Sistema de Gerenciamento para Barbearias: Um Estudo de Caso do DevBarberShop

Curso: Ciência da Computação

Johnny Matheus N. de Medeiro

Nathaniel Nicolas Rissi Soares

Nelson Ramos Rodrigues Junior

Resumo

O presente artigo descreve o desenvolvimento de um sistema web para gerenciamento de uma barbearia, incluindo controle de clientes, agendamentos, pagamentos, produtos, serviços e funcionários. O objetivo principal foi proporcionar uma solução centralizada e de fácil uso, empregando o framework CodeIgniter 4 e banco de dados PostgreSQL. Foram aplicados princípios de arquitetura MVC, com interface responsiva construída em Bootstrap 5. Testes unitários, funcionais e de carga foram realizados para validar a robustez e o desempenho da aplicação.A documentação desse projeto poderá ser acessada em nosso github disponível em: https://github.com/JohnnyMatheus/Sistema-Gerenciamento-para-Barbearia. O Vídeo da apresentação da programação 3 está disponível em nosso canal do youtube: https://www.youtube.com/watch?v=JFX14Llh-3U e o vídeo de engenharia de software também está disponível em nosso canal falando sobre o conteúdo solicitado pelo professor Roberson. Disponível em : https://www.youtube.com/watch?v=I9paBoALR_c

Palavras-chave:Gestão de barbearias; CodeIgniter 4; PostgreSQL; Bootstrap 5; MVC; testes de software.

1 Introdução:

Gerenciar operações em barbearias exige um sistema eficiente e robusto, que integre funções administrativas e operacionais de forma automatizada. O projeto DevBarberShop foi desenvolvido para abordar esses desafios, permitindo maior controle sobre dados de clientes, serviços e produtos. Com base nos conhecimentos de disciplinas específicas do curso de Ciência da Computação, o sistema foi projetado para ser funcional, seguro e escalável. Além disso, o projeto é fundamentado em práticas de modelagem de sistemas e desenvolvimento orientado a objetos.

2 Desenvolvimento

A documentação desse projeto poderá ser acessada em nosso github disponível em: https://github.com/JohnnyMatheus/Sistema-Gerenciamento-para-Barbearia. O Vídeo da apresentação da programação 3 está disponível em nosso canal do youtube: https://www.youtube.com/watch?v=JFX14Llh-3U e o vídeo de engenharia de software também está disponível em nosso canal falando sobre o conteúdo solicitado pelo professor Roberson. Disponível em : https://www.youtube.com/watch?v=I9paBoALR_c .

2.1 Descrição do Projeto

O DevBarberShop centraliza as principais operações de uma barbearia em uma plataforma única, possibilitando:

- Cadastro de clientes, fornecedores e funcionários.
- Gerenciamento de produtos e serviços.
- Controle de agendamentos e pagamentos.
- Geração de relatórios e análise de históricos de serviços.

2.2 Levantamento de Requisitos

2.2.1 Funcionais:

- Cadastro de Clientes: O sistema deve permitir o cadastro de clientes com informações como nome, telefone, e-mail, endereço, sexo e data de nascimento.
- Cadastro de Fornecedores: O sistema deve permitir o cadastro de fornecedores com informações como nome, telefone, e-mail e endereço.
- Cadastro de Funcionários: O sistema deve permitir o cadastro de funcionários com informações como nome, telefone, e-mail, cargo e salário.
- Cadastro de Serviços: O sistema deve permitir o cadastro de serviços oferecidos pela barbearia, incluindo nome, descrição e preço.
- Cadastro de Produtos: O sistema deve permitir o cadastro de produtos, incluindo nome, descrição, quantidade em estoque, preço, e o fornecedor responsável.
- Agendamento de Serviços: O sistema deve permitir que os clientes agendem serviços com os funcionários, especificando a data, hora e status do agendamento.
- Registro de Pagamentos: O sistema deve permitir o registro de pagamentos realizados pelos clientes para os serviços agendados, incluindo valor, forma de pagamento e data/hora do pagamento.
- Histórico de Serviços: O sistema deve registrar o histórico de serviços prestados, com detalhes sobre o serviço realizado, o cliente atendido, e o funcionário responsável.
- Relatórios e Consultas: O sistema deve possibilitar a consulta e a geração de relatórios sobre agendamentos, pagamentos e histórico de serviços.

2.2.2 Não Funcionais:

- Desempenho: O sistema deve ser capaz de processar as transações de agendamento e pagamento rapidamente, sem causar lentidão no sistema.
- Segurança: O sistema deve garantir a segurança dos dados, com criptografia das informações sensíveis, como e-mails e números de telefone, além de controle de acesso para diferentes perfis (clientes, funcionários e administradores).
- Escalabilidade: O sistema deve ser capaz de se adaptar ao crescimento da barbearia, permitindo a adição de novos funcionários, clientes, serviços e produtos sem impacto significativo no desempenho.
-
- Compatibilidade: O sistema deve ser compatível com diferentes dispositivos e navegadores, garantindo uma experiência fluida para os usuários.
- Usabilidade: A interface do sistema deve ser simples, intuitiva e fácil de navegar, com feedback claro para o usuário em cada interação.
- Backup e Recuperação de Dados: O sistema deve permitir a criação de backups periódicos dos dados, garantindo que as informações possam ser recuperadas em caso de falha.
- Acessibilidade: O sistema deve ser acessível a pessoas com deficiência, seguindo as diretrizes de acessibilidade web, como W3C WCAG.

2.2.3 Requisitos de Domínio:

🔸 Cliente:

- Cada cliente deve ter um código de identificação único, nome, telefone, e-mail, endereço (bairro, rua, cidade), sexo e data de nascimento.
- Os dados do cliente são essenciais para o agendamento de serviços e registro de histórico.

🔸 Fornecedor:

- Cada fornecedor deve ter um código único, nome, telefone, e-mail e endereço, facilitando o controle dos produtos e insumos.

🔸 Funcionário:

- Os funcionários devem ter um código único, nome, telefone, e-mail, cargo e salário.
- Funcionários são essenciais para o agendamento e execução de serviços.

🔸 Serviço:

- Cada serviço deve ter um código, nome, descrição e preço.
- Os serviços são agendados pelos clientes e registrados no histórico de serviços prestados.

🔸 Produto:

- Cada produto deve ter um código único, nome, descrição, quantidade em estoque e preço.
- Produtos são vinculados a fornecedores e são essenciais para a execução de certos serviços.

🔸 Agendamento:

- Cada agendamento é identificado por um código e contém a data/hora do serviço, cliente e funcionário responsáveis.
- Os agendamentos representam a reserva de um serviço e devem ser gerenciados de acordo com políticas de cancelamento e alteração.

🔸 Pagamento:

- Pagamentos são identificados por um código e contêm informações de valor, data/hora e forma de pagamento, além do cliente e agendamento vinculados.
- O pagamento finaliza a prestação de serviço.

🔸 Histórico de Serviços:

- O histórico de serviços contém registros de todos os serviços realizados, com data/hora, serviço prestado, cliente e funcionário envolvidos.
- Este histórico é importante para consultas futuras sobre serviços realizados e pode ser usado para análise do perfil do cliente.

2.3 Prototipagem e Modelagem

O sistema foi modelado utilizando UML no Visual Paradigm e draw.io, com os seguintes diagramas:

- Diagrama de Casos de Uso: Define as principais interações entre usuários e o sistema.
- Diagrama de Sequência: Representa o fluxo de execução das operações.
- Diagrama de Atividades e Estados: Mapeia os estados do sistema em diferentes cenários.
- Diagrama de Classes: Estrutura as entidades e suas relações no sistema.

A seguir, serão apresentados imagens do protótipo criado tela de login e os diagramas :

Nas figuras 1 e 2 são apresentadas imagens do protótipo:

