



BreShow: Um Microsistema de ERP de Código Aberto para Pequenas e Médias Empresas do Setor de Brechó

Venâncio Pessoa Igrejas Lopes Neto

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Claudio Miceli de Farias

Rio de Janeiro

Maio de 2022

BreShow: Um Microsistema de ERP de Código Aberto para
Pequenas e Médias Empresas do Setor de Brechó

Venâncio Pessoa Igrejas Lopes Neto

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO
DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA POLITÉCNICA
DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS
REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGEN-
HEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:

Venâncio Pessoa Igrejas Lopes Neto

Orientador:

Prof. Claudio Miceli de Farias, D. Sc.

Examinador:

Prof. Flávio Luis de Mello, D. Sc.

Examinador:

Prof. Toacy Cavalcante de Oliveira, D. E.

Rio de Janeiro

Maio de 2022

Declaração de Autoria e de Direitos

Eu, *Venâncio Pessoa Igrejas Lopes Neto* CPF 145.689.997-07, autor da monografia *BreShow: Um Microsistema de ERP de Código Aberto para Pequenas e Médias Empresas do Setor de Brechó*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetua-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e ideias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.

Venâncio Pessoa Igrejas Lopes Neto

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Escola Politécnica - Departamento de Eletrônica e de Computação

Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária

Rio de Janeiro - RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmear ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es).

DEDICATÓRIA

Dedico este trabalho aos meus pais, ao meu orientador e toda população brasileira. Aos meus pais por toda a educação e ensinamentos impecáveis que me deram e dão até hoje, a população brasileira por me proporcionar a possibilidade de fazer um curso de nível superior de alta qualidade e ter investido em mim e nos meus esforços na minha vida acadêmica. Por ultimo e não menos importante, ao meu orientador que sem ele não teria conseguido concluir esta difícil tarefa.

AGRADECIMENTOS

Por mais que este trabalho tenha sido feito por uma pessoa, não teria conseguido concluir sem a ajuda de inúmeras pessoas que me deram apoio ao longo de toda a minha vida. Tentarei mencionar todos nessa sessão sabendo que irei falhar, pois são muitos. Dessa forma, já agradeço de antemão a quem não mencionei aqui por tudo que fez por mim. Saibam que se eu cheguei aqui, é porque vocês tiveram um papel fundamental.

Primeiramente, gostaria de agradecer aos meus pais Jussara e Adalberto, minhas irmãs Júlia e Luísa e minha namorada Larissa por tudo que fizeram por mim todos esses anos e pelo apoio de vocês na minha vida, que tem sido de extrema importância para ter chegado onde cheguei.

Aos meus professores por todo carinho, dedicação e paciência que tiveram ao me ensinar. Em especial, ao professor José Guilherme que me ensinou a maneira correta de se estudar no ensino médio, a professora de francês Ana Sílvia pelo aprendizado e conselhos, aos professores do Colégio Ponto de Ensino e do preparatório militar que me deram uma ótima base para a faculdade e aos professores universitários da UFRJ pelo conhecimento transmitido.

Agradeço as pessoas que influenciaram diretamente, ou indiretamente, nas minhas iniciações científicas tanto no laboratório do Grupo de Telecomunicações e Automação (GTA), como no Laboratório de Processamento de Sinais (LPS). Primeiramente, agradeço ao professor Otto Carlos Muniz Bandeira Duarte, que infelizmente nos deixou recentemente, e ao professor José Manoel de Seixas, os quais tive o imenso prazer de tê-los como orientadores durante o período de iniciação científica. Agradeço ao Lucas Airam e Lucas Santiago pelo apoio e por tudo que passamos no GTA e também ao Nathanael Moura Júnior, Gustavo Augusto Mascarenhas Goltz, Pedro Lisboa e Vinícius Mello pela força e ajuda nos conhecimentos práticos e teóricos de aprendizagem de máquina no LPS.

Ao meu orientador, Cláudio Miceli de Farias que foi presente e solícito na redação deste trabalho, assim como nas nossas reuniões sobre o mesmo.

Por ultimo mas não menos importante, gostaria de agradecer aos meus amigos e colegas de curso, sem eles não estaria me formando agora e não teria superado os momentos mais difíceis e complicados da vida acadêmica. Em especial, Lucas Fontes, Yuri Pereira, Daniel Porto, Camyla Romão, Letícia Ecard, Andrews Monzato, Brenno Rodrigues, Leonardo Barreto e Lucas Lessa pelo apoio, estudos em grupo e inúmeros trabalhos que fizemos ao longo dos períodos.

RESUMO

Com a popularização da internet e serviços a partir dela, o mercado de ERPs se modernizou em poucos anos e se tornou mais acessível, entretanto, custo e complexidade do sistema ainda podem ser um obstáculo para algumas categorias de empreendedores. Dessa forma, esse trabalho tem por objetivo o desenvolvimento de um ERP voltado para a análise de vendas onde, inicialmente, é focado para o setor de brechós com o intuito de auxiliar o empreendedor nas suas tomadas de decisões. O projeto é de código aberto e de fácil customização, onde em uma tela é feita o cadastro das vendas pelo usuário e da outra, é gerado a análise macro das informações das vendas filtradas por um período de data. A sua arquitetura e sistemas utilizados na integração são populares no mercado e também de código aberto, tornando-o de baixo custo e assim, viável para pequenos e médios empreendedores.

Palavras-Chave: ERP, Brechó, Aplicação Web, Arquitetura de Software, Docker, Grafana, Kecloak, NestJs, Angular

ABSTRACT

With the popularization of the internet and services from it, the market is becoming more and more competitive. Thus, this work aims to develop a web system aimed at sales analysis where, initially, it is aimed at the thrift store sector in order to assist the entrepreneur in its decision-making. The project is open source and easy to customize, where the user registers sales on one screen, and on the other, it is a macro for analyzing sales information filtered by a period of data. Its architecture and systems used in the integration are popular and also open source, making it low cost and thus viable for small and enterprising entrepreneurs.

Keywords: ERP, Thrift shop, Web application, Software architecture, Docker, Grafana, Kecloak, NestJs, Angular

Índice

Índice de Figuras	xi
1 Introdução	1
1.1 Motivação	2
1.2 Problema	3
1.3 Contribuição	5
1.4 Organização do Texto	5
2 Conceitos Básicos	7
2.1 Princípios SOLID	7
2.2 Arquitetura MVC	8
2.3 Clean Architecture	9
2.4 Folder-by-Feature	11
2.5 ERP	12
3 Trabalhos Relacionados	13
4 Proposta	17
4.1 Arquitetura de software	17
4.2 Módulos	18
4.2.1 Aplicação Web	19
4.2.2 Dashboard	21
4.2.3 Gerenciamento de Usuários	22
5 Implementação	26
5.1 Arquitetura	26
5.1.1 Orquestrador	28

5.1.2	Banco de dados	29
5.2	Aplicações	31
5.2.1	Gerenciador de usuários	31
5.2.2	Aplicação Web	34
5.2.3	Análise dos dados	40
5.3	Desafios	43
6	Prova de Conceito	47
6.1	Implantação	47
6.1.1	Docker	47
6.1.2	Github	47
6.1.3	Npm	48
6.1.4	Projeto Local	48
6.2	Usuário	49
6.2.1	Criação de Conta	49
6.2.2	Tela principal	49
6.2.3	Tela de Categoria	50
6.2.4	Tela de Fornecedor	53
6.2.5	Tela de Produtos	54
6.2.6	Tela de Dashboard	57
6.3	Administrador	60
6.3.1	Tela de Gerenciamento de Usuários	60
6.3.2	Tela de Dashboard	62
7	Conclusões e Trabalhos Futuros	64
7.1	Conclusão	64
7.2	Trabalhos Futuros	65
	Bibliography	66

Índice de Figuras

1.1	Exemplo do marketing da empresa TOTVS sobre o seu serviço de ERP	3
1.2	Exemplo dos valores do plano da empresa Bling sobre a sua plataforma	4
1.3	Exemplo dos valores dos micro serviços proporcionado pela Odoo . . .	4
2.1	modelo Clean Architecture desenvolvido por Uncle Bob.[1]	10
3.1	Quadro comparativo entre o Odoo(OpenERP) e Xtuple.[2]	14
3.2	Quadro comparativo entre o Odoo, MyCollab, OrangeScrum e Open Project.[3]	16
4.1	Diagrama de Venn dos Módulos do projeto	19
4.2	Fluxograma da tela de produtos	20
4.3	Fluxograma da tela de fornecedor	21
4.4	Fluxograma da tela de categorias	22
4.5	Caso de uso do módulo de dashboard	23
4.6	Lógica do login e aquisição dos tokens para a aplicação	24
5.1	Arquitetura macro do projeto separada por contêiner	27
5.2	Diagrama entidade relacionamento do banco de dados da aplicação .	30
5.3	Tela do cliets do realm BreshowAD do Keycloak	33
5.4	Estrutura de pastas do frontend da aplicação web	36
5.5	Estrutura da pasta “Core” do frontend da aplicação web	37
5.6	Arquivos relacionados as configurações do Keycloak no backend . . .	38
5.7	Estrutura da entidade de categoria do servidor	39
5.8	Código da regra de negócio de atualização de uma entidade de venda que está no arquivo “product.service.ts”	40
5.9	Painel de valores do dashboard de um usuário no Grafana	42

5.10	Painel de valores em barra do dashboard de um usuário no Grafana .	43
5.11	Painel de valores em estatística do dashboard de um usuário no Grafana	44
5.12	Painel de valores por fornecedor do dashboard de um usuário no Grafana	45
5.13	Dashboard completo das vendas de um usuário no Grafana	45
5.14	Aba de configuração de um dashboard no Grafana	46
6.1	Página para entrar na aplicação web.	50
6.2	Página de cadastro da aplicação web.	51
6.3	Página principal da aplicação web.	52
6.4	Página da categoria da aplicação web.	52
6.5	Janela flutuante da categoria.	53
6.6	Tabela da categoria listando a Blusa após ter sido criada.	53
6.7	Página dos fornecedores na aplicação web.	53
6.8	Janela flutuante da tela de fornecedores.	54
6.9	Tabela de fornecedores listando o novo fornecedor após ter sido criado.	55
6.10	Página dos produtos na aplicação web.	55
6.11	Janela flutuante da tela de produtos.	56
6.12	Tabela de produtos listando o novo produto após ter sido criado. . . .	56
6.13	Tela principal listando todas as informações adicionadas do usuário em tabelas distintas.	57
6.14	Tela de login do módulo de análise.	57
6.15	Tela principal do usuário.	58
6.16	Painel de listas gerais expandido.	58
6.17	Painel de fornecedores e categorias expandido.	59
6.18	Painel de fornecedores e categorias expandido.	59
6.19	Página Inicial do Keycloak.	60
6.20	Página de login da área de administração.	61
6.21	Página Principal do Keycloak.	61
6.22	Tela de gerenciamento do usuário Venâncio.	62
6.23	Página de credenciais do usuário Venâncio.	62
6.24	Área de gerenciamento de usuários do módulo de dashboard.	63
6.25	Área de configuração	63

Capítulo 1

Introdução

O Planejamento de Recursos Empresariais, conhecido pela sigla em inglês ERP, tem sido implementado a anos em empresas de grande e médio porte melhorando a integração dos sistemas de informação, dentre outros benefícios. Entretanto, a maioria dos médios e pequenos empreendedores consideram o custo como principal fator na implantação de um sistema ERP e mostram-se relutantes em investir em melhorias após certo tempo de uso. Além disso, alguns desses sistemas não atendem à essas categorias de empreendedores devido às suas complexidades e funcionalidades, que acabam não sendo utilizadas.

Dessa forma, o presente trabalho tem como propósito a criação de um sistema ERP a partir de um projeto web de análise de vendas para auxiliar o microempreendedor, ou empreendedor informal, a compreender melhor suas vendas ajudando-os a tomarem melhores decisões baseadas em estatísticas e gráficos apresentados pelo sistema. Nele, encontram-se micro serviços para gerenciamento de usuários, painéis de dados e telas de cadastros das informações referentes ao empreendimento do usufruidor. Uma vez que o projeto é de código aberto, o indivíduo não precisará investir em mensalidades ou contar com alguns dias para teste antes de precisar investir capital no programa, assim como são feitos atualmente em sistemas semelhantes no mercado. Basta baixar o código fonte, seguir o tutorial de instalação e rodar o sistema.

Ao longo do texto será explicitado sobre as funcionalidades, arquiteturas e teorias por trás da aplicação, tais como motivações, problemas e trabalhos futuros.

1.1 Motivação

Com o avanço da tecnologia, produtos e serviços que um dia eram de exclusividade de grandes empresas, atualmente encontram-se presentes no cotidiano de companhias menores. Banda larga, internet via Wi-fi, smartphones com poder de processamento maior que antigos computadores básicos e inteligência artificial(IA) em produção são alguns dos exemplos de tecnologias presentes que facilitam a forma de viver e trabalhar.

Todo esse avanço obrigou as empresas a se tornarem mais complexas e desenvolvidas, afinal, os sistemas de informação se tornaram mais eficientes, reduziram seus custos e aperfeiçoaram suas logísticas, impactando diretamente na competitividade do mercado. Um exemplo de sistema de informação que pode ajudar as companhias a aprimorarem a eficiência de suas performances seria o ERP, visto que esse sistema pode integrar informações de vários segmentos da companhia, como financeiro, time, tempo, materiais, capacidade, entre outros [4].

Os ERPs são ferramentas estratégicas que sincronizam e integram sistemas de uma empresa para gerenciar de forma automática recursos, estoques, setor financeiro e informações de venda, agilizando a resposta e entrega de seus serviços aos clientes, procurando minimizar qualquer barreira entre os múltiplos setores da companhia. Com a melhora na eficiência da internet e a popularização da computação em nuvem, empresas desenvolveram ERPs como Plataforma Como Serviço (PaaS)[5] para disponibilizá-los como Software Como Serviço (SaaS)[6] para seus clientes e, com isso, não havendo a necessidade de instalar hardwares físicos na companhia do cliente e nem em suas próprias localidades, dado que, em sua grande maioria, os servidores são hospedados na nuvem, tornando o preço mais competitivo no mercado. Na Figura 1.1, é explicitado um exemplo de marketing da Empresa TOTVS vendendo um SaaS de ERP.

Em virtude dos fatores supracitados, torna-se importante o uso desses sistemas para todo tipo de empreendedor, ou companhia, por conta da visibilidade que os gestores terão dos setores da empresa e desta como um todo para melhorar as tomadas de decisões e, conseqüentemente, um controle maior da produção.

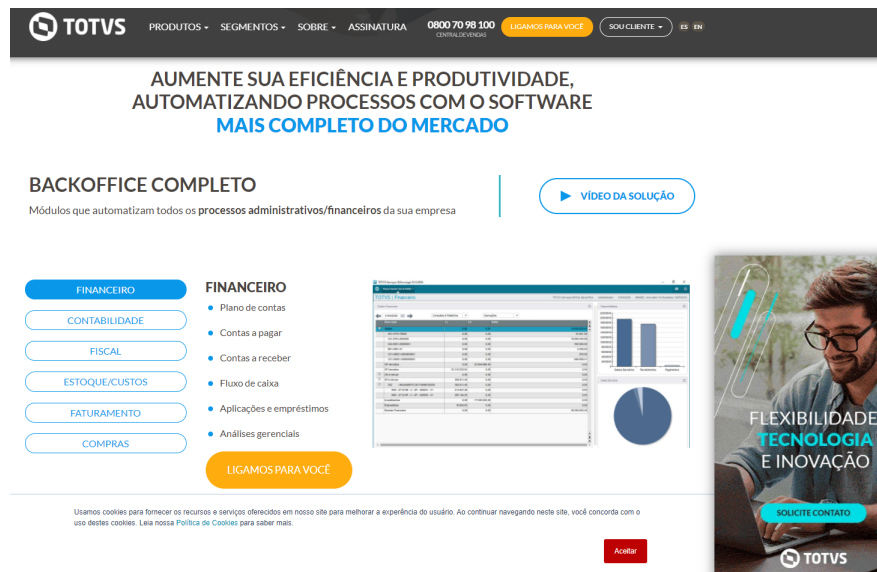


Figura 1.1: Exemplo do marketing da empresa TOTVS sobre o seu serviço de ERP

1.2 Problema

O ERP se tornou uma ferramenta importante e fundamental no mercado, todavia, empresas que disponibilizam esse serviço para seus clientes cobram valores que, em determinadas situações, microempreendedores não estão dispostos ou não possuem condições de arcar. O valor e formato de cobrança varia entre as empresas, como exemplos disso temos a TOTVS com uma central de vendas que entra em contato com o cliente, a Senior que envia uma proposta através do e-mail, e a Bling, empresa famosa no ramo e com grandes clientes, que apresenta planos que começam com o valor de 25,00 reais e vão até 100,00 reais por mês de acordo com a necessidade do cliente e existe um período gratuito para testar a plataforma, conforme observado na figura 1.2.

Outro ponto a ser levantado é o fato do cliente pagar o valor fixo de todo o pacote de benefícios e não apenas o que ele irá utilizar. Existem poucos serviços de ERPs que disponibilizam micro serviços a serem pagos separadamente, assim o usuário escolhe quais destes deseja para sua empresa. Um dos exemplos disso é a empresa Odoo que faz esse tipo de transação, conforme explicitado na figura 1.3, além de cobrar por usuário e ser em dólar. A cobrança pelo pacote inteiro se torna ineficiente para o cliente, pois entende-se que serão utilizadas todas as funcionalidades e sistemas do software vendido. Por outro lado, a cobrança por

Funcionalidades	Plano 15 usuários	Plano 10 usuários	Plano 5 usuários	Plano 2 usuários
Finanças	✓	✓	✓	✓
Boleto Registrado	✓	✓	✓	✓
Conta Digital	✓	✓	✓	✓
Indústria	✓	✓	✗	✗
Recursos	✓	✓	✓	✓
Suporte	✓	✓	✓	✓
Arquivo auxiliar do SPED	✓	✗	✗	✗
Espaço	180MB	120MB	60MB	20MB
Usuários	15 usuários	10 usuários	5 usuários	2 usuários
	R\$ 100,00 Por mês	R\$ 75,00 Por mês	R\$ 50,00 Por mês	R\$ 25,00 Por mês
	teste grátis	teste grátis	teste grátis	teste grátis

Figura 1.2: Exemplo dos valores do plano da empresa Bling sobre a sua plataforma

microserviço pode encarecer o produto final dependendo do tamanho da empresa e do número de funcionários. A solução seria a avaliação de várias plataformas no mercado para encontrar uma que seja mais adequada a realidade individual das empresas. Contudo, continuam os problemas de ser necessário investir um valor na plataforma e ter avaliação gratuita temporária pelo uso da plataforma.

Preços Odoo

Escolha o número de Usuários

1 Usuários \$6.00 USD \$6.00 USD/usuário/mês

Escolha seus Aplicativos

- CRM \$8.00 USD / mês
- Website \$8.00 USD / mês
- Contabilidade \$8.00 USD / mês
- Fabricação \$16.00 USD / mês
- Marketing por email \$4.00 USD / mês
- Faturamento \$4.00 USD / mês
- Comércio Eletrônico \$4.00 USD / mês
- Projeto \$8.00 USD / mês
- Compras \$4.00 USD / mês
- Despesas \$4.00 USD / mês
- Vendas \$4.00 USD / mês
- Ponto de Venda \$8.00 USD / mês
- Inventário \$12.00 USD / mês
- Planilhas de horários \$4.00 USD / mês
- Eventos \$4.00 USD / mês

Resumo:

- 1 Usuários \$8.00 USD
- Desconto de usuário ⁽¹⁾ -\$2.00 USD
- 0 Aplicativos \$0.00 USD
- Total / mês ⁽²⁾ \$6.00 USD

⁽¹⁾ Cobrado anualmente: \$72.00 USD

EXPERIMENTE AGORA
15 dias teste grátis

COMPRE AGORA

⁽¹⁾ Novos clientes obtêm um desconto no número inicial de usuários adquiridos. (\$6.00 USD em vez de \$8.00 USD).

Figura 1.3: Exemplo dos valores dos micro serviços proporcionado pela Odoo

Ainda na atualidade, existem microempreendedores e empreendedores informais que utilizam ferramentas básicas para gerenciar seus negócios, a exemplo do Excel, esta sendo uma ferramenta de tabelas da Microsoft, ou cadernos pautados usados para anotações. Essas ferramentas auxiliam no dia a dia, porém, para fazer uma

análise a longo prazo das estatísticas do negócio ou quais decisões tomar, elas não são tão úteis e requerem um esforço maior para analisar todos os dados presentes.

