

License Text Viewer for Simplified Open Source Compliance

BACHELOR THESIS

Julian Schütz

Submitted on 24 March 2025



Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Martin Wagner, M.Sc.
Prof. Dr. Dirk Riehle, M.B.A.



Friedrich-Alexander-Universität
Faculty of Engineering

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

Erlangen, 24 March 2025

License

This work is licensed under the Creative Commons Attribution 4.0 International license (CC BY 4.0), see <https://creativecommons.org/licenses/by/4.0/>

Erlangen, 24 March 2025

Abstract

The software development landscape has changed drastically over the years. With an estimated up to 98% of codebases including open-source software (Nagle et al., 2022) and the rise of package managers and development frameworks, it has never been simpler to start incorporating numerous amounts of open-source projects. So simple, in fact, that managing different dependencies' software licenses went from a cumbersome task to an impossible one, especially when licenses are mentioned in various files outside of dedicated license declarations. This creates a demand for tools that assist in the process of managing license compliance, as carefree misuse of open-source code can quickly turn into a legal liability for companies and private developers alike.

This thesis presents a License Text Viewer feature for one such assisting tool: SCA Tool. The License Text Viewer makes recognizing modified licenses and evaluating the impact of the modifications easy by providing the user with the context of the original file of a mentioned license and displaying the difference to the closest unmodified license text in simple and efficient fashion.

Contents

1	Introduction	1
2	Literature Review	3
2.1	License Fundamentals	3
2.1.1	Permissive Licenses	3
2.1.2	Copyleft Licenses	4
2.1.3	Proprietary Licenses	4
2.2	Additional Tools	4
2.2.1	OSS Review Toolkit	4
2.2.2	ScanCode Toolkit	5
2.2.3	NPM:diff	5
2.2.4	REST and GraphQL API	6
2.3	Related Work	7
2.3.1	ScanCode Workbench	7
2.3.2	OpossumUI	7
3	Requirements	9
3.1	Functional Requirements	9
3.2	Non-functional Requirements	10
4	Architecture	11
4.1	Analyzing and Scanning	11
4.2	Database	12
4.3	Application Programming Interface (API)	12
4.4	Web Interface	13
5	Design and Implementation	15
5.1	Backend	15
5.1.1	License Finding API	15
5.1.2	License Text API	16
5.2	Frontend	16
5.2.1	GitHub API	17

5.2.2 Web Interface	19
6 Evaluation	23
7 Conclusion	25
References	27

List of Figures

5.1	Sample Governance View Table	17
5.2	Steps performed on initial start of License Text Viewer	18
5.3	License Text Viewer page	20

Acronyms

FOSS Free and Open-Source Software

OSS Open-Source Software

ORT OSS Review Toolkit

REST REpresentational State Transfer

API Application Programming Interface

URL Uniform Resource Locator

1 Introduction

In the software development world of today, open-source components have become the foundation of any new application or system. It is estimated that Free and Open-Source Software (FOSS) is included in up to 98% of codebases (Nagle et al., 2022). Applications can easily incorporate hundreds of open-source dependencies due to transitive dependencies. This collaborative nature accelerates development, but also introduces more license compliance and copyright issues than a more isolated approach would. It has never been simpler to include a few packages in a project, which themselves often also have dependencies, all without taking even one glance at the licenses and copyright statements involved in using that code.

Software licenses form the legal groundwork for open-source development to function, but they also specify constraints and obligations like copyright does. There are countless variations of software licenses, ranging from permissive licenses like the MIT license, to copyleft licenses like the GPL license, each with at least slightly differing requirements or obligations.

This situation can lead to multiple problems: Identifying which licenses are a part of a codebase is an arduous task when it just involves visiting each dependency's page and checking the declared license, but becomes an insurmountable task when considering transitive dependencies and the fact that it is incredibly easy to miss a deeply nested license and miscategorize a project. Issues like this lead to conflicts between declared licenses and the actual licenses of code within a project - and others that incorporate incorrectly licensed code. This means the declared license is unfortunately not meaningful enough on its own and any license information embedded in source files or other adjacent documents has to be a part of the consideration process. Another issue is the potential for anyone to slightly modify an existing license text for their specific project, which drastically increases the complexity in reviewing licenses since modifications, of course, do not change the legal binding of the license. When failing to properly manage licenses and their obligations, legal disputes about the damaged intellectual property are just waiting to happen.

SCA Tool is a software composition analysis tool that aims to tackle these problems by providing users with a wide range of features that assist in managing license compliance for a project that implements open-source code. SCA Tool provides users with an overview of dependencies, including transitive dependencies, and lists not only the declared license for each dependency, but also the detected licenses mentioned in other files. In addition, SCA Tool informs users about known vulnerabilities in any used dependency and can produce a software bill-of-material, which lists all components that are included in a project.

This thesis presents an addition to the SCA Tool feature set, the License Text Viewer. It aims to help solve the issue of manual license modifications going unnoticed and simplify the process of evaluating if a dependency should be removed. The License Text Viewer achieves this by displaying detected licenses in the context of their source file and not relying solely on the automatic license detection, displaying the default license text of the detected license as well as highlighting the differences between the two. The navigation between files is kept simple, as the manual evaluation process needs to be kept as intuitive and simple as possible for users. After all, this process otherwise being an arduous task is a part of the reason some projects end up categorized incorrectly in the first place.

