Architectural Reconstruction of Zeeguu Api

Thor Liam Møller Clausen (tcla)

Introduction

What is the system

I chose to analyse Zeeguu's api repository[1]. It is the backend for their website[2] and is written in python flask. It is a website where users can learn languages similar to Duolingo. More information can be found in the repository's *README* file and on the website.

What is the problem

The Zeeguu api application has over a hundred files. This can be overwhelming for people when they are introduced to the system. The goal of this project is to use abstractions and visualizations to get a better understanding of the general structure of the program, where the different logics are implemented and which parts of the system that are more important in terms of dependency.

Methodology

I want to analyse the python files in the codebase for architectural reconstruction. I want to get an overview of which modules depend on each other, and which are more "central" than others. By that I mean, it has more modules depending on it. This should be extracted as a module view.

Tools

I have chosen to find and use already implemented tools for analysing files and python code. The tools I ended up using are GitTruck[3] and Pyan3[4].

GitTruck

GitTruck gives an overview of files and folders, as well as commit history, file size etc. It works by running the following command which will open a window:

```
# from repository's root folder
npx -y git-truck
```

From here there are several settings to choose from. The goal was to get an initial overview of the file structure of the Zeeguu application. The settings I used are in appendix *A GitTruck settings*. Furthermore, I used GitTruck for file specific statistics like file sizes, commits per file etc.

[1] https://github.com/zeeguu/api
[2] https://zeeguu.org/
[3] https://github.com/git-truck/git-truck
[4] https://github.com/Technologicat/pyan

Pyan3

Pyan3 is a static call graph generator made specifically for python. It can be installed with the following command:

```
pip install pyan3==1.1.1
```

(The newest version had a bug that made it unusable. Hence why I use version 1.1.1)

It uses graphviz[5] which I had to install from their website for it to work.

To generate the graph, I ran the following command from the *Zeeguu* folder:

```
pyan3 *.py */*.py */*/*.py */*/*/*.py -c -G -g -d --dot-rankdir LR --svg > ../../../output/graph.svg
```

The tool does not support recursive wildcard search[6], so I used python paths like **.py* and **/*.py* in the command with enough subfolders to cover all files. The arguments are explained in appendix *B Pyan3 arguments explained*.

The goal was to create a graph that made it easy to read which files and methods/functions call each other. I tried all the different arguments the tool had and ended with these because it best fulfilled the goal.

[5] https://graphviz.org/

[6] The path ***/*.py* can sometimes be used to recursively search subfolders for python files.

Results

GitTruck overview

GitTruck was very useful for getting an overview of how files were arranged.

