UNIVERSIDADE DO OESTE DE SANTA CATARINA Campus de São Miguel do Oeste

Curso: Ciência da Computação Área: Ciências Exatas e Tecnológicas Disciplina: Engenharia de Software II Professor: Roberson Junior Fernandes Alves

Gabriela Mendes Demossi

Gabriel Morin Werner

# TRABALHO A1

São Miguel do Oeste – SC

2025

## Descrição Geral do Projeto

O sistema desenvolvido é um aplicativo web em PHP utilizando CodeIgniter que permite o controle de caminhões de uma empresa e a gestão de usuários (colaboradores e administradores), com acesso restrito via login. O objetivo principal é oferecer um ambiente centralizado para registrar informações sobre caminhões (placa, cor, documentos, quilometragem, motorista) e manter usuários autenticados e autorizados, garantindo segurança e facilidade de manutenção.

O projeto integra modelagem de software, implementação de banco de dados e aplicação dos padrões de projeto para criar um sistema funcional e organizado.

## Modelagem de Software

### Diagrama de Casos de Uso

O diagrama ilustra claramente as ações disponíveis para os dois atores: Admin (com acesso completo, incluindo cadastro de usuários) e Usuário (restrito a ações sobre caminhões). As funcionalidades incluem login, visualização de caminhões, cadastro, edição, exclusão de caminhões e logout. Para o Admin, inclui também o cadastro de novos usuários.

Justificativa: separação das permissões foi essencial para garantir que apenas administradores tenham controle sobre a gestão de contas, enquanto usuários podem trabalhar exclusivamente com os registros de caminhões.

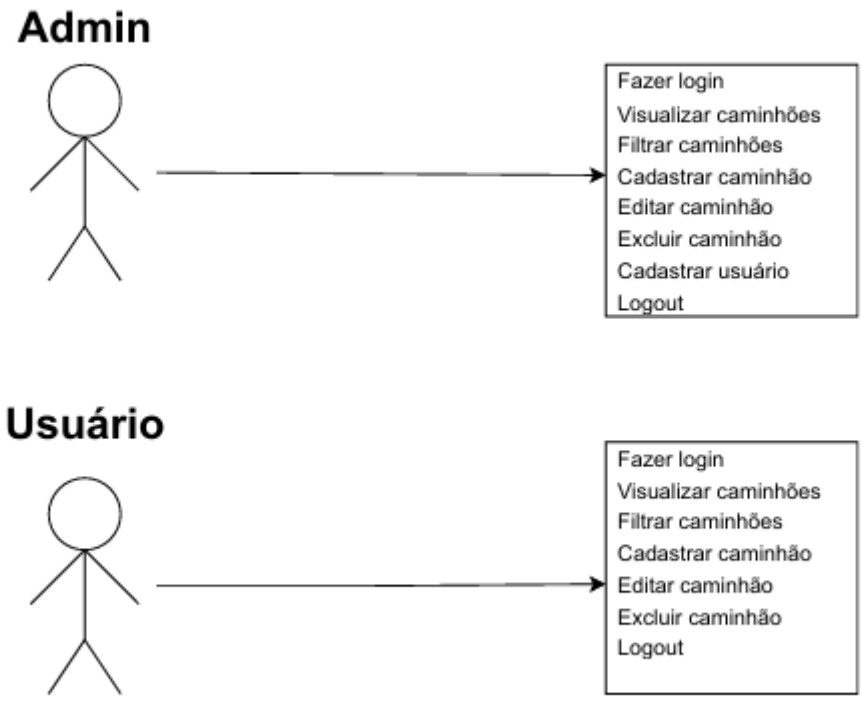



*Diagrama de Casos de Uso*

**Diagrama de Sequência**

*O diagrama de sequência representa fluxos críticos do sistema: login, visualização de caminhões e cadastro de caminhões. Ele demonstra a interação entre os objetos Usuário/Admin, Sistema Web e Banco de Dados, evidenciando a ordem das chamadas de métodos e retornos.*

***Justificativa:*** *foi importante modelar como as requisições trafegam do front-end até o banco, para garantir integridade dos dados e comportamento previsível do sistema.*


*Diagrama de Sequência*

***Diagrama de Classes***

O diagrama define as entidades Usuario, Caminhao, Sistema e Banco de Dados. Cada classe apresenta seus atributos e métodos principais, mostrando os relacionamentos e responsabilidades. A modelagem reflete a estrutura do código implementado, que segue o padrão MVC do CodeIgniter.