2 Literature Review

The first section of this chapter provides an overview of the software licensing landscape and presents the major categories one needs to familiarize themselves with when handling the different licenses used by an open-source software project, as incorrect assessments of licenses can cause conflicts between the targeted license and those contained within.

Afterwards, some of the tools and architectures used by SCA Tool will be detailed. Finally, the current state of technology in regards to license evaluation tools, as well as the differences between SCA Tool and existing tools are presented.

2.1 License Fundamentals

When working with software licensing, understanding the restrictions and obligations of different kinds of licenses is key to avoid misuse of software and legal disputes. License identifiers like `spdxIDs` can be used to easily differentiate licenses. While it is possible to divide software licenses into very thin categories, knowledge of the three types of licenses discussed in this chapter, moving from the least restrictive permissive licenses over the copyleft licenses to the most restrictive proprietary licenses is sufficient for the context of this thesis. It allows an understanding of the conflicts that can occur when multiple licenses are included in the same project and showcase the need for proper classification. It is important to be alert of modifications to license templates, as even small changes like including or removing a *"not"* can fundamentally change the content of a license.

2.1.1 Permissive Licenses

Permissive licenses offer the most flexibility to users, allowing them to redistribute and modify the licensed code with barely any restrictions. Licenses like MIT, Apache or FreeBSD require the licensed material to include the license and copyright notice, but allow users to freely use the software and its code for almost any purpose, including proprietary commercial products. Permissive licenses are by

far the most common licenses found in open-source communities, where building on each other's work is highly encouraged. (Ballhausen, 2019)

2.1.2 Copyleft Licenses

The most crucial condition shared by copyleft licenses is the requirement that any distribution of the licensed material must share the same license as the original material. Copyleft licenses still allow code to be used, modified and distributed, even for commercial products (Ballhausen, 2019). In the latter case, this commercial product would then also need to be published under the same license, e.g. the GPL license.

This eliminates the case of a company using open-source software, modifying it and then selling it while keeping their modifications closed-source, as it would be possible - and frequently happens - with permissive licenses.

2.1.3 Proprietary Licenses

Proprietary licenses have very strict terms and usually come with licensing agreements of varying contents for use. They are typically used by companies wishing to keep their software under control and commercially viable.

Code associated with a proprietary license is usually not publicly available. As such, the specific conditions of the various types of proprietary licenses are not often encountered when working with open-source. The more likely way to come into contact with them is when working on a project which is going to use a proprietary license itself.

2.2 Additional Tools

This section will cover tools that have been integrated into the architecture of SCA Tool or are being integrated into it for the purpose of implementing the License Text Viewer.

2.2.1 OSS Review Toolkit

The OSS Review Toolkit (ORT) is a free open-source toolkit designed to efficiently manage dependencies for open-source software. ORT can assist in automating a variety of tasks surrounding open-source compliance, like generating a software bill of materials, correcting package metadata or conducting policy checks. The toolkit consists of seven tools that can be combined as needed. The Analyzer acts as the foundation, the output of which is used as the input for all other tools. It detects the dependencies of a software project by querying used

package managers and builds a full list of dependencies including transitive dependencies. Several package managers for multiple programming languages are currently supported, including, but not limited to, Gradle and Maven for Java, NPM and Yarn for JavaScript, PIP for Python and Cargo for Rust. ('Introduction | OSS Review Toolkit', n.d.)

The other tools of the ORT include the Downloader, which downloads the source code for all dependencies listed in the Analyzer's output. The Advisor queries a security advisor service for known vulnerabilities within the Analyzer result. The Scanner wraps license scanners like ScanCode and allows storing their results to reduce resources spent scanning files multiple times by reusing scan results ('Scanner | OSS Review Toolkit', n.d.). The Evaluator runs a check of custom policy rules and detects any policy violations. The Reporter is used to visualize results in a readable manner and the Notifier can send notifications of results via channels like e-mail. ('Introduction | OSS Review Toolkit', n.d.)

2.2.2 ScanCode Toolkit

The ScanCode Toolkit is a tool for software composition analysis of open-source software. It can identify copyrights, dependencies, and licenses and has existing integration into ORT. When a scan of a codebase is initiated, ScanCode first classifies the files based on their file types, then extracts any archived or binary files if applicable. Afterwards, a large amount of license detection rules are used to detect license snippets, which are then queried against a license search index to find matches. To detect copyright notices, a parser specialized in the forms of copyright statements is used. Finally, metadata about packages is collected before the scan result is saved in one of the available formats like JSON, XML, CSV or HTML, ready to be used for another tool next ('Overview — ScanCode-Toolkit documentation', n.d.). Each file's scan result contains its path in the codebase, the copyright statements and licenses the scanner detected in addition to the line range the license or copyright was detected in, and reference information about the detected license. ('ScanCode Toolkit Documentation', n.d.)

