APLICAÇÃO DE AGENDAMENTO DE TAREFAS COM CAKEPHP

Guilherme Morigi 1

Ivan da Silva 2

Nathan Henrique Riffel 3

Thiago Emanuel Correia Heckler 4

RESUMO

Este artigo detalha o processo de desenvolvimento de uma aplicação web para agendamento e gerenciamento de tarefas, concebida como projeto para a disciplina de Programação 3. A plataforma foi construída utilizando o framework CakePHP 5, com PHP 8.1 e o sistema de gerenciamento de banco de dados PostgreSQL. O sistema implementa funcionalidades essenciais, incluindo um robusto sistema de autenticação de usuários, permitindo que cada indivíduo se cadastre, faça login e gerencie sua própria lista de tarefas de forma privada e segura. O projeto segue o padrão arquitetural MVC (Model-View-Controller), fazendo uso de ferramentas como Composer para gerenciamento de dependências e Migrations para o versionamento do banco de dados. O objetivo é apresentar uma solução funcional e didática para o problema de organização de atividades diárias, demonstrando a aplicação prática de conceitos de engenharia de software em um projeto real.

Palavras-chave*: CakePHP, MVC, Agendador de Tarefas, Desenvolvimento Web, PostgreSQL*

ABSTRACT

This article details the development process of a web application for scheduling and managing tasks, designed as a project for the Programming 3 course. The platform was built using the CakePHP 5 framework, with PHP 8.1 and the PostgreSQL database management system. The system implements essential functionalities, including a robust user authentication system, allowing individuals to register, log in, and privately and securely manage their own task lists. The project follows the MVC (Model-View-Controller) architectural pattern, utilizing tools such as Composer for dependency management and Migrations for database versioning. The goal is to present a functional and didactic solution for daily activity organization, demonstrating the practical application of software engineering concepts in a real-world project.

Keywords: CakePHP, MVC, Task Scheduler, Web Development, PostgreSQL.

1 INTRODUÇÃO

A gestão de tarefas diárias é um desafio comum na vida moderna, onde a sobrecarga de informações e responsabilidades pode levar à desorganização e à perda de produtividade. Ferramentas digitais de gerenciamento de tarefas surgem como soluções eficazes para mitigar esse problema, oferecendo um meio centralizado e acessível para organizar, priorizar e acompanhar atividades. No entanto, muitas soluções disponíveis podem ser excessivamente complexas ou onerosas para o usuário comum.

Diante desse cenário, este projeto aborda o desenvolvimento de uma aplicação web de agendamento de tarefas, denominada "Agendador de Tarefas". O sistema foi concebido com o objetivo de oferecer uma plataforma simples, intuitiva e segura para o gerenciamento de tarefas pessoais. O problema central que se busca resolver é a necessidade de uma ferramenta minimalista que permita a usuários cadastrados criar, visualizar, editar e remover suas próprias tarefas, garantindo a privacidade e a integridade de seus dados.

Desenvolvido como uma atividade acadêmica para a disciplina de Programação 3, o projeto não apenas visa criar um produto funcional, mas também aplicar e aprofundar os conhecimentos em tecnologias e conceitos fundamentais do desenvolvimento de software, como a arquitetura MVC, programação orientada a objetos, sistemas de autenticação e versionamento de código e banco de dados.

2 DESENVOLVIMENTO

O processo de construção do "Agendador de Tarefas" foi estruturado em etapas que abrangeram desde a fundamentação teórica até a implementação e os testes, utilizando uma combinação de tecnologias modernas e metodologias de desenvolvimento consolidadas.

Fundamentação Técnica

O projeto foi solidamente baseado em princípios e padrões de arquitetura de software para garantir manutenibilidade, escalabilidade e segurança.

· **Programação Orientada a Objetos (POO):** O PHP, sendo uma linguagem com forte suporte à POO, foi a base do desenvolvimento. O framework CakePHP potencializa esse paradigma, onde cada componente da aplicação — como controladores, modelos (Tables e Entities) e views — é uma classe com responsabilidades bem definidas, promovendo a reutilização de código e a encapsulaçã**Arquitetura MVC (Model-View-Controller):** A aplicação foi estruturada seguindo o padrão MVC, uma diretriz central do CakePHP, que separa as responsabilidades da aplicação em três camadas:

o **Model:** Responsável pela lógica de negócio e interação com o banco de dados. Implementado através das classes Table (ex: TasksTable.php, UsersTable.php), que gerenciam as consultas e a validação dos dados, e das classes Entity (ex: Task.php, User.php), que representam os registros individuais do banco de dados.

o **View:** Camada de apresentação responsável por exibir os dados ao usuário. Localizada no diretório templates/, consiste em arquivos que renderizam a interface, como index.php para listagem e add.php para formulários de adição.

o **Controller:** Atua como intermediário, recebendo as requisições do usuário, acionando as operações no Model e selecionando a View apropriada para a resposta. Os arquivos TasksController.php e UsersController.php contêm as ações (actions) que implementam as funcionalidades de CRUD de tarefas e autenticação de usuários, respectivamente.

