university of groningen / faculty of science and engineering

Exploring the effectiveness of search engines for finding architectural knowledge in open source repositories

Author: **Tom den Boon (s3176096)**

Supervisor: **dr. M.A.M. Soliman**

Abstract

Software engineers need architectural knowledge to make suitable design decisions. Finding this knowledge can be quite hard. Researchers are trying to improve the search for architectural knowledge, and so will this paper. We have created a handy search tool for the pursuit in finding architectural knowledge. With this tool, we will perform an empirical study on apache Jira and Mailing lists. In which we perform queries to find sources with architectural knowledge in them. We will then perform a qualitative analysis on how well the search engine performed. And what kind of architectural design decisions were found in the repositories.

August 20, 2021

Contents

1 Introduction

1.1 Software Architecture

Nowadays, society is built on complex software and technologies. An important part in maintaining and evolving this software is the requirement of a well built architectural foundation. Constructing architecture for software is not an easy task. Software architects have a lot of choices they can take to build the architecture. These choices are fluctuating and depend on the system they are designing and what requirements they need to satisfy. This is a very complicated and important process. If done incorrectly the system can become unmanageable, too slow, or any other unwelcome property. Especially the starting stages of developing the architecture are critical. This will be the foundation on which the rest of the architecture depends. Creating mistakes in these early stages is even more essential to prevent. Software architecture is said to help developers with these problems [11]. But this would require extensive knowledge from the developer about software architecture. That is why researchers have been trying to create a set of rules for architectural decisions to help architects create the best possible software.

1.2 Problem with Architectural Knowledge

Architectural knowledge should be of importance to any developer out there. Developers rely for the most part on their experience to design software architecture. With architectural knowledge, we can give developers extra support to make the correct decisions. Architectural knowledge is defined as documentation of the design decisions, rationale, assumptions, context, and other factors that together determine architecture solutions during the architecture design stage.

Scientists have tried to categorize this knowledge and make it available for architects to use. Furthermore they have tried to make architectural knowledge documentation that architects could use when building up their project to aid in constructing a solid architecture [10]. There have been multiple papers that have created tools for system developers to document their architectural design decision [3] [9] [6]. There also has been a qualitative study written by Paris et al [16] that shows how the tools differ from each other. The conclusion was that we should move towards a more universal knowledge repository.

For this repository to be complete we need to know about the undocumented lost design decisions from all the sources. It would take a big amount of resources to document and order the huge amount of architectural knowledge that is out on the web. That is why researchers have tried to make automatic systems that help with problems related to architectural knowledge [7]. It would be of great help if we could use disordered and undocumented knowledge already existing out there in the world. Ideally, we want the ability to find architecture in any sort of system. But it is incredibly time-intensive to manually search a system for all its architectural knowledge. That is why we will need tools or automated machine learning to help us with this process. If we can gather the knowledge of these systems it could be of great help for future research.

1.3 Proposed Solution

In this study, we will explore the idea of finding architectural knowledge in open source systems with a search engine. There have been multiple studies that have shown how well search engines perform on finding desired information. In the study of Jaime Teevan et al [17] they compared the performance of using a search engine and social media. Concluding that the search engine was the superior choice for finding information. In the article of Mohammed Al-Ubaydli, they found that using web search engines was of help in finding medical information [19].

We will study apache mailing lists and Jira issues. These systems were selected because previous studies have shown that these are rich in architectural knowledge [20]. For the mailing lists, we only select the lists in which primarily the developers communicate. The reasoning is that we want to find architectural knowledge documented by the architects. The developers working on a software project are most of the time also the system's architects. The remaining mailing lists of a project are not of interest to us. Because these lists will have less inter developer communication and thus contain less architectural knowledge. Jira was selected because it is a tool that supports developers in implementing new features and tracking issues. The developers working on Jira put a lot of their thoughts into the issues, making it filled with architectural knowledge.

Normally to manually search these open-source systems for architectural knowledge would require a lot of labor. In this paper, our goal is to speed up this process. A logical tool to do this with is a search engine. With them, you can quickly find resources with the desired topics. When using a search engine there will be trade-offs in the resulting precision and the recall based on the search query and the inner workings of the search engine. That is why the query you construct has to be detailed to get the results you want. There is also a balance in how detailed you make your query. If you overdo it you will end up with almost no results. Make it too broad and you will have a lot of false positives. That is why it is important for us to test multiple queries and to see which direction of query type is most suited for finding architectural knowledge.

In this study, a total of 6 queries will be constructed. five of them will be constructed based on the keywords from the book of Len Bass [4], and 1 group will be based on a previous study from Soliman et al. [12] that found significant architectural knowledge trigger words in stack overflow posts. These 6 queries will be executed on both mailing lists and Jira issues. The top 100 results for each query will be manually searched for architectural knowledge and tagged. This will enable us to answer this paper's research questions. To speed up this process we created a tool that can extract data from mailing lists and Jira issues into your database. The tool creates indexes with Lucene on all these resources. Allowing users to search and tag them accordingly. With the use of the constructed tool, we will be able to answer our research questions.

1.4 Research Questions

To quantify how successful our tool can extract architectural knowledge from the resources we will pose two research questions. Answering these questions also helps in deciding what to do for future research.

Research Question 1: How effective is searching using keywords from literature to find architectural knowledge issues and mailings? This question will be answered by the precision and the nDCG of the top-k results returned by the search engine. In other words what percentage of the search engine results are about architectural knowledge. This will require manual labor to decide which result contain architectural knowledge.

Research Question 2: Which types of architectural design decisions are discussed in architectural issues and mailing lists, which are retrieved using keyword searching? These types of ADDs will be based on the defined decisions from the paper written by A. Jansen et al [10]. After we have manually tagged all the queries. We will be able to analyze the resulting types discovered by the search engine.

2 Related Work

This paper builds on top of the already known main concepts of architectural knowledge. These concepts have been previously explored by researchers in the field of architectural knowledge. The main concepts are design decisions [8], their types [10], rationale of decisions [15]. These studies established the fundamental architectural knowledge concepts, which we consequently will base our queries on. However, they do not propose approaches for capturing or finding architectural knowledge.

There have been no papers yet on building a search engine on top of architectural resources. Thus there are no exact related works. But there has been a study that searches for architectural knowledge with a web search engine. In the research of Mohamed Soliman et al, they performed an empirical study with 53 software engineers, who used Google to make design decisions using the Attribute-Driven-Design method [13]. It concluded that using the web search engine is effective in finding architectural knowledge. This paper also proposed specialized web searching approaches to enhance the effectiveness of searching for architectural knowledge. This compared to our paper is quite similar. Instead, our approach will be exploring the effectiveness of a self build search engine on the resources and not a web-based one.

There have been other papers that have tried to do this with various other methods. For example, Gorton et al. [7] proposed an approach to identify architectural knowledge in technology documentation, and specially identify documents with certain architectural tactics (as one architectural solution). Then Bhat et al. [5] captured architectural knowledge from issue tracking systems. They especially captured and classified the different types of design decisions (as one architectural knowledge concept) in issue tracking systems. They used machine learning to achieve their goals. In our paper, we will also expand this research and try to find architectural knowledge in these systems. But instead, we will be using a search engine.

Another paper by Soliman et al. [14] improved the effectiveness of searching for architectural knowledge in Stack Overflow. This source would also be a good option to find general architectural knowledge. Another study similar to ours is one by Zhuang Xiong et al. [20]. This was an exploratory study into assumptions in the hibernate developer mailing list. Out of 9006 analyzed posts they found that 832 had assumptions within them. Over half of these assumptions were assumptions about the design. Further motivating us to use mailing lists in our study as well.

The paper of Babar et al. [2]. Had claimed to automate finding architectural knowledge in the mail. But they did not share any metrics to determine why their tool would solve that problem. And the example they gave for found architectural knowledge was quite trivial. Other than these studies we did not manage to find any study trying to extract architectural knowledge in already existing systems. So the results of our study will be quite interesting.

3 Archedetector

3.1 Tool Explanation

The tool is build in java with the following frameworks:

- Frontend: Vue.js
- Backend: Spring, Lucene

These frameworks were primarily chosen for their huge communities. Allowing our developers to easily solve most problems they have whilst coding. Of course, there are not significant drawbacks giving us reasons to not choose these well-supported frameworks.

The tool we have constructed will give researchers the ability to perform completely customizable queries on open source repositories. For now, the only repositories that can be searched are Apache mailing lists and Jira issues. In our paper, we are searching these repositories for architectural knowledge. However, you could choose to search for any information in these repositories. All the user has to do is specify which resources they want to search. Then they have to construct a suitable query tailored for which information they want to retrieve. Our tool also gives the user the ability to create custom tags to manually categorize the resources you encounter. Which will be very useful if you want to use this data in later research. You can also choose to export your search queries as a JSON file. Allowing you to analyze your search results.

3.2 Using Lucene

Apache Lucene is an open-source Java-based search library that provides Application Programming Interfaces for performing common search and search-related tasks like indexing, querying, highlighting, language analysis, and many others. Lucene is written and maintained by a group of contributors and committers of the Apache Software Foundation (ASF) [18]

The paper of Yang et al. [21] enabled the use of information retrieval by building a toolkit on top of Lucene. Their message to the information retrieval community was that Lucene is efficient and scalable without compromising effectiveness. Lucene also has the benefit of having a big community supporting it. Making it easy to start using Lucene. We also had the option of creating a self-made search engine. But not only would it require an immense amount of time and knowledge about these kinds of systems. Lucene already completely satisfies what we want our search engine to achieve. That is quick keyword searching with the ability to customize your queries and weights.

Lucene is an inverted full-text index (tf-idf). This means that it takes all the documents, splits them into words, and then builds an index for each word. Since the index is an exact string-match, unordered, it can be extremely fast.

3.3 Architecture of the searching system

The first difficult task of the tool was to extract the Jira and mailing list information into our database. To achieve this we first had to create the entities and the database model. As you can see in the relational diagram in figure 1 is how we decided to model the database. This way the database is quite scalable and satisfies all the functions we desire. We also decided to use a Postgres database since it is fast. We needed a fast database because we are performing a lot of operations on the database when adding a new project.

The back-end that was built on top of the database is called Spring. With spring we created controllers to provide access to our API and to interact with the entities within the table. We used the service pattern to do almost all the business logic. However We had some functions that could be used more generally in a utils package. Then we used JPA repositories to do our SQL transaction. To save time and immediately have all the CRUD operations available within the back-end. Integrating Lucene within this backend was quite easy. We had to write indexers for every entity we wanted to search and also write a searcher that can access those indices and perform a search query. We created and deleted the indices at the same time they were added or removed from the database. We also created a search controller which then allowed users to search the created entities with Lucene queries.

