UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALISSON FRANCISCO EDUARDO MOTA

TÉCNICAS DE B.I APLICADOS EM SISTEMA DE DASHBOARD

BAURU
Julho/2021

ALISSON FRANCISCO EDUARDO MOTA

TÉCNICAS DE B.I APLICADOS EM SISTEMA DE DASHBOARD

Trabalho de Conclusão de Curso do Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. João Pedro Albino

BAURU
Julho/2021

Alisson Francisco Eduardo Mota TÉCNICAS DE B.I APLICADOS EM SISTEMA DE DASHBOARD/ Alisson Francisco Eduardo Mota. – Bauru, Julho/2021- 49 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Dr. João Pedro Albino
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, Julho/2021.
1. Tags 2. Para 3. A 4. Ficha 5. Catalográfica

Alisson Francisco Eduardo Mota

TÉCNICAS DE B.I APLICADOS EM SISTEMA DE DASHBOARD

Trabalho de Conclusão de Curso do Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. João Pedro Albino
Orientador
Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Prof. Dra. Simone das Graças Domingues Prado
Departamento de Computação
Faculdade de Ciências
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Prof. Dr. Kleber Rocha de Oliveira
Campus Experimental de Rosana
Universidade Estadual Paulista "Júlio de Mesquita Filho"

Bauru, 20 de julho de 2021.

Resumo

Na última década, houve um crescimento ascendente na participação econômica de micro, pequenas e médias empresas na contribuição no PIB nacional, sendo responsável por uma grande contribuição que abrange setores do comércio da indústria e do serviço. Entretanto, no último ano houve uma queda brusca nessa contribuição devido a acontecimentos que abalaram a economia mundial e mostrou fragilidade que esses negócios possuem. O presente projeto consistiu na construção de uma aplicação que utiliza-se de métodos de B.I. *(Business Intelligence)* simplificados, para atender o custo benefício desses negócios carentes de ferramentas com essa finalidade e serve como auxiliador do micro, pequeno ou médio empreendedor para gerenciar seus negócios. Foi escolhido como base para a aplicação desses métodos, um sistema em *dashboard* simples desenvolvido utilizando o *framework* Laravel e a linguagem Python para a aplicação desses métodos.

Palavras-chave: Dashboard; Business Intelligence; Métodos; Empresas; Empreendedor; Economia; Python; Laravel.

Abstract

the last decade, there has been a strong growth in the economic participation of micro, small and medium-sized companies in the contribution to the national GDP, being responsible for a large contribution that encompasses sectors, being responsible for a great contribution that covers sectors of commerce, industry and service. However, in the last year there has been a sharp drop in this contribution due to events that shook the world economy and showed the fragility that these businesses have. This project consisted in the construction of an application that uses B.I. (business intelligence) simplified, to meet the cost-effectiveness of these businesses lacking tools for this purpose and I serve as a helper for micro, small or medium entrepreneurs to manage their business.A simple Dashboard system developed using the laravel framework and the Python language to apply these methods was chosen as the basis for applying the methods.

Keywords: Dashboard; Business Intelligence; Methods; Companies; Entrepreneur; Economy; Python; Laravel.

Lista de códigos

Lista de figuras

Lista de abreviaturas e siglas

API	Application Programming Interface
BI	Business Intelligence
CVCS	Centralized Version Control Systems
CRUD	(Create, Read, Update, Delete) Operações básicas para uma base de dados
DSS	Sistema de tomada de decisão
DVCS	Distributed Version Control Systems
DWSL	Ferramenta de Business Intelligence
E2E	End-to-End
ERP	Planejamento de Recursos Empresariais
HTML	HyperText Markup Language
IA	Inteligência Artificial
LVCS	Local Version Control Systems
MariaDB	SGDB que surgiu como fork da base de dados MySQL
MER	Modelo Entidade Relacionamento
MVC	Model-View-Controller
OLAP	Online Analytical Processing
PHP	HyperText Preprocessor
PIB	Produto Interno Bruto
KPI	Indicadores Chave de Performance
RCS	Revision Control Systems
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	Structured Query Language
UNESP	Universidade Estadual Paulista "Júlio de Mesquita Filho"
VCS	Version Control Systems

Sumário

1 Introdução

Essa pesquisa abrange como tema principal o conceito B.I. e todos os assuntos que o envolve, com o objetivo de aplicar métodos abordados por esse conceito, em sistema administrativo simples que não são comumente usados nesse tipo de ferramenta e em ferramenta similares, como os sistemas baseados em *dashboard* por exemplo. Segundo Alves (2018),o termo *Business Intelligence* (Inteligência de negócio) ou simplesmente BI, teve seu conceito estruturado e popularizado na década de 80, pelo Gartner Group (Instituto de pesquisa e análise do setor de tecnologia da informação) e é definido como um processo de coleta, organização e análise de dados, formatação de relatórios e indicadores de gestão que dão suporte às tomadas de decisão no ambiente de negócios (ALVES, 2018). Ou seja, um conjunto de soluções que converte uma grande quantidade de dados em informações úteis para tomadas de decisões estratégicas.

Esse conceito engloba coleta, organização, análise, ação e monitoramento de dados, para proporcionar informações que possam gerar conhecimentos para auxiliar na tomar melhores decisões e saber se os investimentos feitos estão trazendo bons resultados (MORAES, 2018).

Atualmente o B.I. está bem consolidado, tanto que existem ferramentas muito robustas como aponta Morais, que ressalta a importância desse conceito para atender as necessidades de todo tipo de organização sem problemas. De acordo com ele, existem algumas ferramentas que possuem uma alta qualidade no mercado, sendo cada uma aplicada para fornecer a melhor solução para cada empresa específica, como por exemplo o Microsoft Power BI, Qlik e a Tableau.

Após o surgimento desse termo B.I., houve o surgimento do *Data Warehouse*, que são responsáveis por preparar, formatar e armazenar os dados. Isso proporcionou uma evolução dos conceitos DSS (Sistema de suporte à decisão), Planilhas eletrônicas, Geradores de relatórios, Data Marts, OLAP, entre outras. Dando espaço para novas tendências como os espaços para *Chief Data Officer*, voltados especificamente para analisar e transformar dados em estratégias para todas as áreas de uma empresa e as criações de narrativas à partir de dados, gerados depois de análises das informações obtidas de uma empresa (ALVES, 2018).

O presente trabalho pretende demonstrar o funcionamento dos conceitos simplificados de B.I. descrito acima em uma plataforma de *dashboard* simples e com uma boa usabilidade, com o objetivo de oferecer uma ferramenta de apoio focado em micro e pequenas empresas e que não são contempladas com as tecnologias de inteligência de negócios presentes no mercado atual.

No próximo capítulo serão abordados conceitos teóricos importantes para a produção desse projeto; no capítulo três são apresentadas as ferramentas utilizadas como base da

aplicação; por fim, no capítulo quatro falaremos a respeito do desenvolvimento da aplicação, do *dashboard* utilizado como base para a sua implementação e as integrações necessárias para o funcionamento do sistema.

1.1 Problema

As ferramentas de *Business Intelligence* são aplicações que fornecem informações estratégicas que são organizadas, mantidas e formatadas de tal forma que atenda aos gestores da empresa, sendo assim integrados sistemas ERP, fato que eleva a complexidade dos softwares de B.I. (FLORES., 2011).

Foi realizada uma breve pesquisa de campo para avaliar o *status quo* das ferramentas de B.I. presente no mercado e encontrou-se poucas ferramentas com o foco, como o DWSL, ou tendo versões para pequenas empresas, em comparação com os resultados da pesquisa. Apesar da possibilidade do uso dessas ferramentas mais simplificadas, essas aplicações realizam processos computacionais que possam apresentar restrições em relação ao *hardwares* dessas empresas, tendo recursos que extrapolam os serviços que uma pequena empresa necessita. Apesar dos serviços fornecidos por essas ferramentas terem recursos que não serão usados pelo pequeno empreendedor, alguns métodos e recursos simples retirados do B.I. poderiam auxiliá-los a se estabelecer no mercado em momentos de crise e otimização de lucros.

A crise causada pela pandemia levou à suspensão de mais de metade das empresas, causando prejuízos para muitas dessas empresas, por causa de determinações do governo para minimizar os efeitos causados pela crise (SEBRAE, 2021).

O presente trabalho é baseado em fundamentações teóricas envolvendo conceitos relacionados ao B.I. através de artigos e periódicos científicos, esses conceitos também são amplamente utilizados empresas que atuam na implementação das ferramentas de *Business Intelligence*. Além de apresentar uma carência no mercado vigente por produtos que sigam a linha proposta por esse trabalho.

1.2 Objetivos

1.2.1 Objetivo Geral

Criação de uma ferramenta para a analise de dados a partir da modelagem de gráficos em um sistema baseado em *dashboard*, com a finalidade de reproduzir de forma simplificada um sistema de B.I. que seja mais compacta e acessível em comparação aos encontrados no mercado e comumente usados pelas empresas.

1.2.2 Objetivos Específicos

O presente trabalho tem como objetivos específicos:

- Identificar o comportamento de mercado que um negocio está inserido com a analise dos resultados de compras e vendas de produtos que ela possui, sendo consultada uma base de dados compostas por *seeders* fidedignos de produtos desta empresa.
- Construir a infraestrutura necessária para o desenvolvimento de um sistema *dashboard* utilizando *Laravel* e um algoritmo em *Python* utilizando *Docker*;
- Modelar banco de dados necessário para suprir os requisitos do aplicativo;
- Desenvolver algoritmo para analisar os dados e registrar no banco de dados;
- Elaborar *dashboard* para integrá-lo ao algoritmo desenvolvido.

1.3 Justificativa

Registros mostram que em 1985, cerca de 21% do PIB brasileiro compõem a participação dos pequenos e médios negócios e mais recente a isso, uma pesquisa feita em 2011 mostrou uma participação de aproximadamente 27cc dessa contribuição. Atualmente essas empresas são as principais geradoras de riqueza no Comércio no Brasil, já que respondem por 53,4% do PIB nacional, tendo como a maior participação o setor de serviços, que corresponde a 36,3% da participação neste setor (ALVES, 2018).

As micro e pequenas e médias empresas em sua maioria não possuem acesso à informações sobre o mercado que estão inseridos, sendo essa uma das principais causas do fechamento de pequenos negócios (XERPAY, 2019).

Assim uma ferramenta que tem como objetivo de aplicar métodos de outras ferramentas conceituadas no mercado na área de relacionadas ao B.I., tendo como objetivo os estudos das informações obtidas no contexto mercadológico que estão estabelecidas para determinar as decisões ideais para serem executadas, tendo espaço para ser desenvolvida no estado da arte presente.

2 Fundamentação Teórica

2.1 Ferramentas de Business Intelligence

Para o desenvolvimento da pesquisa, serão descritos as ferramentas e soluções baseadas BI, como suas funcionalidades e limitações, com o foco em embasar a aplicação que desenvolvida ao final do presente trabalho. Utilizando fontes de diferentes artigos e livros de profissionais e especialistas na área.

A análise realizada no presente trabalho utilizou aplicações já estabelecidas no mercado, decidindo se basear inicialmente os sistemas de Microsoft Power BI, Qlik e a Tableau.