· **Autenticação e Segurança:** Para a gestão de acesso, foi utilizado o plugin oficial cakephp/authentication. Ele foi configurado em src/Application.php para gerenciar o login, logout e as sessões de usuário. A segurança das senhas é garantida pela função _setPassword na entidade User.php, que utiliza o DefaultPasswordHasher para criptografar as senhas antes de salvá-las no banco. Além disso, o framework oferece proteção nativa contra ataques como Cross-Site Request

Forgery (CSRF), que foi mantida ativa.

· **Versionamento:** O controle de versão do código-fonte foi realizado com o Git, e o projeto foi hospedado no GitHub. O repositório contém um arquivo .gitignore configurado para excluir arquivos sensíveis e desnecessários, como config/app_local.php e o diretório vendor. Para o versionamento da estrutura do banco de dados, utilizou-se o recurso de Migrations do CakePHP.

Metodologia de Desenvolvimento

A equipe, composta por três membros, adotou uma abordagem de desenvolvimento colaborativa e iterativa. As tarefas foram divididas com base nas funcionalidades principais da aplicação (autenticação de usuário e CRUD de tarefas). O GitHub foi utilizado como plataforma central para colaboração, permitindo que os desenvolvedores trabalhassem em paralelo e integrassem suas contribuições de forma organizada. A comunicação constante e a revisão de código foram práticas adotadas para garantir a qualidade e a coesão do projeto final.

Descrição das Tecnologias e Frameworks Utilizados

· **Backend:** A aplicação foi desenvolvida com **PHP 8.1+** e o **CakePHP 5.x**, um framework de desenvolvimento rápido que oferece uma base robusta e ferramentas integradas para acelerar o processo construtivo.

· **Banco de Dados:** Foi escolhido o **PostgreSQL 16**, um sistema de gerenciamento de banco de dados relacional objeto, conhecido por sua robustez e conformidade com os padrões SQL. A configuração da conexão está definida nos arquivos config/app.php e config/app_local.php.

· **Frontend:** A interface do usuário foi construída com **HTML5** e estilizada com o framework **Milligram**, um framework CSS minimalista, complementado por estilos customizados definidos em webroot/css/custom.css para criar um tema escuro e uma identidade visual para a aplicação.

· **Dependências:** O **Composer** foi utilizado para gerenciar as dependências do projeto, incluindo o próprio CakePHP e seus plugins.

· **Servidor de Desenvolvimento:** Para os testes locais, foi utilizado o servidor web embutido no CakePHP, iniciado pelo comando bin/cake server.

Etapas do Processo Construtivo

· **Modelagem de Dados:** A estrutura do banco de dados foi definida e implementada utilizando o recurso de **Migrations** do CakePHP. Foram criados três arquivos de migração principais:

1. 20250617220000_CreateUsers.php: Cria a tabela users com os campos id, email, password e created/modified.

2. 20250610005848_CreateTasksTable.php: Cria a tabela tasks com os campos id, title, description, completed, data_agendada e created/modified.

3. 20250617220100_AddUserIdToTasks.php: Adiciona a coluna user_id à tabela tasks e cria uma chave estrangeira que a relaciona com a tabela users, estabelecendo o relacionamento de que um usuário pode ter várias tarefas.

· **Implementação:**

1. **Autenticação:** Foi criado o UsersController com as ações add, login e logout. A view login.php fornece o formulário de acesso e a add.php o de registro. O AppController foi configurado para carregar o componente de autenticação em toda a aplicação.

2. **CRUD de Tarefas:** O TasksController foi implementado com as ações index, view, add, edit e delete. As consultas foram ajustadas para garantir que um usuário só possa visualizar e manipular suas próprias tarefas, utilizando o id do usuário logado como filtro nas buscas.

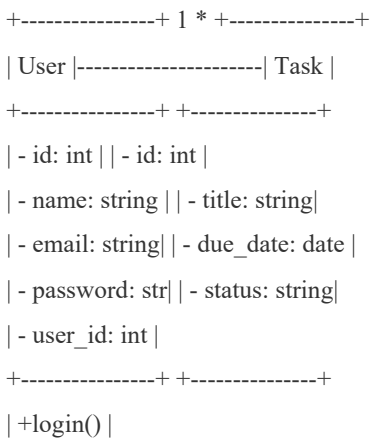
3. **Views e Layout:** Foi desenvolvido um layout padrão (templates/layout/default.php) que inclui um menu de navegação e a área de conteúdo. O layout também verifica se um usuário está logado para exibir o link "Sair". Foram criados templates para cada ação dos controladores.

· **Testes:** A estrutura do CakePHP já fornece um diretório tests/ com suporte para testes unitários e de integração via PHPUnit. Foram gerados arquivos de teste para os controladores e modelos, como TasksControllerTest.php e TasksTableTest.php, embora a implementação detalhada dos casos de teste tenha sido marcada como incompleta, a estrutura está pronta para expansão futura.

· **Deploy (Implantação):** O README.md do projeto fornece um guia detalhado para configurar o ambiente de desenvolvimento e executar a aplicação localmente. O processo envolve clonar o repositório, instalar as dependências com o Composer, configurar o banco de dados e executar as migrações.