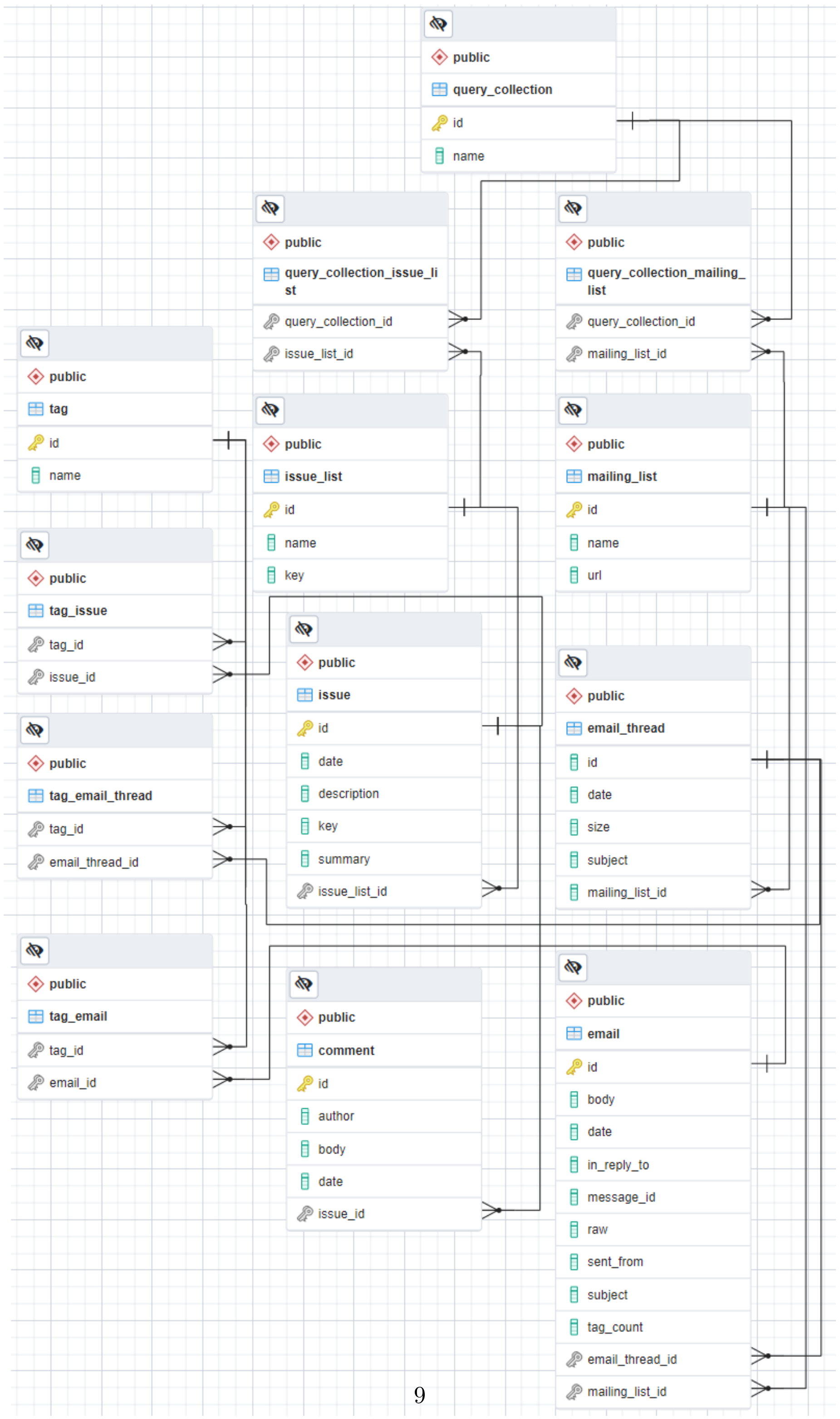


Figure 1: ERD of the database

4 Methodology

4.1 Selecting Sources For The Experiment

The tool allows importing of data from both Jira and Mailing lists. Jira is a tool used for project management and issues/bug tracking. It simplifies the communication process among developers making it a perfect candidate to search for architectural knowledge. Same with the developer mailing lists. Which also is a well-known communication option used by developers. Those are the reasons these repositories were chosen. Now that we know what repositories are being used, we have to select the software projects from which we want to extract the information. The projects used should also be rich in communication between the developers. This way we can have more success in extracting valuable information. For example, Hadoop Common is an essential part/module of the Apache Hadoop Framework. It is also a long-lasting project with a ton of recorded communication within the Jira and mail. This is a good candidate to pick as a project. A bonus is that previous research had been done about architectural knowledge in Jira issues. The paper was able to roughly identify all the issues which contained architectural knowledge. This combined with all the results from this paper could allow us to calculate other metrics, like recall of the search engine. Recall would be a very interesting metric to include. The projects used in the research are from the apache software foundation. This is because they are the biggest open-source software foundation with freely available information to crawl through [1].

When picking which mailing lists to choose within the project, we decided on the developer lists. Since architectural design decisions are made by the developers. Other lists contain way less developer communication. For example in the user mailing list there are way more questions about the code and how to set up the environment. Thus these are the mailing lists we picked:

- Cassandra-dev
- Tajo-dev
- pdfbox-dev
- tika-dev
- hive-dev
- Hadoop-dev
- Hadoop-common-dev

Within Jira, we decided on these projects.

- Cassandra
- Tajo

- Hadoop Common
- Hadoop Mapreduce
- Hadoop HDFS
- Hadoop YARN

4.2 Constructing Search Queries

Since this is an exploratory study we decided on keeping the search queries pretty broad. Architectural knowledge uses keywords from its parent category Software Architecture. Logically this paper will construct query groups with keywords from the book of Len Bass [4]. Taking the main concepts in software architecture we are left with 5 groups called decision factors (table 1), component structure (table 2), tactics (table 3), pattern (table 4) and rationale (table 5). We then looked through the book's table of contents and references in the back of the book. Each word that had an architectural meaning we grouped to one of the groups listed above. For our last query, we decided on constructing it based on a previous paper from Soliman et al. [12]. The keywords were based on their significant keyword triggers for architectural knowledge with Stack Overflow posts. This group is called significant (table 6). These queries are all expected to perform reasonably well. One of the concerns is that they are still too general and will return a lot of non-relevant documents. In each query group, every keyword has the same weight.

4.3 Tags of architectural design decisions

We decided on a total of 8 tags. The first and most general tag is called "Architectural Knowledge". This tag would be placed anytime a resource had any type of architectural knowledge within it. The remaining 7 tags are all types of architectural design decisions taken straight from Kruchten et al. [10]. Called Structural, behavioral, property, ban, technology, tool, and process. We tag a resource with these tags if one or more of these decision types were identified within it. Below are the basic definitions for each ADD. Each time we classified an email or post we returned to these definitions:

- Architectural Knowledge: If one of the definitions below is found we use this tag to signify that it has AK.
- Structural: lead to the creation of subsystems, layers, partitions, components in some view of the architecture.
- Behavioral: are more related to how the elements interact together to provide functionality or to satisfy some non-functional requirement (quality attribute), or connectors.

- Property: states an enduring, overarching trait or quality of the system. Property decisions can be design rules or guidelines (when expressed positively) or design constraints (when expressed negatively)
- Ban: stating that some elements will not appear in the design or implementation.
- Technology: When deciding on which technologies you will use for the software. For example Java, C, Postgres, SQL, or any other
- Tool: If there is a mention about what tool the developers are required to use. A few examples are Github, Jira, IntelliJ, or any other tool a developer might use.
- Process: A process decision is about the overarching process the developers have to follow when programming for the software.

4.4 Research Process

The research process is very simple and straight forward. There are only a few steps the participant had to do. First step is to execute the search query in one of the repositories. Next step is to browse every source one by one and check whether it contained architectural knowledge within it. One way participants could recognise architectural knowledge is with help of the ADD definitions. The final step is to tag the resource with one or multiple ADD tags. The further they got into their tagging process the more examples the participant had. This helped participants in making improved tagging decisions when they check the search results for the second time. These steps are repeated until all 6 queries have been executed and the top 100 results have been tagged for both the Jira and the Mailing list projects. This will yield us a total of 600 mail and 600 issues to perform analysis on. Analysis was done with the help of a python script.

4.5 Measurements of effectiveness

To answer research question 1 we will need to use multiple metrics. These metrics allow for better insight into the effectiveness of our search engine. The first metric we decided on is called precision. Precision is called according to the following formula:

$$precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{retrieved\ documents}$$

Precision is calculating what percentage of your retrieved documents are relevant. This is a good metric to measure if our search engine is returning enough relevant documents. If not then you should tweak your query. If that still does not work it might be that your dataset is not filled with relevant documents.

Ideally, we would have also wanted to calculate the recall of a query. The recall is defined by the following formula:

$$recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{relevant\ documents}$$

the recall is the fraction of the relevant documents that are successfully retrieved. In a good search query, you would want this as high as possible without losing too much precision. The problem with calculating recall is that you are required to know all the relevant documents in your data set. Our data set for the mailing lists alone contained over 70.000 emails. Considering it took 2-3 hours on average to tag 100 resources. This would take around 1400+ hours to complete. Thus we do not know all the relevant documents in the data set and are not able to calculate the recall for the queries. Instead, we decided to go with another metric to see how well the search engine can rank the returned documents. This metric is called the Discounted Cumulative Gain:

$$DCG = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)}$$

The premise of DCG is that highly relevant documents appearing lower in a search result list should be penalized. Thus the graded relevance value is reduced logarithmically proportional to the position of the result. Which we can argue is the functionality we desire from our search engine. Optimally our search engine ranks highly relevant documents higher.

Search result lists vary in length depending on the query. Comparing a search engine's performance from one query to the next cannot be done by using DCG alone. DCG can not tell us how well it ranked the results based on the query's maximum potential. This is because DCG does not take into account what the ideal ranking possibility of a query is. Instead this can be achieved by sorting all documents in the result list by their relative weight and then calculating the IDCG:

$$IDCG = \sum_{i=1}^{|REL_p|} \frac{rel_i}{log_2(i+1)}$$

With this metric we can then compute the nDCG to see how well our query ranked the documents:

$$nDCG = \frac{DCG}{IDCG}$$

If nDCG=1 the ranking of our search is theoretically perfect. So preferably we want this metric to perform well on the search engine.

5 Results

We have to be careful when concluding our results. Ideally, we would have tested the top 100 results for each project. But this would have been way too time intensive. Thus for lower k values, we have to take this into account. Higher k values will be more accurate due to the higher sample size. That is why we value the graphs higher k more than the lower k values. However, this does not mean we discredit the lower k values. They are still relevant. This is also why the graphs are not incredibly smooth. Instead, an algorithm is used to smoothen the graphs. This is done to increase the readability of the figures, without losing any important information.

5.1 RQ1: Effectivity of the search engine

Figure 2 shows the precision of the queries run over the mailing list resources. As we can see all 6 queries show their relative precision over k resources. We can see that some queries are performing way better than others in finding architectural knowledge. This is very much to be expected since good queries are of big importance in finding what you desire. A drastic and easy example is to imagine if the query used is about fruits. We will not find any architectural knowledge. In general the precision in the mailing list is decent. This is especially the case for the better performing queries.

It is immediate to see that the pattern query (Table 4) and tactic query (Table 3) are performing poorly over all the k values. Especially tactic is not well suited to find architectural knowledge in the mail. The returned resources for the tactic query were mostly code question's, bugs, or other random mail between the developers. The same thing counts for the pattern query. It is still good to see that even though they did perform pretty poorly compared to the rest of the queries. They still find more architectural knowledge in the lower k range. Meaning that the query used is not a random unsuitable query. It does find architectural knowledge, but way less impressive than the other queries.

The other 4 queries are more closely matched in precision. We are able to see that Rationale (Table 5) is performing quite well. Especially on $k < 10$ but on $k > 50$ it stays behind. The significant (Table 6), decision Factor (Table 1) and component query (Table 2). They all seem to have $p > 0.8$ for $k < 10$ and

$p \approx 0.45$ for $k = 100$. This is indicating that they are all really good queries for finding architectural knowledge.

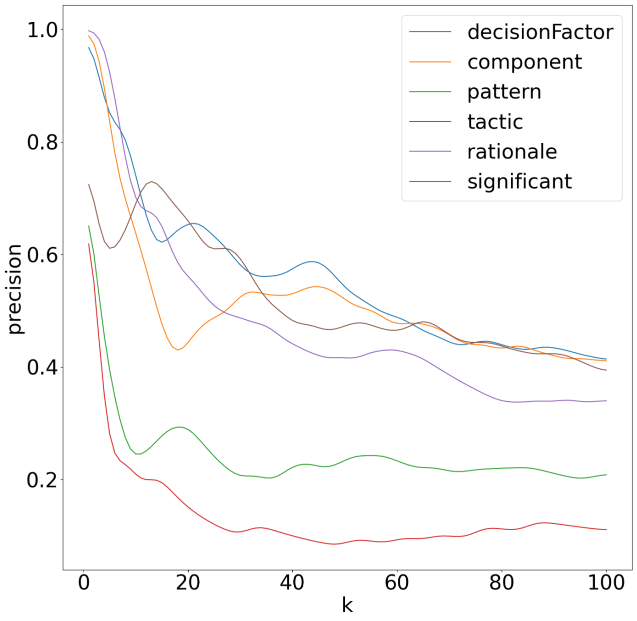


Figure 2: Precision of the queries in the Mailing Lists

What we can see here in Figure 3 is the NDCG over k for mailing lists. We observe that the value quickly does not change that much for $k > 20$. That is because this function is logarithmic. Valuing the earlier hits way more than the latter. It does not tell us too much more than what Figure 2 already told us. It does help us distinguish between the close top 4. And pick the decision factor query as having better effectiveness than the other. Tactic and pattern remain just as lackluster in this department. This of course due to lower precision in general of these queries. Compared to the other queries these 2 seem to not be as suitable for finding architectural knowledge.

