**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**NÚCLEO DE EDUCAÇÃO A DISTÂNCIA

Pós-graduação *Lato Sensu* em Desenvolvimento Web Full Stack

**Adjamir de Moura Galvão**

**SISTEMA PARA CONTROLE DE LOJAS DE PEQUENO PORTE**

Belo Horizonte

2023

**SUMÁRIO**

[1. Apresentação 4](#_Toc128410894)

[1.1. Contexto 4](#_Toc128410895)

[1.1.2. Mapas de empatia 4](#_Toc128410896)

[1.1.2. Business model canvas 7](#_Toc128410897)

[1.2. Público alvo 8](#_Toc128410898)

[1.3. Requisitos 8](#_Toc128410899)

[2.Modelagem 11](#_Toc128410900)

[2.1. Diagrama de casos de uso 11](#_Toc128410901)

[2.2. Atores 11](#_Toc128410902)

[2.3. Detalhamento dos casos de uso 11](#_Toc128410903)

[2.4. Projeto de interface 11](#_Toc128410904)

[2.4.1. *Wireframes* 12](#_Toc128410905)

[2.4.2. Protótipo interativo 17](#_Toc128410906)

[2.5. Modelo de dados 18](#_Toc128410907)

[3. Projeto 20](#_Toc128410908)

[3.1. Arquitetura de *software* 20](#_Toc128410909)

[3.1.1. Diagramas *C4 model* 20](#_Toc128410910)

[3.1.1.1. Nível 1 - contexto 21](#_Toc128410911)

[3.1.1.2. Nível 2 - contêiner 21](#_Toc128410912)

[3.1.1.3. Nível 3 - componentes 22](#_Toc128410913)

[3.2.1. Definição do framework de trabalho 24](#_Toc128410914)

[3.2.1.1. Framework front-end 25](#_Toc128410915)

[3.2.1.2. Framework back-end 26](#_Toc128410916)

[3.2.1.3. Persistência 26](#_Toc128410917)

[3.2. Arquitetura da informação 27](#_Toc128410918)

[4. Testes 28](#_Toc128410919)

[5. URLs 28](#_Toc128410920)

[5.1. Aplicação web 28](#_Toc128410921)

[5.2. Repositório código-fonte e documentação 28](#_Toc128410922)

[5.3. Especificação da API Swagger/OpenAPI 28](#_Toc128410923)

[5.4. Vídeo de apresentação do trabalho 28](#_Toc128410924)

[REFERÊNCIAS 29](#_Toc128410925)

# 1. Apresentação

## 1.1. Contexto

Pequenas lojas/comércios precisam de uma infraestrutura que dê apoio ao controle de estoque e custos dos produtos. Além disso, precisam fazer o registro de suas vendas. Estas informações facilitam a formação do preço de venda dos produtos, otimização das compras e, consequentemente, a lucratividade da empresa.

O atendimento aos clientes ocorre na loja física e o registro de vendas pode ser feita simultaneamente pelo caixa tradicional e vendedores. Desta forma é possível minimizar a formação de longas filas para o pagamento, agilizando o atendimento.

Avaliando o mercado, existem diversos produtos para o gerenciamento de lojas, como por exemplo, o Totvs varejo lojas - linha protheus[[1]](#footnote-1). Entretanto, estas soluções apresentam um alto custo de aquisição/manutenção. Além disso, normalmente não se adequam ao contexto de pequenos negócios cujos donos desejam soluções simplificadas e de baixíssimo custo de manutenção.

Mapas de empatia e *Business Model Canvas (BMC)* são ferramentas complementares que podem ser usadas juntas durante o processo de *Design Thinking*. Sendo assim, com o objetivo de entender melhor as necessidades, desejos e expectativas dos usuários em relação ao produto, foram desenvolvidos os mapas de empatia para os diferentes públicos alvo do sistema e *BMC* do negócio.

## 1.1.2. Mapas de empatia

O mapa de empatia ajuda a entender as necessidades e desejos dos clientes, funcionários e outros *stakeholders* em relação a um produto ou serviço. Ele pode ser usado para explorar os aspectos emocionais, sociais e comportamentais das pessoas em relação a um produto ou serviço, permitindo que sejam criadas soluções que atendam às suas necessidades. As figuras a seguir apresentam os mapas de empatia do negócio.

**Figura 1 – Mapa de empatia para o usuário gestor de estoque**



**Figura 2 – Mapa de empatia para o usuário vendedor**

**Figura 3 – Mapa de empatia para o usuário cliente**



**Figura 4 – Mapa de empatia para o usuário dono do negócio**

### 1.1.2. Business model canvas

O *Business Model Canvas* *(BMC)* é uma ferramenta visual que permite descrever, avaliar e iterar o modelo de negócios de uma empresa ou produto. Ele fornece informações sobre a proposta de valor da empresa, os segmentos de clientes, os canais de distribuição, as fontes de receita, entre outros aspectos importantes para a construção de uma solução.