2.1.1 *Microsoft Power BI*

Microsoft Power BI tem foco no acesso aos dados, proporcionando modelá-los e visualizá-los para a criação de relatórios de fácil personalização com seus KPIs e sua marca. Obtendo respostas rápidas impulsionadas por IA para perguntas de negócios, tendo como foco a proteção de dados de ponta a ponta, integração com o Azure e o Office e conectores de dados extensivos.

2.1.2 *Qlik*

Qlik tem o foco de acelerar o caminho para o valor do negócio. Fechando as lacunas entre dados, *insights* e ação com *active intelligence* fornecida pela única solução ponta a ponta para integração e análise de dados, tendo como pilares a integração e análise de dados e um plataforma de desenvolvimento.

2.1.3 *Tableau*

Tableau é uma plataforma de análise visual interativa para o cliente e fornece a capacitação de pessoas e organizações com o foco de obter o máximo de dados possíveis para modela-los de uma forma otimizada, tendo pacotes que variam com base na equipe que irá utilizá-lo e com as integrações com outros sistemas que serão feitas.

2.2 Arquitetura de *Software*

O estudo da Arquitetura de *Software* consiste em entender como os sistemas e aplicativos são desenhados e construídos, por meio de um conjunto das principais decisões de estruturação de um sistema. A arquitetura inclui algumas decisões estruturais, que consistem em uma

evolução constante dos padrões do sistema. A formação desse modelo é essencial para a produção e evolução de produtos de qualidade e por composição de métricas (MEDVIDOVIC; TAYLOR, 2010).

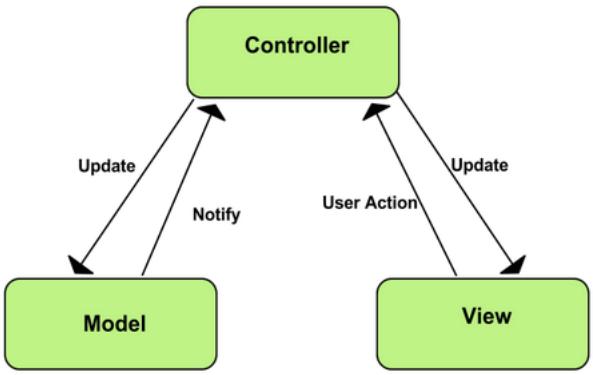
2.2.1 Arquitetura MVC

A arquitetura MVC é um padrão de arquitetura que contém três módulos: *model*, *view* e *controller*. Os módulos isolados uns dos outros facilitam o entendimento e modificação de cada módulo separadamente, sem ter que conhecer tudo sobre os outros módulos (POP; ALTAR, 2014). O significado de cada um dos termos segue abaixo:

- *Model* - Representação do modelo de dados, contendo a lógica para a leitura, escrita e validações;
- *View* - Representação da visualização dos dados que o *Model* contém, sendo a interface de comunicação da aplicação para o usuário;
- *Controller* - Controla os dados no *Model* por requisições e métricas de tratamento dos dados e tem como responsabilidade atualizar as *views* quando necessário.

O esquema da Figura 1 apresenta o funcionamento da arquitetura MVC.

Figura 1 – Arquitetura *Model-View-Controller*



Fonte: https://ahex.co/wp-content/uploads/2018/08/0H9Vj.png. Acesso em: 29 de junho de 2021.

2.2.2 *Framework*

O *framework* é uma ferramenta para o desenvolvimento de *software* que reúne diversos módulos de códigos genéricos e básicos usados como um pacote por desenvolvedores para o desenvolvimento de sites, sistemas e aplicações. Dessarte, esse pacote de códigos prontos serve como base quando um projeto é iniciado, facilitando no desenvolvimento de aplicações sem a necessidade de codificar toda sua estrutura, proporcionando uma qualidade e padronização, que possa ser utilizado em quaisquer outros projetos (SOUZA, 2019).

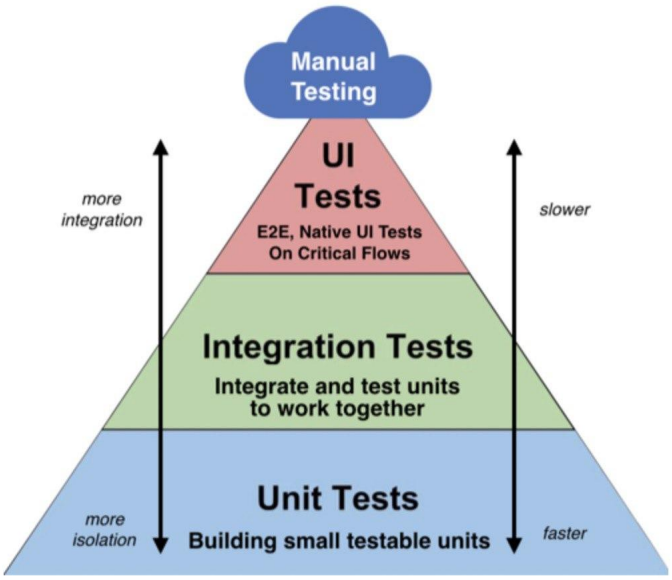
2.2.3 Testes de *software*

Conforme o desenvolvimento de *software* foi se tornando mais complexo passou-se a existir a necessidade de testá-lo e garantir seu funcionamento. Na computação moderna existem três categorias principais de testes de *software*: os testes unitários, os testes de integração e os testes End-to-End (E2E).

- Teste Unitário: como diz o próprio nome, tem como objetivo testar uma unidade do *software*; é o teste mais simples dos três, ele espera uma entrada e uma saída; é mais utilizado para garantir o funcionamento de funções e lógicas simples; geralmente, quando são necessários informações externas, são utilizados dados *mokados* simulando os parâmetros reais.
- Testes de integração: visam testar a integração entre diferentes módulos do *software*; geralmente são executados utilizando *requests* reais para garantir o funcionamento do sistema como um todo;
- Testes E2E: visam simular a interação de um usuário real com o *software*, testando todo o fluxo de interações existentes dentro dele, do começo ao fim; com isso, é possível englobar todos os módulos; é o teste mais difícil a ser executado, entretanto é o que apresenta a maior confiabilidade.

A Figura 2 apresenta uma pirâmide que sugere a proporção de testes que devem existir dentro de um sistema, como também a relação de integração e velocidade.

Figura 2 – Relação de testes de *software*



Fonte: https://miro.medium.com/max/1494/1*IqWygfNJqWQ4VCyjecQ6Eg.png. Acesso em: 29 de novembro de 2020.

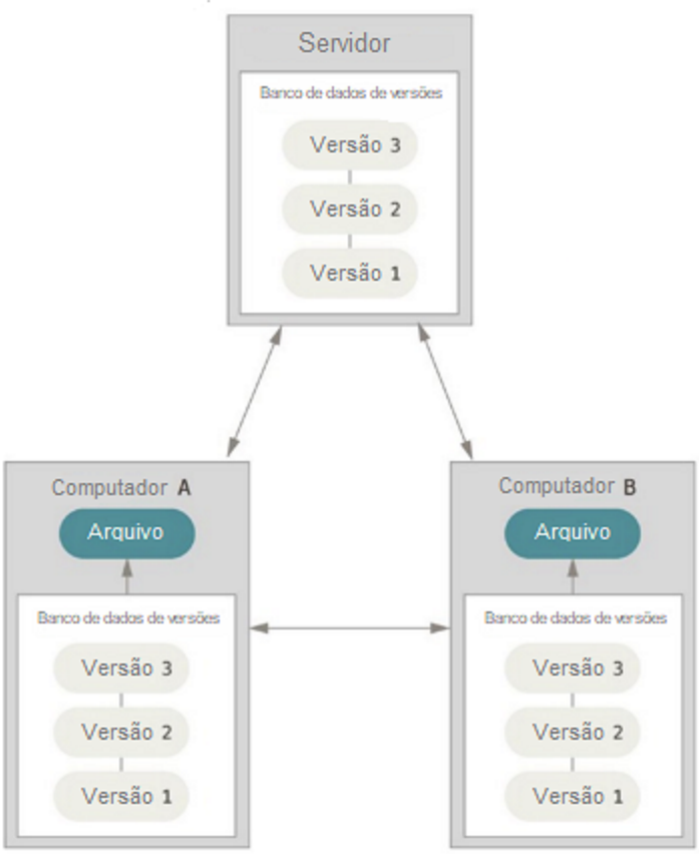
2.3 Sistema de controle de versões

Um sistema de controle de versões (VCS) tem como objetivo armazenar um histórico de *updates* feitas nos arquivos ao decorrer do período de desenvolvimento em um projeto, para assim, ser possível resgatar versões anteriores.

Paralelamente, permite-se realizar comparações com mudanças que foram feitas, revertê-las e verificar o autor de cada alteração feita em projetos compartilhados ou privados. Gerando benefícios ao projeto, como a gestão dos arquivos de projetos e refatorações para a correção de *bugs* especificos. São usados três tipos principais de VCS (BITBUCKET, 2020):

- *Local Version Control Systems* (LVCS): Uma solução desenvolvida para controlar o versionamento localmente. Possui um banco de dados para manter salvo as mudanças nos arquivos que estão versionados. Como as ferramenta de LVCS é o RCS.
- Centralized Version Control Systems (CVCS): Sistema de controlamento de versões que possui um único servidor contendo todos os arquivos versionados, e todas as máquinas precisarão acessar os arquivos através dele. Porem essa solução é deficitária pois apresenta servidor único, fazendo com que em casos de falhas, há riscos do historico de versionamentos serem perdidos. Um exemplo é o Subversion.
- Distributed Version Control Systems (DVCS): Tendo como principal atualmente a ferramenta Git a ferramenta, a principal diferença entre o DVCS e o CVCS são os espelhamentos de todos os arquivos do projeto, possuindo toda a estrutura de versionamento do repositório, como mostra a Figura 3. Isso é vantajoso, pois caso algum servidor apresente alguma falha basta apenas alguma máquina ter acessado o repositório para ter todas as informações do controlamento de versão.

Figura 3 – Exemplo da arquitetura DSVC.

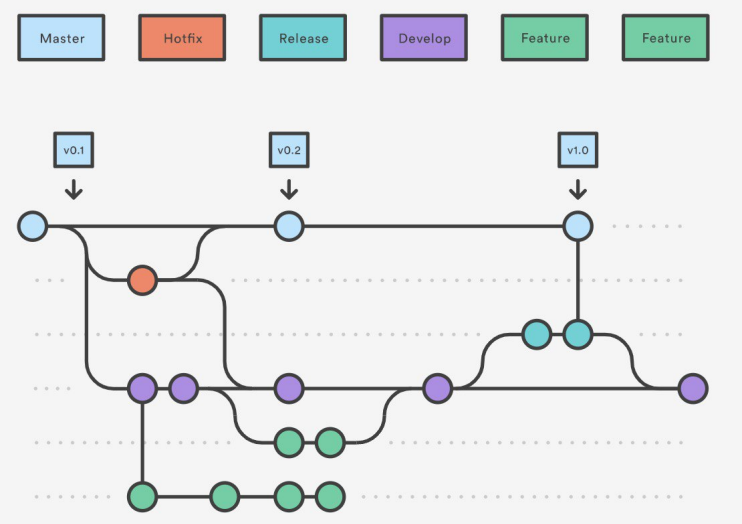


Fonte: Chacon; Straub (2016).

