senha válido e não expirado.
     * @param string $token O token a ser buscado.
     * @return array|false Os dados do token, ou false se não encontrado ou
expirado.
```

```
     */
    public function getPasswordResetToken($token) { /* ... */ }

    /**
     * Atualiza a senha de um usuário.
     * @param int $userId ID do usuário.
     * @param string $newPasswordHash O hash da nova senha.
     * @return bool True em caso de sucesso.
     */
    public function updatePassword($userId, $newPasswordHash) { /* ... */ }

    /**
     * Exclui um token de redefinição de senha.
     * @param string $token O token a ser excluído.
     * @return bool True em caso de sucesso.
     */
    public function deletePasswordResetToken($token) { /* ... */ }
}
```

Plan.php

O modelo `Plan` gerencia os planos de treino e dieta gerados pela inteligência artificial para os usuários. Ele interage com as tabelas `plans`, `exercises` e `meals` no banco de dados, sendo responsável pela persistência e recuperação desses dados estruturados.

Propriedades Principais

- `protected $table = 'plans';` : Define a tabela principal associada ao modelo.
- `protected $primaryKey = 'id';` : Define a chave primária da tabela.
- `protected $fillable` : Array de campos que podem ser preenchidos em massa (`user_id`, `type`, `title`, `content`, `status`).
- `protected $timestamps = true;` : Indica que o modelo gerencia automaticamente `created_at` e `updated_at`.

Métodos Principais

PHP

```
class Plan extends Model {
    /**
     * Obtém todos os planos de um usuário, opcionalmente filtrados por tipo
(treino ou dieta).
     * O conteúdo JSON dos planos é decodificado automaticamente.
     * @param int $userId ID do usuário.
```

```
     * @param string|null $type Tipo de plano ('treino' ou 'dieta').
     * @return array Array de planos.
     */
    public function getUserPlans($userId, $type = null) { /* ... */ }

    /**
     * Obtém o plano ativo mais recente de um tipo específico para um
usuário.
     * @param int $userId ID do usuário.
     * @param string $type Tipo de plano ('treino' ou 'dieta').
     * @return array|null O plano ativo, ou null se nenhum for encontrado.
     */
    public function getActivePlan($userId, $type) { /* ... */ }

    /**
     * Cria um novo plano (treino ou dieta) para um usuário.
     * Desativa planos anteriores do mesmo tipo para garantir que apenas um
plano esteja ativo por vez.
     * Se for um plano de treino, cria os exercícios detalhados. Se for de
dieta, cria as refeições detalhadas.
     * @param int $userId ID do usuário.
     * @param string $type Tipo de plano ('treino' ou 'dieta').
     * @param string $title Título do plano.
     * @param array $content Conteúdo detalhado do plano (exercícios ou
refeições).
     * @return int O ID do plano recém-criado.
     */
    public function createPlan($userId, $type, $title, $content) { /* ... */
}

    /**
     * Métodos privados para criar exercícios e refeições associados a um
plano.
     * `createExercises($planId, $exercises)`: Salva os exercícios na tabela
`exercises`.
     * `createMeals($planId, $meals)`: Salva as refeições na tabela `meals`.
     */
    private function createExercises($planId, $exercises) { /* ... */ }
    private function createMeals($planId, $meals) { /* ... */ }

    /**
     * Obtém os exercícios de um plano, opcionalmente filtrados por dia da
semana.
     * @param int $planId ID do plano.
     * @param string|null $dayOfWeek Dia da semana (ex: 'Segunda-feira').
     * @return array Array de exercícios.
     */
    public function getExercises($planId, $dayOfWeek = null) { /* ... */ }
```

```
    /**
     * Obtém as refeições de um plano, opcionalmente filtradas por tipo de
refeição.
     * Os ingredientes JSON são decodificados automaticamente.
     * @param int $planId ID do plano.
     * @param string|null $mealType Tipo de refeição (ex: 'cafe_manha').
     * @return array Array de refeições.
     */
    public function getMeals($planId, $mealType = null) { /* ... */ }

    /**
     * Atualiza o status de um plano (ex: 'ativo', 'inativo').
     * @param int $planId ID do plano.
     * @param string $status Novo status.
     * @return bool True em caso de sucesso.
     */
    public function updatePlanStatus($planId, $status) { /* ... */ }

    /**
     * Exclui um plano e todos os exercícios e refeições associados a ele.
     * @param int $planId ID do plano a ser excluído.
     * @return bool True em caso de sucesso.
     */
    public function deletePlan($planId) { /* ... */ }

    /**
     * Obtém o treino semanal completo para um usuário, organizado por dia
da semana.
     * @param int $userId ID do usuário.
     * @return array|null Array contendo o plano ativo e os exercícios
semanais, ou null se não houver plano ativo.
     */
    public function getWeeklyWorkout($userId) { /* ... */ }

    /**
     * Obtém as refeições diárias para um usuário, juntamente com os totais
de calorias e macronutrientes.
     * @param int $userId ID do usuário.
     * @return array|null Array contendo o plano ativo, as refeições e os
totais nutricionais, ou null se não houver plano ativo.
     */
    public function getDailyMeals($userId) { /* ... */ }

    /**
     * Obtém estatísticas sobre os planos de um usuário (total, ativos,
treino, dieta).
     * @param int $userId ID do usuário.
```

```
     * @return array Array associativo com as estatísticas dos planos.
     */
    public function getPlanStats($userId) { /* ... */ }
}
```

Message.php

O modelo `Message` é responsável por gerenciar todas as interações de chat entre os usuários e a inteligência artificial. Ele armazena as mensagens enviadas pelos usuários e as respostas geradas pela IA, além de fornecer métodos para recuperar o histórico de conversas e estatísticas.

Propriedades Principais

- `protected $table = 'messages';` : Define a tabela principal associada ao modelo.
- `protected $primaryKey = 'id';` : Define a chave primária da tabela.
- `protected $fillable` : Array de campos que podem ser preenchidos em massa (`user_id` , `message` , `response` , `type` , `context`).
- `protected $timestamps = true;` : Indica que o modelo gerencia automaticamente `created_at` e `updated_at` .

Métodos Principais

PHP