Suas informações foram utilizadas para entender e mapear a lógica do negócio identificando os custos envolvidos em fornecer um sistema que atenda às necessidades e desejos dos usuários identificados no mapa de empatia. Logo, o *BMC* ajudou na tomada de decisão sobre o que construir e como construir. A figura abaixo apresenta o *BMC* do negócio.



**Figura 5 - *Business Model Canvas***

## 1.2. Público alvo

Na seção anterior identificamos através dos mapas de empatia alguns perfis para o público alvo. Nesta seção apresentamos mais detalhes sobre estes perfis. São eles:

* Dono do negócio: Em pequenos negócios, muitas vezes os donos têm dificuldades para acompanhar os custos dos seus produtos e gerir o processo de controle de estoque. Esta dificuldade pode ser tanto relativa a falta de informações ou até mesmo de conhecimento técnico. Logo, faltam subsídios para avaliar quais os períodos mais propícios para reposição do estoque e definir os preços dos produtos de forma equilibrar a equação rentabilidade x competitividade do negócio.
* Gestor do estoque: É responsável pela compra dos produtos e está subordinado do dono do negócio. Eles precisam de informações que ajudem a mapear o fluxo do estoque de forma a otimizar o fluxo de compras. Muitas vezes esta atribuição é do próprio dono do negócio.
* Vendedores: São os responsáveis pelas vendas de produtos e identificação dos clientes. Eles estão subordinados ao dono da loja, mas necessitam ter flexibilidade na negociação das vendas para melhor atender aos clientes. Embora não façam o controle de estoque, precisam destas informações para evitar vendas de produtos sem estoque disponível. São usuários que precisam de agilidade no registro de vendas para um melhor atendimento aos clientes.
* Clientes: Efetuam compras na loja e precisam de informações sobre compras passadas para comprovação ou até mesmo efetuar devoluções.

## 1.3. Requisitos

Com base no exposto anteriormente, foram identificados os seguintes requisitos:

Requisitos Funcionais:

* Gerenciamento de fornecedores: o sistema deve permitir o cadastro, atualização e exclusão de fornecedores.
* Gerenciamento de vendedores: o sistema deve permitir o cadastro, atualização e exclusão de vendedores.
* Gerenciamento de clientes: o sistema deve permitir o cadastro, atualização e exclusão de clientes.
* Gerenciamento de produtos e estoque: o sistema deve permitir o cadastro, atualização e exclusão de produtos. Além disso, deve permitir o controle do estoque e a atualização automática do estoque quando um produto é vendido.
* Gerenciamento de vendas: o sistema deve permitir a realização de vendas podendo identificar ou não vendedores e clientes. O vendedor deve ser identificado automaticamente ao acessar o sistema, mas deve permitir a escolha de outro. Além disso, o sistema deve permitir ao vendedor oferecer descontos nos produtos vendidos.
* Auxílio a Gestão: o sistema deve exibir indicadores sobre vendas e compras de acordo com as necessidades de informações do dono do negócio, vendedores e gestores de estoque. Além disso deve poder identificar os produtos mais vendidos.
* Controle de acesso: o sistema deve permitir o controle de acesso aos usuários, definindo diferentes perfis de acesso para cada tipo de usuário. Além disso, deve permitir que os usuários possam alterar seus dados cadastrais e que novos usuários possam se cadastrar de forma autônoma.

Requisitos Não Funcionais:

* Usabilidade: o sistema deve apresentar uma interface flexível para que possa ser utilizado em ambientes desktop e mobile simultaneamente.
* Desempenho: o sistema deve ser rápido e responsivo, com tempos de resposta baixos.
* Confiabilidade: o sistema deve ser confiável e estar sempre disponível, minimizando a possibilidade de erros e falhas.
* Segurança: o sistema deve ter política de controle de acesso, protegendo-se de acessos não autorizados.
* Escalabilidade: o sistema deve ser capaz de crescer com o negócio, suportando um número crescente de produtos, vendas e usuários.

Sistemas para controles de lojas, podem apresentar muitas outras funcionalidades, como, por exemplo, compra direta pelo site por clientes utilizando diversos meios de pagamento, controle de folha de pagamento e gestão individualizada do negócio para cada filial da empresa. Entretanto, como este sistema é focado em pequenos negócios e com escopo reduzido, foi dado ênfase nas principais necessidades do sistema.

