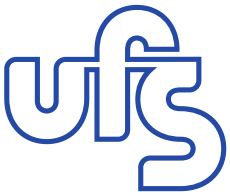


UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Rivet: Aplicação do Padrão SPDX no Gerenciamento da Cadeia de Suprimentos de Software

Trabalho de Conclusão de Curso

Daniel Santos Rodrigues

São Cristóvão – Sergipe

2026

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Daniel Santos Rodrigues

Rivet: Aplicação do Padrão SPDX no Gerenciamento da Cadeia de Suprimentos de Software

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Profª. Drª. Edilayne Meneses Salgueiro

São Cristóvão – Sergipe

2026

Agradecimentos

Primeiramente, agradeço a Deus, por ter me concedido saúde, força e perseverança ao longo de toda a minha trajetória acadêmica, possibilitando a superação dos desafios enfrentados durante a realização deste trabalho.

Agradeço aos meus pais, pelo apoio incondicional, incentivo constante e por acreditarem em mim em todos os momentos. O suporte oferecido por eles foi fundamental para que eu pudesse seguir firme em meus estudos e alcançar mais essa etapa da minha formação.

Agradeço também à minha namorada, pelo companheirismo, paciência e compreensão ao longo desse período, especialmente nos momentos de maior dedicação ao trabalho acadêmico. Seu apoio foi essencial para manter a motivação e o equilíbrio durante o desenvolvimento deste trabalho.

O sucesso é a soma de pequenos esforços repetidos dia após dia.
(Robert Collier)

Resumo

Este trabalho apresenta o desenvolvimento de uma aplicação desktop denominada Rivet, voltada ao gerenciamento da cadeia de suprimentos de software por meio da utilização do padrão SPDX. O sistema tem como objetivo apoiar a geração de Listas de Materiais de Software (SBOMs), bem como a identificação de vulnerabilidades e a análise de conformidade de licenças associadas aos componentes de um projeto de software, contribuindo para maior transparência e segurança no uso de dependências de terceiros. Para fundamentar a proposta, foi realizado um mapeamento sistemático da literatura, a partir do qual foram analisados conceitos relacionados à cadeia de suprimentos de software, ao uso de SBOMs e às soluções existentes na área. Com base nesse embasamento, a aplicação foi projetada e implementada de forma a integrar funcionalidades de geração e análise de informações em uma interface unificada. Como resultado, obteve-se uma solução funcional que facilita a identificação de riscos de segurança e questões de licenciamento em projetos de software, configurando-se como uma contribuição prática para o gerenciamento da cadeia de suprimentos de software, culminando no registro do software desenvolvido.

Palavras-chave: SPDX. SBOM. Cadeia de suprimentos de software. Segurança de software. Licenças de software.

Abstract

This work presents the development of a desktop application named Rivet, aimed at managing the software supply chain through the use of the SPDX standard. The system is designed to support the generation of Software Bills of Materials (SBOMs), as well as the identification of vulnerabilities and the analysis of license compliance associated with the components of a software project, contributing to greater transparency and security in the use of third-party dependencies. To support the proposal, a systematic literature mapping was conducted, through which concepts related to software supply chain security, the use of SBOMs, and existing solutions in the field were analyzed. Based on this foundation, the application was designed and implemented to integrate generation and analysis functionalities into a unified interface. As a result, a functional solution was obtained that facilitates the identification of security risks and licensing issues in software projects, constituting a practical contribution to software supply chain management and culminating in the registration of the developed software.

Keywords: SPDX. SBOM. Software supply chain. Software security. Software licenses.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Sumário

1 Introdução

Em um cenário em que o mundo se torna cada vez mais digitalizado, soluções de software são desenvolvidas de forma acelerada para otimizar processos, facilitar o cotidiano e impulsionar a inovação em diferentes setores da sociedade. Esse avanço, no entanto, vem acompanhado de um fluxo crescente de dados, incluindo informações sensíveis de indivíduos e organizações, que passam a ser coletadas, processadas e armazenadas em ambientes digitais. Nesse contexto, garantir a segurança, a integridade e a conformidade legal desses sistemas torna-se não apenas uma necessidade técnica, mas também um compromisso ético e social. Diante disso, é extremamente necessário proteger todos os componentes de software durante todas as etapas de desenvolvimento e implantação, de forma a reduzir riscos, assegurar a qualidade e manter a confiabilidade das aplicações.

De acordo com SECURITY (2024), a chamada cadeia de suprimentos de software abrange todos os elementos, recursos e processos envolvidos na concepção, desenvolvimento, distribuição e manutenção de uma aplicação, englobando todo o seu ciclo de vida. Garantir a proteção dessa cadeia significa adotar práticas que assegurem a integridade de cada componente, desde bibliotecas de terceiros até módulos desenvolvidos internamente, bem como monitorar e validar todas as atividades e etapas de implantação.

Para isso, é essencial que equipes de desenvolvimento e fornecedores priorizem o uso de componentes livres de vulnerabilidades conhecidas, estabeleçam mecanismos de verificação de integridade das compilações e mantenham processos capazes de detectar possíveis indícios de adulteração não autorizada (SECURITY, 2024). Essa abordagem fortalece não apenas a segurança técnica, mas também a confiança de usuários e parceiros no produto final.

O crescimento do software de código aberto (OSS) impulsionou a reutilização de componentes como bibliotecas e frameworks, que são distribuídos sob licenças específicas. Garantir o cumprimento dessas licenças é fundamental, pois a não conformidade pode gerar consequências legais, financeiras e reputacionais (WINTERSGILL, 2024). Assim, torna-se

evidente a necessidade de ferramentas e processos que verifiquem e assegurem a conformidade desses componentes ao longo do desenvolvimento.

Uma ferramenta relevante para a gestão da segurança na cadeia de suprimentos é a Lista de Materiais de Software (SBOM). Ela funciona como um inventário detalhado de todos os componentes de um projeto, incluindo bibliotecas, dependências, versões e informações de licenciamento, promovendo transparência e apoiando a conformidade, segurança e qualidade (SECURITY, 2024). Apesar da existência de padrões consolidados, muitas organizações ainda encontram dificuldades para escolher qual dos dois principais padrões de SBOM do mercado adotar: o CycloneDX ou o SPDX.

Segundo SECURITY (2023), o CycloneDX, padrão de código aberto desenvolvido pela Open Web Application Security Project (OWASP) em 2017, facilita a análise de componentes e a detecção de vulnerabilidades, além de auxiliar no cumprimento de requisitos de licenciamento, identificação de componentes obsoletos e automação de processos relacionados ao SBOM. Por sua flexibilidade, é adotado tanto por projetos de software livre quanto por empresas privadas.

O SPDX é um padrão mantido pela Linux Foundation, também aberto e legível por máquina. Criado em 2011, seu propósito inicial era simplificar o gerenciamento de licenças de software de código aberto. Com o tempo, foi expandido e aprimorado para atender a novas demandas, especialmente relacionadas à segurança e à interoperabilidade com outros formatos de SBOM. Em 2021, o SPDX alcançou um marco importante ao ser oficialmente reconhecido pela International Standards Organization (ISO) como o único padrão internacional para construção de SBOMs. Sua versão mais recente foi alinhada ao guia da NTIA sobre “Elementos Mínimos para uma Lista de Materiais de Software”, reforçando sua relevância e abrangência (SECURITY, 2023). Atualmente, o SPDX é amplamente utilizado para o compartilhamento de informações detalhadas sobre componentes de software ao longo de toda a cadeia de suprimentos, oferecendo suporte tanto a equipes de desenvolvimento quanto a organizações que precisam garantir transparência, conformidade e segurança em seus sistemas.

Por conta dos fatos relatados e pelo reconhecimento pela ISO, o foco principal desta pesquisa está na aplicação do padrão SPDX no gerenciamento da cadeia de suprimentos de software. Nesse cenário, propõe-se o desenvolvimento de um *software* denominado Rivet, capaz de realizar a verificação automática de licenças e monitorar vulnerabilidades, oferecendo ferramentas para a conformidade contínua e a detecção precoce de falhas, promovendo um ambiente de desenvolvimento mais seguro, confiável e transparente.

Na redação deste trabalho, foram utilizadas ferramentas de Inteligência Artificial generativa exclusivamente como auxílio na revisão de linguagem, com foco em correções gramaticais e sugestões de melhorias textuais.

1.1 Justificativa

O uso intensivo de componentes de software de código aberto tornou-se uma prática comum no desenvolvimento de sistemas modernos, impulsionando ganhos significativos de produtividade e inovação. No entanto, essa ampla adoção também introduz desafios relevantes, especialmente no que se refere à conformidade de licenças e à identificação de vulnerabilidades em dependências de terceiros. A ausência de mecanismos adequados para gerenciar essas informações pode resultar em riscos legais, falhas de segurança e perda de confiabilidade dos sistemas.

Nesse contexto, a gestão da cadeia de suprimentos de software exige maior transparência e controle sobre os componentes utilizados ao longo de todo o ciclo de vida das aplicações. Embora existam práticas e ferramentas voltadas à geração de SBOMs, muitas organizações ainda enfrentam dificuldades na padronização, interpretação e uso efetivo dessas informações, sobretudo quando o processo depende de análises manuais ou soluções fragmentadas.

O padrão SPDX destaca-se como uma alternativa consolidada para enfrentar esses desafios, por ser um padrão aberto, reconhecido internacionalmente e amplamente adotado na indústria. Sua capacidade de representar informações detalhadas sobre licenciamento, relacionamentos entre componentes e aspectos de segurança justifica sua aplicação no gerenciamento da cadeia de suprimentos de software. Dessa forma, o desenvolvimento de uma solução que utilize o SPDX para proporcionar a verificação de licenças e o monitoramento de vulnerabilidades mostra-se pertinente e relevante, contribuindo para a adoção de práticas mais seguras, confiáveis e alinhadas às exigências atuais do desenvolvimento de software.

1.2 Objetivos

1.2.1 Objetivo geral

Este trabalho tem como objetivo geral desenvolver um *software desktop*, entitulado Rivet, voltado ao monitoramento da cadeia de suprimentos de software usando o padrão SPDX, capaz de identificar e analisar componentes utilizados em projetos, verificando a conformidade de suas licenças e monitorando vulnerabilidades conhecidas. A solução proposta destina-se a apoiar equipes de desenvolvimento e gestores de tecnologia na adoção de práticas mais seguras e transparentes, reduzindo riscos legais, técnicos e reputacionais associados ao uso de bibliotecas e dependências de código aberto.

1.2.2 Objetivos específicos

- Compreender em profundidade o padrão SPDX, analisando sua estrutura, funcionalidades e aplicação prática na construção de SBOMs.

- Analisar bibliotecas e ferramentas existentes que implementam ou dão suporte ao padrão SPDX.
- Identificar vulnerabilidades em componentes de software que podem ser monitoradas e representadas por meio do SPDX.
- Projetar e implementar um *software desktop* chamado Rivet que utilize o padrão SPDX para gerenciar informações sobre licenciamento e vulnerabilidades.
- Avaliar a eficácia e a usabilidade da solução proposta em cenários simulados de uso real.

1.3 Metodologia

Este trabalho caracteriza-se como uma pesquisa de natureza aplicada, com foco no desenvolvimento de uma solução tecnológica voltada ao gerenciamento da cadeia de suprimentos de software por meio do padrão SPDX. A abordagem adotada combina uma revisão sistemática da literatura com o desenvolvimento de software, permitindo não apenas fundamentar teoricamente a solução proposta, mas também identificar o estado da técnica das implementações existentes na área. A revisão sistemática contribuiu, assim, para a consolidação dos conceitos empregados e para a análise de anterioridade, situando a proposta desenvolvida no contexto das soluções já disponíveis.

Inicialmente, foi conduzido um mapeamento sistemático da literatura com o objetivo de identificar estudos, abordagens e ferramentas relacionadas ao uso do padrão SPDX, à conformidade de licenças e ao monitoramento de vulnerabilidades em componentes de software. Para isso, foram definidas strings de busca específicas envolvendo os termos “SPDX”, “software compliance”, “software licensing” e “software metadata”, aplicadas em bases acadêmicas e repositórios técnicos, conforme critérios de inclusão e exclusão previamente estabelecidos. Ao todo, foram identificados 61 artigos, dos quais 10 foram selecionados por sua relevância, servindo de base para a definição do escopo do trabalho e dos requisitos do sistema a ser desenvolvido.

Com base no mapeamento sistemático realizado, foi possível identificar os principais problemas relacionados ao uso de componentes de software de código aberto, como a conformidade de licenças e a presença de vulnerabilidades. A partir dessas informações, foram definidos os requisitos que irão orientar o desenvolvimento do software, de forma a atender às necessidades levantadas. Os requisitos principais são:

- Gerar SBOMs no padrão SPDX, permitindo rastrear os componentes usados no software.
- Verificar a conformidade das licenças dos componentes usados no software.
- Monitorar vulnerabilidades em dependências e bibliotecas utilizadas.
- Fornecer informações claras e objetivas, que ajudem desenvolvedores a tomar decisões.

O desenvolvimento do *software* foi conduzido por meio de métodos ágeis, com foco em entregas incrementais e iterativas. Essa abordagem possibilitou implementar e validar gradualmente as funcionalidades do sistema, garantindo maior flexibilidade frente a ajustes nos requisitos identificados durante o mapeamento sistemático da literatura. A adoção de métodos ágeis também favoreceu a integração contínua de novos recursos, a priorização de funcionalidades críticas e a adaptação a eventuais mudanças, assegurando que o *software* atendesse de forma eficiente às necessidades de monitoramento de licenças e vulnerabilidades em componentes de software.

1.4 Estrutura do Documento

Este trabalho está organizado em cinco capítulos, conforme descrito a seguir:

- Capítulo 2 - **Fundamentação Teórica**: aborda os principais conceitos relacionados ao padrão SPDX e apresenta o mapeamento sistemático da literatura.
- Capítulo 3 - **Busca de Anterioridade**: discute ferramentas e soluções existentes voltadas à geração de SBOMs, análise de vulnerabilidades e verificação de licenças.
- Capítulo 4 - **Desenvolvimento do Produto de Software**: apresenta a especificação dos requisitos, a definição da arquitetura, os diagramas, a prototipagem das telas e a implementação da aplicação.
- Capítulo 5 - **Resultados**: apresenta a validação da aplicação por meio de estudo de caso, evidenciando a geração de SBOMs, a identificação de vulnerabilidades, a análise de licenças e a efetividade da solução no gerenciamento da cadeia de suprimentos de software.
- Capítulo 6 - **Considerações Finais**: apresenta as conclusões do trabalho, destacando os resultados alcançados, as limitações identificadas e as possibilidades de trabalhos futuros.

2
Fundamentação Teórica

2.1 Mapeamento Sistemático

O uso de componentes de software de código aberto tem se tornado cada vez mais comum. No entanto, essa ampla adoção traz desafios, como a garantia da conformidade das licenças entre os componentes utilizados, além da necessidade de identificar possíveis vulnerabilidades que possam comprometer a segurança do projeto (GANDHI; GERMONPREZ; LINK, 2018). Nesse contexto, a evolução das práticas de segurança na cadeia de suprimentos de software tem impulsionado o crescimento e a adoção de padrões de *Software Bill of Materials* (SBOM), que permitem rastrear componentes e mitigar riscos de maneira mais eficaz.

Entre os diferentes padrões de SBOM existentes, destaca-se o *System Package Data Exchange* (SPDX), cuja padronização pela *International Standards Organization* (ISO) conferiu legitimidade e consolidou sua adoção em diversos contextos industriais e acadêmicos. O SPDX busca padronizar a forma de descrever metadados sobre pacotes e componentes de software, viabilizando maior transparência, rastreabilidade e segurança na cadeia de suprimentos.

Neste trabalho, adota-se a metodologia de Mapeamento Sistemático da Literatura com o propósito de identificar e analisar estudos, ferramentas e aplicações relacionadas ao uso do SPDX. Essa estratégia possibilita compreender como o padrão tem sido aplicado, quais benefícios oferece e quais lacunas ainda permanecem, servindo como base para fundamentar a proposta de desenvolvimento desta pesquisa.

2.1.1 Estratégias de Busca

Para garantir que a busca por estudos relevantes fosse abrangente e consistente, foram definidas estratégias sistemáticas de pesquisa, contemplando a formulação de strings específicas e a seleção criteriosa de bases de dados. Essa etapa teve como objetivo assegurar a recuperação de publicações alinhadas ao escopo deste mapeamento, reduzindo a probabilidade de omissão de

trabalhos pertinentes. As palavras-chave e operadores booleanos foram combinados de forma a cobrir variações terminológicas e sinônimos relacionados a *Software Bill of Materials* (SBOM), licenciamento e segurança na cadeia de suprimentos de software.

2.1.1.1 Strings de Busca

As strings de busca foram definidas a partir de um levantamento inicial de termos e conceitos centrais ao tema desta pesquisa, com base em leituras preliminares e na terminologia adotada pela comunidade e pela documentação oficial do padrão SPDX. A escolha dos termos buscou contemplar tanto o nome do padrão quanto aspectos específicos de sua aplicação, como metadados de software, conformidade de licenças e gerenciamento de dependências.