Diagrama de Classes

Diagrama de classes-Entidades principais



-User e Task são as entidades centrais (em Model/Entity/)
-Relação de 1:N entre User e Task (um usuário pode ter várias tasks)

Modelo Entidade-Relacionamento

USERS

---

id (PK)
name
email
password
created
modified

TASKS

---

id (PK)
title
description
due_date

status

user_id (FK → USERS.id)

created

modified

Relacionamentos:

USERS (1) -> TASKS (N) via user_id

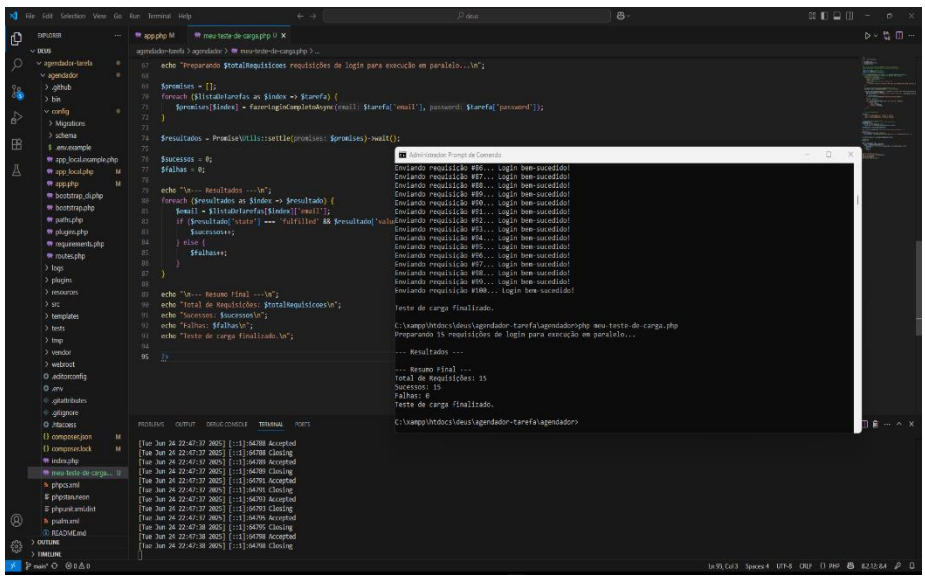
Tabelas seguem convenção CakePHP com campos created e modified

Teste de carga

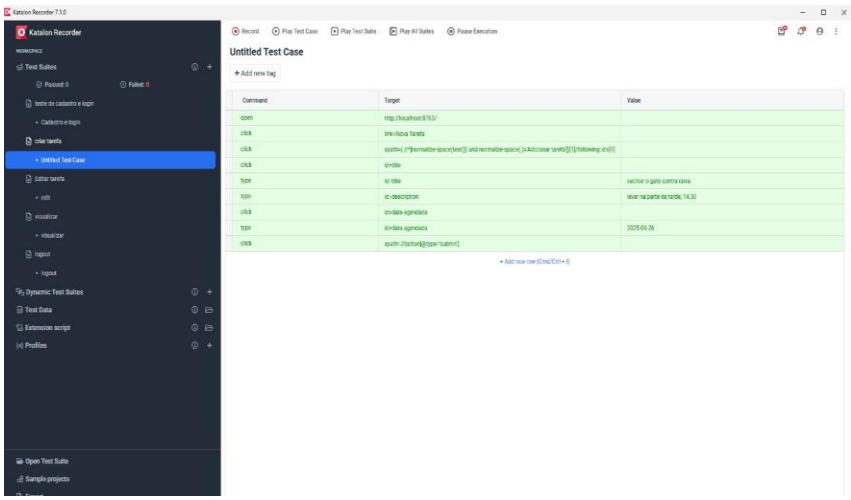
```
<?php

require 'vendor/autoload.php';

use GuzzleHttp\Client;
use GuzzleHttp\Cookie\CookieJar;
use GuzzleHttp\Promise;

/**
 * Esta função agora faz o processo COMPLETO de login de forma assíncrona. * Ela cria seu próprio cliente para garantir que a sessão seja 100% isolada. *
 * @param string $email O email do usuário.
 * @param string $password A senha do usuário.
 * @return \GuzzleHttp\Promise\PromiseInterface
 */
function fazerLoginCompletoAsync(string $`email, string `$password): Promise\PromiseInterface {
// Cada chamada a esta função cria um cliente e um cookie jar novos e isolados.
$client = new Client([
'base_uri' => 'http://localhost:8765',
'cookies' => new CookieJar(),
'timeout' => 30.0,
// Adicionamos um header para simular um navegador real
```

```
'headers' => [
'User-Agent' => 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36' ]
]);

// O resto da lógica é a mesma: pega o token e depois faz o login $`promiseGet = `$client-
>requestAsync('GET', '/users/login');

return $`promiseGet->then(function (`$response) use ($`client, `$email, $password) {
$`html = (string) `$response->getBody();
preg_match('/<input type="hidden" name="_csrfToken" value="(.*?)"/', $`html, `$matches);

if (!isset($matches[1])) {
throw new \Exception("Token CSRF não encontrado para $email."); }
$`csrfToken = `$matches[1];

return $client->requestAsync('POST', '/users/login', [
'form_params' => [
'email' => $email,
'password' => $password,
'_csrfToken' => $csrfToken
]
]);
});
}

// --- BLOCO DE EXECUÇÃO PRINCIPAL ---
$usuarios = [
['email' => 'vamostestar@gmail.com', 'password' => '123456'], ['email' =>
'thigas1234@gmail.com', 'password' => '123456'], ['email' => 'ivan.js23silva@gmail.com',
'password' => '123456'], ];

// COMECE COM UM NÚMERO MENOR PARA TESTAR!
$fatorDeRepeticao = 5; // Total de 3 * 5 = 15 requisições simultâneas $listaDeTarefas =
[];
for ($`i = 0; `$i < $`fatorDeRepeticao; `$i++) {
$`listaDeTarefas = array_merge(`$listaDeTarefas, $usuarios); }
$`totalRequisicoes = count(`$listaDeTarefas);

echo "Preparando $totalRequisicoes requisições de login para execução em paralelo...\n";

$promises = [];
foreach ($`listaDeTarefas as `$index => $tarefa) {
$`promises[`$index] = fazerLoginCompletoAsync($`tarefa['email'],
```

```
`$tarefa['password']);
            }

            $`resultados = Promise\Utils::settle(`$promises)->wait();

            $sucessos = 0;
            $falhas = 0;

            echo "\n--- Resultados ---\n";
            foreach ($`resultados as `$index => $resultado) {
            $`email = `$listaDeTarefas[$index]['email'];
            if ($`resultado['state'] === 'fulfilled' && `$resultado['value']- >getStatusCode() == 200) {
            $sucessos++;
            } else {
            $falhas++;
            }
            }

            echo "\n--- Resumo Final ---\n";
            echo "Total de Requisições: $totalRequisicoes\n";
            echo "Sucessos: $sucessos\n";
            echo "Falhas: $falhas\n";
            echo "Teste de carga finalizado.\n";

            ?>
```