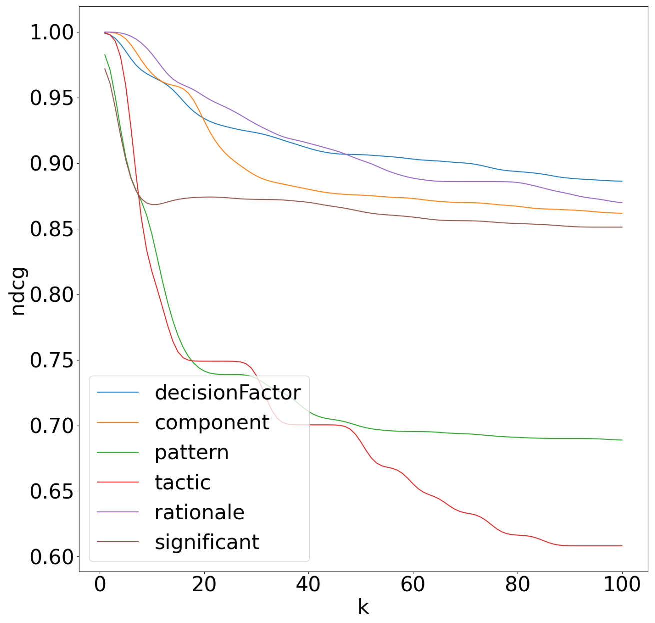


Figure 3: nDCG of the queries in the Mailing Lists

Figure 4 shows the Precision of the queries over k in Jira projects. Decision factor and component query are performing exceptionally well. This could be due to Jira being rich in those kinds decisions. Surprisingly though most of them contained architectural knowledge. Tactic query has some really terrible precision over $0 < k <= 100$. This again shows that the tactic query is not very suitable for finding architectural knowledge compared to the rest of the queries. The pattern is also not doing well. But at least is performing a little bit better than a tactic. One positive of the tactic query is that it found resources that were not found by all the other queries. This is because of the unique keywords used in the tactic query. The significant query starts strong but surprisingly falls off quite a bit in the latter part. Rationale performs quite good over $0 < k <= 100$. All the queries are performing roughly the same as in Figure 2. This re-ensures that the queries carry over into different types of resources. And should be expected to perform roughly the same.

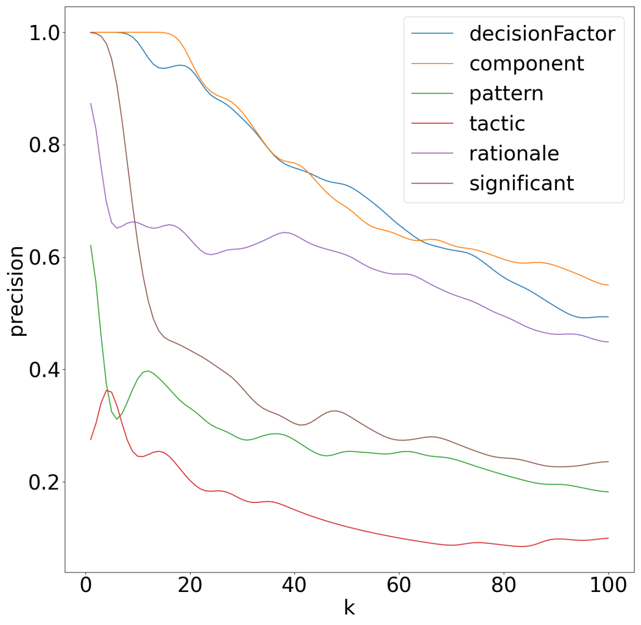


Figure 4: Precision of the queries in the Jira Projects

Figure 5 contains the NDCG over k for each query in the Jira projects. Also, we can see that the tactic query missed $k = 1$. Resulting in it drastically underperforming. But even if it did hit the first resource it still would have been bad. The pattern also belongs to the worse queries based on the NDCG value. Then the rationale and significant queries both have good NDCG values. But the decision factor and component query blow the other queries out of the water. This is due to the incredibly high precision in the $k < 20$ for both these queries.

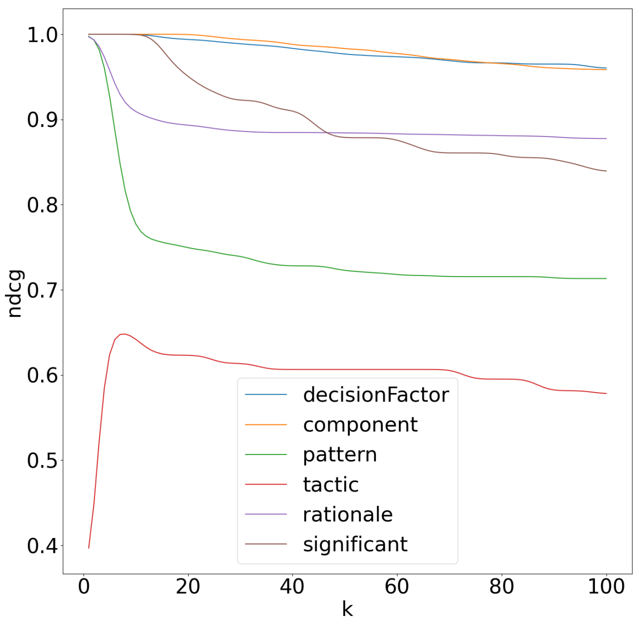


Figure 5: nDCG of the queries in the Jira Projects

Figure 6 shows the average precision for both the Jira and mail so we can easily compare the two. We can see that for $k < 5$ the mail has a better precision than Jira. This is a result of Jira missing a few hits in the earlier stages. This does not tell the entire story since only 6 queries have been tested. Which is too low of an amount to see how well the resource performed in the earlier stages. We can quickly see that Jira starts to take over for $k > 5$. We do observe that the precisions are starting to merge. This can mean that both resources start having the same k value for $k > 100$.

But since we are talking about a search engine we preferably want the search engine to perform well on the first pages. That is where we look at the average NDCG value in figure 7. But as stated before the NDCG uses binary values for our experiment. This is very sub-optimal not telling the whole story. Each identified architectural knowledge resource can greatly vary in how much relevance they have in that subject. For example for some resources, we only find 1-2 sentences. Whilst for others we find pages of architectural knowledge. But it is still relatively useful to assure that the search engine hits more resources at $k < 20$ after this the NDCG barely changes value. Due to one of the queries of Jira missing the first hit. The Jira average NDCG suffers a great hit in the lower k range. And so the mail has a superior NDCG for $k < 17$. But at $k > 18$

Jira manages to outperform the mailing list on ndcg.

What we can also observe is that the precision is directly related to the NDCG. This is even more so because we are using a binary NDCG. High precision in $k < 10$ automatically results in a good NDCG. Which is not entirely correct. The other reason the NDCG is quite high is because of the IDCG. For this, we sorted all relative resources per k giving us the IDCG per k. Quick visual example is for $k = 5$ the returned search is $\{0, 0, 1, 0, 1\}$. For the IDCG this set would be sorted and look like this $\{1, 1, 0, 0, 0\}$. Whilst you could argue that it should always hit a true positive all of the time. That way the set would look like $\{1, 1, 1, 1, 1\}$. This however is unrealistic, instead, you want to have tagged the entire project. That way you can derive the IDCG from that data. But for now, sorting it seemed to be the most viable way to calculate the NDCG.

In general for both Jira and Mailing lists are able to have about $p > 0.6$ for $k < 10$. Even for $k = 100$ the $p \approx 0.35$. This is a very good sign considering the queries are far from optimized. Making more specific queries and pruning bad queries can greatly increase the precision of the engine.

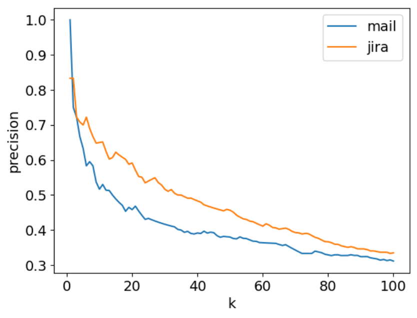


Figure 6: Average precision over k

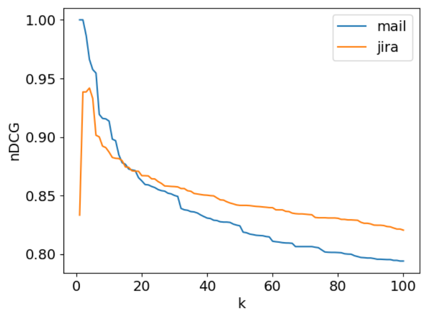


Figure 7: Average query nDCG

5.2 RQ2: Which ADDs are discussed

Observed in Figure 8 is a distribution of architectural design decisions per query. We can see that all the types of ADD were discussed except for the bans. Bans are not found at all in the 300 mail reviewed. This is most likely since these decisions are rarely documented.

Tool and Property decisions were also barely found. Tool decisions are of course scarce. Due to not often changing your editor or tools used like GitHub. But we still found them in the mail which is quite interesting to see. Also notable is that the significant query found the most tool decisions. A property decision states an enduring, overarching trait or quality for the system. Not too often do the developers discuss the property decisions. But you should still be able to find them which is what was barely able to achieve.

Technology decisions were found better than expected in the mail. And especially process decisions we did not expect this many off. But since it is a developer mailing list, it will be discussing the processes developers working on the project have to adhere to. For both technology and process decisions the top query for the mailing list was the significant query. Since this query was based on significant trigger words for architectural knowledge. It is good to see that it performs so well in the macro department (technology, process, tool) of architectural knowledge.

As for structural and behavioral decisions, they should occur the most often in an architectural system. Since these decisions are taken the most frequently. Thus we should see this happening in our results. It is pretty surprising that relatively the same amount of decisions were found for structural, technology, and process. All queries performed relatively well in this department. This could mean is that mailing lists are a suitable candidate for finding the macro decisions (technology, process, tool). It could also mean the mailing list contains a bad amount of structural and behavioral decisions. Or of course, a combination of the two is also possible. The component query did perform the best for finding both structural and behavioral decisions. We also found that the most occurring decision is the behavioral decision. This is what we expected to see. Because behavioral decisions are more related to how elements interact together. And there are a lot of elements in a software architecture system. Naturally, these decisions will also be made much more.

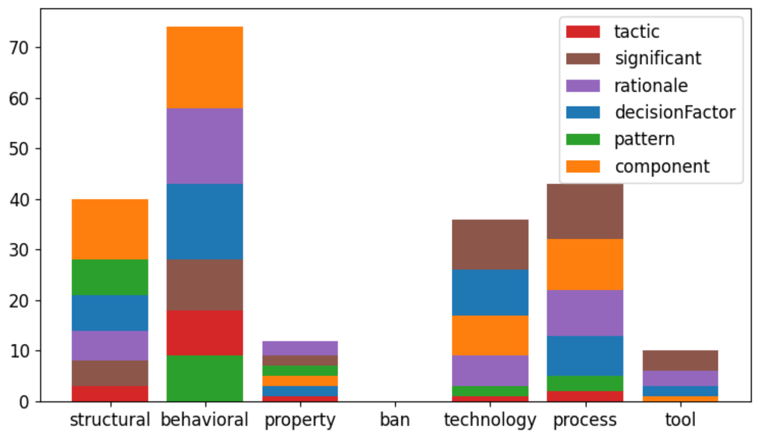


Figure 8: Distribution of ADD per Query in Mailing Lists

We can see how the Jira ADDs are distributed in Figure 9. Immediately obvious is that it performs very well at finding structural and behavioral decisions. But is extremely awful at finding all the other decisions. Zero process and tool decisions were found. Jira Issues are more about the features, bugs, and other coding-related issues. This explains why there is such an incredible amount of structural and behavioral decisions. This can also explain why the technology process and tool decisions were not present. Process and tool decisions could be made outside of the Jira issues. Whilst technology decisions should be present but are rare. Which is what the graph reflects. As for the ban and property decision, we discussed in Figure 8 it is in general harder to find these. Although more ban decisions were predicted.

The Component query excelled at finding structural decisions within the Jira issues. It still performed well at finding behavioral. And was one of the only queries that found technology decisions. Again the tactic and pattern query were not finding too many decisions. Only a few behavioral and minimal structural decisions. The decision factor query found a significant amount of behavioral decisions. The rationale query again performed well. But was still outdone by the component and decision factor queries.

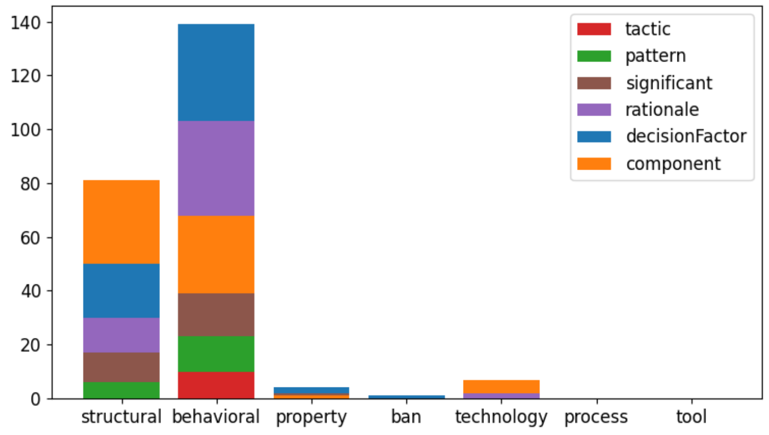


Figure 9: Distribution of ADD per Query in Jira Projects

There were quite a few interesting differences between Jira and the Mailing List. Jira found 2 times the amount of structural and behavioral decisions. Instead finding almost none of the other decisions. Also interesting to see is that when a query performed well in finding a certain decision. It also found other decisions more easily. Whilst if a query found fewer decisions it would find fewer decisions in all of the decision types. This could tell us that the queries we used are too broad to find a specific decision and we would need to tailor the query more towards a certain decision type.

