Documentação Técnica - Escape Room Game

Sumário

Visão Geral

O Escape Room Game é uma aplicação web desenvolvida em Laravel/PHP que simula um jogo de escape room. O sistema permite que os usuários criem personagens, naveguem por diferentes fases, resolvam desafios, interajam com NPCs, coletem itens e enfrentem armadilhas.

A aplicação segue o padrão de arquitetura MVC (Model-View-Controller) do Laravel, com uma clara separação entre a lógica de negócios, a apresentação e o controle de fluxo. O banco de dados utilizado é o SQLite, escolhido pela sua simplicidade e facilidade de implantação.

Arquitetura do Sistema

O sistema segue a arquitetura MVC do Laravel:

- **Models**: Representam as entidades do sistema e suas relações, como User, Character, Phase, Challenge, etc.
- **Views**: Implementadas usando Blade, o sistema de templates do Laravel, com Bootstrap para o frontend.
- **Controllers**: Gerenciam o fluxo de dados entre os models e as views, processando as requisições do usuário.

Além disso, o sistema utiliza:

- **Middlewares**: Para autenticação e controle de acesso.
- **Migrations**: Para versionamento e criação do esquema do banco de dados.
- **Seeders**: Para popular o banco de dados com dados iniciais.
- **Routes**: Para definir os endpoints da aplicação.

Tecnologias Utilizadas

Backend

- **Laravel 10.x**: Framework PHP para desenvolvimento web
- **PHP 8.1**: Linguagem de programação
- **SQLite**: Sistema de gerenciamento de banco de dados
- **Composer**: Gerenciador de dependências para PHP

Frontend

- **Bootstrap 5**: Framework CSS para desenvolvimento responsivo
- **JavaScript/jQuery**: Para interatividade no lado do cliente
- **HTML5/CSS3**: Para estruturação e estilização das páginas
- **Blade**: Sistema de templates do Laravel

Ferramentas de Desenvolvimento

- **Git**: Sistema de controle de versão
- **npm**: Gerenciador de pacotes para JavaScript
- **Laravel Mix**: Para compilação de assets

Estrutura do Banco de Dados

O banco de dados do Escape Room Game é composto pelas seguintes tabelas:

Tabela: users

Armazena informações dos usuários registrados no sistema.

Campo	Tipo	Descrição
id	bigint	Identificador único do usuário
name	varchar	Nome do usuário
email	varchar	Email do usuário (único)
password	varchar	Senha do usuário (hash)
is_admin	boolean	Indica se o usuário é administrador
remember_token	varchar	Token para "lembrar de mim"
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: characters

Armazena informações dos personagens criados pelos usuários.

Campo	Tipo	Descrição
id	bigint	Identificador único do personagem
user_id	bigint	ID do usuário proprietário
name	varchar	Nome do personagem
level	integer	Nível do personagem
experience	integer	Pontos de experiência
health_points	integer	Pontos de vida atuais
max_health_points	integer	Pontos de vida máximos

Campo	Tipo	Descrição
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: phases

Armazena informações das fases do jogo.

Campo	Tipo	Descrição
id	bigint	Identificador único da fase
name	varchar	Nome da fase
description	text	Descrição da fase
order	integer	Ordem da fase no jogo
hint	text	Dica opcional para a fase
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: challenges

Armazena informações dos desafios em cada fase.

Campo	Tipo	Descrição
id	bigint	Identificador único do desafio
phase_id	bigint	ID da fase à qual o desafio pertence
name	varchar	Nome do desafio
description	text	Descrição do desafio
type	varchar	Tipo do desafio (puzzle, code, sequence)
difficulty	varchar	Dificuldade do desafio (fácil, médio, difícil)
points	integer	Pontos de experiência concedidos ao completar

Campo	Tipo	Descrição
order	integer	Ordem do desafio na fase
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: codes

Armazena os códigos de solução para os desafios.

Campo	Tipo	Descrição
id	bigint	Identificador único do código
challenge_id	bigint	ID do desafio ao qual o código pertence
code	varchar	Código de solução
hint	text	Dica opcional para o código
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: npcs

Armazena informações dos NPCs (Non-Player Characters) do jogo.

Campo	Tipo	Descrição
id	bigint	Identificador único do NPC
phase_id	bigint	ID da fase à qual o NPC pertence
name	varchar	Nome do NPC
description	text	Descrição do NPC
dialog	text	Diálogo do NPC
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: items

Armazena informações dos itens do jogo.

