**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](#_Toc128254692)

[1.1. Contexto 4](#_Toc128254693)

[1.2. Público alvo 8](#_Toc128254694)

[1.3. Requisitos 8](#_Toc128254695)

[2. Modelagem 11](#_Toc128254696)

[2.1. Diagrama de casos de uso 11](#_Toc128254697)

[2.2. Atores 11](#_Toc128254698)

[2.3. Detalhamento dos casos de uso 11](#_Toc128254699)

[2.4. Projeto de Interface 11](#_Toc128254700)

[2.5. Diagrama de classes 11](#_Toc128254701)

[3. Projeto 12](#_Toc128254702)

[3.1. Arquitetura de *software* 12](#_Toc128254703)

[3.1.1. Diagramas C4 Model 12](#_Toc128254704)

[3.1.1.1. Nível 1 - Contexto 13](#_Toc128254705)

[3.1.1.2. Nível 2 - Contêiner 13](#_Toc128254706)

[3.1.1.3. Nível 3 - Componentes 14](#_Toc128254707)

[3.2.1. Definição do Framework de Trabalho 16](#_Toc128254708)

[3.2.1.1. Framework Front End 17](#_Toc128254709)

[3.2.1.2. Framework Back End 18](#_Toc128254710)

[3.2.1.3. Persistência 18](#_Toc128254711)

[3.2. Arquitetura da informação 19](#_Toc128254712)

[4. Testes 20](#_Toc128254713)

[5. URLs 20](#_Toc128254714)

[5.1. Aplicação web 20](#_Toc128254715)

[5.2. Repositório código-fonte 20](#_Toc128254716)

[5.3. Especificação da API Swagger/OpenAPI 20](#_Toc128254717)

[5.4. Vídeo de apresentação do trabalho 20](#_Toc128254718)

[REFERÊNCIAS 21](#_Toc128254719)

# 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 loja.

O atendimento aos clientes ocorre na loja física e o registro de vendas pode ser feita simultaneamente pelo caixa tradicional e por 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.

Com o objetivo de entender melhor as necessidades, desejos e expectativas dos usuários em relação ao produto, apresentamos os mapas de empatia para os diferentes públicos alvo do sistema.

**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**

Partindo dessas informações iniciais, foi elaborado o *Business Model Canvas* da aplicação. Utilizada para modelar o modelo de negócios de uma empresa, é uma das fontes de informações utilizadas no processo de descoberta do produto. Ela 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.

A figura 1 apresenta o *Business Model Canvas (BMC)* utilizado no processo de descoberta de produto (*Product Disconvery*). Suas informações são utilizadas para entender e mapear a lógica do negócio e as necessidades dos usuários, ajudando na tomada de decisão sobre o que construir e como construir.



**Figura 3 - *Business Model Canvas***

## 1.2. Público alvo

Na seção anterior identificamos, através dos mapas de empatia e do *Business Model Canvas*,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. 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. 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.

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, controle de folha de pagamento e gestão individualizada do negócio para cada filial da empresa. Entretanto, como este sistema é focado em pequenos negócios, 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.

## 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

Apresente o fluxo do usuário no sistema e o layout das interfaces, incluindo uma descrição dos campos e comandos em cada uma delas. Como forma de layout, use um wireframes ou as interfaces já implementadas.

Se necessário, use um diagrama de estados para representar o comportamento de uma interface ou fluxo de um processo associado à funcionalidade em questão.

## 2.5. Diagrama de classes

Apresente o diagrama de classes da aplicação web, descrevendo, sucintamente, as classes e as relações entre elas.

# 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 modelo C4 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 6 – 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 7 – 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 8 – Diagrama de container para o *front-end*

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

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



Figura 9 – 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 RESTfull.

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.

