Desenvolvimento de um Sistema de Controle de Tarefas utilizando o Framework PHP Symfony

AUTORES

João Gabriel Zangalli

Curso de Ciência da Computação, Campus São Miguel Do Oeste

E-mail: joaozangalli025@gmail.com

Emanuel Previatti

Curso de Ciências da Computação, Campus São Miguel Do Oeste

E-mail: emanuelpreviatti@gmail.com

Vinícius Scholtze Pires da Cunha

Curso de Ciências da Computação, Campus São Miguel Do Oeste

E-mail: viniciuscholtze@gmail.com

1. Introdução

Este relatório tem como objetivo apresentar o processo de modelagem, desenvolvimento e validação do sistema Controle de Tarefas (To-Do List), criado como parte das disciplinas Engenharia de Software II e Programação III, do curso de Ciência da Computação da UNOESC – Universidade do Oeste de Santa Catarina, no 5º semestre. A atividade é orientada pelo professor Roberson Junior Fernandes Alves e envolve a aplicação de conceitos fundamentais de engenharia de software, incluindo modelagem UML, uso de padrões de projeto e validação por testes.

O sistema desenvolvido é uma aplicação web para gerenciamento de tarefas pessoais. Seu objetivo é proporcionar uma interface simples e eficiente para que usuários possam cadastrar, editar, listar, priorizar e acompanhar suas tarefas diárias.

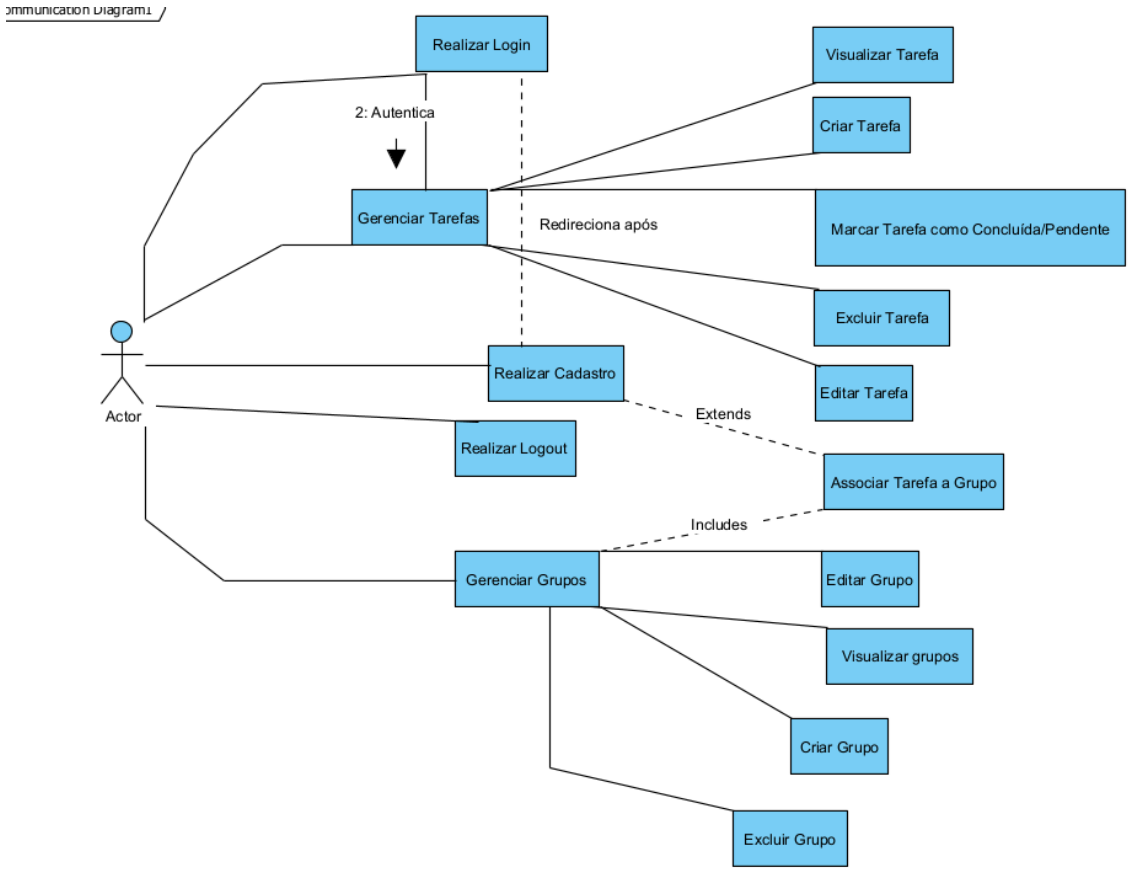
O projeto está sendo desenvolvido em PHP utilizando o framework Symfony, com banco de dados MySQL, utilizando uma estrutura relacional tradicional. O MySQL foi escolhido por sua robustez, ampla documentação, compatibilidade com o Symfony e facilidade de manipulação de dados relacionais. Ao longo do relatório, serão apresentados os principais artefatos de modelagem (diagramas de casos de uso, sequência, classes e modelo relacional), os padrões de projeto utilizados.