2.4 Gitflow

Gitflow é um fluxo de controle de versão definido por Vincent Driessen. Nele é definida uma estrutura de *branches*, onde cada *branch* tem sua funcionalidade específica (Figura 4), com o objetivo de prover uma estrutura mais definida de desenvolvimento, onde é possível rastrear as mudanças feitas no código com mais facilidade, aumentar a velocidade de desenvolvimento, facilitar a resolução de conflitos de código e ter um controle de versão mais organizado. O funcionamento do Gitflow ocorre a partir de duas *branches* principais, a *master* e a *develop*. Sendo que na *master* deve ficar o código que será publicado, e na *develop* deve ficar o código que receberá novas *features*, no qual para cada *features* que será desenvolvida, deve-se criar uma *branch* especifica para essa modificação. (BITBUCKET, 2020).

Figura 4 – Exemplo de estrutura de *branches* utilizando Gitflow.



Fonte: Atlassian (2020).

2.5 Banco de dados

Um banco de dados é uma coleção organizada de informações, ou dados, de forma estruturadas, normalmente armazenadas eletronicamente em um sistema de computador estruturado em linhas e colunas. Um banco de dados é controlado por um sistema de gerenciamento de banco de dados (DBMS), podendo ser relacional (SQL) ou não relacional (NoSQL), como por exemplo Oracle Database, MySQL, Firebase e MongoDB. Juntos, os dados e o DBMS, juntamente com os aplicativos associados, são descritos como sistema de banco de dados (ORACLE, 2021).

2.5.1 Linguagem de Query

SQL é uma linguagem de programação usada por quase todos os bancos de dados relacionais para consultar, manipular e definir dados e fornecer controle de acesso através de *queries*, por meio de *requests*. O SQL foi desenvolvido pela primeira vez na IBM nos anos 1970, com a Oracle como principal contribuinte, o que levou à implementação do padrão SQL ANSI. Embora o SQL ainda seja amplamente usado, novas linguagens de programação estão começando a aparecer (ORACLE, 2021).

3 Ferramentas

3.1 Git

O Git é um DVCS que originou-se da necessidade de realizar a manutenção de grandes projetos com muitos desenvolvedores trabalhando simultaneamente. Criado por Linus Torvalds em 2005 para auxiliar no desenvolvimento do *kernel Linux* após um conflito entre o criador e a empresa dona do DVCS proprietário BitKeeper (GIT, 2020).

Apresentando suporte para programação não linear, eficiência na manipulação de uma massa de dados e controle dos históricos de modificações em projetos públicas e privados.

3.2 Docker

Docker é um *software open source* desenvolvido com o objetivo de facilitar o desenvolvimento e o *deploy* de aplicações. Tem a capacidade de criar uma plataforma unificada que possui a habilidade de compilar e executar *software* dentro de um *container* isolado.

Containers são processos que são executados isolados da máquina *host*. Dentro de um *container* há disponível um *kernel Linux*, no caso de não estar sendo executado nativamente em um sistema operacional baseado em Linux e um próprio sistema de arquivos disponibilizado por uma *Docker image*, uma vez que sendo executado em Linux, ele compartilha os recursos do *kernel* (DOCKER, 2020).

3.3 PHP

O PHP é uma linguagem de *script open source* dinâmica, que pode ser utilizada em diversas situações, como *server-side scripting*, *command line scripting* e em aplicações de *desktop*. A principal vantagem da linguagem PHP é a capacidade de ser acoplada dentro de linguagem de marcação HTML.

Focando-se na experiência no desenvolvimento *web*, em conjunção com a capacidade de ser executada do lado do servidor, lhe permite criar conteúdos dinâmicos, trabalhar com *cookies* e gerar formulários e *cruds* (PHP, 2020).

3.4 Javascript

JavaScript é uma linguagem de alto nível, interpretada e baseada em objetos com funções de primeira classe, mais conhecida como a linguagem de script para páginas *web*,

embora possua igual pertinência em ambientes sem a utilização de *browser*, tais como Node.js, Apache CouchDB e Adobe Acrobat.Baseando-se em protótipos, multi-paradigma e dinâmica, suportando estilos de orientação a objetos, imperativos e declarativos (WDN-WEB-DOCS, 2021).

3.5 Laravel

Laravel é um *framework* PHP de código aberto gratuito e destinado ao desenvolvimento de aplicações *web* seguindo o padrão de arquitetura MVC. Concebido utilizando diversos fundamentos, entre os principais estão (LARAVEL, 2020):

- *Model*: Determina a modelagem da regra de negócio dentro da arquitetura MVC, sendo responsável por interagir com o banco de dados por meio de operações de criar, atualizar, ler e deletar.
- *View*: Tem como função apresentar os dados para o usuário separando a lógica de apresentação da de negócio. Geralmente é representada por arquivos HTML, que podem ser modularizados contendo partes de uma página da *web*, como arquivos contendo o *header* e *footer*.
- *Controller*: Encarregado de transitar os dados entre o *model* e a *view*.
- *Migration*: Executa o versionamento do banco de dados, com o objetivo de manter o histórico das alterações realizadas, e assim, ser possível revertê-las caso haja necessidade.
- *Eloquent ORM*: Transforma *queries* complexas do SQL em código simplificado em PHP.

3.6 PhpMyAdmin

PhpMyAdmin é uma ferramenta de software livre em PHP, destinada a administração do MySQL na *web*, suportando uma ampla gama de operações no MySQL e MariaDB. Operações frequentemente usadas, como por exemplo gerenciamento de bancos de dados, tabelas, colunas, relações, índices, usuários e permissões, podem ser executadas por meio da interface do usuário, enquanto você ainda tem a capacidade de executar diretamente qualquer instrução SQL. (PHPMYADMIN, 2003)

4 Desenvolvimento

O projeto originou-se da necessidade de suprir as necessidades de micro e pequenas empresas, principalmente empresas familiares, para analisar dados de compra e venda de produtos ou serviços, com a finalidade de auxiliar na otimização de lucros e prevenção de perdas. Embasado-se em métodos simples de estatística e conceitos de *business intelligence*, tendo como foco as principais ferramentas de B.I. presentes no estado da arte vigente.

A aplicação foi desenvolvida utilizando ferramentas como Laravel e Python para o desenvolvimento do *back-end* e tendo apoio do Blade, adjunto do Javascript para proporcionar dinamicidade e criação de tabelas, para o desenvolvimento do *front-end* e PhpMyAdmin como gerenciador de banco de dados.

4.1 Pré-desenvolvimento

4.1.1 Levantamento dos requisitos

Antes de iniciar, foi necessário avaliar o escopo de projeto e o desenvolvimento diante da proposta realizada. Como dito anteriormente, foi feito uma aplicação com base nos conceitos de B.I. para a implementação de sistemas simples, como o *dashboard*.

Para o desenvolvimento do presente projeto, foi necessário o desenvolvimento paralelo de um *dashboard* simples para acoplar o algoritmo proposto

Para alcançar esse objetivo, foi importante definir todas as telas e funcionalidades que estão presentes no projeto. No âmbito das telas foram definidas:

- Tela de *login* do usuário;
- Tela de tabela de clientes;
- Tela de tabela de funcionários;
- Tela de tabela com o histórico de vendas;
- Tela de tabela com o histórico de compras;
- Tela de gráficos com previsões para vendas/compras futuras;

Todavia, ocorreram alguns percalços que inviabilizaram algumas decisões tomadas no planejamento do projeto, precisando adequar o projeto para os novos contextos que foram se revelando.

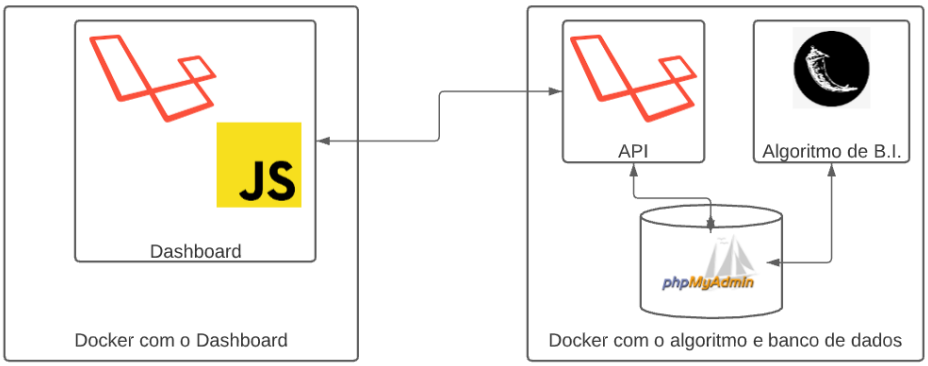
4.1.2 Percalços encontrados

Primeiramente tentou-se uma abordagem que focasse na separação dos serviços pertinentes para o funcionamento do *dashboard* e os vinculados ao algoritmo em Python.

Tendo em vista esse objetivo, foi projetado anteriormente uma arquitetura compostas por dois *containers* para compor esses serviços independentemente, afim de distinguir os serviços pertinentes ao sistema em *dashboard* a API e o algoritmo de *Business Intelligence*. Dessarte, um dos *containers* ficou responsável por englobar o projeto em Laravel e algumas aplicações em *javascript*, para o outro foi destinado o algoritmo em Flask, juntamente com o banco de dados PhpMyAdmin e uma API em Laravel, sendo ambos os *containers* em Docker.

A Figura 5, mostra como teriam sido estruturados os serviços e como seria a comunicação entre esses serviços. O *container* que engloba o algoritmo de B.I. e o banco de dados, teria como intermediador do *dashboard* a API, que enviaria por meio de um *json* as informações do banco de dados e receberia as *requests* de *update* oriundas do *dashboard* e os dados no banco seria tratados pelo algoritmo para serem reenviados para o *dashboard* via API.

Figura 5 – Arquitetura antiga do projeto.



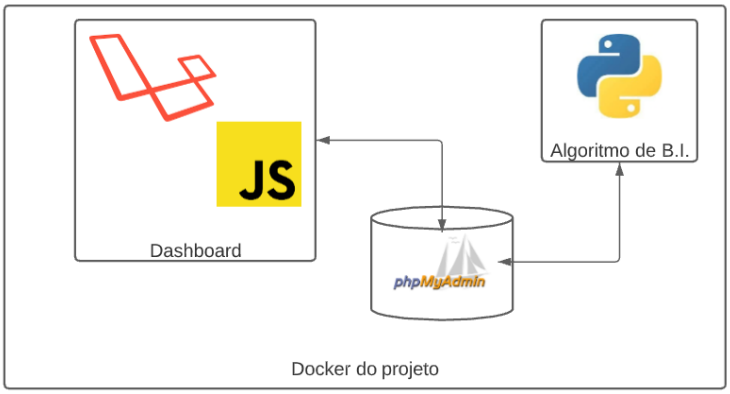
Fonte: Elaborado pelo autor.

Todavia, essa arquitetura se mostrou inviável para o projeto, pois a comunicação entre os ambientes não foi efetiva, precisando criar redundâncias de serviços para cada um dos *containers* e apresentar problemas de comunicação em relação a uma massa de dados, fazendo com que haja *overclock* e lentidão no fluxo de dados, além de alguns contextos o sistema apresentar instabilidade causada pelo não reconhecimento dos *containers* entre si.