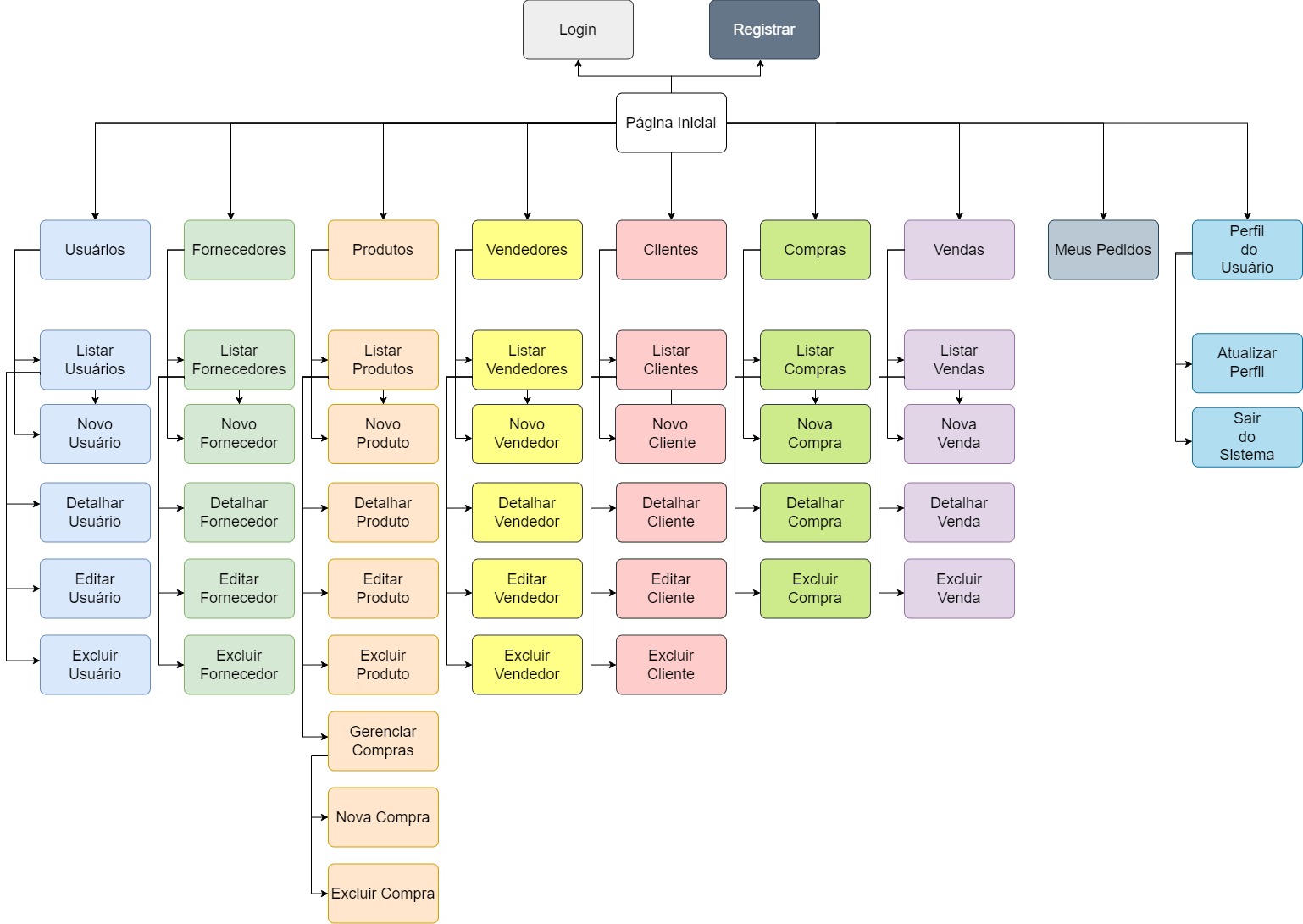


Figura 10 – Diagrama de mapa do site

# 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

O código fonte da aplicação encontra-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)