UNIVERSIDADE POSITIVO

GABRIEL SANTOS CAMARGO

HENRIQUE MACHADO ALGAUER

**SISTEMA INTERNO DE LOJA: PRODUTOS, VENDAS E FINANCEIRO**

CURITIBA

2024

**GABRIEL SANTOS CAMARGO**

**HENRIQUE MACHADO ALGAUER**

**SISTEMA INTERNO DE LOJA: PRODUTOS, VENDAS E FINANCEIRO**

Trabalho de conclusão de curso de Graduação apresentada como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Universidade Positivo.

Orientador(a): Fábio Inocencio Kravertz.

**CURITIBA**

**2024**

SUMÁRIO

[1 INTRODUÇÃO 4](#_Toc168404936)

[1.1 Estrutura do trabalho 4](#_Toc168404937)

[1.2 Objetivos 4](#_Toc168404938)

[2 revisão bibliográfica 5](#_Toc168404939)

[2.1 Arquitetura MVC 5](#_Toc168404940)

[3 Desenvolvimento 6](#_Toc168404941)

[3.1 Modelagem do Banco De Dados 6](#_Toc168404942)

[3.2 Funcionamento do Sistema 9](#_Toc168404943)

[3.3 Módulos do sistema 29](#_Toc168404944)

[3.3.1 Módulo Produto 30](#_Toc168404945)

[3.3.2 Módulo Venda 30](#_Toc168404946)

[3.3.3 Módulo Financeiro 30](#_Toc168404947)

[4 Conclusão 31](#_Toc168404948)

[REFERÊNCIAS 33](#_Toc168404949)

1. INTRODUÇÃO

Este Trabalho tem como objetivo o desenvolvimento de um Sistema interno de Loja abrangendo 3 funcionalidades (Produtos, Vendas e Financeiro) sendo cada uma gerando um CRUD que segue Regras de Negócio, foi utilizado como principal linguagem de programação o *PHP*, mas também foi utilizado *Javascript*, para estilização e marcação foi utilizado *BootStrap* e *HTML5*, no que tange a parte de banco de dados foi utilizado o *MySQL*.

Como Padrão de arquitetura de software foi utilizado o MVC (*Model*, *View* e *Controller*) para organização de pastas e arquivos do projeto, além de facilitar a divisão das áreas de acesso do Usuário.

## Estrutura do trabalho

Este trabalho está organizando em 3 principais partes, na introdução, será tratado brevemente o que é o sistema e as tecnologias utilizadas, seguido dos objetivos deste trabalho acadêmico, em segunda instancia será feito uma breve revisão bibliográfica quanto a arquitetura de software MVC, seguido do desenvolvimento do trabalho que irá explicar o funcionamento do sistema e também os módulos que o compõem e por final a conclusão do trabalho acadêmico

## Objetivos

Este trabalho tem como objetivo geral o desenvolvimento de um sistema de interno para uma loja, utilizando o PHP e a arquitetura MVC, passando por pontos importantes como:

* Análise de requisitos do sistema
* Modelagem do banco de dados adequado
* Apresentação de 3 páginas públicas antes do login no sistema
* Implementar CRUD e funcionalidades nas páginas de Produtos, Vendas e Financeiro
* Testes e validações do sistema

1. revisão bibliográfica

Esta seção é dedicada para embasamento de conceitos com o intuito de uma melhor síntese do conteúdo apresentado neste trabalho.

## Arquitetura MVC

A Arquitetura MVC é um padrão de Arquitetura para o desenvolvimento de software, proporcionando uma estrutura organizada que possibilita a manutenibilidade, a reutilização de código e a testabilidade. Gerando Códigos mais sólidos e com alta capacidade de escalabilidade no decorrer do tempo da vida útil do software.