1.3 Contribuição

A contribuição deste trabalho consiste no desenvolvimento de um projeto acerca de um modelo de ERP que seja de Código Aberto, disponível gratuitamente, modularizável e escalável. Por conta disso, foram utilizadas linguagens e sistemas de repositório aberto, tais como Keycloak, Composição de Docker, NestJs, Angular, Grafana e Postgres. Além disso, contou-se com o uso de teorias de boas práticas de desenvolvimento de software, como os princípios SOLID, Clean Architecture, arquitetura MVC e estrutura Folder-by-Feature para, dentre outras vantagens, facilitar a sua manutenção.

O projeto consiste apenas no módulo de vendas do ERP, significando que contém um micro serviço para cadastro dos produtos vendidos e informações adicionais sobre ele, além de um micro serviço de painel de controle de série temporal para análise macro das vendas. Além disso, pelo fato de ser escalável e modularizável, pode ser ajustado para outro segmento com apenas algumas alterações no código fonte. O nome do projeto é BreShow e está disponível publicamente no seguinte link do GitHub para clonar e instalar: <https://github.com/VenancioIgrejas/BreShow>

Na pagina do projeto no Github contem informações de instalação e requisitos do hardware, bastando rodar a composição de Docker para levantar os micro serviços e, após terminarem as configurações automaticamente, estarão disponíveis no navegador nos links informados no guia do projeto.

1.4 Organização do Texto

O seguinte trabalho foi separado em sete capítulos. O primeiro consiste na introdução, onde são percorridos parágrafos sobre a motivação e contribuição do projeto desenvolvido, e quais são os problemas que impulsionaram a criação desse sistema. O segundo capítulo é voltado para conhecimentos básicos, onde são apresenta-

dos conceitos de arquiteturas e boas práticas de programação utilizados no trabalho. O terceiro capítulo aborda trabalhos relacionados na literatura e o quarto consiste na proposta do projeto, onde são tratadas as estruturas do sistema e as divisões dos seus módulos. O quinto é sobre a sua implementação, tais como, quais frameworks foram usados em seus módulos, como o código está organizado e as informações de sua estrutura. O sexto discute a prova de conceito, onde é apresentado o funcionamento do sistema tanto pelo lado do usuário, quanto pelo administrador. Por fim, no sétimo capítulo, é evidenciada a conclusão, onde foi descrito como ocorreu o processo do desenvolvimento do presente trabalho, tanto na parte estrutural e conhecimento, quanto na administração do tempo e problemas. Para fechar, expõe-se trabalhos futuros e melhorias do projeto.

Capítulo 2

Conceitos Básicos

Neste capítulo são discutidas, apresentadas e conceituadas teorias utilizadas no presente trabalho, com foco em aplicar técnicas de arquiteturas de software para tornar o código robusto, simplificando a manutenção deste a medida que se torna mais complexo, estabelecendo o código independente de ferramentas periféricas, entre outras vantagens.

2.1 Princípios SOLID

O termo SOLID representa um acrônimo para os cinco primeiros princípios de Design Orientado a Objetos (OOD) desenvolvido pelo Robert C. Martin, conhecido como Uncle Bob.[7] Esses princípios estabelecem práticas que ajudam o programador a ter códigos mais transparentes, organizados, facilitando a manutenção ou evolução a medida que cresce o programa e a sua leitura para outros desenvolvedores.[8]

A letra S de SOLID origina-se de Princípio da Responsabilidade Única (*Single Responsibility Principle*). Esse princípio decorre que toda classe ou estrutura similar possua apenas um propósito, não sendo, por exemplo, como um canivete suíço onde um objeto detém várias funções. Em outras palavras, uma classe deve ter métodos ou variáveis relacionadas com o seu propósito apenas e não uma única classe para todos os métodos do projeto.

A letra O provém de Princípio Aberto-Fechado (*Open-Close Principal*) e está relacionado aos objetos ou entidades estarem abertos para extensões, todavia, fechados

para modificações. Esse princípio significa que as classes bases ou já existentes não devem ser mudadas quando a regra de negócio mudar, mas estendidas em classes novas.

O Princípio da Substituição de Liskov (*Liskov Substitution Principal*) é representado pela letra L no SOLID e sua definição formal introduzida por Barbara Liskov diz que: “Se para cada objeto o1 do tipo S há um objeto o2 do tipo T de forma que, para todos os programas P definidos em termos de T, o comportamento de P é inalterado quando o1 é substituído por o2, então S é um subtipo de T”. Em outras palavras, qualquer classe derivada de uma classe pai deve ter os mesmos comportamentos que ela sem a necessidade de alterá-la. Se houver necessidade, a classe filha não deve ser herdada daquela classe pai.

A letra I representa o Princípio da Segregação da Interface (*Interface Segregation Principle*) onde explicita que as classes não deveriam ser forçadas a implementar interfaces que não serão utilizadas. Esse princípio diz para não gerar uma interface genérica para todas as classes que serão criadas, mas sim, para aplicar os métodos de herança quando necessário nas interfaces para segregá-las o máximo possível.

Por último e não menos importante, o Princípio da Inversão de Dependência (*Dependency Inversion Principle*) representado pela letra D de SOLID diz que o código deve depender das suas abstrações e não das implementações, ou seja, deve-se, por exemplo, ser possível alterar o sistema do banco de dados sem precisar alterar a estrutura do código do projeto, sendo feito utilizando abstrações como as interfaces.

2.2 Arquitetura MVC

A sigla MVC significa *Model-View-Controller*, traduzindo-se para Modelo-Visualização-Controle. Essas três palavras significam as camadas onde o projeto será dividido, e cada uma tem um significado ou papel importante específico dentro do código.[11]

O padrão de arquitetura MVC é utilizado em muitos projetos por possuir benefícios, como o isolamento das camadas, onde uma camada não interfere a outra. Ou seja,

por exemplo, o código da regra de negócio que está presente na camada do modelo não interfere no código da interface com o usuário que está na visualização. Esse isolamento proporciona flexibilidade e reuso de classes, além de permitir que mais de um desenvolvedor trabalhe ao mesmo tempo no projeto, facilitando a manutenção ou adição de recursos.

A camada de controle serve para interpretar algum evento disparado pelo usuário na tela como algum dispositivo de entrada e, assim, mapeá-las em comandos que são enviados para a camada modelo. Após o processamento dos dados por esse mesmo fluxo, o controle envia a informação para o usuário que é informado na tela pela camada de visualização.

Por sua vez, a camada de visualização é responsável por apresentar as informações oriundas das outras camadas na tela do usuário na forma de texto, gráfico, tabelas ou listas. Essa camada é apenas uma carcaça que ajeita da melhor forma os dados entregues para melhor compreensão do usuário.

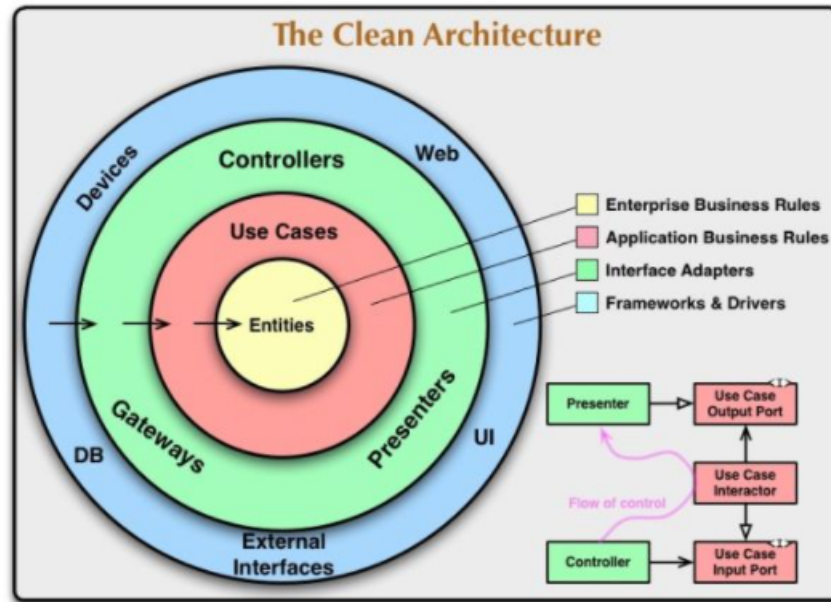
Por fim, e não menos importante, o modelo gerencia um ou mais elementos de dados processando os dados tanto oriundos da camada de controle, quanto do banco de dados. É dito que nessa camada as regras de negócio do projeto são desenvolvidas e, por isso, é considerada o coração da arquitetura, sendo quem modela o problema a ser resolvido.

2.3 Clean Architecture

A Clean Architecture foi desenvolvida pelo Uncle Bob no ano de 2012 com a proposta de unificar as teorias de outras arquiteturas famosas da época. Uma delas é a Onion Architecture, desenvolvida por Jeffrey Palermo. Essa arquitetura tem por finalidade tornar o projeto mais independente dos frameworks, das interfaces dos usuário, banco de dados e de qualquer agentes externos.[14]

Na figura 2.1, tem-se um esquema de como a arquitetura funciona na teoria. Os anéis representam diferentes áreas de softwares, já as setas representam as dependências entre as camadas. Ou seja, a camada de “frameworks e drives” apenas

enxerga a camada de “adaptadores de interface”, assim, não tendo o conhecimento de camadas mais internas da arquitetura. De acordo com Uncle Bob, quanto mais próximo a camada estiver do centro, maior será a sua abstração em relação ao projeto.



Fonte: <https://bit.ly/2a1px8t>

Figura 2.1: modelo Clean Architecture desenvolvido por Uncle Bob.[1]

A principal regra da Clean Architecture é chamada de Princípio da Dependência, onde as camadas de dentro para fora não devem apresentar nenhuma dependência externa, nem de forma indireta. Mantendo-se mais isoladas quanto maior for a sua abstração.

A camada “Entities” abrange as entidades do sistema e algumas regras de negócio da aplicação, estando presentes as regras que terão menor probabilidade de mudança e são mais próximas das regras as quais o projeto fora feito. É possível associá-la a interfaces e classes que apresentam uma representação do núcleo do negócio, e estes são provavelmente a última porção a ser alterada após uma modificação na parte externa da aplicação.

A camada “Use Cases” apresenta as regras de negócio mais específicas do sistema, sendo onde implementam todos os casos de usos da aplicação e o processamento dos

dados. Espera-se que apenas mudanças de requisitos do sistema afetem essa camada, ou seja, mudanças externas como banco de dados, interface do usuário e framework não a influenciam.

A “Interface Adapters” são softwares que servem para serem intermediadores entre as camadas mais internas como Entities e Use Case, e as externas. Nela são apresentados adaptadores que convertem os dados externos para se adequarem ao formato de dados das camadas mais internas, e vice-versa. Um exemplo disso seria um AutoMapper, framework de ORM e Controles.

A camada mais externa é conhecida pelo nome de “Frameworks”, consistindo em ferramentas como banco de dados, dispositivos de consultas ao projeto, interfaces externas e telas. Normalmente, ocorre pouca alteração no seu código, apenas para interligar suas camadas com a próxima mais interna.

A Clean Architecture tem seus lados positivos e negativos, contudo, sua ideia de arquitetura torna o código mais desacoplado e menos preso a determinadas tecnologias, afinal, estão sempre em constantes mudanças e atualizações.

2.4 Folder-by-Feature

A estrutura “Folder-by-Feature” consiste em agrupar os arquivos de um projeto MVC em pastas cujo conteúdo apresenta as mesmas características. Em outras palavras, uma entidade chamada “Usuário” terá o seu arquivo de controle, negócio, transferência de objeto e a entidade em si dentro de uma pasta chamada “Usuário”. [16]

Esse padrão é contrário ao utilizado normalmente por aplicações onde são separados as pastas pelo tipo de responsabilidade. Ou seja, por exemplo, a pasta “controller” de uma aplicação terá todos os controles das entidade do projeto. A priori, essa estrutura parece fazer mais sentido para organizar e juntar arquivos que estão divididos pelas camadas MVC. Entretanto, com o aumento do projeto e o número de arquivos e entidades, o desenvolvedor vê a necessidade de procurar dentro de poucas pastas e arquivos relacionados àquela entidade, os quais provavelmente estariam

dentro de pastas com inúmeros outros arquivos de baixo interesse do indivíduo no momento.

O fato de utilizar esse padrão facilita a manutenção e evolução do código, pois todas as informações e características de uma entidade se encontram dentro de uma mesma pasta. Além disso, torna-se viável o desenvolvimento com múltiplos desenvolvedores trabalhando ao mesmo tempo, afinal, cada um pode ser responsável por uma entidade e não por uma camada. Vale ressaltar que essa estrutura tem os seus lados negativos e, nesse caso, ela é recomendada para projetos que possam escalar de tamanho muito rápido ou projetos que já são grandes desde o início.

2.5 ERP

A sigla ERP significa Enterprise Resource Planning que, traduzido para o português, se torna “Planejamento dos Recursos Empresariais”. Dessa forma, esse sistema consiste em um software empresarial que serve para o gerenciamento automático dos recursos, estoques, financeiro, informações de vendas e outros setores dentro da empresa. Logo, o ERP passa a ter um papel fundamental funcionando como um centralizador de fluxo de trabalho, afinal, cada departamento, normalmente, necessita de uma ferramenta própria para poder exercer as tarefas e gerar relatórios.[17]

O ERP concentra as informações de forma inteligente e sem redundâncias, o que gera às empresas a autonomia na execução de tarefas repetitivas interligando a comunicação entre as áreas, evitando-se que cada setor possua um software isolado com dados redundantes de outros sistemas, podendo ocorrer a inconsistência de informações dentro das empresas. Por conta disso, a implementação de uma ferramenta integrada é garantia de informações assertivas e sólidas para a empresa.

Capítulo 3

Trabalhos Relacionados

Por conta do vasto interesse em ERPs, artigos relacionados a estudos de caso sobre funcionamento, implementação e análises são publicados extensivamente. Contudo, poucos artigos acadêmicos são voltados para um sistema de código aberto de ERPs e, em contrapartida, há um número maior de publicações no meio não acadêmico sobre o assunto, revelando uma lacuna entre a preocupação da academia e a demanda empresarial.

Trabalhos que visam micro, pequenas e médias empresas (MPME) exploram o uso de sistemas de ERPs de código aberto por conta do seu baixo custo e customização, quando comparados a sistemas robustos e solidificados no mercado, tais como SAP, Oracle, entre outros [18]. Além disso, a implementação de um sistema desse traz benefícios para a empresa, como redução de custos, aumento nas colaborações, análises da saúde da empresa de forma mais precisa e melhorada, aumento da produtividade, satisfação do cliente, eficiências nos negócios, permitido ampliação do seu mercado.

Existem muitos ERPs de código aberto (CA) no mercado, tais como WebERP [20], Compiere [21], Jfire [22], ERP5 [23], Odoo [24], Front Accounting [25]. Esses ERPs apresentam funcionalidades similares como setor de compra, vendas, controle de estoque, gestão de produção, gestão de relacionamento com o cliente (CMR), entre outros, mas diferem-se em pequenos pontos que são levados em conta na hora de escolher qual sistema se enquadra nas necessidades do empreendimento ou empresa, como, por exemplo, a OFBIz [26] que apresenta a funcionalidade de e-

commerce que a ADEMPIRE ERP não apresenta; O JPIRE [27] suporta o volume de dados de empresas de todos os portes dado a sua flexibilidade e abrangência, algo que a XTUPLE [28] não engloba, pois é projetada apenas para empresas de médio e pequeno porte, todavia, apresenta adaptação para a moeda brasileira em suas funcionalidades.

Trabalhos recentes vêm utilizando o sistema Odoo nas implementações e casos de estudo, pois apresenta mais resultados positivos que outros concorrentes. Por exemplo, os sistemas Odoo e Xtuple ostentam programas traduzidos no idioma português, diferentemente de seus concorrentes, mas, ainda assim, o Odoo possui maiores funcionalidades como adaptação para impostos e alíquotas brasileiras e possibilidade de emissão de nota fiscal eletrônica, tópicos esses que o Xtuple não contempla (Figura 3.1).

Outro caso está presente na Figura 3.2, onde há comparação entre 4 sistemas de CA ERPs e apenas o Odoo apresenta todos os requisitos impostos e necessários no trabalho.

REQUISITOS	SOFTWARE	
	OpenERP	XTuple
Acesso para <i>download</i> pela internet	✓	✓
Ter a possibilidade de rodar o sistema em interface Linux	✓	✓
Possuir banco de dados gratuito	✓	✓
Possuir tutoriais de instalação disponibilizados na internet	✓	✓
Possuir tutoriais de implantação e operação disponíveis na internet	✓	✓
Possuir comunidades de discussão sobre o sistema	✓	✓
Possuir adaptação para moeda brasileira	✓	✓
Possuir adaptação para impostos e alíquotas brasileiras	✓	
Possuir possibilidade de emissão de nota fiscal eletrônica	✓	

Figura 3.1: Quadro comparativo entre o Odoo(OpenERP) e Xtuple.[2]

O sistema Odoo foi desenvolvido em 2005 com o nome de TinyERP e, após três anos, mudou para OpenErp já que foi incorporado e reconhecido por grandes empresas. Logo em seguida, após a adição dos módulos CRM, Website e E-commerce, a empresa adotou o nome de Odoo para o seu sistema. Em 2015, entrou para a lista das companhias que mais crescem na Europa. A aplicação é capaz de automatizar quase toda uma empresa de forma performática, cobrindo quase toda as suas neces-

sidades e integrando processo empresarial. Ele é dividido em módulos e aplicativos que vão desde o faturamento e a manufatura, até gerenciamento de projeto e gestão de armazenamento. Sua versão gratuita está disponível na Odoo Community, tanto para baixar através de Download, quanto na versão de Docker, apresentando uma gama de módulos de ERP e o uso de um banco de dados de CA gratuito. Existe a possibilidade de migrar para a versão paga, que dá direto a suporte e todos os módulos de ERPs existentes na aplicação.

Dessa forma, como explicitado, os sistemas de ERPs atuais apresentam um modo gratuito, porém, alguns possuem acessibilidade limitada da plataforma, como o caso do Odoo, e outros estão desatualizados em relação ao design, serviços ou tecnologia. Assim, o seguinte trabalho tem por contribuição o desenvolvimento de um ERP totalmente gratuito e de código aberto voltado primeiramente para o segmento de brechós como prova de conceito, no qual apresenta o módulo de gerenciamento de usuário, de vendas e de análise, uma vez que esses módulos apresentam uma importância primária para esse setor.

	Ferramentas de Gestão de Projetos	Odoo	MyCollab	OrangeScrum	Open Project
	Requisitos presentes na literatura				
1	Fiabilidade	X	X		
2	Performance	X			
3	Usabilidade	X			
4	Extensibilidade	X			
5	Comunidade – existência de suporte técnico	X	X	X	X
6	Conhecimentos da equipa informática	X			
7	Capacidades de manutenção a longo prazo	X			
8	Atualização funcional futura	X			
9	Tipo de licença – licença OS	X	X	X	X
	Requisitos apresentados pelos colaboradores				
10	Acesso a informação detalhada do projeto	X	X	X	
11	Definição datas de entrega e prioridades	X	X	X	X
12	Definição e atribuição de tarefas a executar em cada projeto	X	X	X	X
13	Alocação de gestor a cada projeto	X	X	X	
14	Criação base de dados de clientes	X			
15	Gestão de agendamento de reuniões	X		X	X
16	Diferentes níveis de acesso à informação	X	X		
17	Notificação de alterações	X	X	X	X
18	Notificação de início de tarefas	X	X	X	X
19	Avaliação do estado do projeto	X	X	X	X
20	Anexar documentos	X	X	X	X
21	Visualização de histórico de alterações	X	X		
22	Integração com <i>Microsoft Outlook</i>	X	X	X	X
23	Facilidade de uso	X		X	
24	Flexibilidade	X			
25	Mensagens instantâneas	X	X		

Figura 3.2: Quadro comparativo entre o Odoo, MyCollab, OrangeScrum e Open Project.[3]

Capítulo 4

Proposta

A proposta do presente trabalho consiste no desenvolvimento de um sistema de ERP de código aberto e gratuito disponibilizado na plataforma GitHub, voltado para o segmento de brechós como prova de conceito. O motivo se dá pelo fato de, atualmente, alguns microempreendedores e trabalhadores autônomos (MTA) ainda utilizarem blocos de notas ou editores de planilhas digitais, como o Excel (Microsoft), para desfrutarem da análise e o controle de suas vendas, acarretando num possível erro de cálculo, além de trabalhos manuais repetitivos de forma desnecessária. Entretanto, sistemas ERPs presentes no mercado, em sua grande maioria, consistem em sistemas complexos com algumas ferramentas desnecessárias para os MTA, impactando na plataforma e bloqueando-os, além da sua instalação ser confusa para leigos.

