IT UNIVERSITY OF COPENHAGEN

Deliverable 5: (Individual) Architectural Recovery

Master of Science in Computer Science
IT-University of Copenhagen

Course Name: Software Architecture
Course Code: KSSOARC2KU
Submission Date: 11th May 2025

Author	**Email**
Rasmus Ole Routh Herskind	rher@itu.dk

Table of contents

1 Introduction

This report aims at doing an Architectural Recovery for the Zeeguu API Backend: *Zeeguu-API is an open API that allows tracking and modeling the progress of a learner in a foreign language, with the goal of recommending paths to accelerate vocabulary acquisition.*[1]
The report addresses the lack of architectural documentation in the system by reconstructing its structure using a module view, and conducting static analysis to identify key components, their relationships, and responsibilities.

2 Methodology

2.1 Tool support

To do the recovery, I have mainly used the Python scripts provided in the course, along with additional ones to provide different views of the architecture. Some of them I have modified a bit, but their goal stays the same: get hierarchical views of the modules in the source code and integrate new evaluation metrics as described below.

2.2 Data gathering and knowledge inference

The scripts all focus on statically analyzing the API modules by drawing dependency graphs between them. Each node represents a given module, with directed edges illustrating a dependency from one module to the other, which are found using simple regular expressions from the root content folder. To facilitate the creation of the graphs, the following metrics were used:

- Lines of Code (LoC). The bigger the node size, the higher the amount of LoC.
- Code Churn (lines of code modified across all git commits, based on the specific module view). This is illustrated by a viridis color map, going from purple (low churn) to yellow (high churn).
- Module hierarchy, by viewing module dependencies at different levels.

Using the general scripts provided in class and the above-mentioned metrics, I have retrieved the source view presented in figure 1.

As the view is rather difficult to get any meaningful information out of at first glance, I will, throughout the results section, section 3, dive deeper into module-based views to get a better understanding of the Zeeguu-API. This was an iterative process, hence only the last iterations are shown in the source code and this report.

[1]Mircea, 2025.

In general, I have extended the provided scripts with some helper functions for creating new graphs. Some of these functions include the functionality to abstract a provided graph to a given depth, filter the graph (e.g., removing specific nodes), or zoom into a specific node. From the source view presented in figure 1, I will present the top-level view of the system before iteratively exploring different modules while still preserving the graph-based module view.

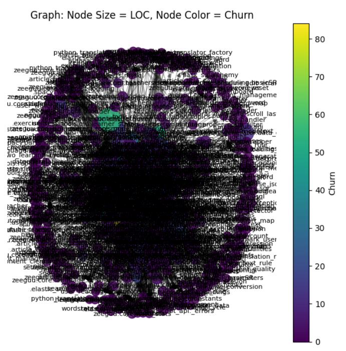
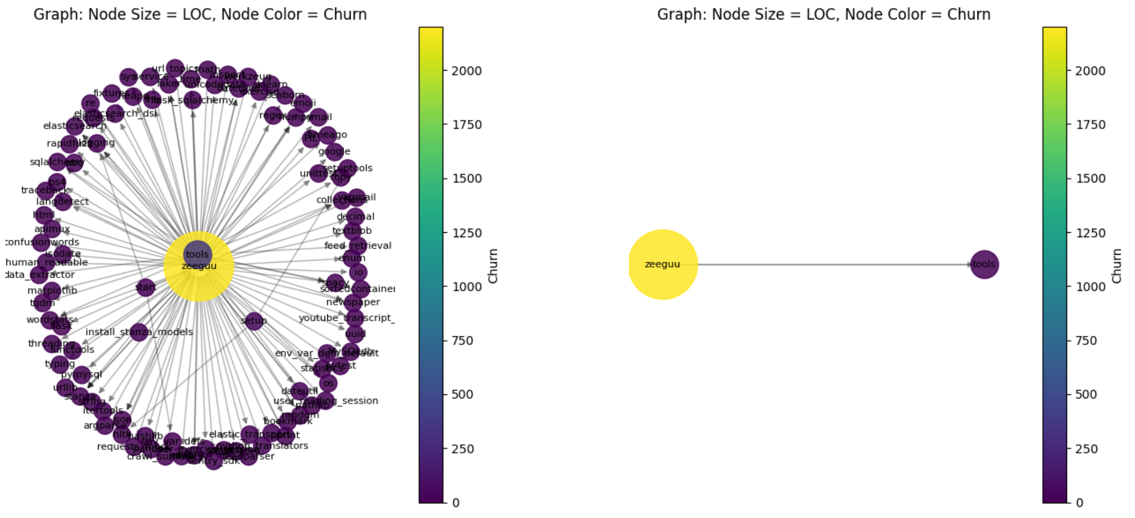