```
class Message extends Model {
    /**
     * Obtém as mensagens de um usuário, limitando a quantidade.
     * As mensagens são retornadas em ordem cronológica inversa (mais
recentes primeiro) e o contexto JSON é decodificado.
     * @param int $userId ID do usuário.
     * @param int $limit Limite de mensagens a serem retornadas (padrão: 50).
     * @return array Array de mensagens.
     */
    public function getUserMessages($userId, $limit = 50) { /* ... */ }

    /**
     * Salva uma mensagem enviada pelo usuário.
     * @param int $userId ID do usuário.
     * @param string $message Conteúdo da mensagem do usuário.
     * @param array|null $context Contexto adicional da mensagem (opcional).
     * @return int O ID da mensagem salva.
     */
    public function saveUserMessage($userId, $message, $context = null) { /*
```

```
... */ }

    /**
     * Salva a resposta gerada pelo bot (IA).
     * @param int $userId ID do usuário.
     * @param string $response Conteúdo da resposta do bot.
     * @param array|null $context Contexto adicional da resposta (opcional).
     * @return int O ID da resposta salva.
     */
    public function saveBotResponse($userId, $response, $context = null) {
/* ... */ }

    /**
     * Salva uma conversa completa (mensagem do usuário e resposta do bot)
em uma única transação.
     * @param int $userId ID do usuário.
     * @param string $userMessage Mensagem do usuário.
     * @param string $botResponse Resposta do bot.
     * @param array|null $context Contexto da conversa (opcional).
     * @return bool True em caso de sucesso.
     * @throws Exception Em caso de erro na transação.
     */
    public function saveConversation($userId, $userMessage, $botResponse,
$context = null) { /* ... */ }

    /**
     * Obtém o histórico de conversas formatado para exibição.
     * @param int $userId ID do usuário.
     * @param int $limit Limite de mensagens a serem retornadas (padrão: 50).
     * @return array Array de mensagens formatadas.
     */
    public function getConversationHistory($userId, $limit = 50) { /* ... */
}

    /**
     * Obtém estatísticas sobre as mensagens de um usuário (total, mensagens
do usuário, respostas do bot, conversas).
     * @param int $userId ID do usuário.
     * @return array Array associativo com as estatísticas das mensagens.
     */
    public function getMessageStats($userId) { /* ... */ }

    /**
     * Exclui todas as mensagens de um usuário.
     * @param int $userId ID do usuário.
     * @return bool True em caso de sucesso.
     */
    public function deleteUserMessages($userId) { /* ... */ }
```

```
    /**
     * Exclui mensagens antigas do sistema com base em um número de dias.
     * @param int $days Número de dias para considerar mensagens antigas
(padrão: 90).
     * @return bool True em caso de sucesso.
     */
    public function deleteOldMessages($days = 90) { /* ... */ }

    /**
     * Pesquisa mensagens de um usuário com base em uma query.
     * @param int $userId ID do usuário.
     * @param string $query Termo de busca.
     * @param int $limit Limite de resultados (padrão: 20).
     * @return array Array de mensagens que correspondem à busca.
     */
    public function searchMessages($userId, $query, $limit = 20) { /* ... */
}

    /**
     * Obtém os tópicos mais populares das mensagens dos usuários.
     * @param int|null $userId ID do usuário (opcional, para filtrar por
usuário).
     * @param int $limit Limite de tópicos a serem retornados (padrão: 10).
     * @return array Array de tópicos populares com suas frequências.
     */
    public function getPopularTopics($userId = null, $limit = 10) { /* ...
*/ }

    /**
     * Obtém a atividade recente de um usuário com base em um período de
horas.
     * @param int $userId ID do usuário.
     * @param int $hours Período em horas (padrão: 24).
     * @return array Array com a contagem de mensagens e a última atividade.
     */
    public function getRecentActivity($userId, $hours = 24) { /* ... */ }
}
```

Database.php

A classe `Database` é o componente central para todas as interações com o banco de dados MySQL no sistema Vitexa. Ela implementa o padrão Singleton para garantir uma única instância de conexão com o banco de dados, otimizando recursos e evitando múltiplas conexões desnecessárias. Utiliza PDO (PHP Data Objects) para operações seguras e eficientes, prevenindo ataques de SQL Injection através do uso de prepared statements.

Propriedades Principais

- `private static $instance = null;` : Armazena a única instância da classe `Database`.
- `private $connection;` : Armazena o objeto de conexão PDO.

Construtor (`__construct()`)

O construtor privado inicializa a conexão PDO. Ele utiliza as constantes de configuração definidas em `config/config.php` (DB_HOST, DB_NAME, DB_USER, DB_PASS, DB_CHARSET) para estabelecer a conexão. Configura o PDO para lançar exceções em caso de erro (`PDO::ERRMODE_EXCEPTION`), definir o modo de busca padrão como array associativo (`PDO::FETCH_ASSOC`) e desabilitar a emulação de prepared statements para maior segurança e performance (`PDO::ATTR_EMULATE_PREPARES => false`).

Métodos Principais

PHP

```
class Database {
    /**
     * Retorna a única instância da classe Database (padrão Singleton).
     * Se a instância ainda não existir, ela é criada.
     * @return Database A instância única da classe Database.
     */
    public static function getInstance() { /* ... */ }

    /**
     * Retorna o objeto de conexão PDO subjacente.
     * @return PDO O objeto PDO da conexão com o banco de dados.
     */
    public function getConnection() { /* ... */ }