O capítulo está dividido em tópicos e subtópicos de arquitetura de software e módulos, respectivamente. O primeiro consiste numa proposta de aplicar os conceitos e teorias para tornar o código mais robusto às atualizações e otimizações. A segunda parte diz respeito aos módulos existentes no projeto e suas devidas importâncias para estudo de caso presente no trabalho.

4.1 Arquitetura de software

Ao longo das últimas duas décadas, os softwares passaram a ter uma importante influência sobre nossas vidas, afinal, estão mais complexos e robustos. Os

sistemas estão cada vez mais agilizando, organizando e nos informando, passando a ter funções chaves em operações extremamente complexas e, para isso, também foram necessários aprimoramentos e melhorias em seus códigos, evidenciando a importância da arquitetura de software.

Os benefícios de se ter um software bem arquitetado vão além da performance do sistema. Além do aumento do desempenho, tem-se a garantia de escalabilidade, um termo que as empresas preservam e almejam para seus sistemas, pois refletirá no tamanho do orçamento da manutenção devido a refatoração do sistema legado para se adequar a nova atualização, caso não seja escalável. Ademais, a personalização do sistema se torna um trabalho menos árduo de ser feito caso haja uma evolutiva.

Dessa forma, o presente trabalho foca em teorias e embasamentos afim de que o projeto apresente uma boa arquitetura de software e possa ser performático, escalável e personalizável. Para isso, utilizam-se ferramentas de código aberto que estão presentes no mercado e que estão consolidados, além de aplicar os conceitos de Clean Architecture e princípios Solid.

4.2 Módulos

São três módulos que compreendem o projeto: Aplicação Web, Dashboard e Gerenciamento de Usuários. O primeiro e o último são obrigatórios, pois consistem em, respectivamente, alimentar o sistema com informações das vendas e gerenciamento dos usuários que controla o acesso de cada indivíduo ao sistema.

Mesmo o projeto sendo composto por três módulos, o usuário terá acesso apenas a dois deles conforme informado na figura 4.1. A Aplicação Web para adicionar as informações dos produtos vendidos e, dependendo do nível de autorização pelo administrador, o acesso a uma parte da aplicação de dashboard para consultar e acompanhar as informações globais das vendas adicionadas na aplicação. O administrador, por outro lado, terá acesso ao Dashboard e Gerenciador de Usuários, onde irá gerenciar quais usuários terão acesso a quais painéis de visualização, e, no caso do Gerenciamento, terá controle dos usuários tais como informações pessoais, reenvio

de senha e alterações de informações.

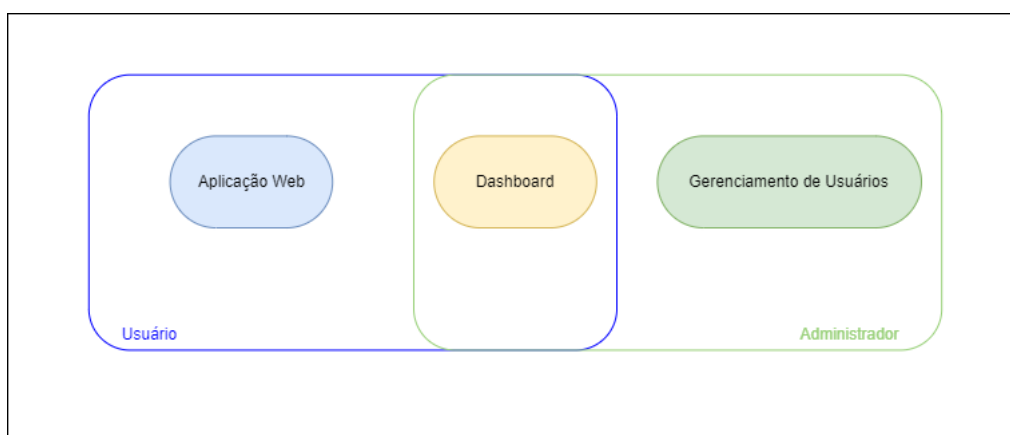


Figura 4.1: Diagrama de Venn dos Módulos do projeto

4.2.1 Aplicação Web

A aplicação consiste numa tela web onde o usuário consegue fazer a adição e edição de uma venda, fornecedor e tipo do produto. Ao adicionar uma venda, o indivíduo preenche um formulário com as informações do valor da venda, quantidade, fornecedor do produto, e o tipo do produto, além de outros elementos menos relevantes para identificá-lo de forma mais organizada. Na tela principal de cada aba, há uma lista do que foi adicionado na mesma, por exemplo, na tela principal da categoria, há uma lista com as qualidades já criadas pelo usuário.

O micro-serviço supracitado apresenta 4 abas dentro de um mesmo domínio do site: Produtos, Categorias, Fornecedor e Principal. As figuras 4.2, 4.3, 4.4 representam o fluxograma do caso de uso da aba de produtos, fornecedor e categorias, respectivamente. Na aba produtos, temos a lista de todos os itens cadastrados com os seus respectivos valores, categorias e fornecedores, sendo esses campos obrigatórios na hora do cadastro. Além disso, temos a criação, edição e exclusão de itens nessa mesma página. No caso da aba de fornecedores, o formato se assemelha com a de produtos, contudo, possui campos referentes a cadastro de pessoa com nome, número de telefone, porcentagem do lucro e observação, assim como na parte de categoria, onde o usuário cria a categoria que melhor servir, pois é sabido que nem sempre as categorias padrões existentes nas aplicações web do mercado se encaixam

na estrutura de produtos da empresa.

Por fim, tem-se a tela principal que lista as informações das três outras abas, sendo, nesse caso, apenas para leitura e com filtros para uma rápida pesquisa com intuito de mostrar uma informação específica. Para aquisição de dados processados sobre uma visão macro da aplicação, recomenda-se acessar o serviço de dashboard.

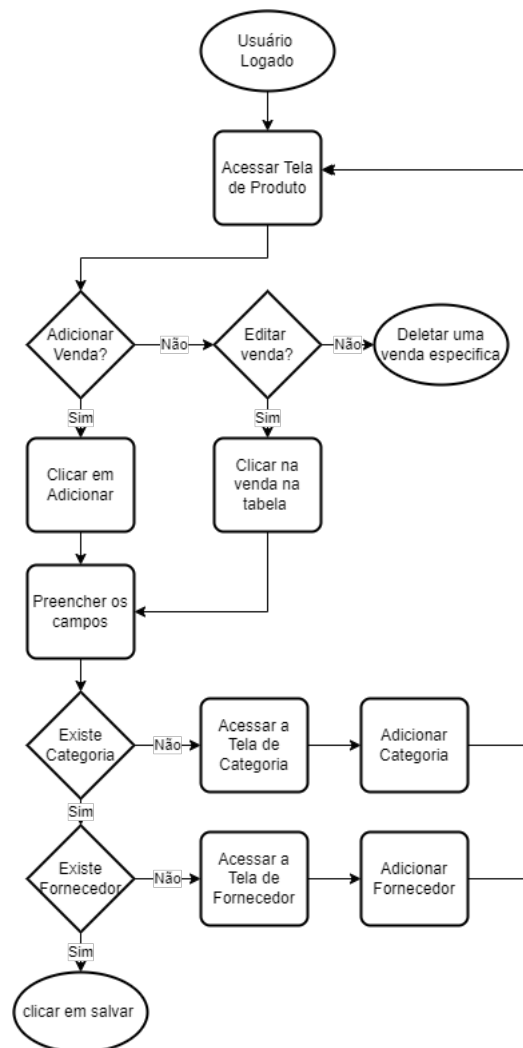


Figura 4.2: Fluxograma da tela de produtos

Dessa forma, a aplicação web funciona como uma adição de informações ao sistema, assim como o empreendedor que utiliza planilhas e blocos de nota para adicionar alguma venda. Além disso, para esse módulo, aplicaram-se embasamentos teóricos de arquitetura de software, como Clean Architecture para deixar o código e o fluxo desacoplados, permitindo uma personalização ou mudança com o mínimo

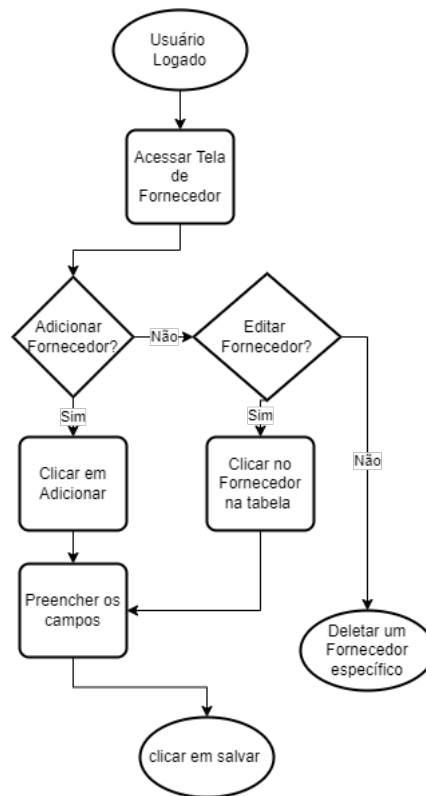


Figura 4.3: Fluxograma da tela de fornecedor

de alteração nos arquivos, utilizando-se os princípios SOLID para disponibilizar um código organizado, de fácil entendimento e manutenção.

4.2.2 Dashboard

No serviço de dashboard, visualizam-se informações macro da saúde de vendas do comércio. Nele constarão gráficos de porcentagem das vendas por fornecedores e categorias, além de séries temporais do lucro e do total de vendas diárias. Vale ressaltar que o filtro de período é aplicado para todos os gráficos simultaneamente, facilitando a compreensão tanto do total quanto de um período, seja diário, semanal, mensal, anual ou até do início da contabilização.

Nesse serviço existem apenas dois tipos de contas, a de usuários e de administradores. O primeiro só poderá ter acesso a leitura dos gráficos e ajustes dos filtros para selecionar o período de análise, já o segundo poderá alterar os os tipos de gráficos, posições e também o que deve e quais dados serão mostrados. Além disso, o administrador terá controle dos usuários que poderão ler as tabelas e dados. A

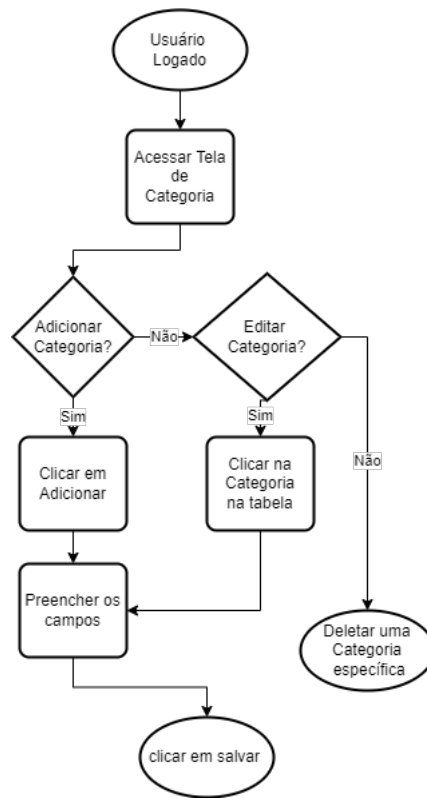


Figura 4.4: Fluxograma da tela de categorias

figura 4.5 retrata o caso de uso do dashboard na visão tanto do usuário, quanto do administrador.

A ideia de ter o dashboard separado da aplicação consiste no fato do projeto mostrar-se mais fragmentado e com aparência de micro serviços, afinal, esse tipo de divisão torna mais alta a manutenção e o controle de qualidade do serviço, pois apresenta um mínimo de impacto no restante das atividades e do projeto como um todo, facilitando na hora de analisar alguma melhoria e permitindo a divisão de uma equipe para manutenção.

4.2.3 Gerenciamento de Usuários

O Gerenciamento de Usuários é essencial para trabalhar em conjunto com a aplicação web, pois faz parte do seu gerenciamento de usuários. Uma ideia inicial da aplicação web se baseia em cada usuário ter seus próprios dados, não possuindo acesso à informações de outros e, para que isso ocorra, há a necessidade do uso de um gerenciador de usuários.

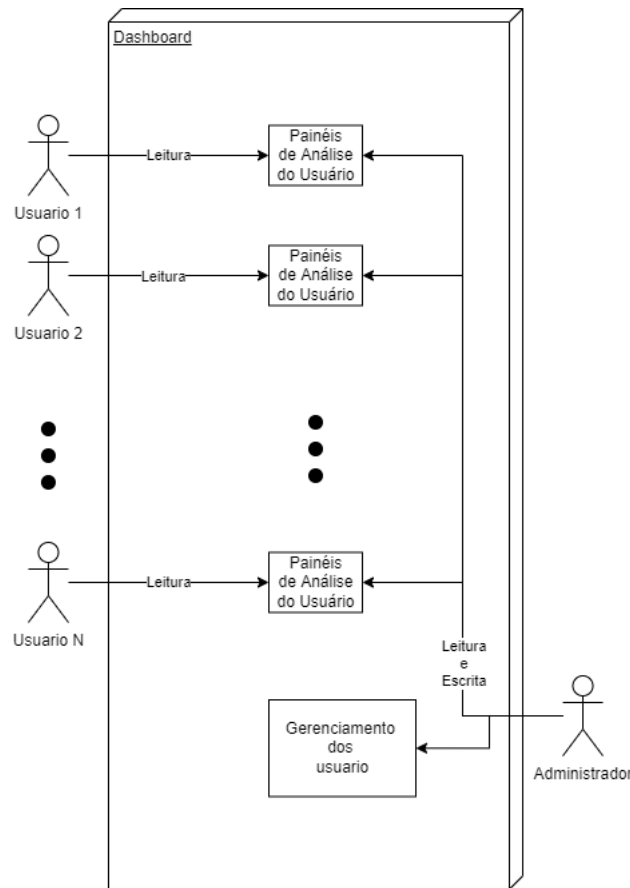


Figura 4.5: Caso de uso do módulo de dashboard

Com o crescente aumento e avanço dos sistemas de software e tecnologias, países e seus governantes seguem adotando legislações mais rígidas em relação ao armazenamento de dados dos usuários e suas consecutivas utilizações, como, por exemplo, a Lei Geral de Proteção de Dados Pessoais (LGPD) [30]. Dessa forma, pensando na segurança dos dados dos usuários, o projeto conta com um gerenciamento de usuários que armazena os dados em um ambiente separado da aplicação web, ou seja, tanto a tela de login, quanto a parte de segurança de navegação, é feita por este micro serviço.

Visando melhorar a segurança, a página web trabalha com tokens que expiram, ao contrário do formato de acesso padrão. A aplicação reconhece os usuários por um ID de 12 caracteres contendo letras e números, aplicando uma camada de segurança na aplicação e dificultando no roubo de dados dos clientes, afinal, os dados sensíveis estarão dentro do banco do micro serviço de gerenciamento de usuários.

Sendo assim, para esse módulo, utilizaram-se de Código de Terceiros e Aberto, evitando trabalho desnecessário e a redundância de aplicação, dado que se aplica ao contexto e apresenta versões estáveis com poucos problemas, além de inúmeras extensões que permitem a personalização de layout e contas.

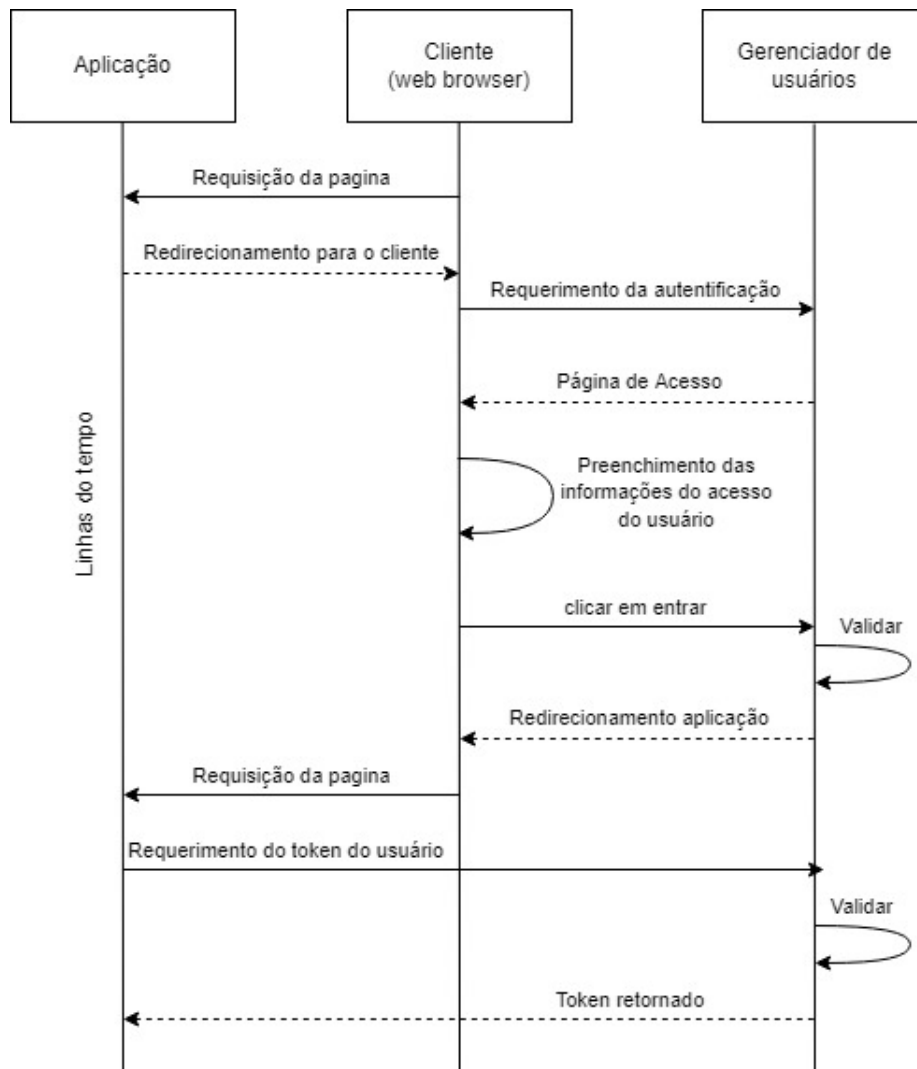


Figura 4.6: Lógica do login e aquisição dos tokens para a aplicação

A figura 4.1 representa a lógica por trás da tela de login e da aquisição do token pela aplicação. Nesse fluxo, tem-se três objetos principais: A aplicação web, o cliente (usuário que acessa a página pelo navegador de rede) e o gerenciador. O usuário faz uma primeira requisição para a aplicação e esta o redireciona para o gerenciador. Se o usuário já tiver logado anteriormente e o token de acesso não expirou, o gerenciador o redireciona direto para a aplicação. Caso contrário, irá redirecioná-lo para a tela de login. Ao acessar essa tela, o cliente preenche as informações para login ou

cria um usuário através da tela de cadastro. Uma vez tendo clicado em "entrar", o gerenciador verifica se as informações batem com as informações em seu banco de dados, e estando corretas, o redireciona para a aplicação permitindo o acesso às páginas internas. Dentro da aplicação, quaisquer informações que desejam ser salvas, ou o acesso de páginas, a aplicação irá requerir o token do usuário para dar continuidade. Se o login estiver expirado ou incorreto, o cliente é redirecionado para tela de login ou a ação não se torna permitida.

Conforme mostrado na figura 4.1, verificam-se que as únicas informações do usuário que a aplicação tem ciência é que este pode logar nas camadas mais internas dela ou alterar e visualizar certos dados através das informações que vem do gerenciador. Por outro lado, os dados sensíveis do cliente como usuário e senha são passados apenas para o gerenciador e somente este determina se o usuário pode ou não logar na aplicação. Em contrapartida, o gerenciador não conhece os dados internos da aplicação, como o valor total de vendas ou quem são os fornecedores daquele usuário. Essa separação dos dois serviços torna a aplicação mais segura e robusta contra exposição de dados sensíveis a indivíduos com segundas intenções.