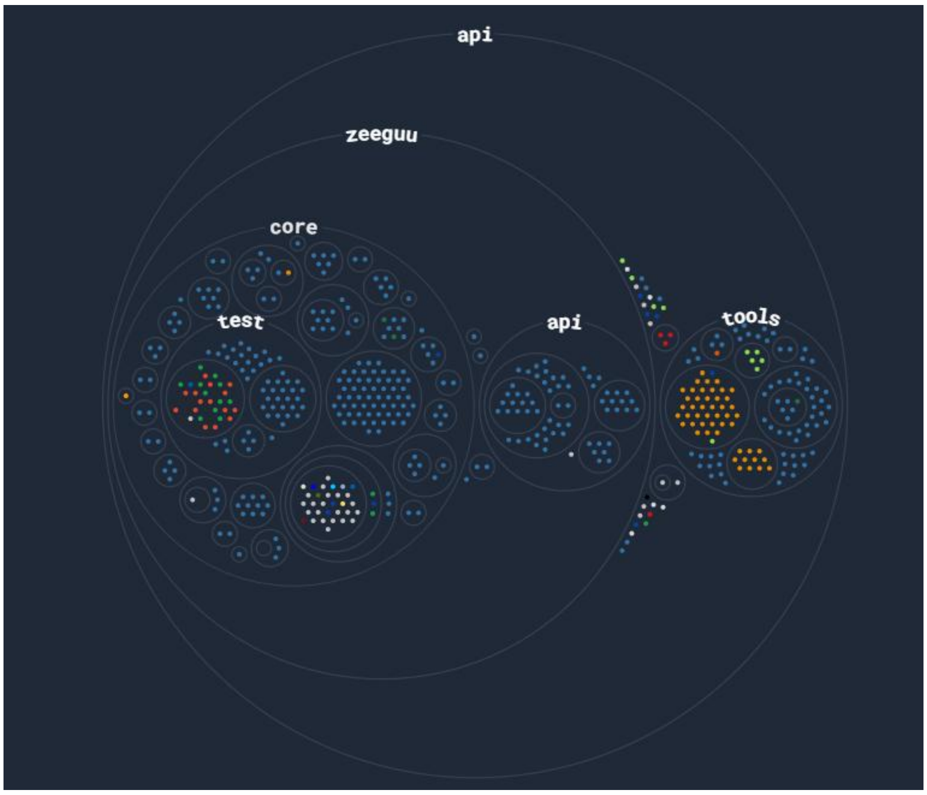


Figure 1 GitTruck bubble chart of Zeeguu's api repository.

After browsing the chart in Figure 1, I made an abstraction with the most important parts of the file structure for my own overview. It can be found in the appendix *C Boiled down file structure of Zeeguu*.

Pyan3

The initial graph it generated was huge and messy with overlapping nodes, which meant some filtering and abstraction was necessary. Based on the overview I got from GitTruck I decided not to include the *tools* folder and focused on the *Zeeguu* folder since it seemed most of the application logic was located there, while the *tools* folder was more about database connections, shell scripts, report generators etc. I also abstracted the test folders away.

Areas of interest

The graph was still too big to fit on a screen, so I browsed it and identified some areas of interest based on incoming calls.

Pyan3 call graph legend
Grey box: File/class containing functions/methods
Coloured node: Function/method/class
White node: Folder/file
Solid line arrows: Arrow from A to B means the method/function A calls method/function B.
Dotted line arrows: Arrow from A to B means A defines B. (Class/function/method with the ***def*** keyword has an outgoing defines arrow)

A module in *core* with a lot of incoming call dependencies (Figure 2):

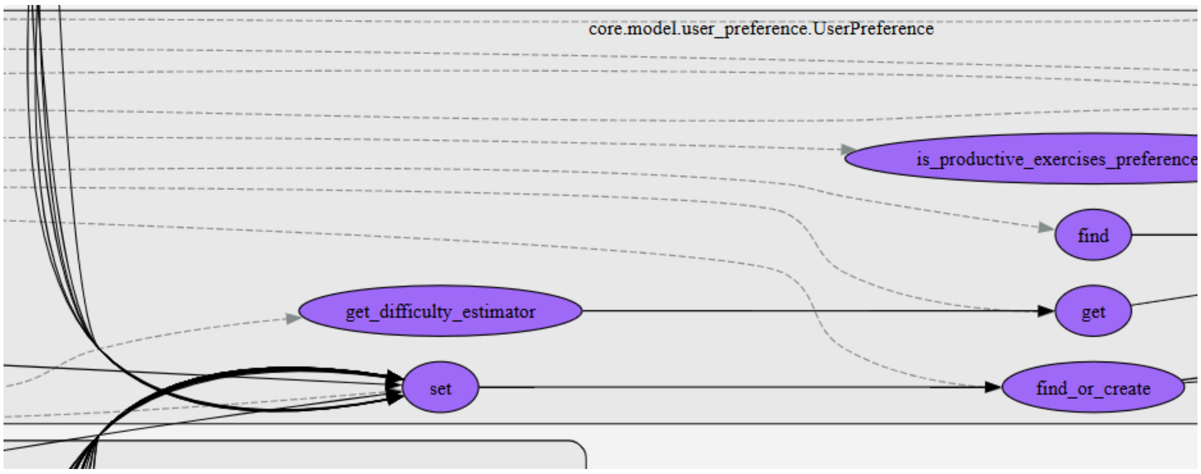

core.model.user_preference.UserPreference
is_productive_exercises_preference
find
get_difficulty_estimator
get
set
find_or_create

Figure 2 UserPreference "set" method from pyan3 call graph.