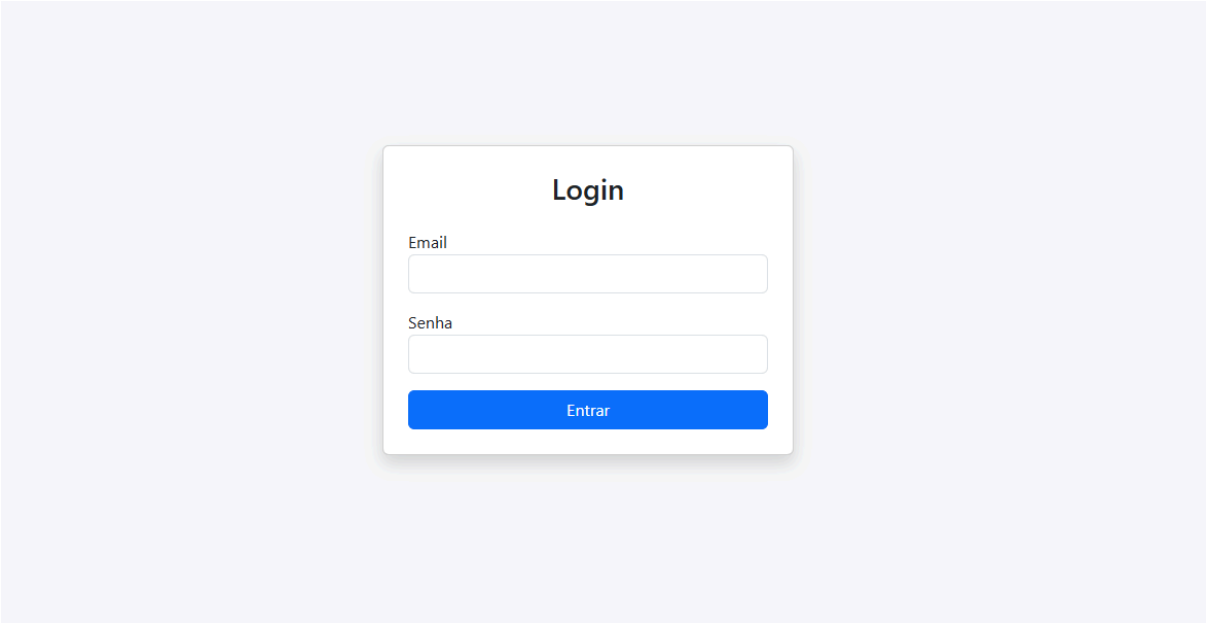


Figura 1: Página de login

Figura 2: Tela do sistema

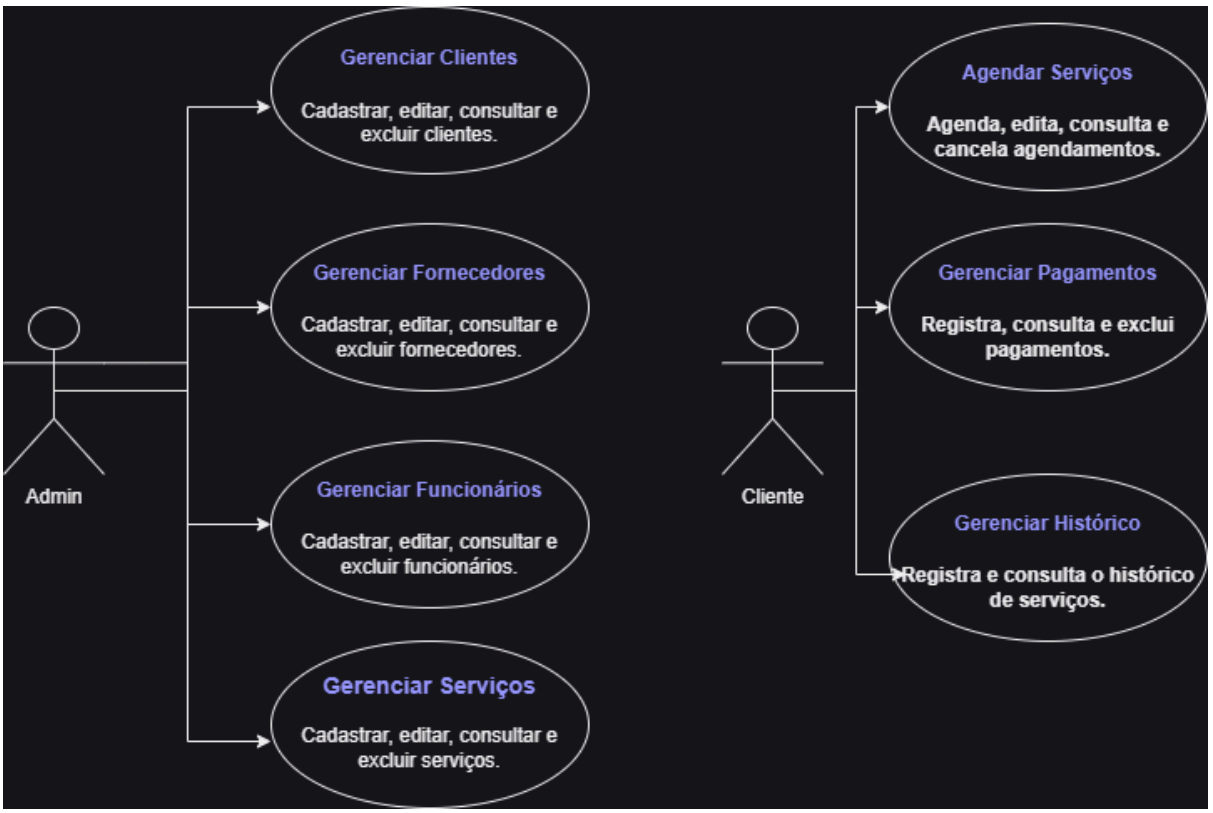


Figura 3: Diagrama de Caso de Uso

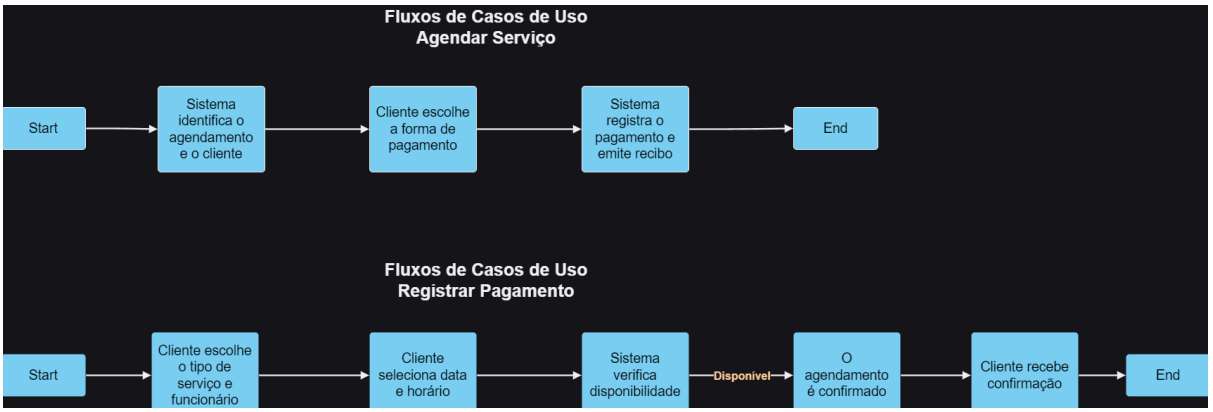


Figura 4: Diagrama de Fluxo

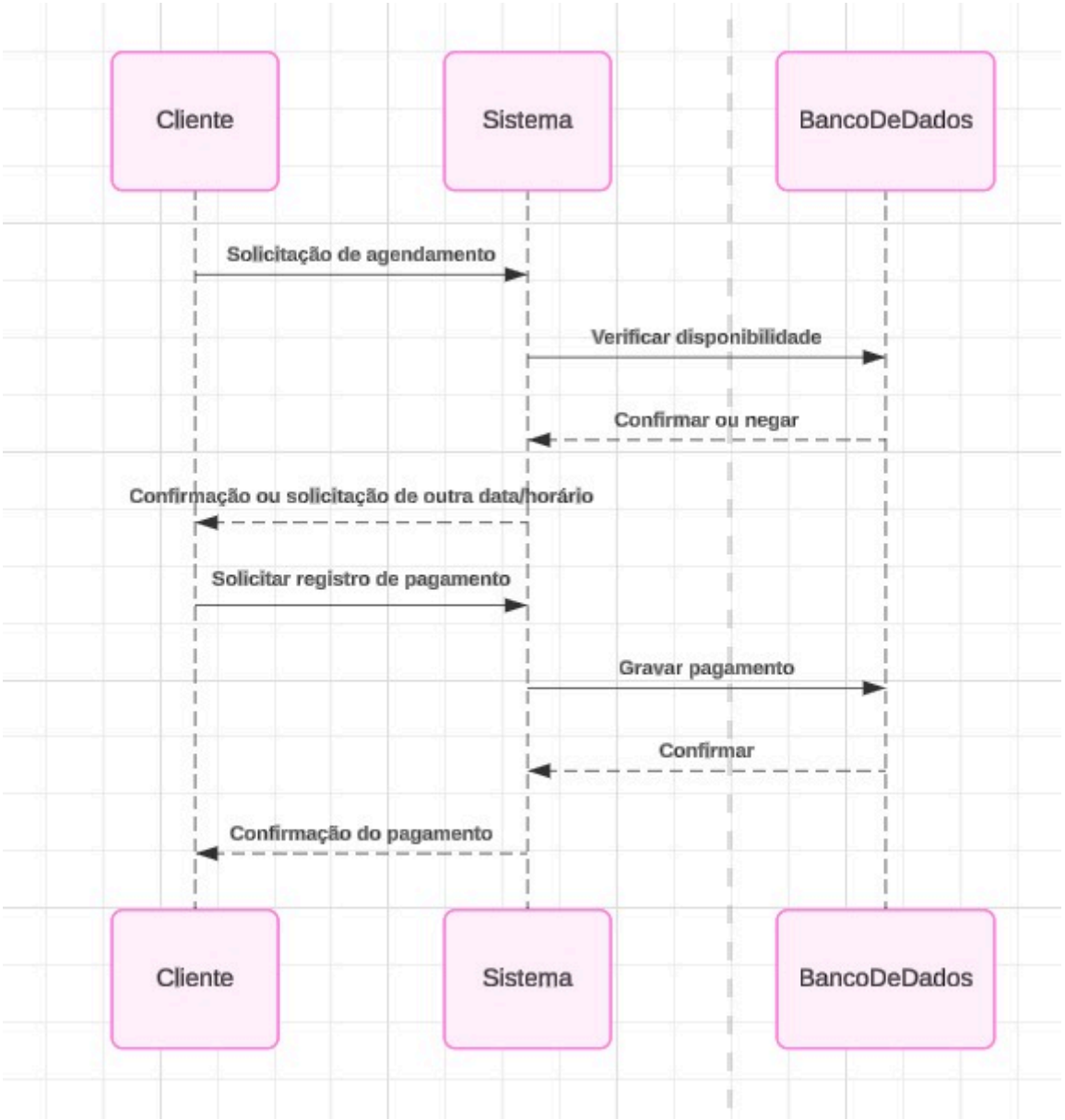


Figura 5: Diagrama de sequência

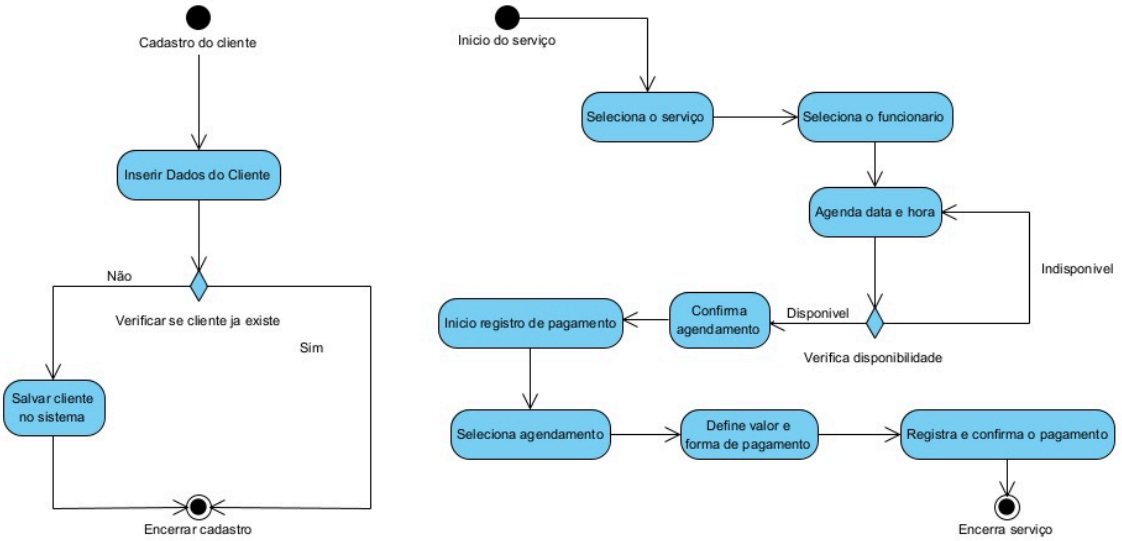


Figura 6: Diagrama de atividades

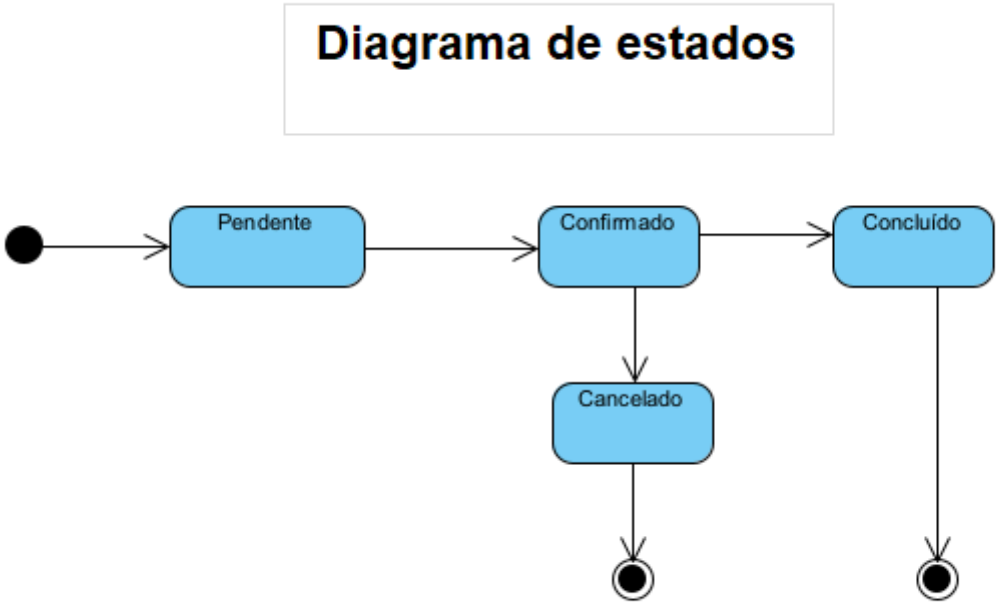


Figura 7: Diagrama de estados

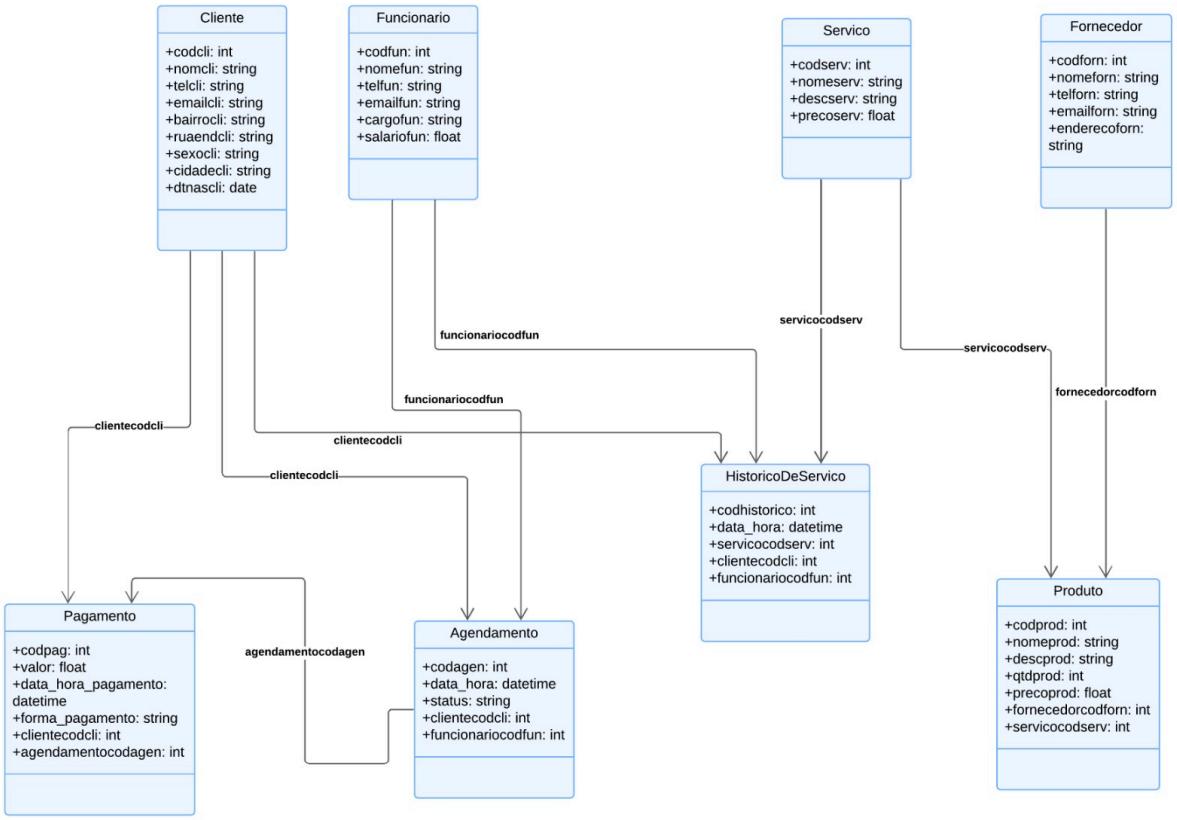


Figura 8: Diagrama de classes

2.4 Ferramentas Utilizadas

HTML5: Linguagem de marcação padrão para construção de páginas web, utilizada na estruturação dos elementos e conteúdos do sistema DevBarberShop.

CSS3: Folhas de estilo em cascata que permitem personalizar e padronizar a aparência visual do sistema, contribuindo para uma interface moderna e responsiva.

Bootstrap 5: Framework front-end que fornece componentes prontos e estilos padronizados, acelerando o desenvolvimento da interface com design responsivo.

PHP: Linguagem de programação de código aberto, amplamente utilizada para aplicações web dinâmicas, empregada na construção da lógica de negócios do sistema.

PostgreSQL: Sistema gerenciador de banco de dados relacional, escolhido por sua robustez, escalabilidade e suporte a recursos avançados de integridade e performance.