In both resources the tactic and pattern query underperformed. Finding way fewer hits than the other 4 queries. One of the reasons this happened is because in these queries there are a few keywords that hit a lot of false positives. One example would be in the Jira resources the keyword 'ping'. This was used by the developers to bump their issues. This way more developers would see this issue in their timeline. In this research, we did not track which keywords were hitting what kind of resources. So it is hard to speculate about this part of the research. The Component and Decision Factor performed well over all the decisions. But especially on behavioral and structural decisions. The rationale query is overall performing quite well, but this query found mostly behavioral decisions. Significant performed quite well in finding technology and tool decisions in the mailing lists. But when it came to finding them in the Jira it did not find them. This is probably because these decisions are not very prevalent in Jira.

6 Discussion

6.1 RQ1: How effective is searching with the search engine

1) implication for practitioners: In our study, it is shown yet again how useful searching using engines are. Since the queries are constructed based on all the keywords of a software architecture book. We doubt that real-world practitioners can extract any useful information with these broad queries. However, we also show that some queries do perform significantly better at finding architectural knowledge than other queries. What this could mean for practitioners is that they could use the search engine for finding very specific resources within the Jira and Mailing List. As long as they know almost exactly what they want to find within them.
2) implication for researchers: Figures 4 & 2 show that search engines are quite promising in finding architectural knowledge. That is why researchers could try and further confirm that search engines can find an adequate amount of architectural knowledge. One way they could do this is to build a complex query with machine learning and see just how high you could push the recall and precision of the search engine.

6.2 RQ2: Which ADDs are discussed

1) implication for practitioners: We can recommend practitioners that are looking for specific architectural design decisions within the developer systems. That they should try to first find behavioral and structural decisions in Jira. Or if instead, they are looking for technology and process decisions that they should

look in the mailing lists. Practitioners also have to be very mindful of the queries they construct when using a search engine. Since our research shows that different queries can return different results. We caution practitioners again to be specific in constructing their queries.
2) implication for researchers: Our experiment result Figures 8 & 9 show clear distinctions between the types of ADDs in different systems. This could encourage researchers to extend their research on more systems. For example, if they were to do architectural knowledge research with Jira. You might want to check if other systems are also interesting to include in your research.

7 Threats To Validity

The experiment was performed by a single student with minimal experience with architectural knowledge. Of course, he did train in being able to recognize architectural design decisions. So the error's made due to inexperience should be minimal. The researcher also had access to good definitions which helped further improve the accuracy. Some of the results were also approved by the supervisor of the experiment. Which further reduces the fault rate. There could also be human errors whilst tagging the resources. For example, forgetting to save the tags. Or maybe even skipping over a resource. These errors were further reduced by checking the queries at least 2 times. Also because the student was alone he could have been biased towards different architectural design decisions. Or when a source contained architectural knowledge or not.

The results are error-prone because we run every query only once per resource. It would have been better if we had been able to search on 10 groups of projects individually. This is to see the average performance of the query. This would have made the results more robust. Now with the found results from the query, it can be quite different if we had run the test on another group of projects. We could have also found more project-specific results which would have been interesting.

8 Conclusion

Our main goals in this paper were to determine whether a search engine is effective at finding architectural knowledge in open-source repositories, and what kind of architectural design decisions were made. To achieve these goals we performed an exploratory study. The results of this study are quite interesting. The pattern and tactic query were deemed inefficient compared to the other queries. But still not very terrible. The component and decision factor queries performed extremely well as shown in the results. If you have limited resources and can not just scour through all of the resources manually. A search engine would be of great benefit to speed up finding relevant architectural knowledge in both Jira and Mailing Lists. In our study, we did not find the efficiency to drastically differ between the two.

As for the types of ADDs found in the resources. Jira is more suitable for finding architectural and behavioral knowledge. Especially component, rationale, and decision factor are suitable queries to find architectural knowledge within Jira. On the other hand mailing lists are more suitable for finding the macro architectural design decisions. The Significant query performed well in finding process and technology decisions in the mailing list. Component, rationale, and decision factor also performed well in general in the mailing lists.

You have to be very careful about what query you construct. The queries were greatly increasing or decreasing the efficiency of the search engine. They also found reasonably different ADD types so tailor your queries with this in mind. The best queries we found for architectural knowledge were component, rationale, and decision factor.

9 Future Work

To improve this research it also would have been interesting to include the documents attached to the emails and jiras. As of this moment, the tool does not have any kind of implementation for this area yet. In quite a few Jira issue's that were about architectural knowledge, we observed that they also had a design document attached. It would be quite valuable if we were able to view this and search this documen.

It would be very fascinating to see the recall of the search engine. That is why we propose to do similar research on a completely tagged project. With the recall and precision, we could determine the accuracy of the search engine. Which is a really helpful metric for search engines. Furthermore after having completed all the tagging in a project. You could continue to improve the queries with some sort of supervised learning algorithm. Either that approach or try to construct an improved query yourself by analyzing the results.

Since the tool helps researchers with finding relevant resources. It would also be of interest to integrate more systems into the available resources. Right now these are only Mailing lists and Jira. But we could also try to include Git or Trello or any other kind of developer tool. To see which of them is the most architectural rich. We also propose to instead of manually tagging the resources. Use a supervised machine learning algorithm to learn from your previous tagged projects and see how well they perform in classifying the resources.

10 References

Table 1: Keywords for Decision Factors Group

#	Decision Factor
1	Goal
2	quality
3	scenario
4	requirement
5	issue
6	criteria
7	demand
8	performance
9	security
10	availability
11	modifiability
12	reusability
13	flexibility
14	reliability
15	usability
16	testability
17	safety
18	interoperability
19	variability
20	portability
21	scalability
22	Mobility
23	Safety
24	Conceptual
25	integrity
26	runtime
27	realtime
28	port
29	scale
30	constraint
31	context
32	limitation
33	cost
34	“time to market”
35	bottleneck
36	resources
37	attack
38	“down time”
39	“time to repair”

40	complexity
41	complex
42	effort
43	coupling
44	cohesion
45	latency
46	throughput
47	efficiency
48	confidentiality
49	integrity
50	authorization
51	authentication
52	satisfaction
53	confidence
54	“user needs”
55	bandwidth
56	network
57	rate
58	speed
59	cost

Table 2: Keywords for Component Structure Group

#	Component Structure
1	structure
2	module
3	component
4	connector
5	element
6	relation
7	architecture
8	design
9	decision
10	abstract
11	behavior
12	interaction
13	service
14	peer
15	client
16	server
17	process
18	“data store”
19	thread

20	directory
21	file
22	decomposition
23	submodule
24	interface
25	layer
26	class
27	system
28	model
29	view
30	database
31	storage
32	coordination
33	communication
34	“run in parallel”
35	parallelism
36	concurrency
37	dependencies
38	parallel
39	processor
40	memory
41	exchange
42	notify
43	notification
44	choose
45	distributed
46	backend
47	frontend
48	middleware
49	share
50	communicate
51	messaging
52	delivery
53	centralized
54	platform
55	publish
56	subscribe
57	producer
58	consumer
59	persistence
60	cluster
61	queue
62	message

Table 3: Keywords for Tactics Group

#	Tactic
1	tactic
2	ping
3	echo
4	monitor
5	heartbeat
6	redundancy
7	transaction
8	replication
9	spare
10	rollback
11	orchestrate
12	encapsulate
13	refactor
14	intermediary
15	concurrency
16	event
17	caching
18	“load balancer”
19	intrusion
20	encrypt
21	separate
22	revoke
23	lock

Table 4: Keywords for Pattern Group

#	Pattern
1	pattern
2	layers
3	tier
4	broker
5	MVC
6	“pipe and filter”
7	“peer to peer”
8	SOA
9	ESB
10	synchronous
11	asynchronous
12	publish
13	subscribe
14	shared data

15	adapter
16	composite
17	proxy
18	decorate
19	decorator
20	observer
21	factory
22	facade
23	strategy
24	visitor
25	alternative
26	versus
27	opinion
28	choice
29	choose
30	oriented

Table 5: Keywords for Rationale Group

#	Rationale
1	trade off
2	trade offs
3	risk
4	risks
5	assumption
6	assumptions
7	benefit
8	benefits
9	drawback
10	drawbacks
11	advantage
12	advantages
13	advantageous
14	disadvantage
15	disadvantages
16	disadvantageous
17	comparison
18	pros
19	cons
20	boost
21	fast
22	easier
23	easy

24	decide
25	quick
26	good
27	better
28	difference
29	difficult
30	select
31	slow
32	improve
33	improvement

Table 6: Keywords for Significant Group

#	Significant
1	lightweight
2	complex
3	overkill
4	recommend
5	suggest
6	propose
7	good
8	outperform
9	important
10	requirement
11	criteria
12	demand
13	depend
14	implement
15	“count on”
16	need
17	require
18	want
19	ask
20	offer
21	provide
22	supply
23	support
24	select
25	choose
26	use
27	prefer
28	“go with”
29	which

30	what
31	when
32	how
33	versus
34	vs
35	against
36	contrast
37	difference
38	distinction
39	fast
40	slow
41	heavy
42	quick

References

[1] Aleem Akhtar. *Role of Apache Software Foundation in Big Data Projects.* May 2020.

[2] Aman-ul-haq and Muhammad Ali Babar. "Tool support for automating architectural knowledge extraction". In: *Sharing and Reusing Architectural Knowledge, ICSE Workshop on* 0 (May 2009), pp. 49–56. DOI: `10.1109/SHARK.2009.5069115`.

[3] Muhammad Ali Babar and Ian Gorton. "A Tool for Managing Software Architecture Knowledge". In: *Second Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI'07: ICSE Workshops 2007).* 2007, pp. 11–11. DOI: `10.1109/SHARK-ADI.2007.1`.

[4] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* 3rd. Addison-Wesley Professional, 2012. ISBN: 0321815734.

[5] Manoj Bhat et al. "Automatic Extraction of Design Decisions from Issue Management Systems: A Machine Learning Based Approach". In: *Software Architecture.* Ed. by Antónia Lopes and Rogério de Lemos. Cham: Springer International Publishing, 2017, pp. 138–154. ISBN: 978-3-319-65831-5.

[6] R. Capilla, F. Nava, and C. Carrillo. "Effort Estimation in Capturing Architectural Knowledge". In: *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering.* ASE '08. USA: IEEE Computer Society, 2008, pp. 208–217. ISBN: 9781424421879. DOI: `10.1109/ASE.2008.31`. URL: `https://doi.org/10.1109/ASE.2008.31`.

[7] Ian Gorton et al. "Experiments in Curation: Towards Machine-Assisted Construction of Software Architecture Knowledge Bases". In: *2017 IEEE International Conference on Software Architecture (ICSA)*. 2017, pp. 79–88. DOI: `10.1109/ICSA.2017.27`.

[8] A. Jansen and J. Bosch. "Software Architecture as a Set of Architectural Design Decisions". In: *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. 2005, pp. 109–120. DOI: `10.1109/WICSA.2005.61`.

[9] Anton Jansen et al. "Tool Support for Architectural Decisions". In: *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*. 2007, pp. 4–4. DOI: `10.1109/WICSA.2007.47`.

[10] Philippe Kruchten, Patricia Lago, and Hans Vliet. "Building Up and Reasoning About Architectural Knowledge". In: vol. 4214. Dec. 2006, pp. 43–58. ISBN: 978-3-540-48819-4. DOI: `10.1007/11921998_8`.

[11] Dewayne E. Perry and Alexander L. Wolf. "Foundations for the Study of Software Architecture". In: *SIGSOFT Softw. Eng. Notes* 17.4 (Oct. 1992), pp. 40–52. ISSN: 0163-5948. DOI: `10.1145/141874.141884`. URL: `https://doi.org/10.1145/141874.141884`.

[12] Mohamed Soliman, Matthias Galster, and Matthias Riebisch. "Developing an Ontology for Architecture Knowledge from Developer Communities". In: *2017 IEEE International Conference on Software Architecture (ICSA)*. 2017, pp. 89–92. DOI: `10.1109/ICSA.2017.31`.

[13] Mohamed Soliman et al. *Exploring Web Search Engines to Find Architectural Knowledge*. Mar. 2021.

[14] Mohamed Soliman et al. "Improving the Search for Architecture Knowledge in Online Developer Communities". In: *2018 IEEE International Conference on Software Architecture (ICSA)*. 2018, pp. 186–18609. DOI: `10.1109/ICSA.2018.00028`.