Assim como explicitado no módulo de Dashboard, o gerenciador de usuários da aplicação também utiliza código de terceiros e código aberto, pois sua implementação é completa, robusta e apresentar mais camadas de segurança quando comparado a um código desenvolvido de caráter próprio.

Capítulo 5

Implementação

Neste capítulo será elucidado como fora realizada a implementação do projeto, assim como as escolhas das arquiteturas e conhecimentos teóricos aplicados. Haverá a apresentação dos detalhes a nível de código e diagramas com maior transparência possível para o leitor que almeje replicá-lo. Como anteriormente citado, o código se encontra disponível no link <https://github.com/VenancioIgrejas/BreShow>.

O capítulo é dividido em duas seções: na primeira parte tem-se a arquitetura do projeto e os sistemas periféricos internos para seu funcionamento, na segunda parte encontra-se a implementação das aplicações ou módulos do projeto, sendo, nesse caso, os microsserviços dentro do sistema.

5.1 Arquitetura

A idealização da arquitetura do projeto está de forma a facilitar sua instalação e configuração, optando-se pela containerização dos aplicativos e serviços, bastando ao usuário rodar um comando para que o programa seja instalado e configurado quase em sua totalidade. Entretanto, mesmo com a containerização, fora necessário um meio de unir os contêineres e fazer os serviços conversarem entre si. Dessa forma, utilizou-se uma composição de contêiner para orquestrar todos os serviços de maneira simples e unificados.

Cada aplicação está dentro de um contêiner. Ao todo são quatro contêineres dentro da composição, sendo cada uma isolada da outra porém compartilhando da

mesma rede. Dispõe-se de Dashboard, consistindo em uma aplicação de visualização de dados tratados do tipo interface gráfica para a análise de indicadores relacionados às informações de vendas que vieram do banco. A segunda aplicação é de Gerenciador de Identidade e Aplicação (GIA) que serve para gerenciar, validar e autenticar usuários para acessarem a aplicação web. A terceira é a aplicação de única página ou aplicação web (SPA), onde são fornecidos interfaces para o usuário adicionar informações referentes a vendas, fornecedores e categorias que, futuramente, serão utilizados pela aplicação de Dashboard. Por último tem-se a aplicação de banco de dados, o qual se conecta com todos os outros contêineres para prover ou armazenar dados ligados tanto ao fluxo de negócio, quanto metadados das aplicações (figura 5.1).

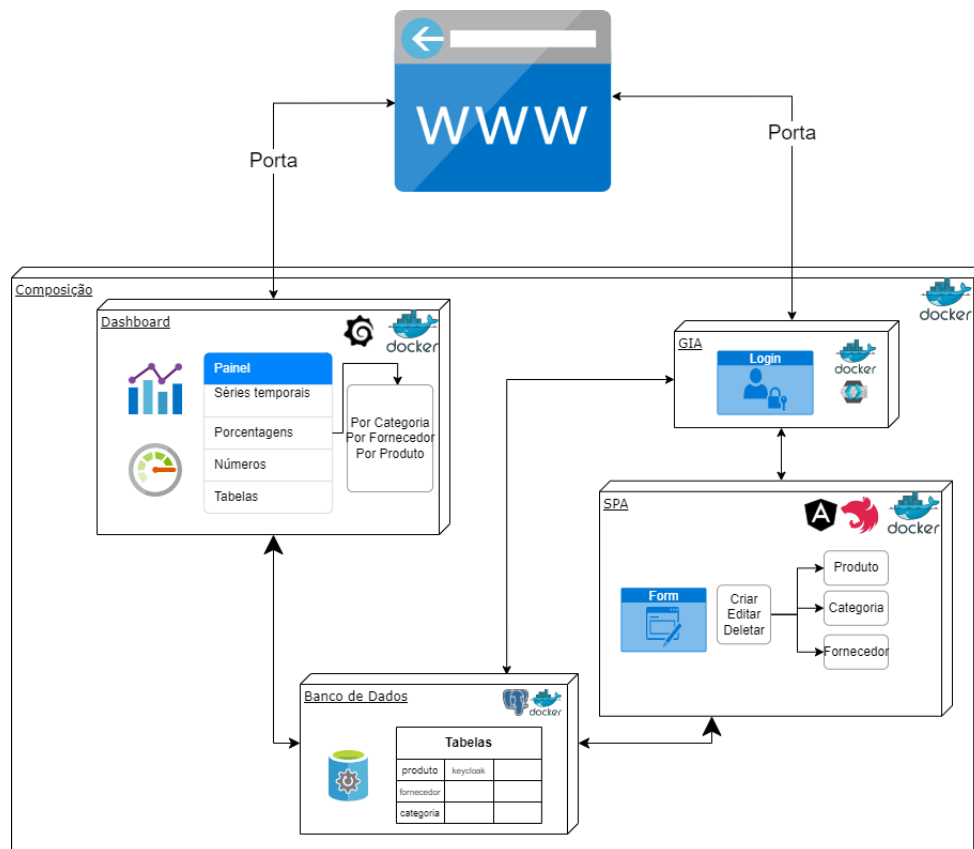


Figura 5.1: Arquitetura macro do projeto separada por contêiner

O fluxo consiste de duas URL's que o usuário pode acessar através de portas. Uma da aplicação web que é intermediado pelo GIA, e a outra pela aplicação de dashboard. Por sua vez, apenas eles e a SPA tem acesso ao banco de dados, onde são armazenadas informações cruciais dos usuários e para seu funcionamento. Dentro

da composição, os contêineres se comunicam através de portas e chaves como, por exemplo, o Dashboard, GIA e SPA que se utilizam da porta 5432 e autenticação por usuário para acessarem os dados no contêiner de banco de dados, já a comunicação entre o GIA e o SPA vem da porta 28080. Essas informações estão explicitadas no arquivo `docker-compose.yml` presente na pasta raiz do projeto.

5.1.1 Orquestrador

O orquestrador escolhido para o projeto foi o Docker, pois se trata de uma Plataforma Como Serviço (PaaS) que já está consolidada no mercado e consegue isolar do sistema operacional do computador o software que nele for instalado. Também é possível escrever comandos em um arquivo nominado `Dockerfile` para ordenar uma sequência de configurações em tempo de execução de instalação do contêiner.

Como cada contêiner necessita do seu próprio `Dockerfile`, este fato torna a configuração e instalação demoradas, caso precise realizar essa atividade em um número elevado de aplicações em seus próprios contêineres. Por conta disso, utilizou-se a ideia de composição, ou seja, algo que já existe no Docker. O arquivo de configuração da composição se encontra no caminho:

```
{raiz do projeto}/docker-compose.yml
```

Esse arquivo é uma junção de vários `Dockerfile` para instalar e configurar de forma paralela inúmeros contêineres. Para instalar os pacotes apropriados já com uma configuração padrão, utilizaram-se imagens pré-configuradas das ferramentas disponibilizadas pela respectiva empresa que contém os direitos. Dessa forma, as imagens utilizadas dentro de `Dockfiles` e `docker-compose` foram a `node:alpine` com `Nestjs` para o backend do contêiner da SPA, `node:14` com `Angular 12` para o frontend da SPA, `grafana/grafana:7.3.6` para o Dashboard, `postgres-dev:1.0.0` para o banco de dados e, por último, `jboss/keycloak` para o GIA. Uma vez os arquivos configurados, deve-se rodar o comando “`docker-compose up`” do terminal na raiz do projeto para iniciar a composição.

5.1.2 Banco de dados

O banco de dados utilizado para o projeto fora o Postgresql, por se tratar de um banco de dados relacional de código aberto, documentação completa, com comunidade ativa e com imagem no DockerHub. A configuração do contêiner está no arquivo docker-compose com as seguintes linhas:

```
postgres:
  container_name: postgres_containerbreshow
  image: postgres-dev:1.0.0
  build:
    context: .
    target: development
    dockerfile: ./data/Dockerfile
  environment:
    POSTGRES_USER: ${POSTGRES_USER}
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
  volumes:
    - ./data/postgres-data:/var/lib/postgresql/data
  ports:
    - "5432:5432"
  networks:
    - postgres
```

Após a ativação do contêiner citado, não há necessidade de alteração, pois a criação das tabelas informadas na figura 5.1 são feitas automaticamente pelas outras aplicações. Além disso, visando a segurança, as informações que advém do campo “enviroment” no docker-compose.yaml são extraídas do arquivo de ambiente (.env) do projeto.

5.1.2.1 DER do Banco de Dados

A modelagem do projeto consiste na existência de três tabelas importantes dentro do banco. São elas product, category e provider, as quais são representadas pelas entidades Produto, Categoria e Fornecedor, respectivamente. A tabela prod-

uct apresenta duas chaves estrangeiras que são da tabela category e provider, afinal, toda venda deve necessariamente ser categorizada e pertencente a um fornecedor específico. Todas as tabelas apresentam uma coluna chamada “idUser” que retrata o ID do usuário da aplicação que adicionou o dado, além de algumas colunas em comum que são metadados oriundos do pacote TypeORM. O Diagrama Entidade Relacionamento (DER) gerado por um gerenciador de PostgreSQL, está representado na figura 5.2. Vale ressaltar que a tabela “migrations_typeorm” serve para o controle de versionamento das migrações feitas pelo TypeORM, sendo uma tabela de metadado.

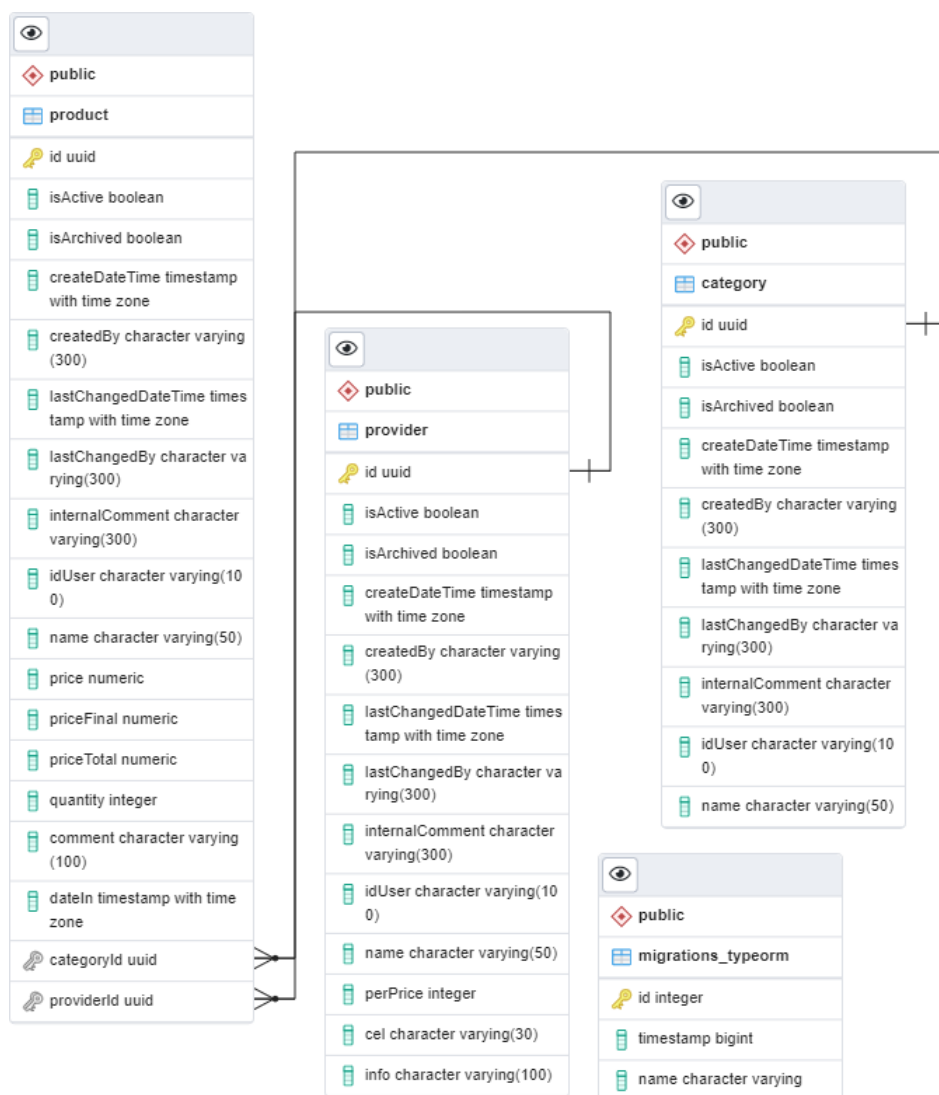


Figura 5.2: Diagrama entidade relacionamento do banco de dados da aplicação

5.2 Aplicações

Atualmente o projeto conta com três aplicações, onde duas delas, o gerenciador de usuários e a aplicação web, trocam informações entre si na parte de segurança. Em contrapartida, o dashboard é o único que funciona de forma isolada das outras aplicações.

Após análise da estrutura da aplicação web, fora estipulado que a parte de identificação, segurança e usuários seria delegado a uma terceira aplicação já existente e consolidada no mercado. Essa escolha aumenta a segurança e confiabilidade do usuário pelo projeto.

5.2.1 Gerenciador de usuários

O software de gerenciador de usuários escolhido foi o Keycloak por se tratar de um código aberto, com recursos gratuitos completos e utilizado por muitas empresas com notabilidade. Realizou-se uma análise de outros software de GIA, como Firebase Authentication e Auth0, porém ambos apresentam limitações na versão gratuita, indo contra a proposta do trabalho de ser um projeto totalmente gratuito.

Essa aplicação é instanciada quando levantado os contêineres usando o comando docker-compose. A parte responsável do Keycloak no arquivo docker-compose.yml é representado por:

```
keycloak:
  container_name: keycloak_containerbreshow
  image: jboss/keycloak
  volumes:
    - ./keycloakImports:/opt/jboss/keycloak/imports
  environment:
    DB_VENDOR: postgres
    DB_ADDR: postgres
    DB_DATABASE: ${POSTGRES_DB}
    DB_USER: ${POSTGRES_USER}
    DB_PASSWORD: ${POSTGRES_PASSWORD}
```

```
DB_PORT: 5432
KEYCLOAK_USER: admin
KEYCLOAK_PASSWORD: password
JDBC_PARAMS: "ssl=false"
KEYCLOAK_IMPORT: /opt/jboss/keycloak/imports/realm-export.json
-Dkeycloak.profile.feature.upload-scripts=enabled
ports:
  - "28080:8080"
networks:
  - postgres
depends_on:
  - postgres
```

Válido lembrar que a imagem do Docker do Keycloak é original da empresa e as variáveis de ambiente devem ser alteradas para uma maior segurança da aplicação. Foram utilizados os volumes para manter as informações salvas de configurações do realm e clients, que são essenciais para o funcionamento do projeto.

Ao acessar o link <http://localhost:28080/>, logar com o usuário e senha presentes no código acima do docker-compose, deve-se colocar no realm BreshowAD, pois é a que o projeto fora configurado. Ao clicar na aba “client” na esquerda, aparecerão os clients tanto do próprio sistema, quanto os criados manualmente no centro da tela. O resultado final será parecido com o da figura 5.3. Os clients “angular-front” e “nestjs-back” foram criados manualmente com o intuito de serem entidades de dentro da aplicação web para fazerem as requisições necessárias no Keycloak para autenticação, consulta de informações do usuário, ou se possuem permissões de acesso, e visualização de dados.

O client angular-front refere-se ao front-end da aplicação web, ou seja, é porta de entrada da aplicação e é quem redireciona o usuário à tela de login do Keycloak (da aplicação web) caso tente acessar alguma URL interna do sistema web não estando logado. Uma vez logado, o client permite que o usuário acesse as páginas internas da aplicação.

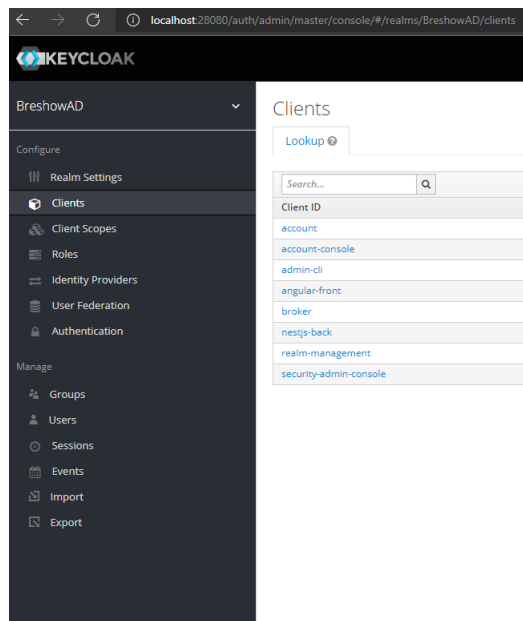


Figura 5.3: Tela do cliets do realm BreshowAD do Keycloak

O método de login mencionado no parágrafo anterior é possível devido ao acesso do client ser público, sendo uma opção selecionável na tela de configurações do próprio client. Esse tipo de login faz com que a aplicação necessite de uma tela de login para validar o acesso do usuário a aplicação. Vale ressaltar que é necessário configurar a rota raiz para a aplicação web (<http://localhost:4200>), pois será onde o Keycloak redirecionará o usuário após o login na plataforma. Além disso, é possível configurar quais rotas são válidas para o client acessar, tornando a aplicação mais segura a possíveis tentativas de roubo de dados.

O client `nestjs-back` é referente ao back-end da aplicação web e foi configurado pra ser do tipo “bearer-only”. Essa configuração obriga que, ao fazer uma requisição para o servidor, seja necessário que passe no cabeçalho da requisição um JSON Web Token (JWT), comprovando que o indivíduo tem permissão para acessar ou atualizar informações ao servidor.

No caso do presente trabalho, O JWT pode ser requisitado pela aplicação web após o usuário entrar na tela de login e utilizá-lo para fazer qualquer operação no backend, tudo por baixo da aplicação web. Em outras palavras, o JWT é um intermédio de segurança que o front-end e o back-end da aplicação usam para trocar informações entre eles e, assim, não permitindo que requisições maliciosas tentem

roubar informações do backend. Pelo lado do Keycloak, o client angular-front fornece um JWT do usuário logado para o client do nestjs-back fazer o controle do acesso.

Na sessão sobre a aplicação web, será mostrado e explicado mais a fundo as configurações desses clientes nas respectivas arquiteturas, como faz sua instalação e que frameworks utilizar para implantar o Keycloak. Observa-se que o uso de dois clientes para a mesma aplicação se tornou útil, pois transformou o backend em um microsserviço que pode ser consumido por outros sistemas, afinal basta ter o cabeçalho correto e o corpo da requisição no mesmo padrão para isso ocorrer.

A etapa da implementação foi dificultosa e observaram-se muitos problemas, pois os erros que apareciam na aplicação referentes ao Keycloak eram genéricos e foi necessário utilizar o método de tentativa e erro para conseguir configurá-lo. Por conta da documentação ser complexa para leigos em cibersegurança, e a comunidade ainda estar em expansão, algumas configurações de segurança adicionais não foram aplicadas por aparecerem bugs nos sistemas com logs confusos.

5.2.2 Aplicação Web

A aplicação web fora a única feita do zero e baseada em boas práticas de programação, utilizando princípios como o SOLID e Clean Architecture. O seu intuito é, através do navegador, permitir que os usuários insiram informações sobre as suas vendas para alimentar o banco de dados com os dados refinados para uso posterior, fazendo-o de forma mais simples e facilitada.

De acordo com a regra de negócio estipulada no projeto, as vendas que são adicionadas precisam estar relacionadas a uma categoria e a um fornecedor. Dessa forma, precisou-se criar três tabelas referentes a cada entidade, denominadas “product”, “category” e “provider”, sendo referenciadas as entidades de vendas, categorias e fornecedores respectivamente.

Essa aplicação foi separada em dois blocos: o primeiro bloco refere-se ao código que está rodando no navegador e é o responsável pela parte visual da aplicação, conhecido como frontend. No segundo bloco, ocorre o processamento dos dados e

a aplicação da regra de negócio, chamada-se de backend. Essa metodologia segue o padrão de projeto MVC e o código foi implementado seguindo os seus princípios.