Além disso, os operadores booleanos foram utilizados para ampliar o escopo e permitir a recuperação de publicações que combinassem mais de um conceito de interesse. Optou-se por termos em inglês devido à predominância dessa língua nas publicações científicas e técnicas da área. As consultas aplicadas nas bases de dados foram:

```
"SPDX" AND "software metadata"
"SPDX" AND "Software Compliance" AND "software licensing"
"SDPX" AND "software dependency"
"SPDX" AND "software licensing"
```

Cada uma dessas combinações foi concebida para abordar um conjunto específico de aspectos:

Tabela 1 – Strings de busca e respectivos focos de pesquisa

String de busca	**Foco/Objetivo**
`"SPDX"AND "software metadata"`	Localizar trabalhos que tratem do padrão SPDX no contexto de metadados de software, incluindo formatos e estrutura de arquivos.
`"SPDX"AND "Software Compliance"AND "software licensing"`	Identificar estudos sobre o uso do SPDX no apoio à conformidade legal e ao gerenciamento de licenças de software.
`"SDPX"AND "software dependency"`	Encontrar publicações que relacionem o padrão (incluindo variações de escrita como “SDPX”) ao gerenciamento de dependências de software.
`"SPDX"AND "software licensing"`	Buscar pesquisas que explorem o uso do SPDX especificamente na área de licenciamento de software.

As buscas foram realizadas por meio do Google Acadêmico, que serviu como ponto central para localizar publicações indexadas em diferentes repositórios. A partir dele, foram identificados artigos provenientes das seguintes bases de dados: *IEEE Xplore*, *ScienceDirect* (Elsevier), *SSOAR* e *ACM Digital Library*. A Tabela 2 apresenta um resumo consolidado das

strings de busca, as bases de dados associadas, o número total de resultados encontrados e a quantidade de artigos selecionados após a triagem.

Tabela 2 – Strings de busca, bases utilizadas e resultados encontrados

String de busca	**Bases de dados**	**Resultados encontrados**	**Artigos selecionados**
`"SPDX"AND "software metadata"`	IEEE Xplore, ScienceDirect, SSOAR, ACM Digital Library	36	2
`"SPDX"AND "Software Compliance"AND "software licensing"`	IEEE Xplore, ScienceDirect, SSOAR, ACM Digital Library	24	5
`"SDPX"AND "software dependency"`	IEEE Xplore, ScienceDirect, SSOAR, ACM Digital Library	1	1
`"SPDX"AND "software licensing"`	IEEE Xplore, ScienceDirect, SSOAR, ACM Digital Library	149	2

A forma como as strings de busca foram elaboradas, aliada à escolha cuidadosa das bases de dados, possibilitou uma investigação ampla e consistente da literatura. Esse processo permitiu reunir desde estudos de caráter mais exploratório até pesquisas aplicadas que discutem o uso do padrão SPDX em diferentes cenários. A variedade de termos e operadores adotados ajudou a minimizar possíveis lacunas na busca, aumentando as chances de encontrar trabalhos relevantes, inclusive aqueles focados em aspectos mais específicos, como licenciamento, metadados e gerenciamento de dependências.

Conforme apresentado na Tabela 2, as diferentes combinações de termos produziram volumes distintos de resultados nas bases consultadas. A string *"SPDX" AND "software metadata"* retornou 36 publicações, das quais 2 foram selecionadas após a triagem. Já a combinação *"SPDX" AND "Software Compliance" AND "software licensing"* resultou em 24 trabalhos, com 5 selecionados. A busca por *"SPDX" AND "software dependency"* identificou 1 publicação, que foi incluída na amostra final, enquanto a string *"SPDX" AND "software licensing"* apresentou o maior número de ocorrências, com 149 resultados, dos quais 2 atenderam aos critérios estabelecidos. No total, foram identificadas 210 publicações, das quais 10 compuseram o conjunto final de estudos analisados, formando uma base consistente para a etapa subsequente do mapeamento.

2.1.1.2 Critérios de Inclusão e Exclusão

A escolha das publicações seguiu um processo criterioso e bem estruturado, baseado em critérios definidos previamente para assegurar que apenas estudos realmente relevantes e com qualidade metodológica fossem incluídos na análise. O uso de regras claras ajudou não só a

evitar vieses na seleção, mas também a garantir que o processo possa ser reproduzido por outros pesquisadores, preservando a transparência e a consistência dos resultados obtidos.

Para os critérios de inclusão, foram considerados elegíveis para análise os estudos que atendessem aos seguintes parâmetros:

- **Relevância temática**: O estudo deveria tratar, de forma direta e substancial, do padrão SPDX ou de conceitos intimamente ligados, como metadados de software, licenciamento, conformidade de licenças e gerenciamento de dependências.
- **Recorte temporal**: Foram incluídos trabalhos publicados entre 2015 e 2024. Esse intervalo foi definido considerando que o padrão SPDX já possuía maior consolidação em seu uso, com sucessivas atualizações e adoção crescente pela indústria e pela academia.
- **Disponibilidade de acesso**: O texto completo do estudo deveria estar disponível para leitura e análise, seja de forma aberta ou por meio de acesso institucional.

Em relação aos critérios de exclusão, foram descartados os trabalhos que apresentassem qualquer uma das seguintes características:

- **Duplicidade**: Estudos identificados mais de uma vez em diferentes bases de dados foram mantidos apenas uma vez, evitando contagem duplicada.
- **Irrelevância temática**: Publicações que mencionassem o termo "SPDX" ou conceitos relacionados apenas de forma superficial, sem desenvolver discussões pertinentes ao escopo da pesquisa.
- **Ausência de texto completo**: Estudos disponíveis apenas na forma de resumos, pôsteres, slides de apresentações ou descrições não acadêmicas foram excluídos.

Esse conjunto de critérios foi fundamental para reduzir o número inicial de resultados obtidos para um conjunto manejável e alinhado ao objetivo da pesquisa, garantindo que o foco permanecesse na investigação de evidências robustas e diretamente relacionadas ao tema.

2.1.1.3 Palavras-chave Utilizadas

A escolha das palavras-chave é um elemento central em qualquer pesquisa bibliográfica sistemática, pois determina o alcance e a precisão dos resultados recuperados. Neste estudo, as palavras-chave foram selecionadas a partir de leituras exploratórias iniciais, análise de glossários técnicos e observação da terminologia presente na documentação oficial do padrão SPDX, bem como em publicações acadêmicas que tratam de temas correlatos.

A formulação das combinações buscou contemplar três eixos principais:

1. **Identificação direta do padrão**: Utilizando o termo oficial “SPDX” e a sua forma expandida *“System Package Data Exchange”*, garantindo a recuperação de publicações que abordassem diretamente o padrão, sua aplicação e documentação oficial.

2. **Aspectos técnicos e contextuais**: Incluindo termos como *“software metadata”* e *“software compliance”* para capturar estudos voltados a metadados e à conformidade regulatória no desenvolvimento de software.

3. **Gestão e licenciamento**: Empregando termos como *“software licensing”* e *“software dependency”* para identificar pesquisas relacionadas ao licenciamento e à gestão de dependências de software, áreas onde o padrão SPDX tem ampla aplicação.

A Tabela 3 apresenta as palavras-chave organizadas por categoria temática:

Tabela 3 – Palavras-chave utilizadas nas buscas

Categoria	**Termos utilizados**
Identificação do padrão	`"SPDX"`, `"Software Package Data Exchange"`, `"SDPX"`
Aspectos técnicos e contextuais	`"software metadata"`, `"software compliance"`
Gestão e licenciamento	`"software licensing"`, `"software dependency"`, `"supply chain security"`

O uso dessas palavras-chave, combinadas com operadores booleanos, foi fundamental para garantir que as buscas retornassem um conjunto de resultados consistente e relevante apresentados na Tabela 4, evitando dispersão desnecessária. Essa abordagem contribuiu de forma direta para a construção de um referencial teórico sólido e bem fundamentado.

2.1.1.4 Trabalhos Selecionados

Após a aplicação dos critérios de inclusão e exclusão descritos anteriormente, foram selecionados 10 estudos para compor o conjunto final do mapeamento sistemático. A Tabela 4 apresenta a relação dos trabalhos selecionados, indicando os autores, uma síntese de suas contribuições e a base de dados em que foram encontrados. Esses estudos servem de fundamento para a construção da análise teórica apresentada nas subseções seguintes, oferecendo suporte conceitual e técnico às discussões desenvolvidas ao longo do capítulo.

Tabela 4 – Trabalhos selecionados após aplicação dos critérios de inclusão e exclusão

Autores	**Descrição**	**Base**
(KAPITSAKI; TSELIKAS; FOUKARAKIS, 2015)	Analisa ferramentas para identificação e verificação de compatibilidade de licenças em software livre, classificando abordagens e discutindo limitações.	IEEE Xplore
(KAPITSAKI; KRAMER; TSELIKAS, 2017)	Propõe a automação da verificação de compatibilidade de licenças por meio do padrão SPDX, modelando relações entre licenças para detecção sistemática de conflitos.	IEEE Xplore
(KOLTUN, 2011)	Discute conformidade de licenças sob perspectiva operacional e propõe três pilares para integração da compliance aos processos organizacionais.	ScienceDirect
(OMBREDANNE, 2020)	Analisa ferramentas de Software Composition Analysis (SCA), destacando técnicas de identificação automatizada de componentes e licenças.	ScienceDirect
(BALLHAUSEN, 2019)	Explica fundamentos jurídicos das licenças FOSS, diferenciando modelos copyleft e permissivos e suas principais obrigações.	IEEE Xplore
(KEMPPAINEN, 2023)	Investiga o uso de SBOM para gestão de componentes e vulnerabilidades, comparando ferramentas em estudo baseado em design science.	IEEE Xplore
(LASOTA, 2023)	Apresenta a iniciativa REUSE, baseada no padrão SPDX, propondo boas práticas para declaração padronizada de copyright e licenças.	SSOAR
(POGREBNOY et al., 2021)	Apresenta o SORREL, plugin para IDE que detecta incompatibilidades de licenças e auxilia na escolha adequada.	IEEE/ACM
(WINTERSGILL, 2024)	Investiga práticas atuais de compliance de licenças, identificando lacunas em ferramentas e propondo melhorias automatizadas e educacionais.	ACM
(PASCHALIDES; KAPITSAKI, 2016)	Apresenta o SPDX License Validation Tool (SLVT), que valida arquivos SPDX e detecta incompatibilidades por meio de grafo de licenças.	ACM

A análise dos trabalhos selecionados dispostos na Tabela 4 evidencia uma concentração de estudos voltados à conformidade de licenças, ferramentas de apoio à análise de componentes e uso do padrão SPDX em contextos específicos. Observa-se que, embora existam soluções e propostas direcionadas à automação de processos relacionados a licenciamento e validação de artefatos, ainda há espaço para abordagens que integrem de forma unificada a geração de SBOM, a análise de vulnerabilidades e a verificação de conformidade em uma única aplicação. Nesse contexto, o presente trabalho posiciona-se como uma contribuição complementar às

iniciativas identificadas, buscando consolidar essas funcionalidades em uma ferramenta acessível e integrada.

2.2 Cadeia de Suprimentos de Software

A cadeia de suprimentos de software compreende o conjunto de processos, componentes, ferramentas e agentes envolvidos ao longo de todo o ciclo de vida de um sistema, desde a concepção e desenvolvimento até a distribuição e implantação (SECURITY, 2024). Cada etapa desse processo integra uma rede de dependências que inclui bibliotecas de terceiros, repositórios, ferramentas de *build*, infraestrutura e mecanismos de distribuição. A gestão adequada dessa cadeia tem como objetivo assegurar a integridade, a segurança e a conformidade dos componentes utilizados, reduzindo riscos associados a vulnerabilidades e problemas de licenciamento.

Nos últimos anos, a cadeia de suprimentos de software passou a receber maior atenção da comunidade acadêmica e da indústria, especialmente em virtude do aumento da complexidade dos ecossistemas de desenvolvimento e da crescente dependência de componentes de terceiros (SECURITY, 2024). A utilização intensiva de bibliotecas open source, serviços externos e ferramentas automatizadas ampliou a superfície de ataque dos sistemas, tornando a visibilidade sobre os componentes utilizados um fator essencial para a gestão de riscos. Nesse contexto, mecanismos que promovam rastreabilidade, transparência e controle sobre dependências tornaram-se fundamentais para o fortalecimento da segurança e da conformidade ao longo do ciclo de vida do software.

2.2.1 Vulnerabilidades da cadeia de suprimentos de software

Atualmente, os processos de desenvolvimento e entrega de software encontram-se amplamente automatizados, apoiados por pipelines de integração e entrega contínuas (CI/CD). Nesse cenário, os projetos dependem de uma variedade crescente de ferramentas e serviços automatizados, podendo incorporar centenas ou até milhares de dependências de código aberto (SECURITY, 2024). Essa elevada interdependência amplia a complexidade do gerenciamento dos componentes utilizados e reforça a necessidade de mecanismos que garantam visibilidade, controle e rastreabilidade ao longo do ciclo de vida do software.

Diante desse cenário, os ataques à cadeia de suprimentos de software buscam explorar vulnerabilidades presentes em dependências de código aberto amplamente utilizadas no desenvolvimento de sistemas. Esses ataques podem comprometer a integridade, a confidencialidade e a disponibilidade das aplicações, uma vez que exploram componentes incorporados ao projeto de forma legítima. De acordo com SECURITY (2024), a seguir são apresentados os principais vetores de ataque associados à cadeia de suprimentos de software.

- **Ataques a gerenciadores de pacotes open source:** Nesse tipo de ataque, agentes maliciosos

exploram vulnerabilidades ou falhas lógicas nos gerenciadores de pacotes para disseminar bibliotecas maliciosas disfarçadas de componentes legítimos. Em alguns casos, pacotes contendo código malicioso podem ser publicados de forma a aparentar associação com mantenedores confiáveis, sem o seu consentimento, induzindo desenvolvedores a incorporá-los inadvertidamente aos seus projetos. Uma vez integrados ao sistema, esses pacotes podem permitir o acesso indevido a dados sensíveis, a execução de código arbitrário ou o comprometimento de diferentes etapas da cadeia de suprimentos de software.

- **Ataques ao pipeline de integração e entrega contínuas (CI/CD):** O ambiente moderno de desenvolvimento de software apresenta vulnerabilidades associadas aos processos automatizados que compõem os pipelines de integração e entrega contínuas. Diversos ataques à cadeia de suprimentos têm como objetivo comprometer essas etapas intermediárias, inserindo código malicioso durante as fases de *build*, teste ou distribuição. A limitação de visibilidade e de mecanismos de controle nesses ambientes dificulta a identificação de adulterações no código ou nos artefatos gerados.

- **Execesso de dependências de terceiros:** O desenvolvimento moderno de software depende fortemente de bibliotecas e componentes externos, especialmente de código aberto. Embora essa prática acelere o desenvolvimento, ela amplia a superfície de risco do sistema, sobretudo devido às dependências indiretas e transitivas, que reduzem a visibilidade sobre os componentes efetivamente utilizados. Como resultado, vulnerabilidades ou problemas de licenciamento podem ser incorporados ao projeto sem que haja plena consciência da equipe, afetando a segurança e a conformidade da aplicação.

2.2.2 Mecanismos de Proteção da Cadeia de Suprimentos de Software

A proteção da cadeia de suprimentos de software pode ser estruturada a partir de diferentes classes de controles de segurança, que atuam em momentos distintos do ciclo de vida do desenvolvimento. De acordo com SECURITY (2024), de forma geral, esses controles podem ser organizados em três categorias principais: (1) controles voltados à configuração segura do ciclo de desenvolvimento, (2) controles destinados à mitigação de riscos associados a dependências vulneráveis ou maliciosas e (3) mecanismos de validação da integridade e da construção segura dos artefatos de software. A seguir, essas classes são descritas de forma sucinta.

1. **Configuração segura do ciclo de desenvolvimento:** Essa classe de controles concentra-se na proteção da infraestrutura e dos processos internos utilizados para desenvolver o software. Problemas como credenciais comprometidas, permissões inadequadas e sistemas de *build* mal configurados ampliam a superfície de ataque do produtor do software. A adoção de ferramentas e práticas de segurança permite identificar lacunas na postura de segurança, fortalecer controles de acesso e reduzir a possibilidade de manipulação indevida de artefatos ou exposição de segredos sensíveis.

2. **Mitigação de dependências vulneráveis ou maliciosas:** Considerando o uso intensivo de componentes de terceiros, essa classe busca reduzir riscos associados a vulnerabilidades conhecidas, pacotes maliciosos e projetos comprometidos. Entre as práticas recomendadas estão a atualização contínua de dependências, a realização de varreduras de vulnerabilidade e a disponibilização de informações estruturadas sobre os componentes utilizados, como por meio de SBOM. Essas medidas ampliam a transparência e permitem que produtores e consumidores adotem ações corretivas de forma informada.

3. **Validação da integridade e da construção segura dos artefatos:** Essa classe de controles tem como objetivo assegurar que os artefatos gerados durante o processo de desenvolvimento não tenham sido adulterados. Para isso, são empregados mecanismos que permitem atestar continuamente a integridade das *builds* e do processo de desenvolvimento, frequentemente apoiados por técnicas criptográficas. Essa abordagem fortalece a confiança entre produtores e consumidores de software e dificulta ataques que buscam comprometer artefatos ao longo da cadeia de suprimentos.

Em síntese, proteger a cadeia de suprimentos de software envolve a adoção de controles distribuídos ao longo de todo o ciclo de desenvolvimento, desde a organização segura do ambiente até a verificação da integridade dos artefatos produzidos. A aplicação dessas medidas contribui para reduzir riscos associados a dependências vulneráveis e fortalecer a segurança dos sistemas desenvolvidos.

2.3 Conceito de Open Source

O conceito de open source vai além da definição de free software. Enquanto free software possui um enfoque na liberdade dos usuários, garantindo-lhes o direito de executar, copiar, distribuir, estudar, modificar e aprimorar o software, o termo *"free"* refere-se à liberdade, e não necessariamente à gratuidade (GNU, 2025). Já o open source é definido por um conjunto de 10 critérios que devem ser atendidos pelos termos de distribuição do software (BALLHAUSEN, 2019). Apesar do nome, esses critérios abrangem mais do que a simples acessibilidade ao código-fonte.