A module in *api* with a lot of incoming call dependencies (Figure 3):

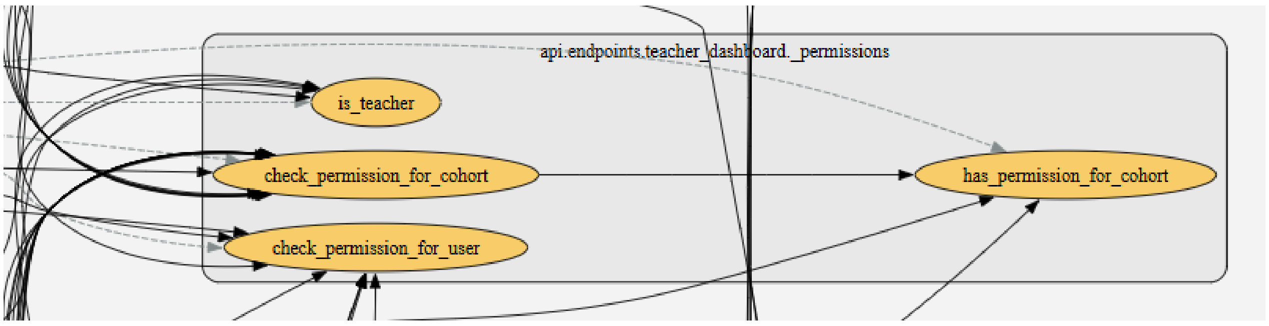

api.endpoints.teacher_dashboard._permissions
is_teacher
check_permission_for_cohort
has_permission_for_cohort
check_permission_for_user

Figure 3 Functions in the permissions file from pyan3 call graph.

Module view based on Pyan3 call graph

For further abstraction I decided to make my own smaller graphs with the important highlights. I took the most used functions/methods based on incoming calls and traced the callers to extract the dependencies. They can be seen in Figure 4 and Figure 5.

Module view legend
Box (module): File with file name and file size in kilobytes.
Arrow (dependency): Arrow from A to B means A has a call to B.
Arrow thickness shows the number of functions from A calling functions from B also denoted by the edge weight (no weight means weight=1).