    /**
     * Executa uma consulta SQL genérica com prepared statements.
     * Ideal para operações INSERT, UPDATE, DELETE ou SELECT que não
precisam de retorno específico.
     * @param string $sql A string SQL a ser executada.
     * @param array $params Um array associativo de parâmetros para o
prepared statement.
     * @return PDOStatement O objeto PDOStatement resultante da execução da
consulta.
     * @throws Exception Em caso de erro na consulta ao banco de dados.
     */
    public function query($sql, $params = []) { /* ... */ }

    /**
```

```
     * Executa uma consulta SELECT e retorna a primeira linha do resultado.
     * @param string $sql A string SQL SELECT.
     * @param array $params Parâmetros para o prepared statement.
     * @return array|false A primeira linha do resultado como um array
associativo, ou false se não houver resultados.
     */
    public function fetch($sql, $params = []) { /* ... */ }

    /**
     * Executa uma consulta SELECT e retorna todas as linhas do resultado.
     * @param string $sql A string SQL SELECT.
     * @param array $params Parâmetros para o prepared statement.
     * @return array Um array de arrays associativos, representando todas as
linhas do resultado.
     */
    public function fetchAll($sql, $params = []) { /* ... */ }

    /**
     * Insere uma nova linha em uma tabela.
     * Constrói automaticamente a query INSERT a partir dos dados fornecidos.
     * @param string $table O nome da tabela.
     * @param array $data Um array associativo onde as chaves são os nomes
das colunas e os valores são os dados a serem inseridos.
     * @return string O ID da última linha inserida.
     */
    public function insert($table, $data) { /* ... */ }

    /**
     * Atualiza linhas em uma tabela.
     * Constrói automaticamente a query UPDATE a partir dos dados e da
cláusula WHERE.
     * @param string $table O nome da tabela.
     * @param array $data Um array associativo com os dados a serem
atualizados.
     * @param string $where A cláusula WHERE (ex: "id = :id").
     * @param array $whereParams Parâmetros para a cláusula WHERE.
     * @return PDOStatement O objeto PDOStatement resultante.
     */
    public function update($table, $data, $where, $whereParams = []) { /*
... */ }

    /**
     * Exclui linhas de uma tabela.
     * @param string $table O nome da tabela.
     * @param string $where A cláusula WHERE (ex: "id = :id").
     * @param array $params Parâmetros para a cláusula WHERE.
     * @return PDOStatement O objeto PDOStatement resultante.
     */
```

```
    public function delete($table, $where, $params = []) { /* ... */ }

    /**
     * Inicia uma transação no banco de dados.
     * @return bool True em caso de sucesso.
     */
    public function beginTransaction() { /* ... */ }

    /**
     * Confirma uma transação pendente.
     * @return bool True em caso de sucesso.
     */
    public function commit() { /* ... */ }

    /**
     * Reverte uma transação pendente.
     * @return bool True em caso de sucesso.
     */
    public function rollback() { /* ... */ }
}
```

Router.php

A classe `Router` é o coração do sistema de roteamento do Vitexa, responsável por mapear as requisições HTTP de entrada para os controladores e ações apropriados. Ela oferece uma API fluente para definir rotas e suporta middlewares para pré-processamento de requisições, como autenticação e validação de CSRF.

Propriedades Principais

- `private $routes = [];`: Um array que armazena todas as rotas registradas na aplicação.
- `private $middlewares = [];`: Um array (atualmente não utilizado diretamente, mas pode ser expandido para middlewares globais).

Métodos de Definição de Rotas

Os métodos `get()`, `post()`, `put()`, e `delete()` são a interface principal para registrar rotas para os respectivos métodos HTTP. Todos eles chamam o método privado `addRoute()`.

```php
class Router {
```

```
    /**
     * Registra uma rota GET.
     * @param string $path O caminho da URL (pode incluir parâmetros como
'/users/{id}').
     * @param string|callable $callback A string 'Controller@method' ou uma
função anônima.
     * @param array $middlewares Um array de classes de middleware a serem
executadas antes do callback.
     */
    public function get($path, $callback, $middlewares = []) { /* ... */ }

    /**
     * Registra uma rota POST.
     * @param string $path O caminho da URL.
     * @param string|callable $callback A string 'Controller@method' ou uma
função anônima.
     * @param array $middlewares Um array de classes de middleware.
     */
    public function post($path, $callback, $middlewares = []) { /* ... */ }

    /**
     * Registra uma rota PUT.
     * @param string $path O caminho da URL.
     * @param string|callable $callback A string 'Controller@method' ou uma
função anônima.
     * @param array $middlewares Um array de classes de middleware.
     */
    public function put($path, $callback, $middlewares = []) { /* ... */ }

    /**
     * Registra uma rota DELETE.
     * @param string $path O caminho da URL.
     * @param string|callable $callback A string 'Controller@method' ou uma
função anônima.
     * @param array $middlewares Um array de classes de middleware.
     */
    public function delete($path, $callback, $middlewares = []) { /* ... */ }

    /**
     * Adiciona uma rota ao array interno de rotas.
     * Método privado chamado pelos métodos públicos (get, post, etc.).
     */
    private function addRoute($method, $path, $callback, $middlewares = [])
{ /* ... */ }
}
```

Métodos de Resolução de Rotas

PHP

```
class Router {
    /**
     * Resolve a requisição HTTP atual, encontrando a rota correspondente e
executando seus middlewares e callback.
     * Este é o método principal que inicia o processo de roteamento.
     */
    public function resolve() { /* ... */ }

    /**
     * Obtém o caminho da URL da requisição, removendo a query string e o
subdiretório base da aplicação.
     * Garante que o caminho sempre comece com '/'.
     * @return string O caminho da URL limpo.
     */
    private function getPath() { /* ... */ }