Entre os dez critérios do open source, os quatro primeiros refletem as quatro liberdades do software livre, definidas por Richard Stallman em conjunto com a Free Software Foundation (BALLHAUSEN, 2019). Dentre eles, o segundo critério estabelece a obrigatoriedade da acessibilidade ao código-fonte, um requisito que anteriormente era considerado apenas uma pré-condição para as quatro liberdades.

1. A redistribuição de forma gratuita do software como um componente de distribuição de software agregada possuindo fontes diversas, não pode ser restringida. A concessão de licença não pode exigir qualquer tipo de taxa ou pagamento.

2. O código-fonte do software deve estar acessível, preferencialmente ao ser distribuído ou, através de um meio com ampla divulgação para obtê-lo sem custo superior ao de reprodução razoável, idealmente através de download gratuito na internet. Código deliberadamente ofuscado ou formas intermediárias como a saída de um pré-processador ou tradutor não são permitidas.

3. Devem ser permitidas as modificações e criações de trabalhos derivados, assim como a distribuição sob os mesmos termos da licença do software original.

4. A distribuição de versões modificadas do software só pode ser restringido com o intuito de conservar a integridade do código-fonte do autor. A distribuição de software produzido a partir de código-fonte modificado não pode ser restringida. No caso da licença restringir a distribuição do código-fonte em sua forma modificada, deve-se permitir a disseminação de arquivos de patch junto ao código-fonte, para que permita a modificação do programa no instante da compilação.

De acordo com Ballhausen (2019), os últimos seis critérios costumam ser menos claros, apesar de frequentemente serem o principal diferencial entre FOSS em relação aos demais modelos de licenciamento perante os quais o software pode ser distribuído gratuitamente.

5. Os direitos de uso devem ser concedidos sem descriminação de nenhuma pessoa ou grupo.

6. A licença não deve restringir o uso do software em nenhum dos campos de aplicação.

7. Os direitos de uso do software devem ser aplicados a todos a quem o programa seja redistribuído, sem necessidade de execução de uma licença adicional.

8. Os direitos de uso devem ser aplicáveis de forma independente do produto onde o software está sendo utilizado.

9. A licença não deve impor restrições em relação a outros sotwares que sejam distruídos com o software licenciado. Não pode haver a obrigatoriedade dos demais softwares serem de código aberto.

10. A licença não pode determinar nenhuma linguagem ou estilo de interface específica para o software.

Dessa forma, o conceito de open source se estabelece como um modelo de desenvolvimento e distribuição de software que vai além do simples acesso ao código-fonte. Ao definir critérios que garantem liberdade de uso, modificação e distribuição, ele promove a colaboração, a inovação e a transparência no ecossistema de software. Embora compartilhe princípios fundamentais com o free software, o open source se diferencia por seu foco na viabilidade prática e na adoção ampla por diversas indústrias e comunidades.

2.4 Entendendo o SPDX

O System Package Data Exchange (SPDX) é um padrão aberto, desenvolvido pela Linux Foundation, que define um formato padronizado para a comunicação de informações sobre componentes de software em um projeto. Ele permite a especificação estruturada de dados como procedência, licenciamento, segurança e outras características relevantes (SPDX, 2025). O SPDX facilita o compartilhamento de dados essenciais entre organizações e comunidades, reduzindo redundâncias, e simplificando e aprimorando a conformidade, a segurança e a confiabilidade do software.

O projeto SPDX conta com a participação de representantes de diversas organizações, incluindo fornecedores de software, desenvolvedores de sistemas e ferramentas, fundações e integradores de sistemas (SPDX, 2025). Sua estrutura de trabalho é organizada em três subgrupos: a equipe de tecnologia, responsável pelos aspectos técnicos; a equipe jurídica, focada em questões de licenciamento e conformidade; e a equipe de divulgação, encarregada de promover e disseminar o padrão. A composição do projeto é constituída por:

- A especificação SPDX em si.
- A lista de licenças SPDX englobando exceções, diretrizes de correspondência, identificadores de licença (IDs) e a sintaxe para expressões de licença.
- Conjunto de ferramentas e bibliotecas SPDX para manipulação de documentos e da lista de licenças do padrão.

O SPDX é organizado em áreas de interesse, conhecidas como perfis, que atendem a diferentes necessidades dos usuários. O padrão abrange oito perfis distintos, cada um com um foco específico: SPDX Security, SPDX Licensing, SPDX IA, SPDX Data, SPDX Build, SPDX Lite, SPDX Core e SPDX Software (SPDX, 2025). Essa estrutura modular permite maior flexibilidade e adaptação às diversas demandas do ecossistema de software.

- SPDX Security: captura informações relacionadas à segurança em um Documento de Segurança SPDX. Especificamente, fornece detalhes sobre possíveis vulnerabilidades de software, sua gravidade e um mecanismo para indicar como essas vulnerabilidades afetam componentes específicos do software, além de informar se uma correção está disponível.
- SPDX Licensing: captura informações relevantes sobre o licenciamento de software e propriedade intelectual. Especificamente, ele facilita a identificação das licenças e dos avisos de direitos autorais estabelecidos por indivíduos ou ferramentas automatizadas para aplicação nas distribuições de software.
- SPDX IA: oferece um inventário completo de componentes de software e dependências relacionadas a modelos e sistemas de IA e aprendizado de máquina. Esse perfil abrange

estruturas de software, bibliotecas e outros componentes utilizados para o desenvolvimento e implantação de sistemas de IA, fornecendo informações detalhadas sobre versões, licenças, questões de segurança, além de aspectos éticos e de conformidade.

- SPDX Data: Um documento ou representação que captura informações essenciais sobre os conjuntos de dados usados em sistemas de IA ou outros aplicativos, incluindo nomes, versões, fontes, metadados, licenciamento e outras características relevantes. Ele descreve a estrutura, formato, conteúdo e propriedades dos dados, facilitando sua análise e compreensão pelos usuários.
- SPDX Build: captura detalhes sobre o processo de construção de software, incluindo entradas, saídas, procedimentos, ambientes e atores envolvidos, além das evidências associadas, descrevendo como o software é gerado e transformado.
- SPDX Light: captura as informações essenciais para garantir a conformidade de licenças na cadeia de fornecimento de software. Ele inclui dados sobre a criação do SBOM, listas de pacotes com licenciamento e outros itens relacionados, juntamente com seus respectivos relacionamentos.

Segundo SPDX (2025), entre os oito perfis definidos pelo padrão, SPDX Core e SPDX Software formam a base sobre a qual os demais perfis são estruturados. O perfil Core abrange as definições de classes, propriedades e vocabulários essenciais, sendo aplicável a todos os perfis SPDX. Já o perfil SPDX Software estende o perfil Core para aplicações específicas em software. Embora compartilhe várias semelhanças com o Core, ele é utilizado em todos os perfis que lidam com software.

2.5 A Especificação SPDX

A especificação SPDX é, essencialmente, um padrão, assim como o HTML ou o IEEE 802.11g. Seu propósito é facilitar a troca eficiente de software, apoiando cadeias de fornecimento que dependem de software de código aberto (OSS). As organizações colaboram na construção da especificação SPDX ao compartilhar suas próprias rotinas e negociar como elas devem ser representadas no padrão. A primeira versão da especificação, por exemplo, foi desenvolvida sem testes prévios, baseada em suposições sobre as práticas de gerenciamento de riscos em OSS e sua documentação em uma especificação compartilhada (GANDHI; GERMONPREZ; LINK, 2018). Com cada nova versão, as experiências de implementação e o feedback dos membros das organizações contribuíram para a evolução e aprimoramento do SPDX, tornando-o mais alinhado aos desafios reais do gerenciamento de riscos em OSS.

De acordo com SPDX (2025), existem três formas distintas de se envolver com o SPDX, e elas são mutuamente exclusivas, ou seja, pode-se escolher qualquer uma sem a necessidade de

realizar as outras. No entanto, cada uma delas oferece valor de forma única, contribuindo para a identificação de licenças e, por fim, para garantir a conformidade.

Uma das formas mais simples de começar com o SPDX é utilizando a lista de licenças SPDX ou os identificadores de licença diretamente no código-fonte. No entanto, o principal potencial do SPDX está na criação e/ou utilização de documentos SPDX.

Um documento SPDX armazena metadados sobre um pacote de software, sendo organizado conforme a especificação SPDX (GANDHI; GERMONPREZ; LINK, 2018). Ele inclui campos para o nome do pacote, número da versão, licença, URLs para acessar anúncios de vulnerabilidades e os relacionamentos do pacote com outros pacotes (como, por exemplo, "é uma cópia de"ou "é pré-requisito para").

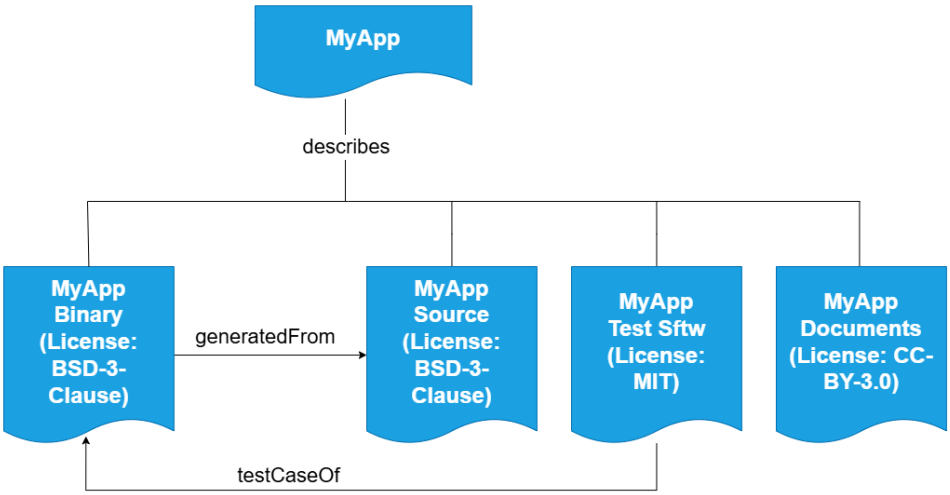
Ainda que a lista de licenças SPDX e os identificadores de licença no código-fonte garantam maior precisão na transmissão de uma licença, é o documento SPDX que oferece um nível mais detalhado de informações por pacote ou arquivo. Para isso, ele pode ser gerado em cinco formatos distintos: tag/valor (.spdx), JSON (.spdx.json), YAML (.spdx.yml), RDF/XML (.spdx.rdf) e planilhas (.xls) (SPDX, 2025). Qualquer um desses formatos pode ser utilizado, e há ferramentas disponíveis para converter entre eles conforme necessário. A escolha do formato ideal deve considerar as necessidades do projeto de software e a compatibilidade com as ferramentas empregadas.

2.5.1 Pacotes e Relacionamentos

A introdução da noção de relacionamentos na versão 2.0 da especificação SPDX ampliou a capacidade dos documentos SPDX, permitindo que abordem casos de uso mais complexos, permitindo que muitos casos de uso do mundo real fossem analisados e capturados (SPDX, 2025). Com essa melhoria, os documentos podem se referir entre si, indicando claramente o tipo de relacionamento existente entre eles.

Para exemplificar sua utilização, pode-se utilizar um exemplo apresentado na Figura 1 a seguir, onde o uso de relacionamentos descreve o licenciamento de todos os itens que fazem parte de um aplicativo, demonstrando também como essas partes se conectam.

Figura 1 – Exemplo de relacionamento entre componentes de um software através da especificação SPDX.



Fonte: Autor (2025).

Nesse exmeplo, o aplicativo possui softwares de teste e documentação que o acompanham. O aplicativo "MyApp"aponta para cada um deles para demonstrar que estão "relacionados"com o aplicativo e que vieram com ele. Por sua vez, todos os demais documentos (binário, código-fonte, teste e documentação) descrevem como se encaixam e qual o licenciamento específico de cada um deles. Com isso é possível fornecer uma visão completa sobre o licenciamento, já que as fontes usadas para criar o aplicativo e tudo que é vinculado a ele são componentes já conhecidos.

A introdução de relacionamentos na especificação SPDX permite enxergar o software com uma visão mais ampla do que apenas como um aglomerado de pacotes de componentes. O relacionamento traz uma carga semântica para cada componente que faz parte do software, possibilitando descrever como esses componentes interagem, se dependem uns dos outros. Isso enriquece significativamente a análise de conformidade, segurança e governança, permitindo um rastreamento mais preciso da origem, função e impacto de cada parte dentro da cadeia de fornecimento de software.

2.5.2 Lista de Licenças SPDX

Como parte integrante da especificação SPDX, a Lista de Licenças SPDX é composta por licenças e exceções comumente encontradas em softwares, dados, hardwares ou documentações livres, abertos ou colaborativos. A Lista de Licenças SPDX é composta por um identificador curto padronizado, o nome completo da licença, o texto correspondente a ela e uma URL permanente e canônica para cada uma das licenças e exceções. O objetivo principal da Lista de Licenças SPDX é proporcionar, de forma eficiente e confiável, a identificação dessas licenças e exceções, seja em documentos SPDX, arquivos de código-fonte ou em outros locais.

A Lista de Licenças SPDX apresentada na Figura 2, em junho de 2025, já contava com mais de 690 licenças e exceções catalogadas. Ela está disponível no site oficial do projeto SPDX, com URLs permanentes que não sofrem alterações, garantindo referências estáveis. Um de seus principais diferenciais é o uso de uma linguagem de expressão simples, que permite representar relações de licenciamento tanto conjuntivas quanto disjuntivas. Cada licença ou exceção é identificada por um identificador curto padronizado, facilitando sua referência e uso, especialmente em arquivos de código-fonte e documentos técnicos (SPDX, 2025). Além disso, a lista fornece o texto exato de cada licença, acompanhado de diretrizes de correspondência que auxiliam na verificação de compatibilidade entre textos de licença e os modelos incluídos. Esses modelos indicam claramente quais trechos do texto são opcionais ou substituíveis, conforme as diretrizes estabelecidas, promovendo precisão e consistência na identificação de licenças.

Figura 2 – Algumas das licenças e exceções encontradas na Lista de Licenças SPDX.

Full name	Identifier	FSF Free/Libre?	OSI Approved?
BSD Zero Clause License	0BSD		Y
Attribution Assurance License	AAL		Y
Abstyles License	Abstyles		
AdaCore Doc License	AdaCore-doc		
Adobe Systems Incorporated Source Code License Agreement	Adobe-2006		
Adobe Glyph List License	Adobe-Glyph		
Amazon Digital Services License	ADSL		
Academic Free License v1.1	AFL-1.1	Y	Y
Academic Free License v1.2	AFL-1.2	Y	Y
Academic Free License v2.0	AFL-2.0	Y	Y
Academic Free License v2.1	AFL-2.1	Y	Y
Academic Free License v3.0	AFL-3.0	Y	Y
Afmparse License	Afmparse		
Affero General Public License v1.0 only	AGPL-1.0-only		
Affero General Public License v1.0 or later	AGPL-1.0-or-later		
GNU Affero General Public License v3.0 only	AGPL-3.0-only	Y	Y
GNU Affero General Public License v3.0 or later	AGPL-3.0-or-later	Y	Y
Aladdin Free Public License	Aladdin		
AMD's plpa_map.c License	AMDPLPA		
Apple MIT License	AML		
Academy of Motion Picture Arts and Sciences BSD	AMPAS		
ANTLR Software Rights Notice	ANTLR-PD		
ANTLR Software Rights Notice with license fallback	ANTLR-PD-fallback		
Apache License 1.0	Apache-1.0	Y	
Apache License 1.1	Apache-1.1	Y	Y
Apache License 2.0	Apache-2.0	Y	Y
Adobe Postscript AFM License	APAFML		
Adaptive Public License 1.0	APL-1.0		Y
App::s2p License	App-s2p		
Apple Public Source License 1.0	APSL-1.0		Y
Apple Public Source License 1.1	APSL-1.1		Y
Apple Public Source License 1.2	APSL-1.2		Y
Apple Public Source License 2.0	APSL-2.0	Y	Y
Arphic Public License	Arphic-1999		
Artistic License 1.0	Artistic-1.0		Y

Fonte: https://spdx.dev/learn/overview/

2.5.2.1 Identificadores de Licença SPDX no Código-fonte

Em projetos que utilizam software de código aberto, a identificação precisa da licença é essencial não apenas para a geração de relatórios e auditorias, mas também para assegurar que o uso, a modificação e a redistribuição do código estejam em conformidade com os termos legais estabelecidos pela licença.

Entretanto, definir a licença a ser utilizada pode ser uma tarefa desafiadora, seja pela ausência de dados ou pela ambiguidade das informações disponíveis. Mesmo quando há indícios de licenciamento, a falta de uma notação padronizada para representá-los pode dificultar a automatização da detecção de licenças, exigindo, assim, um esforço humano significativo (SPDX, 2025). O grupo de trabalho SPDX recomenda o uso de identificadores de licença SPDX para especificar a licença no nível de arquivo. Entre as vantagens dessa abordagem, destacam-se as seguintes:

- **Eliminação de ambiguidades:** evita variações no texto do cabeçalho da licença, garantindo precisão na identificação.
- **Neutralidade linguística:** os identificadores são independentes do idioma, facilitando o uso em contextos internacionais.
- **Facilidade de processamento automatizado:** são facilmente interpretados por máquinas, o que favorece a automação.
- **Concisão:** os identificadores são curtos e diretos, ocupando pouco espaço no código.
- **Acompanhamento da licença com o arquivo:** útil especialmente quando arquivos de licença são removidos ou não incluídos no projeto.
- **Simplicidade de uso:** podem ser utilizados com facilidade, mesmo em ambientes interpretados como JavaScript.
- **Imutabilidade:** cada identificador SPDX é permanente e não sofre alterações ao longo do tempo.
- **Orientação clara para desenvolvedores:** ajuda os autores do código a garantirem que a licença aplicada seja compreendida e respeitada.