**Justificativa:** esse diagrama foi essencial para visualizar as responsabilidades de cada parte do sistema, reduzindo a complexidade durante o desenvolvimento e facilitando futuras manutenções.

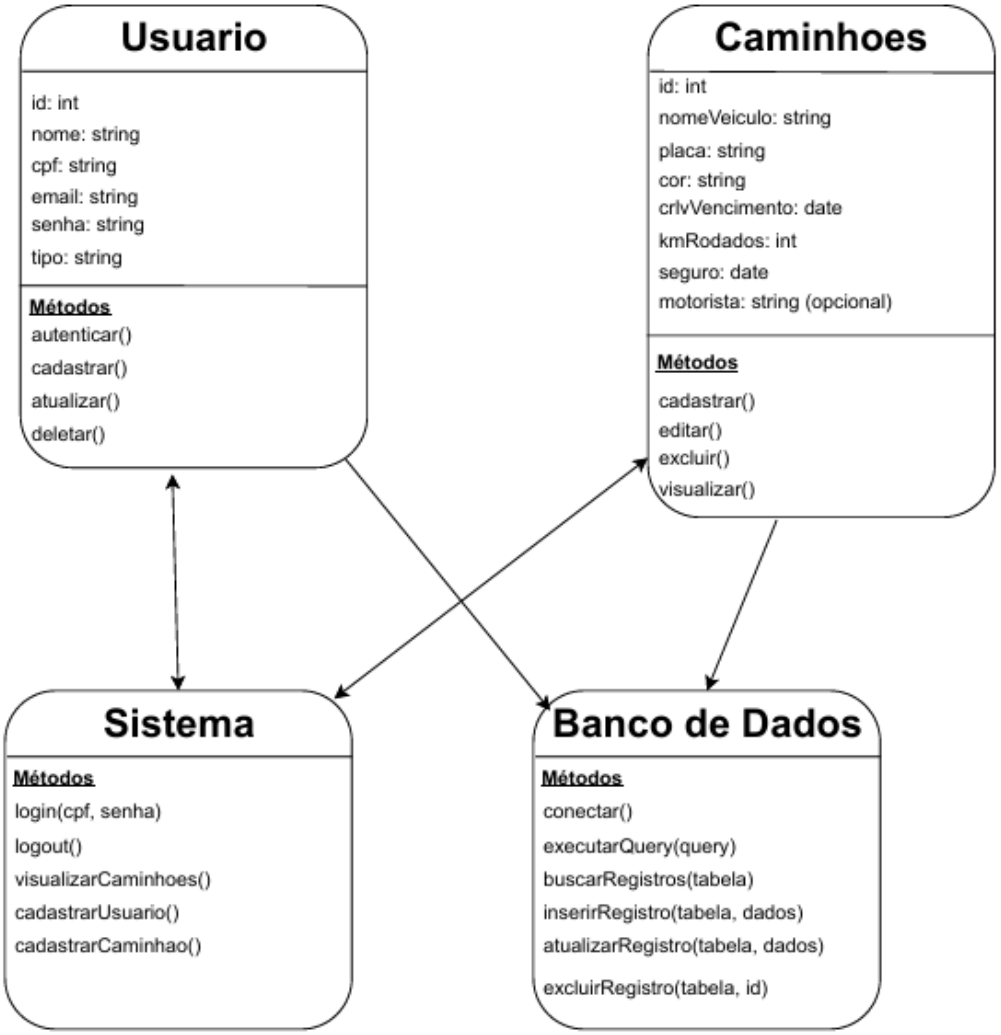
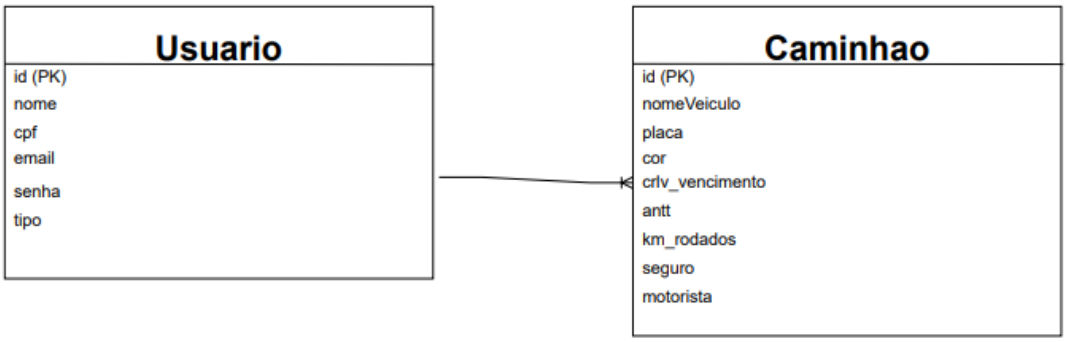