[15] Antony Tang, Yan Jin, and Jun Han. "A rationale-based architecture model for design traceability and reasoning". In: *Journal of Systems and Software* 80.6 (2007), pp. 918–934. ISSN: 0164-1212. DOI: `https://doi.org/10.1016/j.jss.2006.08.040`. URL: `https://www.sciencedirect.com/science/article/pii/S0164121206002287`.

[16] Antony Tang et al. "A comparative study of architecture knowledge management tools". In: *Journal of Systems and Software* 83.3 (2010), pp. 352–370. ISSN: 0164-1212. DOI: `https://doi.org/10.1016/j.jss.2009.08.032`. URL: `https://www.sciencedirect.com/science/article/pii/S0164121209002295`.

[17] Jaime Teevan, Katrina Panovich, and Meredith Morris. *A Comparison of Information Seeking Using Search Engines and Social Networks*. May 2010.

[18] *The Apache Software Foundation*. `http://www.apache.org/`. Accessed: 2021-07-20.

[19] Mohammad Al-Ubaydli. “Using Search Engines to Find Online Medical Information”. In: *PLOS Medicine* 2.9 (Aug. 2005), null. DOI: 10.1371/journal.pmed.0020228. URL: https://doi.org/10.1371/journal.pmed.0020228.

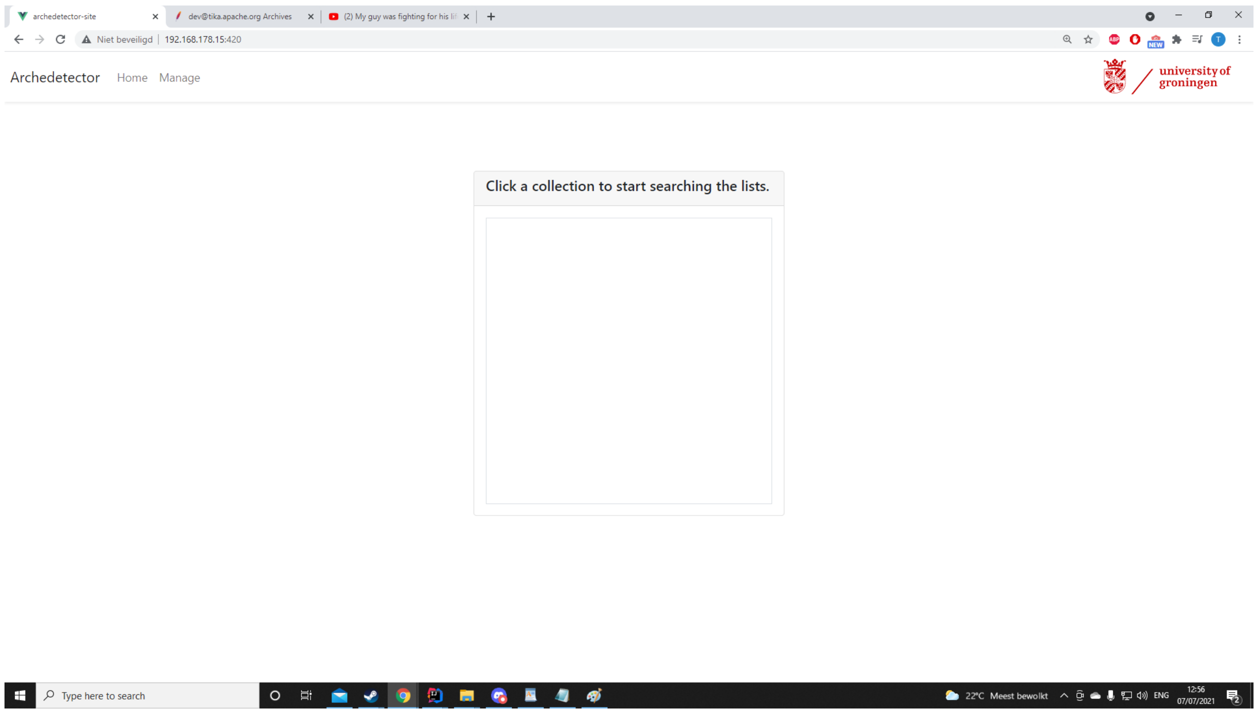
[20] Zhuang Xiong et al. “Assumptions in OSS Development: An Exploratory Study through the Hibernate Developer Mailing List”. In: Dec. 2018. DOI: 10.1109/APSEC.2018.00060.

[21] Peilin Yang, Hui Fang, and Jimmy Lin. “Anserini: Enabling the Use of Lucene for Information Retrieval Research”. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’17. Shinjuku, Tokyo, Japan: Association for Computing Machinery, 2017, pp. 1253–1256. ISBN: 9781450350228. DOI: 10.1145/3077136.3080721. URL: https://doi.org/10.1145/3077136.3080721.

11 Appendix

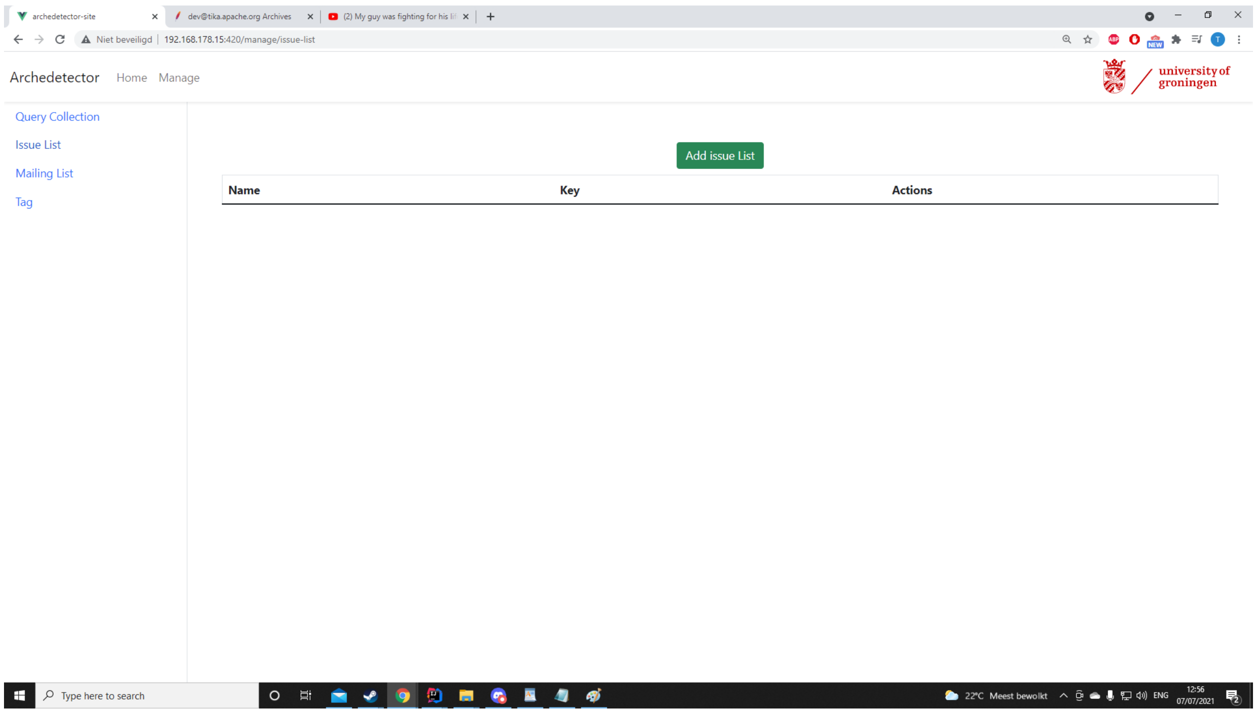
11.1 User Guide

To start browsing a collection we will first have to create one. For this we will need to first add the project you want to browse to the database. At the moment we are only able to add from apache jira and apache mailing archive. Navigate to manage page in the navigation bar.



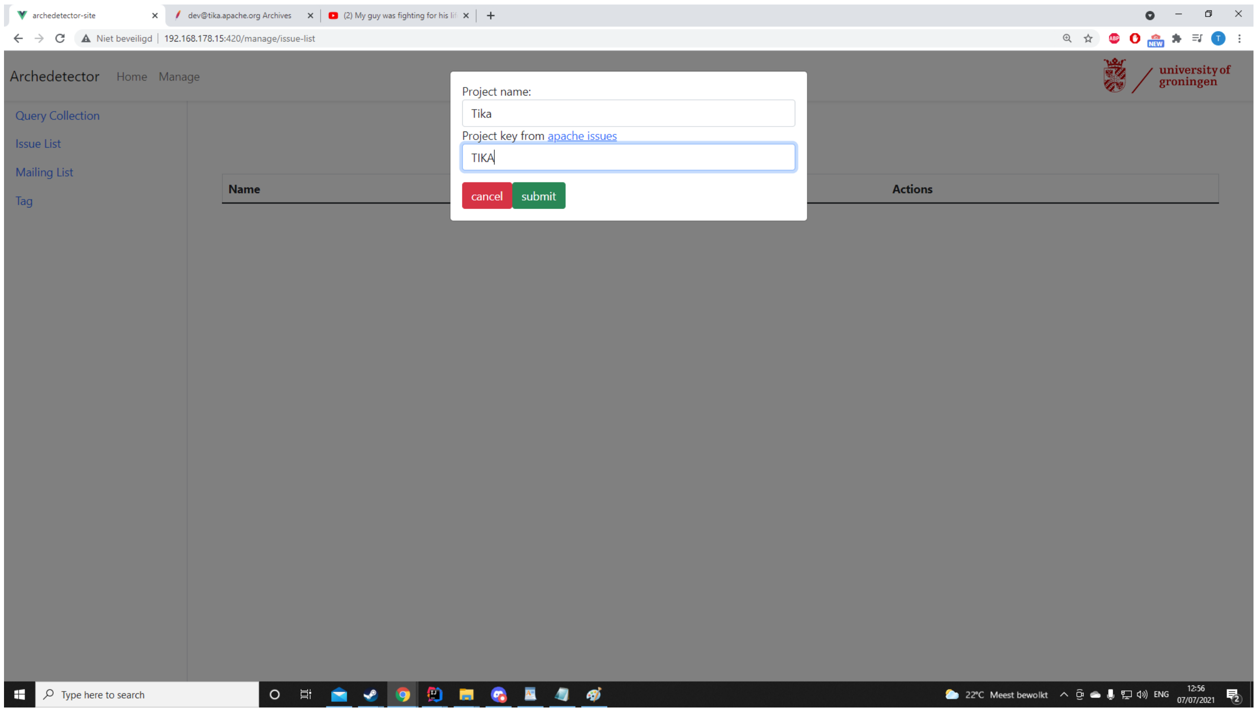
Click a collection to start searching the lists.

Click Issue List in the sidebar and then the add issue list button



Name	Key	Actions

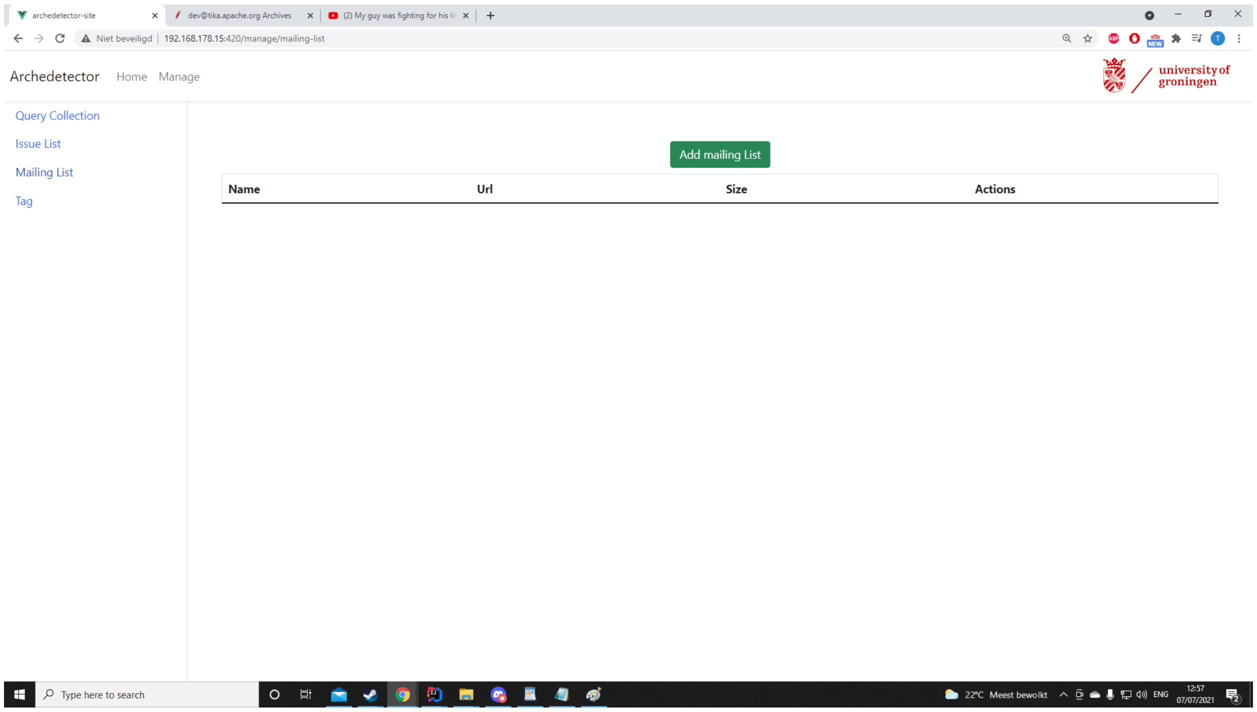
Search for the desired project you want to add at https://issues.apache.org/jira/secure/BrowseProjects.jspa?selectedCategory=all&selectedProjectType=software. Then give the key to the jira project and name it as you want, probably the same name as the jira project. Then click submit. Now this will add all the issues belonging to the project key to the database. This might take a while depending on the project size.



