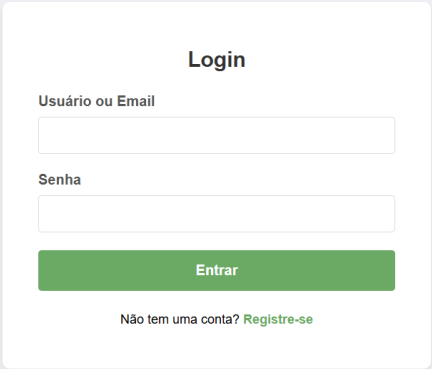
Semana 02 - Criação da Página Log-in

Nessa semana, criei o sistema de log-in para meu sistema:



Ela é bem simples, exigindo apenas e-mail e senha, sem opção de log-in por SSO.

O schema do banco de dados do CloneTermoo é:

```
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(254) NOT NULL UNIQUE,
    senha_hash VARCHAR(255) NOT NULL
);

CREATE TABLE palavras (
    id INT AUTO_INCREMENT PRIMARY KEY,
    palavra VARCHAR(5) UNIQUE,
    dificuldade ENUM('facil', 'medio', 'dificil')
);

CREATE TABLE jogos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    usuario_id INT,
    palavra_id INT,  -- referência à palavra secreta
    tentativas INT DEFAULT 0,
    max_tentativas INT DEFAULT 6,  -- limite de tentativa
    venceu BOOLEAN DEFAULT FALSE,
    data_inicio TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    data_fim TIMESTAMP NULL,
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id) ON DELETE SET NULL, -- se
usuário for deletado, permite manter o jogo
    FOREIGN KEY (palavra_id) REFERENCES palavras(id)
```

Os arquivos desse sistema são organizados da seguinte forma:

```
CloneTermoooPHP/ (Pasta raiz do projeto)
│
├── public/ (Esta é a pasta raiz do servidor web - "DocumentRoot")
│   │
│   ├── index.php          (Página principal do jogo)
│   ├── login.html          (O formulário de login)
│   ├── registrar.html      (O formulário de registro)
│   │
│   ├── processa_login.php    (Endpoint do AJAX)
│   ├── processa_registro.php (Endpoint do AJAX de registro)
│   ├── logout.php          (Script para destruir a sessão)
│   │
│   └── assets/ (Todos os arquivos CSS, JS e imagens)
│       ├── css/
│       │   ├── style_login.css
│       │   └── style_game.css
│       └── js/
│           ├── login.js
│           └── registrar.js
│
└── src/ (Código "privado" do backend e lógica de negócios)
    │
    ├── db_config.php       (A conexão com o BD)
    └── auth_check.php      (Um script para verificar se o usuário está
logado)
```

Códigos do projeto

A seguir, são apresentados os códigos que geram o sistema.

Front-end

No front-end, as tecnologias utilizadas forma HTML, CSS e JS

HTML da tela de login

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login - Clone Termooo</title>
    <link rel="stylesheet" href="assets/css/style_login.css">
</head>
<body>

    <div class="login-container">
```

```
        <form id="loginForm">
            <h2>Login</h2>

            <div id="message" class="message"></div>

            <div class="input-group">
                <label for="loginUser">Usuário ou Email</label>
                <input type="text" id="loginUser" name="loginUser" required
autocomplete="username">
            </div>

            <div class="input-group">
                <label for="loginPass">Senha</label>
                <input type="password" id="loginPass" name="loginPass" required
autocomplete="current-password">
            </div>

            <button type="submit" id="loginButton">Entrar</button>

            <div class="link-registro">
                <p>Não tem uma conta? <a href="registrar.html">Registre-se</a></p>
            </div>
        </form>
    </div>

    <script src="assets/js/login.js"></script>
</body>
</html>
```

JS da tela de login

```
document.addEventListener('DOMContentLoaded', () => {
    const loginForm = document.getElementById('loginForm');
    const messageDiv = document.getElementById('message');
    const loginButton = document.getElementById('loginButton');

    loginForm.addEventListener('submit', async (e) => {
        e.preventDefault(); // Impede o recarregamento da página

        // Desabilita o botão e mostra "Carregando..."
        loginButton.disabled = true;
        loginButton.textContent = 'Carregando...';
        messageDiv.className = 'message';
        messageDiv.textContent = '';

        // Coleta os dados do formulário
        const formData = new FormData(loginForm);
        const data = Object.fromEntries(formData.entries());

        try {
            const response = await fetch('processa_login.php', {
                method: 'POST',
```

```
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(data)
            });

            const result = await response.json();

            if (result.success) {
                // Sucesso
                messageDiv.textContent = 'Login bem-sucedido! Redirecionando...';
                messageDiv.className = 'message success';

                // Redireciona para a página principal do jogo (ex: index.php)
                setTimeout(() => {
                    // O destino deve ser a página principal do jogo
                    window.location.href = 'index.php';
                }, 2000);

            } else {
                // Erro
                messageDiv.textContent = result.message || 'Erro desconhecido.';
                messageDiv.className = 'message error';
                loginButton.disabled = false;
                loginButton.textContent = 'Entrar';
            }
        } catch (error) {
            console.error('Erro na requisição:', error);
            messageDiv.textContent = 'Erro de conexão com o servidor.';
            messageDiv.className = 'message error';
            loginButton.disabled = false;
            loginButton.textContent = 'Entrar';
        }
    });
});
```

Back-End

O back-end desse projeto será todo desenvolvido por PHP

db_config.php

Responsável pela conexão do sistema com o banco de dados.