2. Descrição do Sistema

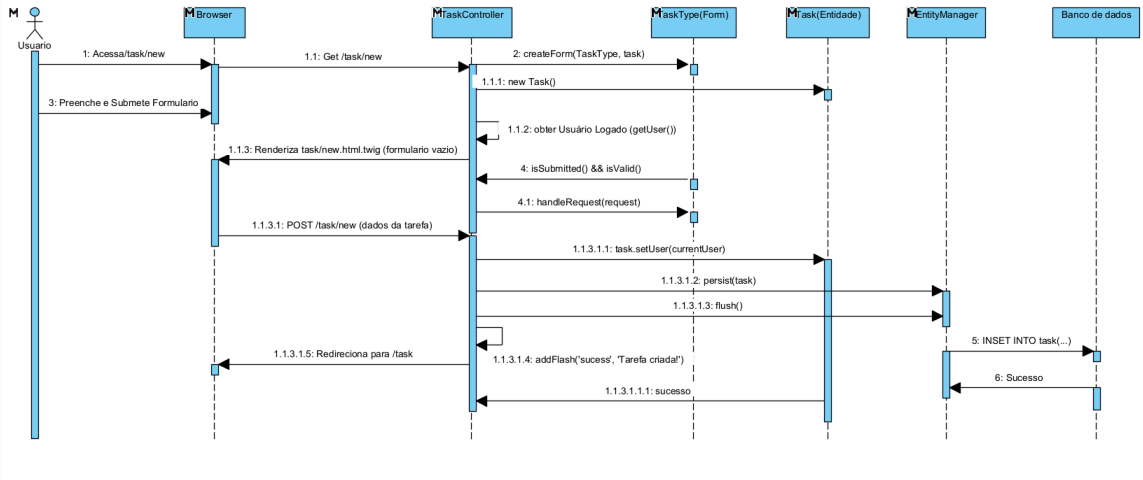
O sistema Controle de Tarefas (To-Do List) é uma aplicação web destinada à organização de tarefas pessoais. Os usuários podem se cadastrar, fazer login e gerenciar suas atividades do dia a dia com recursos como definição de prioridades, prazos, marcação de tarefas feitas ou faltantes e visualização das tarefas do dia.