5.2.2.1 FrontEnd

Inicialmente, escolheu-se React para essa parte da aplicação, porém foram observados alguns problemas com as bibliotecas do Keycloak para validação dos usuários. Dessa forma, por conta desse problema e o NestJs ter sido escolhido como o framework para o backend, escrito em Typescript e baseado na estrutura do angular, decidiu-se o Angular 12 como motor do frontend. Além disso, para customização de componentes mais rebuscados, foram utilizados componentes desenvolvidos pelo PrimeNG.

O código dessa porção do trabalho encontra-se dentro da pasta “client”, localizada na raiz do projeto e seu esquema pode ser visualizado na figura 5.4. A organização do código consiste na criação de pastas voltadas para a fácil manutenção, compreensão e expansão deste ao expandir a regra de negócio. A parte principal do frontend está na pasta client/src/app, onde as telas são montadas. A pasta “component” armazena os componentes utilizados em mais de uma tela como, por exemplo, a estrutura padrão. Já a pasta “module” possui os mapeamentos das entidades utilizadas na aplicação, como categoria, fornecedor e produtos(vendas) que são da regra de negócio, jwt-token, e router utilizados internamente no angular.

A pasta “service” está atrelada a montagem das requisições feitas pelo usuário ao disparar uma ação na tela, como um botão de salvar, deletar ou criar. Vale ressaltar que cada entidade tem o seu serviço separado, respeitando os princípios SOLID. Na pasta guard e init estão os arquivos voltados para as configurações do Keycloak para o client “angular-front” do realm BreshowAD. A classe AuthGuard do arquivo guard/auth.guard.ts serve para validar e bloquear possíveis usuários que queiram acessar URLs internas da aplicação sem autorização pelo Keycloak. Além disso, o arquivo keycloak-init.ts tem o intuito de forçar a inicialização primeiro da configuração do serviço do Keycloak e, só após o retorno do serviço, o angular inicia a aplicação.

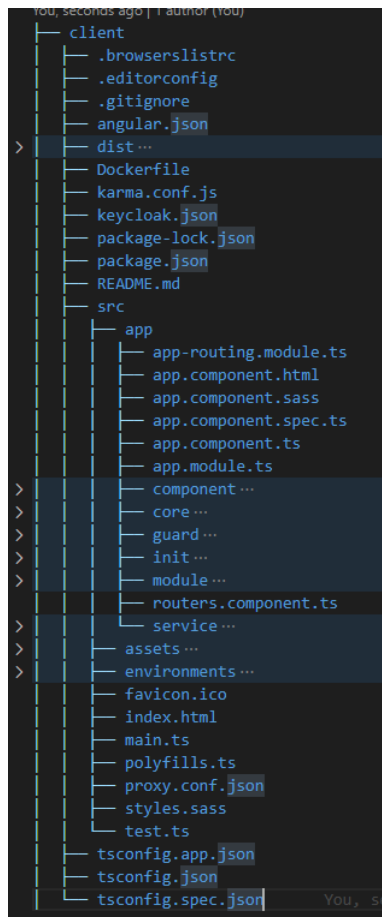


Figura 5.4: Estrutura de pastas do frontend da aplicação web

Na pasta “core” estão os códigos referentes as abas da aplicação web. Na figura 5.5, observa-se que cada entidade apresenta sua própria pasta e suas diferenças. Esse rearranjo facilita na hora da manutenção de cada aba, pois estão separadas e contam também com a aba de home. A estrutura padrão do angular fora importante para criar um modelo dentro de cada pasta que continha a página html, o arquivo de estilo (sass) e o arquivo de componente, que seria a “maquina de estados” por traz da tela. Decidiu-se separar o código do componente de formulário de cada entidade (entidade-page) em um novo componente chamado entidade-form para deixar o código mais organizado.

Outros arquivos que não foram mencionados no texto, todavia, estão presentes na figura 5.4 são relacionados a configurações do Keycloak, pacotes de bibliotecas utilizados, compilações, ajustes de rotas e configurações do Typescript e do Angular.

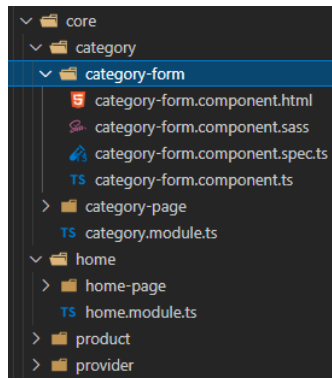


Figura 5.5: Estrutura da pasta “Core” do frontend da aplicação web

O desenvolvimento do frontend fora mais simples, demorando mais a confecção da aba de categorias, a primeira a ser criada. As outras, que possuíam seus estilos parecidos, desenvolveram-se bem rápido. Por conta da arquitetura robusta da aplicação web, a parte estética das telas serão para trabalhos futuros.

5.2.2.2 Backend

Para o servidor, escolheu-se a ferramenta Nestjs por se tratar de um framework em Node e apresentar uma grande aceitação pela comunidade, além de uma documentação simples e completa. Outro ponto positivo desse framework é o fato de existir um pacote pronto e simples para conectar o client do Keycloak no servidor. O backend da aplicação consiste na aplicação que está na pasta “api”, raiz do projeto.

A configuração do Keycloak no backend consiste em dois arquivos, conforme mostrado na figura 5.6. Um para adição dos módulos que estão no arquivo `app.module.ts` no caminho raiz do projeto `/api/src`, e o outro é o arquivo `keycloak.json` onde são configuradas as informações do client na aplicação do Keycloak. No arquivo `app.modules.ts`, o pacote `'nest-keycloak-connect'` se encarrega de todo o trabalho árduo de configuração e troca de informações entre as plataformas do nestjs e Keycloak. O que é passado para ele são as informações que estão contidas no arquivo de configuração, como a URL usada para autenticação, realm, client, a palavra secreta do client e o tipo de acesso que, nesse caso, é apenas utilizando requisitos que utilizem a autenticação do tipo “bearer”.

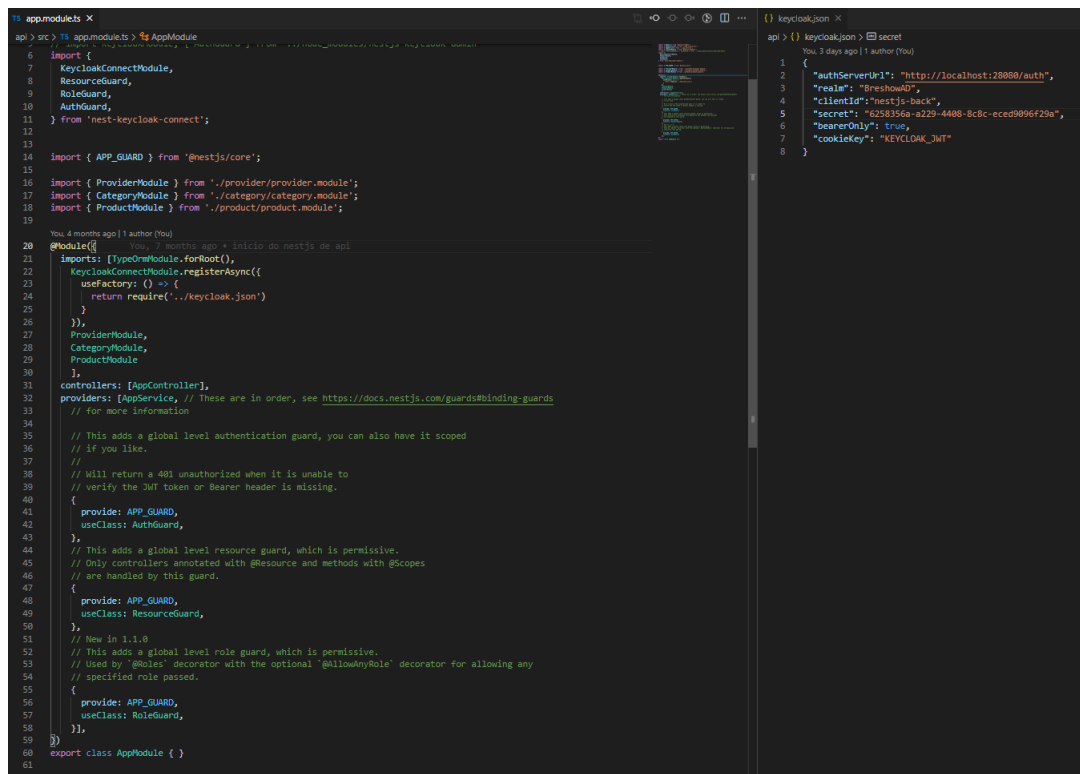


Figura 5.6: Arquivos relacionados as configurações do Keycloak no backend

A segunda parte trata da configuração dos guardiões do nestjs, sendo classes que se encarregam de bloquear uma requisição na hipótese das suas informações não estarem de acordo com o padrão estipulado pela aplicação do Keycloak. O pacote do Keycloak possui três guardiões que bloqueiam as requisições de acordo com as configurações do client e do usuário, onde são instanciados em uma lista do objeto “providers” no arquivo app.module.ts. O mais utilizado é a classe AuthGuard, onde bloqueia a requisição caso o bearer ou o jwt esteja incorreto, retornando o erro http 401, que significa não autorizado. Vale ressaltar que, uma vez adicionado esses guardiões, todos os endpoints do servidor terão os padrões ativados e apenas serão desativados se forem adicionados um decorador neles.

Outro ponto importante de ser observado é a aplicação de teorias e padrões de software utilizados na arquitetura do servidor. A estrutura montada seguiu a proposta da Clean Architecture e, para isso, utilizou-se uma técnica de mapeamento de objeto relacional, do inglês Object Relational Mapper (ORM), para desacoplar a dependência da aplicação a algum determinado tipo de banco de dados. A nível de código, foi utilizada a biblioteca TypeORM e Migração para fazer o mapeamento

das entidades e a criação, ou destruição, de tabelas apenas iniciando um script, respectivamente. A própria biblioteca se encarrega de traduzir o código para a língua nativa do banco de dados escolhido que, no caso do presente trabalho, fora o PostgreSQL. Além disso, introduziu-se a ideia de objeto de transferência de dados, em inglês Data Transfer Object (DTO), para transferir os dados que vem das requisições do frontend para as entidades no servidor.

Além de aplicar o ORM e o DTO, relacionados a camada de adaptadores das interfaces na abordagem da Clean Arctecture, o código foi separado por regra de negócio. Sendo assim, tanto a entidade de vendas, quanto a categoria e o fornecedor têm a sua própria camada de negócio e modelos bem definida. A nível de código, por exemplo, a pasta da categoria (figura 5.7) apresenta o arquivo de “controller”, onde são feitas as requisições que estão diretamente ligadas ao arquivo de “service” que, por sua vez, necessita da “entity” para fazer suas alterações ou regras de negócio da aplicação. O fluxo apresentado acima mostra um nível maior de dependência que a aplicação tem com as entidades, do que para periféricos e adaptadores. Assim, é possível observar que o código e sua arquitetura estão alinhados com os princípios da Clean Architecture.

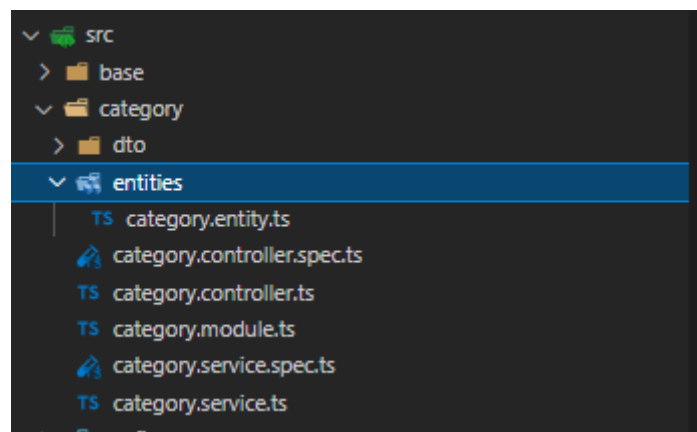


Figura 5.7: Estrutura da entidade de categoria do servidor

Além desse padrão de software, incorporou-se a estrutura Folder-by-Feature, já que é um dos padrões usados pelo próprio Nestjs na criação de suas pastas e possui a manutenção facilitada, pois tudo que for relacionado àquela entidade ou categoria, está dentro do seu próprio repositório, como mostrado na figura 5.7.

Em relação aos Endpoints, criaram-se cinco para cada entidade, sendo elas: Criar entidade, listar todas as entidades, trazer informações de uma específica, editar e deletar. Apenas a entidade de vendas apresenta uma regra a mais de negócio devido a cálculos voltados para o valor total final da venda, principalmente por conta do fornecedor, conforme mostrado na figura 5.8.

```
async update(id: string, updateProductDto: UpdateProductDto) {
  const provider = await this.providerRepository.findOne(updateProductDto.providerId);
  const priceFinal = (1 - provider.perPrice/100.0) * updateProductDto.quantity * updateProductDto.price;

  return this.productRepository.save(<Product>{
    id: id,
    name: updateProductDto.name,
    price: updateProductDto.price,
    quantity: updateProductDto.quantity,
    priceTotal: updateProductDto.price * updateProductDto.quantity,
    priceFinal: priceFinal,
    comment: updateProductDto.comment,
    dateIn: updateProductDto.dateIn,
    provider: <Provider>{
      id: updateProductDto.providerId
    },
    category: <Category>{
      id: updateProductDto.categoryId
    }
  });
}
```

Figura 5.8: Código da regra de negócio de atualização de uma entidade de venda que está no arquivo “product.service.ts”

O código do servidor da aplicação não se apresentou tão complexo para conclusão. Foram observados alguns problemas com bugs na escrita do código do arquivo de Migração, pois alguns comandos não funcionaram na hora da tradução interna de código para linguagem de banco. Por último, adversidades se fizeram presentes na rota e SSL no momento em que foram postos em produção dentro do contêiner, pressionando a aplicação web a funcionar fora do Docker.

5.2.3 Análise dos dados

Inicialmente, a aplicação de análise de dados seria uma parte da aplicação web na qual uma aba mostra gráficos com uma visão macro das vendas daquele usuário. Porém, optou-se utilizar uma ferramenta já conhecida no mercado com esse propósito. Após breve pesquisa, fora escolhido o Grafana. A vantagem de ser código aberto, ter uma gama de extensões gratuitas e ser utilizada por grandes empresas, tais como Siemens e Paypal, foram pontos importantes para a escolha da ferramenta. Outro ponto vantajoso é a existência de uma imagem dela pronta no DockerHub, permitindo agregar sua imagem à composição de contêineres.

No arquivo de composição do docker (`docker-compose.yml`) na raiz do projeto, a porção responsável pelo grafanar é o código:

```
grafana:
  image: grafana/grafana:7.3.6
  ports:
    - 8084:3000
  volumes:
    - ./grafana:/var/lib/grafana
  networks:
    - postgres
```

O docker instancia uma imagem do Grafana, onde salva todas as informações importantes na pasta `grafana` do projeto e inicia a aplicação na porta 8084 do localhost. A aplicação monta os gráficos a partir dos dados vindos diretamente do banco de dados, ou seja, o código por trás dos dados que aparecem em um painel são linguagens de banco de dados como o SQL. O banco é configurado na aba de “Datasource”, que, no caso, foi o postgresql, aplicando-se informações como host, usuário e senha.

Cada gráfico fica dentro de um painel, podendo ser customizado o tipo de gráfico, layouts e eixos. Por exemplo, a figura 5.9 mostra o gráfico de vendas totais e lucros por dia e, nele, podemos customizar o título, cor de cada linha ou área no gráfico, mostrando os números nos eixos e quais tipo de números são esses. No painel tem o símbolo da moeda brasileira, caso algum eixo seja representado em moeda. No próprio painel pode-se escrever o código para trazer dados que desejam aparecer, e existem palavras reservadas para aplicar os filtros temporais, além de necessariamente precisar de, pelo menos, um campo no formato de data.

Além do gráfico de vendas ser série temporal, tem-se o gráfico de barras que representa a quantidade de produtos vendidos por dia, número estatístico que representa o número total de vendas naquele período do filtro e o gráfico em pizza que mostra a porcentagem de vendas por fornecedor, também com textos. Os gráficos estão exemplificados e seus códigos à mostra, respectivamente, nas figuras 5.10, 5.11, 5.12

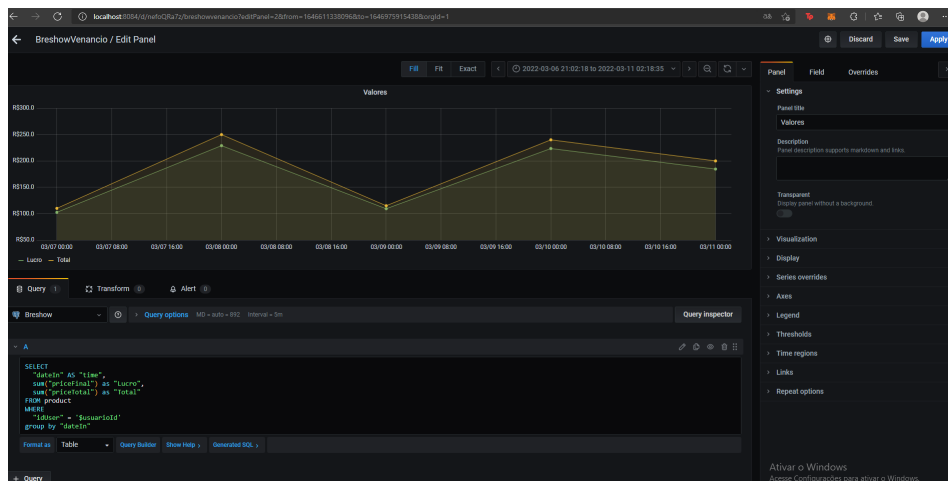


Figura 5.9: Painel de valores do dashboard de um usuário no Grafana

Após a configuração e criação de cada painel, é possível organizá-los dentro do dashboard do usuário e, no exemplo utilizado como teste, tem-se a seguinte visão geral mostrada na figura 5.13 a respeito das vendas no período de 08/03/2022 até 08/03/2022. Todos os dados vieram do banco de dados integrado com a aplicação web do projeto, tornando tanto o aplicativo de análise de dados, quanto a da aplicação web, isolados e sem interferência um do outro, apenas compartilhando a mesma base.

Infelizmente, caso um novo usuário seja criado na aplicação web, o mesmo deve ser feito no Grafana na parte de gerenciamento de usuários, pois o Keycloak não é compartilhado com o Grafana. No caso do dashboard, recomenda-se apenas duplicar algum dashboard existente e alterar a variável “usuários” no campo de configuração do dashboard, duplicando para o nome do novo usuário (figura 5.14).

É de suma importância que o administrador do Grafana crie usuários específicos para cada um da aplicação web e coloquem-os com apenas a visualização de seus respectivos dashboards, todavia, essa atividade fica a escolha do usuário. A versão atual presente no github possui pelo menos um dashboard padrão pronto, apenas necessitando alterar a variável “usuarios” para que ele traga os dados corretos nos gráficos.

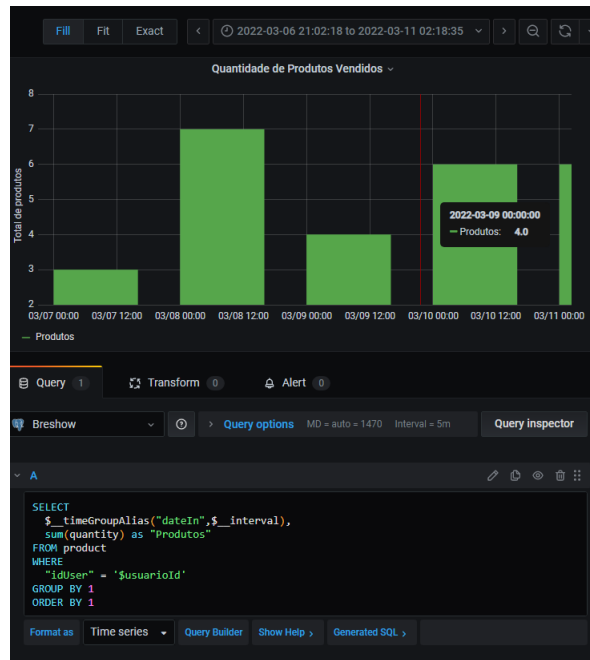


Figura 5.10: Painel de valores em barra do dashboard de um usuário no Grafana

Na parte do desenvolvimento não foram encontrados muitos obstáculos, pois a documentação do Grafana é simples e intuitiva, além da sua imagem no Docker ser estável na versão em que está. Montar os gráficos e as consultas à banco foram trabalhosas, mas mais fáceis que outras partes do projeto. A interface gráfica facilita a configuração com o banco de dados, sendo essa a única interação necessária com a aplicação.