Project name:
Tika

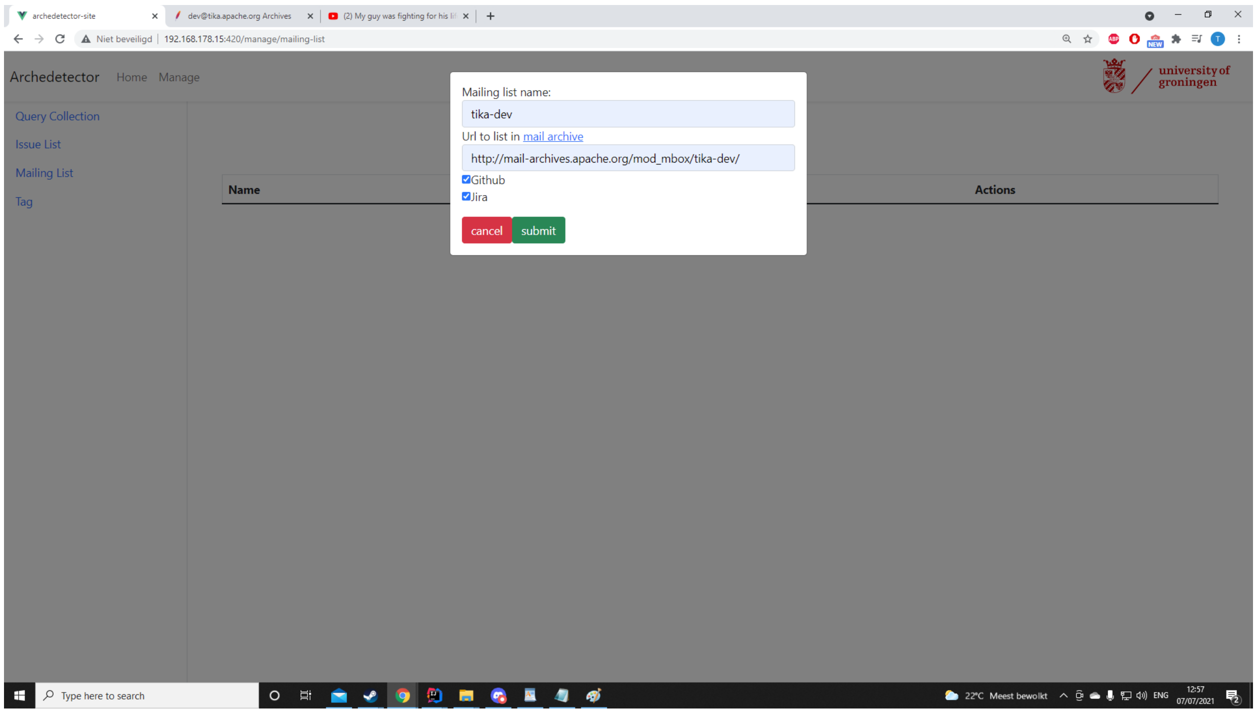
Project key from apache issues
TIKA

Now we do the same for the mailing list. Click add mailing list.



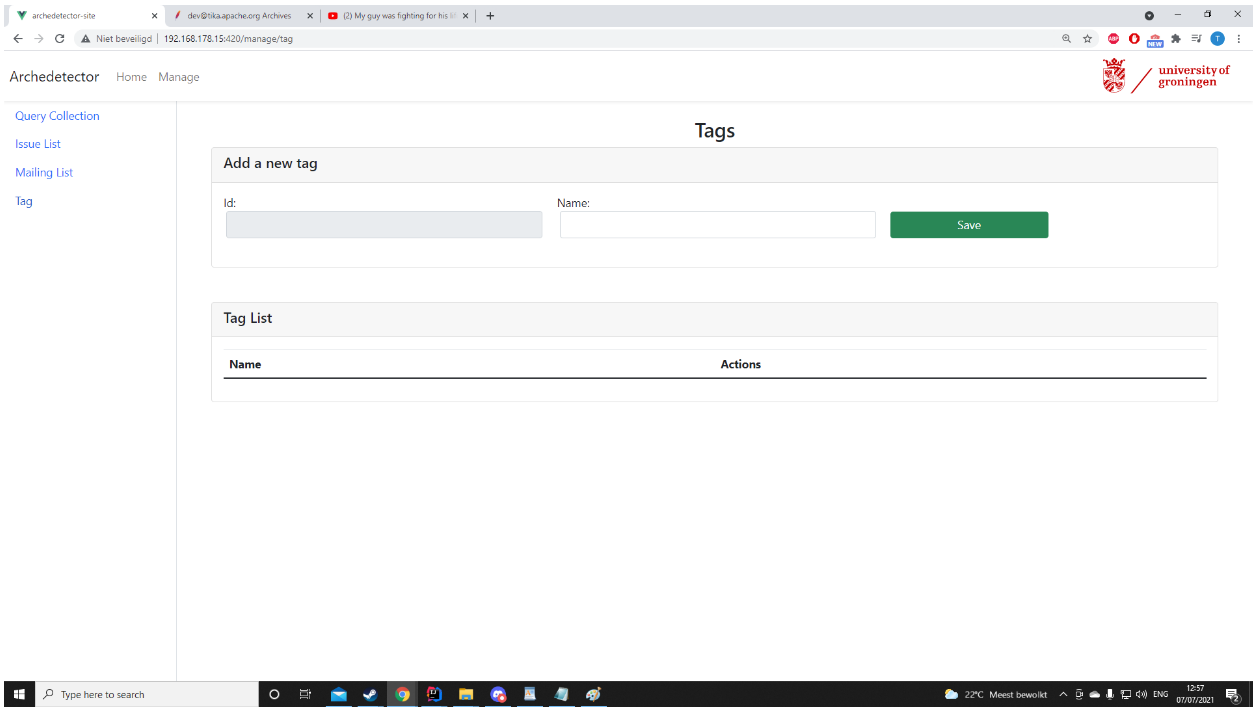
Name	Url	Size	Actions

Search for the desired mailing list you want to browse at http://mail-archives.apache.org/mod_mbox/. Find the mailing list you want to add and set it as url, the format should look like this: http://mail-archives.apache.org/mod_mbox/tika-dev/. Then name it as you desire. Probably just the mailing list name as well. And select you want to filter out the github and jira emails. Then press submit. This will add the mailing list to the database. This might take a while depending on the project size.



Mailing list name:
tika-dev
Url to list in mail archive
http://mail-archives.apache.org/mod_mbox/tika-dev/
☑ Github
☑ Jira

You also have the possibility to add tags to the lists you want to browse. Go to the manage tag page.



Tags

Add a new tag

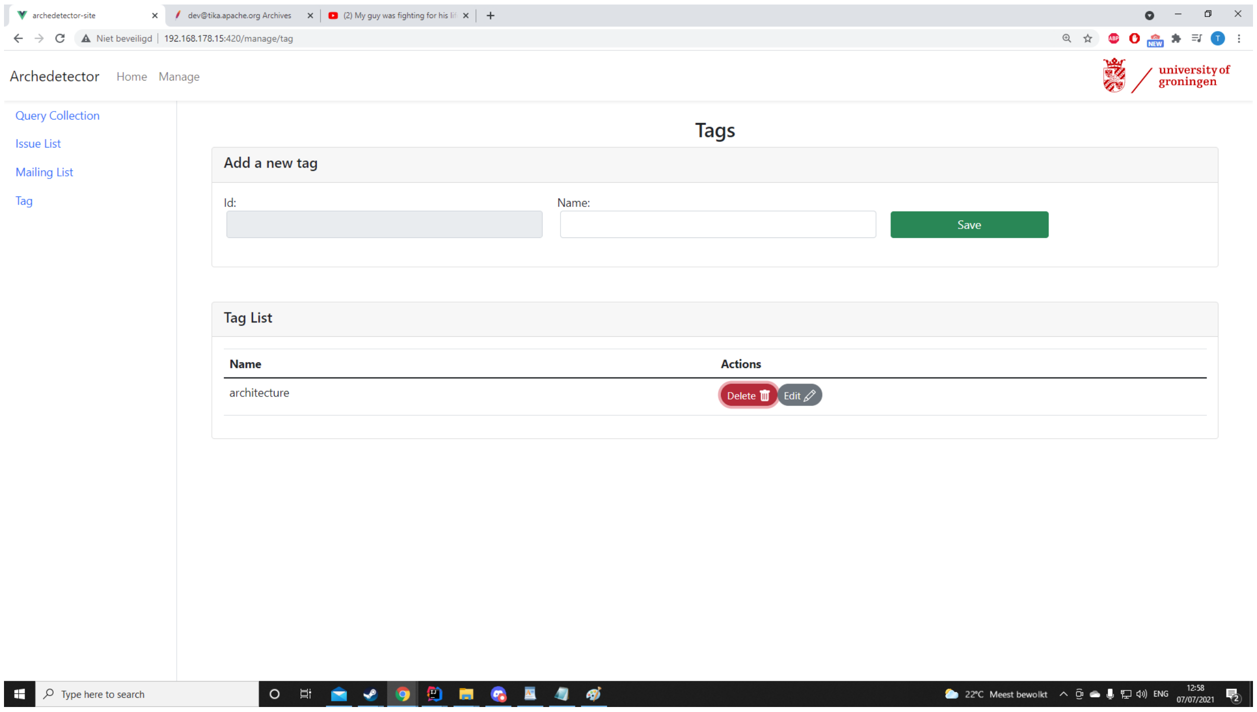
Id:

Name:

Tag List

Name	Actions

Add any tags that might be suitable for your use case. For me it would for example be architecture.



Archedetector Home Manage

Query Collection
Issue List
Mailing List
Tag

Tags

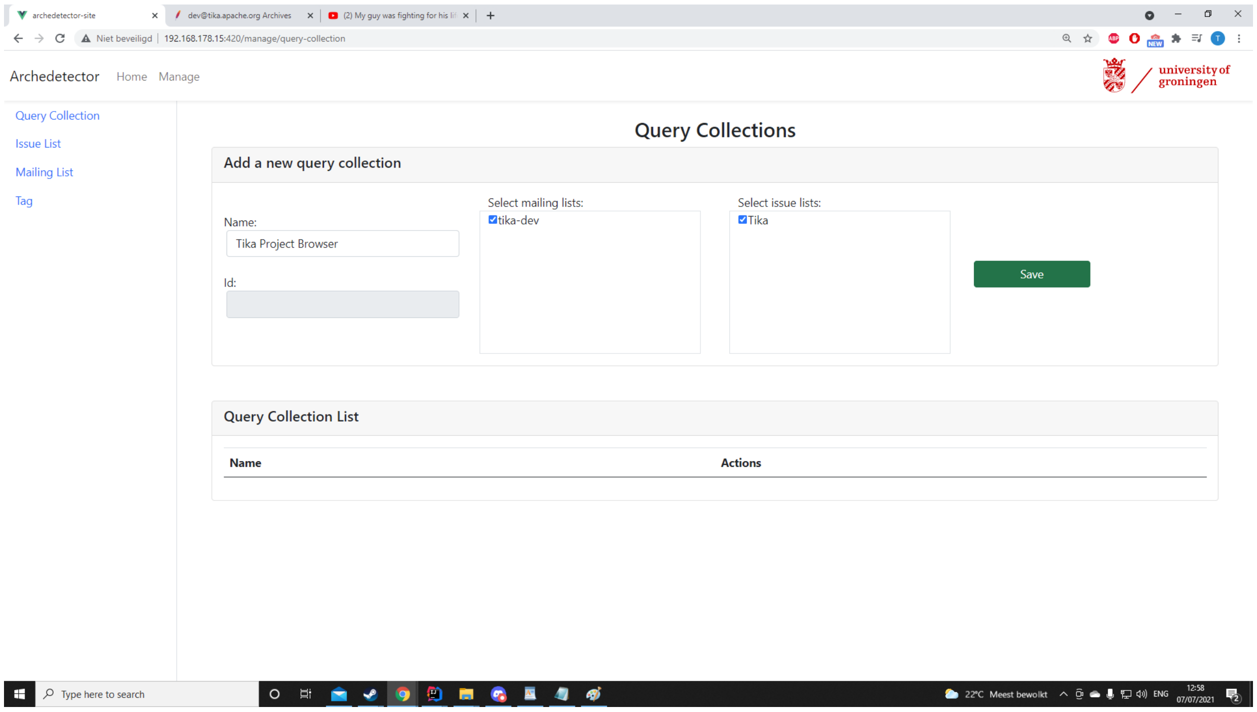
Add a new tag

Id:
Name:
Save

Tag List

Name	Actions
architecture	Delete Edit

Then finally navigate to the query collections tab and select the lists you want to browse and name it. Press save and then you should be ready to browse the lists.