DBeaver: Ferramenta gráfica de gerenciamento de bancos de dados, utilizada para criação de tabelas, visualização de dados e execução de scripts SQL.

CodeIgniter 4: Framework PHP que adota a arquitetura MVC (Model-View-Controller), promovendo organização, segurança e facilidade de manutenção no código.

GitHub Desktop: Aplicativo que facilita o controle de versão e a colaboração entre os integrantes do grupo, por meio de um repositório no GitHub.

XAMPP: Pacote que integra servidor Apache, PHP e banco de dados, utilizado para testar o sistema localmente de forma simples e rápida.

Visual Studio Code (VSCode): Editor de código moderno, com recursos de autocompletar, debug e integração de extensões, que otimizou a produtividade da equipe no desenvolvimento.

Katalon Studio:Ferramenta de automação de testes que possibilita criação de testes de interface e API.

Apache JMeter: Ferramenta de testes de desempenho e carga para aplicações web.

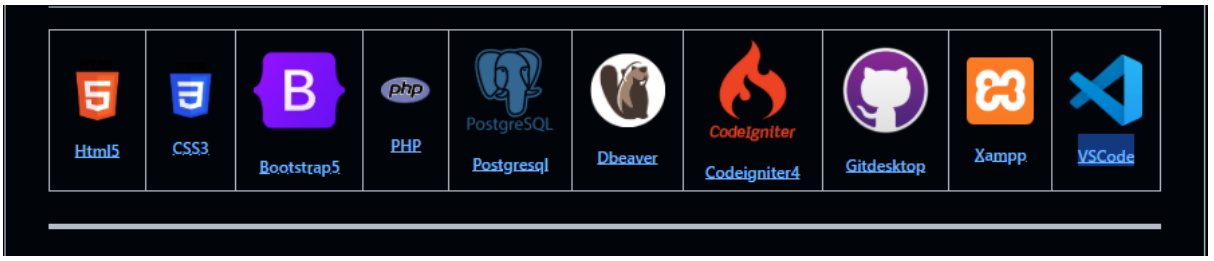


Figura 9: Ferramentas utilizadas

2.5 Implementação do Banco de Dados

O banco de dados foi estruturado para garantir integridade e eficiência, contendo as tabelas:

- Usuarios
- Cliente
- Fornecedor
- Funcionário
- Serviço
- Produto
- Agendamento
- Pagamento

A seguir a imagem do modelo ER(Entidade Relacionamento), feito no Visual Paradigm:

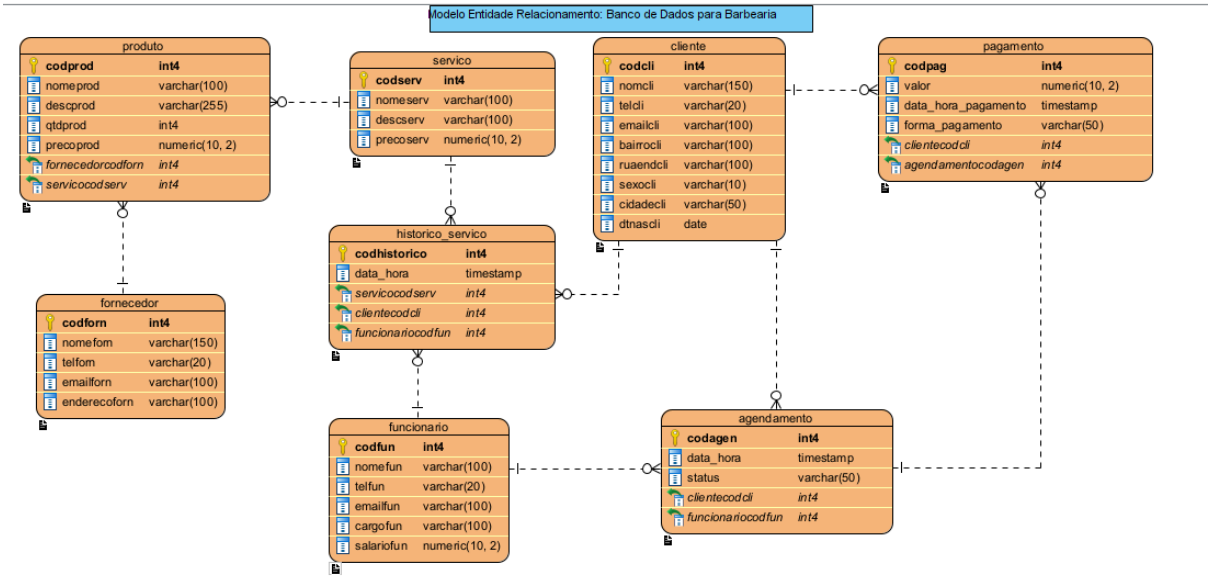


Figura 10: Modelo Entidade

2.5.1 Tabela Usuário

```
create table usuarios (
  id serial primary key,
  nome varchar(100) not null,
  email varchar(100) not null unique,
  senha varchar(255) not null
);
comment on table usuarios is 'Tabela de usuários do sistema';
comment on column usuarios.id is 'ID do usuário';
comment on column usuarios.nome is 'Nome do usuário';
comment on column usuarios.email is 'Email do usuário';
comment on column usuarios.senha is 'Senha criptografada';
insert into usuarios (nome, email, senha)

values ('Administrador', 'admin@devbarbershop.com', crypt('admin123', gen_salt('bf')));
```

Fonte: Os autores

2.5.2 Tabela Cliente

```
create table cliente(
  codcli int generated by default as identity primary key,
  nomcli varchar(150) not null,
  telcli varchar(20) not null,
  emailcli varchar(100) not null,
  bairrocli varchar(100) not null,
  ruaendcli varchar(100) not null,
  sexocli varchar(10) not null,
  cidadecli varchar(50) not null,
  dtnascli date not null
);
comment on table cliente is 'Tabela de clientes da Barbearia';
comment on column cliente.codcli is 'Código do cliente';
comment on column cliente.nomcli is 'Nome do cliente';
comment on column cliente.telcli is 'Telefone do cliente';
comment on column cliente.emailcli is 'E-mail do cliente';
comment on column cliente.bairrocli is 'Bairro do cliente';
comment on column cliente.ruaendcli is 'Rua do endereço do cliente';
comment on column cliente.sexocli is 'Sexo do cliente';
comment on column cliente.cidadecli is 'Cidade do cliente';
comment on column cliente.dtnascli is 'Data de nascimento do cliente';
-------------------------------------------------------------------------------------
```

Fonte: Os autores

2.5.3 Tabela Fornecedor

```
create table fornecedor(
  codforn int generated by default as identity primary key,
  nomeforn varchar(150) not null,
  telforn varchar(20),
  emailforn varchar(100),
  enderecoforn varchar(100)
);
comment on table fornecedor is 'Tabela de fornecedores';
comment on column fornecedor.codforn is 'Código do fornecedor';
comment on column fornecedor.nomeforn is 'Nome do fornecedor';
comment on column fornecedor.telforn is 'Telefone do fornecedor';
comment on column fornecedor.emailforn is 'E-mail do fornecedor';
comment on column fornecedor.enderecoforn is 'Endereço do fornecedor';
```

Fonte: Os autores

2.5.4 Tabela Funcionário

```
create table funcionario(
  codfun int generated by default as identity primary key,
  nomefun varchar(100) not null,
  telfun varchar(20) not null,
  emailfun varchar(100) not null,
  cargofun varchar(100) not null,
  salariofun numeric(10, 2) not null
);
comment on table funcionario is 'Tabela de funcionários da Barbearia';
comment on column funcionario.codfun is 'Código do funcionário';
comment on column funcionario.nomefun is 'Nome do funcionário';
comment on column funcionario.telfun is 'Telefone do funcionário';
comment on column funcionario.emailfun is 'E-mail do funcionário';
comment on column funcionario.cargofun is 'Cargo do funcionário';
comment on column funcionario.salariofun is 'Salário do funcionário';
```

Fonte: Os autores

2.5.5 Tabela Serviço

```
create table servico(
  codserv int generated by default as identity primary key,
  nomeserv varchar(100) not null,
  descserv varchar(100),
  precoserv numeric(10, 2) not null
);
comment on table servico is 'Tabela de serviços oferecidos pela Barbearia';
comment on column servico.codserv is 'Código do Serviço';
comment on column servico.nomeserv is 'Nome do serviço';
comment on column servico.descserv is 'Descrição do serviço oferecido pela Barbearia';
comment on column servico.precoserv is 'Preço dos serviços oferecidos pela Barbearia';
```

Fonte: Os autores

2.5.6 Tabela Produto

```
create table produto(
  codprod int generated by default as identity primary key,
  nomeprod varchar(100) not null,
  descprod varchar(255),
  qtdprod int not null,
  precoprod numeric(10, 2) not null,
  fornecedorcodforn int references fornecedor(codforn),
  servicocodserv int references servico(codserv)
);
comment on table produto is 'Tabela de produtos';
comment on column produto.codprod is 'Código do produto';
comment on column produto.nomeprod is 'Nome do produto';
comment on column produto.descprod is 'Descrição do produto';
comment on column produto.qtdprod is 'Quantidade do produto em estoque';
comment on column produto.precoprod is 'Preço do produto';
comment on column produto.fornecedorcodforn is 'Código do fornecedor do produto';
comment on column produto.servicocodserv is 'Código do serviço relacionado ao produto';
--------------------------------------------------------------------------------
```

Fonte: Os autores

2.5.7 Tabela Agendamento

```
create table agendamento(
  codagen int generated by default as identity primary key,
  data_hora timestamp not null,
  status varchar(50) not null,
  clientecodcli int references cliente(codcli),
  funcionariocodfun int references funcionario(codfun)
);
comment on table agendamento is 'Tabela de agendamentos';
comment on column agendamento.codagen is 'Código do agendamento';
```

```
comment on column agendamento.data_hora is 'Data e hora do agendamento';
comment on column agendamento.status is 'Status do agendamento';
comment on column agendamento.clientecodcli is 'Código do cliente';
comment on column agendamento.funcionariocodfun is 'Código do funcionário';
```

Fonte: Os autores

2.5.8 Tabela Pagamento

```
create table pagamento(
  codpag int generated by default as identity primary key,
  valor numeric(10, 2) not null,
  data_hora_pagamento timestamp not null,
  forma_pagamento varchar(50) not null,
  clientecodcli int references cliente(codcli),
  agendamentocodagen int references agendamento(codagen)
);
comment on table pagamento is 'Tabela de pagamentos';
comment on column pagamento.codpag is 'Código do pagamento';
comment on column pagamento.valor is 'Valor do pagamento';
comment on column pagamento.data_hora_pagamento is 'Data e hora do pagamento';
comment on column pagamento.forma_pagamento is 'Forma de pagamento';
comment on column pagamento.clientecodcli is 'Código do cliente';
comment on column pagamento.agendamentocodagen is 'Código do agendamento';
```

Fonte: Os autores

2.5.9 Tabela Histórico serviço

```
create table historico_servico(
  codhistorico int generated by default as identity primary key,
  data_hora timestamp not null,
  servicocodserv int references servico(codserv),
  clientecodcli int references cliente(codcli),
  funcionariocodfun int references funcionario(codfun)
);
comment on table historico_servico is 'Tabela de histórico de serviços prestados';
comment on column historico_servico.codhistorico is 'Código do histórico de serviço';
comment on column historico_servico.data_hora is 'Data e hora do serviço';
comment on column historico_servico.servicocodserv is 'Código do serviço';
comment on column historico_servico.clientecodcli is 'Código do cliente';
comment on column historico_servico.funcionariocodfun is 'Código do funcionário';
```

Fonte: Os autores

Destaques técnicos:

- Triggers: Validação e auditoria de operações.
- Views e Consultas Complexas: Relatórios detalhados e otimizações.
- Políticas de Backup: Segurança e recuperação de dados.