*Diagrama de Classes*

## Modelo Entidade-Relacionamento (ER)

O modelo ER apresenta as tabelas principais do banco: usuarios e caminhoes, com seus campos e relacionamentos. Ele demonstra como as entidades do sistema estão persistidas no banco de dados MySQL.

**Justificativa:** a estrutura do banco foi pensada para manter a coerência dos dados, garantindo integridade e facilidade de consultas e manutenção.




*Modelo Entidade-Relacionamento (ER)*

## Teste Unitário

Objetivo: Validar individualmente o comportamento básico do model CaminhaoModel garantindo que operações essenciais — como inserir, buscar e excluir registros — funcionem como esperado em isolamento.

O que foi testado:

- Inserção de um caminhão no banco em memória.
- Verificação se o caminhão pode ser buscado corretamente após a inserção.
- Exclusão do caminhão e confirmação de que ele foi removido do banco.

Como foi feito:

- O teste usa um banco SQLite em memória para manter os dados isolados a cada execução.
- Antes de cada teste, a tabela caminhoes é criada do zero, garantindo ambiente limpo.
- Foram realizadas 4 asserts para confirmar o sucesso das operações:
    - Confirma que o ID de inserção é numérico.
    - Verifica se o caminhão é retornado corretamente.

- Valida se os dados inseridos batem com os esperados.
- Confirma que o caminhão foi removido do banco.

Benefício: Esse teste garante que o model funcione corretamente independentemente de outras camadas do sistema (controllers ou views), identificando erros em alterações futuras.

Aqui o teste feito e seu resultado:

```php
<?php

use PHPUnit\Framework\TestCase;
use App\Models\CaminhaoModel;

class CaminhaoTest extends TestCase
{
    protected $caminhaoModel;

    protected function setUp(): void
    {
        parent::setUp();

        // Sobrescreve a conexão do model para usar SQLite em memória
        $this->db = \Config\Database::connect([
            'DBDriver' => 'SQLite3',
            'database' => ':memory:',
        ]);

        // Cria a tabela necessária para os testes
        $this->db->query("
            CREATE TABLE caminhoes (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                nome_veiculo TEXT,
                placa TEXT UNIQUE,
                cor TEXT,
                crlv_vencimento TEXT,
                antt TEXT,
                km_rodados INTEGER,
                seguro TEXT,
                motorista TEXT,
                created_at TEXT,
                updated_at TEXT
            );
        ");
```