    /**
     * Compara o caminho da rota definida com o caminho da requisição,
suportando parâmetros na URL.
     * Converte o caminho da rota em uma expressão regular para
correspondência.
     * @param string $routePath O caminho da rota definida (ex:
'/users/{id}').
     * @param string $requestPath O caminho da URL da requisição (ex:
'/users/123').
     * @return bool True se os caminhos corresponderem, false caso contrário.
     */
    private function matchPath($routePath, $requestPath) { /* ... */ }

    /**
     * Executa o callback associado a uma rota correspondente.
     * Suporta callbacks como string ('Controller@method') ou funções
anônimas.
     * Extrai parâmetros da URL e os passa para o callback.
     * @param string|callable $callback O callback a ser executado.
     * @param string $requestPath O caminho da URL da requisição.
     * @param string $routePath O caminho da rota definida.
     * @return mixed O resultado da execução do callback.
     */
    private function executeCallback($callback, $requestPath, $routePath) {
/* ... */ }

    /**
     * Extrai os valores dos parâmetros de uma URL com base na definição da
rota.
     * Por exemplo, para a rota '/users/{id}' e requisição '/users/123',
```

```
retorna ['id' => '123'].
     * @param string $routePath O caminho da rota definida.
     * @param string $requestPath O caminho da URL da requisição.
     * @return array Um array associativo com os nomes dos parâmetros e seus
valores.
     */
    private function extractParams($routePath, $requestPath) { /* ... */ }

    /**
     * Executa um middleware associado a uma rota.
     * Suporta middlewares como string (nome da classe) ou funções anônimas.
     * @param string|callable $middleware O middleware a ser executado.
     * @return bool True se o middleware permitir a continuação da
requisição, false caso contrário.
     */
    private function executeMiddleware($middleware) { /* ... */ }

    /**
     * Lida com rotas não encontradas (erro 404).
     * Define o código de status HTTP 404 e exibe uma mensagem de erro.
     */
    private function notFound() { /* ... */ }

    /**
     * Redireciona o navegador para uma nova URL.
     * @param string $url A URL para a qual redirecionar.
     * @param int $statusCode O código de status HTTP para o
redirecionamento (padrão: 302).
     */
    public function redirect($url, $statusCode = 302) { /* ... */ }
}
```

Controller.php

A classe `Controller` serve como a base para todos os controladores da aplicação Vitexa, fornecendo um conjunto de métodos utilitários e funcionalidades comuns que simplificam o desenvolvimento e garantem consistência. Ela abstrai operações como renderização de views, manipulação de respostas JSON, redirecionamentos, sanitização e validação de inputs, gerenciamento de tokens CSRF, verificação de autenticação e manipulação de mensagens flash.

Propriedades Principais

- `protected $view;`: Uma instância da classe `View`, utilizada para renderizar templates.
- `protected $session;`: Uma instância da classe `Session`, utilizada para gerenciar sessões e mensagens flash.

Construtor (`__construct()`)

O construtor da classe `Controller` inicializa as instâncias de `View` e `Session`, tornando-as disponíveis para todos os controladores que herdam desta classe.

Métodos Principais

```php
class Controller {
    /**
     * Renderiza um template de view, passando dados para ele.
     * @param string $template O caminho do template da view (ex: 'dashboard/index').
     * @param array $data Um array associativo de dados a serem passados para a view.
     * @return string O conteúdo HTML renderizado da view.
     */
    protected function render($template, $data = []) { /* ... */ }

    /**
     * Envia uma resposta JSON para o cliente e encerra a execução.
     * Define o cabeçalho Content-Type como 'application/json' e o código de status HTTP.
     * @param array $data O array de dados a ser codificado como JSON.
     * @param int $statusCode O código de status HTTP da resposta (padrão: 200).
     */
    protected function json($data, $statusCode = 200) { /* ... */ }

    /**
     * Redireciona o navegador para uma URL especificada e encerra a execução.
     * @param string $url A URL para a qual redirecionar.
     * @param int $statusCode O código de status HTTP para o redirecionamento (padrão: 302).
     */
    protected function redirect($url, $statusCode = 302) { /* ... */ }

    /**
     * Obtém um valor de input (POST ou GET) de forma segura.
     * @param string $key A chave do input a ser recuperado.
     * @param mixed $default O valor padrão a ser retornado se a chave não
```

```
existir (padrão: null).
     * @return mixed O valor do input ou o valor padrão.
     */
    protected function input($key, $default = null) { /* ... */ }

    /**
     * Valida um array de dados com base em um conjunto de regras.
     * @param array $data O array de dados a ser validado.
     * @param array $rules Um array associativo onde as chaves são os nomes
dos campos e os valores são as regras de validação (ex:
'required|email|min:6').
     * @return array Um array de erros de validação, onde as chaves são os
nomes dos campos e os valores são arrays de mensagens de erro.
     */
    protected function validate($data, $rules) { /* ... */ }

    /**
     * Método privado auxiliar para validar um único campo com base em uma
regra específica.
     * @param string $field O nome do campo a ser validado.
     * @param mixed $value O valor do campo.
     * @param string $rule A regra de validação (ex: 'required', 'email',
'min:6').
     * @return string|null Uma mensagem de erro se a validação falhar, ou
null se for bem-sucedida.
     */
    private function validateField($field, $value, $rule) { /* ... */ }

    /**
     * Sanitiza dados de entrada para prevenir ataques como XSS.
     * Aplica htmlspecialchars() e trim() recursivamente em arrays.
     * @param mixed $data Os dados a serem sanitizados (string ou array).
     * @return mixed Os dados sanitizados.
     */
    protected function sanitize($data) { /* ... */ }

    /**
     * Gera e armazena um token CSRF na sessão, se ainda não existir.
     * @return string O token CSRF gerado ou existente.
     */
    protected function generateCsrfToken() { /* ... */ }