Dentro deste Trabalho o MVC é tratado com duas principais pastas, sendo elas *Public* e *App,* contendo na pasta *Public* arquivos de acesso público que são de livre acesso do sistema, diferente da pasta *App* que possuí 4 outras pastas que ajudam a delimitar os acessos sendo elas *Model, View, Controller* e *Core* as quais possuem funcionalidades específicas e delimitadas como:

* *Models*: Conter as interações com o banco de dados e delimitar as tabelas e colunas do banco de dados que serão usadas na classe que terá o nome da tabela referenciada, podendo conter algumas funções específicas das tabelas.
* *Views*: Contém as páginas de interação e visualização do usuário, que são carregadas mediante seus devidos *controllers*.
* *Controllers*: Contém os arquivos que irão carregar as páginas de visualizações devidas ao usuário, além de verificações de acesso as páginas e delimitação de regras de negócios e validações de entradas do usuário.
* *Core*: Dentro da Pasta Core temos os arquivos do “Núcleo do sistema”, utilizada para agrupar componentes e funcionalidades essenciais e reutilizáveis, contendo também configurações de acesso ao banco de dados.

1. Desenvolvimento

Nesta parte irá ser abordado a relação de pontos chave no desenvolvimento do sistema interno de loja, bem como os seus 3 módulos, juntamente com explicações de seu funcionamento como um todo.

## Modelagem do Banco De Dados

O modelo de Banco de Dados relacionais foi utilizado o MySQL, foi criado 5 principais tabelas que armazenam registros. Foi utilizado a criação de um Trigger para atualizar o estoque e registrar a venda no financeiro após uma venda, deste modo facilitando o controle financeiro e o estoque dos produtos.

Segue estruturação das tabelas utilizadas:

* Financeiro: O intuito desta tabela é armazenar registros de movimentações financeiras de vendas de produtos, recebendo como colunas: ID, Tipo, Nome, Valor, Data. Tendo o campo ‘Tipo’ para armazenar a informação se é um débito ou crédito para a loja e o campo ‘nome’ para armazenar a referência da venda ou o nome do lançamento referente a este financeiro.
* Produtos: Esta tabela armazena os registros dos produtos cadastrados no sistema, possuindo as seguintes colunas: ID, nome, Preco e Estoque. Sendo colunas que carregam informações para controle de valor e estoque dos produtos cadastrados no sistema.
* Usuário: Nesta tabela são armazenadas as informações de login e senha do usuário cadastrado no sistema.
* Vendas: Esta tabela seria o local para armazenamento dos registros das vendas no sistema, recebendo a forma de pagamento, valor total e a data da venda, gerando um ID que será usado como *Foreign key* em outra tabela.
* VendasItens: Esta tabela é destinada para armazenamento da venda de cada Item pertencente em uma venda, deste modo facilitando o controle financeiro e o controle de estoque em cada produto, recebendo como *Foreign key* o id da venda referente ao item vendido.

Deste Modo foi estruturado o Banco de dados para utilização no sistema, abaixo será deixado o código de geração do banco de dados na linguagem SQL.

Código 1 – Query Geradora das tabelas do banco de dados.

CREATE DATABASE IF NOT EXISTS loja;

-- Usar banco de dados

USE loja;

-- Criar tabela de produtos

CREATE TABLE IF NOT EXISTS produtos (

id INT AUTO\_INCREMENT PRIMARY KEY,

nome VARCHAR(100) NOT NULL,

preco DECIMAL(10, 2) NOT NULL,

estoque INT NOT NULL DEFAULT 0

);

-- Criar tabela de vendas

CREATE TABLE IF NOT EXISTS vendas (

id INT AUTO\_INCREMENT PRIMARY KEY,

formaPagamento VARCHAR(50) NOT NULL,

valorTotal DECIMAL(10, 2) NOT NULL,

dataVenda TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Criar tabela de vendasItens

CREATE TABLE IF NOT EXISTS vendasItens (

id INT AUTO\_INCREMENT PRIMARY KEY,

codVenda INT NOT NULL,

codProduto INT NOT NULL,

quantidade INT NOT NULL,

valorItem DECIMAL(10, 2) NOT NULL,

valorTotal DECIMAL(10, 2) AS (quantidade \* valorItem) STORED,

FOREIGN KEY (codVenda) REFERENCES vendas(id),

FOREIGN KEY (codProduto) REFERENCES produtos(id)

);