Considerando os benefícios apresentados, adotar identificadores de licença SPDX diretamente no código-fonte é uma maneira prática de lidar com os obstáculos comuns na identificação e verificação de licenças em projetos de código aberto. Essa abordagem padroniza a forma como as informações de licenciamento são fornecidas, tornando o processo mais transparente e menos sujeito a erros. Além disso, ela ajuda a evitar problemas legais e facilita tanto o trabalho de quem desenvolve quanto o de quem realiza auditorias ou utiliza ferramentas automatizadas para análise de código.

2.6 Tecnologias e Frameworks

O desenvolvimento de aplicações desktop pode adotar diferentes abordagens, destacando-se o desenvolvimento nativo, voltado a um sistema operacional específico, e o desenvolvimento

híbrido ou multiplataforma, que permite a execução da aplicação em diferentes ambientes a partir de um único código-base.

No desenvolvimento nativo, a aplicação é construída considerando as particularidades de cada sistema operacional, o que possibilita maior controle sobre os recursos e melhor desempenho, porém com maior esforço de desenvolvimento e manutenção. Em contraste, a abordagem híbrida e multiplataforma favorece a reutilização de código, reduz a complexidade do processo de desenvolvimento e facilita a manutenção da aplicação, ao integrar interfaces desenvolvidas com tecnologias web a camadas nativas do sistema operacional.

Diante das características do sistema proposto e dos requisitos definidos neste trabalho, optou-se pela adoção de uma abordagem híbrida para o desenvolvimento da aplicação desktop. Para isso, foi utilizado o framework Tauri, com a interface desenvolvida com o React (biblioteca JavaScript) em conjunto com o Vite, enquanto a camada nativa responsável pela comunicação com o sistema operacional é implementada em Rust, linguagem utilizada pelo Tauri por oferecer desempenho e segurança.

A partir dessas definições, as tecnologias e frameworks empregados no desenvolvimento do sistema são apresentados e detalhados a seguir.

2.6.0.1 Tauri

O Tauri (TAURI, 2026) é um framework para o desenvolvimento de aplicações desktop multiplataforma que combina interfaces construídas com tecnologias web a uma camada nativa responsável pela interação com o sistema operacional. Essa abordagem possibilita a criação de aplicações leves e eficientes, uma vez que o framework utiliza o mecanismo de renderização web nativo de cada plataforma, reduzindo o consumo de recursos.

A camada nativa do Tauri é implementada em Rust e é responsável pelo acesso aos recursos do sistema operacional, oferecendo bom desempenho e segurança de memória. A comunicação entre a interface e essa camada ocorre por meio de comandos bem definidos, favorecendo a organização da aplicação e a separação de responsabilidades. Assim, o Tauri 2.0 mostra-se adequado ao desenvolvimento do sistema proposto, em consonância com a arquitetura e os requisitos estabelecidos neste trabalho.

2.6.0.2 React

O React (REACT, 2026) é uma biblioteca JavaScript voltada para a construção de interfaces de usuário, baseada na composição de componentes reutilizáveis. Essa abordagem facilita a organização da interface e a manutenção do código, permitindo a construção de aplicações de forma modular e estruturada.

No contexto deste trabalho, o React é utilizado para o desenvolvimento da interface gráfica da aplicação desktop, possibilitando a criação de uma experiência de usuário responsiva e

organizada. Sua integração com o Tauri permite a separação clara entre a camada de apresentação e a lógica nativa da aplicação, contribuindo para a coerência da arquitetura proposta.

2.6.0.3 Vite

O Vite (VITE, 2026) é uma ferramenta de build e desenvolvimento voltada a aplicações frontend, projetada para oferecer um ambiente de desenvolvimento rápido e eficiente. Ele utiliza recursos modernos do ecossistema JavaScript para proporcionar inicialização ágil do projeto e recarregamento rápido durante o desenvolvimento.

Neste trabalho, o Vite é empregado como ferramenta de apoio ao desenvolvimento da interface construída com React, facilitando a organização do projeto e o processo de build da aplicação. Sua utilização contribui para maior produtividade durante o desenvolvimento e para a integração da interface com o ambiente desktop fornecido pelo Tauri.

2.6.0.4 Rust

Rust (RUST, 2026) é uma linguagem de programação de sistemas projetada com foco em desempenho, segurança de memória e confiabilidade. Seu modelo de gerenciamento de memória, baseado em verificação em tempo de compilação, reduz a ocorrência de erros comuns, como acessos inválidos e vazamentos de memória, sem a necessidade de um coletor de lixo.

No contexto deste trabalho, o Rust é utilizado na camada nativa da aplicação por meio do framework Tauri, sendo responsável pela interação com o sistema operacional e pela execução das funcionalidades que demandam acesso a recursos nativos. O uso de Rust contribui para uma implementação robusta e segura, alinhada às exigências de desempenho e estabilidade do sistema proposto.

2.7 Ferramentas de Análise de SBOM

Além das tecnologias empregadas no desenvolvimento da aplicação, o sistema integra ferramentas especializadas para a geração e análise de SBOMs. Essas ferramentas são responsáveis por funcionalidades essenciais do sistema, como a geração e o enriquecimento da SBOM no padrão SPDX, bem como a verificação de vulnerabilidades e licenças, sendo descritas a seguir.

2.7.0.1 Syft

O Syft (ANCHORE, 2026c) é uma ferramenta utilizada para a geração de Listas de Materiais de Software (SBOM), permitindo a identificação dos componentes e dependências presentes em um projeto de software. Ele suporta a geração de SBOMs em diferentes formatos padronizados, incluindo o padrão SPDX, sendo amplamente empregado em processos de análise e conformidade de software.

No contexto deste trabalho, o Syft é utilizado como a principal ferramenta para a geração inicial da SBOM a partir do projeto analisado, fornecendo as informações básicas necessárias para as etapas subsequentes de enriquecimento e análise.

2.7.0.2 Parlay

O Parlay (SNYK, 2026) é uma ferramenta voltada ao enriquecimento de SBOMs, permitindo a complementação das informações previamente geradas com metadados adicionais relevantes. Seu uso contribui para a melhoria da qualidade e da completude da SBOM, facilitando análises posteriores relacionadas à segurança e à conformidade de licenças.

Neste sistema, o Parlay é empregado após a geração da SBOM pelo Syft, sendo responsável por enriquecer o arquivo no padrão SPDX antes de sua utilização nas análises de vulnerabilidades e licenças.

2.7.0.3 Grype

O Grype (ANCHORE, 2026b) é uma ferramenta especializada na identificação de vulnerabilidades em componentes de software, realizando análises com base em uma SBOM previamente gerada. A partir dessas informações, o Grype compara os componentes identificados com bases conhecidas de vulnerabilidades, permitindo a detecção de possíveis riscos de segurança.

No sistema desenvolvido, o Grype é utilizado para a verificação de vulnerabilidades a partir da SBOM no padrão SPDX, fornecendo informações que auxiliam na avaliação da segurança dos componentes utilizados no projeto analisado.

2.7.0.4 Grant

O Grant (ANCHORE, 2026a) é uma ferramenta voltada à análise de licenças de software, permitindo a identificação das licenças associadas aos componentes descritos em uma SBOM. Essa análise auxilia na verificação de conformidade e no entendimento das implicações legais relacionadas ao uso de dependências de terceiros.

No contexto deste trabalho, o Grant é utilizado para realizar a análise de licenças com base na SBOM no padrão SPDX, disponibilizando informações relevantes sobre as licenças empregadas e seus respectivos níveis de copyleft.

3

Busca de Anterioridade

3.1 Ferramentas Desenvolvidas Pela Comunidade SPDX

Reconhecendo a importância de tornar o uso da especificação mais acessível e eficiente, o projeto SPDX incentiva o desenvolvimento de ferramentas que estejam em estrita conformidade com suas diretrizes, as quais auxiliem tanto aos elaboradores quanto aos consumidores de documentos SPDX.

Segundo SPDX (2025), a comunidade SPDX desenvolve e dá suporte a um conjunto diversificado de ferramentas destinadas à aplicação da especificação. Tais ferramentas estão organizadas em três categorias distintas: ferramentas online, de acesso gratuito, voltadas à validação, conversão e comparação de documentos SPDX; ferramentas de construção, com ênfase na criação de plugins para gerenciadores de pacotes como Gradle e Maven, desenvolvidas com o apoio da comunidade; e bibliotecas compatíveis com linguagens de programação amplamente utilizadas, como Java, Python e Go.

A SPDX Online Tool (SPDX PROJECT, 2025), apresentada na Figura 3 disponibiliza uma plataforma abrangente voltada ao tratamento de documentos compatíveis com a especificação. Por meio dessa ferramenta, é possível realizar o carregamento de documentos SPDX para fins de análise, validação, comparação e conversão entre diferentes formatos. Além disso, o portal permite consultas diretas à Lista de Licenças SPDX, facilitando a verificação e o uso de identificadores padronizados.

As ferramentas de construção mantidas pela comunidade SPDX oferecem integração com projetos que utilizam o Gradle, uma ferramenta de automação de *builds* amplamente empregada em aplicações Java, Android e demais linguagens baseadas na JVM. Além disso, há suporte para o Maven, uma das principais ferramentas de automação, construção e gerenciamento de dependências em projetos Java.

Figura 3 – Ferramenta SPDX Online Tool.

Fonte: https://tools.spdx.org/app/about/

O plugin para Gradle tem como principal objetivo a geração de SBOMs (*Software Bill of Materials*) no formato JSON, conforme a especificação SPDX (SDPX-GRADLE, 2023). Embora os recursos relacionados a direitos autorais e licenciamento ainda não estejam completamente finalizados, eles já apresentam um funcionamento satisfatório para dependências gerenciadas via Maven.

O plugin para Maven oferece suporte à geração de documentos compatíveis com a especificação SPDX durante o processo de *build*. Seu principal objetivo é produzir arquivos SPDX com base nos artefatos descritos no arquivo POM do projeto. O tratamento de licenças no Maven pode ser complexo, e recomenda-se, sempre que possível, o uso de identificadores de licença padrão da SPDX (SDPX-MAVEN, 2015). Caso não haja uma licença padrão aplicável, é necessário declarar uma nonStandardLicense como parâmetro, informando um identificador exclusivo e incluindo o texto completo da licença.

As bibliotecas para Java, Python e Go, mantidas pela comunidade SPDX, têm como objetivo fornecer utilitários que auxiliam na criação, conversão, comparação e validação de documentos SPDX em diversos formatos. Cada uma dessas bibliotecas oferece suporte específico para sua respectiva linguagem-alvo, visando facilitar a adoção das especificações SPDX em projetos de software desenvolvidos com essas tecnologias.

3.2 Ferramentas de Terceiros Com Suporte ao SPDX

Além das ferramentas e plugins mantidos pela comunidade SPDX, existem diversas outras soluções comerciais e de código aberto que oferecem suporte à especificação. Entre as opções de código aberto, destacam-se ferramentas como Fossology, Parlay e Syft, amplamente utilizadas para análise de licenças, geração de SBOMs e verificação de conformidade de software.

O Fossology (FOSSOLOGY PROJECT, 2025) é um sistema e kit de ferramentas de código aberto voltado para a verificação de conformidade de licenças de software. Como kit de ferramentas, permite a execução de análises de licenças e direitos autorais diretamente pela linha de comando, além de oferecer opções de exportação. Como sistema, disponibiliza uma interface web e um banco de dados para facilitar o gerenciamento e o fluxo de trabalho relacionado à conformidade. O Fossology permite gerar arquivos no formato SPDX ou arquivos readme contendo todos os avisos de direitos autorais identificados em um determinado software.

O Parlay (SNYK, 2026) é uma ferramenta cujo objetivo é enriquecer SBOMs com dados adicionais obtidos por meio de serviços de terceiros. Esse processo de enriquecimento consiste na adição de informações ao SBOM original, que muitas vezes contém apenas dados básicos, como o nome e a versão de determinados pacotes. A partir disso, gera-se um SBOM mais completo e informativo, o que contribui para uma tomada de decisão mais embasada em relação aos componentes utilizados no software.

Por sua vez, Syft (ANCHORE, 2026c) é uma ferramenta de linha de comando e também uma biblioteca escrita em Go, projetada para gerar Listas de Materiais de Software (SBOM) a partir de imagens de contêiner e sistemas de arquivos. Sua eficiência pode ser ampliada quando utilizada em conjunto com o Grype (ANCHORE, 2026b), um scanner de vulnerabilidades que realiza a detecção de falhas de segurança diretamente em imagens de contêiner e sistemas de arquivos, promovendo uma análise integrada e automatizada.

Para posicionar o Rivet no contexto das ferramentas que utilizam o padrão SPDX, realizou-se uma análise comparativa das principais funcionalidades oferecidas por cada solução disponível. Essa comparação busca identificar convergências, limitações e lacunas nas ferramentas atualmente utilizadas, permitindo compreender de que forma o sistema desenvolvido se insere nesse cenário. Foram considerados aspectos como geração de SBOM, análise de vulnerabilidades, verificação de licenças, disponibilidade de interface gráfica, integração das análises e possibilidade de uso local, por se tratarem de características diretamente relacionadas ao gerenciamento da cadeia de suprimentos de software. A definição desses critérios contemplou tanto funcionalidades técnicas quanto elementos ligados à usabilidade e à centralização das análises. O Quadro 1 sintetiza os resultados dessa comparação, evidenciando o posicionamento do Rivet em relação às demais ferramentas.

Quadro 1 – Comparação de funcionalidades entre ferramentas que utlizam o padrão SPDX e o Rivet

Funcionalidades	**SPDX Online Tool**	**Fossology**	**Syft**	**Grype**	**Parlay**	**Rivet**
Geração de SBOM (SPDX)	Sim	Sim	Sim	Não	Não	Sim
Análise de vulnerabilidades	Não	Não	Não	Sim	Não	Sim
Análise de licenças	Não	Não	Não	Não	Não	Sim
Interface gráfica	Sim (web)	Sim (web)	Não	Não	Não	Sim (desktop)
Integração das análises	Não	Não	Não	Não	Não	Sim
Uso local	Não	Não	Sim	Sim	Sim	Sim

Conforme evidenciado no Quadro 1, observa-se que as ferramentas analisadas apresentam foco predominante em funcionalidades específicas e isoladas. O Syft, por exemplo, destaca-se na geração de SBOM, enquanto o Grype concentra-se na análise de vulnerabilidades. Já ferramentas como SPDX Online Tool e Fossology oferecem interface gráfica baseada na web, porém não integram múltiplas análises em uma única solução. Nesse contexto, o Rivet distingue-se por reunir, em uma aplicação desktop unificada, funcionalidades de geração de SBOM, análise de vulnerabilidades e verificação de licenças, além de permitir execução local e consolidação dos resultados em um único ambiente. Essa abordagem integrada reforça a proposta do sistema ao promover maior centralização, organização e simplificação no gerenciamento da cadeia de suprimentos de software.

A identificação dos componentes FOSS em uso é a etapa inicial fundamental para qualquer iniciativa de conformidade. Essa tarefa, no entanto, pode ser complexa e demorada, especialmente quando envolve o uso de diversos componentes de software de terceiros em um programa. Para lidar com essa complexidade de forma mais eficiente, é possível recorrer a ferramentas e processos automatizados, que oferecem ganhos significativos em tempo e precisão. Atualmente, a prática de identificar esses componentes é amplamente conhecida como Análise de Composição de Software (SCA), termo amplamente adotado por analistas da indústria, como Gartner e Forrester (OMBREDANNE, 2020). A SCA vai além da simples verificação de conformidade com licenças FOSS, abrangendo também a identificação de vulnerabilidades de segurança e atributos de qualidade dos componentes utilizados.

4

Desenvolvimento do Software

Este capítulo descreve o desenvolvimento do software Rivet, uma aplicação desktop voltada ao gerenciamento da cadeia de suprimentos de software por meio da utilização do padrão SPDX. A solução foi concebida com o objetivo de integrar, em uma única interface, funcionalidades de geração de SBOMs, verificação de vulnerabilidades e análise de conformidade de licenças.

A partir desse contexto, são apresentadas as decisões de projeto, a arquitetura adotada e as etapas de implementação realizadas para atender aos requisitos definidos. O desenvolvimento foi conduzido de forma incremental, permitindo a implementação e validação progressiva das funcionalidades. Também são detalhados o levantamento de requisitos, os diagramas utilizados na definição da arquitetura e os mecanismos empregados para a geração de SBOMs no padrão SPDX, a análise de licenças e o monitoramento de vulnerabilidades em componentes de software.

4.1 Especificação e Estruturação da Solução

Esta etapa foi dedicada ao levantamento dos requisitos do software, incluindo requisitos funcionais e não funcionais, bem como a definição de suas respectivas prioridades. Também são apresentados os diagramas utilizados para a definição da arquitetura e a delimitação do escopo da solução.

4.1.1 Requisitos Funcionais e Não Funcionais

A definição de requisitos de um software consiste no processo de identificar, analisar e documentar as necessidades que o sistema deve atender, especificando suas funcionalidades, restrições e atributos de qualidade, de modo a orientar o projeto, o desenvolvimento e a validação da solução.

Os requisitos de software podem ser classificados em requisitos funcionais, que descrevem as funcionalidades do sistema, requisitos não funcionais, que estabelecem atributos de qualidade e restrições, e requisitos de domínio, que refletem regras e características específicas do contexto em que o sistema está inserido.

Com o objetivo de delimitar de forma clara o escopo da solução proposta, os requisitos do software foram identificados e organizados a partir das necessidades levantadas durante a etapa de análise. Esses requisitos estão classificados em requisitos funcionais e requisitos não funcionais, conforme apresentado no Quadro 2 e Quadro 3 respectivamente.