3. Modelagem de Software

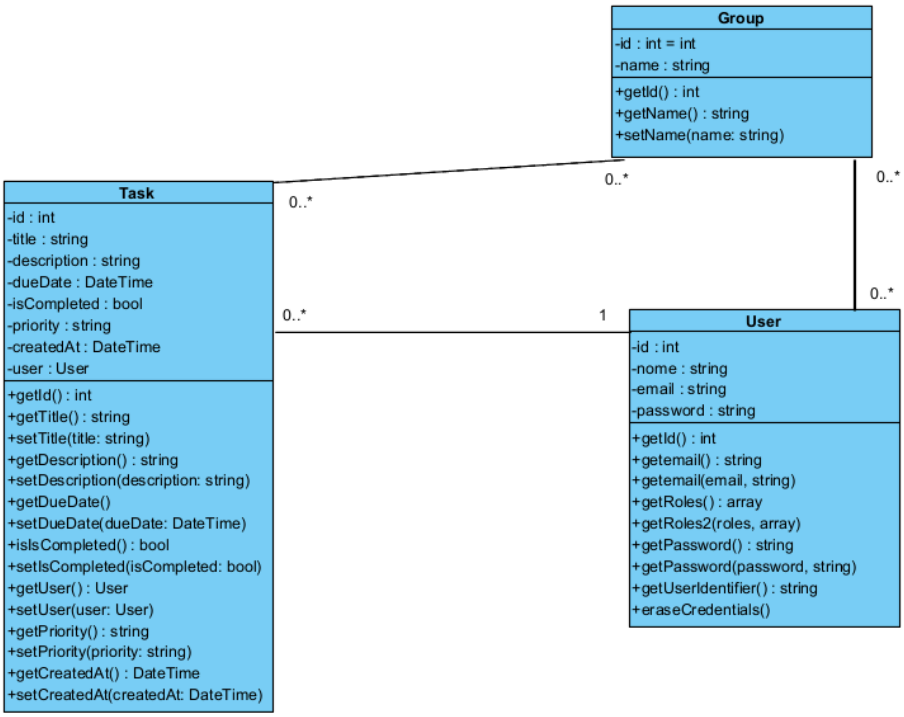
3.1 Diagrama de Casos de Uso



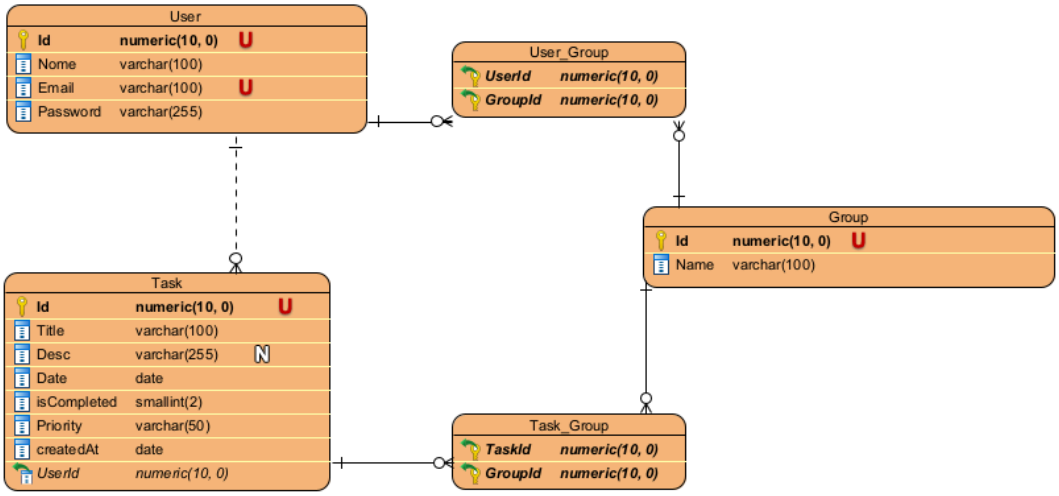
3.2 Diagrama de Sequência



3.3 iagrama de Classes



3.4 Modelo Relacional (MySQL)



3.5 Padrões de Projeto Utilizados

- Repository Pattern: Aplicado para abstrair o acesso ao banco de dados. Por exemplo, o TarefaRepository lida com criação, listagem e exclusão de tarefas sem que o controller precise acessar diretamente a camada de persistência.