2.2.3 NPM:diff

Diff is a popular library for JavaScript integrated into the NPM package manager that provides tools to identify changes between two blocks of text. It is especially useful for precisely highlighting small differences, like when reviewing code changes, or for the purposes of SCA Tool, reviewing differences in license texts. The functionality of Diff is based on Eugene W. Myers' "An O(ND) Difference Algorithm and Its Variations"¹ with some small changes. Diff provides functions with differing token sizes like characters in `diffChars`, words in `diffWords`, lines

¹<http://www.xmailserver.org/diff2.pdf>

in `diffLines` or sentences in `diffSentences`. Depending on the function, the texts are split into tokens of corresponding size. Then, using the aforementioned algorithm, the smallest set of insertions and deletions of single tokens to change the first input text into the second one are determined. The functions then return a list of change objects, listing the tokens added, deleted or kept chronologically from the beginning of the first input text to its end. (`'diff'`, 2024)

If the options available for tokens do not fit the user's needs, they can also split the texts into tokens beforehand and then pass arrays of tokens to Diff's `diffArray` function. Diff also provides options for each function like the option to ignore leading and trailing whitespaces and the option to perform the difference calculation case-insensitive. (`'diff'`, 2024)

2.2.4 REST and GraphQL API

REpresentational State Transfer (REST) and GraphQL are two widely used API architectures. Both handle the exchange of data between servers and clients and are supported by GitHub, but they differ greatly in structure, efficiency, and flexibility (`'Comparing GitHub's REST API and GraphQL API'`, n.d.).

In REST-based APIs, every endpoint handles one resource and supports standard HTTP methods such as `GET`, `PUT` or `DELETE`. Parameters can be used to filter results, but the fields each resource returns are fixed. For example, when querying GitHub's REST API about repositories with `GET /search/repositories?q=stars:>100`, it is possible to filter the number of results with a parameter by limiting the repositories to ones with over 100 stars, but the data returned per repository is fixed (Brito & Valente, 2020). This results in constantly retrieving more data from an endpoint than necessary.

GraphQL on the other hand works with just a single endpoint that handles the client's query. The data is instead exposed by a graph. Each node is an object containing fields and edges, which can reference another object. GraphQL also supports the creation of type schemas which define the fields available in objects of said type. The user's GraphQL query, unlike a REST API query, specifies the exact requested information by traversing through the data graph and listing the fields necessary. This solves the REST API's issue of over-fetching at the cost of slightly more complex server structure and client query code. (Brito & Valente, 2020)

Another advantage of GraphQL over REST is that the complexity available in GraphQL queries can achieve amounts of information retrieval in a single query which would only be possible with REST by issuing multiple queries due to the fixed nature of endpoints in REST architecture. This advantage is especially important when working with a high number of requests and rate-limits being a concern. REST architecture has the benefit of being able to use standard built-in

HTTP caching, whereas GraphQL would need a custom solution for its dynamic queries.

2.3 Related Work

In this section, two similar tools to SCA Tool that aim to make handling open-source license compliance easier will be discussed and their differences to SCA Tool will be highlighted.

2.3.1 ScanCode Workbench

ScanCode Workbench² is an open-source application built to provide a visual user interface for scan results created by ScanCode Toolkit. As such, its features are vastly more limited than what SCA Tool provides. After manually loading the JSON file of a scan result, the ScanCode Workbench saves the data from the scan result in a SQLite file and allows users to navigate the directory structure of the scanned codebase and view the detected licenses accompanied by the path of the source file ('ScanCode Workbench Documentation', n.d.). This is not too different from the existing SCA Tool governance view as detailed in section 4.4. However, the addition of the License Text Viewer will provide users with more in-depth information about specific license detections.

The ScanCode Workbench can also present information about the detected copyrights as well as the dependencies contained in the codebase. As the ScanCode Workbench works with only one scan result at a time, checking the licenses used by those dependencies requires scanning them one at a time and then loading the scan results ('ScanCode Workbench Documentation', n.d.). Additionally, due to the nature of the ScanCode Workbench being a visualizer for all the information contained in a scan result, there is a substantial portion of information available that might not interest users who just want to prevent licensing issues as simple as possible, like for example the specific rule used to reach a license match, the runtime of the scan or statistics about the programming languages, licenses, and file types used in the codebase. SCA Tool aims to not overwhelm users with unnecessary masses of information and to keep the focus on providing the important information concisely.

2.3.2 OpossumUI

OpossumUI³ is an open-source tool with a focus on auditing and managing open-source license compliance data from sources like a ScanCode result JSON file.

²<https://github.com/aboutcode-org/scancode-workbench>

³<https://github.com/opossum-tool/OpossumUI>

2. Literature Review

When using OpossumUI, users are presented with statistics about licenses in the codebase and can navigate through the codebase’s directory structure, similar to ScanCode Workbench. OpossumUI’s focus lies in the process of reviewing and editing the license and copyright information, called Attributions in OpossumUI, for specific files. When working from a scan result provided by ScanCode, the relevant files are marked as having a potential Attribution by the scan result but still require the user accepting that Attribution. Users can fill in gaps in the scan result’s information or fill out the Attribution themselves, which SCA Tool does not currently support. The fully manually reviewed Attributions can then be exported in formats like a **SPDX** software bill-of-materials. (‘OpossumUI/USER_GUIDE.md · opossum-tool/OpossumUI’, n.d.)

Similar to ScanCode Workbench, OpossumUI works on a single scan result at a time. The application is focused on completing and reviewing a scan result as opposed to SCA Tool’s approach of encompassing all the basic needs for license compliance.

3 Requirements

This chapter lists the requirements for the License Text Viewer contribution to SCA Tool which are motivated by the issues described in the Introduction and the initial thesis listing. They were extended with additional features deemed adequate during development. They are divided into functional and non-functional requirements. Each requirement is assigned a numerical value which will be used to refer to specific requirements for the rest of this thesis. The fulfillment of the requirements will be evaluated at the end of this thesis.