Campo	Tipo	Descrição
id	bigint	Identificador único do item
phase_id	bigint	ID da fase onde o item pode ser encontrado
name	varchar	Nome do item
description	text	Descrição do item
type	varchar	Tipo do item (health_potion, key, tool, artifact)
effect_value	integer	Valor do efeito do item (ex: quantidade de cura)
is_consumable	boolean	Indica se o item é consumível
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: inventory

Armazena os itens no inventário de cada personagem.

Campo	Tipo	Descrição
id	bigint	Identificador único do registro de inventário
character_id	bigint	ID do personagem
item_id	bigint	ID do item
quantity	integer	Quantidade do item
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: traps

Armazena informações das armadilhas do jogo.

Campo	Tipo	Descrição
id	bigint	Identificador único da armadilha
phase_id	bigint	ID da fase à qual a armadilha pertence
name	varchar	Nome da armadilha
description	text	Descrição da armadilha
damage	integer	Dano causado pela armadilha
trigger_condition	text	Condição que ativa a armadilha
is_active	boolean	Indica se a armadilha está ativa
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: player_progress

Armazena o progresso dos jogadores nas fases e desafios.

Campo	Tipo	Descrição
id	bigint	Identificador único do registro de progresso
character_id	bigint	ID do personagem
phase_id	bigint	ID da fase (opcional)
challenge_id	bigint	ID do desafio (opcional)
status	varchar	Status (not_started, in_progress, completed)
completed_at	timestamp	Data de conclusão
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Tabela: action_history

Armazena o histórico de ações dos jogadores.

Campo	Tipo	Descrição
id	bigint	Identificador único do registro de ação
character_id	bigint	ID do personagem
action_type	varchar	Tipo de ação (challenge_completed, trap_triggered, etc.)
description	text	Descrição da ação
points	integer	Pontos ganhos/perdidos (opcional)
damage	integer	Dano recebido (opcional)
created_at	timestamp	Data de criação
updated_at	timestamp	Data de atualização

Modelos e Relacionamentos

User

```
class User extends Authenticatable
{
    // Relacionamentos
    public function characters()
    {
        return $this->hasMany(Character::class);
    }
}
```

Character

```
class Character extends Model
{
    // Relacionamentos
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    public function inventory()
    {
        return $this->hasMany(Inventory::class);
    }
```

```
    public function progress()
    {
        return $this->hasMany(PlayerProgress::class);
    }

    public function actionHistory()
    {
        return $this->hasMany(ActionHistory::class);
    }
}
```

Phase

```
class Phase extends Model
{
    // Relacionamentos
    public function challenges()
    {
        return $this->hasMany(Challenge::class);
    }

    public function npcs()
    {
        return $this->hasMany(Npc::class);
    }

    public function items()
    {
        return $this->hasMany(Item::class);
    }

    public function traps()
    {
        return $this->hasMany(Trap::class);
    }
}
```

Challenge

```
class Challenge extends Model
{
    // Relacionamentos
    public function phase()
    {
        return $this->belongsTo(Phase::class);
    }
```

```
    public function code()
    {
        return $this->hasOne(Code::class);
    }
}
```

Code

```
class Code extends Model
{
    // Relacionamentos
    public function challenge()
    {
        return $this->belongsTo(Challenge::class);
    }
}
```

Npc

```
class Npc extends Model
{
    // Relacionamentos
    public function phase()
    {
        return $this->belongsTo(Phase::class);
    }
}
```

Item

```
class Item extends Model
{
    // Relacionamentos
    public function phase()
    {
        return $this->belongsTo(Phase::class);
    }

    public function inventories()
    {
        return $this->hasMany(Inventory::class);
    }
}
```

Inventory

```
class Inventory extends Model
{
    // Relacionamentos
    public function character()
    {
        return $this->belongsTo(Character::class);
    }

    public function item()
    {
        return $this->belongsTo(Item::class);
    }
}
```

Trap

```
class Trap extends Model
{
    // Relacionamentos
    public function phase()
    {
        return $this->belongsTo(Phase::class);
    }
}
```

PlayerProgress

```
class PlayerProgress extends Model
{
    // Relacionamentos
    public function character()
    {
        return $this->belongsTo(Character::class);
    }

    public function phase()
    {
        return $this->belongsTo(Phase::class);
    }

    public function challenge()
    {
        return $this->belongsTo(Challenge::class);
```

```
    }
}
```

ActionHistory

```
class ActionHistory extends Model
{
    // Relacionamentos
    public function character()
    {
        return $this->belongsTo(Character::class);
    }
}
```

Controladores

GameController

Gerencia a exibição das páginas principais do jogo.

```
class GameController extends Controller
{
    public function index()
    {
        // Exibe a página inicial do jogo
    }

    public function showPhase(Phase $phase)
    {
        // Exibe uma fase específica
    }
}
```