- MVC – Model View Controller: Utilizado nativamente pelo Symfony. O código está organizado da seguinte forma:

    - Model: Entidades Usuario e Tarefa.

    - View: Templates em Twig.

    - Controller: Manipula as requisições e envia dados para o serviço ou view.

- Service Layer: Responsável pela lógica de negócio. Classes como TarefaService são usadas para processar dados antes de enviá-los ao repositório.

- DTO – Data Transfer Object: Usado para transferir dados entre as camadas de forma limpa, evitando vazamento de lógica entre controller e model.

4. Validação e Testes

4.1 Teste de Controller

Para verificar o correto funcionamento das rotas principais do sistema, foi criado um teste simples para o controller das tarefas. Este teste envia uma requisição HTTP do tipo GET para a rota /task e verifica se a resposta foi bem-sucedida (código HTTP 200), garantindo que a página de listagem de tarefas está acessível.

```
<?php

namespace App\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

final class TaskControllerTest extends WebTestCase
{
    public function testIndex(): void
    {
        $client = static::createClient();
        $client->request('GET', '/task');

        self::assertResponseIsSuccessful();
    }
}
```

Este teste assegura que a rota /task está funcionando corretamente, retornando uma resposta HTTP de sucesso, o que é fundamental para a funcionalidade principal de listar tarefas do sistema.

4.2 Testes Funcionais de Registro e Login

Para garantir que as funcionalidades de registro e autenticação de usuários funcionam corretamente, foram implementados testes funcionais que simulam o comportamento do usuário no sistema.

```
<?php

namespace App\Tests\Functional;

use App\Entity\User;
use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;
use Doctrine\ORM\EntityManagerInterface;

class RegistrationAndLoginTest extends WebTestCase
{
    private ?EntityManagerInterface $entityManager;
    private ?UserPasswordHasherInterface $passwordHasher;

    protected function setUp(): void
    {
        parent::setUp();
        $kernel = self::bootKernel();
        $this->entityManager =
$kernel->getContainer()->get('doctrine')->getManager();
        $this->passwordHasher =
$kernel->getContainer()->get(UserPasswordHasherInterface::class);
    }

    protected function tearDown(): void
    {
        parent::tearDown();
        $this->entityManager->close();
        $this->entityManager = null;
        $this->passwordHasher = null;
    }

    public function testRegistration(): void
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/register');

        $this->assertResponseIsSuccessful();
        $this->assertSelectorTextContains('h1', 'REGISTRE-SE');

        $form = $crawler->selectButton('Criar Conta')->form([
            'registration_form[email]' => 'test_register_func@example.com',
            'registration_form[plainPassword]' => 'password123',
            'registration_form[agreeTerms]' => true,
        ]);

        $client->submit($form);

        $this->assertResponseRedirects('/login');
```

```
        $crawler = $client->followRedirect();
        $this->assertSelectorTextContains('.flash-success', 'Sua conta foi
criada com sucesso!');

        $userRepository = $this->entityManager->getRepository(User::class);
        $user = $userRepository->findOneBy(['email' =>
'test_register_func@example.com']);
        $this->assertNotNull($user);
        $this->assertTrue($this->passwordHasher->isPasswordValid($user,
'password123'));
    }

    public function testLoginSuccessful(): void
    {
        $user = new User();
        $user->setEmail('test_login_func@example.com');
        $user->setPassword($this->passwordHasher->hashPassword($user,
'password123'));
        $user->setRoles(['ROLE_USER']);
        $this->entityManager->persist($user);
        $this->entityManager->flush();

        $client = static::createClient();
        $crawler = $client->request('GET', '/login');

        $this->assertResponseIsSuccessful();
        $this->assertSelectorTextContains('h1', 'LOGIN');

        $form = $crawler->selectButton('Entrar')->form([
            'email' => 'test_login_func@example.com',
            'password' => 'password123',
        ]);

        $client->submit($form);

        $this->assertResponseRedirects('/task');

        $crawler = $client->followRedirect();
        $this->assertSelectorTextContains('h1', 'Minhas Tarefas');
    }

    public function testLoginFailure(): void
    {
        $client = static::createClient();
        $crawler = $client->request('GET', '/login');

        $form = $crawler->selectButton('Entrar')->form([
            'email' => 'nonexistent@example.com',
            'password' => 'wrongpassword',
        ]);

        $client->submit($form);

        $this->assertResponseRedirects('/login');
        $crawler = $client->followRedirect();
        $this->assertSelectorTextContains('.auth-alert-error', 'Credenciais
```

```
inválidas.');
    }
}
```