-- Criar tabela de financeiro

CREATE TABLE IF NOT EXISTS financeiro (

id INT AUTO\_INCREMENT PRIMARY KEY,

tipo VARCHAR(50) NOT NULL,

nome VARCHAR(100) NOT NULL,

valor DECIMAL(10, 2) NOT NULL,

data DATE NOT NULL

);

create table usuario (

login VARCHAR(20) PRIMARY KEY,

senha VARCHAR(255) NOT NULL

);

-- Trigger para atualizar o estoque e registrar a venda no financeiro após uma venda

DELIMITER //

CREATE TRIGGER atualizar\_estoque\_venda

AFTER INSERT ON vendasItens

FOR EACH ROW

BEGIN

-- Atualizar estoque

UPDATE produtos

SET estoque = estoque - NEW.quantidade

WHERE id = NEW.codProduto;

-- Inserir registro no financeiro

INSERT INTO financeiro (tipo, nome, valor, data)

VALUES ('credito', CONCAT('venda id:', NEW.codVenda), NEW.valorTotal, (SELECT dataVenda FROM vendas WHERE id = NEW.codVenda));

END;

//

DELIMITER ;

Fonte: Os Autores.

## Funcionamento do Sistema

Neste Módulo vamos abordar quanto as funcionalidades do sistema e seu funcionamento em geral, como funciona a decisão de qual *controller* será chamada para carregar sua referente *view.*

Neste Sistema por conta da arquitetura MVC, é feito uma diferenciação de nível de acesso de um usuário e apenas um visitante, a principal diferença de um usuário logado e um visitante seria a capacidade de acesso a *dashboard,* enquanto um visitante pode navegar apenas pelas páginas de apresentação do sistema.

Alguma Configurações referentes ao .htaccess e também o robots.txt são setadas como padrão, tendo como suas respectivas funcionalidades:

* .htaccess: arquivo de configuração do Apache é usado para controlar e modificar o comportamento do servidor web, redirecionando todas as requisições que não são arquivos ou diretórios existentes para index.php, passando a URL solicitada como um parâmetro.
* Robots.txt: Informa aos mecanismos de busca para não indexarem os diretórios /admin/ e /ajax/, protegendo áreas administrativas e evitando indexação de conteúdo irrelevante.

Abaixo será indexado os respectivos códigos de funcionamento do .htaccess e também robots.txt

Código 2 - .htaccess

# RewriteEngine On

# #acecept loading of actual files and directions

# RewriteCond %{REQUEST\_FILENAME} !-f

# RewriteCond %{REQUEST\_FILENAME} !-d

# #send everything else to the index page

# RewriteRule ^(.\*)$ index.php?url=$1 [L,QSA]

RewriteEngine On

RewriteBase /testPHP/testMVC-Login/public/

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule ^(.\*)$ index.php?url=$1 [QSA,L]

Fonte: Os autores

Código 3 – Robots.txt

User-agent: \*

Disallow: /admin/

Disallow: /ajax/

Fonte: Os autores

Quanto ao funcionamento do sistema, ao ser a acessado a URL do sistema o usuário é direcionado para o arquivo *index.php* que está localizado na pasta *public.*

Código 3 – index.php

<?php

session\_start();

require "../app/core/init.php";

if(DEBUG){

ini\_set('display\_errors', 1);

}else{

ini\_set('display\_errors', 0);

}

$app = new App;

$app->loadController();

Fonte: Os autores

Dentro deste arquivo é iniciado a sessão do usuário no sistema, juntamente com a requisição do arquivo *init.php* que está no path App/Core, este arquivo init.php irá carregar todos os arquivos do core para que possam estar prontos para uso dentro do projeto, usando uma função de *autoload* e os requires nos arquivos do Core.

Código 4 – Arquivo init.php.

<?php

spl\_autoload\_register(function($classname){

require $filename ="../app/models/".ucfirst($classname).".php";

});

require 'config.php';

require 'Database.php';

require 'App.php';

require 'Model.php';

require 'Controller.php';

require 'functions.php';

Fonte: Os Autores.