Testes Katalon (Teste funcionalidade)



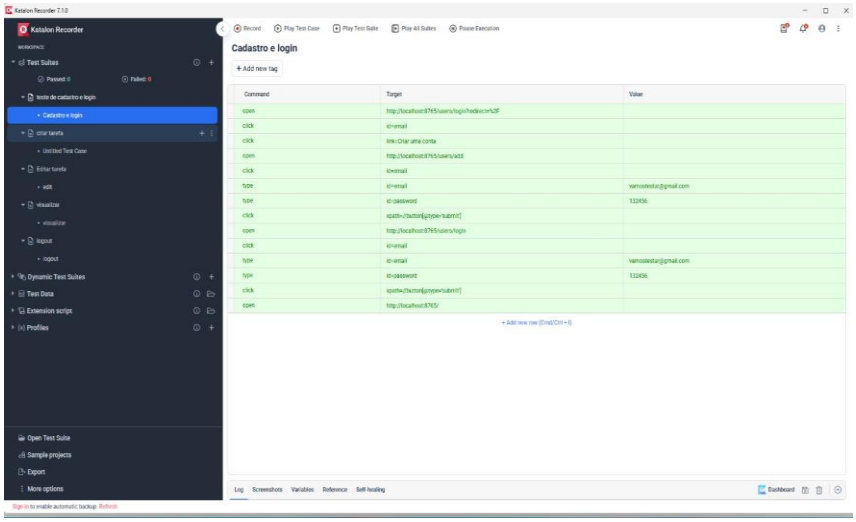


Diagrama de Caso de Uso

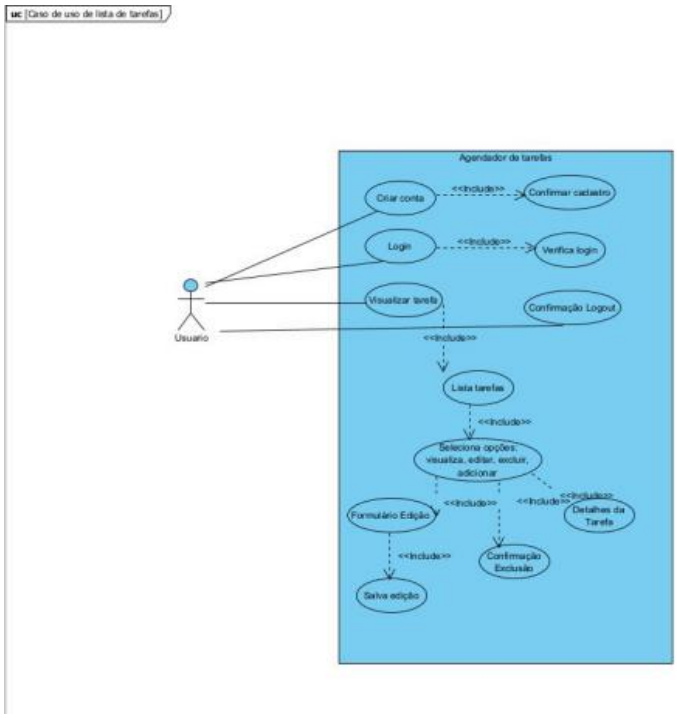
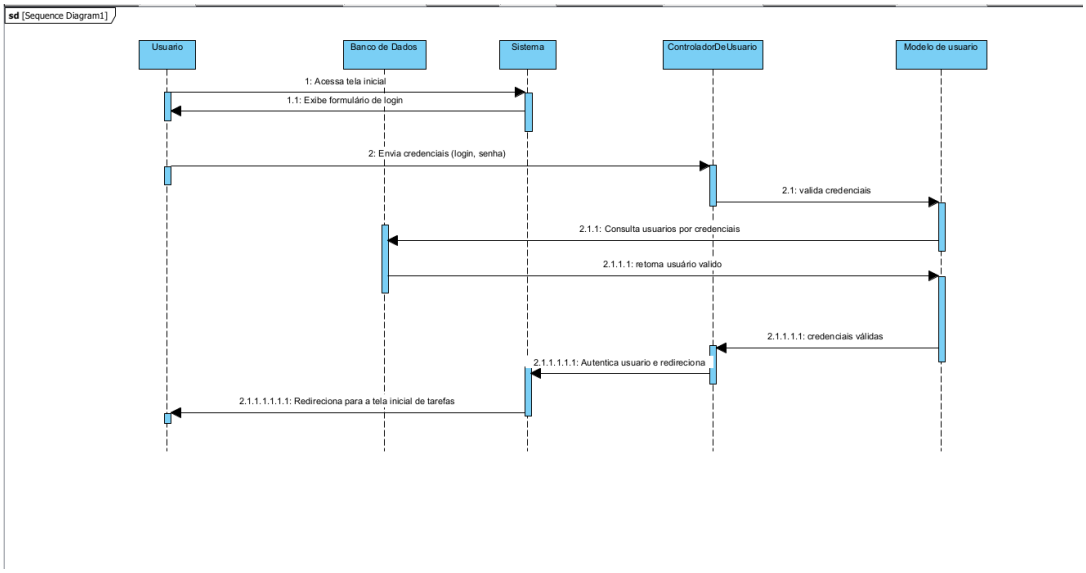