Quadro 2 – Requisitos funcionais do sistema

Código	**Título**	**Prioridade**	**Descrição**
RF01	Criar arquivo SBOM SPDX	Alta	O sistema deve permitir a geração de uma Lista de Materiais de Software (SBOM) no padrão SPDX.
RF02	Visualizar SBOM	Alta	O sistema deve permitir a visualização do arquivo SBOM gerado.
RF03	Exportar SBOM	Alta	O sistema deve permitir a exportação do arquivo SBOM gerado no formato *spdx.json*.
RF04	Verificar vulnerabilidades	Alta	O sistema deve permitir a verificação de vulnerabilidades nos componentes do software analisado com base no arquivo SBOM gerado.
RF05	Listar vulnerabilidades	Alta	O sistema deve permitir a listagem das vulnerabilidades identificadas.
RF06	Detalhar vulnerabilidades	Média	O sistema deve permitir a disponibilização de informações relevantes sobre cada vulnerabilidade identificada.
RF07	Exportar vulnerabilidades	Média	O sistema deve permitir a exportação das vulnerabilidades identificadas para um arquivo no formato JSON.
RF08	Verificar licenças	Alta	O sistema deve permitir a verificação das licenças associadas a cada componente do software analisado com base no arquivo SBOM gerado.
RF09	Listar licenças	Alta	O sistema deve permitir a listagem das licenças associadas a cada componente do software analisado.
RF10	Exibir nível de copyleft	Média	O sistema deve permitir a disponibilização do nível de *copyleft* da licença de cada componente do software analisado.
RF11	Exportar licenças	Média	O sistema deve permitir a exportação das informações de licenciamento para um arquivo no formato JSON.

Quadro 3 – Requisitos não funcionais do sistema

Código	**Título**	**Prioridade**	**Descrição**
RNF01	Usabilidade	Alta	O sistema deve possuir interface gráfica intuitiva, permitindo que usuários com conhecimento básico em desenvolvimento de software consigam utilizá-lo sem necessidade de treinamento prévio.
RNF02	Confiabilidade	Alta	O sistema deve garantir a integridade das informações geradas, evitando inconsistências nos dados da SBOM.
RNF03	Portabilidade	Alta	O sistema deve ser executável nos principais sistemas operacionais de uso comum em ambientes de desenvolvimento.
RNF04	Manutenibilidade	Alta	O sistema deve possuir arquitetura modular, facilitando a manutenção e a evolução das funcionalidades.
RNF05	Segurança	Alta	O sistema deve garantir que os arquivos analisados não sejam alterados durante o processo de geração da SBOM.

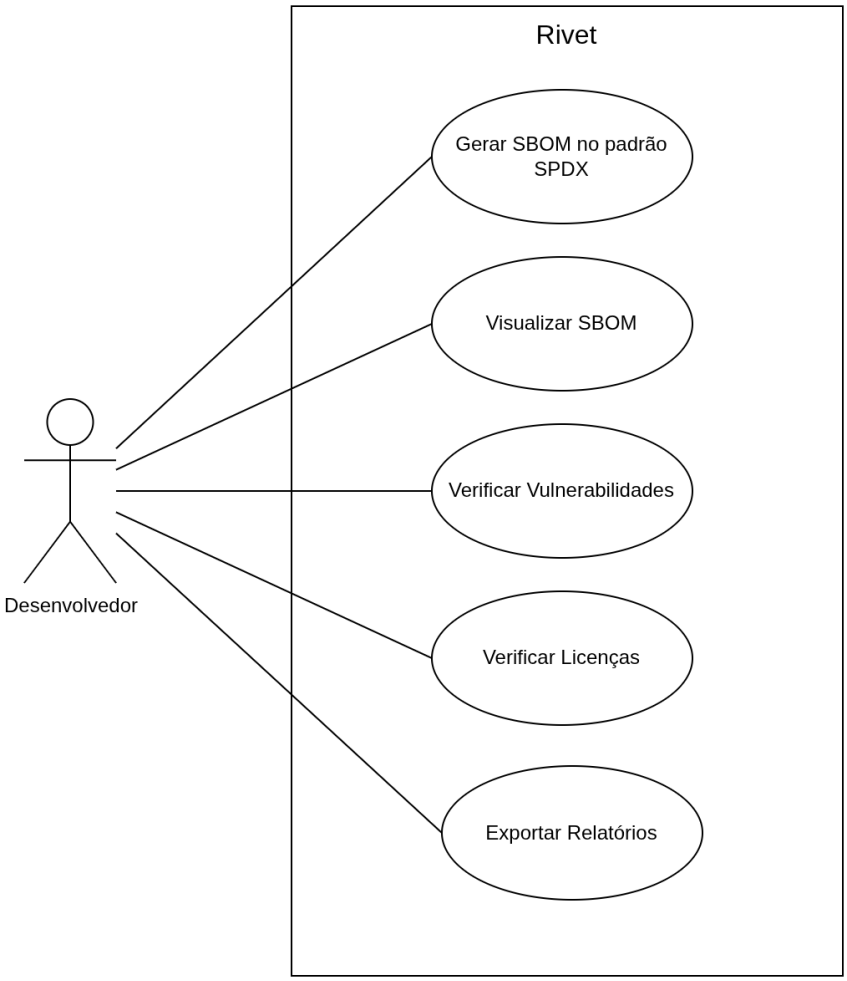
4.1.2 Diagramas

Com base nos requisitos funcionais e não funcionais definidos, faz-se necessária a utilização de diagramas para auxiliar na compreensão do funcionamento do sistema proposto. Esses diagramas têm como objetivo representar, de forma visual e organizada, as interações entre os usuários e o sistema, bem como o fluxo geral das funcionalidades, contribuindo para a delimitação do escopo da solução. Dessa forma, os diagramas apresentados nesta subseção servem como base conceitual para a posterior definição da arquitetura do software, facilitando o entendimento do comportamento esperado da aplicação e de seus principais processos.

4.1.2.1 Diagrama de Casos de Uso

Os casos de uso constituem uma forma de representar as principais interações que os usuários podem realizar em um sistema. Além disso, fornecem uma visão clara das funcionalidades oferecidas pela aplicação, auxiliando na definição e validação dos requisitos do sistema de maneira visual e estruturada. A Figura 4 apresenta o diagrama de casos de uso do sistema proposto.

Figura 4 – Diagrama de Casos de Uso.

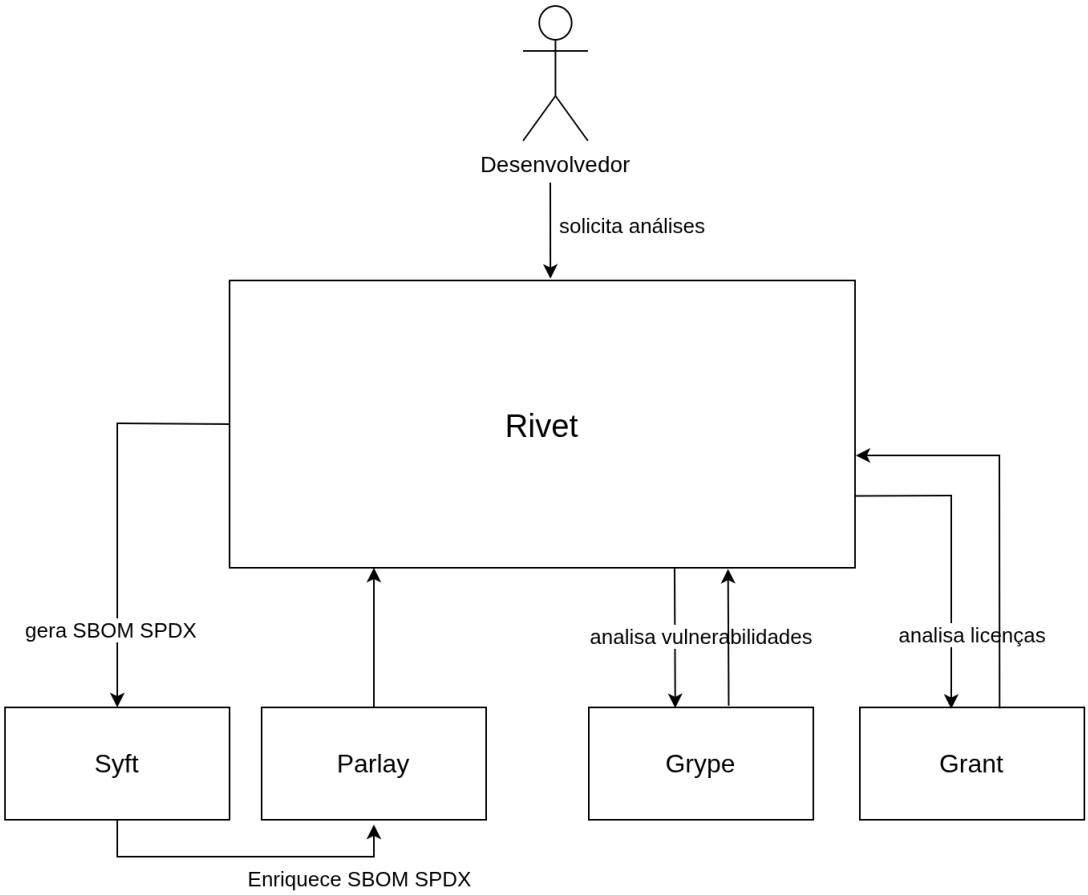


Fonte: Autor (2025).

4.1.2.2 Diagrama de Contexto

O diagrama de contexto apresenta uma visão geral do sistema proposto, evidenciando suas interações com o usuário e com as ferramentas externas utilizadas no processo de análise. No contexto deste trabalho, o sistema integra-se às ferramentas Syft e Parlay para a geração e o enriquecimento da SBOM no padrão SPDX, bem como às ferramentas Grype e Grant para a verificação de vulnerabilidades e licenças, respectivamente. A Figura 5 apresenta o diagrama de contexto do sistema desenvolvido.

Figura 5 – Diagrama de Contexto.



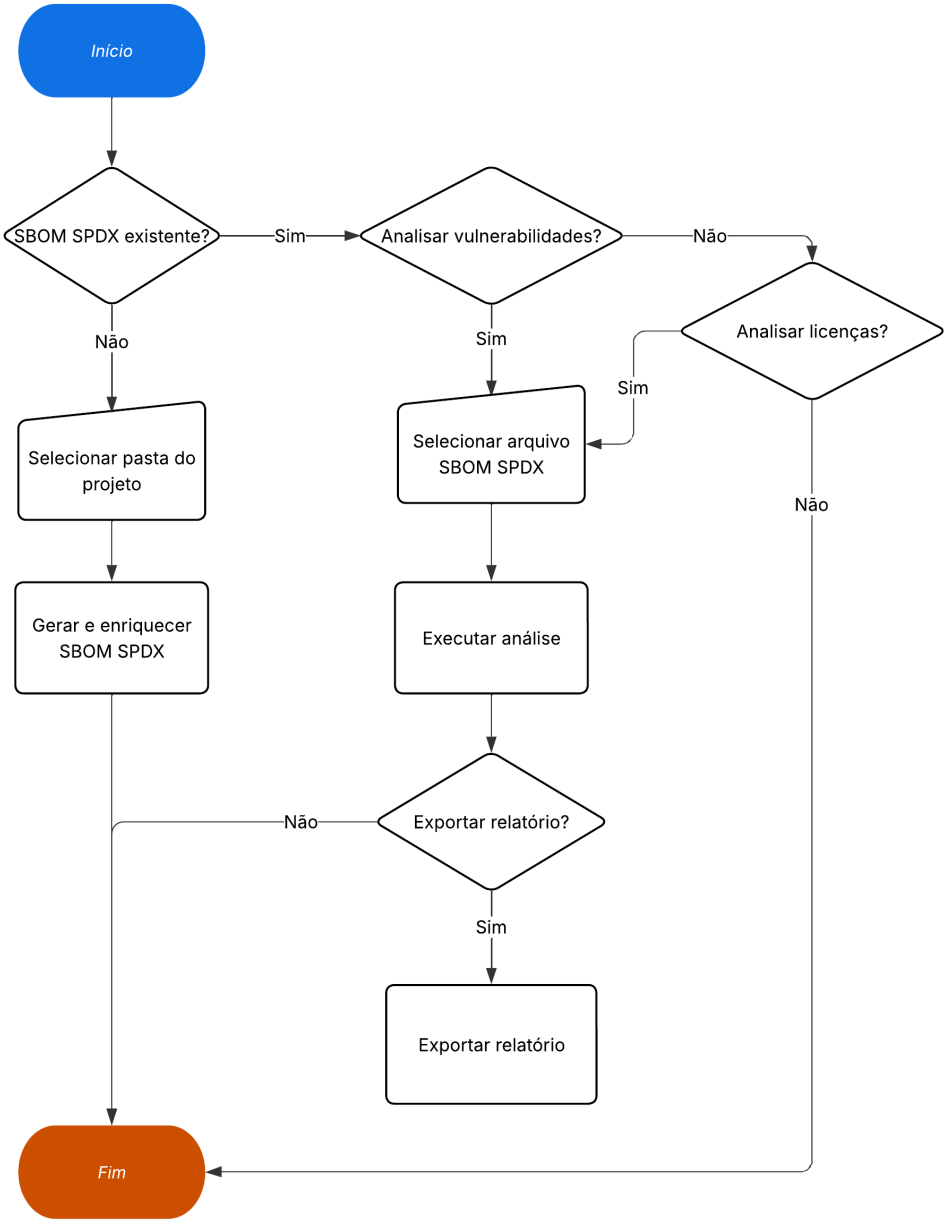
Fonte: Autor (2025).

Conforme ilustrado na Figura 5, o sistema Rivet interage com o desenvolvedor e integra-se a ferramentas externas para realizar suas funcionalidades principais. A geração e o enriquecimento da SBOM no padrão SPDX são realizados com o apoio das ferramentas Syft e Parlay, enquanto a análise de vulnerabilidades e licenças é conduzida por meio das ferramentas Grype e Grant, respectivamente.

4.1.2.3 Diagrama de Fluxo

O diagrama de fluxo é utilizado para representar, de forma sequencial, o funcionamento do sistema e as decisões envolvidas durante sua execução. Por meio dessa representação, é possível visualizar o encadeamento das etapas desde a seleção do projeto ou do arquivo SBOM no padrão SPDX até a realização das análises e a eventual exportação dos resultados. Dessa forma, o diagrama apresentado na Figura 6 a seguir auxilia na compreensão do fluxo principal de utilização do sistema e de suas principais possibilidades de operação.

Figura 6 – Diagrama de Fluxo.



Fonte: Autor (2025).

Conforme ilustrado na Figura 6, o fluxo de funcionamento do sistema inicia-se com a verificação da existência de uma SBOM no padrão SPDX. Caso não exista, o usuário seleciona a pasta do projeto ao qual deseja realizar a geração e o enriquecimento da SBOM. Em seguida, o sistema permite a escolha do tipo de análise a ser executada, podendo contemplar a verificação de vulnerabilidades e/ou de licenças com base na SBOM gerada ou selecionada. Ao final do processo, o usuário pode optar pela exportação dos resultados obtidos, encerrando o fluxo de execução do sistema.

4.2 Definição da Arquitetura

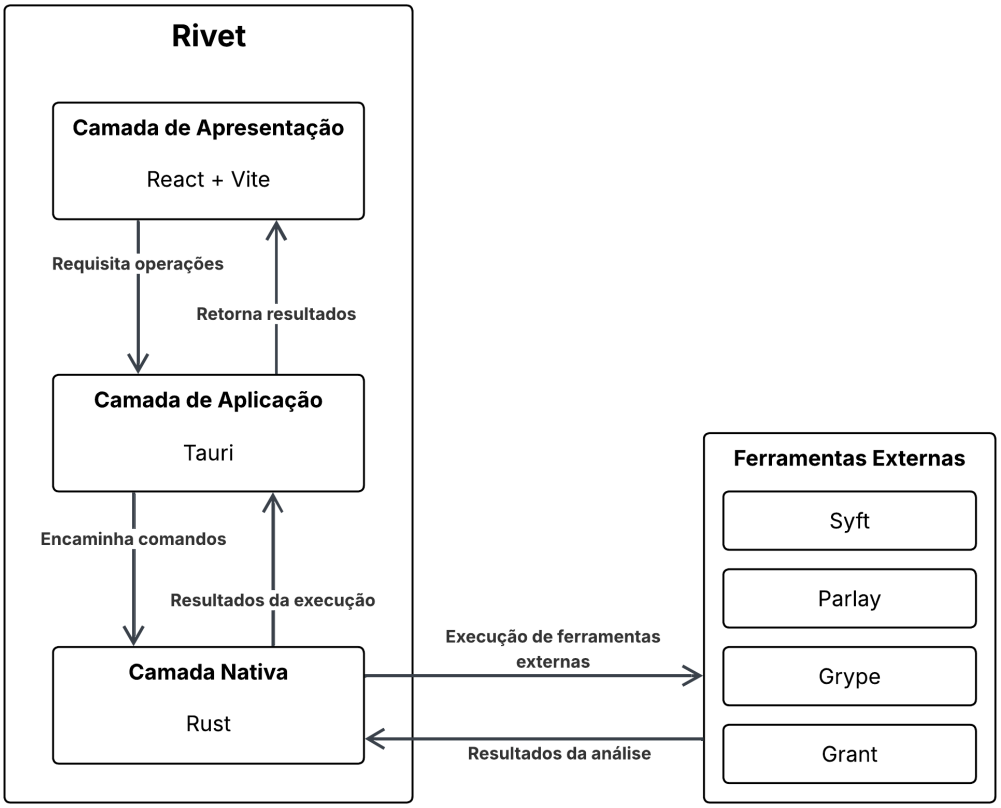
Com base nos requisitos definidos e nos diagramas apresentados anteriormente, esta seção descreve a arquitetura do sistema proposto. São abordadas as principais decisões arquiteturais, bem como a organização dos componentes e suas responsabilidades, com o objetivo de garantir a correta integração das funcionalidades de geração e enriquecimento da SBOM no padrão SPDX, análise de vulnerabilidades e verificação de licenças. A definição da arquitetura busca assegurar uma solução coerente, modular e alinhada aos objetivos estabelecidos neste trabalho.