Para viabilizar o ideal do projeto apresentado anteriormente, seria consentâneo realizar adições de outros serviços ao trabalho e adequar os existentes, para adaptar o sistema proposto para variados contextos de funcionamento. Tais mudanças acarretaram em um aumento exponencial da complexidade do projeto, obrigando o usuário a possuir uma infraestrutura que possa não se enquadrar com o seu cotidiano e tornando inviável a proposta inicial do presente trabalho em focar no micro e pequeno empreendedor.

Simplificando o presente trabalho, se alcançou uma arquitetura monólito de ambiente singular, que extinguiu o problema de comunicação em detrimento de algumas técnicas de estatísticas que precisaram ser simplificadas, entretanto conseguiram acertar a proposta do trabalho. Na Figura 6, apresenta-se o esboço da arquitetura adotada para o sistema proposto.

Figura 6 – Arquitetura do projeto atual.



Fonte: Elaborado pelo autor.

Embora o problema de fluxo de dados tenha sido resolvido, a incompatibilidade encontrada em conter o *framework* Laravel e Flask em um mesmo ambiente de desenvolvimento, acarretando erros ligados a concorrência no uso dos serviços, sendo resolvidos implementando um *script* com o algoritmo de B.I. adaptado e simplificado, resolvendo o percalço e se aliando com a proposta vigente.

Com a finalidade de estruturar os dados tratados oriundos do algoritmo e do *dashboard*, precisou ser realizada uma modelagem no banco de dados, para facilitar as operações SQL realizados por esses serviços.

4.1.3 Modelagem do banco de dados

Na modelagem do banco de dados o objetivo foi definir as tabelas existentes, os campos dentro destas e a relação entre eles. Essa etapa do projeto foi muito importante, pois após a definição da modelagem é muito difícil realizar qualquer modificação na mesma, pois gera muitos efeitos colaterais, sendo necessário fazer alterações em muitos lugares.

Diante disso, a definição das tabelas e das colunas foi realizada da seguinte maneira:

- Usuarios: id, nome, email, password, created_at, update_at e deleted_at;
- Funcionarios: id, nome, funcao, created_at, update_at e deleted_at;
- Produtos: id, nome, descricao, created_at, update_at e deleted_at;
- Compras: id, id_funcionario, id_produto, preco, created_at, deleted_at;

- Vendas: id, id_funcionario, id_produto, preco, created_at, deleted_at;
- Previsao: id, id_produto, preco, created_at, deleted_at;

Os campos created_at, update_at e deleted_at tem como registrar os dados que foram criados, atualizados e deletados respectivamente. Já o campo id presente em todas as tabelas são *primary keys* e id_produto presente nas tabelas Compras e Vendas são *foreign keys*, respectivamente como objetivo identificar cada registro individualmente e fazer a conexão entre as tabelas dependendo da tabela que tem como objetivo.

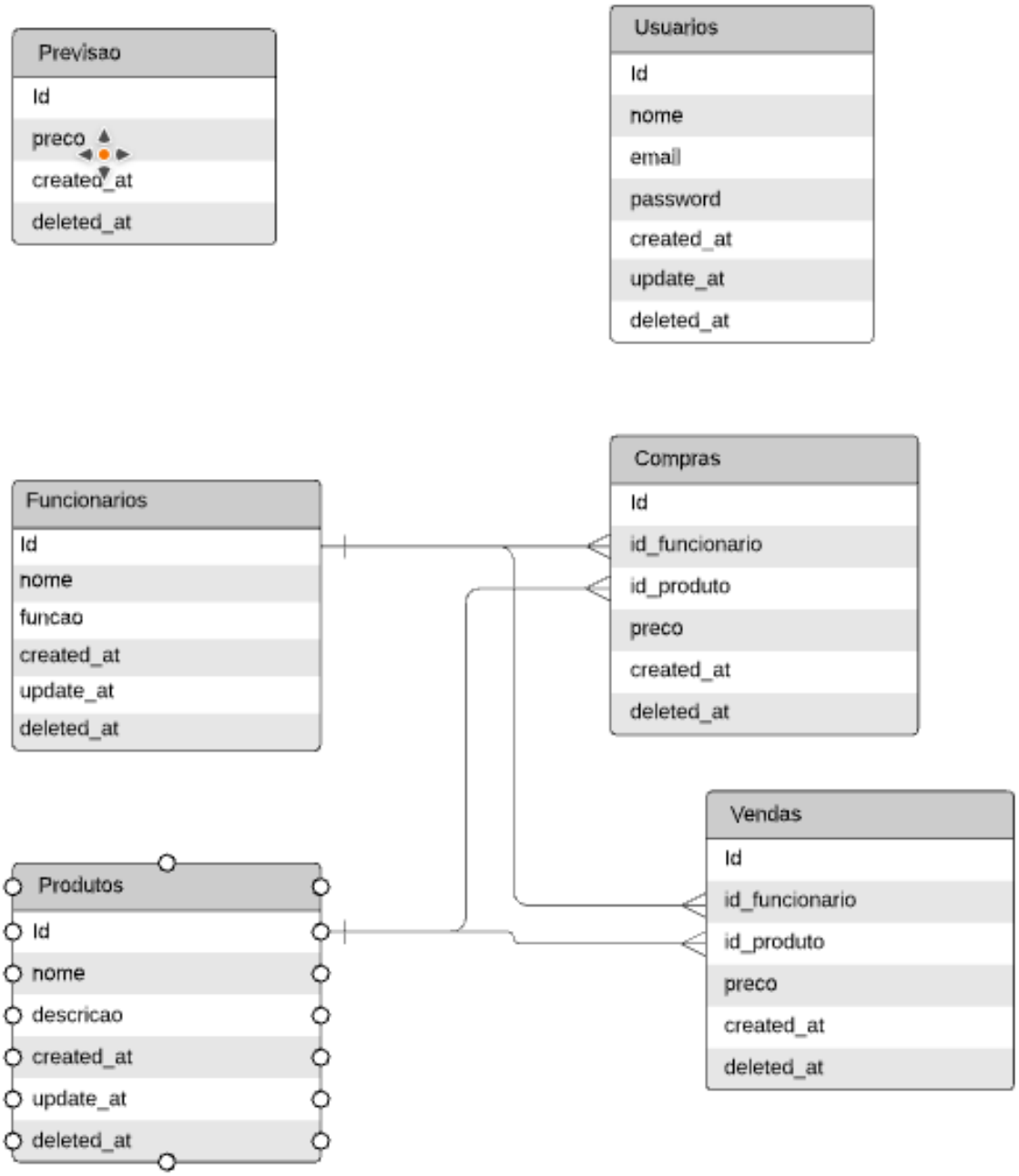
As tabelas Funcionarios e Produtos são relacionadas com as tabelas Compras e Vendas e as tabelas Usuarios, Funcionarios e Previsao não são relacionadas, sendo o algoritmo externo ao *dashboard* responsável por manusear a tabela Previsao e com os dados lidos da tabela Compras e Vendas.

Já as conexões entre essas tabelas, são relacionadas na tabela da seguinte forma:

- Usuarios não se relacionam com as outras tabelas;
- Previsao e Produtos possuem a relação 1 para 1 e de maneria oposta 1 para N;
- Funcionarios e Compras possuem a relação 1 para N e de maneira oposta 1 para 1;
- Funcionarios e Vendas possuem a relação 1 para N e de maneira oposta 1 para 1;
- Produtos e Compras possuem a relação N para 1 e de maneira oposta 1 para 1;
- Produtos e Vendas possuem a relação N para 1 e de maneira oposta 1 para 1;

o Modelo Entidade Relacionamento (MER) representativo de todas as relações encontra-se na Figura 7.

Figura 7 – MER do banco de dados



Fonte: Elaborado pelo autor.

Posteriormente ao levantamento de requisitos das telas e suas funcionalidade, tal como o planejamento da arquitetura e da modelagem do banco de dados, houve uma distinção dos processos que foram atribuídos para cada componente pertencente ao projeto.

4.1.4 Estrutura do projeto

O atual projeto consiste em serviços implementados no *framework* Laravel, sendo que suas *views* desenvolvidas utilizando Blade e Javascript *vanilla* e processando paralelamente um *script* com o algoritmo em Python, integrando todos esses módulos em um ambiente desenvolvido com a ferramenta Docker.

4.1.4.1 Docker

A implementação do Docker no projeto, necessitou a sua instalação e a do *software* Docker Compose e desse modo sendo criado um arquivo chamado *docker-compose.yml* na raiz do projeto em Laravel. Dentro deste arquivo estão as configurações das *Docker images* necessárias para serem executados o Mysql, funcionando com o apoio da ferramenta de suporte PhpMyAdmin e a linguagem PHP, sendo suportada por um servidor local Nginx. O Código 1 mostra o arquivo necessário para executar essas imagens em Docker.

Código 1 – Imagens *dockerizadas* do projeto.

```
version: "3.1"

services:
  #MySQL Service
  mysql:
    container_name: mysql
    image: mysql:5.6
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: root
    ports:
    - "3306:3306"
    volumes:
      - ../data/mysql:/var/lib/mysql
    networks:
      - app-network
  #PhpMyAdmin Service
  phpmyadmin:
    container_name: phpmyadmin
    image: phpmyadmin/phpmyadmin
    restart: unless-stopped
    links:
    - "mysql:db"
    ports:
    - "3380:80"
    networks:
      - app-network
  #PHP Service
  php:
    container_name: php
    build:
      context: ./docker-dashboard/php
    restart: unless-stopped
    volumes:
      - ../:/var/www
      - ~/.ssh:/home/www/.ssh
      - ./docker-dashboard/php/php/local.ini:/usr/local/etc/php/conf.d/local.ini
```

```
    networks:
      - app-network
  #Nginx Service
  nginx:
    image: nginx:alpine
    container_name: nginx
    restart: unless-stopped
    volumes:
      - ../:/var/www
      - ./docker-dashboard/nginx/conf.d:/etc/nginx/conf.d
    tty: true
    ports:
      - "80:80"
      - "443:443"
    networks:
      - app-network

volumes:
  mongovolume:
    driver: local

#Docker Networks
networks:
  app-network:
    driver: bridge
```

Todavia, ainda houve a necessidade da criação do *script*, apresentado no Código 2, para a instalações de extensões e dependências necessária para melhorar a experiência de desenvolvimento e oferecer suporte para as implementações desenvolvidas em Python e Javascript.