Core files

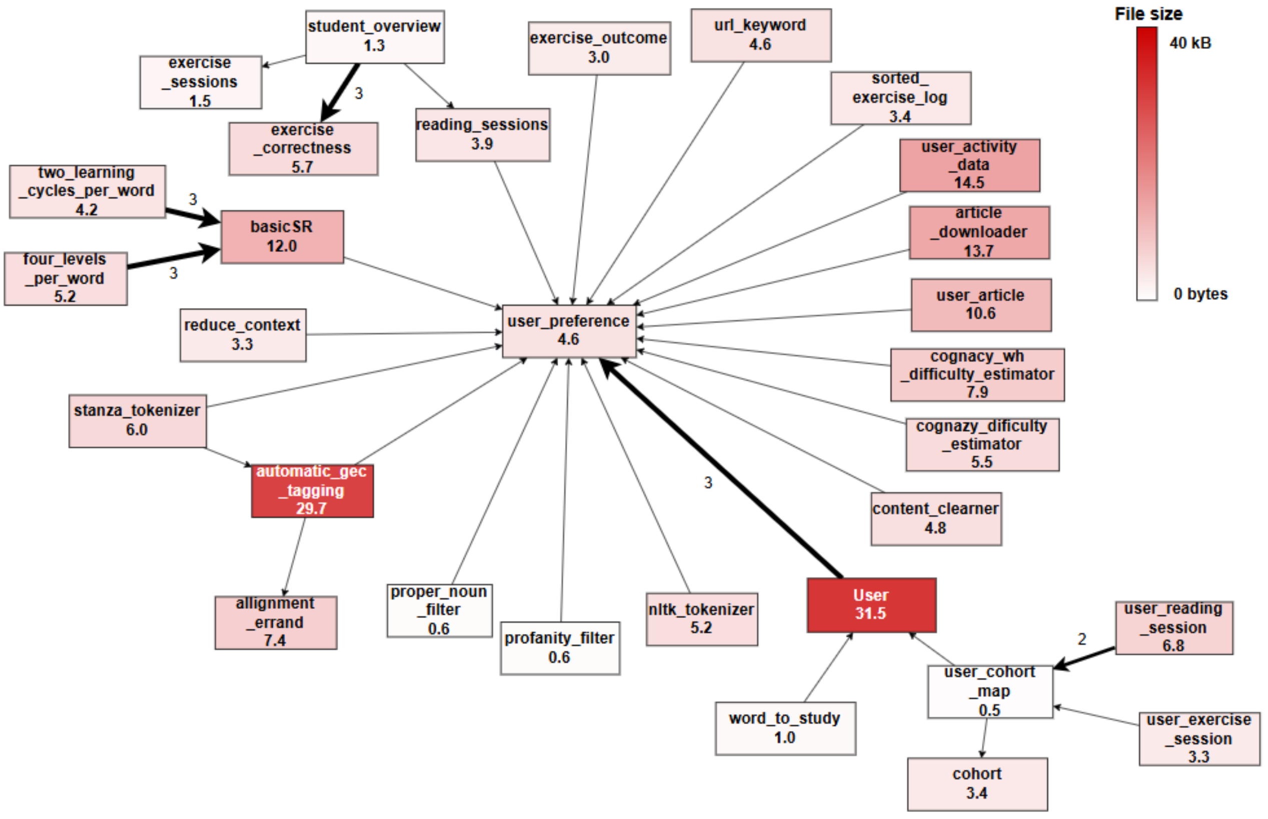


Figure 4 Module view for the files located in the core folder. File sizes were read from GitTruck.