4.2.1 Modelo Arquitetural

O modelo arquitetural adotado neste trabalho define a organização estrutural do sistema, bem como a divisão de responsabilidades entre seus principais componentes. Esse modelo tem como objetivo facilitar a compreensão da solução proposta, evidenciando a separação entre a interface do usuário, a lógica da aplicação e as ferramentas externas integradas ao sistema. A definição do modelo arquitetural busca garantir modularidade, manutenibilidade e coerência com os requisitos funcionais e não funcionais estabelecidos.

O sistema foi projetado com base em uma arquitetura em camadas, abordagem que organiza a solução em níveis com responsabilidades bem definidas. Essa organização favorece a separação de preocupações e contribui para a manutenção e a evolução do sistema ao longo do tempo. A arquitetura proposta contempla camadas responsáveis pela apresentação da interface ao usuário, pelo controle do fluxo da aplicação e pelo acesso aos recursos do sistema operacional, além da integração com ferramentas externas especializadas utilizadas no processamento das SBOMs no padrão SPDX.

Figura 7 – Diagrama de Arquitetura.



Fonte: Autor (2025).

Conforme ilustrado na Figura 7, a camada de apresentação é responsável pela interface gráfica e pela interação com o usuário, sendo implementada com React e Vite. As requisições realizadas nessa camada são encaminhadas à camada de aplicação, baseada no framework Tauri, que coordena o fluxo de execução do sistema. A camada nativa, implementada em Rust, realiza o acesso aos recursos do sistema operacional e executa as ferramentas externas Syft, Parlay, Grype e Grant, utilizadas para a geração e análise de SBOMs. Os resultados dessas análises são então retornados pelas camadas até a interface, onde são apresentados ao usuário.

4.3 Prototipagem das Telas

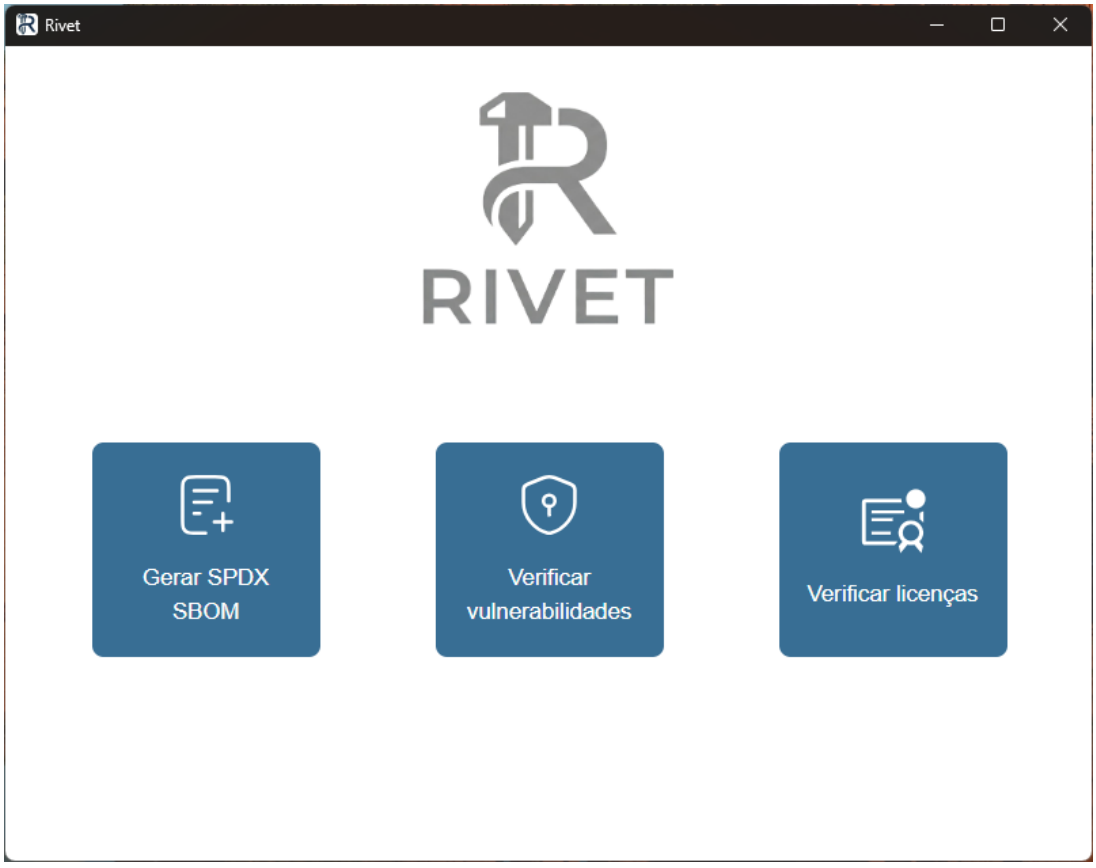
A prototipagem de telas consiste em uma etapa fundamental no processo de desenvolvimento de sistemas interativos, pois permite a visualização prévia da interface e da organização dos elementos que compõem a aplicação. Por meio dos protótipos, é possível representar de forma clara a disposição das funcionalidades, os fluxos de navegação e a interação do usuário com o sistema, auxiliando na validação das decisões de design antes da implementação. Nesta seção, são apresentados os protótipos das principais telas do sistema Rivet, desenvolvidos com o objetivo de apoiar a definição da interface e garantir alinhamento com os requisitos funcionais

estabelecidos.

4.3.1 Tela Inicial

A tela inicial do sistema Rivet tem como objetivo centralizar o acesso às principais funcionalidades da aplicação. Por meio dessa interface, o usuário pode iniciar a geração de uma SBOM no padrão SPDX, realizar a verificação de vulnerabilidades e efetuar a análise de licenças dos componentes de software. A disposição dos elementos foi definida de forma a priorizar a usabilidade, permitindo que as ações principais sejam facilmente identificadas e executadas, funcionando como ponto inicial para a interação com o sistema. A Figura 8 apresenta a interface descrita.

Figura 8 – Tela inicial do app desktop Rivet.



Fonte: Autor (2025).

4.3.2 Tela de Geração de SBOM no Padrão SPDX

A tela de geração de SBOM no padrão SPDX permite ao usuário selecionar o diretório do projeto localizado em seu computador por meio do botão *"Buscar pasta"*. Após a escolha do diretório, o botão *"Criar SBOM SPDX"* inicia o processo de análise e geração da Lista de Materiais de Software. Concluída a execução, o SBOM resultante é exibido na área de

visualização da própria interface. Além disso, o botão *"Exportar SBOM"* possibilita salvar o arquivo gerado para utilização posterior nas funcionalidades de análise de vulnerabilidades e verificação de licenças do Rivet.

Figura 9 – Tela de geração de SBOM no padrão SPDX do app desktop Rivet.

Fonte: Autor (2025).

4.3.3 Tela de Análise de Vulnerabilidades

A tela de análise de vulnerabilidades exibe os resultados obtidos a partir da verificação de um arquivo SBOM no padrão SPDX selecionado por meio do botão *"Buscar arquivo"*. Após a seleção, o botão *"Verificar vulnerabilidades"* inicia o processo de análise, identificando possíveis vulnerabilidades associadas aos componentes listados no SBOM. Além disso, a interface possibilita a exportação dos resultados por meio do botão *"Exportar"*, permitindo o armazenamento das informações para análise ou registro posterior. A Figura 10 abaixo apresenta a interface geral da funcionalidade.

Figura 10 – Tela de análise de vulnerabilidades do app desktop Rivet.

Fonte: Autor (2025).

Para cada vulnerabilidade encontrada, são apresentadas informações detalhadas, como o nome do componente afetado, a versão instalada, o identificador da vulnerabilidade, a versão corrigida (quando disponível), a classificação de severidade e um link para consulta de informações adicionais. Esses dados permitem ao usuário compreender o impacto potencial da vulnerabilidade e apoiar a avaliação dos riscos de segurança do projeto. A Figura 11 abaixo demonstra o detalhamento de uma vulnerabilidade identificada.

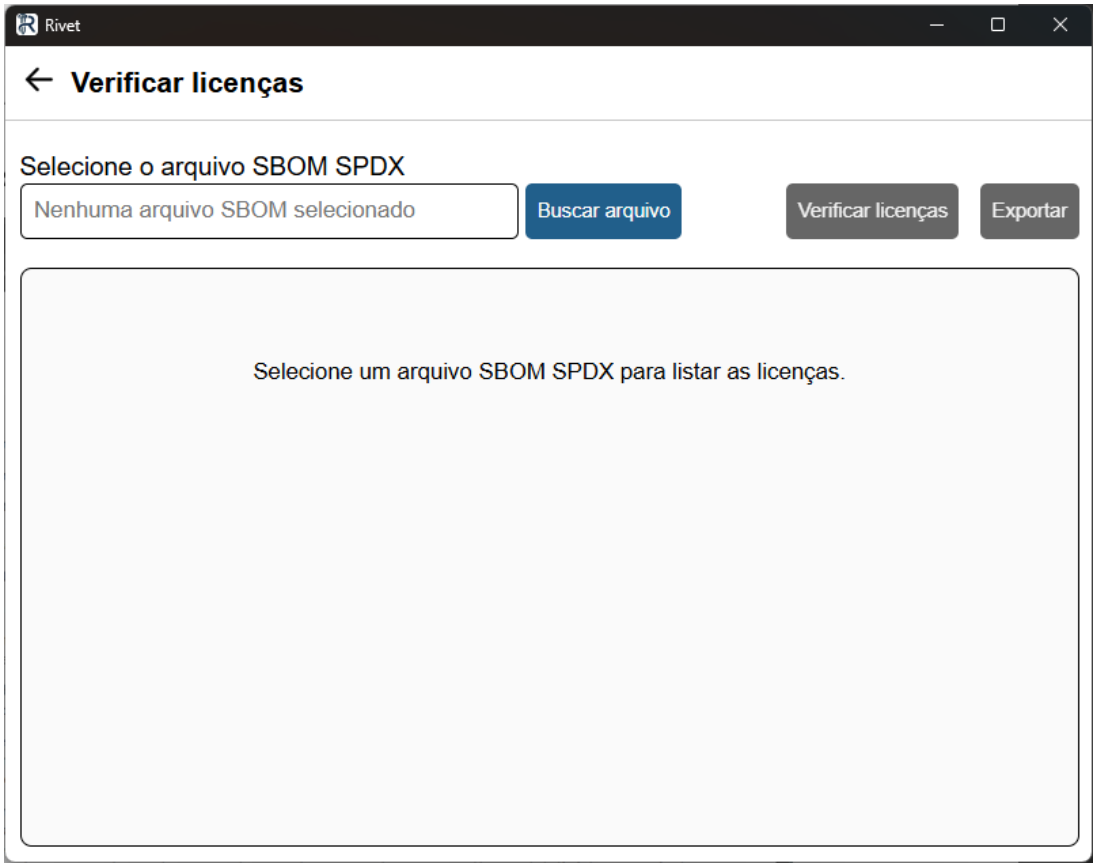
Figura 11 – Demonstração de uma vulnerabilidade encontrada pelo app desktop Rivet.

Fonte: Autor (2025).

4.3.4 Tela de Análise de Licenças

A tela de análise de licenças exibe os resultados obtidos a partir da verificação de um arquivo SBOM no padrão SPDX selecionado por meio do botão *"Buscar arquivo"*. Após a seleção, o botão *"Verificar licenças"* inicia o processo de análise, identificando as licenças associadas aos componentes listados no SBOM. Além disso, a interface possibilita a exportação dos resultados por meio do botão *"Exportar"*, permitindo o armazenamento das informações para consulta ou documentação posterior. A Figura 12 abaixo apresenta a interface geral da funcionalidade.

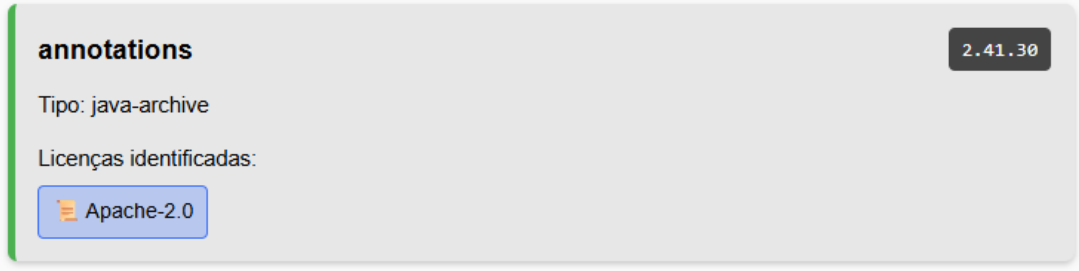
Figura 12 – Tela de análise de licenças do app desktop Rivet.



Fonte: Autor (2025).

Para cada componente identificado, são apresentadas informações como o nome, o tipo, a versão instalada e as licenças associadas. A interface também indica o nível de *copyleft* por meio de um marcador visual localizado no canto esquerdo do componente, cuja coloração varia do vermelho, representando licenças mais restritivas, ao verde, indicando licenças mais permissivas, facilitando a análise de possíveis implicações legais. Ao selecionar uma licença específica, é possível visualizar seu texto completo, quando disponível. A Figura 13 abaixo demonstra o detalhamento de um componente com sua respectiva licença identificada.

Figura 13 – Demonstração de componente e sua licença detectada pelo app desktop Rivet.



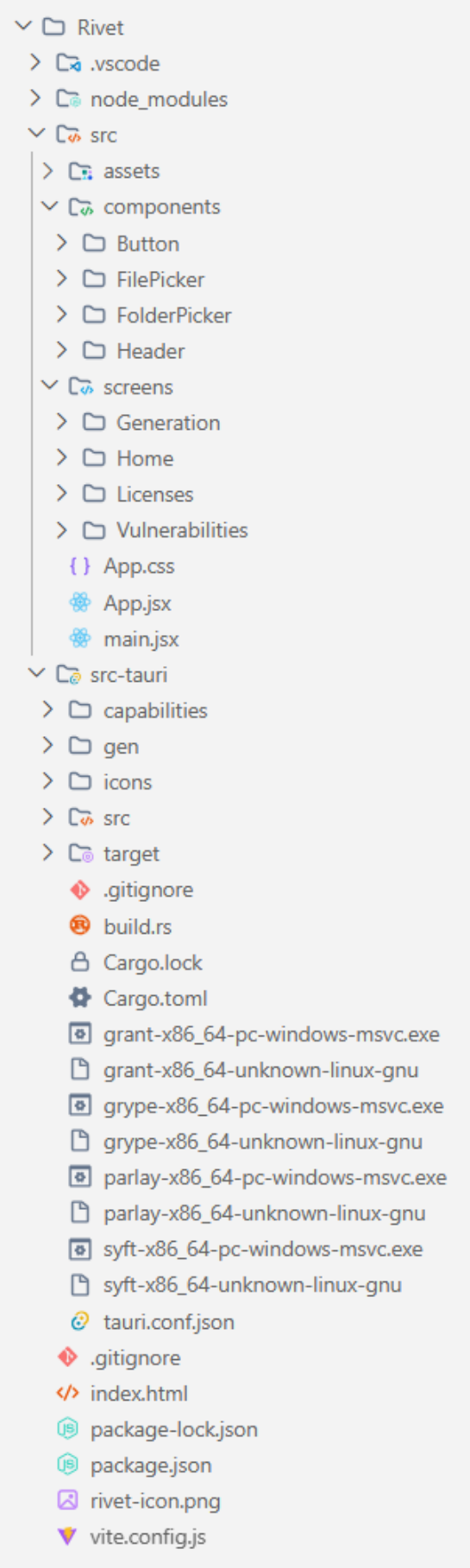
Fonte: Autor (2025).

4.4 Implementação da Aplicação

Esta seção descreve a implementação da aplicação desenvolvida neste trabalho, abordando os principais aspectos técnicos envolvidos na construção do sistema Rivet. São apresentados os detalhes relacionados à organização do código, à integração entre as camadas da aplicação e à utilização das tecnologias e ferramentas adotadas. O objetivo desta seção é evidenciar como as funcionalidades definidas nos requisitos foram efetivamente implementadas, destacando as decisões técnicas que viabilizaram o funcionamento do sistema.

Com o objetivo de contextualizar a implementação do sistema, a Figura 14 apresenta a organização geral do projeto, evidenciando a separação entre a camada de apresentação e a camada responsável pela lógica e integração com o sistema operacional.

Figura 14 – Estrutura de pastas do Projeto.



Fonte: Autor (2025).

Conforme ilustrado na Figura 14, o projeto está organizado de forma modular, separando os arquivos relacionados à interface gráfica, desenvolvida em React, daqueles responsáveis pela camada de aplicação e integração nativa, implementados com o framework Tauri e a linguagem Rust. Essa organização reflete as decisões arquiteturais adotadas e facilita a manutenção e a evolução do sistema.

4.4.1 Arquitetura de Componentes

A camada de apresentação do sistema foi desenvolvida com base em uma arquitetura de componentes, abordagem amplamente adotada no desenvolvimento de interfaces modernas. Nessa arquitetura, a interface é estruturada a partir de componentes reutilizáveis, cada um responsável por uma funcionalidade ou elemento específico da interface, favorecendo a modularidade, a reutilização de código e a manutenção da aplicação.