Diagrama de sequência



Uso de padrões de projeto

O relatório sobre o \"Agendador de Tarefas\" identifica e explica dois padrões de projeto principais utilizados no sistema, que são inerentes ao framework CakePHP:

1. Model-View-Controller (MVC): Este padrão separa a aplicação em três partes: o Modelo (lógica de negócios e dados, como TasksTable.php), a Visão (interface do usuário, como templates/Tasks/index.php) e o Controlador (intermediário que gerencia as interações, como TasksController.php). Essa separação facilita a organização, manutenção e escalabilidade do código.

2. Active Record: Embora o CakePHP use uma variação (Table Gateway/Data Mapper),

ele oferece uma experiência similar ao Active Record. Este padrão simplifica a interação com o banco de dados, permitindo que as operações de CRUD (Criar, Ler, Atualizar, Deletar) sejam realizadas diretamente através de objetos (TasksTable.php e Task.php), tornando o código mais intuitivo e produtivo.

Em resumo, o uso desses padrões no \"Agendador de Tarefas\" resulta em um código bem organizado, fácil de manter e escalável, essencial para o desenvolvimento de
aplicações web eficientes.

1. Testes para TasksController

O TasksController é responsável por gerenciar as ações relacionadas às tarefas, como listar, visualizar, adicionar, editar e excluir. Os testes a seguir verificam cada uma dessas ações.

Arquivo:
agendador/tests/TestCase/Controller/TasksControllerTest.php **2. Testes para UsersController**

O UsersController lida com a autenticação dos usuários. Os testes abaixo verificam as funcionalidades de login, logout e adição de novos usuários.

Arquivo:
agendador/tests/TestCase/Controller/UsersControllerTest.php **3. Testes para TasksTable**

Os testes para a classe TasksTable garantem que as regras de validação e as associações do modelo estão funcionando corretamente.

Arquivo: agendador/tests/TestCase/Model/Table/TasksTableTest.php

Imagens do Relatório Adicional

```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\Controller;

use App\Controller\TasksController;
use Cake\TestSuite\IntegrationTestTrait;
use Cake\TestSuite\TestCase;
use Cake\ORM\TableRegistry;

/**
 * App\Controller\TasksController Test Case
 *
 * @uses \App\Controller\TasksController
 */
0 references | 0 implementations | ···
class TasksControllerTest extends TestCase
{
    use IntegrationTestTrait;

    /**
     * Fixtures
     *
     * @var array<string>
     */
    0 references
    protected $fixtures = [
        'app.Tasks',        You, há 11 minutos • Uncommitted changes
        'app.Users',
    ];

    /**
     * setUp method
     *
     * @return void
     */
    Tabnine | Edit | Test | Fix | Explain | Document | 3 references | 0 overrides
    public function setUp(): void
    {
        parent::setUp();
        $this->Users = $this->getTableLocator()->get(alias: 'Users');
        $this->Tasks = $this->getTableLocator()->get(alias: 'Tasks');
        $this->user = $this->Users->get(primaryKey: 1);
        $this->session(data: ['Auth' => $this->user]);
    }

    /**
     * Test index method
     *
     * @return void
     * @uses \App\Controller\TasksController::index()
     */
    Tabnine | Edit | Test | Explain | Document | 0 references | 0 overrides
    public function testIndex(): void
    {
        $this->get(url: '/tasks');
        $this->assertResponseOk();
        $this->assertResponseContains(content: 'Tarefas');
    }

    /**
     * Test view method
     *
     * @return void
     * @uses \App\Controller\TasksController::view()
     */
    Tabnine | Edit | Test | Explain | Document | 0 references | 0 overrides
    public function testView(): void
    {
        $this->get(url: '/tasks/view/1');
        $this->assertResponseOk();
        $this->assertResponseContains(content: 'Lorem ipsum dolor sit amet');
    }

    /**
     * Test add method
     *
     * @return void
     * @uses \App\Controller\TasksController::add()
     */
    Tabnine | Edit | Test | Explain | Document | 0 references | 0 overrides
    public function testAdd(): void
    {
        $this->enableCsrfToken();
        $this->post(url: '/tasks/add', data: [
            'title' => 'Nova Tarefa',
            'description' => 'Descrição da nova tarefa',
            'completed' => 0,
            'user_id' => 1
        ]);
```