*Código teste unitário 1*

```
        ");

        // Inicializa o model usando a conexão de teste
        $this->caminhaoModel = new CaminhaoModel($this->db);
    }

    public function testInserirBuscarExcluirCaminhao()
    {
        $dados = [
            'nome_veiculo'    => 'Teste Unidade',
            'placa'           => 'TEST1234',
            'cor'             => 'VERDE',
            'crlv_vencimento' => '2026-12-31',
            'antt'            => '12345678',
            'km_rodados'      => 10000,
            'seguro'          => '2025-12-31',
            'motorista'       => 'Motorista Teste'
        ];

        $id = $this->caminhaoModel->insert($dados);
        $this->assertIsNumeric($id, "Falha ao inserir caminhão: id não gerado.");

        $caminhao = $this->caminhaoModel->find($id);
        $this->assertNotNull($caminhao, "Caminhão inserido não foi encontrado.");
        $this->assertEquals('TEST1234', $caminhao['placa'], "Placa não corresponde.");

        $this->caminhaoModel->delete($id);
        $caminhaoRemovido = $this->caminhaoModel->find($id);
        $this->assertNull($caminhaoRemovido, "Caminhão não foi removido do banco.");
    }
}
?>
```

*Código Teste Unitário 2*

```
PS C:\xampp\htdocs\controle-caminhoes> vendor\bin\phpunit tests\unit/CaminhaoTest.php
PHPUnit 11.5.25 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12
Configuration: C:\xampp\htdocs\controle-caminhoes\phpunit.xml.dist

.

Time: 00:00.018, Memory: 12.00 MB

There was 1 PHPUnit test runner warning:

1) No code coverage driver available

OK, but there were issues!
Tests: 1, Assertions: 4, PHPUnit Warnings: 1.
```

*Resultado Teste Unitário*

**Teste de Funcionalidade**

Objetivo: Validar cenários do fluxo de negócio relacionados ao cadastro de caminhões, simulando comportamentos mais próximos do uso real do sistema.

O que foi testado:
Inserção bem-sucedida de um caminhão:

- Insere um caminhão completo e verifica se os dados persistem corretamente no banco.

Bloqueio de placas duplicadas:

- Tenta inserir dois caminhões com a mesma placa.
- Espera que o sistema lance um erro de duplicidade (chave única), garantindo a integridade dos dados.

Como foi feito:

- Cada teste usa um banco SQLite em memória limpo, configurado com DROP TABLE IF EXISTS para garantir isolamento.
- Em cada execução, a tabela caminhoes é recriada.
- O teste de duplicidade captura a exceção esperada e valida que a mensagem contém a palavra “UNIQUE”.

Benefício: Esses testes simulam situações reais do cadastro de caminhões, como cenários de sucesso e erros, prevenindo bugs críticos relacionados à integridade de dados no sistema.

Conclusão:
Esses dois testes são complementares:

- O teste unitário garante que as operações básicas do model funcionam de forma isolada.
- O teste de funcionalidade verifica cenários do fluxo de negócio, validando que regras como unicidade de placas são aplicadas corretamente.

Eles reforçam a qualidade do sistema, facilitam manutenção e aumentam a confiança em futuras evoluções.

Aqui o teste feito e seu resultado:

```php
<?php

use PHPUnit\Framework\TestCase;
use App\Models\CaminhaoModel;

class CaminhaoFuncionalidadeTest extends TestCase
{
    protected $caminhaoModel;

    protected function setUp(): void
{
    parent::setUp();

    // Banco em memória
    $this->db = \Config\Database::connect([
        'DBDriver' => 'SQLite3',
        'database' => ':memory:',
    ]);

    // Drop para evitar erro se a tabela já existir
    $this->db->query("DROP TABLE IF EXISTS caminhoes");

    // Criação da tabela para os testes
    $this->db->query("
        CREATE TABLE caminhoes (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            nome_veiculo TEXT,
            placa TEXT UNIQUE,
            cor TEXT,
            crlv_vencimento TEXT,
            antt TEXT,
            km_rodados INTEGER,
            seguro TEXT,
            motorista TEXT,
            created_at TEXT,
            updated_at TEXT
        );
```