# 2.Modelagem

## 2.1. Diagrama de casos de uso

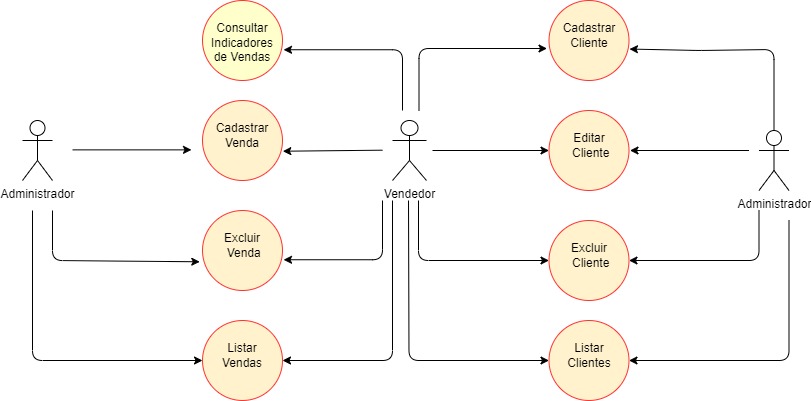
O diagrama de casos de uso oferece uma visão global dos casos de uso e dos atores que dele participam. As imagens abaixo apresentam os casos de uso do sistema que serão detalhados logo em seguida. Elas foram agrupadas de acordo com os atores para facilitar a leitura.

**Figura 6 – Diagrama de casos de uso para todos os atores**

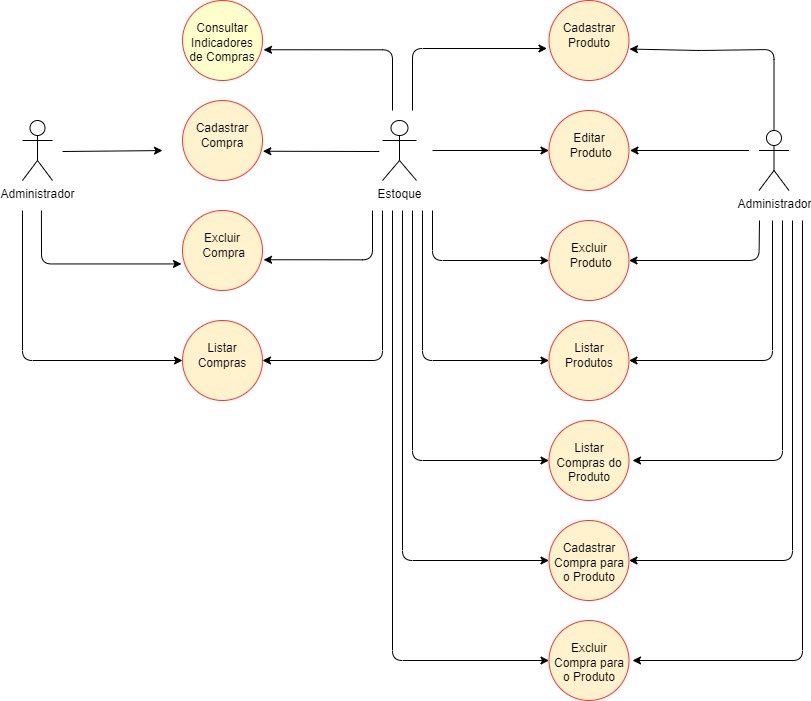


Na figura acima podemos verificar que os usuários ‘Gestor’, ‘Estoque’, ‘Administrador’, ‘Vendedor’ e ‘Clientes’ podem executar os mesmos três casos de uso. Por sua vez, uma pessoa não cadastrada só pode ser registrar ou fazer o login social.

**Figura 7 – Diagrama de casos de uso para os atores Vendedor e Administrador**

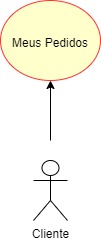


No diagrama acima, destacamos que o ator ‘Administrador’ executa as mesmas funções de um ‘Vendedor’, exceto pela consulta de indicadores de vendas.



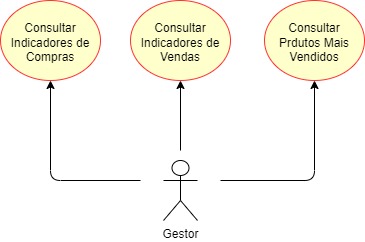
**Figura 8 – Diagrama de casos de uso para os atores Vendedor e Administrador**

No diagrama acima, destacamos que o ator ‘Administrador’ executa as mesmas funções de um ator ‘Estoque’, exceto pela consulta de indicadores de compras.



**Figura 9 – Diagrama de casos de uso para os atores Cliente**

No diagrama acima, temos a ação exclusiva para o ator ‘Cliente’.



**Figura 9 – Diagrama de casos de uso para os atores Gestor**

Conforme apresentado na imagem acima, o ator ‘Gestor’ tem acesso a indicadores e dados que lhe ajudam na análise e tomada de decisão do negócio.

A seguir temos o detalhamento dos casos de uso

## 2.2. Atores

Liste cada um dos atores que participarão dos casos de uso, oferecendo uma descrição sucinta para cada um deles. Os atores são todos aqueles que interagem com o sistema (usuários, outros sistemas, etc.). Agora, como já estamos falando da modelagem do sistema, já podemos incluir os usuários responsáveis pela administração do sistema

## 2.3. Detalhamento dos casos de uso

Nesta seção, cada caso de uso deve ser detalhado. Esse detalhamento deve incluir uma descrição do caso de uso, a lista de atores que participam do caso de uso, as pré e pós-condições e os fluxos de eventos (básico, alternativo, de exceção, sub-fluxos, etc.).