3.1 Functional Requirements

The functional requirements indicate the behavior relating to the License Text Viewer that SCA Tool has to offer after the contribution.

- F-01:** The system has to be able to provide an overview of licenses used and highlight ones that deviate from the corresponding standard license template.
- F-02:** The system has to be able to display the contents of a License Finding in context of its source file to allow users to personally evaluate the use of that dependency in their project.
- F-03:** The system has to be able to display the closest license template to a detected license.
- F-04:** The system has to be able to display multiple license templates simultaneously.
- F-05:** The system has to be able to accommodate F-02 and F-03.
- F-06:** The system has to be able to highlight the difference in license texts procured by F-02 and F-03.
- F-07:** The system has to be able to toggle the visibility of the difference described in F-06 as displaying the difference is not appropriate when they have no similarities to each other.

- F-08:** The system has to be able to switch the license template used to generate the difference described in F-06 if there are multiple templates.
- F-09:** The system has to be able to display relevant information about License Findings including the number of Findings available as well as the full license expression for each.

3.2 Non-functional Requirements

The non-functional requirements list the quality attributes which the License Text Viewer should provide.

- NF-01:** The display of text should be scaled properly for any screen size to support a variety of device environments.
- NF-02:** The display of multiple licenses templates should be efficient and viewing each different template should not require manual scrolling.
- NF-03:** The progress of fetching license texts should take less than 10 seconds.
- NF-04:** Any API calls should be performed efficiently to improve performance and avoid any rate-limits as much as possible.

4 Architecture

In this chapter, the architecture of SCA Tool that is relevant for the License Text Viewer will be introduced. SCA Tool's backend is built in Java and uses the Spring Boot framework which helps simplify development of Java applications and provides easy integration with the frontend, built in TypeScript for additional type safety over JavaScript. The frontend also uses the React library to create components for the user interface.

This chapter will examine the inner processes that happen in SCA Tool when a user starts an AnalysisTask of a project, reflected in the structure of this chapter. To initiate the AnalysisTask for a project, a user that is logged in creates a new project in SCA Tool's web interface and performs the setup for a new CodeUnit. This entails giving SCA Tool access to the repository in question, e.g. by providing the Uniform Resource Locator (URL) for the git repository.

4.1 Analyzing and Scanning

The first step of the analyzing process is to build a dependency graph for the Code Unit which includes not only the dependencies directly mentioned in the repository, but also all recursive dependencies. For this task, SCA Tool's analyzer unit makes use of ORT, specifically its Analyzer. The ORT Analyzer detects various package managers for different programming languages within a project, e.g. NPM for JavaScript, and then creates a list containing all dependencies managed by the package managers.

The complete list of dependencies is then passed to the scanner unit, which creates individual ScanTasks for each dependency package that needs to be scanned. The scanning of all files is handled by ScanCode, which scans all files in a directory for copyright and licenses, among other information. Every dependency's files are scanned, the results of which are then saved to SCA Tool's database.

4.2 Database

SCA Tool saves the information extracted from a ScanTask to the database in the form of packages, not to be confused with the dependency packages managed by package managers. Each package in the context of SCA Tool's database includes metadata about the scanned dependency, the declared license, as well as the concluded license. Additionally, every instance of a detected License Finding is saved with the path of the file it was found in, the start and endline of said snippet and the score given by ScanCode, ranging from 0 to 100. This score however, is only a mark of how accurate a Finding matches the data from the database. For example, simple lines such as "licensed under MIT license" will receive a score of 100 in the same way the complete license text of the MIT license would receive a score of 100.

SCA Tool uses a mirror of the ScanCode license database⁴ as a baseline of default license texts bundled with their corresponding `spdxID`. This `license_texts` table receives new entries from successful scans, adding any detected license snippets or mentions to the database. The `license_texts` table acts as a collection of the standard licenses that the License Text Viewer can use to compare them to specific Findings.

4.3 API

SCA Tool contains multiple API endpoints that allow the browser-based frontend to communicate with the backend and retrieve all the information that is necessary to run the website. The OpenAPI Generator Gradle plugin is used to generate the basic Controllers as well as the interface the frontend uses to make API queries based on the API specification. However, the code in the backend Controllers that retrieves data according to the API query and prepares it is not generated.

For this thesis, the most important pre-existing API Controller is the Governance Controller. It provides the data needed when users visit the governance tab, including the governance rulesets and the governance view, which handles the content displayed on the governance tab. The Governance View API provides the Web Interface with a number of rows, one for every dependency, as well as the licenses and corresponding categories of each row.

⁴<https://scancode-licensedb.aboutcode.org/>

4.4 Web Interface

For this thesis, the relevant portion of the Web Interface is the governance tab, which is one of the options users have to view the results of the scanning process. There, users choose a default template to decide the intended use case of the project, which will later determine what kind of licenses are marked as problematic, e.g. a copyleft license is marked as prohibited for a proprietary project.