*Código teste de Funcionalidade 1*

```
    ");

    $this->caminhaoModel = new CaminhaoModel($this->db);
}

    public function testInserirCaminhaoComSucesso()
    {
        $dados = [
            'nome_veiculo'    => 'Funcional Scania',
            'placa'           => 'XYZ5678',
            'cor'             => 'AZUL',
            'crlv_vencimento' => '2026-05-10',
            'antt'            => '87654321',
            'km_rodados'      => 20000,
            'seguro'          => '2025-07-15',
            'motorista'       => 'Motorista Funcional'
        ];

        $id = $this->caminhaoModel->insert($dados);
        $this->assertIsNumeric($id, "Caminhão não foi inserido com sucesso.");

        $caminhao = $this->caminhaoModel->find($id);
        $this->assertEquals('XYZ5678', $caminhao['placa']);

        // Limpeza: remover o registro
        $this->caminhaoModel->delete($id);
    }

    public function testNaoPermitePlacaDuplicada()
    {
        $dados = [
            'nome_veiculo'    => 'Primeiro Caminhão',
            'placa'           => 'DUPLIC123',
            'cor'             => 'PRETO',
            'crlv_vencimento' => '2025-01-01',
```

*Código teste de Funcionalidade 2*

```php
    public function testNaoPermitePlacaDuplicada()
    {
        $dados = [
            'nome_veiculo'    => 'Primeiro Caminhão',
            'placa'           => 'DUPLIC123',
            'cor'             => 'PRETO',
            'crlv_vencimento' => '2025-01-01',
            'antt'            => '11223344',
            'km_rodados'      => 15000,
            'seguro'          => '2024-12-31',
            'motorista'       => 'Teste Dup'
        ];

        // Insere o primeiro caminhão
        $id1 = $this->caminhaoModel->insert($dados);
        $this->assertIsNumeric($id1, "Primeiro caminhão não foi inserido.");

        // Tentar inserir o segundo com a mesma placa
        try {
            $this->caminhaoModel->insert($dados);
            $this->fail("Esperava falha ao inserir placa duplicada, mas não ocorreu.");
        } catch (\Exception $e) {
            $this->assertStringContainsString('UNIQUE', $e->getMessage());
        }

        // Limpeza: remover registro
        $this->caminhaoModel->delete($id1);
    }
}
```

*Código teste de Funcionalidade 3*

```
PS C:\xampp\htdocs\controle-caminhoes> vendor\bin\phpunit tests\unit/CaminhaoFuncionalidadeTest.php
PHPUnit 11.5.25 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12
Configuration: C:\xampp\htdocs\controle-caminhoes\phpunit.xml.dist

..

Time: 00:00.019, Memory: 12.00 MB

There was 1 PHPUnit test runner warning:

1) No code coverage driver available

OK, but there were issues!
Tests: 2, Assertions: 4, PHPUnit Warnings: 1.
```

*Resultado Teste de Funcionalidade*

## Teste de carga

Para validar o desempenho do sistema, realizamos um teste de carga utilizando o Apache JMeter, simulando requisições simultâneas ao endpoint de listagem de caminhões.