CharacterController

Gerencia a criação e manipulação de personagens.

```
class CharacterController extends Controller
{
    public function index()
    {
        // Lista os personagens do usuário
    }
```

```
    public function create()
    {
        // Exibe o formulário de criação de personagem
    }

    public function store(Request $request)
    {
        // Cria um novo personagem
    }

    public function show(Character $character)
    {
        // Exibe os detalhes de um personagem
    }

    public function edit(Character $character)
    {
        // Exibe o formulário de edição de personagem
    }

    public function update(Request $request, Character
$character)
    {
        // Atualiza um personagem
    }

    public function destroy(Character $character)
    {
        // Remove um personagem
    }

    public function inventory(Character $character)
    {
        // Exibe o inventário de um personagem
    }

    public function progress(Character $character)
    {
        // Exibe o progresso de um personagem
    }
}
```

GameplayController

Gerencia as interações do jogador com o jogo.

```
 class GameplayController extends Controller
{
    public function solveChallenge(Request $request,
$challengeId)
```

```
    {
        // Processa a tentativa de resolver um desafio
    }

    public function triggerTrap(Request $request, $trapId)
    {
        // Processa a interação com uma armadilha
    }

    public function talkToNpc($npcId)
    {
        // Processa a interação com um NPC
    }

    public function useItem(Request $request, $itemId)
    {
        // Processa o uso de um item
    }

    public function collectItem($itemId)
    {
        // Processa a coleta de um item
    }

    public function advancePhase($phaseId)
    {
        // Processa o avanço para a próxima fase
    }

    public function reviveCharacter()
    {
        // Processa a revivificação de um personagem
    }
}
```

PhaseController

Gerencia a exibição e manipulação de fases.

```
 class PhaseController extends Controller
{
    public function index()
    {
        // Lista todas as fases
    }

    public function show(Phase $phase)
    {
        // Exibe os detalhes de uma fase
```

```
    }
}
```

AdminController

Gerencia as funcionalidades administrativas.

```
class AdminController extends Controller
{
    public function dashboard()
    {
        // Exibe o painel administrativo
    }

    // Outros métodos administrativos
}
```

Rotas

Rotas Públicas

```
// Rota inicial
Route::get('/', function () {
    return view('welcome');
});

// Rotas de autenticação
Auth::routes();
```

Rotas Protegidas por Autenticação

```
Route::middleware(['auth'])->group(function () {
    // Rotas de personagens
    Route::resource('characters', CharacterController::class);
    Route::get('characters/{character}/inventory',
[CharacterController::class, 'inventory'])-
>name('characters.inventory');
    Route::get('characters/{character}/progress',
[CharacterController::class, 'progress'])-
>name('characters.progress');

    // Rotas de fases
    Route::resource('phases', PhaseController::class);
```

```
    // Rotas de jogo
    Route::get('game', [GameController::class, 'index'])-
>name('game.index');
    Route::get('game/phase/{phase}', [GameController::class,
'showPhase'])->name('game.phase');

    // Rotas de gameplay
    Route::post('gameplay/challenge/{challenge}/solve',
[GameplayController::class, 'solveChallenge'])-
>name('gameplay.solve_challenge');
    Route::post('gameplay/trap/{trap}/trigger',
[GameplayController::class, 'triggerTrap'])-
>name('gameplay.trigger_trap');
    Route::get('gameplay/npc/{npc}/talk',
[GameplayController::class, 'talkToNpc'])-
>name('gameplay.talk_to_npc');
    Route::post('gameplay/item/{item}/use',
[GameplayController::class, 'useItem'])-
>name('gameplay.use_item');
    Route::post('gameplay/item/{item}/collect',
[GameplayController::class, 'collectItem'])-
>name('gameplay.collect_item');
    Route::post('gameplay/phase/{phase}/advance',
[GameplayController::class, 'advancePhase'])-
>name('gameplay.advance_phase');
    Route::post('gameplay/character/revive',
[GameplayController::class, 'reviveCharacter'])-
>name('gameplay.revive_character');
});
```

Rotas Administrativas

```
Route::middleware(['auth', 'admin'])->prefix('admin')-
>group(function () {
    Route::get('dashboard', [AdminController::class,
'dashboard'])->name('admin.dashboard');
    Route::resource('challenges', ChallengeController::class);
    Route::resource('items', ItemController::class);
    // Outras rotas administrativas
});
```

Middlewares

AdminMiddleware

Verifica se o usuário é um administrador.

```
 class AdminMiddleware
{
    public function handle(Request $request, Closure $next)
    {
        if (!auth()->check() || !auth()->user()->is_admin) {
            return redirect('/')->with('error', 'Acesso não
autorizado.');
        }

        return $next($request);
    }
}
```

Frontend

Layouts

O sistema utiliza um layout principal (`app.blade.php`) que define a estrutura básica das páginas, incluindo o menu de navegação e o rodapé.