5.3 Desafios

Ao longo do desenvolvimento, alteraram-se algumas ferramentas por conta de problemas de compatibilidade entre elas, além da curva de aprendizagem serem bem inclinadas ou longas demais. Foi o caso da escolha do frontend, inicialmente utilizado o React. Contudo, mudou-se para Angular 12 pelo fato da estrutura do backend, o Nestjs, ser baseado no Angular e, assim, facilitando o desenvolvimento da aplicação. Além disso, a biblioteca de integração do Keycloak para Angular 12 ser mais completa e documentada do que a de React. A escolha do gerenciamento de usuários seria o sistema Firebase da Google, porém apresenta alguns recursos que só são possíveis utilizar na versão paga. Por conta disso, escolheu-se um sistema com os recursos totalmente gratuitos, como o Keycloak.

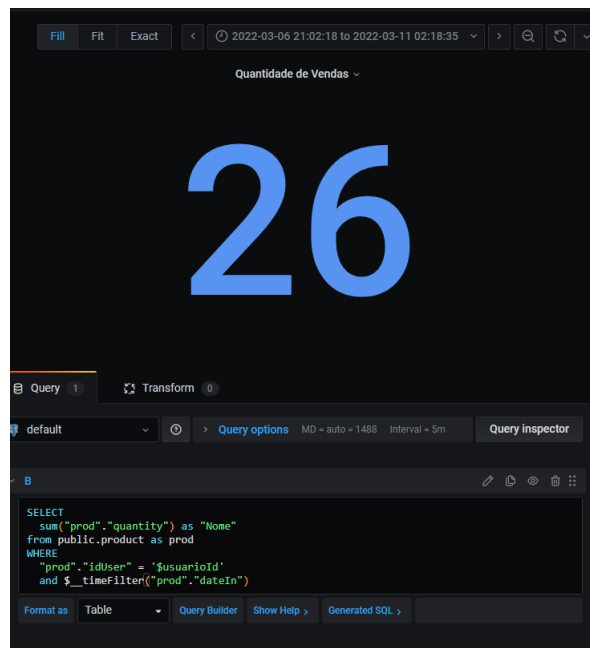


Figura 5.11: Painel de valores em estatística do dashboard de um usuário no Grafana

O desenvolvimento da aplicação como um todo foi relativamente rápida, pois a maioria das ferramentas que foram utilizadas apresentavam uma documentação clara, chegando a ter vídeos com exemplos práticos como, por exemplo, o Nestjs, Docker e Angular 12. Contudo, no caso do Keycloak, a documentação não é tão clara para alguém que detenha pouco conhecimento na área de segurança de rede e, gerando problemas na configuração de segurança entre o Keycloak e as outras aplicações. Por conta disso, a aplicação acabou passando do prazo estipulado do trabalho, necessitando algumas semanas a mais de desenvolvimento.

O sistema rodar em produção, ou seja, no seu estado final, foi outro empecilho aparente. Para que ele funcione da forma como almejado, era preciso que todos os módulos estivessem dentro de contêineres do Docker. Para isso, foi necessário a configuração das URLs dos micro serviços da aplicação web para conversarem entre eles e, por conta de protocolos de segurança da rede, descobriu-se que precisaria acrescentar uma camada a aplicação, causando no aumento do tempo de desenvolvimento. Dessa forma, o atual projeto roda apenas localmente na máquina onde fora instalado e o módulo da aplicação web precisou ficar fora do contêiner, dificultando a instalação do projeto como um todo.

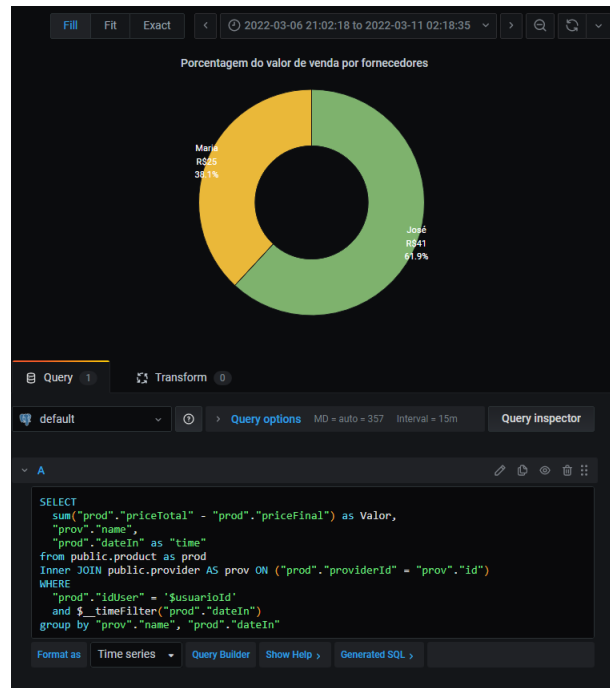


Figura 5.12: Painel de valores por fornecedor do dashboard de um usuário no Grafana



Figura 5.13: Dashboard completo das vendas de um usuário no Grafana

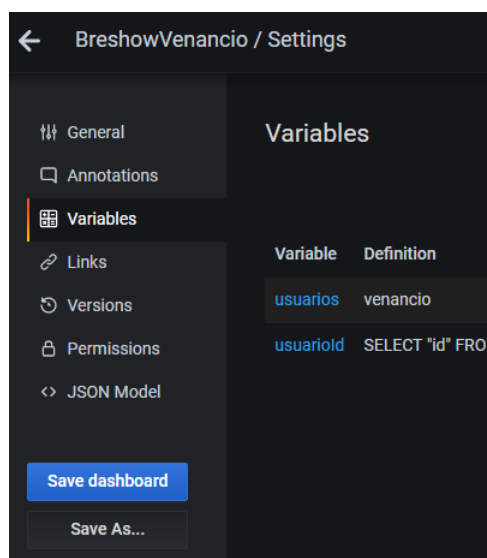


Figura 5.14: Aba de configuração de um dashboard no Grafana

Capítulo 6

Prova de Conceito

Neste capítulo, apresenta-se a implantação do projeto presente no Github, assim como seu funcionamento na visão do Usuário e do Administrador.

6.1 Implantação

Na fase de implantação dos módulos, é necessário um conhecimento prévio sobre Docker, Git, GitHub, NodeJs e comandos para bash, que é a linha de comando para o sistema Linux. A implantação consiste no funcionamento de todo o sistema e tem o intuito de ser simples para o indivíduo que irá fazê-la. Atualmente, o sistema funciona apenas de forma local e sua documentação está no próprio repositório do projeto, caso seja de interesse do implantador.

6.1.1 Docker

Inicialmente, precisa-se do Docker instalado na máquina, podendo ser encontrado e baixado no link <https://www.docker.com/products/docker-desktop>. O Docker é necessário, pois todos os módulos e configurações das aplicações já estão pré-configuradas nele e, dessa forma, torna a instalação menos complicada e permite que as aplicações estejam desacopladas do sistema operacional do computador.

6.1.2 Github

O próximo passo é a clonagem do repositório do projeto no link <https://github.com/VenancioIgrejas/BreShow>, ou rodar a seguinte linha de comando `git clone https://`

`github.com/VenancioIgrejas/BreShow.git` no prompt de comando(cmd) ou bash dependendo do sistema operacional. Caso não tenha o *Git* instalado, basta acessar o site <https://git-scm.com/downloads> e seguir o passo a passo de instalação atentando-se apenas para qual sistema operacional utiliza.

6.1.3 Npm

Por conta da aplicação web rodar em Node.js, é preciso que o gerenciador de pacotes também esteja instalado. Dessa forma, basta entrar no link <https://nodejs.org/en/download/> e baixar a versão mais recente. Uma vez instalado, deve-se rodar o comando “node -version” e “npm -version” em um *prompt de comando*. Se em ambos os casos aparecer a letra V e alguns números, significa que a instalação do npm foi um sucesso.

6.1.4 Projeto Local

A fase atual do projeto não foi feita para rodar em produção, pois ainda há algumas alterações de segurança e de rota a serem feitas, todavia, funciona perfeitamente em um ambiente local.

6.1.4.1 Composição do docker

Uma vez o Docker instalado, deve-se rodar o comando “*docker-compose up*” na raiz da solução para que os serviços sejam criados e configurados. Esse processo pode demorar um pouco, pois serão instanciados quatro serviços ao mesmo tempo. São eles: a imagem do Keycloak, do Grafana, do Angular12 e do NestJS.

6.1.4.2 Rodando a aplicação web

Primeiramente, é necessário esperar a composição do docker terminar as configurações dos serviços, levando em torno de 3 minutos. Após esse período, é possível rodar a aplicação web e, para isso, deve-se escrever o comando na linha de comando da raiz do projeto “*./installApp.sh*”. Esse comando irá instalar todas as dependências e os módulos a serem utilizados pelo cliente e servidor da aplicação

web, assim como configurações de compilação e tabelas de banco, tudo automatizado.

Após a instalação da aplicação, basta rodar o comando na linha de comando da raiz do projeto “*./startDevApp.sh*” que irá iniciar os dois micro-serviços da aplicação web, estando prontos para o uso. Caso não tenha alterado algum arquivo de ambiente, a aplicação estará rodando no caminho <http://localhost:4200/>

6.2 Usuário

Nesse ponto, considere o usuário o indivíduo que irá colocar as informações sobre fornecedores, categoria dos produtos e as vendas, lembrando da obrigatoriedade da informação sobre qual categoria aquele produto se encontra e qual o seu fornecedor. Além do mais, cada usuário tem acesso apenas às informações dos seus produtos. Já na parte de dashboard, o controle é feito pelo administrador.

Quando o usuário acessar o link <http://localhost:4200/>, será redirecionado para a tela de login do sistema, afinal o mesmo ainda não está logado. Conforme podemos observar na figura 6.1

6.2.1 Criação de Conta

Inicialmente, o usuário precisa criar uma conta. Para isso, basta clicar no texto em azul “*Register*” conforme aparece na figura 6.1. Ao clicar, será direcionado para uma tela de cadastro e adicionar informações pessoais tais como nome, e-mail, usuário, senha, dentre outros. Um exemplo está representado na figura 6.2

6.2.2 Tela principal

A figura 6.3 mostra a tela principal da aplicação web do projeto. Após a criação do usuário de exemplo, é possível visualizar no canto superior esquerdo as abas por onde é possível navegar, marcadas de vermelho. No canto superior direito circulado de azul, estão as informações do nome do usuário e um botão para se deslogar da aplicação. Por fim, no centro, circulado de laranja, está um resumo sobre os

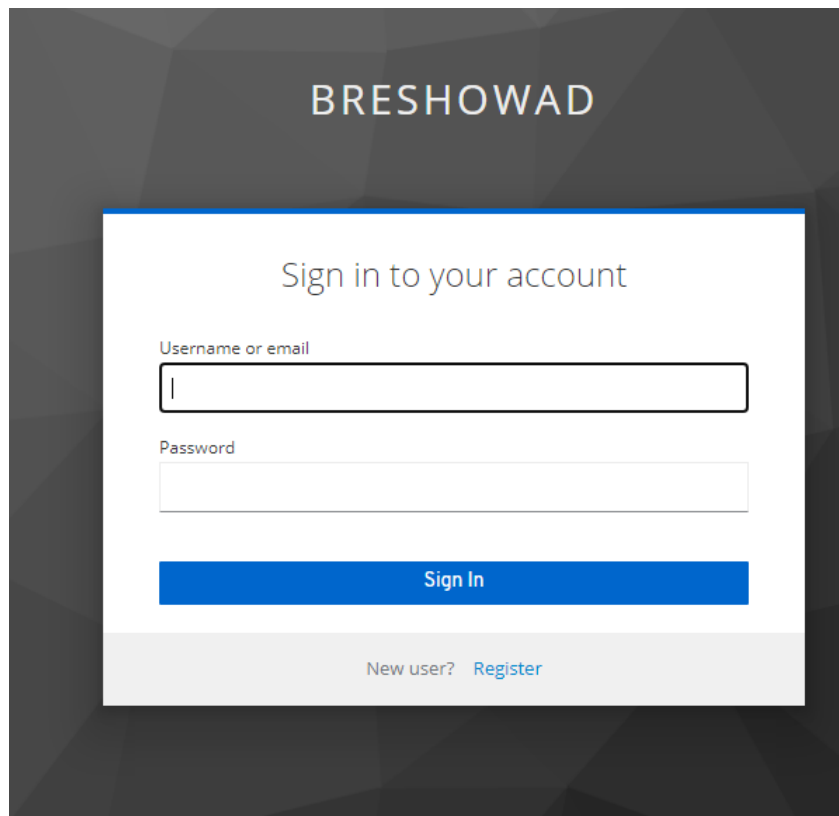


Figura 6.1: Página para entrar na aplicação web.

produtos, categorias e os fornecedores já cadastrados pelo usuário e seu intuito é uma rápida análise no âmbito micro das tabelas.

Para acessar as páginas relacionadas aos produtos vendidos, basta clicar no botão desejado na parte superior esquerda indicado com círculo vermelho.

6.2.3 Tela de Categoria

A tela de categoria apresenta uma tabela com a lista de todas as categorias adicionadas pelo usuário e um botão de adicionar logo acima da lista para criar um registro novo de categoria, conforme mostrado na figura 6.4.

O intuito da categoria é ser um texto livre, onde o usuário possa criar a categoria que quiser e não se prender às pré-configuradas ou as padronizadas, como, por exemplo, gênero, objeto e cor.

BRESHOWAD

Register

First name
Venâncio

Last name
Igrejas

Email
venancio@teste.br

Username
venancio

Password
.....

Confirm password
.....

[« Back to Login](#)

Register

Figura 6.2: Página de cadastro da aplicação web.

6.2.3.1 Criando uma categoria

Ao clicar no botão de adicionar, uma janela flutuante aparecerá e o usuário deverá preencher com as informações relacionadas aos itens de vendas que futuramente serão adicionados em outra tela. A categoria serve para agrupar os produtos na hora de adicioná-los ao sistema. A janela flutuante apresenta apenas um campo chamado “Nome”, que significa o nome da categoria a ser criada. Na figura 6.5, está formando uma categoria chamada “blusa”. Após clicar em salvar, aparece a categoria criada na lista da tabela (figura 6.6)

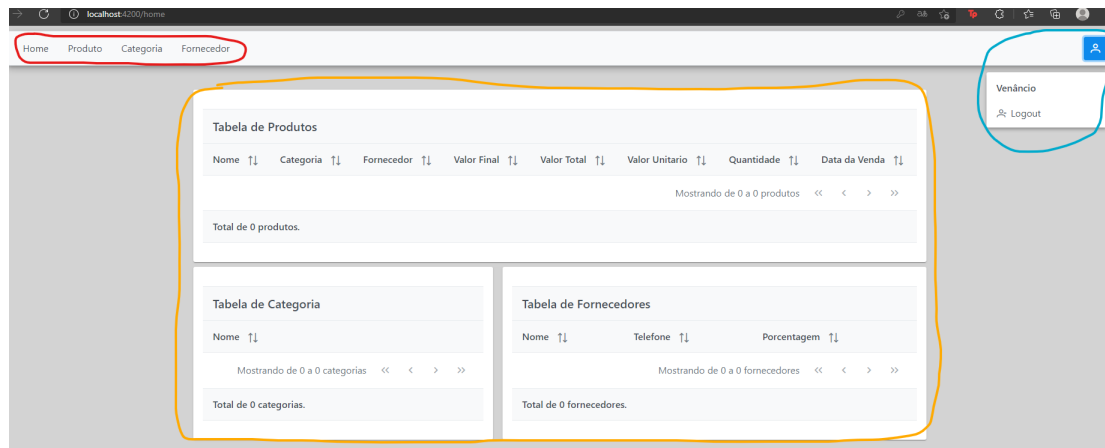


Figura 6.3: Página principal da aplicação web.

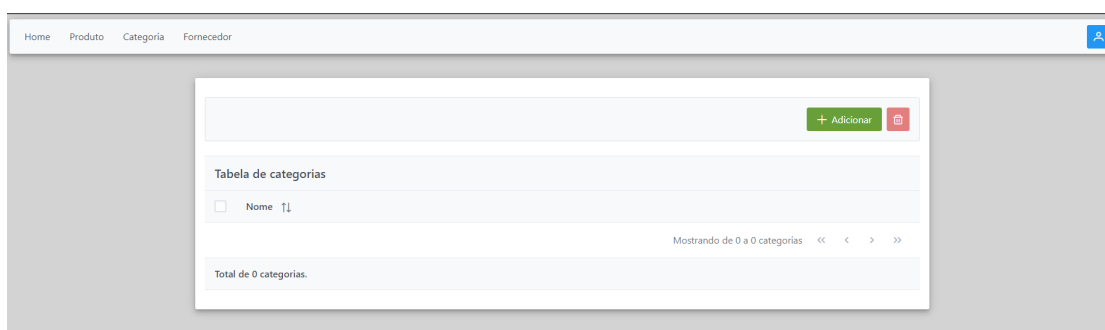


Figura 6.4: Página da categoria da aplicação web.

6.2.3.2 Editando ou deletando uma categoria

Para alterar o nome de uma categoria, é necessário clicar no botão verde redondo com símbolo de um lápis, localizado na mesma horizontal que o nome da categoria que deseja editar (figura 6.6). Ao clicar, aparecerá a mesma janela flutuante que a de adição, com o campo "nome" preenchido, bastando alterar e salvar. Após esses passos, a tela irá recarregar com a alteração completa.

Para deletar a categoria, basta clicar no botão redondo laranja que está no mesmo nível que o nome da categoria desejada (figura 6.6). Ao clicar, a categoria e *todos os registros relacionados a ela* serão deletados do sistema e a tela será recarregada com a nova alteração.

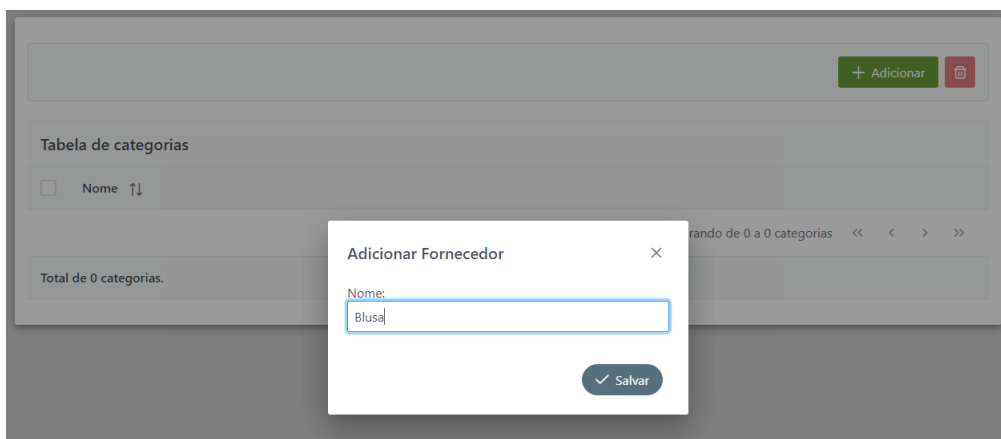


Figura 6.5: Janela flutuante da categoria.

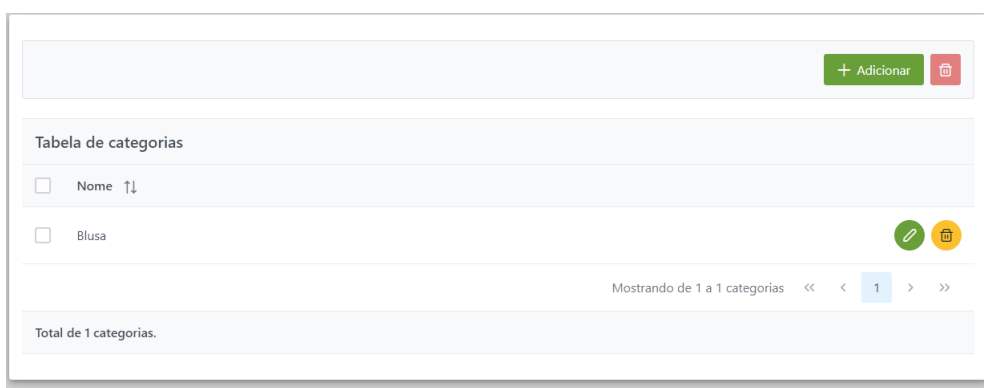


Figura 6.6: Tabela da categoria listando a Blusa após ter sido criada.