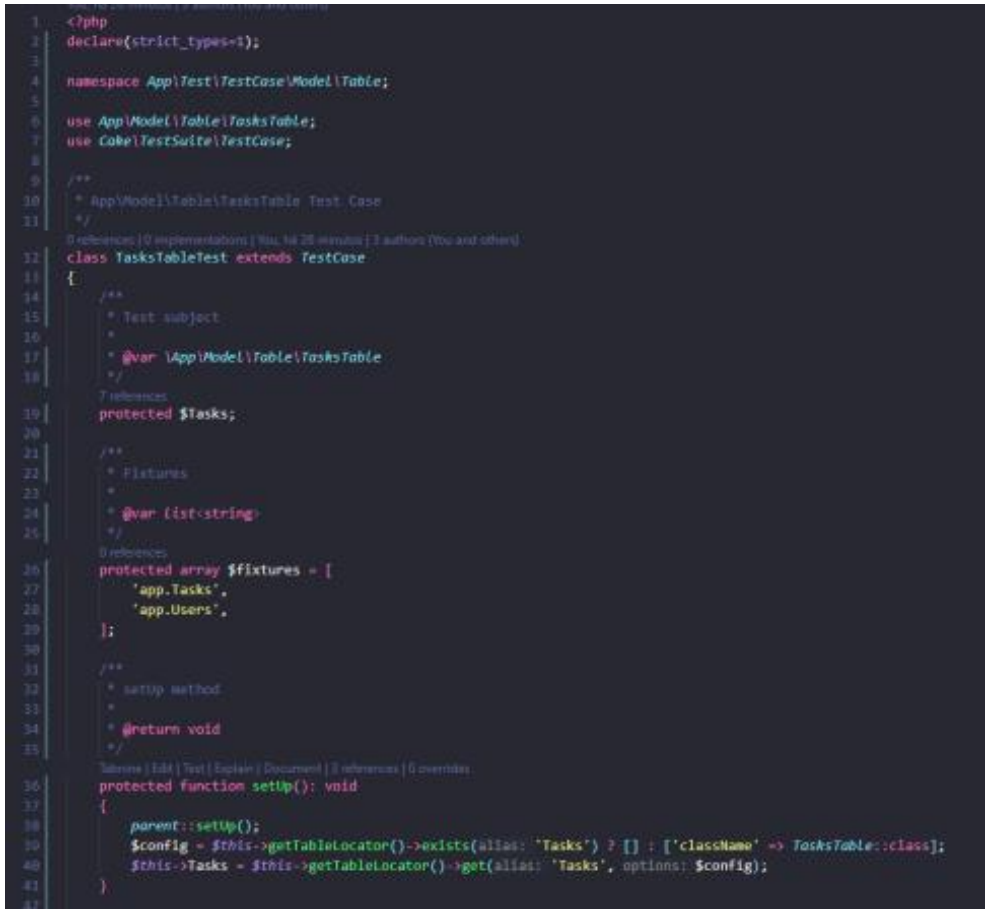
```php
        ]);
        $this->assertRedirect(url: '/tasks');
        $this->assertFlashMessage(expected: 'A tarefa foi salva.', key: 'flash.success');
    }

    /**
     * Test edit method
     *
     * @return void
     * @uses \App\Controller\TasksController::edit()
     */
    public function testEdit(): void
    {
        $this->enableCsrfToken();
        $this->put(url: '/tasks/edit/1', data: [
            'title' => 'Tarefa Editada',
            'description' => 'Descrição da tarefa editada',
            'completed' => 1,
        ]);
        $this->assertRedirect(url: '/tasks');
        $this->assertFlashMessage(expected: 'A tarefa foi salva.', key: 'flash.success');

        $task = $this->Tasks->get(primaryKey: 1);
        $this->assertEquals(expected: 'Tarefa Editada', actual: $task->title);
    }

    /**
     * Test delete method
     *
     * @return void
     * @uses \App\Controller\TasksController::delete()
     */
    public function testDelete(): void
    {
        $this->enableCsrfToken();
        $this->delete(url: '/tasks/delete/1');
        $this->assertRedirect(url: '/tasks');
        $this->assertFlashMessage(expected: 'A tarefa foi deletada.', key: 'flash.success');

        $this->assertCount(expectedCount: 0, haystack: $this->Tasks->find()->where(conditions: ['id' => 1])->all());
    }
}
```

```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\Controller;

use App\Controller\UsersController;
use Cake\TestSuite\IntegrationTestTrait;
use Cake\TestSuite\TestCase;
use Cake\ORM\TableRegistry;

/**
 * App\Controller\UsersController Test Case
 *
 * @uses \App\Controller\UsersController
 */
0 references | 0 implementations | ···
class UsersControllerTest extends TestCase
{
    use IntegrationTestTrait;

    /**
     * Fixtures
     *
     * @var array<string>
     */
    0 references
    protected $fixtures = [
        'app.Users',
        'app.Tasks',
    ];

    /**
     * Test login method
     *
     * @return void
     */
    Tabnine | Edit | Test | Explain | Document | 0 references | 0 overrides
    public function testLogin(): void
    {
        $this->enableCsrfToken();
        $this->post(url: '/users/login', data: [
            'email' => 'test@example.com',
            'password' => 'password'
        ]);
        $this->assertRedirectContains(url: '/tasks');
    }
```