Views

As principais views do sistema são:

- **welcome.blade.php**: Página inicial do jogo
- **home.blade.php**: Dashboard do usuário após login
- **characters/index.blade.php**: Lista de personagens do usuário
- **characters/create.blade.php**: Formulário de criação de personagem
- **characters/show.blade.php**: Detalhes de um personagem
- **characters/edit.blade.php**: Formulário de edição de personagem
- **characters/inventory.blade.php**: Inventário de um personagem
- **characters/progress.blade.php**: Progresso de um personagem
- **game/index.blade.php**: Página principal do jogo
- **game/phase.blade.php**: Página de uma fase específica
- **phases/index.blade.php**: Lista de todas as fases

Componentes

O sistema utiliza componentes Blade para elementos reutilizáveis:

- **character-card.blade.php**: Card de personagem
- **challenge-card.blade.php**: Card de desafio
- **npc-card.blade.php**: Card de NPC

- **item-card.blade.php**: Card de item
- **trap-card.blade.php**: Card de armadilha

JavaScript e AJAX

O sistema utiliza JavaScript e AJAX para proporcionar uma experiência interativa sem recarregar a página:

game.js

Arquivo principal de JavaScript que gerencia as interações do jogo:

```
// Inicialização
document.addEventListener('DOMContentLoaded', function() {
    // Inicializar tooltips do Bootstrap
    var tooltipTriggerList =
[].slice.call(document.querySelectorAll('[data-bs-
toggle="tooltip"]'));
    tooltipTriggerList.map(function (tooltipTriggerEl) {
        return new bootstrap.Tooltip(tooltipTriggerEl);
    });

    // Obter IDs da página atual
    currentPhaseId = document.getElementById('phase-
container')?.dataset.phaseId;
    currentCharacterId = document.getElementById('character-
info')?.dataset.characterId;

    // Inicializar barras de vida
    updateHealthBar();

    // Adicionar listeners para modais
    setupModalListeners();
});

// Funções para interagir com a API do jogo
function solveChallenge(challengeId, code) {
    // Envia o código para o servidor via AJAX
}

function triggerTrap(trapId) {
    // Envia a interação com a armadilha para o servidor via
AJAX
}

function talkToNpc(npcId) {
    // Carrega os dados do NPC via AJAX
}
```

```
function useItem(itemId) {
    // Envia o uso do item para o servidor via AJAX
}

function collectItem(itemId) {
    // Envia a coleta do item para o servidor via AJAX
}

function advancePhase() {
    // Envia a solicitação de avanço de fase para o servidor via
AJAX
}

function reviveCharacter() {
    // Envia a solicitação de revivificação para o servidor via
AJAX
}

// Funções auxiliares
function updateHealthBar() {
    // Atualiza a barra de vida do personagem
}

function showAlert(message, type) {
    // Exibe um alerta na tela
}

function setupModalListeners() {
    // Configura os listeners para os modais
}
```

Segurança

Autenticação

O sistema utiliza o sistema de autenticação padrão do Laravel, com algumas personalizações:

- Adição do campo `is_admin` na tabela `users` para controle de acesso administrativo
- Middleware `admin` para proteger rotas administrativas

Validação de Dados

Todas as entradas de usuário são validadas utilizando as regras de validação do Laravel:

```
$request->validate([
    'name' => 'required|string|max:255',
    'email' => 'required|string|email|max:255|unique:users',
    'password' => 'required|string|min:8|confirmed',
]);
```

Proteção CSRF

Todas as requisições POST, PUT e DELETE são protegidas contra ataques CSRF utilizando o middleware `csrf` do Laravel.

Políticas de Acesso

O sistema utiliza políticas de acesso para garantir que os usuários só possam acessar recursos aos quais têm permissão:

```
class CharacterPolicy
{
    public function view(User $user, Character $character)
    {
        return $user->id === $character->user_id;
    }

    public function update(User $user, Character $character)
    {
        return $user->id === $character->user_id;
    }

    public function delete(User $user, Character $character)
    {
        return $user->id === $character->user_id;
    }
}
```

Desempenho

Otimização de Consultas

O sistema utiliza eager loading para evitar o problema N+1 em consultas ao banco de dados:

```
$characters = Character::with('user', 'inventory.item')->get();
```