```
<?php
// db_config.php
define('DB_HOST', 'localhost');
define('DB_USER', 'root');       // <-- Seu usuário do MySQL
define('DB_PASS', '');           // <-- Sua senha do MySQL
define('DB_NAME', 'termooo_db'); // O banco de dados da sua documentação
define('DB_CHARSET', 'utf8mb4');
```

```
$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];

try {
    $dsn = "mysql:host=" . DB_HOST . ";dbname=" . DB_NAME . ";charset=" .
DB_CHARSET;
    $pdo = new PDO($dsn, DB_USER, DB_PASS, $options);
} catch (PDOException $e) {
    // Em um ambiente de produção, você não deve exibir erros detalhados.
    // Você deve logar o erro e mostrar uma mensagem genérica.
    error_log($e->getMessage()); // Loga o erro no servidor

    // Resposta de erro genérica para o frontend (caso seja uma API)
    // header('Content-Type: application/json');
    // echo json_encode(['success' => false, 'message' => 'Erro de conexão com o
banco de dados.']);

    // Ou uma página de erro
    die("Erro crítico de conexão com o banco de dados. Por favor, tente mais
tarde.");
}
?>
```

processa_login.php

Segmenta as informações recebidas como arquivo JSON do documento login.js e verifica se o e-mail digitado pelo usuário e a senha correspondem com algum usuário já cadastrado no banco de dados, na tabela "usuarios".

```
<?php
// processa_login.php

// 1. Inicia a sessão (ESSENCIAL para manter o usuário logado)
// Deve ser a primeira coisa no script
session_start();

// 2. Inclui a conexão com o banco
require '/../src/db_config.php';

// 3. Define o tipo de resposta como JSON
header('Content-Type: application/json');

// 4. Prepara a resposta padrão
$response = ['success' => false, 'message' => 'Requisição inválida.'];

// 5. Verifica se o método é POST
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
```

```
    // Pega os dados JSON enviados pelo JavaScript
    $data = json_decode(file_get_contents('php://input'), true);

    // Validação básica
    if (empty($data['loginUser']) || empty($data['loginPass'])) {
        $response['message'] = 'Usuário/Email e senha são obrigatórios.';
        echo json_encode($response);
        exit;
    }

    $loginUser = trim($data['loginUser']); // Pode ser username ou email
    $password = $data['loginPass'];

    try {
        // 6. Prepara a query (Seguro contra SQL Injection)
        // O login funciona tanto com 'username' quanto com 'email'
        $sql = "SELECT id, username, senha_hash FROM usuarios WHERE username =
:login OR email = :login LIMIT 1";

        $stmt = $pdo->prepare($sql);
        $stmt->execute(['login' => $loginUser]);

        $user = $stmt->fetch();

        // 7. Verifica se o usuário existe E se a senha está correta
        if ($user && password_verify($password, $user['senha_hash'])) {

            // SUCESSO!

            // 8. Regenera o ID da sessão para evitar "Session Fixation"
            session_regenerate_id(true);

            // 9. Armazena os dados do usuário na sessão
            $_SESSION['user_id'] = $user['id'];
            $_SESSION['username'] = $user['username'];
            $_SESSION['logged_in'] = true;

            $response['success'] = true;
            $response['message'] = 'Login bem-sucedido!';

        } else {
            // Falha (usuário não encontrado ou senha errada)
            // Usamos uma mensagem genérica por segurança
            $response['message'] = 'Usuário ou senha inválidos.';
        }

    } catch (PDOException $e) {
        // Loga o erro e envia uma resposta genérica
        error_log($e->getMessage());
        $response['message'] = 'Erro no servidor. Tente novamente mais tarde.';
    }
}

// 10. Retorna a resposta (seja sucesso ou falha) como JSON
```

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

Maiores dificuldades do projeto

Minha maior dificuldade foi descobrir como trabalhar com o JSON que é enviado de "login.js" para "processa_login.php".

Sei que JSON é uma prática comum no mercado, já se tornou uma obrigação saber trabalhar com essa nomenclatura, mas foi a primeira vez que tive fazer algo mais "complexo" com essa estrutura de arquivo