## 2.4. Projeto de interface

Para facilitar o acesso a informação, o sistema foi estruturado em grupos de opções de menu para cada entidade do sistema. São elas: Usuários, Fornecedores, Produtos, Vendedores, Clientes, Compras, Vendas. Além disso, temos as opções de ‘’Gestão, ‘Meus Pedidos’ e ‘Login do usuário’ (é substituída pelo login efetivo do usuário).

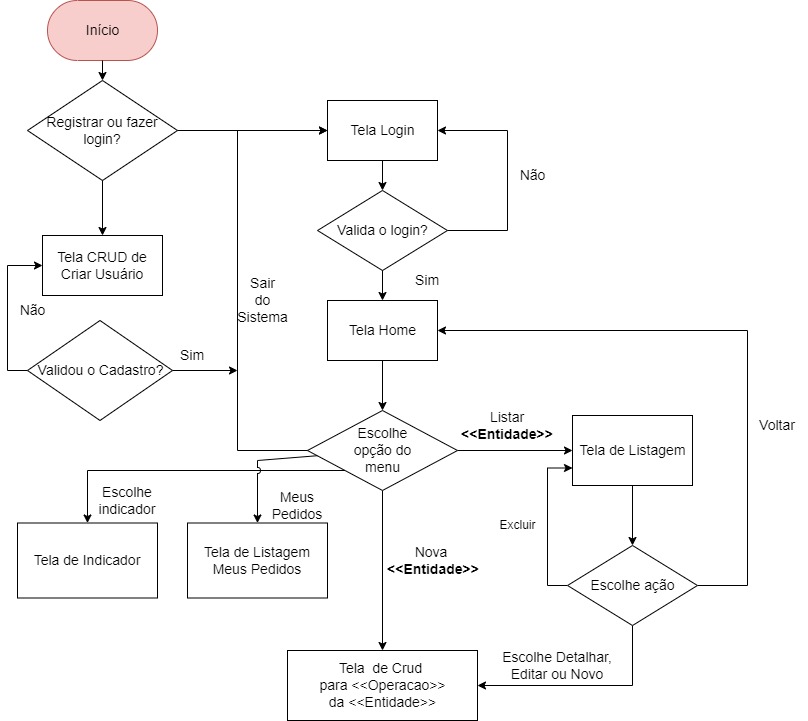
Para as entidades, são apresentadas opções de listagem, que dá acesso as opções de criar, detalhar, editar e excluir do *CRUD*. Além da opção de listagem, para facilitar o acesso ao usuário, também foi criado uma opção de ‘Novo’ para cada entidade. Esta opção dá acesso a mesma funcionalidade de ‘criar’ existente na tela de listagem.

A opção de ‘Gestão’ apresenta as opções de ‘Indicadores para Vendas’ e ‘Indicadores para Compras’ que apresentam os indicadores do sistema. Além disso, apresenta a opção de ‘Produtos mais vendidos’, que é uma tela com layout similar as telas de listagem, mas acrescentando alguns filtros.

A opção ‘Meus pedidos’ é uma tela de listagem que apresenta a listagem de vendas para o usuário logado.

Na opção ‘Login do Usuário’ temos as opções de ‘Atualizar meu perfil’ para atualizar os dados do usuário logado e a opção de sair do sistema.

O diagrama de fluxo do usuário abaixo apresenta o esquema lógico de navegação para as opções apresentadas acima.



**Figura 6 – Diagrama de fluxo do usuário**

### 2.4.1. *Wireframes*

A seguir, apresentamos os *wireframes* que foram utilizados como base para o desenvolvimento dos protótipos. Os wireframes foram construídos de forma genérica. Esta abordagem apresenta vantagens, destacando-se aqui: economia de tempo, fácil adaptação e consistência visual.

**Figura 7 – *Wireframe* da tela de login (página incial)**



Conforme diagrama de fluxo apresentado anteriormente, a tela de login é a página inicial do sistema. Nela o usuário poderá fazer o login, caso já possua cadastro, registrar-se ou fazer login utilizando *OAuth* via Google. Em caso de sucesso do cadastro, login ou login via Google, o sistema irá para tela ‘Home’, apresentada a seguir.



**Figura 8 – *Wireframe* da tela Home (após efetuar o login)**

A partir da tela home, o usuário terá acesso as diversas telas do sistema apresentadas abaixo.

As telas para as entidades ‘Compras’ e ‘Vendas’ são do tipo mestre-detalhe. Por sua vez, as telas para as entidades de Usuários, Fornecedores, Produtos, Vendedores e Clientes são *cruds (Create, Read, Update, Delete)* simples.

As opções de ‘Atualizar meu Perfil’ e ‘Registrar’ são do tipo *crud* simples. ‘Meus pedidos’ é uma tela de listagem com apenas a ação de ‘detalhar’.

Os próximos *wireframes* detalham estes tipos de telas.