Figure 1: Source view of Zeeguu-API

3 Results

To get a more structured view on the system, I started by getting the top view of the system. However, it was quite clear that the system had a lot of external dependencies, which even further clutter the views. By filtering those out, as illustrated in figure 2b, we see that the system at the root level has two core components: *zeeguu* and *tools*:

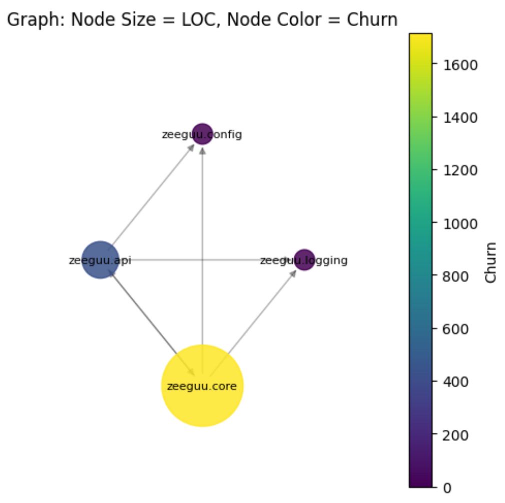


(a) Top view with external dependencies (b) Top view with only internal dependencies

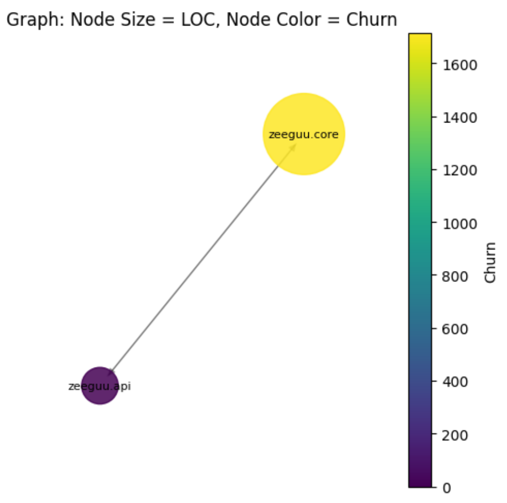
Figure 2: Top view of Zeeguu API

Given the *zeeguu* module scored high in terms of Churn and LoC, I looked further into it, as shown in figure 3, by expanding the depth level to 2. The module mainly consists of *zeeguu.core* and *zeeguu.api*. There are also some dependencies towards *zeeguu.config* and *zeeguu.logging*, but since they both have a low LoC and Churn, I filtered all the next graphs to only show relevant modules (which I defined to be modules with a specific LoC and Churn[2]).

[2]The values for each graph were found through trial and error. Other values could lead to different results, which is also discussed in section 4



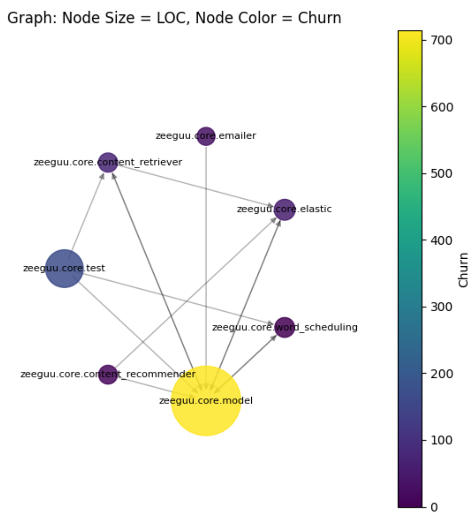
(a) All modules



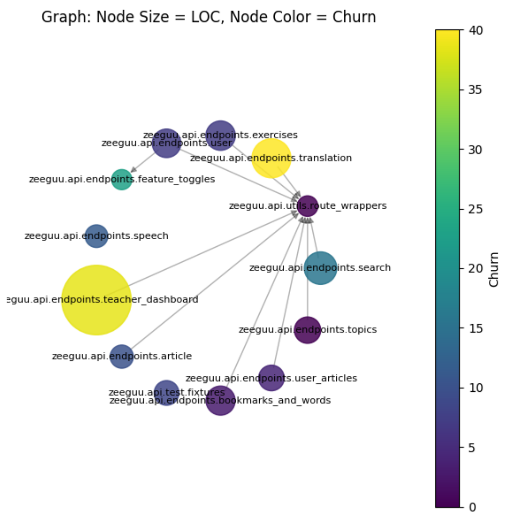
(b) Modules with $Churn > 50$ and $LoC > 50$

Figure 3: Zeeguu module at depth 2

Looking even deeper into the relevant *zeeguu.core* and *zeeguu.api* modules at depth 3 and 4 respectively, we can see how modules with either high churn or a reasonably high amount of lines of code depend on each other in the two modules:


Graph: Node Size = LOC, Node Color = Churn
zeeguu.core.emailer
zeeguu.core.content_retriever
zeeguu.core.elastic
zeeguu.core.test
zeeguu.core.word_scheduling
zeeguu.core.content_recommender
zeeguu.core.model
Churn
0
100
200
300
400
500
600
700

(a) Zeeguu.core module at depth 3 with $Churn > 50$ and $LoC > 50$


Graph: Node Size = LOC, Node Color = Churn
zeeguu.api.endpoints.exercises
zeeguu.api.endpoints.user
zeeguu.api.endpoints.translation
zeeguu.api.endpoints.feature_toggles
zeeguu.api.utils.route_wrappers
zeeguu.api.endpoints.speech
zeeguu.api.endpoints.search
eguu.api.endpoints.teacher_dashboard
zeeguu.api.endpoints.topics
zeeguu.api.endpoints.article
zeeguu.api.endpoints.user_articles
zeeguu.api.test.fixtures
zeeguu.api.endpoints.bookmarks_and_words
Churn
0
5
10
15
20
25
30
35
40

(b) Zeeguu.api module at depth 4 with $Churn > 10$ and $LoC > 50$

Figure 4: Zeeguu core and API modules

Interesting to note is how the API modules rarely depend on each other internally, which is in vast contrast to the core module. Here, a lot of the modules show internal dependencies, indicating a strong coupling between them. Furthermore, the churn in the core module—especially the *zeeguu.core.model* and *zeeguu.core.test* modules—is much higher than the highest churn in the API module. This clearly indicates that an even more fine-grained look at these two core modules could be relevant.

Lastly, I present a view showing how the API and core modules relate to one another. Here, we see that most underlying modules in the API and core depend on each other, which clearly opens a path for future work considerations.

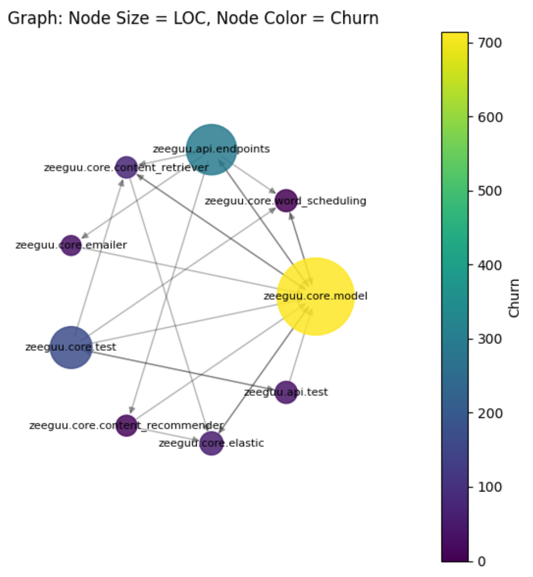


Figure 5: Zeeguu module at depth 3 with $Churn > 50$ and $LoC > 50$

4 Discussion and Conclusion

The main limitation of this project has also been its strength: the metrics used. I have, throughout the script-writing, filtered out specific modules solely based on lines of code and churn of the specific files. Some files and/or even modules could be relevant despite not scoring high in these metrics. I did try many different values of both churn and LoC, but the ones I have chosen for this report, I generally found to be the best. It could be interesting to also have included more metrics, for example a minimal amount of imports.

Given more time, I would have liked to create some more visually appealing diagrams showcasing the views, e.g., UML diagrams. This would help newcomers to the Zeeguu project understand the project and its components more easily, instead of being shown relatively abstract graphs.
It would also be interesting to dive even further into more modules, e.g., the *zeeguu.core.model* module. The complexity, both from LoC and churn, is quite high, which probably also means

rather complex code and potentially room for improvement. This is generally true for the entire project, which is also a clear limitation of my method, given I only went as deep as level 3/4.

4.1 Conclusion

In the report, I have shown a high-level view of the Zeeguu-API backend through inspecting the module hierarchy, lines of code, and code churn. The project mainly consists of two modules from the content root: zeeguu and tools. Given that zeeguu had both a higher number of lines of code and churn, I inspected it even further. This showed rather tight couplings between its API and core module, which would be interesting to look even further into.

A Link to code

All code used during the iterative recovery progress has been written using Google Colab and can be viewed in this notebook.

B Time Allocation

Most of my time was spent writing, testing and iterating thought the scripts on Google Colab. Throughout the process I wrote the report with new discoveries. In total I have spent aroung 60-70% of my time writing scripts, 5-10% on anylysing the results and 20-30% writing the report.

References

Mircea, L. (2025). Zeeguu api [Accessed: 2025-05-11].