    /**
     * Verifica se um token CSRF fornecido corresponde ao token armazenado
na sessão.
     * @param string $token O token CSRF a ser verificado.
     * @return bool True se os tokens corresponderem, false caso contrário.
     */
```

```
    protected function verifyCsrfToken($token) { /* ... */ }

    /**
     * Garante que o usuário esteja autenticado. Se não estiver, redireciona
para a página de login.
     */
    protected function requireAuth() { /* ... */ }

    /**
     * Obtém os dados do usuário atualmente logado a partir da sessão.
     * @return array|null Os dados do usuário, ou null se nenhum usuário
estiver logado.
     */
    protected function getCurrentUser() { /* ... */ }

    /**
     * Verifica se há um usuário autenticado na sessão.
     * @return bool True se um usuário estiver autenticado, false caso
contrário.
     */
    protected function isAuthenticated() { /* ... */ }

    /**
     * Adiciona uma mensagem flash à sessão para ser exibida na próxima
requisição.
     * @param string $type O tipo da mensagem (ex: 'success', 'error',
'warning', 'info').
     * @param string $message O conteúdo da mensagem.
     */
    protected function flashMessage($type, $message) { /* ... */ }

    /**
     * Obtém e limpa todas as mensagens flash da sessão.
     * @return array Um array associativo de mensagens flash, agrupadas por
tipo.
     */
    protected function getFlashMessages() { /* ... */ }
}
```

Configurações Globais e Variáveis de Ambiente

O arquivo `config/config.php` é o ponto central para todas as configurações da aplicação Vitexa. Ele define constantes que controlam o comportamento do sistema, desde detalhes do banco de dados até integrações com serviços

externos e configurações de segurança. A maioria dessas configurações pode ser sobrescrita por variáveis de ambiente (`.env`), facilitando a gestão de diferentes ambientes (desenvolvimento, produção).

Estrutura do `config/config.php`

```php
<?php

// Configurações básicas da aplicação
define(\'APP_NAME\', $_ENV[\'APP_NAME\'] ?? \'Vitexa\');
define(\'APP_VERSION\', $_ENV[\'APP_VERSION\'] ?? \'1.0.0\');
define(\'APP_URL\', $_ENV[\'APP_URL\'] ?? \'http://vitexa.local\');
define(\'APP_ENV\', $_ENV[\'APP_ENV\'] ?? \'development\');
define(\'APP_DEBUG\', $_ENV[\'APP_DEBUG\'] ?? true);

// Configurações do banco de dados
define(\'DB_HOST\', $_ENV[\'DB_HOST\'] ?? \'localhost\');
define(\'DB_NAME\', $_ENV[\'DB_NAME\'] ?? \'vitexa_db\');
define(\'DB_USER\', $_ENV[\'DB_USER\'] ?? \'root\');
define(\'DB_PASS\', $_ENV[\'DB_PASS\'] ?? \'\');
define(\'DB_CHARSET\', \'utf8mb4\');

// Configurações de segurança
define(\'CSRF_TOKEN_NAME\', \'_token\');
define(\'SESSION_LIFETIME\', 3600); // 1 hora
define(\'PASSWORD_MIN_LENGTH\', 8);

// Configurações da API OpenAI
define(\'OPENAI_API_KEY\', $_ENV[\'OPENAI_API_KEY\'] ??
\'sua_chave_openai\');
define(\'OPENAI_API_BASE\', $_ENV[\'OPENAI_API_BASE\'] ??
\'https://api.openai.com/v1\');

// Configurações de Cache
define(\'CACHE_ENABLED\', $_ENV[\'CACHE_ENABLED\'] ?? true);
define(\'CACHE_TTL\', $_ENV[\'CACHE_TTL\'] ?? 3600); // 1 hora
define(\'CACHE_DIR\', $_ENV[\'CACHE_DIR\'] ?? \'/tmp/vitexa_cache/\');

// Configurações de Email/SMTP
define(\'MAIL_ENABLED\', $_ENV[\'MAIL_ENABLED\'] ?? true);
define(\'MAIL_FROM\', $_ENV[\'MAIL_FROM\'] ?? \'contato@vitexa.com.br\');
define(\'MAIL_FROM_NAME\', $_ENV[\'MAIL_FROM_NAME\'] ?? \'Vitexa\');
define(\'MAIL_HOST\', $_ENV[\'MAIL_HOST\'] ?? \'br55-pl.valueserver.net\');
define(\'MAIL_PORT\', $_ENV[\'MAIL_PORT\'] ?? 465);
define(\'MAIL_USERNAME\', $_ENV[\'MAIL_USERNAME\'] ??
\'contato@vitexa.com.br\');
define(\'MAIL_PASSWORD\', $_ENV[\'MAIL_PASSWORD\'] ?? \'aaa222\');
```

```
define(\'MAIL_ENCRYPTION\', $_ENV[\'MAIL_ENCRYPTION\'] ?? \'ssl\'); // tls,
ssl ou null

// Configurações de Push Notifications
define(\'PUSH_ENABLED\', $_ENV[\'PUSH_ENABLED\'] ?? false);
define(\'FIREBASE_SERVER_KEY\', $_ENV[\'FIREBASE_SERVER_KEY\'] ?? \'\');

// Configurações de Webhook
define(\'WEBHOOK_URL\', $_ENV[\'WEBHOOK_URL\'] ?? \'\');

// Configurações de Cron Jobs
define(\'CRON_TOKEN\', $_ENV[\'CRON_TOKEN\'] ?? \'vitexa_cron_2024\');

// Configurações de Upload
define(\'UPLOAD_MAX_SIZE\', 5 * 1024 * 1024); // 5MB
define(\'UPLOAD_ALLOWED_TYPES\', [\'jpg\', \'jpeg\', \'png\', \'gif\']);