Cache

O sistema utiliza cache para armazenar dados frequentemente acessados:

```
$phases = Cache::remember('phases', 60*24, function () {
    return Phase::with('challenges')->orderBy('order')->get();
});
```

Paginação

Listas longas são paginadas para melhorar o desempenho:

```
$characters = Character::paginate(10);
```

Testes

Testes Unitários

O sistema inclui testes unitários para as principais funcionalidades:

```
class CharacterTest extends TestCase
{
    use RefreshDatabase;

    public function test_user_can_create_character()
    {
        $user = User::factory()->create();
        $this->actingAs($user);

        $response = $this->post('/characters', [
            'name' => 'Test Character',
        ]);

        $response->assertRedirect('/characters');
        $this->assertDatabaseHas('characters', [
            'name' => 'Test Character',
            'user_id' => $user->id,
        ]);
    }

    // Outros testes
}
```

Testes de Integração

O sistema inclui testes de integração para verificar o funcionamento conjunto dos componentes:

```
 class GameplayTest extends TestCase
{
    use RefreshDatabase;

    public function test_user_can_solve_challenge()
    {
        // Configuração do teste
        $user = User::factory()->create();
        $character = Character::factory()->create(['user_id' =>
$user->id]);
        $phase = Phase::factory()->create();
        $challenge = Challenge::factory()->create(['phase_id' =>
$phase->id]);
        $code = Code::factory()->create([
            'challenge_id' => $challenge->id,
            'code' => 'TEST123',
        ]);

        $this->actingAs($user);

        // Execução do teste
        $response = $this->postJson("/gameplay/challenge/
{$challenge->id}/solve", [
            'code' => 'TEST123',
        ]);

        // Verificação dos resultados
        $response->assertStatus(200);
        $response->assertJson([
            'success' => true,
        ]);

        $this->assertDatabaseHas('player_progress', [
            'character_id' => $character->id,
            'challenge_id' => $challenge->id,
            'status' => 'completed',
        ]);
    }

    // Outros testes
}
```

Implantação

Requisitos do Servidor

- PHP 8.1 ou superior
- Extensões PHP: BCMath, Ctype, Fileinfo, JSON, Mbstring, OpenSSL, PDO, Tokenizer, XML
- Composer
- Node.js e npm (para compilação de assets)

Processo de Implantação

1. Clone o repositório: `git clone https://github.com/username/escape-room-game.git`
2. Instale as dependências PHP: `composer install --optimize-autoloader --no-dev`
3. Instale as dependências JavaScript: `npm install && npm run production`
4. Configure o arquivo `.env` com as informações do ambiente
5. Gere a chave da aplicação: `php artisan key:generate`
6. Execute as migrações: `php artisan migrate`
7. Popule o banco de dados: `php artisan db:seed`
8. Configure o servidor web para apontar para o diretório `public`

Configuração do Servidor Web

Apache

```
<VirtualHost *:80>
    ServerName escaperoomgame.com
    DocumentRoot /var/www/escape-room-game/public

    <Directory /var/www/escape-room-game/public>
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Nginx

```
server {
    listen 80;
    server_name escaperoomgame.com;
    root /var/www/escape-room-game/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location = /favicon.ico { access_log off; log_not_found
off; }
    location = /robots.txt  { access_log off; log_not_found
off; }

    error_page 404 /index.php;

    location ~ \.php$ {
        fastcgi_pass unix:/var/run/php/php8.1-fpm.sock;
        fastcgi_param SCRIPT_FILENAME
$realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.(?!well-known).* {
        deny all;
    }
}
```

Manutenção e Atualizações

Backup

É recomendável realizar backups regulares do banco de dados e dos arquivos da aplicação:

```
# Backup do banco de dados
cp /var/www/escape-room-game/database/database.sqlite /backup/
```

```
database_$(date +%Y%m%d).sqlite

# Backup dos arquivos
tar -czf /backup/escape-room-game_$(date +%Y%m%d).tar.gz /var/
www/escape-room-game
```

Atualizações

Para atualizar a aplicação:

1. Faça backup do banco de dados e dos arquivos
2. Atualize o código-fonte: `git pull origin main`
3. Atualize as dependências: `composer install --optimize-autoloader --no-dev`
4. Execute as migrações pendentes: `php artisan migrate`
5. Limpe o cache: `php artisan cache:clear`
6. Recompile os assets: `npm run production`

Monitoramento

É recomendável utilizar ferramentas de monitoramento para acompanhar o desempenho e a disponibilidade da aplicação:

- Laravel Telescope para monitoramento de requisições, consultas SQL, jobs, etc.
- New Relic ou Datadog para monitoramento de desempenho
- Sentry para monitoramento de erros