4.3 Testes de Unidade das Entidades

Para garantir o correto funcionamento das entidades do sistema, foram desenvolvidos testes unitários para as classes Task e User. Esses testes verificam o comportamento dos métodos getters e setters, assegurando a integridade dos dados manipulados nas entidades.

Testes da entidade Task

```php
<?php

namespace App\Tests\Unit\Entity;

use App\Entity\Task;
use App\Entity\User;
use PHPUnit\Framework\TestCase;

class TaskTest extends TestCase
{
    public function testSetTitle(): void
    {
        $task = new Task();
        $title = 'Minha Nova Tarefa';
        $task->setTitle($title);
        $this->assertEquals($title, $task->getTitle());
    }

    public function testSetIsCompleted(): void
    {
        $task = new Task();
        $this->assertFalse($task->isIsCompleted()); // Valor padrão

        $task->setIsCompleted(true);
        $this->assertTrue($task->isIsCompleted());

        $task->setIsCompleted(false);
        $this->assertFalse($task->isIsCompleted());
    }

    public function testSetDueDate(): void
    {
        $task = new Task();
        $dueDate = new \DateTimeImmutable('2025-12-31');
        $task->setDueDate($dueDate);
        $this->assertEquals($dueDate, $task->getDueDate());
    }
```

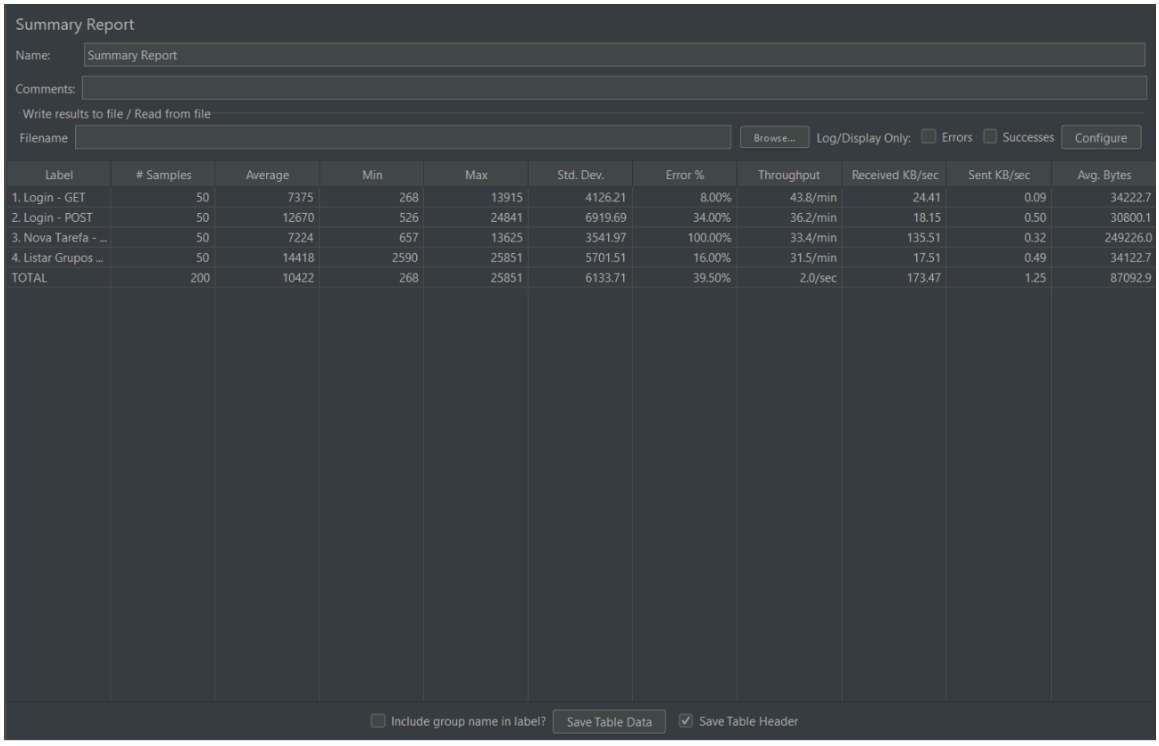
```
    public function testSetUser(): void
    {
        $task = new Task();
        $user = new User();
        $user->setEmail('test@user.com');

        $task->setUser($user);
        $this->assertEquals($user, $task->getUser());
    }

    public function testCreatedAtIsSetAutomatically(): void
    {
        $task = new Task();
        $this->assertInstanceOf(\DateTimeImmutable::class,
$task->getCreatedAt());
    }
}
```