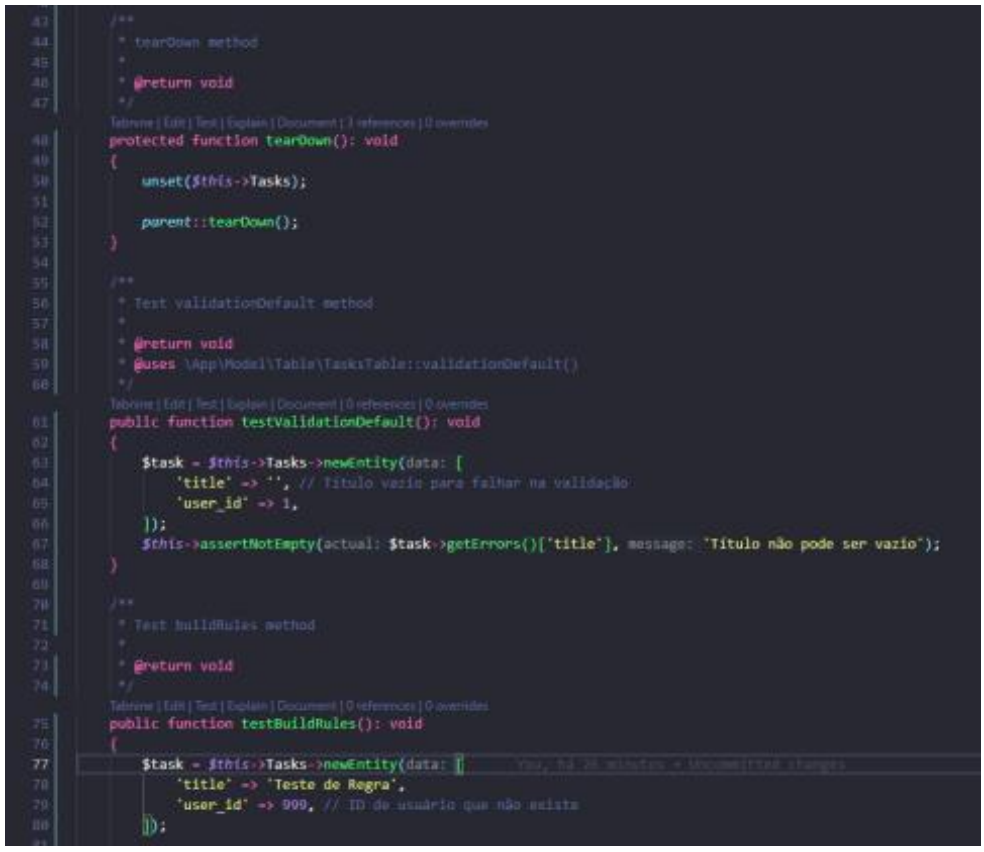
```php
    /**
     * Test login with invalid credentials
     *
     * @return void
     */
    Tabnine | Edit | Test | Explain | Document | 0 references | 0 overrides
    public function testLoginInvalid(): void
    {
        $this->enableCsrfToken();
        $this->post(url: '/users/login', data: [
            'email' => 'test@example.com',
            'password' => 'wrongpassword'
        ]);
        $this->assertResponseOk();
        $this->assertResponseContains(content: 'E-mail ou senha inválidos');
    }

    /**
     * Test logout method
     *
     * @return void
     */
    Tabnine | Edit | Test | Explain | Document | 0 references | 0 overrides
    public function testLogout(): void
    {
        // First, log in a user
        $this->session(data: ['Auth.User.id' => 1]);

        $this->get(url: '/users/logout');
        $this->assertRedirect(url: ['controller' => 'Users', 'action' => 'login']);
        $this->assertSession(expected: null, path: 'Auth');
    }

    /**
     * Test add method
     *
     * @return void
     */
    Tabnine | Edit | Test | Explain | Document | 0 references | 0 overrides
    public function testAdd(): void
    {
        $this->enableCsrfToken();
        $this->post(url: '/users/add', data: [
            'email' => 'newuser@example.com',
            'password' => 'newpassword',
        ]);
```

```php
    Tabnine | Edit | Test | Explain | Document | 0 references | 0 overrides
    public function testAdd(): void
    {
        $this->enableCsrfToken();
        $this->post(url: '/users/add', data: [
            'email' => 'newuser@example.com',
            'password' => 'newpassword',
        ]);
        $this->assertRedirect(url: ['action' => 'login']);
        $this->assertFlashMessage(expected: 'Usuário criado com sucesso! Você já pode fazer o login.');
    }
}
```