No projeto desenvolvido, essa abordagem é refletida na organização do código da interface, na qual componentes genéricos, como botões, seletores de arquivos e cabeçalhos, são definidos separadamente das telas da aplicação. As telas, por sua vez, são responsáveis por compor esses componentes e representar as principais funcionalidades do sistema, como a geração de SBOM, a análise de vulnerabilidades e a verificação de licenças.

4.4.2 Camada de Aplicação

A camada de aplicação é responsável por coordenar o fluxo de execução do sistema, atuando como intermediária entre a interface gráfica e a camada nativa. No sistema desenvolvido, essa camada foi implementada com o framework Tauri, sendo encarregada de receber as solicitações originadas na camada de apresentação, organizar o fluxo das operações e encaminhar os comandos necessários para a execução das funcionalidades do sistema.

A comunicação entre a camada de apresentação e a camada de aplicação ocorre por meio de eventos e comandos disparados a partir das interações do usuário na interface gráfica. Essas solicitações são recebidas pela camada de aplicação, que atua como ponto central de controle, validando as ações solicitadas e determinando o fluxo de execução adequado para cada funcionalidade do sistema.

Além de intermediar a comunicação com a interface, a camada de aplicação é responsável por coordenar o fluxo das operações do sistema. Essa camada organiza a sequência de execução das funcionalidades, garantindo que etapas como a geração da SBOM, a análise de vulnerabilidades e a verificação de licenças sejam realizadas de forma consistente, conforme os requisitos definidos.

4.4.3 Integração com Ferramentas Externas

No sistema desenvolvido, a integração com as ferramentas Syft, Parlay, Grype e Grant é realizada pela camada nativa da aplicação. Essa camada é responsável por executar as ferramentas

externas de forma local, utilizando o arquivo SBOM no padrão SPDX como principal artefato de entrada e saída entre as diferentes etapas do processo. Essa abordagem permite manter a lógica de controle desacoplada da execução das ferramentas, preservando a organização e a clareza da arquitetura.

A execução das ferramentas externas ocorre de acordo com o fluxo definido pela camada de aplicação, que encaminha os comandos necessários à camada nativa conforme a funcionalidade solicitada pelo usuário. Após a execução, os resultados produzidos pelas ferramentas são coletados, tratados e retornados à camada de aplicação, que então os disponibiliza à interface gráfica para visualização e/ou exportação.

Essa estratégia de integração contribui para a modularidade do sistema, permitindo que as ferramentas externas sejam utilizadas como componentes independentes, sem comprometer a estrutura interna da aplicação. Além disso, facilita a manutenção e a evolução do sistema, uma vez que alterações ou atualizações nas ferramentas podem ser realizadas com impacto mínimo sobre as demais camadas da aplicação.

5
Resultados

Este capítulo apresenta os resultados obtidos por meio da aplicação desktop Rivet, desenvolvida ao longo deste trabalho. São descritos os testes realizados, os dados gerados a partir da análise de projetos de software e as evidências que demonstram o atendimento aos requisitos definidos. A partir desses resultados, busca-se avaliar a efetividade da solução na geração de SBOMs, na identificação de vulnerabilidades e na verificação de conformidade de licenças, bem como sua contribuição para o gerenciamento da cadeia de suprimentos de software.

5.1 Validação da Aplicação em Estudo de Caso

O SOSMap consiste em uma aplicação web/mobile voltada ao mapeamento de áreas de risco, desenvolvida com o objetivo de auxiliar órgãos de Proteção e Defesa Civil na mitigação dos impactos causados por desastres naturais. O projeto foi desenvolvido no contexto do Programa Institucional de Bolsas de Iniciação em Desenvolvimento Tecnológico e Inovação (PIBITI), sendo caracterizado por uma arquitetura baseada em serviços e pela utilização de bibliotecas e frameworks de terceiros.

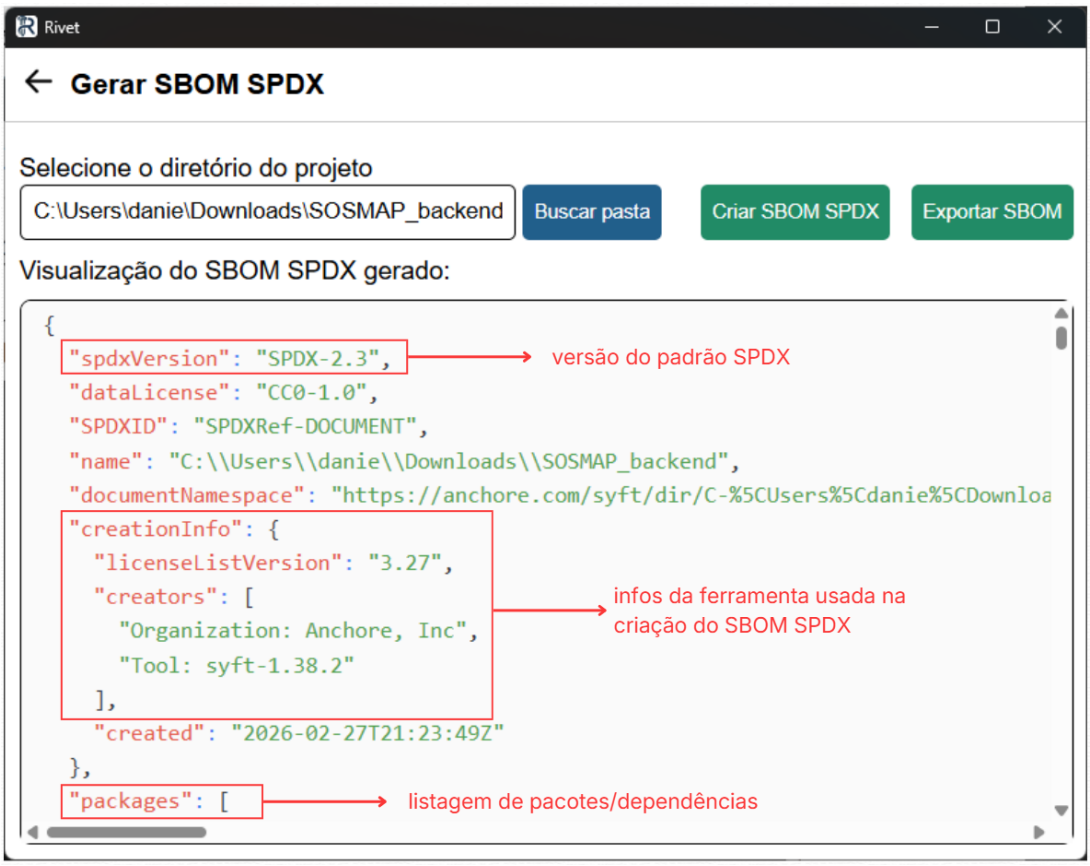
A aplicação incorpora múltiplas dependências externas, incluindo componentes utilizados tanto no front-end quanto no back-end, o que a torna representativa de cenários reais de desenvolvimento contemporâneo. Essa característica a configura como um caso de teste adequado para o Rivet, uma vez que possibilita avaliar, em um contexto prático, a geração de SBOM, a identificação de vulnerabilidades e a verificação de conformidade de licenças associadas às dependências utilizadas.

5.1.1 Geração do SBOM no Padrão SPDX

A geração do SBOM do projeto SOSMap resultou em um documento estruturado no padrão SPDX 2.3, contendo metadados do documento, informações sobre a ferramenta utilizada

na criação e a listagem dos pacotes identificados. A Figura 15 apresenta um trecho do SBOM gerado, destacando campos relevantes como a versão do padrão, as informações de criação e a seção responsável pela listagem das dependências.

Figura 15 – Trecho do SBOM gerado pelo Rivet para o projeto SOSMap.



Fonte: Autor (2025).

Observa-se que o campo *spdxVersion* indica a conformidade com o padrão SPDX 2.3, enquanto a seção *creationInfo* registra informações sobre a ferramenta responsável pela geração do documento. A seção *packages* contém a relação das dependências identificadas, evidenciando a composição do software analisado.

A Figura 16 apresenta o detalhamento de um dos componentes identificados no SBOM. Nesse trecho, são exibidas informações como o nome do pacote, a versão instalada, o fornecedor, a licença declarada e referências externas associadas ao componente.

Figura 16 – Detalhamento de uma dependência identificada no SBOM do projeto SOSMap.


```
{
  "name": "jersey-container-servlet",
  "SPDXID": "SPDXRef-Package-java-archive-jersey-container-servlet-0f93f458104646
  "versionInfo": "2.35",
  "supplier": "Organization: Eclipse EE4J",
  "downloadLocation": "NOASSERTION",
  "filesAnalyzed": false,
  "homepage": "https://projects.eclipse.org/projects/ee4j.jersey",
  "sourceInfo": "acquired package info from installed java archive: \\pom.xml",
  "licenseConcluded": "Eclipse Public License (EPL), Version 2.0",
  "licenseDeclared": "NOASSERTION",
  "copyrightText": "NOASSERTION",
  "description": "Jersey core Servlet 3.x implementation",
  "externalRefs": [
```

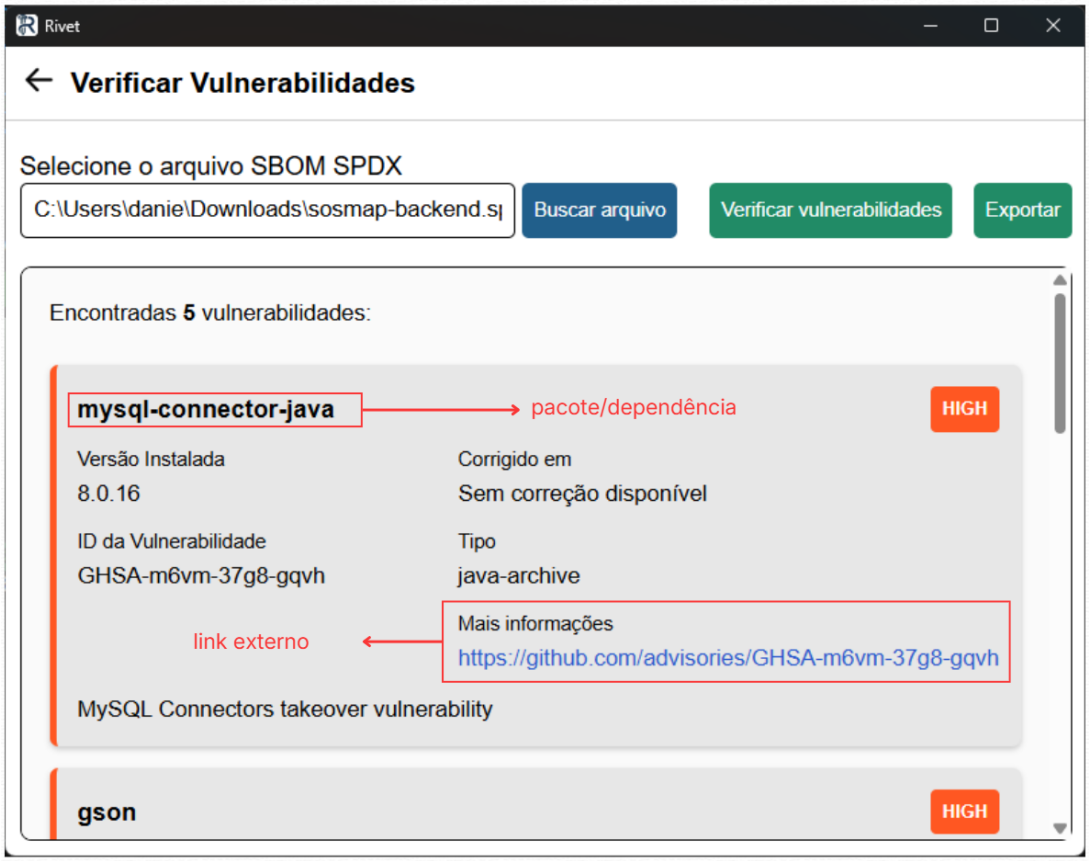

Fonte: Autor (2025).

5.1.2 Análise de Vulnerabilidades

Com base no SBOM previamente gerado, foi realizada a verificação de vulnerabilidades por meio da integração do Rivet com a ferramenta Grype. A análise identificou vulnerabilidades associadas a determinadas dependências do projeto SOSMap, classificadas conforme seu nível de severidade.

A Figura 17 apresenta a tela da aplicação com as vulnerabilidades identificadas, destacando informações como o nome do componente afetado, a versão instalada, o identificador da vulnerabilidade e o link para consulta externa.

Figura 17 – Vulnerabilidades identificadas no projeto SOSMap por meio do Rivet.

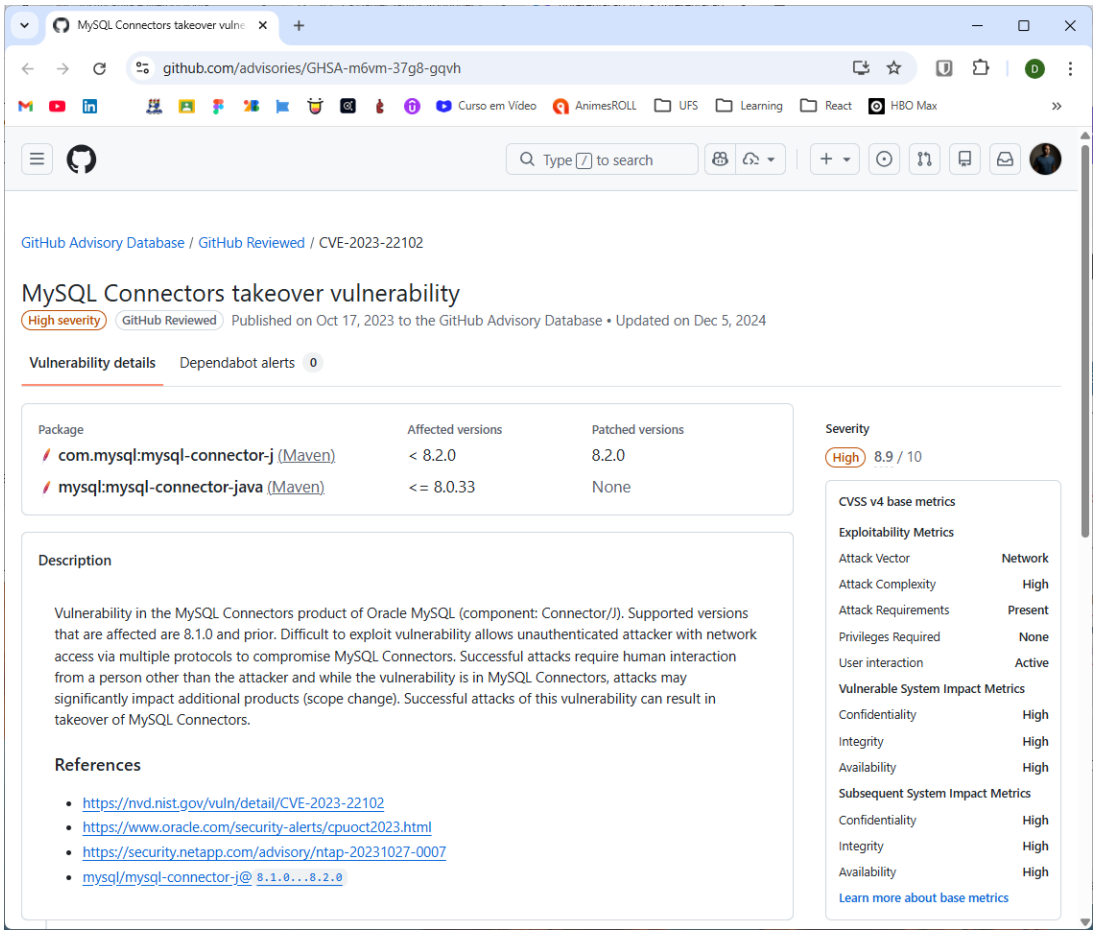


Fonte: Autor (2025).

Ao selecionar o link disponibilizado na interface, o usuário é direcionado para a base de dados oficial da vulnerabilidade, permitindo a consulta de informações detalhadas, como versões afetadas, versões corrigidas, métricas CVSS e descrição técnica do problema.

A Figura 18 demonstra a consulta realizada na base GitHub Advisory Database, evidenciando a severidade da vulnerabilidade identificada.

Figura 18 – Consulta externa à vulnerabilidade identificada, com detalhamento técnico e métricas de severidade.



Fonte: Autor (2025).

Após a atualização das dependências afetadas no SOSMap para versões seguras, foi realizada uma nova verificação de vulnerabilidades por meio do Rivet. Conforme ilustrado na Figura 19, nenhuma vulnerabilidade foi identificada na nova análise, evidenciando a efetividade do processo de correção guiado pelas informações fornecidas pela ferramenta.