Código 2 – *Dockerfile* do projeto.

```
FROM php:7.0-fpm

# Set working directory
WORKDIR /var/www/

# Install dependencies
RUN apt-get update && apt-get install -y \
    build-essential \
    libpng-dev \
    libjpeg62-turbo-dev \
    libfreetype6-dev \
    locales \
    zip \
    jpegoptim optipng pngquant gifsicle \
    vim \
    unzip \
    git \
    curl \
    python2.7 \
    python-pip \
    python \
    libpq-dev \
    libmcrypt-dev \
    zlib1g-dev \
    libicu-dev \
```

```
    libxml2-dev \
    libmemcached-dev \
    libssl-dev \
    openssh-client \
    git-core \
    g++ \
    gnupg && \
    curl -sL https://deb.nodesource.com/setup_8.x | bash - && \
    apt-get update && \
    apt-get install -y --no-install-recommends nodejs && \
    curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | apt-key add - && \
    echo "deb https://dl.yarnpkg.com/debian/ stable main" | tee /etc/apt/sources.list.d/yarn.list &&
    apt-get update && \
    apt-get install -y --no-install-recommends yarn && \
    npm install -g npm


# Clear cache
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

# Install extensions
RUN docker-php-ext-configure gd --with-freetype-dir=/usr/include/ --with-jpeg-dir=/usr/include/ \
  && docker-php-ext-configure intl \
  && docker-php-ext-configure pdo_mysql --with-pdo-mysql=mysqlnd \
  && docker-php-ext-install -j$(nproc) mbstring xml iconv mcrypt gd intl xmlrpc zip bcmath sockets pc

RUN curl https://phar.phpunit.de/phpunit.phar -L > phpunit.phar \
  && chmod +x phpunit.phar \
  && mv phpunit.phar /usr/local/bin/phpuni

COPY ./xdebug.ini /usr/local/etc/php/conf.d/xdebug.ini


# Install composer
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=comp

RUN npm install bower gulp -g

RUN npm install bower-update-all -g

# Add user for laravel application
RUN groupadd -g 1000 www
RUN useradd -u 1000 -ms /bin/bash -g www www

# Copy existing application directory permissions
COPY --chown=www:www . /var/www/Dashboard-bi

# Change current user to www
USER www
```

4.1.4.2 Laravel

Para a criação do projeto em Laravel, o primeiro passo foi a instalação do gerenciador de pacotes Composer localmente, então instalado o pacote do Laravel e gerado um novo projeto, através dos comandos *composer create-project laravel/laravel Dashboard-bi*. Em seguida foi inserido os *scripts* em Docker e suas dependências para o seu funcionamento, juntamente com a

pasta *node-modules*, responsável por armazenar *libs* e processos responsáveis por proporcionar a dinamicidade e a criação de gráficos no projeto.

4.2 Projeto

4.2.1 *Migrations*

No desenvolvimento do projeto, primeiramente, foram criadas as tabelas do banco de dados, por meio das *migrations* via o Laravel. Para gerar uma *migration* foi necessário utilizar o *artisan*, uma *interface* de linha de comando, onde foi executado o comando *php artisan make:migration* Posteriormente a sua execução, foi criado um arquivo dentro da pasta *database/migration* com o nome passado como argumento no comando. Dentro desse arquivo existe uma classe que é estendida de outra classe chamada *Migration*, que fornece diversos métodos úteis para transformar código PHP em SQL. Como é possível visualizar nos códigos abaixo, responsável por criar as respectivamente tabelas Usuarios, Funcionarios, Previsao, Produtos, Compras e Vendas.

Criação da tabela usuário no banco de dados:

Código 3 – *Migration* de Usuario.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsuariosTable extends Migration {

        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
                Schema::create('usuarios', function(Blueprint $table)
                {
                        $table->increments('id');
                        $table->string('nome');
                        $table->string('email')->unique();
                        $table->string('password', 60);
                        $table->rememberToken();
                        $table->timestamps();
            $table->softDeletes();

                });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
```

```
        public function down()
        {
                Schema::drop('usuarios');
        }

}
```

Criação da tabela funcionário no banco de dados:

Código 4 – *Migration* de Funcionário.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateFuncionariosTable extends Migration {

        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
                Schema::create('funcionarios', function(Blueprint $table)
                {
                        $table->increments('id');
                        $table->string('nome');
                        $table->string('funcao');
                        $table->timestamps();
                $table->softDeletes();

                });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
                Schema::drop('funcionarios');
        }

}
```

Criação da tabela previsão no banco de dados:

Código 5 – *Migration* Previsão.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreatePrevisaoTable extends Migration {

        /**
```

```
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
                Schema::create('previsao', function(Blueprint $table)
                {
                        $table->increments('id');
                        $table->string('id_produto');
                        $table->string('preco');
                        $table->timestamps();
                $table->softDeletes();

                });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
        Schema::dropIfExists('previsao');
        }

}
```

Criação da tabela produtos no banco de dados:

Código 6 – *Migration* de Produtos.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateProdutosTable extends Migration {

        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
                Schema::create('produtos', function(Blueprint $table)
                {
                        $table->increments('id');
                        $table->string('nome');
                        $table->string('descricao');
                        $table->timestamps();
                $table->softDeletes();

                });
        }

        /**
         * Reverse the migrations.
```

```
         *
         * @return void
         */
        public function down()
        {
                Schema::drop('produtos');
        }

}
```

Criação da tabela compras no banco de dados:

Código 7 – *Migration* de Compras.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateComprasTable extends Migration {

        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
                Schema::create('compras', function(Blueprint $table)
                {
                        $table->increments('id');
            $table->foreign('id_funcionario')->references('id')->on('funcionario');
            $table->foreign('id_produto')->references('id')->on('produto');
            $table->decimal('preco', 5, 2);
                        $table->timestamps();
            $table->softDeletes();

                });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
                Schema::drop('compras');
        }

}
```

Criação da tabela vendas no banco de dados:

Código 8 – *Migration* de Vendas.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
```

```
class CreateVendasTable extends Migration {

        /**
         * Run the migrations.
         *
         * @return void
         */
        public function up()
        {
                Schema::create('vendas', function(Blueprint $table)
                {
                        $table->increments('id');
            $table->foreign('id_funcionario')->references('id')->on('funcionario');
            $table->foreign('id_produto')->references('id')->on('produto');
            $table->decimal('preco', 5, 2);
                        $table->timestamps();
            $table->softDeletes();

                });
        }

        /**
         * Reverse the migrations.
         *
         * @return void
         */
        public function down()
        {
                Schema::drop('vendas');
        }

}
```

4.2.2 *Models*

Em seguida foi necessária a criação dos *models* respectivos de cada tabela. Um *model* tem como objetivo realizar as configurações de cada tabela, como definir campos privados e públicos, chaves primárias, *timestamps* e relações entre as tabelas. Assim como as *migrations*, as *models* segue o mesmo padrão, existe uma classe que estende da classe Model que realiza funções auxiliares e foi necessário executar o comando *php artisan make:model*, sendo necessário passar o nome da tabela como parâmetro . Como é possível visualizar nos códigos responsáveis por realizar o tratamento dos dados para as respectivamente tabelas Usuarios, Funcionarios, Previsao, Produtos, Compras e Vendas.

Código 9 – *Model* de Usuarios.

```
<?php namespace App;

use Illuminate\Auth\Authenticatable;
```

```
use Illuminate\Database\Eloquent\Model;
use Illuminate\Auth\Passwords\CanResetPassword;
use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
use Illuminate\Contracts\Auth\CanResetPassword as CanResetPasswordContract;

class User extends Model implements AuthenticatableContract, CanResetPasswordContract {

        use Authenticatable, CanResetPassword;

        /**
         * The database table used by the model.
         *
         * @var string
         */
        protected $table = 'usuarios';

        /**
         * The attributes that are mass assignable.
         *
         * @var array
         */
        protected $fillable = ['nome', 'email', 'password'];

        /**
         * The attributes excluded from the model's JSON form.
         *
         * @var array
         */
        protected $hidden = ['password', 'remember_token'];

}
```

Código 10 – *Model* de Funcionários.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\SoftDeletes;

class Funcionarios extends Model
{
    use SoftDeletes,

    protected $table = 'funcionarios';

    protected $dates = ['created_at', 'updated_at', 'deleted_at'];

    protected $guarded = ['id', 'created_at', 'updated_at', 'deleted_at',];

    protected $fillable = ['nome', 'funcao'];

}
```

Código 11 – *Model* Previsao.

```
<?php
```

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\SoftDeletes;

class Previsao extends Model
{
    use SoftDeletes,

    protected $table = 'previs o';

    protected $dates = ['created_at', 'deleted_at'];

    protected $guarded = ['id', 'created_at', 'deleted_at',];

    protected $fillable = ['preco'];


    public function usuario(): BelongsTo
    {
        return $this->belongsTo(Usuario::class, 'id_produto');
    }

}
```

Código 12 – *Model* de Produtos.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\SoftDeletes;

class Produto extends Model
{
    use SoftDeletes,

    protected $table = 'produtos';

    protected $dates = ['created_at', 'update_at', 'deleted_at'];

    protected $guarded = ['id', 'created_at', 'update_at', 'deleted_at',];

    protected $fillable = ['nome', 'descricao'];

}
```

Código 13 – *Model* de Compras.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
```

```
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\SoftDeletes;

class Compras extends Model
{
    use SoftDeletes,

    protected $table = 'compras';

    protected $dates = ['created_at', 'deleted_at'];

    protected $guarded = ['id', 'created_at', 'deleted_at',];

    protected $fillable = ['preco'];


    public function usuario(): BelongsTo
    {
        return $this->belongsTo(Usuario::class, 'id_funcionario');
    }

    public function produto(): HasMany
    {
        return $this->hasMany(Produto::class, 'id_produto');
    }

}
```

Código 14 – *Model* de Vendas.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\SoftDeletes;

class Vendas extends Model
{
    use SoftDeletes,

    protected $table = 'vendas';

    protected $dates = ['created_at', 'deleted_at'];

    protected $guarded = ['id', 'created_at', 'deleted_at',];

    protected $fillable = ['preco'];


    public function usuario(): BelongsTo
    {
        return $this->belongsTo(Usuario::class, 'id_funcionario');
    }

    public function produto(): HasMany
    {
```

```
        return $this->hasMany(Produto::class, 'id_produto');
    }

}
```

4.2.3 Testes de Unitário

Por fim, houve a necessidade de desenvolver alguns testes de integração para garantir o funcionamento do *dashboard* em especial com os acessos ao sistema, para sinalizar a segurança do sistema e detectar *bugs* em relação as funcionalidade de *login* e *logout* . Foram desenvolvidos os testes para as *queries* e as *migrations*, onde as requisições são feitas e é verificado se o resultado é compatível com o objeto esperado. No código 14 é exemplificado um teste para validar o acesso ao sistema.

Código 15 – Teste do *login*.