**Figura 9 – *Wireframe* das telas de Listagem**

No *wireframe* das telas de listagem destacamos que para cada registro listado existe uma coluna com as possíveis ações que podem ser executadas. Elas são tipicamente opções de *crud*, mas podem ser acrescentadas novas.

A tela de listagem para a entidade ‘Produtos’, apresenta duas ações extras: ‘Nova compra’ e ‘Gerenciar compras’. A primeira é um *crud* básico que permite fazer o cadastro simplificado de uma compra apenas para este produto. Por sua vez, a opção de ‘Gerenciar compras’ é uma tela de listagem que apresenta as compras onde produto é um dos itens. Nesta listagem temos a opções de excluir o produto de uma compra e a opção de ‘Nova compra’, que é a mesma apresentada nas ações do produto.

Quando disponíveis, as opções de ‘Exportar’ e ‘Imprimir’ permitem, respectivamente, exportar os dados da tabela em formato *Excel* e imprimir os dados em formato *PDF*.

A opção de ‘Novo(a) <<Entidade>>’ irá direcionar o usuário para a tela de *crud* correspondente a entidade, sendo apresentada no modo de criação de dados. Estas telas são apresentadas a seguir.



**Figura 10 – *Wireframe* das telas de Crud Básico**

Nas telas do tipo *crud* básico são exibidos o nome da operação (novo, editar ou detalhar) seguido do nome da entidade e, opcionalmente, um identificador para ela. Caso a operação escolhida seja de detalhamento, os campos não poderão ser editados. Neste caso, o botão de ‘salvar’ também não será exibido.

A área de campos do formulário exibe os dados da entidade que precisam ser editados. Caso algum dado seja do tipo *lookup*, o usuário poderá escolher o valor na lista, ou digitando o valor. A medida que ele digita o valor, a lista de opções é filtrada, facilitando assim que ele encontre o registro procurado.

Neste sistema, optamos que os filtros de campos do tipo *lookup* seja feito a partir do campo ‘nome’ da entidade, pois os usuários geralmente não lembram o número de identificação, mas se lembram do nome.



**Figura 11 – *Wireframe* das telas de Crud Mestre Detalhe**

Os *cruds* do tipo mestre-detalhe se comportam de forma simular aos *cruds* básicos. A diferença é que nas operações de ‘novo’ e ‘editar’, o usuário além de preencher os campos da entidade mestre, irá preencher campos para a entidade detalhe. As entidades do tipo detalhe são exibidas em uma tabela, conforme layout do *wireframe* de tela de listagem.

A área de edição para a entidade detalhe não é exibida na operação de ‘detalhar’.

**Figura 12 – *Wireframe* das telas de indicadores**



Para os indicadores, primeiramente o usuário irá selecionar alguns campos de filtro e, após clicar no botão ‘carregar’, serão exibidos os gráficos contendo os indicadores. Cada indicador poderá será ocultado/reexibido clicando no título do indicador.

### 2.4.2. Protótipo interativo

Neste projeto optou-se por fazer o protótipo interativo direto na aplicação. Esta abordagem, embora apresente problemas como, por exemplo, tornar mais difícil compartilhar o protótipo com outras pessoas que não têm acesso ao sistema, permite testar a funcionalidade real e simular o ambiente em que o usuário irá interagir com o sistema. Neste caso, o uso de *mocks* de dados acelera o processo de validação.

As telas criadas a partir dos *wireframes* podem ser acessadas conforme instruções apresentadas na seção *5.1. Aplicação web*.

## 2.5. Modelo de dados

A seguir apresentamos o modelo de dados para a solução proposta.



**Figura 13 – Modelo de Dados**

O sistema apresenta as seguintes entidades:

* Fornecedores: Representam os fornecedores dos produtos da empresa. Relacionamentos:
  + Um fornecedor pode estar associado a zero ou mais compras.
* Vendedores: Representam os vendedores da empresa. Relacionamentos:
  + Um vendedor pode estar associado a zero ou mais vendas.
* Clientes: Representam os clientes do sistema. Relacionamentos:
  + Um cliente pode estar associado a zero ou mais vendas.
* Produtos: Representam os produtos do sistema. Nele são armazenados campos para fazer a gestão do preço médio de aquisição dos produtos. Relacionamentos:
  + Um produto pode estar associado a zero ou mais itens de vendas.
  + Um produto pode estar associado a zero ou mais itens de compras.
* Usuários: Representam os usuários do sistema.
* Compras: Representam as compras do sistema. As compras são compostas por vários ‘ItensCompras’. Para atender aos requisitos do sistema, o fornecedor não é um campo obrigatório. Relacionamentos:
  + Uma compra pode estar associada a zero ou um fornecedor.
  + Uma compra deve estar associada a um ou mais itens de compras.
* Vendas: Representam as vendas do sistema. As vendas são compostas por vários ‘ItensVendas’. O vendedor e o cliente não são campos obrigatórios, atendendo assim aos requisitos do sistema. Relacionamentos:
  + Uma venda pode estar associada a zero ou um vendedor.
  + Uma venda pode estar associada a zero ou um cliente.
  + Uma venda deve estar associada a um ou mais itens de vendas.
* ItensCompras: Entidade fraca que representa cada item da compra. Relacionamentos:
  + Um item de compra deve estar associado a uma compra
  + Um item de compra deve estar associado a um produto.
* ItenVendas: Entidade fraca que representa cada item de venda. Além do produto, armazena informações de preço e preço de custo do produto no momento da compra para calcular o a rentabilidade do produto. Relacionamentos:
  + Um item de venda deve estar associado a uma venda.
  + Um item de venda deve estar associado a um produto.