Deste Modo as Funcionalidades principais do sistema são chamadas, sendo que cada uma delas carregas funções essenciais para o funcionamento do sistema, sendo elas:

* Config.php : Neste arquivo é onde será definido as configurações base referentes ao acesso do banco de dados local ou uma instância web, no caso deste projeto é utilizado a instância local
  + Código 5 – arquivo config.php que carrega as informações de acesso do banco de dados

<?php

if($\_SERVER['SERVER\_NAME'] == 'localhost'){

// databse config

define('DBNAME', 'loja');

define('DBHOST', 'localhost');

define('DBUSER', 'root');

define('DBPASS', '1234');

define('DBDRIVER', '');

define('DBPORT', '3306');

define('ROOT', 'http://localhost/testphp/testMVC-Login/public');

}else{

define('ROOT', 'https://info.ghdigitalservices.com.br');

define('DBNAME', 'my\_db');

define('DBHOST', 'localhost');

define('DBUSER', 'root');

define('DBPASS', '1234');

define('DBDRIVER', '');

}

define('APP\_NAME', 'Projeto faculdade de PHP');

define('APP\_DESC', 'Esse projeto ');

/\* true means show errors \*/

define('DEBUG', true);

Fonte: Os autores

* Database.php: Neste arquivo terá as funções de conexão com o Banco de dados ( *connect( )* ), seguido das funções de execução de query ( *query($query, $data= [ ])* ) juntamente com a função de pegar o resultado de uma linha apenas (get\_row*($query, $data= [ ])*)
  + Codigo 6 – código do arquivo database.php

<?php

Trait Database{

public function connect(){

$string = "mysql:hostname=".DBHOST.";port=".DBPORT.";dbname=".DBNAME;

$con = new PDO($string,DBUSER,DBPASS);

return $con;

}

public function query($query, $data=[]){

$con = $this->connect();

$stm = $con->prepare($query);

$check = $stm->execute($data);

if($check){

$result = $stm->fetchAll(PDO::FETCH\_OBJ);

if(is\_array($result) && count($result)){

return $result;

}

}

return false;

}

public function get\_row($query, $data=[]){

$con = $this->connect();

$stm = $con->prepare($query);

$check = $stm->execute($data);

if($check){

$result = $stm->fetchAll(PDO::FETCH\_OBJ);

if(is\_array($result) && count($result)){

return $result[0];

}

}

return false;

}

}

Fonte: Os autores

* App.php: Esta é a classe que irá carregar a função de administração dos controllers e suas devidas view que será tratado de maneira mais específica no próximo tópico
* Model.php: Dentro deste arquivo tem diversas funções que são herdadas pelas Models que irão realizar as consultas no banco de dados, fazendo um código mais reutilizável e modularizado
  + Código 7 – Model.php

<?php

Trait Model{

use Database;

protected $limit = 10;

protected $offset = 0;

protected $order\_type= "desc";

protected $order\_column= 'id';

public $errors = [];

public function findAll(){

$query = "SELECT \* FROM $this->table ORDER BY $this->order\_column $this->order\_type LIMIT $this->limit OFFSET $this->offset";

return $this->query($query);

}

public function where($data, $data\_not = []){

$keys = array\_keys($data);

$keys\_not = array\_keys($data\_not);

$query = "SELECT \* FROM $this->table WHERE ";

foreach($keys as $key){

$query .= $key . " = :". $key . " && ";

}

foreach($keys\_not as $key){

$query .= $key . " != :". $key . " && ";

}

$query = trim($query, " && ");

$query .= " ORDER BY $this->order\_column $this->order\_type LIMIT $this->limit OFFSET $this->offset";

$data = array\_merge($data, $data\_not);

return $this->query($query, $data);

}

public function first($data, $data\_not = []){

$keys = array\_keys($data);

$keys\_not = array\_keys($data\_not);

$query = "SELECT \* FROM $this->table WHERE ";

foreach($keys as $key){

$query .= $key . " = :". $key . " && ";

}

foreach($keys\_not as $key){

$query .= $key . " != :". $key . " && ";

}

$query = trim($query, " && ");

$query .= " limit $this->limit offset $this->offset";

$data = array\_merge($data, $data\_not);

$result = $this->query($query, $data);

if($result)

return $result[0];

return false;

}

public function insert($data){

if(!empty($this->allowedColumns)){

foreach($data as $key => $value) {

if(!in\_array($key, $this->allowedColumns)){

unset($data[$key]);