// Configurações de Rate Limiting
define(\'RATE_LIMIT_ENABLED\', $_ENV[\'RATE_LIMIT_ENABLED\'] ?? true);
define(\'RATE_LIMIT_REQUESTS\', $_ENV[\'RATE_LIMIT_REQUESTS\'] ?? 100); //
Requests por hora
define(\'RATE_LIMIT_WINDOW\', $_ENV[\'RATE_LIMIT_WINDOW\'] ?? 3600); // 1
hora

// Configurações de Log
define(\'LOG_ENABLED\', $_ENV[\'LOG_ENABLED\'] ?? true);
define(\'LOG_LEVEL\', $_ENV[\'LOG_LEVEL\'] ?? \'info\'); // debug, info,
warning, error
define(\'LOG_DIR\', $_ENV[\'LOG_DIR\'] ?? dirname(__DIR__) . \'/logs/\');

// Timezone
date_default_timezone_set($_ENV[\'TIMEZONE\'] ?? \'America/Sao_Paulo\');

// Headers de segurança
if (!headers_sent()) {
    // Proteção XSS
    header(\'X-Content-Type-Options: nosniff\');
    header(\'X-Frame-Options: DENY\');
    header(\'X-XSS-Protection: 1; mode=block\');

    // Content Security Policy
    header("Content-Security-Policy: default-src \'self\'; script-src
\'self\' \'unsafe-inline\' \'blob:\' https://cdn.jsdelivr.net
https://cdn.tailwindcss.com; style-src \'self\' \'unsafe-inline\'
https://cdn.jsdelivr.net https://fonts.googleapis.com; font-src \'self\'
https://fonts.gstatic.com; img-src \'self\' data: https:; connect-src
\'self\' https://api.openai.com;");
```

```
    // HSTS (apenas em produção)
    if (APP_ENV === \'production\' && isset($_SERVER[\'HTTPS\'])) {
        header(\'Strict-Transport-Security: max-age=31536000;
includeSubDomains\');
    }
}

// Autoloader simples
spl_autoload_register(function ($class) {
    $directories = [
        dirname(__DIR__) . \'/app/core/\',
        dirname(__DIR__) . \'/app/controllers/\',
        dirname(__DIR__) . \'/app/models/\'
    ];

    foreach ($directories as $directory) {
        $file = $directory . $class . \'.php\';
        if (file_exists($file)) {
            require_once $file;
            return;
        }
    }
});