6.2.4 Tela de Fornecedor

A tela de fornecedor assemelha-se com a categoria na estrutura, como observado na figura 6.7. O que muda são os campos na janela flutuante ao adicionar ou editar um fornecedor.

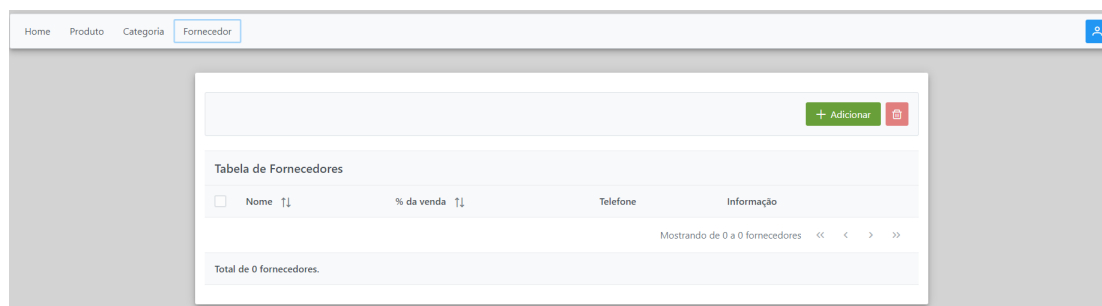


Figura 6.7: Página dos fornecedores na aplicação web.

6.2.4.1 Criando, Editando e deletando um fornecedor

Uma vez que o processo de criação, edição e remoção de um fornecedor é parecido com a de categoria, será mostrado apenas o diferente. No caso da janela flutuante, observa-se na figura 6.8 que existem outros campos a serem preenchidos, tais como o nome do fornecedor, a porcentagem do valor da venda que vai para o fornecedor, seu telefone para contato e uma breve informação. Após o preenchimento obrigatório das informações, deve-se apertar em “Salvar” caso deseje adicionar na aplicação, ou no “X” no canto superior a direita para fechar a janela.

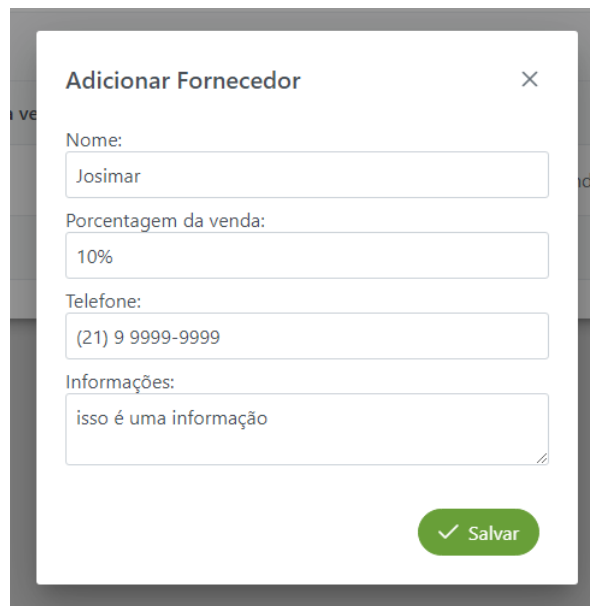
A imagem mostra uma janela flutuante intitulada "Adicionar Fornecedor" com um ícone de fechar (X) no canto superior direito. O formulário contém quatro campos de entrada: "Nome:" com o valor "Josimar", "Porcentagem da venda:" com o valor "10%", "Telefone:" com o valor "(21) 9 9999-9999", e "Informações:" com o texto "isso é uma informação". No canto inferior direito, há um botão verde com um ícone de checkmark e o texto "Salvar".

Figura 6.8: Janela flutuante da tela de fornecedores.

Após uma adição, ou edição, a lista de fornecedores será atualizada na página principal do fornecedor, ficando disponíveis as informações alteradas conforme a figura 6.9. No exemplo mostrado, observa-se que o "Josimar" foi criado e suas informações mais importantes estão a mostra para uma rápida análise.

6.2.5 Tela de Produtos

A tela de produtos se assemelha com a da categoria na estrutura, como pode ser percebido na figura 6.10. O que muda são os campos na janela flutuante ao adicionar ou editar um produto, e todo produto precisa estar relacionado a uma categoria e a um fornecedor obrigatoriamente.

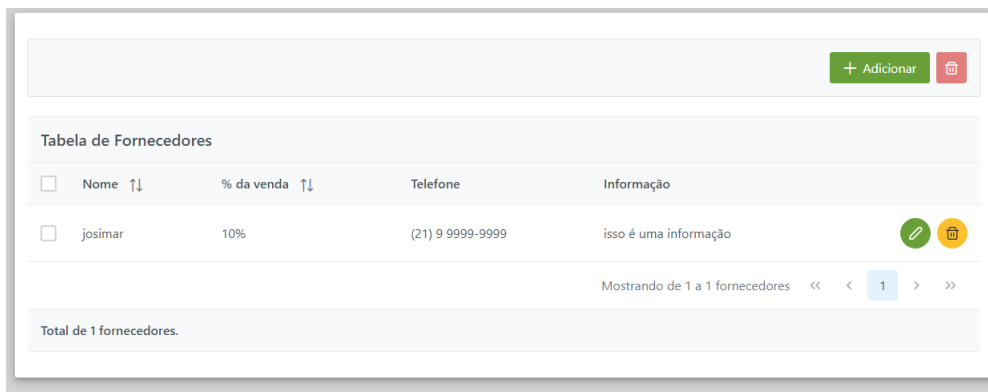


Figura 6.9: Tabela de fornecedores listando o novo fornecedor após ter sido criado.

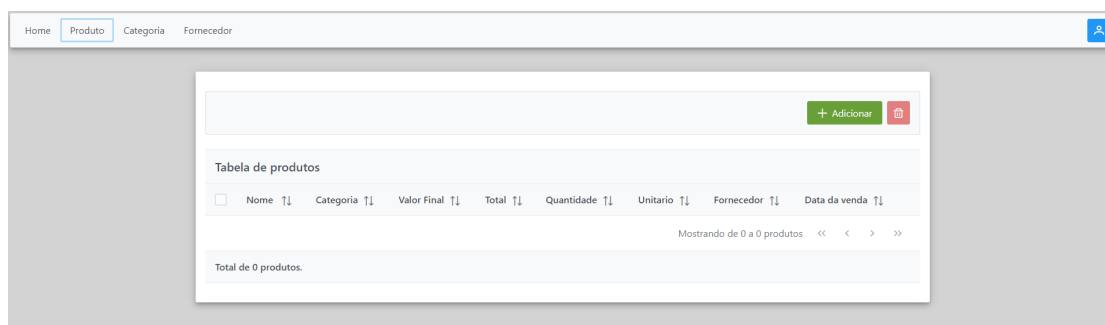


Figura 6.10: Página dos produtos na aplicação web.

6.2.5.1 Criando, Editando e deletando um produto

Como o processo de criação, edição e remoção de um produto é muito parecido com a de categoria. No caso da janela flutuante, é possível observar na figura 6.11 que existem outros campos a serem preenchidos, tais como o nome do produto, a categoria a qual ele pertence, qual o seu fornecedor, valor da venda por unidade, quantidade, data em que o produto foi vendido, ou a data que foi adicionado no sistema, e uma breve informação. Após o preenchimento obrigatório das informações, deve-se apertar em “Salvar” caso queira adicionar na aplicação, ou no “X” no canto superior a direita para fechar a janela.

Após adição ou edição, a lista de fornecedores será atualizada na página principal do produto e aparecerão as informações alteradas, conforme a figura 6.12. No exemplo mostrado, observa-se que a blusa “Lacoste (p)” foi criada e as suas informações mais importantes estão a mostra para uma rápida análise, além do fato de ter informações a mais como o valor total da venda (valor da venda x quantidade) e o valor final, sendo este o total menos a parte que irá pro fornecedor, mostrado na

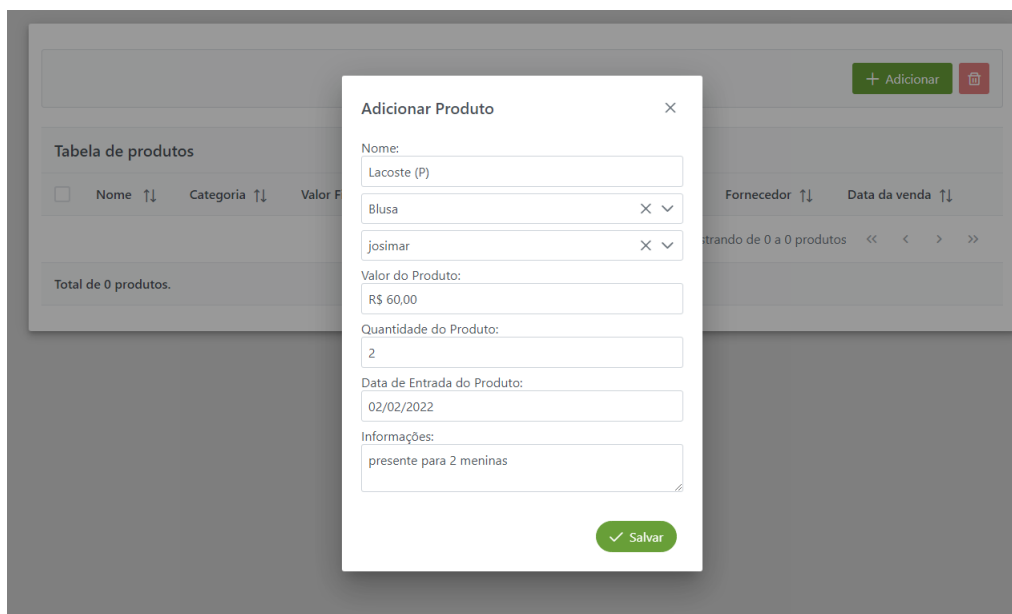


Figura 6.11: Janela flutuante da tela de produtos.

tabela com o nome de “Fornecedor”.

	Nome ↑↓	Categoria ↑↓	Valor Final ↑↓	Total ↑↓	Quantidade ↑↓	Unitário ↑↓	Fornecedor ↑↓	Data da venda ↑↓
<input type="checkbox"/>	Lacoste (P)	Blusa	R\$ 108,00	R\$ 120,00	2	R\$ 60,00	josimar	02/02/2022

Mostrando de 1 a 1 produtos

Total de 1 produtos.

Figura 6.12: Tabela de produtos listando o novo produto após ter sido criado.

Ao término de todas as adições dos dados, é possível seguir para a página inicial onde terá um resumo de todos os dados já inseridos pelo usuário, tais como fornecedores, categorias e produtos (figura 6.13). Essa visualização é apenas de forma micro. Para uma análise macro mais profunda, deve-se utilizar a aplicação de Dashboard discutida na próxima sessão.

Tabela de Produtos							
Nome	Categoria	Fornecedor	Valor Final	Valor Total	Valor Unitario	Quantidade	Data da Venda
Lacoste (P)	Blusa	josimar	R\$ 108,00	R\$ 120,00	R\$ 60,00	2	02/02/2022

Tabela de Categoria	
Nome	
Blusa	

Tabela de Fornecedores		
Nome	Telefone	Porcentagem
josimar	(21) 9 9999-9999	10%

Figura 6.13: Tela principal listando todas as informações adicionadas do usuário em tabelas distintas.

6.2.6 Tela de Dashboard

Utiliza-se o Grafana para análise da aplicação, sendo esta uma aplicação web de análise de código aberto multiplataforma e visualização interativa da web considerada o módulo de visualização de dados. Para acessá-lo, deve-se colocar o link <http://localhost:8084> no navegador e aguardar a tela de login. A figura 6.14, mostra a tela padrão de login. Ao logar com o usuário e senha criados pelo administrador, tem-se a tela principal com as informações macro que foram de criar na aplicação web (figura 6.15).

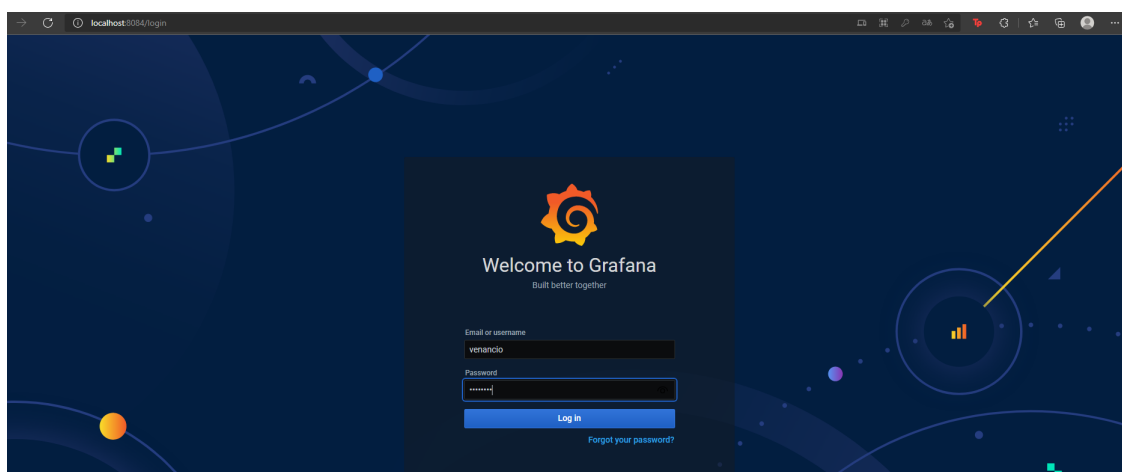


Figura 6.14: Tela de login do módulo de análise.

6.2.6.1 Tela principal

De acordo com a figura 6.15, a tela principal apresenta inúmeras informações sobre a saúde das vendas do empreendimento. A tela é subdividida em agrupamentos de painéis, envolvidos em laranja, onde é possível clicar e expandir cada uma delas. A primeira mostra três listas sobre informações gerais das vendas, categorias e fornecedores, igual a aplicação web (figura 6.16). A segunda está expandida por padrão e mostra informações macro como os valores totais e ganhos reais em um gráfico de série temporal. A quantidade de produtos vendidos está no mesmo tipo de gráfico e o valor quantitativo das informações foram adicionadas na aplicação. A terceira e quarta possuem formatos parecidos, mostrando o lucro total por categoria e o valor total do fornecedor em série temporal, gráfico de pizza e lista (figura 6.16).

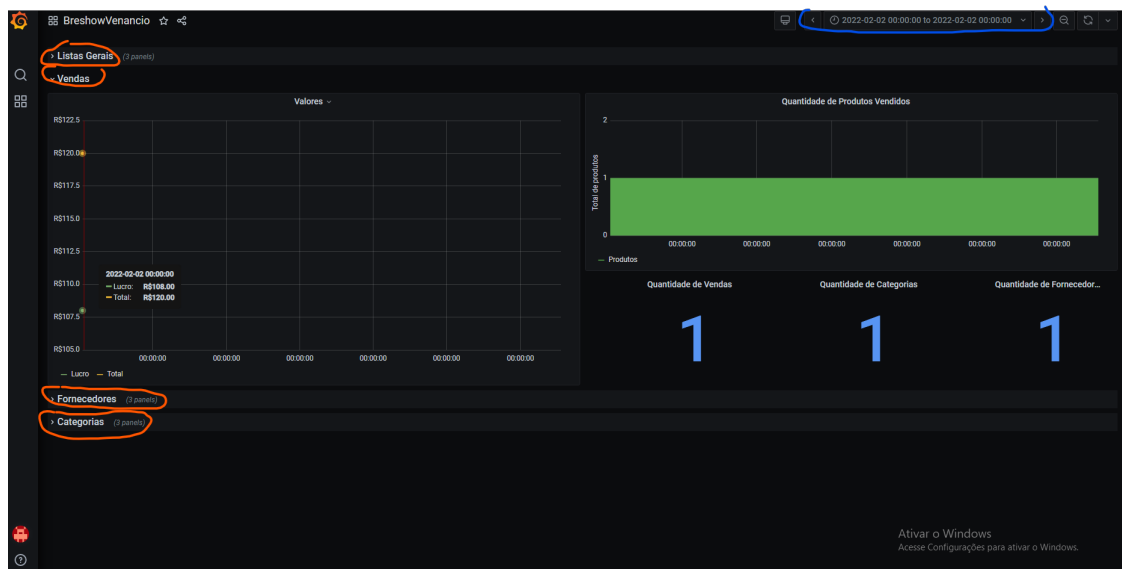


Figura 6.15: Tela principal do usuário.

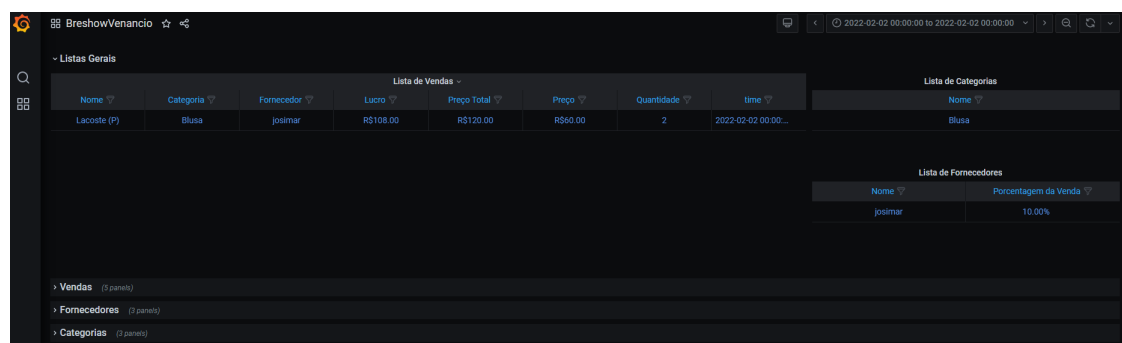


Figura 6.16: Painel de listas gerais expandido.

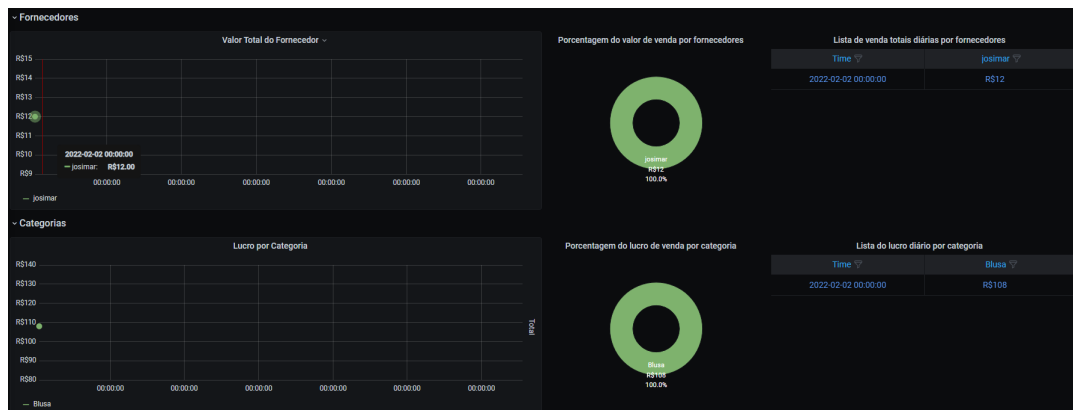


Figura 6.17: Painel de fornecedores e categorias expandido.

Além do mais, todos os dados e painéis estão interligados a um filtro temporal circulado de azul no topo da tela à direita na figura 6.15. Uma vez que esse filtro é alterado para o espaço temporal desejado, todos os valores são recalculados e filtrados para aquele intervalo, ou seja, o usuário pode pegar informações totais de um dia, mensais ou anuais. Além disso, o sistema é atualizado automaticamente, podendo ser configurado no botão que possui uma seta voltada para baixo. O tempo de atualização automática dos dados segue conforme a figura 6.18.