Api files

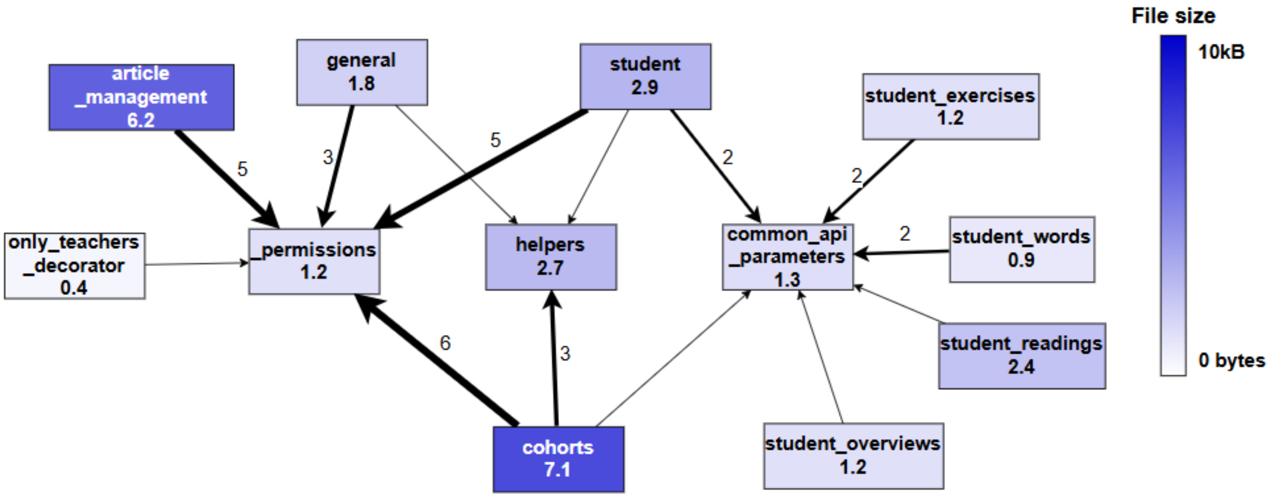


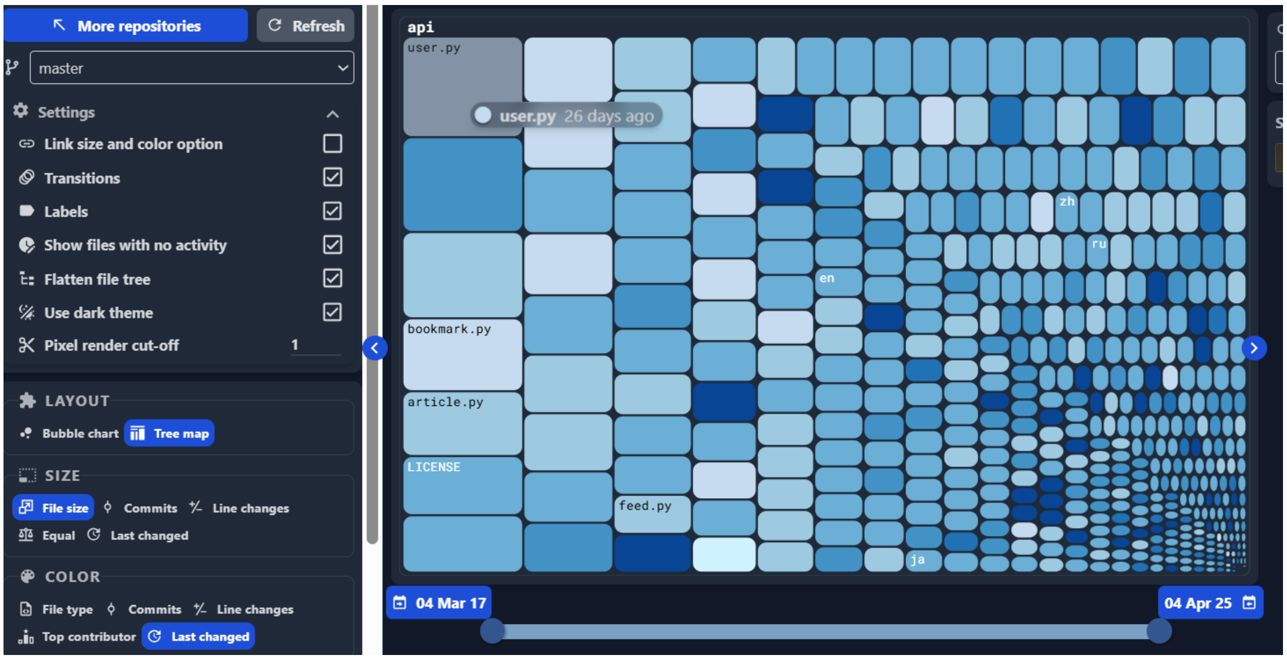
Figure 5 Module view for the files located in the api folder. File sizes were read from GitTruck.

Pyan3 conclusion

After browsing the pyan3 graph it became clear that the app has a lot of logic centralized around users in the core. The api mostly showed calls to helper functions and *_permissions*. The graph does not say much about the files other than the static function/method calls, so I decided to analyse some of them further on GitTruck.