}

}

}

$keys = array\_keys($data);

$query = "INSERT INTO $this->table (".implode(",", $keys).") VALUES (:".implode(",:", $keys).") ";

$this->query($query, $data);

return false;

}

public function update($id, $data, $id\_column = 'id'){

/\*remove unwanted data \*/

if(!empty($this->allowedColumns)){

foreach($data as $key => $value) {

if(!in\_array($key, $this->allowedColumns)){

unset($data[$key]);

}

}

}

$keys = array\_keys($data);

$query = "UPDATE $this->table SET ";

foreach($keys as $key){

$query .= $key . " = :". $key . ", ";

}

$query = trim($query, ", ");

$query .= " WHERE $id\_column = :id";

$data[$id\_column] = $id;

$this->query($query, $data);

return false;

}

public function delete($id, $id\_column = 'id'){

$data[$id\_column] = $id;

$query = "DELETE FROM $this->table WHERE $id\_column = :$id\_column"; // Certifique-se de que o placeholder corresponda ao nome da coluna

return $this->query($query, $data);

}

function test(){

$query = "select \* from users";

$result = $this->query($query);

echo "<pre>";

print\_r($result);

echo "</pre>";

}

public function searchByDescription($description)

{

$query = "SELECT \* FROM $this->table WHERE nome LIKE :description";

$data = [':description' => "%$description%"];

return $this->query($query, $data);

}

}

Fonte: Os autores

* Controller.php: Este arquivo contém o método de carregar a view referente ao controller que será usado em todas as controller
  + Código 8 – controller.php

<?php

Trait Controller{

public function view($name, $data = []){

if(!empty($data)){

extract($data);

}

$filename = "../app/views/".$name.".view.php";

if(file\_exists($filename)){

require $filename;

}else{

$filename = "../app/views/404.view.php";

require $filename;

}

}

}

Fonte: Os autores

* Functions.php: Nesta classe possui endereço de url de varias funções que serão utilizadas em diversas funções, reutilizando códigos que seria repetidos diversas vezes nas views
  + Código 9 – functions.php

<?php

function show($f){

echo "<pre>";

print\_r($f);

echo "</pre>";

}

function esc($str){

return htmlspecialchars($str);

}

function redirect($path){

header("Location: ".ROOT."/".$path);

die;

}

function menu(){

echo "

<header>

<nav>

";

if(empty($\_SESSION['LOGIN'])){

echo "

<ul>

<li><img class='logo' src='http://localhost/testphp/testMVC-Login/public/assets/img/php3d.png' alt='' /></li>

<li><a href='http://localhost/testphp/testMVC-Login/public/'>Home</a></li>

<li><a href='#'>naosei</a></li>

<li><a href='http://localhost/testphp/testMVC-Login/public/sobre'>Sobre</a></li>

</ul>

<div>

<a href='http://localhost/testphp/testMVC-Login/public/login'>

<button type='button' class='nav-button' id='login'>Login</button>

</a>

<a href='http://localhost/testphp/testMVC-Login/public/register'>

<button type='button' class='nav-button' id='register'>Registrar</button>

</a>

";

}else{

$nome = $\_SESSION['LOGIN']->login;

echo "

<ul>

<li><img class='logo' src='http://localhost/testphp/testMVC-Login/public/assets/img/php3d.png' alt='' /></li>

<li><a href='http://localhost/testphp/testMVC-Login/public/dashboard'>Dashboard</a></li>

<li><a href='http://localhost/testphp/testMVC-Login/public/produto'>Produtos</a></li>

<li><a href='http://localhost/testphp/testMVC-Login/public/financeiros'>Financeiro</a></li>

<li><a href='http://localhost/testphp/testMVC-Login/public/venda'>Vendas</a></li>

</ul>

<div class='header-button'>

<h2>Olá, $nome</h2>

<a href='http://localhost/testphp/testMVC-Login/public/logout'>

<button type='button' class='nav-button' id='logout'>Logout</button>

</a>

";

}

echo "

</div>

</nav>

</header>

";

}

function menuExterno(){

echo "

<header>

<nav>

<ul>

<li><img class='logo' src='http://localhost/testphp/testMVC-Login/public/assets/img/php3d.png' alt='' /></li>