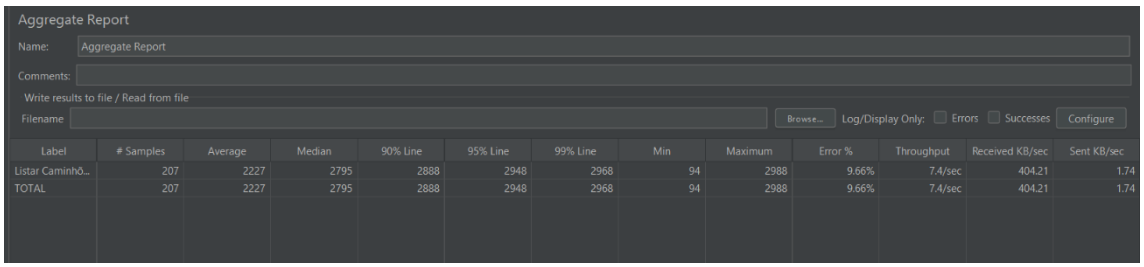
**Configuração do Teste:**

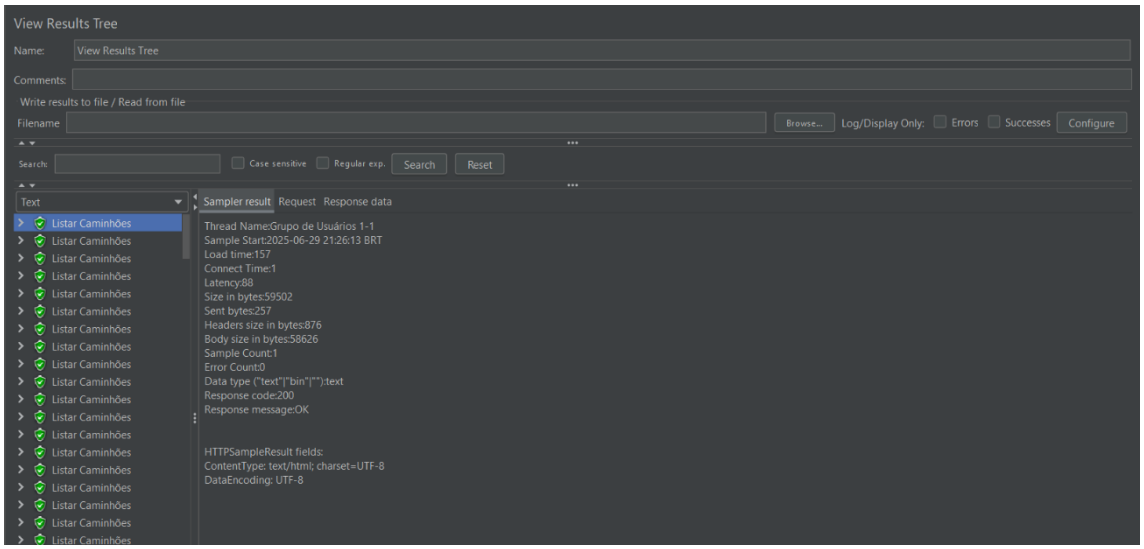
- Endpoint testado: Página de listagem de caminhões.
- Usuários simultâneos: configurados em grupos para simular acessos concorrentes.
- Métricas monitoradas: tempo médio de resposta, latência, throughput, taxa de erro.

**Resultados principais:**

- Conforme o **Aggregate Report**, obtivemos média de resposta em ~2227 ms com throughput de 7.4 requisições/segundo.
- A taxa de erros foi de 9.66%, indicando algumas falhas sob carga alta.
- O **View Results Tree** demonstrou respostas detalhadas com código HTTP 200 (OK) na maioria das requisições.

Esses dados confirmam que o sistema suporta acessos simultâneos moderados, mas sugerem oportunidades de otimização para reduzir erros sob alta concorrência.




View Results Tree
Name:
View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only:
Errors
Successes
Configure
Search:
Case sensitive
Regular exp.
Search
Reset
Text
Sampler result
Request
Response data
Listar Caminhões
Thread Name:Grupo de Usuários 1-1
Sample Start:2025-06-29 21:26:13 BRT
Load time:157
Connect Time:1
Latency:88
Size in bytes:59502
Sent bytes:257
Headers size in bytes:876
Body size in bytes:58626
Sample Count:1
Error Count:0
Data type ("text"|"bin"|""):text
Response code:200
Response message:OK
HTTPSampleResult fields:
ContentType: text/html; charset=UTF-8
DataEncoding: UTF-8