🎲SQL para Relatórios (com Views e Joins)

```
-- View para visualizar agendamentos com informações do cliente e funcionário
create view view_agendamentos_completos as
select ag.codagen, ag.data_hora, ag.status, cl.nomcli, cl.telcli, fn.nomefun
from agendamento ag
join cliente cl on ag.clientecodcli = cl.codcli
join funcionario fn on ag.funcionariocodfun = fn.codfun;
-- View para visualizar pagamentos com detalhes do cliente e agendamento
create view view_pagamentos as
select pg.codpag, pg.valor, pg.data_hora_pagamento, pg.forma_pagamento, cl.nomcli, ag.data_hora
from pagamento pg
join cliente cl on pg.clientecodcli = cl.codcli
join agendamento ag on pg.agendamentocodagen = ag.codagen;
```

🎲Scripts de Criação do Banco de Dados e Índices

```
-- Índices para otimização
create index idx_cliente_nome on cliente(nomcli);
create index idx_fornecedor_nome on fornecedor(nomeforn);
create index idx_funcionario_nome on funcionario(nomefun);
create index idx_servico_nome on servico(nomeserv);
```

🎲Políticas de Acesso

```
-- Criação de usuários e grupos
create role gerente with login password 'senha_gerente';
create role atendente with login password 'senha_atendente';
-- Privilégios para gerente
grant all privileges on all tables in schema public to gerente;
grant select, insert, update, delete on all tables in schema public to atendente;
```

🎲Gatilhos (Triggers)

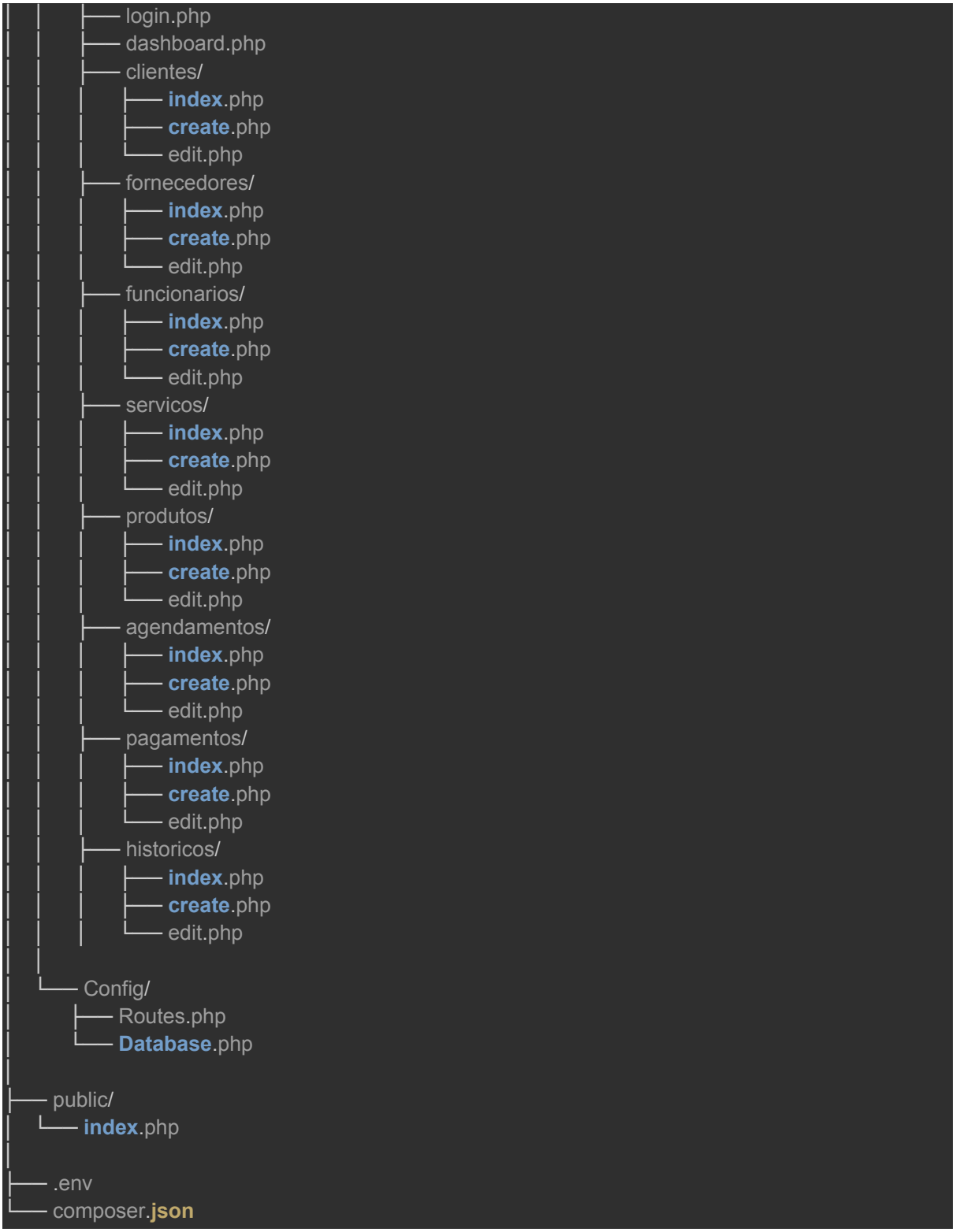
```
-- Trigger para garantir que a quantidade de produtos nunca seja negativa
create or replace function verificar_quantidade_produto()
returns trigger as $$
begin
  if new.qtdprod < 0 then
     raise exception 'A quantidade de produto não pode ser negativa.';
  end if;
  return new;
end;
$$ language plpgsql;
create trigger trg_verificar_quantidade_produto
before insert or update on produto
for each row
execute function verificar_quantidade_produto();
```

```
-- Trigger de auditoria de agendamentos
create or replace function auditoria_agendamento()
returns trigger as $$
begin
  insert into historico_servico (data_hora, servicocodserv, clientecodcli, funcionariocodfun)
  values (now(), new.servicocodserv, new.clientecodcli, new.funcionariocodfun);
  return new;
end;
$$ language plpgsql;
create trigger trg_auditoria_agendamento
after insert on agendamento
for each row
execute function auditoria_agendamento();
## 🎲Políticas e Configuração de Backup e Restore
--Backup
pg_dump -U nome_usuario -F c -b -v -f /caminho/do/backup/devbarbershop.bak devbarbershop
--Restaurar Backup
pg_restore -U nome_usuario -d devbarbershop -v /caminho/do/backup/devbarbershop.bak
```

Fonte: Os autores

4 Estrutura do projeto

```
devbarbershop/
│
├── app/
│   ├── Controllers/
│   │   ├── Login.php
│   │   ├── Dashboard.php
│   │   ├── Clientes.php
│   │   ├── Fornecedores.php
│   │   ├── Funcionarios.php
│   │   ├── Servicos.php
│   │   ├── Produtos.php
│   │   ├── Agendamentos.php
│   │   ├── Pagamentos.php
│   │   └── Historicos.php
│   │
│   ├── Models/
│   │   ├── UsuarioModel.php
│   │   ├── ClienteModel.php
│   │   ├── FornecedorModel.php
│   │   ├── FuncionarioModel.php
│   │   ├── ServicoModel.php
│   │   ├── ProdutoModel.php
│   │   ├── AgendamentoModel.php
│   │   ├── PagamentoModel.php
│   │   └── HistoricoServicoModel.php
│   │
│   ├── Views/
│   │   ├── layouts/
│   │   │   └── main.php
```



Fonte: Os autores

5 Implementação do Sistema DevBarberShop com exemplo Cliente

A implementação do sistema DevBarberShop foi baseada no padrão arquitetural Model-View-Controller (MVC), consagrado no desenvolvimento web moderno por garantir separação de responsabilidades, maior facilidade de manutenção e estabilidade do código.

Neste artigo iremos demonstrar como ficou a estrutura do MVC, apenas do cliente, mas o código e documentação completa poderá ser visto em em nosso repositório do Github disponível em https://github.com/JohnnyMatheus/Sistema-Gerenciamento-para-Barbearia.

No padrão MVC:

- **Model**: concentra a regra de acesso aos dados, definindo como as informações são persistidas e recuperadas do banco de dados.

- **View:** é responsável pela apresentação dos dados ao usuário, por meio de interfaces web responsivas utilizando HTML5, CSS3 e Bootstrap5.

- **Controller**: coordena as requisições do usuário, chamando o Model para buscar ou gravar informações e depois direcionando os resultados para a View.

A seguir, detalha-se um exemplo do módulo Cliente, ilustrando a aplicação prática do padrão MVC.

5.1 Model - ClienteModel.php

O arquivo ClienteModel.php localiza-se na pasta app/Models e representa a abstração da entidade Cliente dentro do sistema. Ele define atributos, validações e a tabela correspondente no banco de dados (PostgreSQL).

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class ClienteModel extends Model

{

    protected $table = 'cliente';

    protected $primaryKey = 'codcli';
```

```
    protected $allowedFields = [

        'nomcli',

        'telcli',

        'emailcli',

        'bairrocli',

        'ruaendcli',

        'sexocli',

        'cidadecli',

        'dtnascli'

    ];

    protected $returnType = 'array';

}
```

Fonte: Os autores

Função do Model: garantir que apenas os campos permitidos sejam gravados, além de centralizar operações de leitura, inserção, atualização e remoção no banco de dados.

5.2 Controller - Clientes.php

O arquivo Clientes.php, localizado em app/Controllers, coordena as ações do usuário relacionadas aos clientes. Ele interage com o ClienteModel e envia dados para as Views.

Exemplo de função dentro do Controller:

```
<?php

namespace App\Controllers;

use App\Models\ClienteModel;

class Clientes extends BaseController
```

```
{
    protected $clienteModel;
    public function __construct()
    {
        $this->clienteModel = new ClienteModel();
    }
    public function index()
    {
        if (!session()->get('usuario')) {
            return redirect()->to('/login');
        }
        $data['clientes'] = $this->clienteModel->findAll();
        return view('clientes/index', $data);
    }
    public function create()
    {
        if (!session()->get('usuario')) {
            return redirect()->to('/login');
        }
        return view('clientes/create');
    }
    public function store()
    {
        $this->clienteModel->save($this->request->getPost());
        return redirect()->to('/clientes');
```

```
    }

    public function edit($id)

    {

        if (!session()->get('usuario')) {

            return redirect()->to('/login');

        }

        $data['cliente'] = $this->clienteModel->find($id);

        return view('clientes/edit', $data);