<li><a href='http://localhost/testphp/testMVC-Login/public/'>Home</a></li>

<li><a href='#'>naosei</a></li>

<li><a href='http://localhost/testphp/testMVC-Login/public/sobre'>Sobre</a></li>

";

if(empty($\_SESSION['LOGIN'])){

echo "

</ul>

<div>

<a href='http://localhost/testphp/testMVC-Login/public/login'>

<button type='button' class='nav-button' id='login'>Login</button>

</a>

<a href='http://localhost/testphp/testMVC-Login/public/register'>

<button type='button' class='nav-button' id='register'>Registrar</button>

</a>

";

}else{

$nome = $\_SESSION['LOGIN']->login;

echo "

<li><a href='http://localhost/testphp/testMVC-Login/public/dashboard'>Painel</a></li>

</ul>

<div class='header-button'>

<h2>Olá, $nome</h2>

<a href='http://localhost/testphp/testMVC-Login/public/logout'>

<button type='button' class='nav-button' id='logout'>Logout</button>

</a>

";

}

echo "

</div>

</nav>

</header>

";

}

function contentBox($title, $img, $text){

echo "

<div class='content-box'>

<div class='content-image'>

<img src='http://localhost/testphp/testMVC-Login/public/assets/img/$img.png' alt='' />

</div>

<div class='content-box-text blur'>

<h3>$title</h3>

<p>

$text

</p>

</div>

</div>

";

}

function footer(){

echo "

<footer id='footer'>

<p>&copy; PROJETO PHP</p>

</footer>

";

}

Fonte: Os autores

Após isso dentro do arquivo *index.php* é testado se ocorreu algum erro durante o acesso desta pasta, após isso é criado uma variável *$app* que irá fazer referência a Classe App que esta instanciada no arquivo *App.php*, o qual se localiza no path App/Core

Deste modo é chamado a função de *loadController( )* da Classe App que irá determinar a partir da URL o controller correto para essa página, o qual irá carregar a devida *View.*

Abaixo será indexado a classe App que tem as funções de carregamento do *controller* ( *loadController( )* ) e a também a função que captura as informações da URL e retira as barras para poder carregar o controller ( *splitURL( )* )

Código 11 – Classe App e as suas respectivas funções

<?php

class App{

private $controller = 'Home';

private $method = 'index';

private function splitURL(){

$URL = $\_GET['url'] ?? 'home';

$URL = explode("/", trim($URL, "/"));

return $URL;

}

public function loadController(){

$URL = $this->splitURL();

/\* Select controller \*/

$filename = "../app/controllers/".ucfirst($URL[0]).".php";

if(file\_exists($filename)){

require $filename;

$this->controller = ucfirst($URL[0]);

unset($URL[0]);

}else{

$filename = "../app/controllers/\_404.php";

require $filename;

$this->controller = "\_404";

}

$controller = new $this->controller;

/\* Select method \*/

if(!empty($URL[1])){

if(method\_exists($controller, $URL[1])){

$this->method = $URL[1];

unset($URL[1]);

}

}

call\_user\_func\_array([$controller, $this->method], $URL);

}

}

Fonte: Os autores

Com isto a controller *default* (página *Home*), com isso as funções Core do sistema juntamente com o carregamento de um controller e a exibição de uma view base.

## Módulos do sistema

O sistema tratado neste trabalho gira em torno de 3 principais Módulos, sendo eles Produtos, Financeiro e Vendas. Os quais possuem funções de um CRUD básico (Create, Read, Update e Delete) seguindo as regras de negócio e também delimitações que evitam conflito de funções e também repetição das mesmas, ou seja , não possui campos de excluir em dois módulos diferentes que exercem a mesma função, deste modo o sistema não cria conflitos de registros que se relacionam.

### Módulo Produto

O módulo de produtos do sistema tem o intuito de criar novos registros de produtos, buscar os registros pelo seu nome e listar os registros presentes no banco de dados, juntamente com as funções de edição e exclusão do produto, para que haja uma administração e controle de estoque e disponibilidade de um produto cadastrado no sistema, podendo relacionar o produto com a venda.