# 3. Projeto

## 3.1. Arquitetura de *software*

O padrão arquitetural adotado é o cliente-servidor com separação de camadas.

Nesta arquitetura, a camada do cliente (*front-end*) renderiza a interface do usuário e controla a interação do usuário, enquanto a camada servidor (*back-end*) fornece dados e serviços a camada cliente.

As vantagens dessa arquitetura incluem:

1. Separação de responsabilidades: A arquitetura cliente-servidor baseada permite uma separação clara das responsabilidades entre o cliente e o servidor. A camada cliente é responsável pela apresentação e interação do usuário, enquanto a camada servidora é responsável pela lógica de negócios e persistência de dados.
2. Escalabilidade: A arquitetura cliente-servidor é altamente escalável. A camada servidora pode ser implantada em vários servidores para atender a um grande número de solicitações de clientes. O uso de serviços em nuvem permite equilibrar a relação custo de manutenção x escalabilidade, pois pode-se alocar mais ou menos recursos de acordo com a demanda da aplicação.
3. Manutenção: A separação clara de responsabilidades entre o cliente e o servidor torna a manutenção da aplicação mais fácil. As atualizações podem ser feitas em cada camada sem afetar a outra.
4. Segurança: A camada servidora pode ser configurada para autenticação e autorização para garantir que os dados do usuário sejam seguros. A separação de responsabilidades também pode ajudar a minimizar a superfície de ataque da aplicação.
5. Flexibilidade: A arquitetura cliente-servidor é altamente flexível, pois permite que sejam utilizadas tecnologias diferentes em cada camada.

### 3.1.1. Diagramas *C4 model*

Para representar a arquitetura adota, nesta seção são apresentados os diagramas do *C4 Model* da aplicação para os níveis de contexto, contêiner e componentes. Os diagramas representam em vários níveis de abstração a arquitetura e como se dá a interação dos atores envolvidos com a aplicação.

#### 3.1.1.1. Nível 1 - contexto

A figura abaixo apresenta o contexto geral da aplicação. Neste nível podemos identificar os atores do sistema (personas) e a interface com entidades externas.

Também estão identificados, em alto nível, como cada ator irá interagir com o sistema, mas ainda não está exposta a arquitetura adota pelo sistema.



Figura 9 – Diagrama de contexto do sistema

#### 3.1.1.2. Nível 2 - contêiner

A figura abaixo apresenta o segundo nível do *c4 model*. Neste nível, contêiner, a aplicação é separada em seus contêineres e como eles se comunicam. Já podemos identificar os componentes da arquitetura adotada (cliente-servidor).



Figura 10 – Diagrama de contexto do sistema

#### 3.1.1.3. Nível 3 - componentes

As figuras abaixo apresentam o terceiro nível do *c4 model*. Neste nível, componente, detalhamos os principais contêineres da aplicação (*front-end* e *back-end*).



Figura 11 – Diagrama de container para o *front-end*

O diagrama acima apresenta com a camada cliente (*front-end*), desenvolvido em *Angular*, ela trata as interações com os diversos usuários do sistema e utiliza a camada servidora (*back-end*) para recuperar informações e executar as ações solicitadas por estes usuários.

O diagrama abaixo apresenta como o *back-end*, desenvolvido em *Node.Js*, recebe as chamadas do *front-end*, e, a partir dos componentes (router, controller, service e model) autentica, autoriza e responde as chamadas oriundas da camada cliente da arquitetura (*front-end*).



Figura 12 – Diagrama de container para o *back-end*

### 3.2.1. Definição do framework de trabalho

Para implementar a arquitetura escolhida para o sistema, cliente servidor, foi preciso definir um framework de trabalho. A escolha deste framework considerou os seguintes aspectos:

1. Aderência aos requisitos do projeto e arquitetura adotada: O framework deve atender aos requisitos funcionais e não funcionais do projeto. Além disso, devem dar suporte ao padrão arquitetural escolhido.
2. Aplicabilidade no ambiente de trabalho do aluno: Tendo em vista que se trata de uma pós-graduação patrocinada e que a instituição possui política própria de escolha de frameworks, o framework utilizado deve ter aplicabilidade no trabalho do aluno.
3. Aprendizado de novas tecnologias: Além de considerar os critérios anteriores, o framework adotado deve servir como estudo de caso para a proposição de novas soluções para a entidade patrocinadora.