    }

    public function update($id)

    {

        $dados = $this->request->getPost();

        $this->clienteModel->update($id, $dados);

        return redirect()->to('/clientes');

    }

    public function delete($id)

    {

        $this->clienteModel->delete($id);

        return redirect()->to('/clientes');

    }

}
```

Fonte: Os autores

No padrão arquitetural MVC (Model-View-Controller), o controller atua como um intermediário entre as views (interface de usuário) e os models (lógica de acesso aos dados).

5.3 View - clientes/index.php

As Views estão localizadas em app/Views/clientes. O arquivo index.php exibe uma tabela listando os clientes cadastrados, com opções para editar ou excluir. Exemplo de estrutura em Bootstrap:

5.4 Módulo Fornecedor

Model: FornecedorModel.php
Representa os fornecedores da barbearia, controlando campos como nome, telefone, email e endereço.

Controller: Fornecedores.php
Coordena operações CRUD de fornecedores e redirecionamentos de acordo com a ação solicitada.

View: fornecedores/index.php
Apresenta a tabela com os fornecedores cadastrados e os botões de edição/exclusão.

5.5 Módulo Funcionário

Model: FuncionarioModel.php
Gerencia informações de funcionários, incluindo nome, cargo, salário e dados de contato.

Controller: Funcionarios.php
Interage com o Model para buscar, salvar ou atualizar dados dos funcionários.

View: funcionarios/index.php
Exibe a relação de funcionários cadastrados, com opções de manutenção de registros.

5.6 Módulo Serviços

Model: ServicoModel.php
Registra os serviços oferecidos pela barbearia, incluindo descrição e preço.

Controller: Servicos.php
Controla a lógica de cadastro, edição e exclusão de serviços.

View: servicos/index.php
Permite visualizar os serviços já cadastrados e acessar opções para manutenção.

5.7 Módulo Produtos

Model: ProdutoModel.php
Gerencia os produtos vendidos ou utilizados na barbearia, incluindo fornecedor relacionado.

Controller: Produtos.php
Orquestra todas as operações de CRUD para produtos, garantindo consistência de dados.

View: produtos/index.php
Apresenta a listagem de produtos com dados relevantes de estoque e preço.

5.8 Módulo Agendamentos

Model: AgendamentoModel.php
Controla o agendamento de horários de serviços, relacionando cliente, funcionário e status.

Controller: Agendamentos.php
Gerencia requisições de marcação, alteração ou exclusão de agendamentos.

View: agendamentos/index.php
Lista os agendamentos ativos, finalizados ou cancelados, possibilitando a gestão completa da agenda.

5.9 Módulo Pagamentos

Model: PagamentoModel.php
Registra os pagamentos realizados pelos clientes, com valores, data e forma de pagamento.

Controller: Pagamentos.php
Garante a consistência dos registros de pagamento e gerencia suas alterações.

View: pagamentos/index.php
Oferece a visualização dos pagamentos já efetuados, com detalhes do cliente e do agendamento vinculado.

5.10 Módulo Histórico de Serviços

Model: HistoricoServicoModel.php
Armazena o histórico de serviços realizados, permitindo consultas futuras e relatórios de desempenho.

Controller: Historicos.php
Coleta informações do histórico e apresenta ao usuário dados relevantes de atendimentos passados.

View: historicos/index.php
Apresenta registros históricos detalhados, filtráveis por cliente, funcionário ou período.

6 Rotas no DevBarberShop

O arquivo de rotas do sistema DevBarberShop, localizado em `app/Config/Routes.php`, centraliza toda a configuração de caminhos de acesso da aplicação. Ele utiliza a funcionalidade do roteamento do CodeIgniter 4 para mapear URLs para métodos específicos dos controladores, seguindo o padrão RESTful e boas práticas de arquitetura limpa.

As rotas implementadas contemplam os módulos principais do sistema, garantindo acesso organizado, seguro e padronizado. Cada recurso possui rotas para:

listagem (index)
criação (create/store)
edição (edit/update)
exclusão (delete)

Além disso, o projeto faz uso de parâmetros dinâmicos (por exemplo `(:num)`) para capturar o identificador de registros de forma flexível, como no caso de edições ou exclusões.

Exemplos de rotas do DevBarberShop

- Clientes
    - `/clientes` → lista clientes
    - `/clientes/create` → exibe o formulário de cadastro
    - `/clientes/store` → salva novo cliente
    - `/clientes/edit/(:num)` → edita cliente
    - `/clientes/update/(:num)` → atualiza cliente
    - `/clientes/delete/(:num)` → exclui cliente

- Fornecedores
    - rotas análogas às de clientes, permitindo o mesmo padrão CRUD

- Funcionários, Serviços, Produtos, Agendamentos, Pagamentos e Histórico de Serviços

    - todos seguem a mesma organização, facilitando a leitura, manutenção e escalabilidade do projeto

Além disso, há rotas específicas de autenticação e segurança:

- `/login` → exibe o formulário de login
- `/login/autenticar` → processa as credenciais
- `/logout` → encerra a sessão do usuário
- `/dashboard` → área inicial do sistema, protegida por sessão

Importância no projeto

O uso do roteamento no CodeIgniter4 trouxe vantagens importantes:

- Centralização → todas as regras de acesso estão concentradas em um único arquivo
- Clareza → fácil identificar quais URLs estão disponíveis no sistema
- Segurança → combinado ao middleware de sessão, garante que apenas usuários autenticados acessem os módulos
- Padrão RESTful → melhora a manutenção e deixa o projeto pronto para futura evolução, inclusive para expor APIs

O mapeamento claro das rotas facilita também a integração com ferramentas de testes de funcionalidade (Katalon) e de carga (JMeter), pois os caminhos e seus parâmetros são previsíveis e documentados, reduzindo erros na automação dos testes.

A seguir será demonstrada as rotas do projeto:

```php
<?php
use CodeIgniter\Router\RouteCollection;
/**
* @var RouteCollection $routes
*/
$routes->get('/', 'Home::index');
$routes->get('/login', 'Login::index');
$routes->post('/login/autenticar', 'Login::autenticar');
$routes->get('/logout', 'Login::logout');
$routes->get('/dashboard', 'Dashboard::index');
//CLIENTE
$routes->get('/clientes', 'Clientes::index');
$routes->get('/clientes/create', 'Clientes::create');
$routes->post('/clientes/store', 'Clientes::store');
$routes->get('/clientes/edit/(:num)', 'Clientes::edit/$1');
$routes->post('/clientes/update/(:num)', 'Clientes::update/$1');
$routes->get('/clientes/delete/(:num)', 'Clientes::delete/$1');
//FORNECEDOR
$routes->get('/fornecedores', 'Fornecedores::index');
$routes->get('/fornecedores/create', 'Fornecedores::create');
$routes->post('/fornecedores/store', 'Fornecedores::store');
```

```
$routes->get('/fornecedores/edit/(:num)', 'Fornecedores::edit/$1');
$routes->post('/fornecedores/update/(:num)', 'Fornecedores::update/$1');
$routes->get('/fornecedores/delete/(:num)', 'Fornecedores::delete/$1');
//FUNCIONARIOS
$routes->get('/funcionarios', 'Funcionarios::index');
$routes->get('/funcionarios/create', 'Funcionarios::create');
$routes->post('/funcionarios/store', 'Funcionarios::store');
$routes->get('/funcionarios/edit/(:num)', 'Funcionarios::edit/$1');
$routes->post('/funcionarios/update/(:num)', 'Funcionarios::update/$1');
$routes->get('/funcionarios/delete/(:num)', 'Funcionarios::delete/$1');
//SERVIÇOS
$routes->get('/servicos', 'Servicos::index');
$routes->get('/servicos/create', 'Servicos::create');
$routes->post('/servicos/store', 'Servicos::store');
$routes->get('/servicos/edit/(:num)', 'Servicos::edit/$1');
$routes->post('/servicos/update/(:num)', 'Servicos::update/$1');
$routes->get('/servicos/delete/(:num)', 'Servicos::delete/$1');
//PRODUTOS
$routes->get('/produtos', 'Produtos::index');
$routes->get('/produtos/create', 'Produtos::create');
$routes->post('/produtos/store', 'Produtos::store');
$routes->get('/produtos/edit/(:num)', 'Produtos::edit/$1');
$routes->post('/produtos/update/(:num)', 'Produtos::update/$1');
$routes->get('/produtos/delete/(:num)', 'Produtos::delete/$1');
//Agendamentos
$routes->get('/agendamentos', 'Agendamentos::index');
$routes->get('/agendamentos/create', 'Agendamentos::create');
$routes->post('/agendamentos/store', 'Agendamentos::store');
$routes->get('/agendamentos/edit/(:num)', 'Agendamentos::edit/$1');
$routes->post('/agendamentos/update/(:num)', 'Agendamentos::update/$1');
$routes->get('/agendamentos/delete/(:num)', 'Agendamentos::delete/$1');
//Pagamentos
$routes->get('/pagamentos', 'Pagamentos::index');
$routes->get('/pagamentos/create', 'Pagamentos::create');
$routes->post('/pagamentos/store', 'Pagamentos::store');
$routes->get('/pagamentos/edit/(:num)', 'Pagamentos::edit/$1');
$routes->post('/pagamentos/update/(:num)', 'Pagamentos::update/$1');
$routes->get('/pagamentos/delete/(:num)', 'Pagamentos::delete/$1');
//Histórico de serviços
$routes->get('/historicos', 'Historicos::index');
$routes->get('/historicos/create', 'Historicos::create');
$routes->post('/historicos/store', 'Historicos::store');
$routes->get('/historicos/edit/(:num)', 'Historicos::edit/$1');
$routes->post('/historicos/update/(:num)', 'Historicos::update/$1');
$routes->get('/historicos/delete/(:num)', 'Historicos::delete/$1');
```

Fonte: Os autores

7 Padrões de Projeto

O projeto utiliza MVC como arquitetura principal, aplica Singleton para conexão de banco, e injeção de dependências para facilitar a manutenção. Esses padrões são aderentes ao contexto do CodeIgniter e ao objetivo acadêmico, garantindo qualidade, testabilidade e clareza no desenvolvimento do sistema.

7.1 Padrão MVC (Model-View-Controller)

Model → regras de acesso ao banco de dados

View → as telas com Bootstrap

Controller → regras de negócio e fluxo de requisições

Vantagem: facilita a manutenção, testes, e colaboração em equipe.

Exemplo no projeto:

O CRUD de clientes, por exemplo, está totalmente separado:

- ClienteModel.php → acesso à tabela - Clientes.php (controller) → orquestra as requisições - views → mostram os dados no Bootstrap

7.2 Singleton (injetado pelo CodeIgniter)

O CodeIgniter usa Singleton internamente para gerenciar a conexão de banco de dados:

Ele abre a conexão apenas uma vez e a reaproveita em todas as requisições

Isso evita múltiplas instâncias da conexão e economiza recursos

Exemplo no projeto:

Quando você chama $this->db ou um Model, a mesma conexão é mantida em singleton no Core do CodeIgniter.

7.3 Dependency Injection (inversão de dependência)

O CodeIgniter facilita injeção de dependências através de construtores (ex.: passando Models nos controllers)

Ajuda a tornar o código mais testável e modular

8 Teste Unitário

Foram criados testes unitários para validar o modelo de autenticação de usuários no sistema. Por exemplo, verificou-se se o administrador padrão está cadastrado corretamente, e se a senha padrão (hash) confere com o esperado, utilizando PHPUnit como framework de testes automatizados.

Diretório de teste:

`tests/unit/LoginTest.php`

8.1 Teste para Login