The user is then presented with a table view where each row is one dependency found in the project. The columns list the name of that dependency, if it is allowed according to the current template's ruleset, and the License Integrity, which marks confidence in the result between **Low**, **Medium** and **High**. The last column provides an overview of the specific license names that were detected in the dependency combined into one license expression. This might be as simple as a single **MIT** but will list all licenses with **AND** if there are multiple. The licenses that are allowed in the current ruleset are marked green and prohibited ones red. If a single file is licensed under multiple licenses, they will be listed as **(first_license OR second_license)**.

This display allows users to notice dependencies that might cause issues. However, it does not allow the user to dig deeper into those issues to evaluate the exact circumstances and determine if a dependency can still be used or not. This thesis aims to solve this issue.

5 Design and Implementation

This chapter details the changes and additions made to SCA Tool’s architecture for the purpose of the License Text Viewer. The changes to the backend will be covered in section 5.1 and the changes to the frontend in section 5.2.

5.1 Backend

The License Text Viewer requires data in the frontend which is not already available with existing API endpoints. To be able to access the necessary data, a new API endpoint was created and one existing endpoint was modified. Both of them included changes in the API specification used by SCA Tool.

5.1.1 License Finding API

The License Finding API is used by the frontend to provide users with easily digestible details about specific findings. Each License Finding contains the package name, the URL, path, revision, starting line, and ending line of the license snippet in the original file, as well as the license snippet’s text, the detected `spdxID`, the score, and the `isLicense` flag.

The License Finding API is integrated into the existing Governance View API described in section 4.3, which is queried when the governance tabs are loaded. The query that retrieves the License Findings thus provides them for all package IDs of the current CodeUnit. Having access to all findings’ data on the governance tab page allows the score attribute to be used prior to starting the License Text Viewer itself, which is further explained in section 5.2.

The information about the original file is used by the GitHub API (section 5.2.1) to display the license snippet within the context of its entire file. The `spdxID` is used in the License Text API as described in section 5.1.2. The detailed uses for each attribute are explained in section 5.2.

5.1.2 License Text API

The License Text API is the newly created endpoint in SCA Tool's Governance Controller. It is responsible for fetching the default license texts for a single package from the `license_texts` table. The response texts are then used to compare the default license texts to the detected license snippet's texts, the method of which is further detailed in section 5.2.2.

As an API query parameter, the License Text API receives a list of license expressions, one from each License Finding of the package in question. It is important to note the potential special cases that can occur within the license expressions. If a single file is licensed under two different licenses, the corresponding license expression is not a simple `spdxID`, but will instead be in the form of `spdxID1 OR spdxID2` or `spdxID1 AND spdxID2`. For this reason, the input list of license expression needs to be prepared before the database can be queried. To achieve this, the license expressions containing `AND` or `OR` are temporarily split up, as the full expression is needed for the response.

The database is then queried with a list of `spdxIDs` and returns a tuple of a `spdxID` and the corresponding default license text for each unique input from the `license_texts` table.

The data is then prepared for the response form. Every unique license expression is paired with the list of license pairs as received in the previous step. For basic license expressions that only contain a simple `spdxID`, this list will just contain one entry. A special case like `MIT OR BSD-3-Clause` will then have two list entries in the form of `[(MIT, licensetext_of_MIT), (BSD-3-Clause, licensetext_of_BSD-3-Clause)]`. This form allows the frontend to access each license text of a license expression separately while retaining the information of which license expression contains which licenses.

5.2 Frontend

The License Text Viewer page can be navigated to via an additional column, titled **License Text Viewer**, which is added to the end of the governance view. Each row of the new column contains a button that allows users to inspect the licenses of that row's package if the scanning process for that package has concluded already. If the scan has not finished, the button is colored in red and will not navigate to the License Text Viewer when clicked. For packages that contain a Finding with a particularly low score, the threshold being 60 out of 100, the button that navigates to the License Text Viewer is colored in orange to signify a point of interest.

Figure 5.1 provides an sample view of the governance tab. In this example,


<div>  No problems found No rule violations have been detected for your components. </div>				
Component	Classification	License Int... ↑	License	License Text Viewer
npm/mime-db@1.52.0	Allowed	Low	MIT	Inspect Package
npm/async@0.4.0	Allowed	Medium	MIT	Inspect Package
npm/axios@1.3.5	Allowed	Medium	MIT	Inspect Package
npm/combined-stream@1.0.8	Allowed	Medium	MIT	Inspect Package
npm/delayed-stream@1.0.0	Allowed	Medium	MIT	Inspect Package
npm/follow-redirects@1.15.2	Allowed	Medium	MIT	Inspect Package
npm/form-data@4.0.0	Allowed	Medium	MIT	Inspect Package
npm/mime-types@2.1.35	Allowed	Medium	MIT	Inspect Package
npm/proxy-from-env@1.1.0	Allowed	Medium	MIT	Inspect Package
Rows per page: 10 ▾ 1-9 of 9 < >				

Figure 5.1: Sample Governance View Table

the package `npm/mime-db@1.52.0` has not finished the scanning process and the package `npm/axios@1.3.5` contains a License Finding with a score below 60.

When navigating to the License Text Viewer’s page, the inspected package’s full license expression of the governance view table, as described in section 4.4, and the portion of License Findings for the inspected package are passed to the License Text Viewer. When first loading the page, the License Text API is queried using a list of `spdxID` attributes from the License Findings. Additionally, the GitHub API is queried using the Finding’s URL, path and revision. Currently, only GitHub links are supported in the scope of this thesis, but the code can be extended in the future to support alternatives such as GitLab as well. The communication and steps performed by the frontend when first loading the License Text Viewer are illustrated in Figure 5.2.