```
<?php

namespace Symfony\Component\Security\Core\Tests\User;

use Symfony\Component\Security\Core\User\User;

class UserTest extends \PHPUnit_Framework_TestCase
{

    public function testConstructorException()
    {
        new User('', 'superpass');
    }


    public function testGetRoles()
    {
        $user = new User('admin', 'superpass');
        $this->assertEquals(array(), $user->getRoles());

        $user = new User('admin', 'superpass', array('ROLE_ADMIN'));
        $this->assertEquals(array('ROLE_ADMIN'), $user->getRoles());
    }


    public function testGetPassword()
    {
        $user = new User('admin', 'superpass');
        $this->assertEquals('superpass', $user->getPassword());
    }


    public function testGetUsername()
    {
        $user = new User('admin', 'superpass');
        $this->assertEquals('admin', $user->getUsername());
    }
```

```
    public function testGetSalt()
    {
        $user = new User('admin', 'superpass');
        $this->assertEquals('', $user->getSalt());
    }


    public function testIsAccountNonExpired()
    {
        $user = new User('admin', 'superpass');
        $this->assertTrue($user->isAccountNonExpired());

        $user = new User('admin', 'superpass', array(), true, false);
        $this->assertFalse($user->isAccountNonExpired());
    }


    public function testIsCredentialsNonExpired()
    {
        $user = new User('admin', 'superpass');
        $this->assertTrue($user->isCredentialsNonExpired());

        $user = new User('admin', 'superpass', array(), true, true, false);
        $this->assertFalse($user->isCredentialsNonExpired());
    }


    public function testIsAccountNonLocked()
    {
        $user = new User('admin', 'superpass');
        $this->assertTrue($user->isAccountNonLocked());

        $user = new User('admin', 'superpass', array(), true, true, true, false);
        $this->assertFalse($user->isAccountNonLocked());
    }


    public function testIsEnabled()
    {
        $user = new User('admin', 'superpass');
        $this->assertTrue($user->isEnabled());

        $user = new User('admin', 'superpass', array(), false);
        $this->assertFalse($user->isEnabled());
    }