```php
<?php
namespace App\Tests\Unit;
use CodeIgniter\Test\CIUnitTestCase;
use App\Models\UsuarioModel;
class LoginTest extends CIUnitTestCase
{
  public function testUsuarioExiste()
  {
    $model = new UsuarioModel();
    $usuario = $model->where('email', 'admin@devbarbershop.com')->first();

    $this->assertNotNull($usuario, 'Usuário administrador não existe no banco');
    $this->assertEquals('admin@devbarbershop.com', $usuario['email']);
  }

  public function testVerificaSenha()
  {
    $model = new UsuarioModel();
    $usuario = $model->where('email', 'admin@devbarbershop.com')->first();
    $senhaCorreta = password_verify('admin123', $usuario['senha']);
    $this->assertTrue($senhaCorreta, 'A senha do usuário administrador está incorreta');
  }
}
```

Fonte: Os autores

Esse teste:

- Confere se existe o usuário “admin” cadastrado
- Verifica se a senha admin123 está funcionando corretamente

Como rodar o teste unitário?

- php vendor/bin/phpunit ou
- php spark test

8.2 Resultados do teste

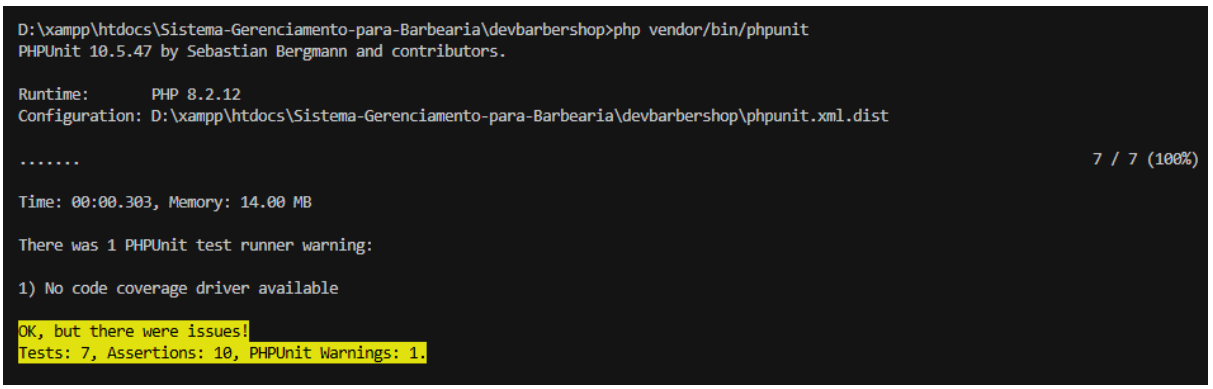


Figura 11: Resultado teste unitário

9 Teste Funcionalidade

Para testes de funcionalidade foi utilizada a ferramenta Katalon Studio, por sua interface intuitiva e recursos de gravação de cenários. Foram validados fluxos completos do sistema DevBarberShop, incluindo login, cadastros, edições, exclusões e navegação entre módulos, assegurando que todas as funcionalidades estejam alinhadas aos requisitos especificados.

- Simular o comportamento real do usuário (navegação, cliques, campos)
- Automatizar login, cadastro, edição e exclusão
- Validar regras de negócio

Os testes poderam ser observados em nosso repositório no Github disponivel em : https://github.com/JohnnyMatheus/Sistema-Gerenciamento-para-Barbearia

10 Teste de Carga com JMeter

O Apache JMeter é uma ferramenta gratuita e de código aberto utilizada para testar o desempenho e a carga de aplicações web. Com ele, é possível simular múltiplos usuários acessando uma aplicação ao mesmo tempo, medindo tempo de resposta, falhas e comportamento do sistema sob estresse.

A seguir serão apresentadas as imagens dos testes realizados no JMeter:

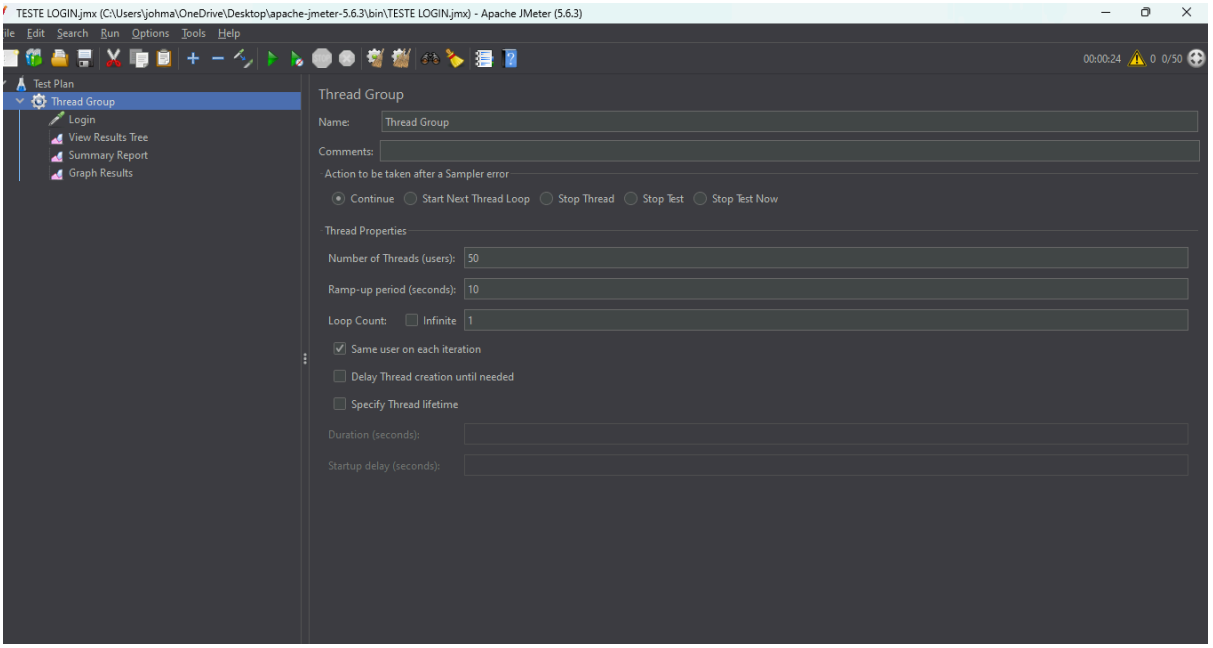


Figura 12: Thread Group

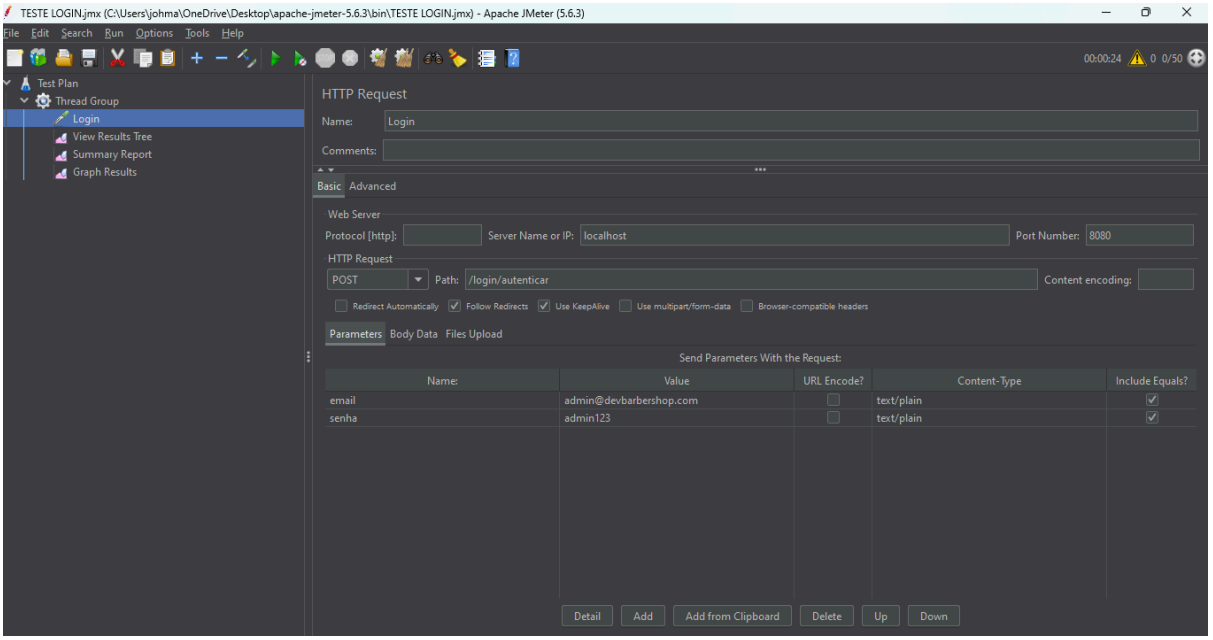


Figura 13: HTTP Request -> Login

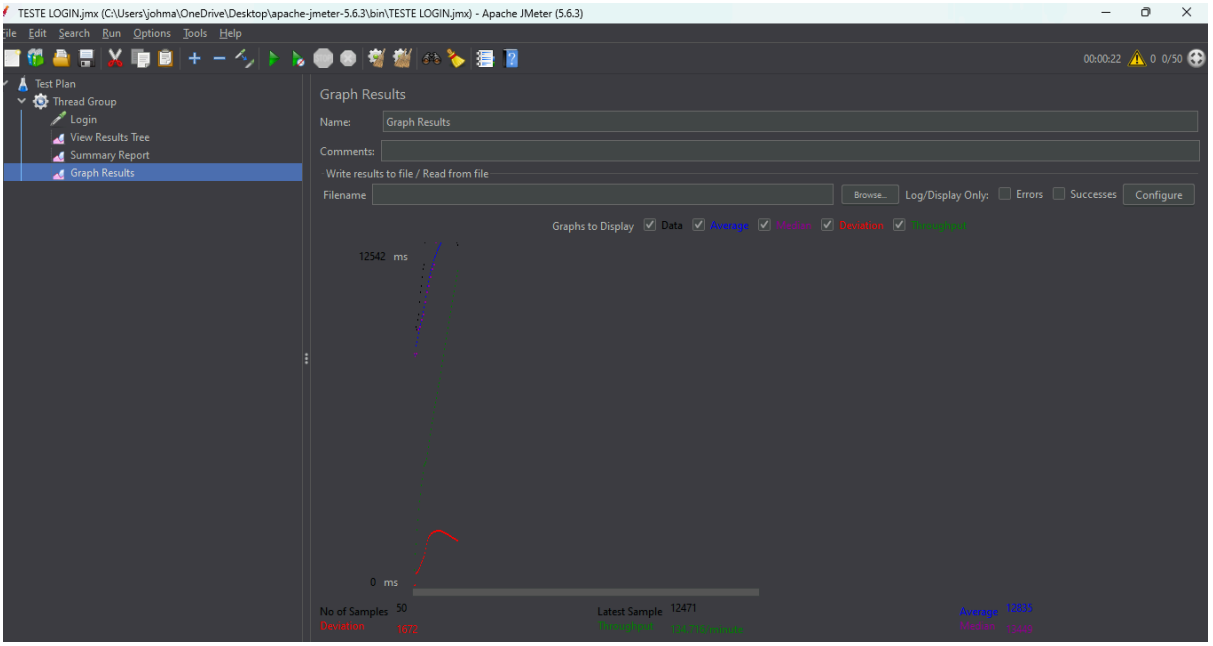


Figura 13: Graph Results

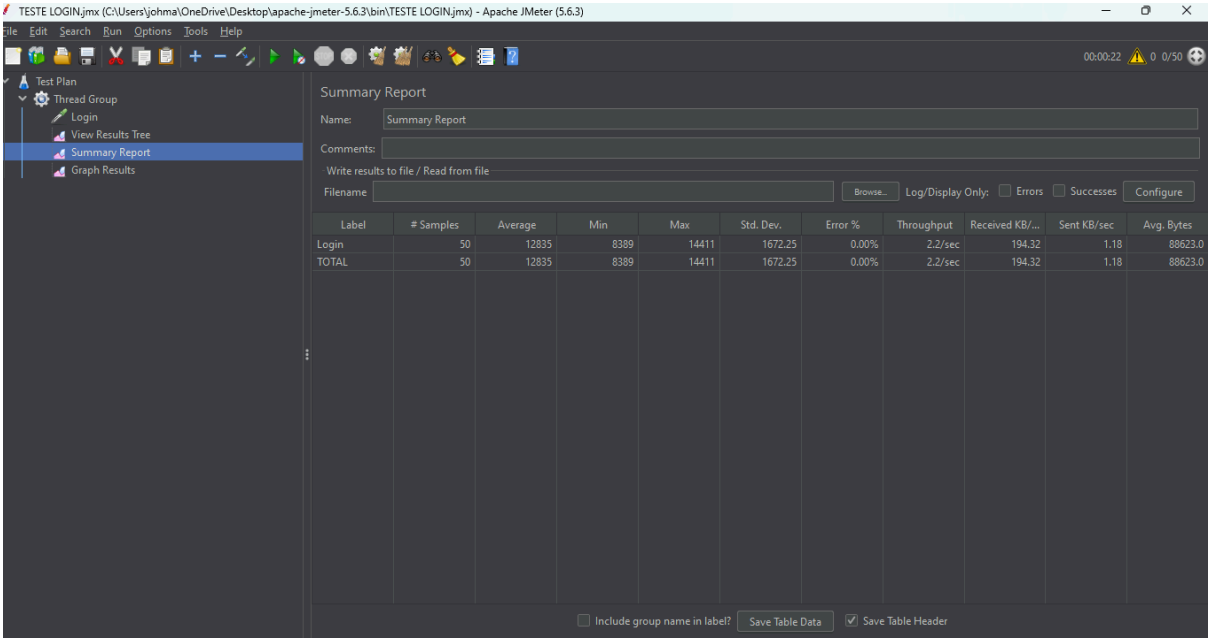


Figura 13: Summary Report

10.1 Resultados Teste de carga Jmeter

Foi realizado um teste de carga com o Apache JMeter, simulando 50 usuários simultâneos acessando o sistema de login. O tempo médio de resposta foi de aproximadamente 22 segundos, sem ocorrência de erros ou falhas de autenticação, demonstrando que o sistema suporta uma carga inicial de acessos dentro do esperado.

Conclusão

O desenvolvimento do sistema DevBarberShop permitiu aplicar na prática os conceitos de engenharia de software, arquitetura MVC, orientação a objetos e padrões de projeto discutidos ao longo do curso. A solução atende plenamente às necessidades de gestão de barbearias, automatizando processos essenciais como agendamentos, controle de estoque, cadastro de clientes, registro de pagamentos e histórico de serviços. Além disso, o uso de testes unitários (PHPUnit), testes de funcionalidade (Katalon Studio) e testes de carga (JMeter) garantiu a confiabilidade e a robustez do sistema.