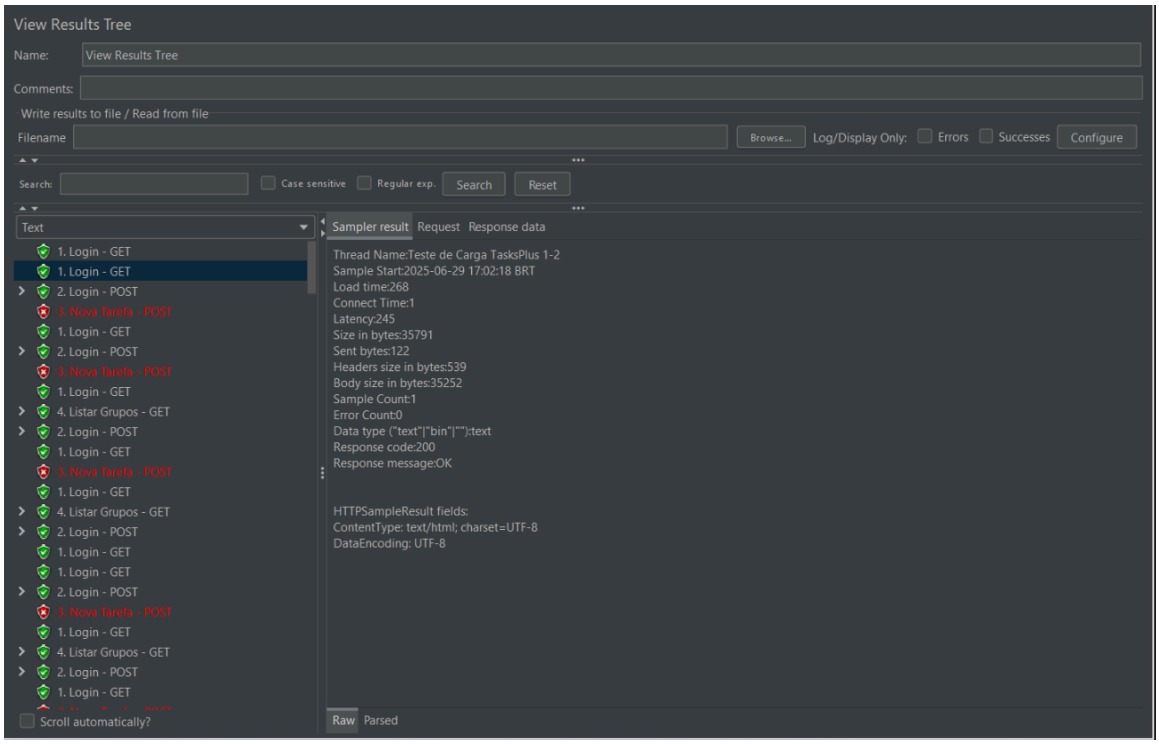
Testes da entidade User

```
<?php

namespace App\Tests\Unit\Entity;

use App\Entity\User;
use PHPUnit\Framework\TestCase;

class UserTest extends TestCase
{
    public function testSetEmail(): void
    {
        $user = new User();
        $email = 'test@example.com';
        $user->setEmail($email);
        $this->assertEquals($email, $user->getEmail());
        $this->assertEquals($email, $user->getUserIdentifier());
    }

    public function testSetPassword(): void
    {
        $user = new User();
        $hashedPassword =
'$2y$13$XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
        $user->setPassword($hashedPassword);
        $this->assertEquals($hashedPassword, $user->getPassword());
    }

    public function testSetRoles(): void
    {
        $user = new User();
        $user->setRoles(['ROLE_ADMIN']);
        $this->assertContains('ROLE_ADMIN', $user->getRoles());
```

```
        $this->assertContains('ROLE_USER', $user->getRoles());
        $this->assertCount(2, $user->getRoles());
    }

    public function testEraseCredentials(): void
    {
        $user = new User();
        $user->setPassword('some_password_hash');
        $user->eraseCredentials();
        $this->assertEquals('some_password_hash', $user->getPassword());
    }

}
```