#### 3.2.1.1. Framework front-end

Para o *Front-End* foram adotadas as seguintes tecnologias:

* *Angular* (versão 15).
* *Bootstrap* (via ng-bootstrap)
* *Material Design* (via Angular Material).

O framework *Angular* permite a criação de aplicações de página única (SPA) altamente interativos e responsivos. Ele é mantido e adotado pelo Google, possuindo uma grande comunidade de usuários. Aplicações desenvolvidas em *Angular* são executadas no navegador do cliente que é responsável pela renderização da interface do usuário.

O *Bootstrap* fornece estruturas de *Cascading Style Sheets* (CSS) para a criação de sites e aplicações responsivas. Ele permite tratar sites para ambiente desktop e mobile da mesma forma. Para o projeto, foi adotado o módulo *ng-bootstrap* que é uma implementação do *bootstrap* para *Angular*.

Uma vez definido o framework, é importante a escolha de uma biblioteca de componentes para acelerar o desenvolvimento da aplicação. Considerando que a adoção da biblioteca de componentes *Angular* da instituição patrocinadora do aluno não poderia ser utilizada devido as regras de sigilo e propriedade intelectual, optou-se pela adoção do *Material Design* através da biblioteca *Material Angular*.

*Material Design* é a especificação de design para interfaces interativas do Google. Já o *Angular Material* é a implementação oficial, para Angular, do Material Design.

Sendo assim, a escolha dessa arquitetura acelera a criação de aplicações com alta responsividade para ambientes desktop e mobile. Além disso, devido a sua grande adoção, permite o acesso a uma grande comunidade de desenvolvedores para compartilhar conhecimento.

#### 3.2.1.2. Framework back-end

Para o *Back-End* foi adotada a seguinte tecnologia:

* *Node.js* com *Express.js*

O *Node.js* é uma plataforma de software de código aberto construída com a linguagem de programação *JavaScript*. Ele permite que os desenvolvedores criem aplicativos de rede escaláveis e de alto desempenho usando *JavaScript*. Um ambiente *Node.js* não exige muitos recursos computacionais e por ser baseado em um modelo de E/S não bloqueante e orientado a eventos, torna se ideal para aplicativos que precisam lidar com muitas conexões simultâneas.

O *Express* é um framework web de código aberto para o *Node.js* que fornece um conjunto de métodos HTTP para criar rotas de acesso ao servidor e lidar com solicitações e respostas HTTP, dando suporte para a criação de APIs *REST*.

Desta forma, a escolha desta arquitetura permitirá a criação de um ambiente de alta disponibilidade, escalável e ao mesmo tempo de baixo custo, uma vez que não possui baixo consumo de recursos computacionais. Além disso, é uma oportunidade de estudo com grandes possibilidades de uso no ambiente de trabalho do aluno.

#### 3.2.1.3. Persistência

Para a persistência foi adotada a seguinte tecnologia:

* *MongoDB*

O *MongoDB* é um banco de dados *NoSQL* (não relacional) de código aberto, que armazena dados em formato de documento *JSON (JavaScript Object Notation).* Ele foi criado para fornecer alta escalabilidade, desempenho e flexibilidade para aplicativos modernos que precisam armazenar e acessar grandes quantidades de dados.

Disponibilizado em conjunto por servidores que hospedam aplicações *Node.js*, o *MongoDB* se torna uma ótima solução para a persistência da aplicação, aliando robustez, escalabilidade e baixo custo.

## 3.2. Arquitetura da informação

A arquitetura da informação tem como objetivo organizar e estruturar as informações de forma lógica e coerente para que seja fácil e intuitivo para os usuários encontrarem e acessarem aquelas que desejam.

Nesta aplicação, as categorias representam as entidades do sistema. Para cada uma, foi construída uma opção de menu que agrupa todas as funcionalidades de forma a facilitar a navegação e o acesso dos usuários às informações que eles procuram. Além disso, as categorias são exibidas de acordo com o perfil do usuário. Desta forma, por exemplo, um usuário com perfil de cliente não irá ver informações relativas a compra de produtos ou fornecedores.

Entre as funcionalidades disponíveis destaca-se a listagem, onde os dados das entidades são exibidos em listas ordenadas e paginadas, facilitando assim a localização de informações e concentrando o acesso as operações que podem ser feias.

A figura abaixo apresenta o diagrama de mapa do site que representa a estrutura e hierarquia do conteúdo da aplicação.