5.2.1 GitHub API

The frontend’s GitHub API aims to provide users with the context of the original file for the license snippets of a package. Requesting a file’s contents at a specific revision is simple using either GitHub’s REST API or GraphQL API. For ease of use in a TypeScript environment, the License Text Viewer uses `npm:octokit/core`⁵. Octokit is a JavaScript client library developed by Git-

⁵<https://www.npmjs.com/package/@octokit/core>

5. Design and Implementation

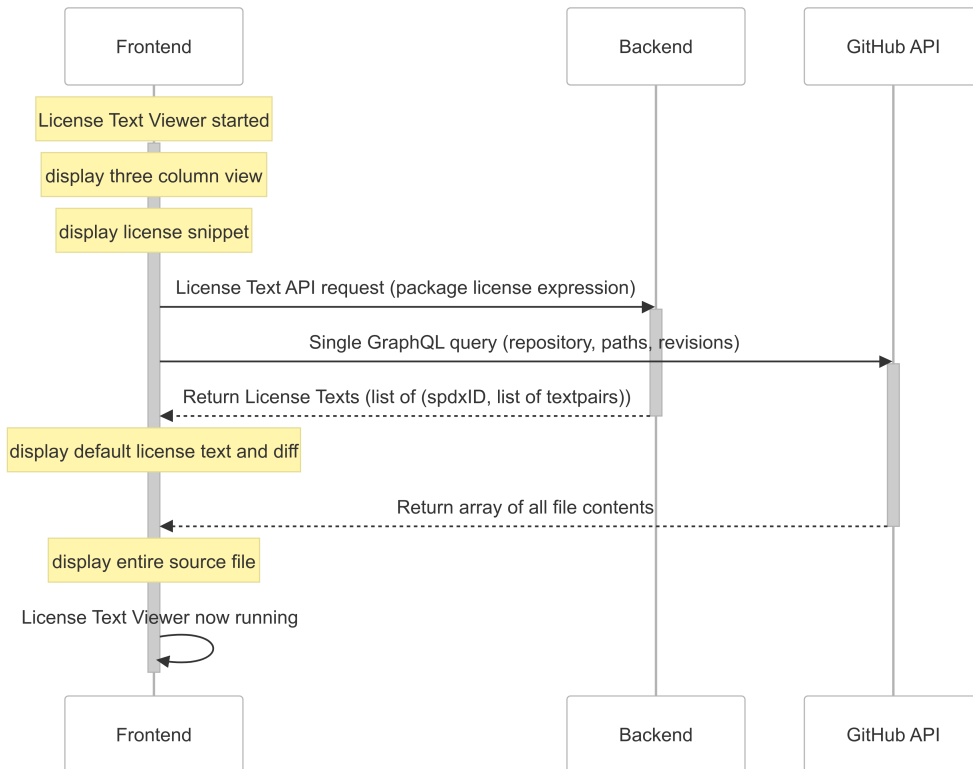


Figure 5.2: Steps performed on initial start of License Text Viewer

Hub. It provides minimalistic methods for common interactions with GitHub paired with thorough documentation, either via the REST API or the GraphQL API and handles the authentication process as well.

While requesting a single file is easy using the REST API, requesting potentially hundreds is not possible in a single request. This combined with the rate-limit of 5000 requests per hour for authenticated users led to the decision to use the GraphQL API instead of the REST API (‘Rate limits and node limits for the GraphQL API’, n.d.). A single GraphQL query can be used to request the files for all paths in the current package’s Findings at the corresponding revision.

The data necessary to form the GraphQL query is gathered by splitting the License Finding’s GitHub URL attribute. The owner of the repository as well as the name of the repository are extracted from the split attribute. Additionally, the query requires the path of the requested file, which is also available as an attribute. The complete query contains as many file requests for the repository in question as there are License Findings available, each specifying the revision and the path of the file. A successful response contains an array, with each entry corresponding to one of the file requests and containing its text.

5.2.2 Web Interface

The License Text Viewer page presents general information at the top. Below this header, three columns of equal width are displayed with their own titles. Each column's content can be scrolled individually. The default license text, as retrieved by the License Text API, resides in the left column and the detected license snippet from the project in the right. The center showcases the difference between the two with a `diff` generated using `npm:diff`.

The header contains metadata about the currently inspected package. It starts with the package ID at the top and the license expression of the entire package below it. The license expression concatenates different licenses with `AND` in the same way as the governance view column. Then follows information concerning the detected License Findings, which are sorted by their license `spdxID` first and their score second. This way, users are presented with the Findings that are most likely to cause issues for each license first. The index of the current Finding and the number of Findings available are listed and similarly, information about the group of Findings with the current license is displayed, including the amount of Findings with the current license and the index of the current Finding within that group.

On either side of the header, two navigation buttons are located. The buttons above the right column traverse the list of Findings forwards or backwards. The ones above the left column instead jump between the license groups, switching to the next or previous Finding that has a different license. In the case of a package only containing one license, the left-side buttons will navigate to the first Finding.

The left columns title is the license expression of the current Finding and its corresponding default license text. If multiple licenses were detected in the current Finding by ScanCode, they are displayed below one another with headlines containing their individual `spdxID` in between. As some license texts can be quite long and scrolling through them to read the next one is impractical, the `spdxIDs` that form the current Finding's license expression can be clicked to automatically scroll to that headline.