// Configurações específicas por ambiente
if (APP_ENV === \'development\') {
    error_reporting(E_ALL);
    ini_set(\'display_errors\', 1);
} else {
    error_reporting(0);
    ini_set(\'display_errors\', 0);
    ini_set(\'log_errors\', 1);
    ini_set(\'error_log\', LOG_DIR . \'php_errors.log\');
}
```

Detalhamento das Configurações

Esta seção descreve cada grupo de configurações e suas implicações no comportamento da aplicação.

1. Configurações Básicas da Aplicação

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas

APP_NAME	APP_NAME	Nome da aplicação. Usado no título das páginas e em textos gerais.	Vitexa	
APP_VERSION	APP_VERSION	Versão atual da aplicação.	1.0.0	
APP_URL	APP_URL	URL base da aplicação. Essencial para redirecionamentos e links absolutos.	http://vitexa.local	**IMPORTANTE**: Deve ser configurado corretamente em produção.
APP_ENV	APP_ENV	Ambiente da aplicação (development , production). Afeta o comportamento de debug e logs.	development	
APP_DEBUG	APP_DEBUG	Habilita/desabilita o modo de debug. Em development , erros são exibidos; em production , são logados.	true	**NUNCA** true em produção.

2. Configurações do Banco de Dados

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
DB_HOST	DB_HOST	Host do servidor de banco de dados.	localhost	
DB_NAME	DB_NAME	Nome do banco de dados.	vitexa_db	
DB_USER	DB_USER	Usuário do banco de dados.	root	
DB_PASS	DB_PASS	Senha do usuário do banco de dados.	(vazio)	**IMPORTANTE**: Use uma senha forte em produção.
DB_CHARSET	N/A	Conjunto de caracteres para a conexão com o	utf8mb4	Recomendado para suporte a emojis e

		banco de dados.		caracteres especiais.

3. Configurações de Segurança

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
`CSRF_TOKEN_NAME`	N/A	Nome do campo de formulário para o token CSRF.	`_token`	
`SESSION_LIFETIME`	N/A	Tempo de vida da sessão em segundos.	`3600` (1 hora)	
`PASSWORD_MIN_LENGTH`	N/A	Comprimento mínimo para senhas de usuário.	`8`	

4. Configurações da API OpenAI

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
`OPENAI_API_KEY`	`OPENAI_API_KEY`	Chave da API para acesso aos serviços da OpenAI (GPT-3.5, etc.).	`sk-proj-060000000` (exemplo)	**OBRIGATÓRIO** para funcionalidades de IA.
`OPENAI_API_BASE`	`OPENAI_API_BASE`	URL base da API OpenAI.	`https://api.openai.com/v1`	Pode ser alterado para APIs compatíveis.

5. Configurações de Cache

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
`CACHE_ENABLED`	`CACHE_ENABLED`	Habilita/desabilita o sistema de cache.	`true`	

CACHE_TTL	CACHE_TTL	Tempo de vida do cache em segundos.	3600 (1 hora)	
CACHE_DIR	CACHE_DIR	Diretório onde os arquivos de cache serão armazenados. Deve ter permissões de escrita.	/tmp/vitexa_cache/	

6. Configurações de Email/SMTP

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
MAIL_ENABLED	MAIL_ENABLED	Habilita/desabilita o envio de e-mails.	true	
MAIL_FROM	MAIL_FROM	Endereço de e-mail do remetente.	contato@vitexa.com.br	
MAIL_FROM_NAME	MAIL_FROM_NAME	Nome do remetente.	Vitexa	
MAIL_HOST	MAIL_HOST	Host do servidor SMTP.	br55-pl.valueserver.net	
MAIL_PORT	MAIL_PORT	Porta do servidor SMTP.	465	Comum: 465 (SSL), 587 (TLS).
MAIL_USERNAME	MAIL_USERNAME	Usuário para autenticação SMTP.	contato@vitexa.com.br	
MAIL_PASSWORD	MAIL_PASSWORD	Senha para autenticação SMTP.	aaa222 (exemplo)	**IMPORTANTE**: Use uma senha de aplicativo se for Gmail.
MAIL_ENCRYPTION	MAIL_ENCRYPTION	Tipo de criptografia (ssl , tls ou null).	ssl	

7. Configurações de Push Notifications

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
`PUSH_ENABLED`	`PUSH_ENABLED`	Habilita/desabilita o envio de push notifications.	`false`	Requer integração com um serviço como Firebase.
`FIREBASE_SERVER_KEY`	`FIREBASE_SERVER_KEY`	Chave do servidor Firebase para envio de notificações.	`(vazio)`	**OBRIGATÓRIO** se `PUSH_ENABLED` for `true`.

8. Configurações de Webhook

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
`WEBHOOK_URL`	`WEBHOOK_URL`	URL para onde os webhooks serão enviados.	`(vazio)`	Usado para integrar com serviços externos.

9. Configurações de Cron Jobs

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
`CRON_TOKEN`	`CRON_TOKEN`	Token de segurança para validar requisições de cron jobs.	`vitexa_cron_2024`	Impede acesso não autorizado ao script de cron.

10. Configurações de Upload

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas

`UPLOAD_MAX_SIZE`	N/A	Tamanho máximo permitido para uploads de arquivos em bytes.	`5 * 1024 * 1024` (5MB)	
`UPLOAD_ALLOWED_TYPES`	N/A	Array de extensões de arquivo permitidas para upload.	`["jpg", "jpeg", "png", "gif"]`	

11. Configurações de Rate Limiting

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
`RATE_LIMIT_ENABLED`	`RATE_LIMIT_ENABLED`	Habilita/desabilita o controle de limite de requisições.	`true`	
`RATE_LIMIT_REQUESTS`	`RATE_LIMIT_REQUESTS`	Número máximo de requisições permitidas dentro da janela de tempo.	`100`	
`RATE_LIMIT_WINDOW`	`RATE_LIMIT_WINDOW`	Janela de tempo em segundos para o limite de requisições.	`3600` (1 hora)	

12. Configurações de Log

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
`LOG_ENABLED`	`LOG_ENABLED`	Habilita/desabilita o sistema de log.	`true`	
`LOG_LEVEL`	`LOG_LEVEL`	Nível mínimo de log a ser registrado (`debug` , `info` , `warning` , `error`).	`info`	
`LOG_DIR`	`LOG_DIR`	Diretório onde os arquivos de log serão armazenados.	`dirname(__DIR__) . '/logs/'`	Deve ter permissões de escrita.

13. Timezone

Constante	Variável de Ambiente	Descrição	Valor Padrão	Notas
N/A	`TIMEZONE`	Fuso horário padrão da aplicação.	`America/Sao_Paulo`	Afeta todas as operações de data e hora.

14. Headers de Segurança HTTP

Esta seção no `config.php` define cabeçalhos HTTP para aumentar a segurança da aplicação contra ataques comuns. Eles são enviados apenas se os cabeçalhos ainda não tiverem sido enviados.

- **`X-Content-Type-Options: nosniff`** : Previne que o navegador "adivinhe" o tipo MIME de um arquivo, o que pode levar a ataques de XSS.
- **`X-Frame-Options: DENY`** : Impede que a página seja carregada em um `<iframe>` , `&lt;frame&gt;` ou `&lt;object&gt;` , protegendo contra ataques de Clickjacking.
- **`X-XSS-Protection: 1; mode=block`** : Ativa o filtro de XSS embutido nos navegadores modernos. Se um ataque for detectado, o navegador bloqueará o carregamento da página.
- **`Content-Security-Policy`** ** (CSP)**: Um cabeçalho poderoso que permite controlar quais recursos (scripts, estilos, imagens, etc.) a página pode carregar. A política atual permite:
  - `default-src 'self'` : Apenas recursos do mesmo domínio são permitidos por padrão.
  - `script-src 'self' 'unsafe-inline' 'blob:' https://cdn.jsdelivr.net https://cdn.tailwindcss.com;` : Scripts permitidos de fontes seguras e inline (para Tailwind).
  - `style-src 'self' 'unsafe-inline' https://cdn.jsdelivr.net https://fonts.googleapis.com;` : Estilos permitidos de fontes seguras e inline.
  - `font-src 'self' https://fonts.gstatic.com;` : Fontes permitidas.
  - `img-src 'self' data: https:;` : Imagens permitidas de fontes seguras e dados embutidos.
  - `connect-src 'self' https://api.openai.com;` : Conexões (AJAX, WebSockets) permitidas para o próprio domínio e para a API da OpenAI.
- **`Strict-Transport-Security`** ** (HSTS)**: Força o navegador a usar HTTPS para todas as futuras conexões com o domínio. Ativado apenas em ambiente de produção e quando a requisição é via HTTPS.

15. Autoloader Simples

O `config.php` inclui um autoloader simples que registra uma função para carregar automaticamente as classes dos diretórios `app/core/` , `app/controllers/` e `app/models/` . Isso elimina a necessidade de incluir manualmente cada arquivo de classe usando `require` ou `include` .

16. Configurações Específicas por Ambiente

O arquivo também define configurações de relatório de erros e exibição de erros com base na constante `APP_ENV` :

- **development** : `error_reporting(E_ALL)` e `ini_set('display_errors', 1)` são ativados para facilitar o debug.
- **production** : `error_reporting(0)` e `ini_set('display_errors', 0)` são ativados para ocultar erros do usuário final, e os erros são logados em `LOG_DIR/php_errors.log` .

View.php

A classe `View` é responsável por gerenciar a renderização de templates e layouts na aplicação Vitexa. Ela fornece métodos para carregar views, passar dados para elas e aplicar layouts, garantindo uma separação clara entre a lógica de negócio (controladores) e a apresentação (views).

Propriedades Principais

- `private $templatePath;` : Caminho base para os arquivos de template das views.
- `private $layoutPath;` : Caminho base para os arquivos de layout.
- `private $data = [];` : Array para armazenar os dados que serão passados para as views.

Construtor (`__construct()`)

O construtor inicializa os caminhos para os templates e layouts, baseando-se na localização do próprio arquivo `View.php` .

Métodos Principais

PHP