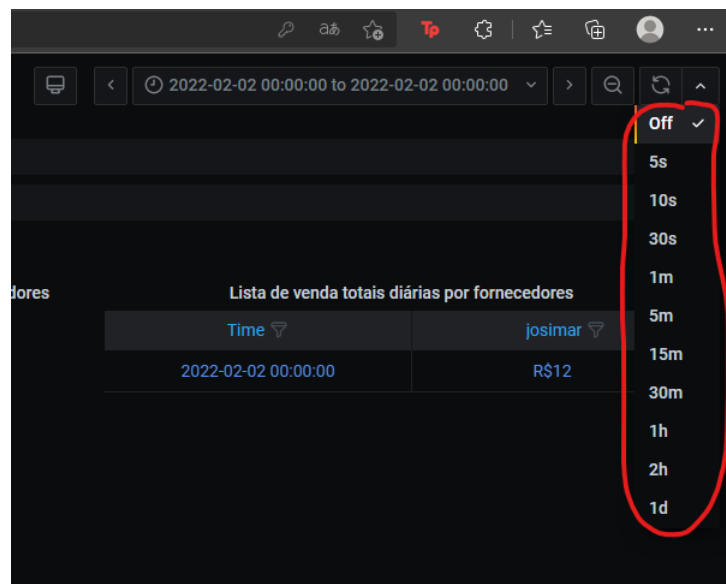


Figura 6.18: Painel de fornecedores e categorias expandido.

6.3 Administrador

Uma conta de administrador serve para gerenciar as contas dos usuários da aplicação web, utilizando-se o módulo de gerenciamento de usuários. Além disso, é responsável pela administração da criação e manutenção das contas dos usuários no módulo de análise e gráficos (dashboard).

6.3.1 Tela de Gerenciamento de Usuários

Utiliza-se o sistema Keycloak como estrutura principal do módulo de gerenciamento de usuários, um produto de software com a finalidade de coordenar identidades e acessos voltados para aplicativos. O administrador, para entrar no gerenciador, emprega uma senha e usuário padrão que está na documentação do módulo no Github.

Primeiramente, o indivíduo deve acessar o link `http://localhost:28080/` e clicar em “Administration Console” (figura 6.19) para entrar na tela de login da administração (figura 6.20). Após colocar os dados padrão de administrador na documentação do GitHub e clicar em entrar, aparecerá uma tela principal com várias opções de navegação numa coluna à esquerda, como mostrado na figura 6.22.

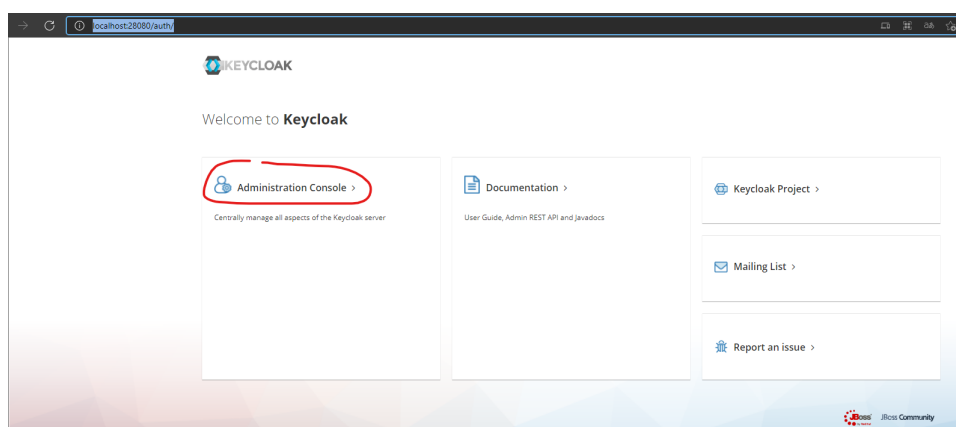


Figura 6.19: Pagina Inicial do Keycloak.

De acordo com a figura 6.22, o círculo em vermelho representa as opções de configurações do BreshowAD, tais como, configuração de segurança de rota, ajuste dos acessos de aplicações externas, roles e grupos. O círculo azul caracteriza a parte

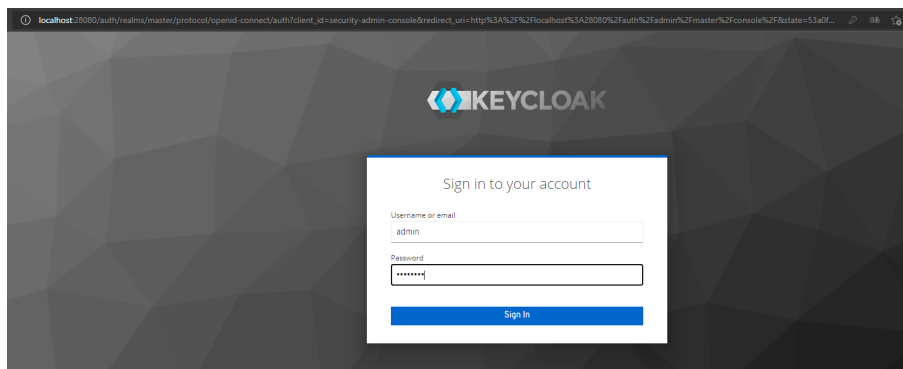


Figura 6.20: Pagina de login da área de administração.

voltada para o gerenciamento de usuários do módulo, onde é possível visualizar informações dos usuários, assim como ter o controle para fazer o logout de aplicações, resetar o login e, até mesmo, criar uma nova senha. É válido ressaltar que a segurança é de suma importância, portanto não existe a possibilidade do administrador ver a senha que o usuário colocou, apenas resetá-la ou criar uma nova. O círculo amarelo corresponde a aba de “Users” e contém o usuário que foi criado no subtópico acima com as respectivas informações colocadas pelo usuário.

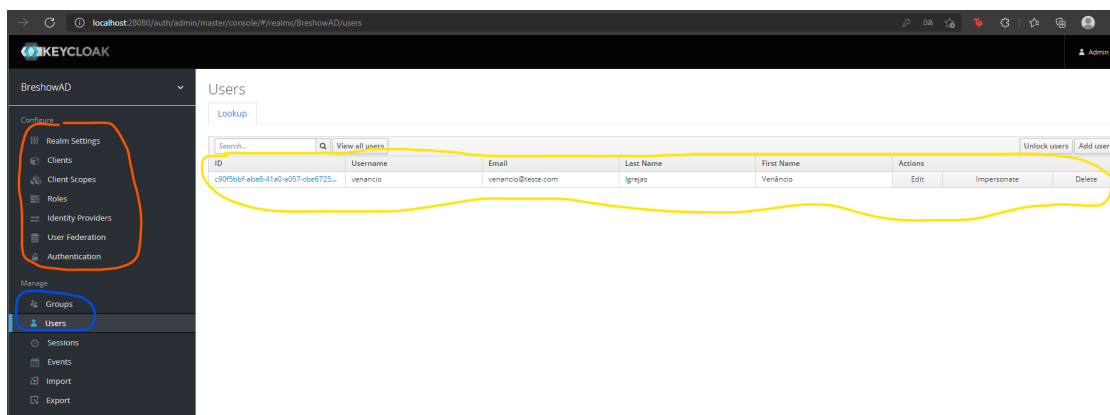


Figura 6.21: Pagina Principal do Keycloak.

Ao clicar no usuário que aparece na figura 6.22, a tela daquele usuário é aberta com suas informações pessoais, onde é possível editar caso necessário. A parte de credenciais é responsável pelo reset da senha e seu gerenciamento (figura 6.23), os grupos de usuários aos quais ele pode pertencer e o campo de sessão (“Sessions”), que permite visualizar quais aplicações estão logadas e se estão ativas, além de poder encerrar a sessão, obrigando-o a se deslogar.

Figura 6.22: Tela de gerenciamento do usuário Venâncio.

Figura 6.23: Pagina de credenciais do usuário Venâncio.

6.3.2 Tela de Dashboard

Para a aplicação de dashboard, o administrador é importante para gerenciar o acesso de cada usuário em suas respectivas vendas, possuindo o acesso a todos os dashboard dos usuários. A criação de dashboard para um usuário novo é feito manualmente. Em contrapartida, basta o administrador copiar uma já existente e alterar a variável global do painel para o nome do usuário que deseja-se criar.

A tela de administrador do servidor (figura 6.24) é a área em que se criam os usuários para a aplicação de dashboard. Vale ressaltar que o usuário da aplicação web não é o mesmo que a do dashbord e, pra isso, o administrador deve criá-los nessa tela.

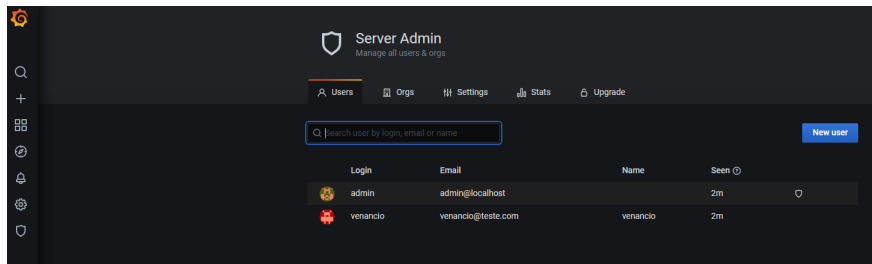


Figura 6.24: Área de gerenciamento de usuários do módulo de dashboard.

Supondo que o usuário e o painel foram criados através do método sugerido no início dessa sessão, é possível adicioná-lo apenas como leitor na tela de configuração do dashboard, e, assim, o mesmo só poderá analisar os próprios gráficos de vendas, conforme mostrado na figura 6.24.

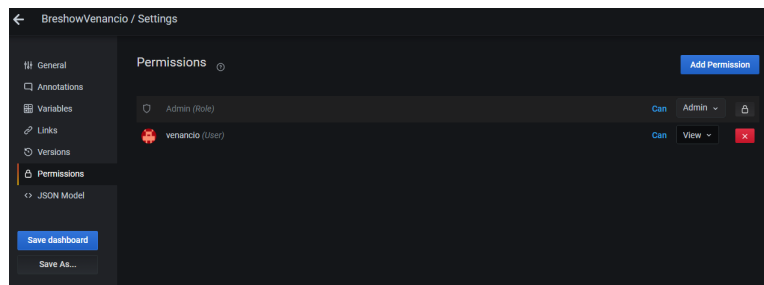


Figura 6.25: Área de configuração .

Capítulo 7

Conclusões e Trabalhos Futuros

O presente capítulo está dividido em duas sessões, onde a primeira contemplará a conclusão do trabalho, mostrando as dificuldades e problemas ao longo do seu desenvolvimento, assim como mudança nas estratégias de escolhas e um resumo. A segunda parte está voltada para os trabalhos futuros, onde serão abordados quais os próximos passos e possíveis melhorias no sistema para torná-lo mais robusto e intuitivo para o usuário.

7.1 Conclusão

Na fase em que se encontra, o projeto consegue ajudar o micro empreendedor a analisar suas vendas de forma macro através da aplicação de dashboard. Para a inserção de dados na base, a aplicação web está minimamente funcional e operacional, onde o usuário consegue fazer as operações básicas de inserção, alteração, visualização e deletar os dados das suas respectivas vendas. O administrador consegue orquestrar e gerenciar usuários da aplicação e do sistema como um todo.

Mesmo com todos os problemas ao longo do desenvolvimento, o projeto funciona da forma como foi proposto, sendo mais robusto e mais simples de ser utilizado que o Excel, porém não apresentando uma complexidade tão grande e desnecessária das ERPs no mercado voltadas para grandes empresas. Além disso, o projeto é totalmente gratuito e de código aberto. O usuário que for utilizá-lo precisará apenas preencher os dados dos fornecedores, categorias e vendas, sendo estes intuitivos na

plataforma web, e, no final, poderá ver toda a análise macro das suas vendas no módulo de dashboard. Apenas na hora da instalação do projeto e na sua configuração que podem haver dificuldades, porém existe a documentação para isso.

7.2 Trabalhos Futuros

Para os trabalhos futuros, existem algumas pendências e melhorias do projeto. Primeiramente, as pendências estão relacionadas a problemas apresentados ao longo do desenvolvimento e não foram possíveis resolvê-las dentro do prazo estipulado do trabalho. O fato do projeto não estar em produção por problemas nas ligações entre os serviços está relacionado a aplicação web, que são o frontend e o backend não estarem dentro de contêineres como as aplicações de banco de dados, análise e o gerenciador de usuários. Outro ajuste necessário seria a configuração de rotas seguras dentro do keycloak, o gerenciador de usuário, afinal, como o projeto roda localmente, não há problema de invasão pela rede, mas, ao colocar em produção, é de extrema importância a configuração correta das rotas.

Em relação as melhorias, é possível criar outros módulos no padrão de micro serviços como módulo de estoque, financeiro e funcionários. Pelo fato de ter sido explorado a descentralização de serviços no sistema, a escolha da arquitetura e framework se torna a gosto do indivíduo. Outro ponto a ser destacado é a aplicação do conhecimento de design da experiência do usuário na aplicação web, afinal, existe apenas um protótipo de tela que está somente funcional, mas sem polimento. Ainda falando sobre a aplicação web, é possível adicionar mais recursos como botão de exportação das tabelas em extensões de arquivos do Excel, alteração de um grupo de vendas que apresentem um mesmo campo, duplicação da venda para evitar reescrever campos iguais, dentre outros recursos.

Referências Bibliográficas

- [1] MARTIN, R. C., “The Clean Code Blog”, [urlhttps://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html](https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html), 2012, [Online; accessed 26-April-2022].
- [2] MARZALL, L. F., OTHERS, “Implantação de um sistema de gestão ERP gratuito em uma empresa de pequeno porte com foco na melhoria de indicadores de desempenho da produção”, , 2016.
- [3] FERNANDES, D. L., “Implementação de uma Ferramenta de Gestão de Projetos numa Consultora”, , 2017.
- [4] UTAMI, S. S., SUSILO, H., RIYADI, “Analisis Penerapan Enterprise Resource Planning (ERP)”, *Jurnal Administrasi Bisnis (JAB)*, v. 33, pp. 65–170, Abril 2016.
- [5] PAHL, C., “Containerization and the paas cloud”, *IEEE Cloud Computing*, v. 2, n. 3, pp. 24–31, 2015.
- [6] SATYANARAYANA, S., “Cloud computing: SAAS”, *Computer Sciences and Telecommunications*, , n. 4, pp. 76–79, Dezembro 2012.
- [7] MARTIN, R. C., “The Principles of OOD”, [urlhttp://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod](http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod), 2005, [Online; accessed 22-April-2022].
- [8] OLORUNTOBA, S., “SOLID: The First 5 Principles of Object Oriented Design”, [urlhttps://www.digitalocean.com/community/conceptual_articles/solid-the-first-five-principles-of-object-oriented-design](https://www.digitalocean.com/community/conceptual_articles/solid-the-first-five-principles-of-object-oriented-design), 2020, [Online; accessed 22 – April – 2022].

- [9] NAIDU, D., “SOLID Principles In C”, url[https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/:text=SOLID%20design%20principles%20in%20C%23,Dependency%20Inversion%20Principle%20\(DIP\).](https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/:text=SOLID%20design%20principles%20in%20C%23,Dependency%20Inversion%20Principle%20(DIP).), 2021, [Online; accessed 22-April-2022].
- [10] PAIXAO, J. R. D., “O que é SOLID: O guia completo para você entender os 5 princípios da POO”, url<https://medium.com/desenvolvendo-com-paixao/o-que-%C3%A9-solid-o-guia-completo-para-voc%C3%AA-entender-os-5-princ%C3%ADpios-da-poo-2b937b3fc530>, 2019, [Online; accessed 22-April-2022].
- [11] DEACON, J., “Model-view-controller (mvc) architecture”, *Online*/[Citado em: 10 de março de 2006.] <http://www.jdl.co.uk/briefings/MVC.pdf>, v. 28, 2009.
- [12] HIGOR, “Introdução ao Padrão MVC”, url<https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>, 2013, [Online; accessed 26-April-2022].
- [13] HERNANDEZ, R., “The Model View Controller Pattern – MVC Architecture and Frameworks Explained”, url<https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>, 2021, [Online; accessed 26-April-2022].
- [14] MARTIN, R. C., GRENNING, J., BROWN, S., *et al.*, *Clean architecture: a craftsman’s guide to software structure and design*, n. s 31. Prentice Hall, 2018.
- [15] FONSECA, E. D., “Introdução a Clean Architecture”, url<https://imasters.com.br/back-end/introducao-clean-architecture>, 2018, [Online; accessed 26-April-2022].
- [16] SAUBER, S., “Feature Folder Structure in ASP.NET Core”, url<https://scottsauber.com/2016/04/25/feature-folder-structure-in-asp-net-core/:text=A%20feature%20folder%20structure%20is,poor%20man’s%20feature%20folder%20structure>, 2016, [Online; accessed 26-April-2022].
- [17] SHEHAB, E., SHARP, M., SUPRAMANIAM, L., *et al.*, “Enterprise resource planning: An integrative review”, *Business process management journal*, , 2004.

- [18] KIM, H., BOLDYREFF, C., OTHERS, “Open source ERP for SMEs.”, , 2005.
- [19] TERMINANTO, A., HIDAYANTO, A. N., “Implementation and configurations open source ERP in ecommerce module (A case study on SME)”. In: *8th International Conference on Industrial Engineering and Operations Management, IEOM 2018*, pp. 1224–1233, IEOM Society, 2018.
- [20] WEBERP, “WebERP”, url<https://www.weberp.org/>, 2022, [Online; accessed 21-June-2022].
- [21] COMPIERE, “Compiere”, url<http://www.compiere.com/products/product-demos/>, 2022, [Online; accessed 21-June-2022].
- [22] JFIRE, “Jfire”, url<https://web.archive.org/web/20100910033244/http://www.jfire.net/>, 2022, [Online; accessed 21-June-2022].
- [23] ERP5, “ERP5”, url<https://www.erp5.com/>, 2022, [Online; accessed 21-June-2022].
- [24] ODOO, “Odoo”, url<https://www.odoo.com/>pt_BR, 2022, [Online; accessed 21-June-2022].
- [25] ACCOUNTING, F., “Front Accounting”, url<https://frontaccounting.com/>, 2022, [Online; accessed 21-June-2022].
- [26] OFBIZ, “OFBiz”, url<https://ofbiz.apache.org/>, 2022, [Online; accessed 21-June-2022].
- [27] JPIRE, “JPIRE”, url<http://www.jfire.net/>, 2022, [Online; accessed 21-June-2022].
- [28] XTUPLE, “Xtuple”, url<https://www.xtuple.com/>, 2022, [Online; accessed 21-June-2022].
- [29] LIMANTARA, N., JINGGA, F., “Open source ERP: ODOO implementation at micro small medium enterprises:(A case study approach at CV. XYZ in module purchasing and production)”. In: *2017 International Conference on Information Management and Technology (ICIMTech)*, pp. 340–344, IEEE, 2017.

- [30] MIRAGEM, B., “A Lei Geral de Proteção de Dados (Lei 13.709/2018) e o direito do consumidor”, *Revista dos Tribunais*, v. 1009, 2019.
- [31] GITHUB, “GitHub”, url<https://github.com/>, 2022, [Online; accessed 26-April-2022].
- [32] DOCKER, “Docker”, url<https://www.docker.com/>, 2022, [Online; accessed 26-April-2022].
- [33] “PostgreSQL”, url<https://www.postgresql.org/>, 2022, [Online; accessed 26-April-2022].
- [34] DOCKER, “Docker Hub”, url<https://hub.docker.com/>, 2022, [Online; accessed 26-April-2022].
- [35] HEAT, R., “Keycloak”, url<https://www.keycloak.org/>, 2022, [Online; accessed 26-April-2022].
- [36] AUTH0, “Jwt”, url<https://jwt.io/>, 2022, [Online; accessed 26-April-2022].
- [37] MYSLIWIEC, K., “NestJs”, url<https://nestjs.com/>, 2022, [Online; accessed 26-April-2022].
- [38] INFORMATICS, P., “primeNG”, url<https://www.primefaces.org/primeng/>, 2021, [Online; accessed 26-April-2022].
- [39] GOOGLE, “Angular”, url<https://angular.io/>, 2022, [Online; accessed 26-April-2022].
- [40] “TypeORM”, url<https://typeorm.io/>, 2022, [Online; accessed 26-April-2022].
- [41] LABS, G., “Grafana”, url<https://grafana.com/>, 2022, [Online; accessed 26-April-2022].