### Módulo Venda

O módulo de vendas é onde são registradas as vendas no sistema, que irão contabilizar créditos no financeiro referente a venda de cada item, dentro da aba de vendas possuem as funções de busca, adição, exclusão e também edição, com um adendo na edição de apenas a forma de pagamento, com um detalhe na inclusão de uma venda que você inclui por item vendidos. A tabela de Venda será povoada pelas informações base e a os itens da venda serão registrados na tabela do vendasItens.

### Módulo Financeiro

O módulo do financeiro consiste nos registro de Créditos e débitos da Loja, sendo os créditos ligados as vendas e os débitos registrados na tela do próprio Financeiro, podendo excluir e editar apenas os débitos, pois os créditos para serem editados ou excluídos devem ser feitos via tela Venda, deste modo o sistema é modularizado para cada seção ter suas particularidades, mesmo estando atreladas diretamente umas com as outras.

1. Conclusão

O desenvolvimento do sistema de gerenciamento interno de loja em PHP, utilizando a arquitetura MVC (Model-View-Controller) e MySQL como banco de dados, permitiu a aplicação prática dos conhecimentos adquiridos ao longo da disciplina de Desenvolvimento de Sistemas. O projeto visou criar uma solução robusta e eficiente para a administração de vendas, produtos e controle financeiro, contemplando três CRUDs completos (*Create*, *Read*, *Update*, *Delete*) para cada um desses módulos.

A estruturação do sistema em módulos distintos, organizados em pastas específicas como *Model*, *View*, *Controller* e *Core*, facilitou a manutenção e evolução do código. A utilização da arquitetura MVC proporcionou uma clara separação das responsabilidades, permitindo que as regras de negócio fossem tratadas nos modelos, as interações do usuário nas *views* e a lógica de controle nos *controllers*. Este design promoveu um código mais limpo, reutilizável e fácil de entender.

Durante o desenvolvimento, foram enfrentados desafios significativos relacionados à integração do banco de dados MySQL com as funcionalidades do sistema, assim como à implementação das operações CRUD. No entanto, esses desafios foram superados com a aplicação de boas práticas de programação, testes contínuos e revisões de código. A pasta 'Core' desempenhou um papel crucial ao centralizar funcionalidades essenciais, como a conexão com o banco de dados e o roteamento de requisições, garantindo a coesão e eficiência do sistema.

O resultado final foi um sistema funcional e intuitivo, capaz de atender às necessidades de gerenciamento de uma loja de maneira eficaz. A experiência adquirida com este projeto foi de grande valia, permitindo não apenas a consolidação de conceitos teóricos, mas também o desenvolvimento de habilidades práticas essenciais para a carreira na área de desenvolvimento de sistemas. Assim, conclui-se que o projeto atingiu seus objetivos, demonstrando a aplicabilidade dos conhecimentos teóricos na construção de soluções reais e contribuindo significativamente para a formação acadêmica e profissional do desenvolvedor.

REFERÊNCIAS

QUICK PROGRAMMING. PHP MVC Framework from scratch. Disponível em: <https://www.youtube.com/watch?v=q0JhJBYi4sw&list=PLY3j36HMSHNUCsG7S1lnBg\_mOg3\_VZrcq&ab\_channel=QuickProgramming>. Acesso em: 4 jun. 2024.

STACKOVERFLOW. PHP PDO prepare and execute. Disponível em: <https://stackoverflow.com/questions/51912344/php-pdo-prepare-and-execute#:~:text=The%20prepared%20method%20will%20generate,the%20so%20famous%20SQL%20injection>. Acesso em: 4 jun. 2024.

STACKOVERFLOW. Using password\_hash and password\_verify. Disponível em: <https://stackoverflow.com/questions/28936141/using-password-hash-and-password-verify>. Acesso em: 4 jun. 2024.

STACKOVERFLOW. Directory Structure for MVC. Disponível em: <https://stackoverflow.com/questions/7959673/directory-structure-for-mvc>. Acesso em: 4 jun. 2024.

STACKOVERFLOW. Get current date, given a timezone in PHP. Disponível em: <https://stackoverflow.com/questions/8006692/get-current-date-given-a-timezone-in-php>. Acesso em: 4 jun. 2024.