You, há 16 minutos • Uncommitted changes



```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\Model\Table;

use App\Model\Table\TasksTable;
use Cake\TestSuite\TestCase;

/**
 * App\Model\Table\TasksTable Test Case
 */
class TasksTableTest extends TestCase
{
    /**
     * Test subject
     *
     * @var \App\Model\Table\TasksTable
     */
    protected $Tasks;

    /**
     * Fixtures
     *
     * @var list<string>
     */
    protected array $fixtures = [
        'app.Tasks',
        'app.Users',
    ];

    /**
     * setUp method
     *
     * @return void
     */
    protected function setUp(): void
    {
        parent::setUp();
        $config = $this->getTableLocator()->exists('Tasks') ? [] : ['className' => TasksTable::class];
        $this->Tasks = $this->getTableLocator()->get('Tasks', $config);
    }
```



```php
    /**
     * tearDown method
     *
     * @return void
     */
    protected function tearDown(): void
    {
        unset($this->Tasks);

        parent::tearDown();
    }

    /**
     * Test validationDefault method
     *
     * @return void
     * @uses \App\Model\Table\TasksTable::validationDefault()
     */
    public function testValidationDefault(): void
    {
        $task = $this->Tasks->newEntity(data: [
            'title' => '', // Título vazio para falhar na validação
            'user_id' => 1,
        ]);
        $this->assertNotEmpty(actual: $task->getErrors()['title'], message: 'Título não pode ser vazio');
    }

    /**
     * Test buildRules method
     *
     * @return void
     */
    public function testBuildRules(): void
    {
        $task = $this->Tasks->newEntity(data: [
            'title' => 'Teste de Regra',
            'user_id' => 999, // ID de usuário que não existe
        ]);
```

```php
    public function testBuildRules(): void
    {
        $task = $this->Tasks->newEntity(data: [
            'title' => 'Teste de Regra',
            'user_id' => 999, // ID de usuário que não existe
        ]);

        $this->assertFalse(condition: $this->Tasks->save(entity: $task), message: 'A tarefa não deve ser salva com um user_id inexistente.');
        $this->assertArrayHasKey(key: 'user_id', array: $task->getErrors());
        $this->assertEquals(expected: 'This value is not valid', actual: $task->getErrors()['user_id']['_existsIn']);
    }
}
```

CONCLUSÃO

O desenvolvimento do "Agendador de Tarefas" cumpriu com sucesso seus objetivos primários: criar uma aplicação web funcional para o gerenciamento de tarefas e servir como um projeto prático para a aplicação de conceitos avançados de programação. A utilização do framework CakePHP 5 se mostrou uma escolha acertada, pois sua arquitetura MVC e suas convenções bem definidas guiaram a equipe na construção de um código organizado, modular e de fácil manutenção.

Os principais aprendizados durante o projeto incluem a implementação de um sistema de autenticação seguro, a importância do versionamento do banco de dados através de migrations para garantir a consistência entre os ambientes de desenvolvimento, e a prática da programação colaborativa utilizando Git e GitHub. A separação clara de responsabilidades imposta pelo padrão MVC facilitou a divisão de trabalho e a compreensão do sistema como um todo.

Como considerações para trabalhos futuros, a aplicação poderia ser expandida para incluir funcionalidades como notificações por e-mail, categorização de tarefas, definição de prioridades e, talvez, a criação de uma API RESTful para permitir a integração com outras aplicações, como clientes mobile. Adicionalmente, a implementação completa da suíte de testes automatizados seria um passo importante para garantir a robustez e a qualidade do software a longo prazo.

Em suma, o projeto representa uma demonstração bem-sucedida da aplicação de tecnologias e metodologias modernas de engenharia de software na resolução de um problema prático e cotidiano.

REFERÊNCIAS

CakePHP. (2024). **CakePHP 5.x Cookbook**. Disponível em: https://book.cakephp.org/5/en/index.html. Acesso em: 30 jun. 2025.

FREEMAN, E.; FREEMAN, E.; SIERRA, K.; BATES, B. **Use a Cabeça! Padrões de Projetos**. 2. ed. Alta Books, 2007.

PHP Group. (2025). **PHP Manual**. Disponível em: https://www.php.net/manual/en/. Acesso em: 30 jun. 2025.

PostgreSQL Global Development Group. (2025). **PostgreSQL 16 Documentation**. Disponível em: https://www.postgresql.org/docs/16/. Acesso em: 30 jun. 2025.

The Composer Maintainers. (2025). **Composer Documentation**. Disponível em: https://getcomposer.org/doc/. Acesso em: 30 jun. 2025.

Sobre o(s) autor(es)

Guilherme Morigi, Ivan Silva, Thiago Emanuel, Nathan Riffel *Titulação:* Acadêmicos do curso de Ciência da Computação *Vínculo:* Universidade do Oeste de Santa Catarina (UNOESC)