The right column contains the path for the current Finding's source file within the original project as its title. The entire source file's contents are displayed below. The section detected as a license snippet, defined by the Finding's `startLine` and `endLine` attributes, is highlighted. As source files can be very long as well, the box containing the text will automatically scroll to the highlighted section when it is first loaded. In the case of the GitHub API query failing, or if a link to a GitHub repository is not available, the highlighted license snippet is instead displayed by itself and a status message next to the path informs the user that the GitHub API query was unsuccessful.

5. Design and Implementation

The middle column lists the title `Diff` and the current visibility status of its contents, either `OFF` or `ON` next to the title. If the visibility is set to `OFF`, the contents of the left and right column each take up half the horizontal screen space instead of a third. The middle columns content is the difference between the default license text of the left column and the detected license snippet of the right column. To achieve this, `npm:diff` is used. For this use case, the `diffWords` function yields better results than `diffChars` and `diffLines`. When using `diffLines`, it can be difficult to spot small difference in long sentences with a quick glance. `diffChars` instead runs into the problem of fragmenting words, making readability more difficult than necessary. Any additions made to the default license by the license snippet are marked in green, any deletion is marked in red. The portion of the text that stays identical between the two is kept black. In cases of a Finding containing multiple licenses, the input used for `diffWords` is set to the first of those licenses by default. When a license in the left columns title is clicked and scrolled to, the `diffWords` input also changes to that license's text accordingly.

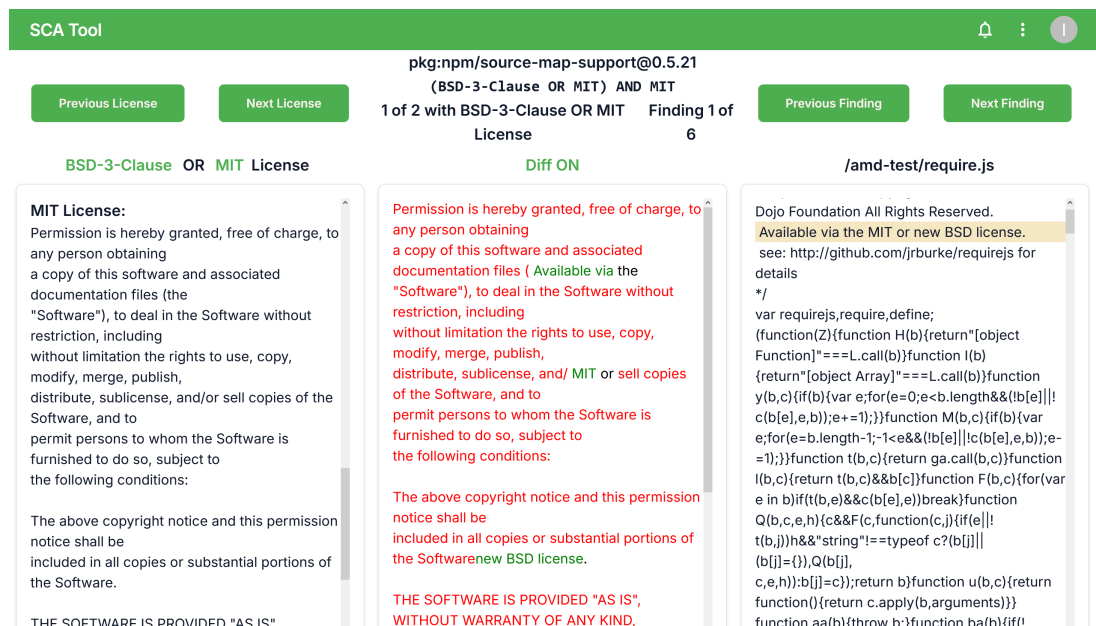


Figure 5.3: License Text Viewer page

Figure 5.3 provides an example of the License Text Viewer page when viewing the first Finding of `npm/source-map-support@0.5.21`, with the Diff column enabled and the license set to MIT.

However, displaying the `diff` between the license snippet and the license default text is not always useful since not all detected license snippets are full license texts, some simply mention a license, like in the line `"license: MIT"`. For this reason, the visibility of the column's content can be toggled on or off by clicking

the **Diff ON/OFF** title. The default setting corresponds to the **isLicense** flag set for the current Finding by ScanCode, which marks if a text snippet is an actual license text or not. It very accurately determines lines like "**license:** MIT" as not being a license text, sets **isLicense** to false and leads to the **Diff** columns contents not being visible by default. There are cases of false negatives, where a license snippet does contain a similar enough text to the default license that displaying the difference does make sense but **isLicense** is false. In cases like this, users can simply click the **Diff OFF** title to switch the display of that **diff** on.

6 Evaluation

This chapter will evaluate the success of each target requirement as defined in chapter 3.