A adoção do framework CodeIgniter 4 foi justificada pela leveza, simplicidade e facilidade de aprendizado, o que permitiu rápida evolução do projeto sem sacrificar a qualidade. Tecnologias complementares, como PostgreSQL para o banco de dados e Bootstrap para a interface, também foram fundamentais para atender aos requisitos de escalabilidade, segurança e usabilidade.

Como resultado, o DevBarberShop se mostrou uma solução eficiente, modular e preparada para evolução futura, promovendo o ganho de produtividade e a modernização do ambiente gerencial de barbearias. O trabalho colaborativo da equipe e a utilização de ferramentas de versionamento (GitHub) viabilizaram o acompanhamento das etapas de desenvolvimento de forma organizada e segura. Assim, o projeto representa não apenas um exercício acadêmico, mas também um protótipo viável para aplicação no mundo real, consolidando o aprendizado e estimulando o desenvolvimento contínuo de habilidades na área de engenharia de software

Barbershop Management System: A DevBarberShop Case Study

Abstract

This article describes the development of a web-based system for managing a barber shop, including control of customers, appointments, payments, products, services and employees. The main objective was to provide a centralized and easy-to-use solution, using the CodeIgniter 4 framework and PostgreSQL database. MVC architecture principles were applied, with a responsive interface built in Bootstrap 5. Unit, functional and load tests were performed to validate the robustness and performance of the application.

Keywords: Barber shop management; CodeIgniter 4; PostgreSQL; Bootstrap 5; MVC; software testing.

Referências

WORLD WIDE WEB CONSORTIUM (W3C). HTML5. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em: 29 jun. 2025.

WORLD WIDE WEB CONSORTIUM (W3C). CSS3. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS. Acesso em: 29 jun. 2025.

GETBOOTSTRAP. Bootstrap 5. Disponível em: https://getbootstrap.com/docs/5.0/getting-started/introduction/. Acesso em: 29 jun. 2025.

THE PHP GROUP. PHP: Hypertext Preprocessor. Disponível em: https://www.php.net/. Acesso em: 29 jun. 2025.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. PostgreSQL. Disponível em: https://www.postgresql.org/. Acesso em: 29 jun. 2025.

CODEIGNITER FOUNDATION. CodeIgniter 4 User Guide. Disponível em: https://codeigniter4.github.io/userguide/. Acesso em: 29 jun. 2025.

ECLIPSE FOUNDATION. DBeaver Community. Disponível em: https://dbeaver.io/. Acesso em: 29 jun. 2025.

MICROSOFT CORPORATION. Visual Studio Code. Disponível em: https://code.visualstudio.com/. Acesso em: 29 jun. 2025.

APACHE SOFTWARE FOUNDATION. Apache JMeter. Disponível em: https://jmeter.apache.org/. Acesso em: 29 jun. 2025.

KATALON LLC. Katalon Studio. Disponível em: https://katalon.com/. Acesso em: 29 jun. 2025.

APACHE SOFTWARE FOUNDATION. XAMPP. Disponível em: https://www.apachefriends.org/pt_br/index.html. Acesso em: 29 jun. 2025.

GITHUB INC. GitHub Desktop. Disponível em: https://desktop.github.com/. Acesso em: 29 jun. 2025.