Testes de Carga





Descrição:
Estes testes de unidade validam os principais métodos das entidades, garantindo que os valores sejam corretamente atribuídos e recuperados. O teste de Task também assegura que a data de criação (createdAt) é configurada automaticamente. Já o teste de User garante que a definição de e-mail, senha e papéis (roles) funcione conforme esperado, incluindo a preservação do papel padrão ROLE_USER.

Descrição dos testes:
- testRegistration: Simula a criação de uma nova conta, preenchendo o formulário de registro, verificando o redirecionamento e a criação do usuário no banco.
- testLoginSuccessful: Insere um usuário no banco e testa o login com credenciais válidas, aguardando o redirecionamento para a lista de tarefas.
- testLoginFailure: Tenta realizar login com credenciais inválidas, verificando que o sistema retorna mensagem de erro e não permite o acesso.

Resultados obtidos:
Os testes confirmaram o correto funcionamento das funcionalidades de registro, autenticação e tratamento de erros no processo de login.

5. Processo de Desenvolvimento

O desenvolvimento foi realizado de forma colaborativa entre os integrantes do grupo. Inicialmente, foram levantados os requisitos básicos do sistema com base nas funcionalidades esperadas. Em seguida, foi feita a modelagem dos diagramas utilizando o StarUML e Draw.io.
A implementação do sistema foi realizada utilizando o framework Symfony, com banco de dados MySQL, integrando os dados através do Doctrine ORM. A aplicação foi estruturada seguindo o padrão MVC, com controllers específicos para login, cadastro de usuários e manipulação de tarefas.

6. Conclusão

O sistema Controle de Tarefas proporcionou uma excelente oportunidade para aplicar os conceitos de engenharia de software de forma prática. A integração entre as disciplinas de

Engenharia de Software II e Programação III foi essencial para transformar a modelagem teórica em um produto funcional e com qualidade.

Com o uso de padrões como MVC e Repository, e a adoção do MySQL como banco de dados relacional, foi possível construir uma aplicação robusta, organizada e escalável. O projeto também reforçou a importância dos testes e da documentação técnica para garantir o bom funcionamento do sistema.

A colaboração em grupo, o uso de boas práticas e ferramentas modernas contribuíram para o desenvolvimento de um sistema completo e didaticamente relevante.