- **F-01:** Was fulfilled by implementing the License Text Viewer.
- **F-02:** Was partially fulfilled. The License Text Viewer can only access the original file of a license snippet if the dependency is on GitHub via the GitHub API.
- **F-03:** Was fulfilled. When the ScanCode scan result detects a license, the `license_texts` table, which uses ScanCode's license database as a baseline, will contain a default license text. The License Text API fetches the text and the frontend displays it and the license snippet even in the case of the GitHub API not being available.
- **F-04:** Was fulfilled. When files are licensed under multiple licenses, the License Text API retrieves all mentioned license texts and the frontend displays them below one another.
- **F-05:** Was fulfilled. The left and right column in the License Text Viewer page contain the necessary texts.
- **F-06:** Was fulfilled. The License Text Viewer uses `npm:diff` to generate the difference between the highlighted detected license snippet and the default license text.
- **F-07:** Was fulfilled. Clicking the diff status header will toggle the visibility of the diff column on or off. By default, the column is visible if the `isLicense` flag is set to `true` for the current License Finding.
- **F-08:** Was fulfilled. Clicking the name of the license in the header of the left column will change the diff input to the text of the selected license and will scroll to the beginning of its license text.
- **F-09:** Was fulfilled. The header section displays the name and full license expression of the current package, the number of Findings with the current

license and the total number in addition of the license expression and path for the current Finding.

- **NF-01:** Was fulfilled. Both smaller and larger screen spaces still contain all information and all functionalities.
- **NF-02:** Was fulfilled. Clicking the license name in the header of the left column will automatically scroll to the beginning of its text.
- **NF-03:** Was fulfilled. For a package with 141 Findings, the GitHub API takes 1.5 seconds and the License Text API which returned 24 license texts takes 160 milliseconds.
- **NF-04:** Was partially fulfilled. The License Text Viewer calls the License Text API as well as the GitHub API once on initial startup but has no caching strategies in place to reduce the amount of necessary API calls when restarting the License Text Viewer.

7 Conclusion

The License Text Viewer presented in this thesis addresses the problem of modified licenses by providing users of SCA Tool with an efficient way to manually evaluate changes in license texts. By displaying detected licenses within the context of their original file and generating a colored visualized diff between the detected license text and the closest default license text, the License Text Viewer drastically simplifies the process of evaluating such modifications. This implementation successfully met most requirements, and the remaining ones at least partially. There are options open for improving the License Text Viewer which were not in the scope of this thesis:

- As of now, the License Text Viewer can only show the context of the original file for packages available on GitHub. The code can be extended to support other file sources as well by adjusting the function detailed in section 5.2.1.
- Performance can be improved slightly and the risk of reaching rate-limits can be reduced by implementing a custom caching solution which stores the result of a GitHub request and prevents the License Text Viewer from sending the same request again when closing and re-opening the License Text Viewer with the same package.
- The accuracy of scan results is not perfect, which is why the License Text Viewer displays the entire source file. Improving the scan results is not a simple task as SCA Tool currently uses ScanCode Toolkit for scans. If it is possible to improve the scan results and the `isLicense` flag in them, users would need to manually toggle the visibility of the diff column less frequently and could place higher trust in the scan results, even when the original source file is not available.

Looking forward, the future of license management appears more promising as tools like SCA Tool and others are developed rapidly to reign in the massive amounts of neglected license management. The improvement of automated license management tools will assist in catching up to the incredible size of modern-day open-source software development.

7. Conclusion

References

- Ballhausen, M. (2019). Free and open source software licenses explained. *Computer*, 52(6), 82–86. <https://doi.org/10.1109/MC.2019.2907766>
- Brito, G., & Valente, M. T. (2020). Rest vs graphql: A controlled experiment. *2020 IEEE International Conference on Software Architecture (ICSA)*, 81–91. <https://doi.org/10.1109/ICSA47634.2020.00016>
- Comparing GitHub’s REST API and GraphQL API* [GitHub docs]. (n.d.). Retrieved March 24, 2025, from <https://docs.github.com/en/rest/about-the-rest-api/comparing-githubs-rest-api-and-graphql-api?apiVersion=2022-11-28>
- Diff* [Npm]. (2024, September 6). Retrieved March 24, 2025, from <https://www.npmjs.com/package/diff>
- Introduction / OSS review toolkit*. (n.d.). Retrieved March 24, 2025, from <https://oss-review-toolkit.github.io/ort/docs/intro>
- Nagle, F., Dana, J., Hoffman, J., Randazzo, S., & Zhou, Y. (2022, March). *Census II of free and open source software — application libraries*. The Linux Foundation. <https://doi.org/10.70828/KHEH5209>
- OpossumUI/USER_guide.md · opossum-tool/OpossumUI* [GitHub]. (n.d.). Retrieved March 24, 2025, from https://github.com/opossum-tool/OpossumUI/blob/main/USER_GUIDE.md
- Overview — ScanCode-toolkit documentation*. (n.d.). Retrieved March 24, 2025, from <https://scancode-toolkit.readthedocs.io/en/latest/reference/overview.html#explain-how-scancode-works>
- Rate limits and node limits for the GraphQL API* [GitHub docs]. (n.d.). Retrieved March 24, 2025, from <https://docs.github.com/en/graphql/overview/rate-limits-and-node-limits-for-the-graphql-api>
- ScanCode toolkit documentation*. (n.d.). Retrieved March 24, 2025, from <https://scancode-toolkit.readthedocs.io/en/latest/>
- ScanCode workbench documentation*. (n.d.). Retrieved March 24, 2025, from <https://scancode-workbench.readthedocs.io/en/develop/>
- Scanner / OSS review toolkit*. (n.d.). <https://oss-review-toolkit.github.io/ort/docs/tools/scanner>