Query Collections

Add a new query collection

Name: Tika Project Browser

Id:

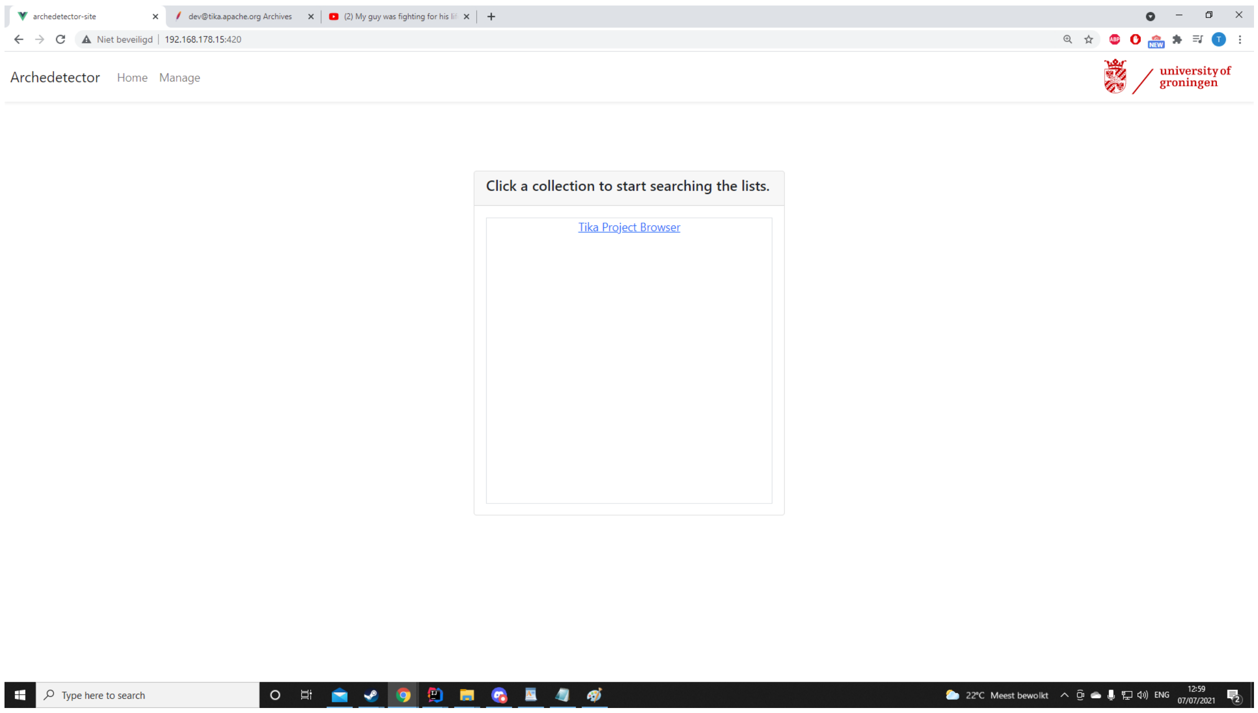
Select mailing lists: ☑tika-dev

Select issue lists: ☑Tika

Query Collection List

Name	Actions

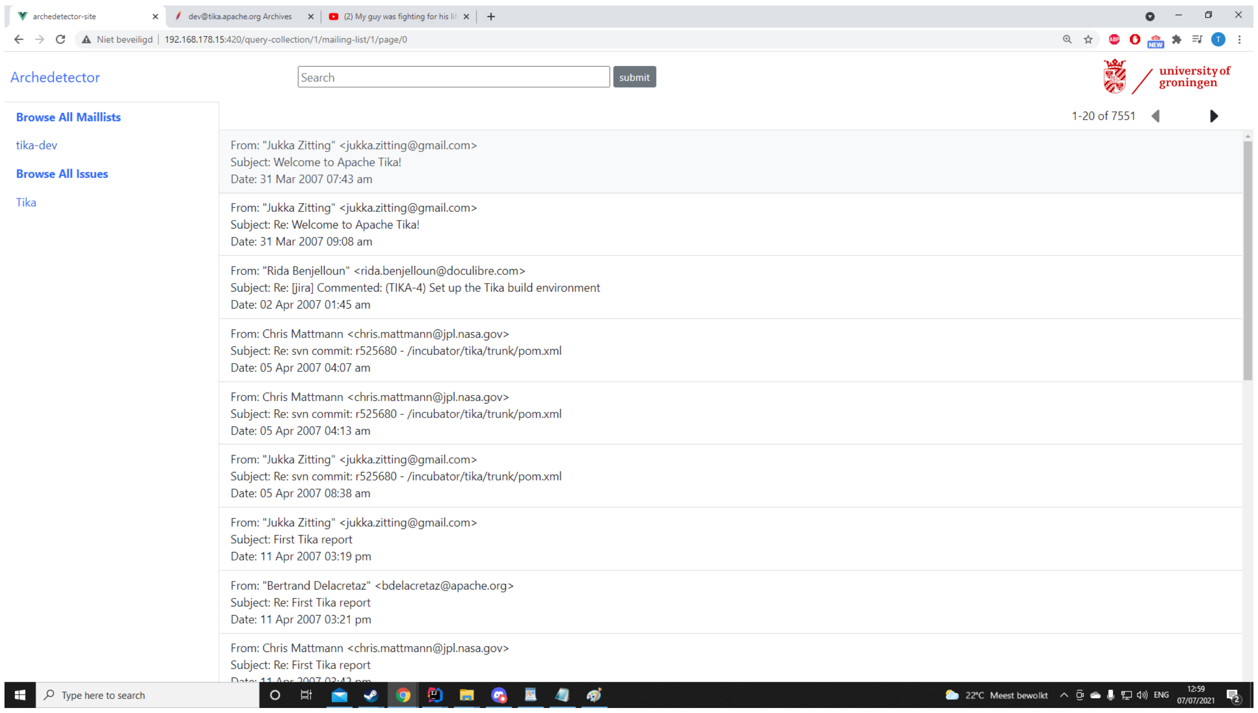
Navigate to home and select the collection you just created. In this case click on Tika Project Browser to navigate to the browsing page.



Click a collection to start searching the lists.

Tika Project Browser

Now you are ready to browse the lists! The arrows are to navigate the pages. The sidebar is to select which list you want to browse.


archedetector-site | dev@tika.apache.org Archives | (2) My guy was fighting for his li

Niet beveiligd | 192.168.178.15:420/query-collection/1/mailing-list/1/page/0

Archedetector

Search | submit

Browse All Maillists

tika-dev

Browse All Issues

Tika

1-20 of 7551

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Welcome to Apache Tika!
Date: 31 Mar 2007 07:43 am

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Re: Welcome to Apache Tika!
Date: 31 Mar 2007 09:08 am

From: "Rida Benjelloun" <rida.benjelloun@doculibre.com>
Subject: Re: [jira] Commented: (TIKA-4) Set up the Tika build environment
Date: 02 Apr 2007 01:45 am

From: Chris Mattmann <chris.mattmann@jpl.nasa.gov>
Subject: Re: svn commit: r525680 - /incubator/tika/trunk/pom.xml
Date: 05 Apr 2007 04:07 am

From: Chris Mattmann <chris.mattmann@jpl.nasa.gov>
Subject: Re: svn commit: r525680 - /incubator/tika/trunk/pom.xml
Date: 05 Apr 2007 04:13 am

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Re: svn commit: r525680 - /incubator/tika/trunk/pom.xml
Date: 05 Apr 2007 08:38 am

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: First Tika report
Date: 11 Apr 2007 03:19 pm

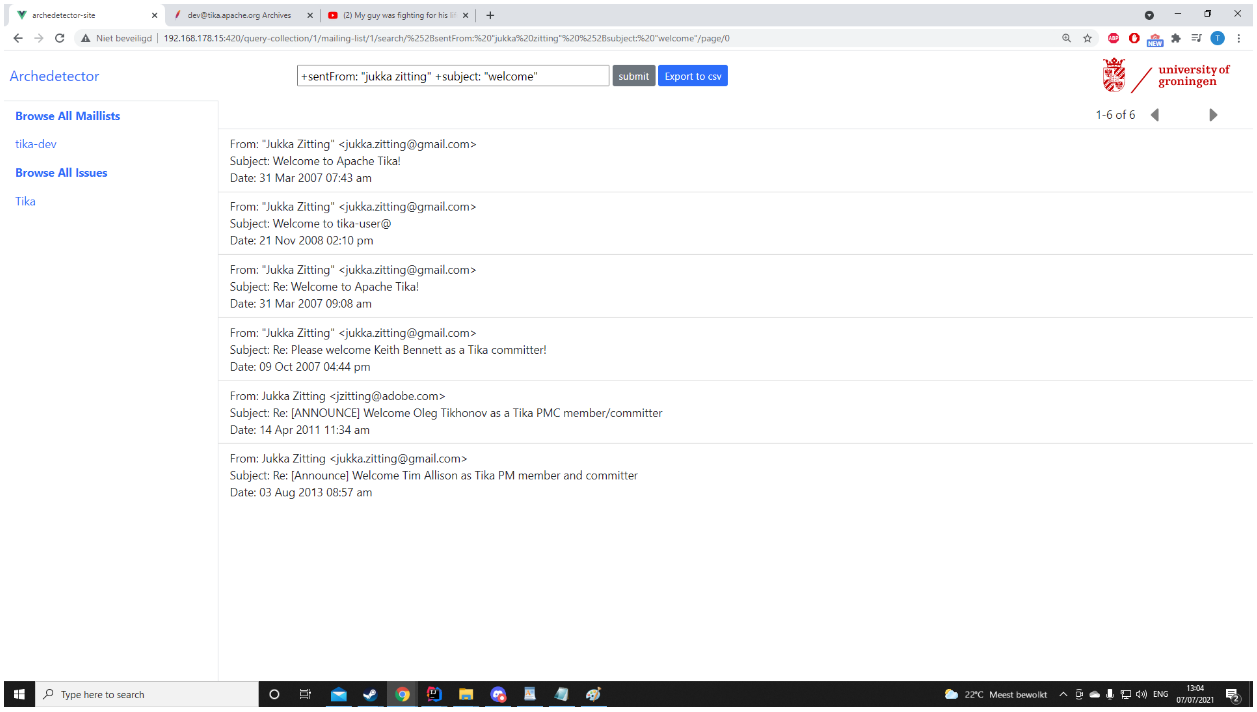
From: "Bertrand Delacretaz" <bdelacretaz@apache.org>
Subject: Re: First Tika report
Date: 11 Apr 2007 03:21 pm

From: Chris Mattmann <chris.mattmann@jpl.nasa.gov>
Subject: Re: First Tika report
Date: 11 Apr 2007 03:42 pm

Type here to search

22°C Meest bewolkt ENG 12:59 07/07/2021

You are able to search in the lists but only with the query parser syntax of lucene https://lucene.apache.org/core/2_9_4/queryparsersyntax.html.
You can also export your search results as a json file.