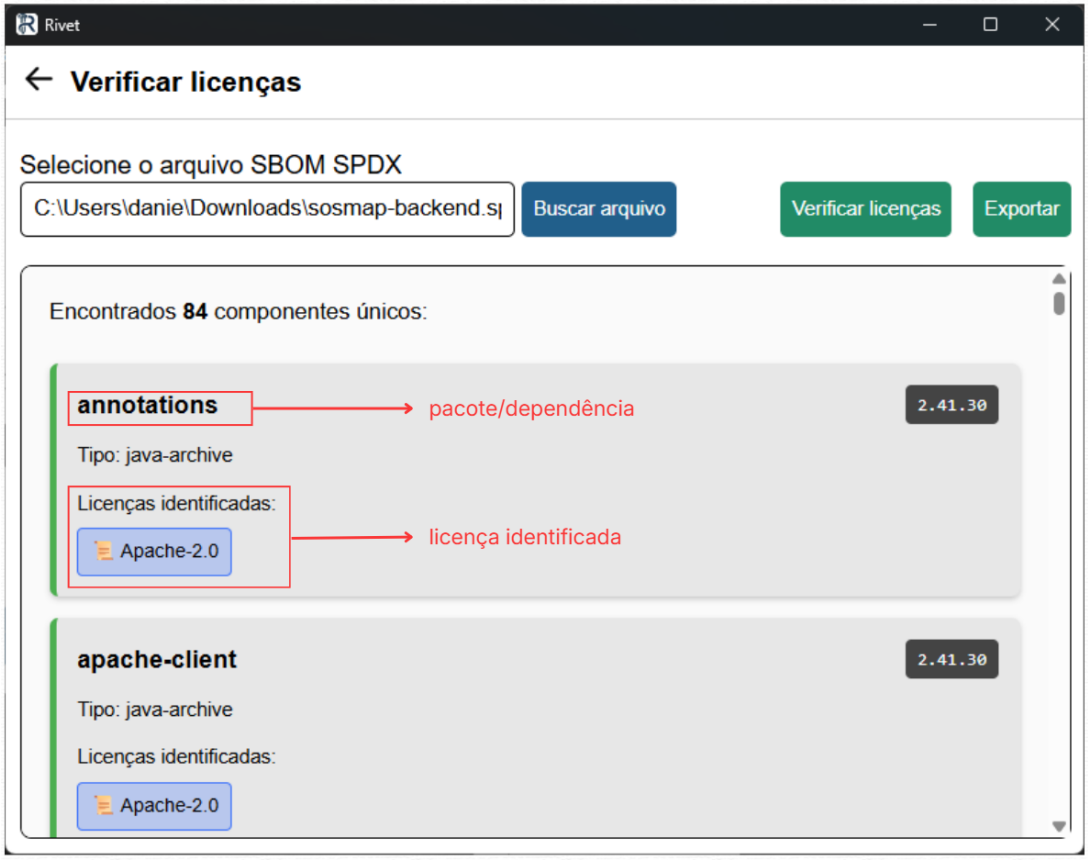
Figura 19 – Nova análise após atualização das dependências, sem vulnerabilidades identificadas.

Fonte: Autor (2025).

5.1.3 Análise de Licenças

Através da utilização do SBOM anteriormente gerado, foi realizada a análise das licenças associadas aos componentes identificados no SBOM do projeto SOSMap. A verificação foi executada por meio da funcionalidade de análise de licenças do Rivet, que lista os componentes encontrados e suas respectivas licenças declaradas. A Figura 20 apresenta a interface da aplicação com as licenças identificadas.

Figura 20 – Licenças identificadas no projeto SOSMap por meio do Rivet.

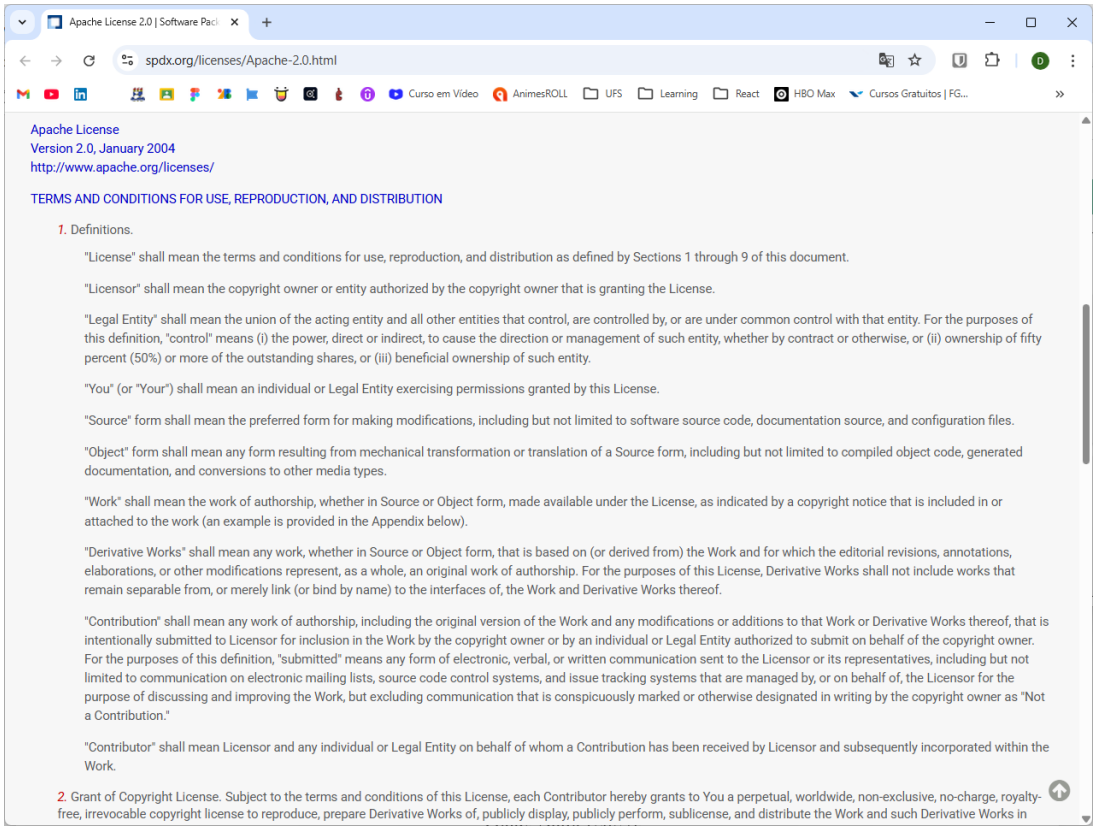


Fonte: Autor (2025).

Para cada componente listado, são exibidas informações como o nome da dependência, o tipo do pacote, a versão instalada e as licenças associadas.A interface também permite a identificação visual do nível de *copyleft*, auxiliando na avaliação de possíveis implicações legais relacionadas ao uso, modificação e redistribuição do software. A indicação é realizada por meio de um marcador colorido, no qual a cor vermelha representa licenças com maior grau de restrição (*copyleft* mais forte), enquanto a cor verde indica licenças mais permissivas. Essa funcionalidade contribui para a verificação de conformidade com requisitos de licenciamento em projetos que utilizam componentes de código aberto.

Ao selecionar uma licença específica, quando disponível, o texto completo da licença é aberto no navegador padrão do sistema, permitindo a consulta integral dos termos e condições. A Figura 21 demonstra a abertura do texto da licença Apache 2.0 no navegador.

Figura 21 – Texto de licença identificada no projeto SOSMap por meio do Rivet.



Fonte: Autor (2025).

De forma geral, a aplicação do Rivet ao projeto SOSMap demonstrou a viabilidade da solução proposta em um cenário real de desenvolvimento. Os resultados obtidos evidenciam que a ferramenta é capaz de gerar SBOMs compatíveis com o padrão SPDX, identificar vulnerabilidades relevantes e analisar a conformidade de licenças, consolidando essas funcionalidades em uma única interface. Essa validação prática reforça a contribuição do sistema para o gerenciamento da cadeia de suprimentos de software.

5.2 Análise dos Resultados Obtidos

A aplicação do Rivet ao projeto SOSMap permitiu avaliar, de forma concreta, a efetividade da solução proposta em um cenário real de desenvolvimento de software. A geração do SBOM no padrão SPDX demonstrou que a ferramenta é capaz de estruturar formalmente as informações relacionadas à composição do sistema, organizando metadados, dependências, versões e referências externas em conformidade com a especificação adotada. Esse resultado evidencia que o Rivet não apenas executa a geração do documento, mas produz um artefato compatível com práticas reconhecidas de transparência na cadeia de suprimentos de software.

No que se refere à identificação de vulnerabilidades, os resultados obtidos confirmam a

capacidade da ferramenta em detectar dependências afetadas e apresentar informações relevantes para análise de risco. A disponibilização de dados como versão instalada, identificador da vulnerabilidade, classificação de severidade e referência externa contribui para uma compreensão mais aprofundada do impacto potencial sobre o sistema analisado. Além disso, a possibilidade de reexecutar a análise após a atualização das dependências e constatar a ausência de vulnerabilidades demonstra a utilidade prática do Rivet como instrumento de apoio à mitigação de riscos e à melhoria contínua da segurança do software.

A análise de licenças, por sua vez, reforça a importância do controle sobre aspectos legais relacionados ao uso de componentes de código aberto. Os resultados mostraram que o Rivet possibilita identificar de forma clara as licenças associadas às dependências do projeto, bem como fornecer subsídios para avaliar o grau de restrição imposto por cada uma delas. Essa funcionalidade é particularmente relevante em projetos que envolvem múltiplas bibliotecas externas, nos quais o desconhecimento das obrigações de licenciamento pode gerar riscos jurídicos e incompatibilidades futuras.

De maneira integrada, os resultados obtidos evidenciam que a consolidação das funcionalidades em uma única aplicação contribui para simplificar o processo de análise da cadeia de suprimentos de software. Ao centralizar a geração de SBOM, a verificação de vulnerabilidades e a análise de licenças em uma interface unificada, o Rivet reduz a necessidade de utilização isolada de diferentes ferramentas em linha de comando, favorecendo maior acessibilidade e eficiência operacional. Assim, a validação realizada com o SOSMap demonstra não apenas o funcionamento técnico da solução, mas também sua aplicabilidade prática e sua contribuição para o fortalecimento de boas práticas de segurança e conformidade no desenvolvimento de software.

6

Considerações Finais

Este trabalho teve como objetivo geral o desenvolvimento de um software desktop denominado Rivet, voltado ao monitoramento da cadeia de suprimentos de software por meio do padrão SPDX. Para isso, foi realizado um estudo aprofundado da estrutura e aplicação prática do padrão na construção de SBOMs, bem como uma análise de ferramentas e bibliotecas que lhe dão suporte. Também foram investigados mecanismos de identificação de vulnerabilidades e de verificação de conformidade de licenças em componentes de terceiros. Com base nesses estudos, a aplicação Rivet foi projetada e implementada, integrando funcionalidades de geração de SBOM, análise de vulnerabilidades e verificação de licenças. A solução foi posteriormente avaliada em cenário prático de uso, permitindo examinar sua aplicabilidade no contexto real de desenvolvimento de software.

O desenvolvimento da solução foi fundamentado por um mapeamento sistemático da literatura, que possibilitou identificar estudos relevantes sobre segurança da cadeia de suprimentos de software, uso do padrão SPDX e práticas de monitoramento de dependências. Esses referenciais teóricos orientaram as decisões de projeto e implementação, assegurando alinhamento entre a base conceitual discutida e a solução construída.

Como principal contribuição, destaca-se a consolidação de ferramentas amplamente adotadas, como Syft, Parlay, Grype e Grant, em uma interface gráfica unificada, reduzindo a complexidade associada ao uso isolado via linha de comando. Essa integração favorece a adoção de práticas relacionadas à segurança da cadeia de suprimentos e à conformidade de licenças em projetos que utilizam componentes de código aberto. A arquitetura definida também contribui para a modularidade, manutenibilidade e extensibilidade do sistema.

Como desdobramento do desenvolvimento realizado, o software Rivet foi submetido a processo de registro, formalizando sua autoria e assegurando a proteção jurídica da solução desenvolvida. Esse procedimento evidencia a materialização concreta dos resultados obtidos, consolidando o trabalho não apenas como investigação acadêmica, mas como produto efetivamente

implementado.

Algumas limitações devem ser consideradas. A aplicação depende das ferramentas externas integradas e da qualidade das bases de dados de vulnerabilidades e licenças disponíveis no momento da análise. Além disso, o escopo atual concentra-se em análises estáticas baseadas em SBOMs e em execução local, não contemplando integrações automatizadas com pipelines de CI/CD ou análises dinâmicas em tempo de execução.

Como perspectivas futuras, sugere-se a ampliação do suporte a outros padrões de SBOM, como o CycloneDX, bem como a integração com pipelines de CI/CD para permitir análises automatizadas ao longo do ciclo de desenvolvimento. Também podem ser exploradas melhorias na visualização dos resultados, como relatórios mais detalhados, mecanismos de alerta e métricas que auxiliem na priorização de correções, ampliando o alcance da ferramenta no contexto do desenvolvimento seguro de software.

Referências

ANCHORE. *Grant – License Compliance Analyzer*. 2026. <https://github.com/anchore/grant>. Acesso em: 23 Outubro 2025. Citado na página 34.

ANCHORE. *Grype – Vulnerability Scanner for Container Images and Filesystems*. 2026. <https://github.com/anchore/grype>. Acesso em: 22 Outubro 2025. Citado 2 vezes nas páginas 34 e 37.

ANCHORE. *Syft – A CLI tool and Go library for generating SBOMs*. 2026. <https://github.com/anchore/syft>. Acesso em: 21 Outubro 2025. Citado 2 vezes nas páginas 33 e 37.

BALLHAUSEN, M. Free and open source software licenses explained. *Computer*, IEEE, v. 52, n. 6, p. 82–86, 2019. Citado 3 vezes nas páginas 21, 24 e 25.

FOSSOLOGY PROJECT. *Fossology – Open Source License Compliance Software*. 2025. <https://www.fossology.org/>. Acesso em: 15 Setembro 2025. Citado na página 37.

GANDHI, R.; GERMONPREZ, M.; LINK, G. J. Open data standards for open source software risk management routines: An examination of spdx. In: *Proceedings of the 2018 ACM International Conference on Supporting Group Work*. [S.l.: s.n.], 2018. p. 219–229. Citado 3 vezes nas páginas 16, 27 e 28.

GNU. *What is Free Software?* 2025. GNU Operating System. Disponível em: <https://www.gnu.org/philosophy/free-sw.html.en>. Acesso em: 22 mar 2025. Citado na página 24.

KAPITSAKI, G. M.; KRAMER, F.; TSELIKAS, N. D. Automating the license compatibility process in open source software with spdx. *Journal of systems and software*, Elsevier, v. 131, p. 386–401, 2017. Citado na página 21.

KAPITSAKI, G. M.; TSELIKAS, N. D.; FOUKARAKIS, I. E. An insight into license tools for open source software systems. *Journal of Systems and Software*, v. 102, p. 72–87, 2015. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121214002945>. Citado na página 21.

KEMPPAINEN, P. *Managing 3rd Party Software Components with Software Bill of Materials*. Dissertação (Mestrado) — Tampere University, May 2023. Citado na página 21.

KOLTUN, P. Free and open source software compliance: An operational perspective. *International Free and Open Source Software Law Review*, v. 3, n. 1, p. 95–101, 2011. Citado na página 21.

LASOTA, L. Reuse software: Making copyright and licensing compliance easier for everyone. In: *Proceedings of the Weizenbaum Conference 2022: Practicing Sovereignty - Interventions for Open Digital Futures*. [S.l.]: Weizenbaum Institute for the Networked Society, 2023. p. 66–71. Citado na página 21.

OMBREDANNE, P. Free and open source software license compliance: Tools for software composition analysis. *Computer*, IEEE, v. 53, n. 10, p. 105–109, 2020. Citado 2 vezes nas páginas 21 e 38.

PASCHALIDES, D.; KAPITSAKI, G. M. Validate your spdx files for open source license violations. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. [S.l.: s.n.], 2016. p. 1047–1051. Citado na página 21.

POGREBNOY, D. et al. Sorrel: an ide plugin for managing licenses and detecting license incompatibilities. *arXiv preprint arXiv:2107.13315*, 2021. Disponível em: <https://arxiv.org/abs/2107.13315>. Citado na página 21.

REACT. *React – A JavaScript library for building user interfaces*. 2026. <https://react.dev/>. Acesso em: 12 Outubro 2025. Citado na página 32.

RUST. *Rust Programming Language (pt-BR)*. 2026. <https://rust-lang.org/pt-BR/>. Acesso em: 13 Outubro 2025. Citado na página 33.

SDPX-GRADLE. *SPDX Gradle Plugin*. 2023. SPDX Community. Disponível em: <https://github.com/spdx/spdx-gradle-plugin/blob/main/README.md>. Acesso em: 08 jul 2025. Citado na página 36.

SDPX-MAVEN. *SPDX Maven Plugin*. 2015. SPDX Community. Disponível em: <https://github.com/spdx/spdx-maven-plugin/blob/master/README.md>. Acesso em: 08 jul 2025. Citado na página 36.

SECURITY, S. *SPDX vs CycloneDX: SBOM Formats Compared*. 2023. <https://scribesecurity.com/blog/spdx-vs-cyclonedx-sbom-formats-compared/>. Acesso em: 15 ago. 2025. Citado na página 12.

SECURITY, S. *What is Software Supply Chain Security? A Deep Dive*. 2024. <https://scribesecurity.com/software-supply-chain-security/>. Acesso em: 15 ago. 2025. Citado 4 vezes nas páginas 11, 12, 22 e 23.

SNYK. *Parlay – Software Composition Analysis and Vulnerability Scanning Tools*. 2026. <https://github.com/snyk/parlay>. Acesso em: 22 Outubro 2025. Citado 2 vezes nas páginas 34 e 37.

SPDX. *About SPDX*. 2025. SPDX. Disponível em: <https://sdpx.dev/about/overview>. Acesso em: 08 mar 2025. Citado 6 vezes nas páginas 26, 27, 28, 30, 31 e 35.

SPDX PROJECT. *SPDX Online Tool*. 2025. <https://tools.spdx.org/app/>. Acesso em: 15 Setembro 2025. Citado na página 35.

TAURI. *Tauri Documentation*. 2026. <https://v2.tauri.app/>. Acesso em: 12 Outubro 2025. Citado na página 32.

VITE. *Vite — Next Generation Frontend Tooling*. 2026. <https://vite.dev/>. Acesso em: 13 Outubro 2025. Citado na página 33.

WINTERSGILL, N. Studying and improving software license compliance in practice. In: *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. [S.l.: s.n.], 2024. p. 225–227. Citado 2 vezes nas páginas 11 e 21.