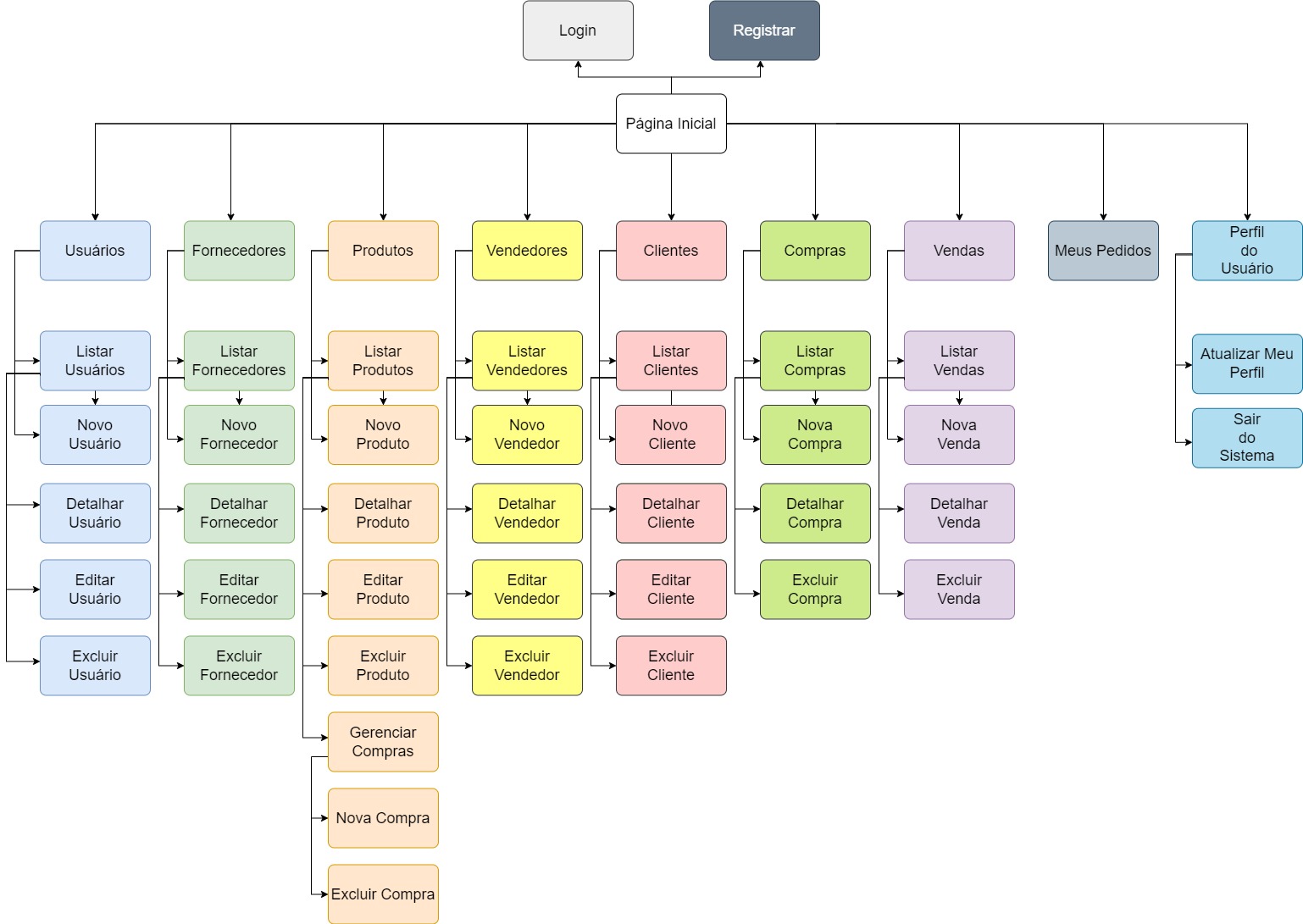


Figura 13 – Diagrama de mapa do site

# 4. Testes

Apresente os planos de testes (testes de software, testes heurísticos ou testes de usabilidade) além dos registros de testes realizados em sua aplicação Web.

# 5. URLs

## 5.1. Aplicação web

A aplicação web encontra-se hospedada em <https://adjamir2318.c41.integrator.host/>.

Para acessar a aplicação com o perfil de administrador deve-se utilizar o login “admin” e a senha “@admin!”.

## 5.2. Repositório código-fonte e documentação

O código fonte e a documentação da aplicação encontram-se em <https://github.com/adjamirgalvao/pucminas>.

## 5.3. Especificação da API Swagger/OpenAPI

A especificaç da API encontra-se em <https://adjamir2318.c41.integrator.host/api-docs/>.

## 5.4. Vídeo de apresentação do trabalho

Inclua o link para o vídeo com a apresentação da sua aplicação. Seu vídeo deve ter duração máxima de 5 minutos e deve apresentar de forma sucinta o seu projeto. Concentre-se principalmente na apresentação da sua aplicação, mostrando o funcionamento de cada um dos requisitos solicitados na proposta escolhida por você.

O software OBS Studio é uma boa sugestão de software para gravação de screencast.

# REFERÊNCIAS

Como um projeto de aplicativo não requer revisão bibliográfica, a inclusão das referências não é obrigatória. No entanto, caso você deseje incluir referências relacionadas às tecnologias ou às metodologias que foram usadas no seu trabalho, relacione-as de acordo com o modelo a seguir.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

SOBRENOME DO AUTOR, Nome do autor. **Título do livro ou artigo.** Cidade: Editora, ano.

1. https: //www.totvs.com/varejo/lojas/ [↑](#footnote-ref-1)