    public function testEraseCredentials()
    {
        $user = new User('admin', 'superpass');
        $user->eraseCredentials();
        $this->assertEquals('superpass', $user->getPassword());
    }
}
```

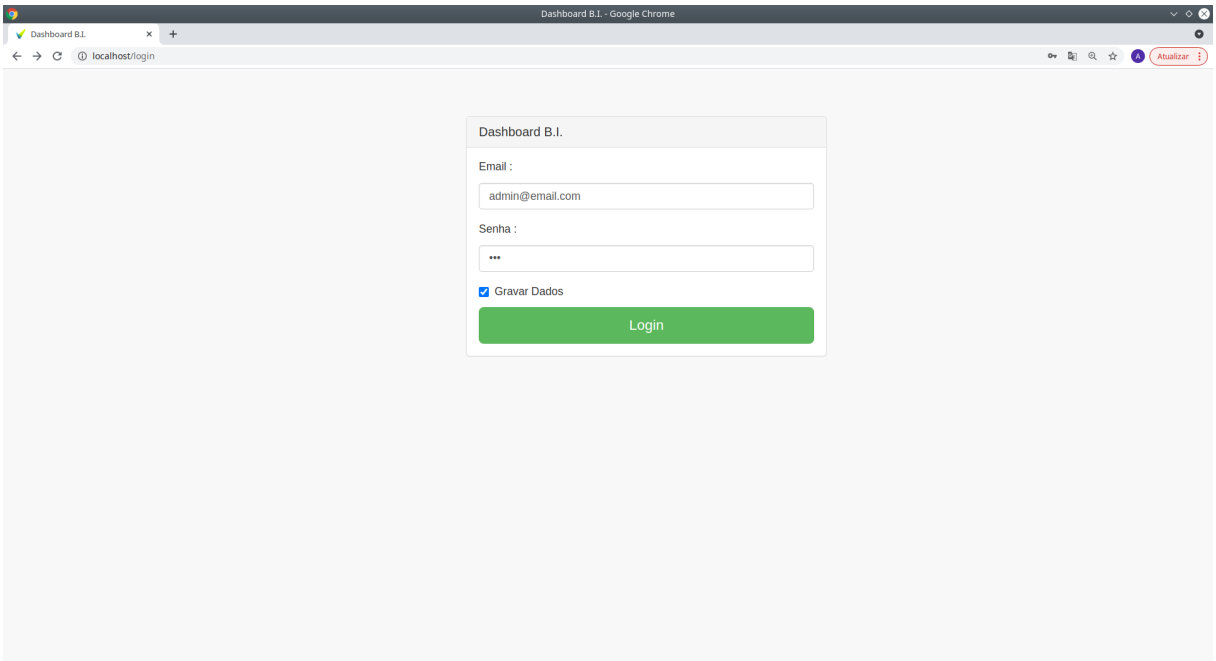
4.2.4 Desenvolvimento do aplicativo

4.2.4.1 *Dashboard*

A construção das páginas seguiu o fluxo de interação do usuário dentro do sistema gerado para a implementação do algoritmo de B.I. A seguir encontram-se as telas feitas com suas respectivas características:

- Tela de *login*: formulário de *login* e botão para realizar *login*. Apresentado na Figura 8;
- Tela principal: título do sistema, *cards* com os produtos, menu lateral e caixa de texto para realizar buscas nos cards. Apresentado na Figura 9;
- Tela Usuarios, Funcionarios, Produtos, Compras e Vendas: título da tela, formulário de criação e tabela de consulta menu lateral. Apresentado na Figura 10, Figura 11, Figura 12, Figura 13 e Figura 14 respectivamente;
- Tela Seviços B.I. : titulo do produto analisado, gráfico de previsão mostrando a *moda* do menor e do maior valor obtido em cada mês dos anos anteriores, duas tabelas que descrevem os resultados dos meses separados de Estáveis, com maior precisão na analise, e Instáveis, com menor precisão na analise . Apresentado na Figura 15 e Figura 16, que apresentam como exemplo a tela que analisa o produto A;

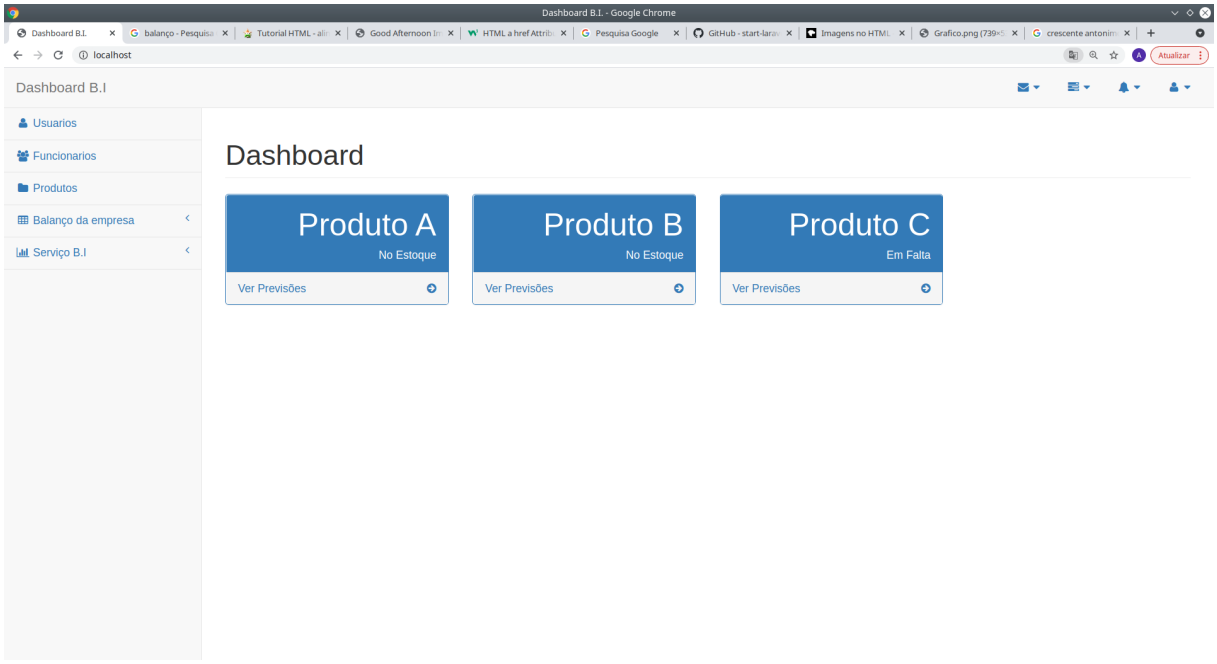
Figura 8 – Tela de login



Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a tela inicial do sistema, contendo um *template* de uma tela de *login* simples, com a funcionalidade de login automático nas próximas vezes que o usuário utilizar o sistema.

Figura 9 – Tela principal

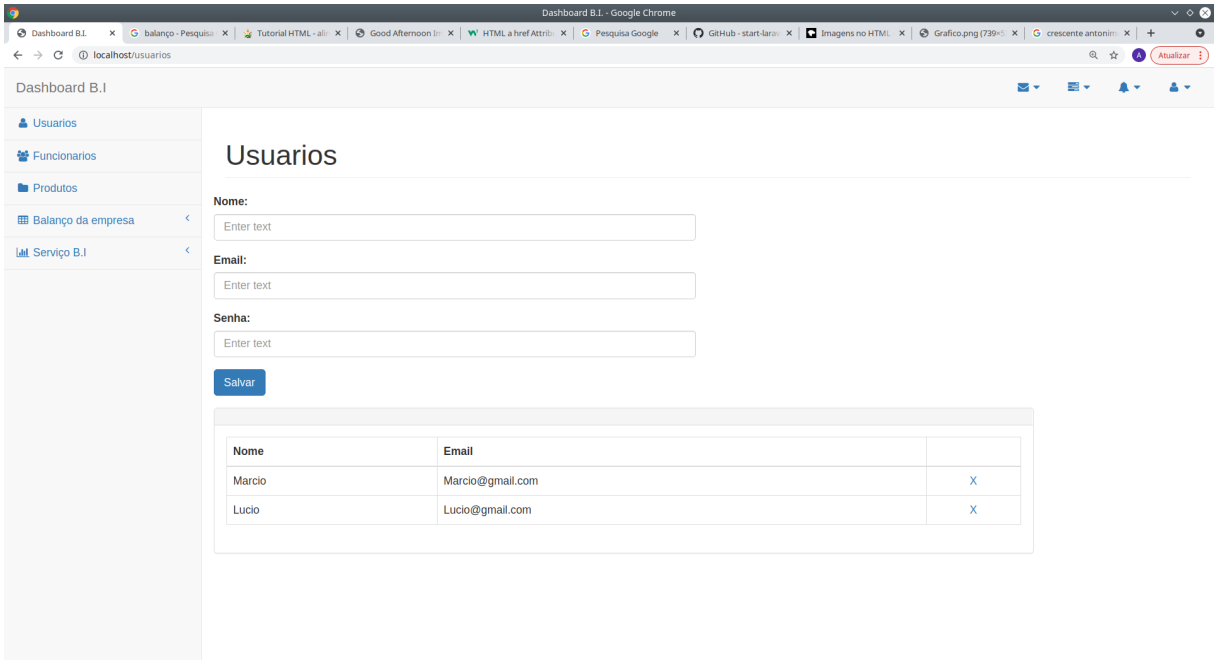


Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a tela principal do sistema, contendo *cards* representando cada item registrado, uma analise de itens em falta ou em estoque (através de uma diferença entre compras e vendas menor ou igual a zero) e um atalho pra visualizar as previsões para o próximo ano de cada item registrado.

Vale ressaltar também o menu no cabeçalho que apenas tem a funcionalidade *logout* desenvolvida e o menu lateral responsável pela navegação no sistema, sendo estes dois menus comuns para as paginas apresentada a seguir.

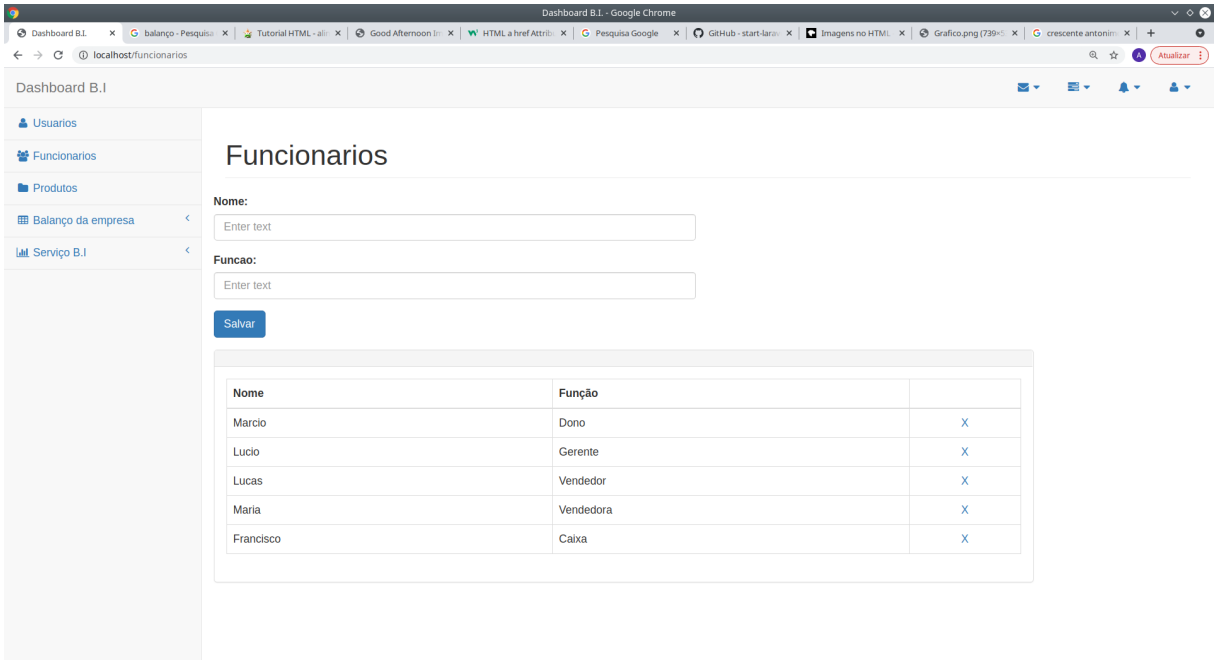
Figura 10 – Tela de usuários



Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a tela de cadastro de usuários formada por um *form* realizando o CRUD e uma tabela que apresenta os usuários inscritos, inicialmente sem distinção de usuário administrador e usuário comum.

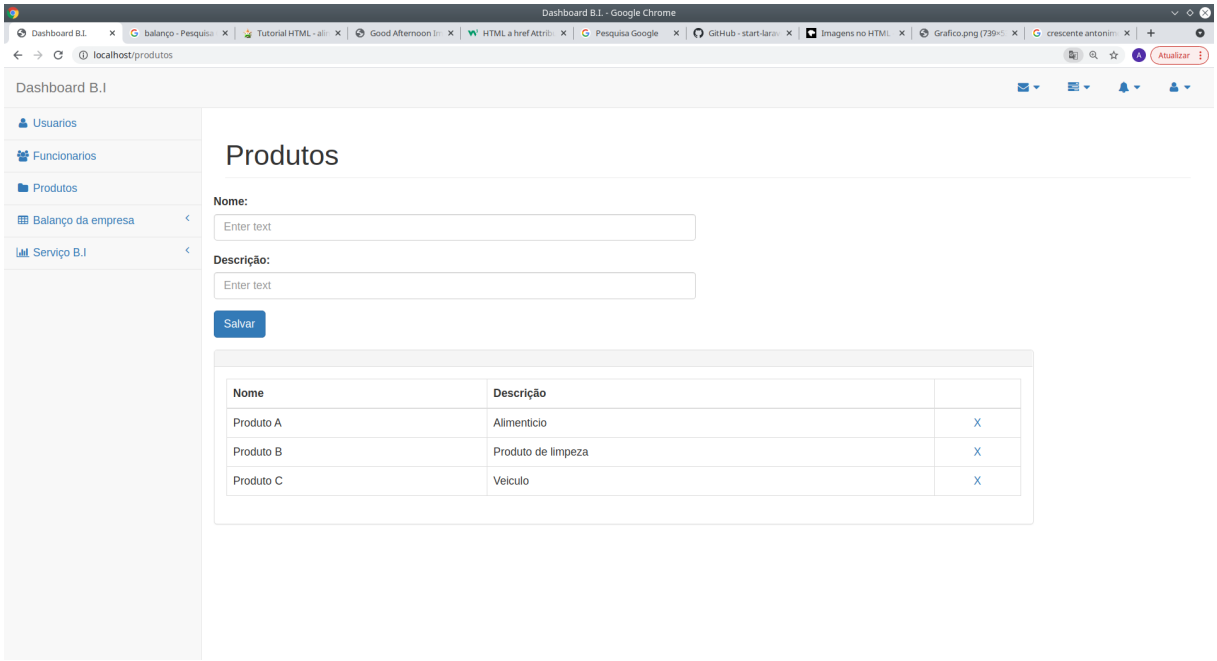
Figura 11 – Tela de funcionários



Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a tela de cadastro de funcionários formada por um *form* realizando o CRUD e uma tabela que apresenta os funcionários responsáveis por realizarem as compras e vendas dos produtos.

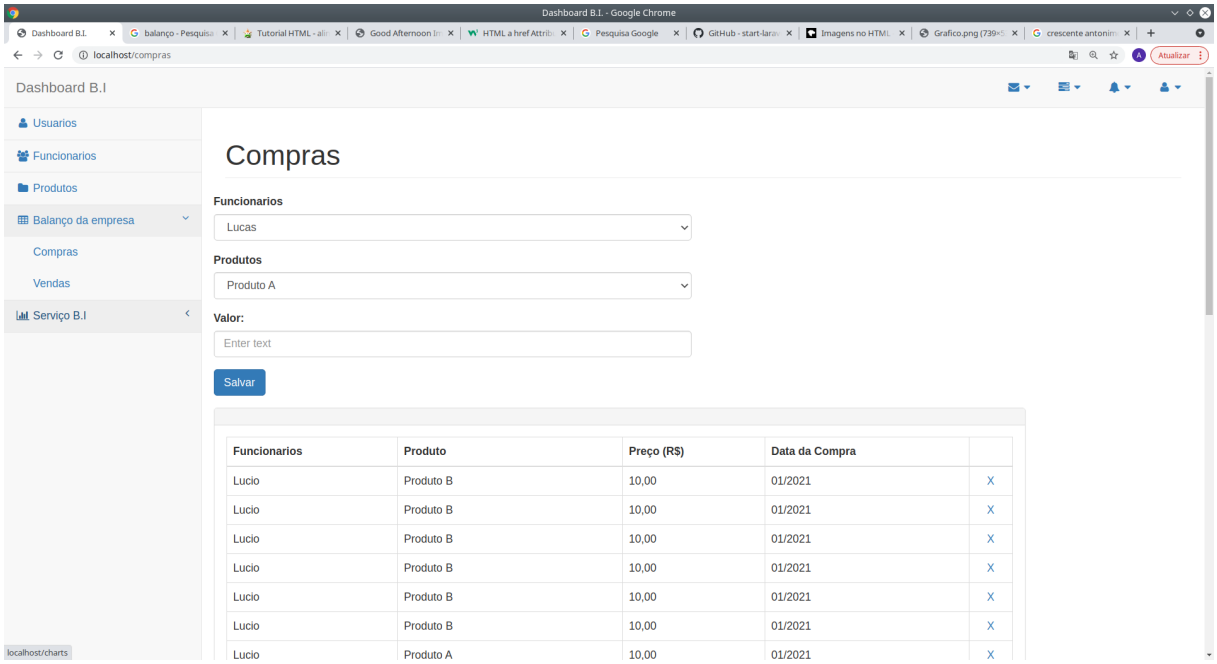
Figura 12 – Tela de produtos



Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a tela de cadastro de produtos formada por um *form* realizando o CRUD e uma tabela que apresenta os produtos que a empresa compra e vende no negocio.

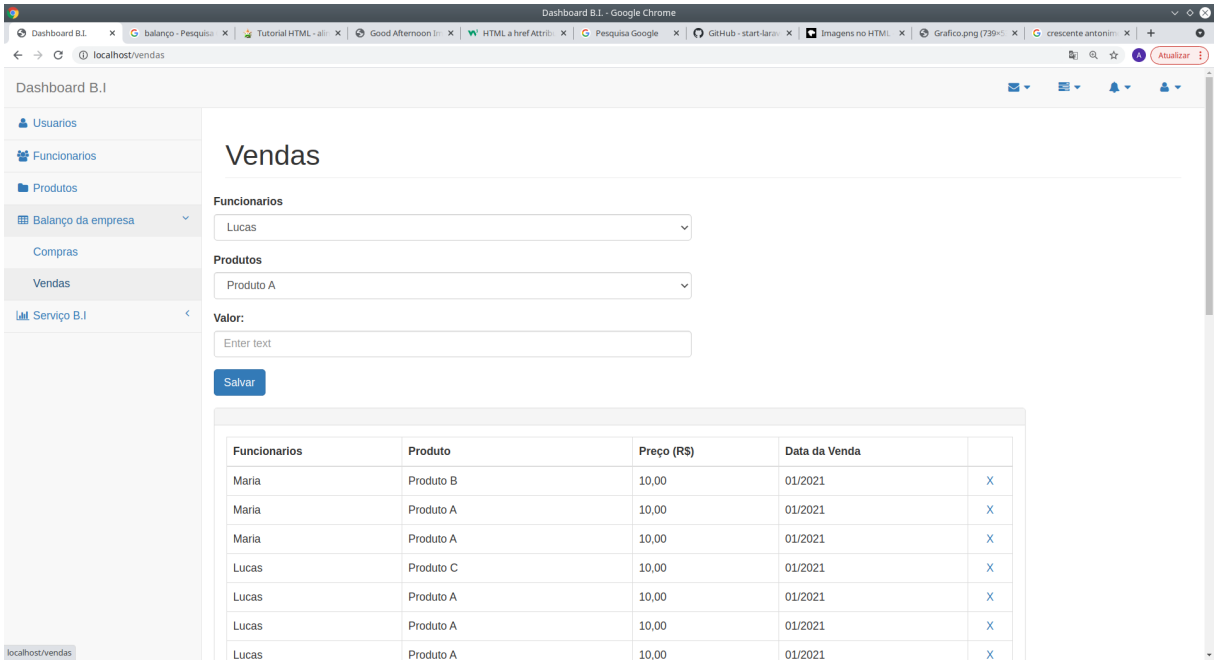
Figura 13 – Tela de compras



Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a tela de cadastro de compras formada por um *form* realizando o CRUD e uma tabela que apresenta as compras dos produtos já registrados.

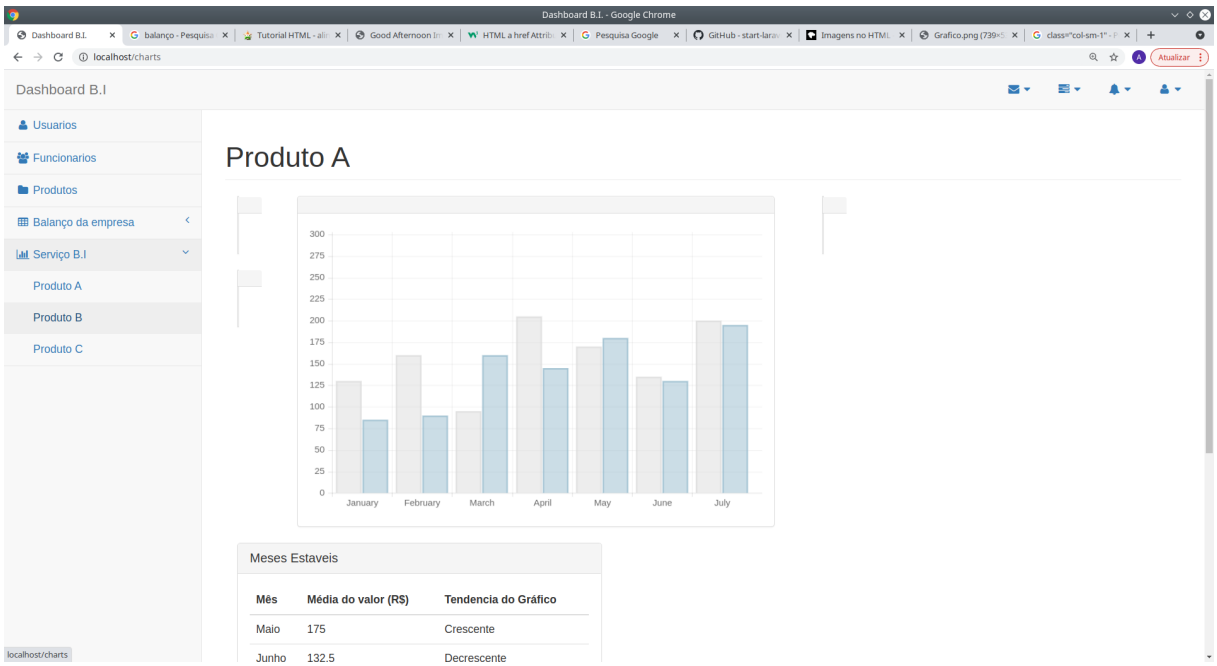
Figura 14 – Tela de vendas



Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a tela de cadastro de compras formada por um *form* realizando o CRUD e uma tabela que apresenta as vendas dos produtos já registrados.

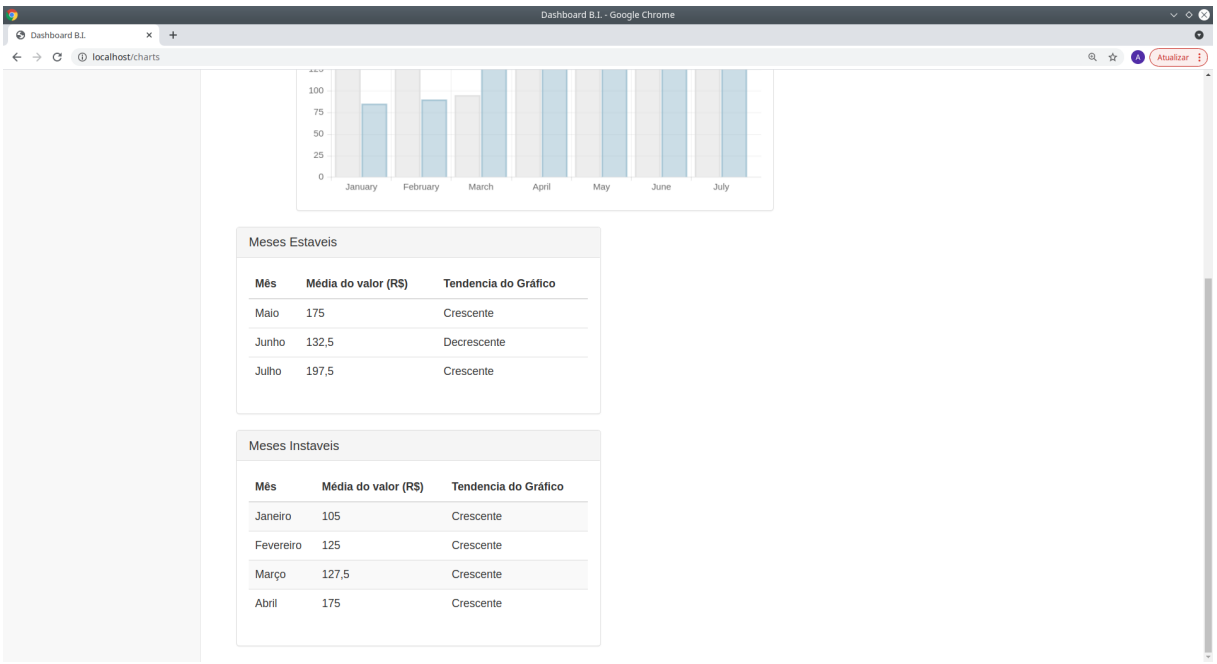
Figura 15 – Tela do Serviço B.I.(gráfico)



Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a tela que representa um gráfico com a previsão de lucros do produto A para o próximo ano, sendo gerado por meio de um cálculo realizado com as vendas realizadas em um mês especifico em comparação com o mesmo mês em anos anteriores.

Figura 16 – Tela do Serviço B.I.(tabela)



Fonte: Elaborado pelo autor.

Nessa imagem é apresentada a mesma tela anterior com o foco em duas tabelas separando os meses em meses convergentes, onde houve uma maior concentração de lucro convergindo para um intervalo em que a diferença entre o maior e o menor valor nesse intervalo seja menor ou igual a 25 reais, os demais são inseridos na tabela de meses divergentes.
Cada tabela contem o mês especifico, média entre os dois valores calculados que constam no gráfico pertinente aquele mês e se a tendência do mês é de queda(decrescente) ou de alta (crescente).

5 Conclusão

O mercado atual se encontra carente de soluções tecnológicas para auxiliar o micro e pequeno empreendedor, pesquisas mostram que mais da metade das empresas não conseguem completar um ano de funcionamento e logo já declaram falência, tendo como motivo a insipiência administrativa e a falta de capital para investir no negócio até ele começar a dar lucros sólidos. Mesmo assim o número de empreendedores cresce exponencialmente no Brasil, muitas vezes motivados por carência de empregos e financeira, o que irá acarretar futuramente uma nova safra de ex-empreendedores endividados caso iniciativas de apoio, como a do Sebrae, não sejam acessíveis e ampliadas.