From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Welcome to Apache Tika!
Date: 31 Mar 2007 07:43 am

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Welcome to tika-user@
Date: 21 Nov 2008 02:10 pm

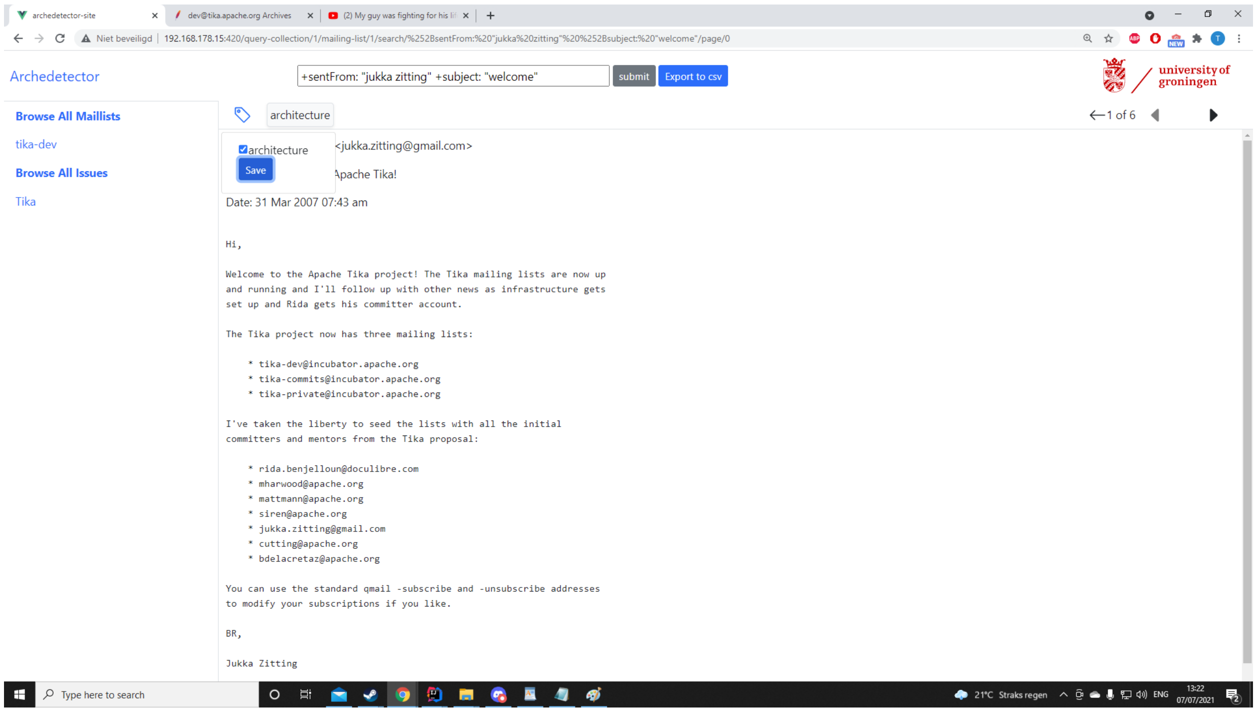
From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Re: Welcome to Apache Tika!
Date: 31 Mar 2007 09:08 am

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Re: Please welcome Keith Bennett as a Tika committer!
Date: 09 Oct 2007 04:44 pm

From: Jukka Zitting <jzitting@adobe.com>
Subject: Re: [ANNOUNCE] Welcome Oleg Tikhonov as a Tika PMC member/committer
Date: 14 Apr 2011 11:34 am

From: Jukka Zitting <jukka.zitting@gmail.com>
Subject: Re: [Announce] Welcome Tim Allison as Tika PM member and committer
Date: 03 Aug 2013 08:57 am

Finally you can tag the resources with the tags you created by clicking the tag icon when browsing in a single result by clicking on in the list browser. Do click save or it will not be stored in the database.


archedetector-site
dev@tika.apache.org Archives
(2) My guy was fighting for his li
Niet beveiligd
192.168.178.15:420/query-collection/1/mailing-list/1/search/%252BsentFrom:%20"jukka%20zitting"%20%252Bsubject:%20"welcome"/page/0
Archedetector
+sentFrom: "jukka zitting" +subject: "welcome"
submit
Export to csv
university of groningen
Browse All Maillists
tika-dev
Browse All Issues
Tika
architecture
1 of 6
architecture
Save
Type here to search
21°C Straks regen
ENG
13:22
07/07/2021

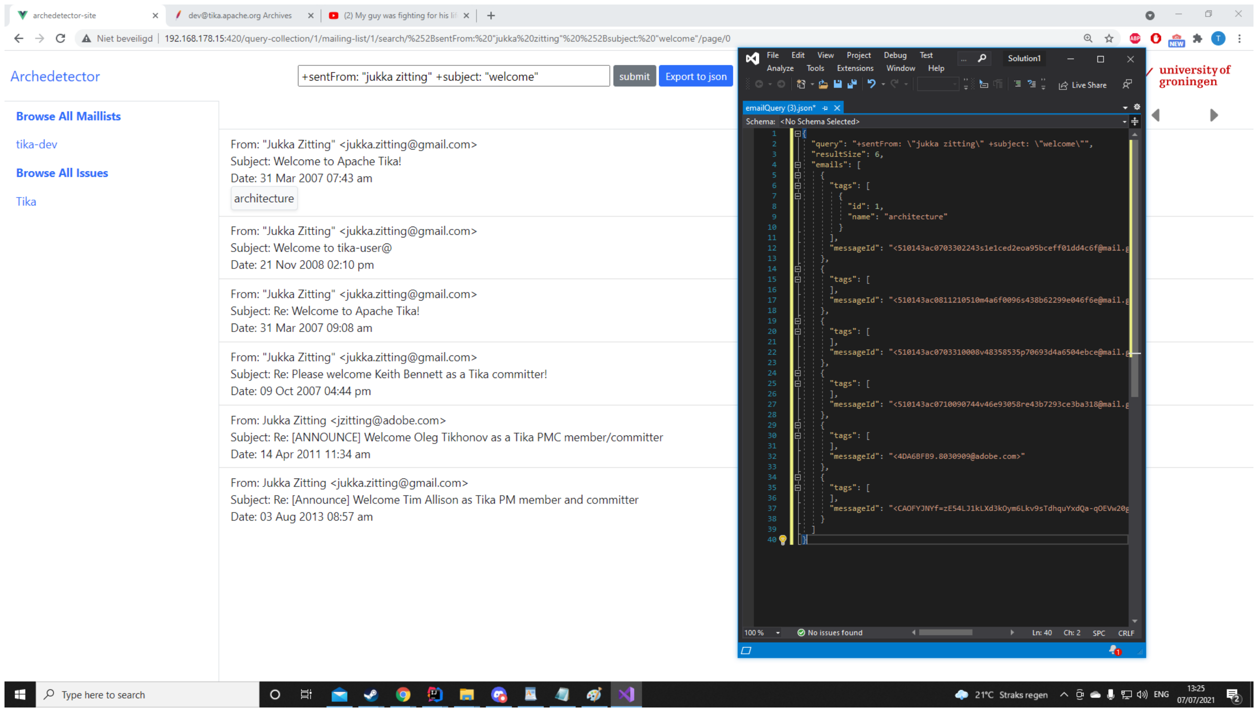
<jukka.zitting@gmail.com>

Apache Tika!

Date: 31 Mar 2007 07:43 am

```
Hi,

Welcome to the Apache Tika project! The Tika mailing lists are now up
and running and I'll follow up with other news as infrastructure gets
set up and Rida gets his committer account.

The Tika project now has three mailing lists:

    * tika-dev@incubator.apache.org
    * tika-commits@incubator.apache.org
    * tika-private@incubator.apache.org

I've taken the liberty to seed the lists with all the initial
committers and mentors from the Tika proposal:

    * rida.benjelloun@doculibre.com
    * mharwood@apache.org
    * mattmann@apache.org
    * siren@apache.org
    * jukka.zitting@gmail.com
    * cutting@apache.org
    * bdelacretaz@apache.org

You can use the standard qmail -subscribe and -unsubscribe addresses
to modify your subscriptions if you like.

BR,

Jukka Zitting
```

Example export with a tag.



192.168.178.15:420/query-collection/1/mailing-list/1/search/%252BsentFrom:%20"jukka%20zitting"%20%252Bsubject:%20"welcome"/page/0

Archedetector

+sentFrom: "jukka zitting" +subject: "welcome"

Browse All Maillists

tika-dev

Browse All Issues

Tika

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Welcome to Apache Tika!
Date: 31 Mar 2007 07:43 am
architecture

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Welcome to tika-user@
Date: 21 Nov 2008 02:10 pm

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Re: Welcome to Apache Tika!
Date: 31 Mar 2007 09:08 am

From: "Jukka Zitting" <jukka.zitting@gmail.com>
Subject: Re: Please welcome Keith Bennett as a Tika committer!
Date: 09 Oct 2007 04:44 pm

From: Jukka Zitting <jzitting@adobe.com>
Subject: Re: [ANNOUNCE] Welcome Oleg Tikhonov as a Tika PMC member/committer
Date: 14 Apr 2011 11:34 am

From: Jukka Zitting <jukka.zitting@gmail.com>
Subject: Re: [Announce] Welcome Tim Allison as Tika PM member and committer
Date: 03 Aug 2013 08:57 am

emailQuery (3).json*

```
{
  "query": "+sentFrom: \"jukka zitting\" +subject: \"welcome\"",
  "resultSize": 6,
  "emails": [
    {
      "tags": [
        {
          "id": 1,
          "name": "architecture"
        }
      ],
      "messageId": "<510143ac0703302243s1e1ced2eoa95bceff01dd4c6f@mail.g
    },
    {
      "tags": [
      ],
      "messageId": "<510143ac0811210510m4a6f0096s438b62299e046f6e@mail.g
    },
    {
      "tags": [
      ],
      "messageId": "<510143ac0703310008v48358535p70693d4a6504ebce@mail.g
    },
    {
      "tags": [
      ],
      "messageId": "<510143ac0710090744v46e93058re43b7293ce3ba318@mail.g
    },
    {
      "tags": [
      ],
      "messageId": "<4DA6BFB9.8030909@adobe.com>"
    },
    {
      "tags": [
      ],
      "messageId": "<CAOFYJNYf=zE54LJ1kLXd3kOym6Lkv9sTdhquYxdQa-qOEVw20g
    }
  ]
}
```

Those are all the functionalities as of now. Good luck using the tool!

11.2 Technical Guide

Archedetector site:

Requirements:

- Node.js

Run npm install in the project directory to get all the dependencies.
Run npm run serve to host the site on your localhost

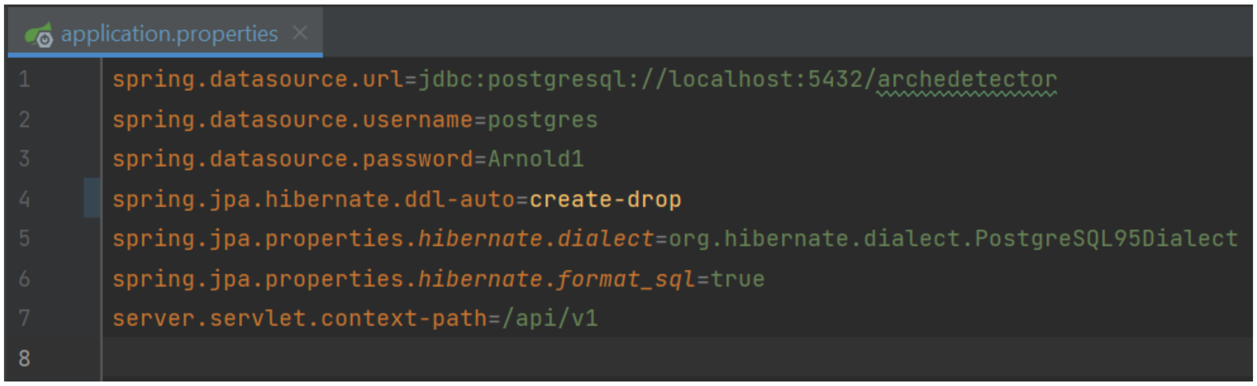
You are able to change api url If your api is at a different location than the one in the env.development file in the project root.

Archedetector api:

Requirements:

- Postgres
- Maven
- Java 16.0.1

Then you need to create an empty database.
https://www.tutorialspoint.com/postgresql/postgresql_create_database.htm
Finally you will need to edit the datasource fields(url, username, password) corresponding to your local properties.The application will create all the tables for you.
spring.jpa.hibernate.ddl-auto=create-drop, will drop and recreate the database on every time you run the application. So change this to update if you do not want your database to be wiped.



```
spring.datasource.url=jdbc:postgresql://localhost:5432/archedetector
spring.datasource.username=postgres
spring.datasource.password=Arnold1
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL95Dialect
spring.jpa.properties.hibernate.format_sql=true
server.servlet.context-path=/api/v1
```

Follow this stackoverflow to start up the application.
https://stackoverflow.com/questions/47835901/how-to-start-up-spring-boot-application-via-command-line

Add maven to your path variables and navigate to the project root of the api. Then you should be able to run the project with the command: mvn spring-boot:run