```
class View {
    /**
     * Renderiza um template de view com dados opcionais e aplica um layout.
     * Adiciona automaticamente dados globais como nome da aplicação, URL,
token CSRF, mensagens flash e usuário atual.
     * @param string $template O nome do template da view a ser renderizado
(ex: 'dashboard/index').
```

```
     * @param array $data Um array associativo de dados a serem passados
para a view.
     * @param string $layout O nome do arquivo de layout a ser aplicado
(padrão: 'app'). Se null, nenhum layout é aplicado.
     * @return string O conteúdo HTML renderizado.
     * @throws Exception Se o template ou layout não for encontrado.
     */
    public function render($template, $data = [], $layout = 'app') { /* ...
*/ }

    /**
     * Renderiza um template de view específico.
     * Método privado auxiliar para `render()`.
     * @param string $template O nome do template.
     * @return string O conteúdo HTML do template.
     * @throws Exception Se o template não for encontrado.
     */
    private function renderTemplate($template) { /* ... */ }

    /**
     * Renderiza um arquivo de layout.
     * Método privado auxiliar para `render()`.
     * @param string $layout O nome do layout.
     * @return string O conteúdo HTML do layout com o template renderizado
inserido.
     * @throws Exception Se o layout não for encontrado.
     */
    private function renderLayout($layout) { /* ... */ }

    /**
     * Define um valor no array de dados da view.
     * @param string $key A chave do dado.
     * @param mixed $value O valor do dado.
     */
    public function set($key, $value) { /* ... */ }

    /**
     * Obtém um valor do array de dados da view.
     * @param string $key A chave do dado.
     * @param mixed $default O valor padrão a ser retornado se a chave não
existir.
     * @return mixed O valor do dado ou o valor padrão.
     */
    public function get($key, $default = null) { /* ... */ }

    /**
     * Verifica se uma chave existe no array de dados da view.
     * @param string $key A chave a ser verificada.
```

```
     * @return bool True se a chave existir, false caso contrário.
     */
    public function has($key) { /* ... */ }

    /**
     * Escapa caracteres especiais HTML para prevenir XSS.
     * @param string $value O valor a ser escapado.
     * @return string O valor escapado.
     */
    public function escape($value) { /* ... */ }

    /**
     * Constrói uma URL completa da aplicação.
     * @param string $path O caminho relativo.
     * @return string A URL completa.
     */
    public function url($path = '') { /* ... */ }

    /**
     * Constrói uma URL para um asset (CSS, JS, imagem).
     * @param string $path O caminho relativo do asset.
     * @return string A URL completa do asset.
     */
    public function asset($path) { /* ... */ }

    /**
     * Recupera um valor de input antigo da sessão (útil para repopular
formulários após erro).
     * @param string $key A chave do input.
     * @param string $default O valor padrão.
     * @return string O valor antigo do input ou o padrão.
     */
    public function old($key, $default = '') { /* ... */ }

    /**
     * Recupera uma mensagem de erro de validação da sessão.
     * @param string $key A chave do erro.
     * @return string|null A mensagem de erro ou null.
     */
    public function error($key) { /* ... */ }

    /**
     * Verifica se existe uma mensagem de erro para uma chave específica.
     * @param string $key A chave do erro.
     * @return bool True se houver erro, false caso contrário.
     */
    public function hasError($key) { /* ... */ }
```

```
    /**
     * Formata uma data para um formato legível.
     * @param string $date A string de data.
     * @param string $format O formato de saída (padrão: 'd/m/Y H:i').
     * @return string A data formatada ou string vazia se a data for
inválida.
     */
    public function formatDate($date, $format = 'd/m/Y H:i') { /* ... */ }

    /**
     * Trunca um texto para um comprimento máximo, adicionando um sufixo.
     * @param string $text O texto a ser truncado.
     * @param int $length O comprimento máximo.
     * @param string $suffix O sufixo a ser adicionado (padrão: '...').
     * @return string O texto truncado.
     */
    public function truncate($text, $length = 100, $suffix = '...') { /* ...
*/ }

    /**
     * Inclui um sub-template dentro de outro template, passando dados
adicionais.
     * Preserva os dados do template pai.
     * @param string $template O nome do sub-template.
     * @param array $data Dados adicionais para o sub-template.
     * @return string O conteúdo HTML do sub-template.
     */
    public function include($template, $data = []) { /* ... */ }

    /**
     * Gera e retorna o token CSRF da sessão.
     * Método privado auxiliar.
     */
    private function generateCsrfToken() { /* ... */ }

    /**
     * Obtém e limpa as mensagens flash da sessão.
     * Método privado auxiliar.
     */
    private function getFlashMessages() { /* ... */ }

    /**
     * Obtém o usuário atualmente logado.
     * Método privado auxiliar.
     */
    private function getCurrentUser() { /* ... */ }
}
```