É papel da ciência a constante produção científica voltada para as necessidades da sociedade, realizando essa integração entre o mundo acadêmico e o mundo externo a academia. Principalmente no momento atual, nunca foi tão necessária a valorização da produção científica, de forma que sua acessibilidade à população, contribua para que esta seja reconhecida por sua devida importância.

O presente projeto busca integrar conceitos de *business Intelligence* com a realização de tecnologias administrativas simples de serem implementadas em ambiente com pouco investimento tecnológico, sendo inicialmente um sistema integrado a um *dashboard* simples, afim de auxiliar no planejamento de micro e pequenos empreendedores.

Neste trabalho foi criada uma aplicação capaz de ser executada em sistemas básicos, utilizando de conceitos básicos de estatística e presentes no contexto do mercado de ações. Utilizando-se principalmente o Laravel para a criação do sistema *dashboard*, Python para o algoritmo de análise de dados de maneira simplificada e otimizada e o Docker para modelar um ambiente compatível com o ambiente que está alocado, foi possível obter um sistema que execute e praticamente qualquer máquina.

Portanto, a utilização dessas tecnologias no trabalho, mostram como a utilização de conceitos e paradigmas atuais possam ser usadas para agregar uma porcentagem da população que ainda não compartilham tecnologias atuais. Afim de proporcionar mais igualdade digital, nesse caso para os micro e pequenos empreendimentos e negócios familiares.

6 Próximos Passos

Como próximos passos para o projeto, é necessário a modularização do algoritmo de B.I., como a utilização de serviços e integrações separados do *dashboard* e transformando o algoritmo em um sistema independente e acoplável a qualquer outro sistema administrativo.

Para isso, seria necessária é implementar algo parecido com a arquitetura que conflitou com o presente trabalho, utilizando-se de uma API que seja alimentada com os dados do sistema usado pelo usuário e o desenvolvimento de um *front-end* aparte para apresentar os resultados e minimizar os conflitos com outros sistemas. Sendo possível a utilização de mais artifícios para aumentar a qualidade do algoritmo, que não foram utilizados, deixando-o mais robusto e otimizado.

Referências

ALVES, C. *Busines Intelligence: tudo que você precisa saber*. 2018. Disponível em: <https://blog.bi9.com.br/business-intelligence/#:~:text=O%20termo%20Business%20Intelligence%20>. Acesso em: 31 de maio de 2021.

BITBUCKET, A. *Fluxo de trabalho de Gitflow*. 2020. Disponível em: <https://www.atlassian.com/br/git/tutorials/comparing-workflows/gitflow-workflow#:~:text=Ele%20foi%20publicado%20pela%20primeira,robusta%20para%20gerenciar%20projetos%20maiores.> Acesso em: 29 de maio de 2020.

DOCKER. *Documentação Docker*. 2020. Disponível em: <https://docs.docker.com/>. Acesso em: 23 de outubro de 2020.

FLORES., J. de S. *Características das ferramentas de Business Intelligence que contribuem para obtenção dos objetivos estratégicos à luz dos princípios de governança corporativa*. 2011. Disponível em: <http://www.repositorio.jesuita.org.br/handle/UNISINOS/5016#:~:text=A%20an%C3%A1lise%20dos%20dados%20resultou,os%20Princ%C3%ADpios%20de%20Governan%C3%A7a%20Corporativa.> Acesso em: 01 de junho de 2021.

GIT. *Documentação Git*. 2020. Disponível em: <https://www.git-scm.com/>. Acesso em: 23 de outubro de 2020.

LARAVEL. *Documentação Laravel*. 2020. Disponível em: <https://laravel.com/docs/6.x>. Acesso em: 23 de outubro de 2020.

MEDVIDOVIC, N.; TAYLOR, R. Software architecture: foundations, theory, and practice. In: *ACM/IEEE International Conference on Software Engineering*. Cape Town, South Africa: ICSE, 2010.

MORAES, D. *Business Intelligence: o que é e como fazer análise de dados de inteligência empresarial?* 2018. Disponível em: <https://inteligencia.rockcontent.com/business-intelligence//>. Acesso em: 27 de setembro de 2020.

ORACLE. *Banco de dados definido*. 2021. Disponível em: <https://www.oracle.com/br/database/what-is-database/>. Acesso em: 29 de maio de 2021.

PHP. *Documentação PHP*. 2020. Disponível em: <https://www.php.net/manual/pt_BR/intro-whatis.php>. Acesso em: 26 de outubro de 2020.

PHPMYADMIN. *Bringing MySQL to the web*. 2003.

POP, D. P.; ALTAR, A. Designing an mvc model for rapid web application development. *Procedia Engineering*, v. 69, p. 1172–1179, 2014.

SEBRAE. *O impacto da pandemia de coronavírus nos pequenos negócios*. 2021. Disponível em: <https://www.sebrae.com.br/sites/PortalSebrae/artigos/o-impacto-da-pandemia-de-coronavirus-nos-pequenos-negocios,192da538c1be1710VgnVCM1000004c00210aRCRD>. Acesso em: 03 de novembro de 2020.

SOUZA, I. de. *Framework: descubra o que é, para que serve e por que você precisa de um para o seu site*. 2019. Disponível em: <https://rockcontent.com/br/blog/framework/#:~:text=O%20framework%20%C3%A9%20um%20pacote,qualidade%20no%20projeto%20e%20produtividade.> Acesso em 1 de setembro de 2020.

WDN-WEB-DOCS. *JavaScript Tutoriais*. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 08 de julho de 2021.

XERPAY. *Saiba quais são as principais causas de fechamento de uma empresa*. 2019. Disponível em: <https://xerpay.com.br/blog/quais-sao-as-principais-causas-de-fechamento-de-uma-empresa/>. Acesso em: 01 de maio de 2021.