GitTruck file analysis

In GitTruck I hid some of the non-python files, and it turns out the *user.py* file in the *core* folder is the largest python file in the repository, and it was last edited just 26 days ago (from April 23. 2025), which if you order files by last changed puts it in 15th place out of over a hundred files. This can be seen in Figure 6.



master

Settings

- Link size and color option ☐
- Transitions ✓
- Labels ✓
- Show files with no activity ✓
- Flatten file tree ✓
- Use dark theme ✓
- Pixel render cut-off 1

LAYOUT

Bubble chart [✓ Tree map]

SIZE

[✓ File size] Commits Line changes Equal Last changed

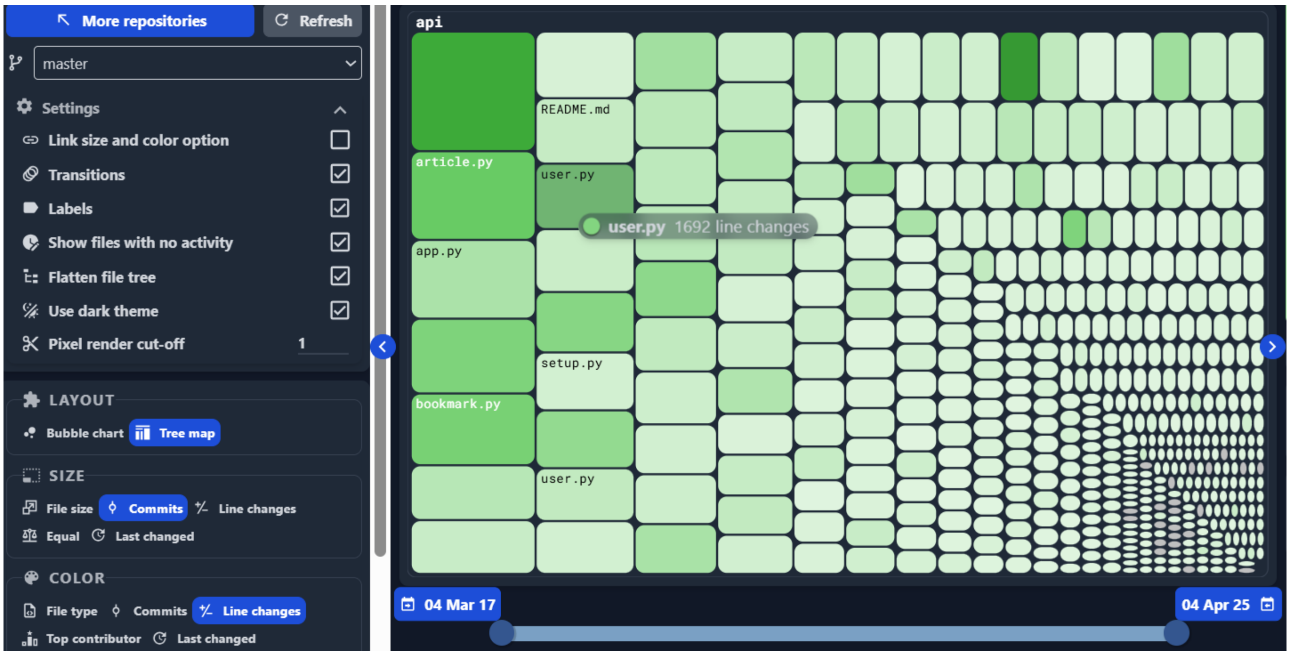
COLOR

File type Commits Line changes Top contributor [✓ Last changed]

04 Mar 17 — 04 Apr 25

Figure 6 Zeeguu api files ordered by file size and coloured by last changed using GitTruck.

Furthermore, the *user.py* file has 52 commits, which puts it in 10th place of files with most commits as seen in Figure 7. The *user_preference.py* file was further down on all GitTruck metrics and was for instance edited 96 days ago.



master

Settings
- Link size and color option ☐
- Transitions ☑
- Labels ☑
- Show files with no activity ☑
- Flatten file tree ☑
- Use dark theme ☑
- Pixel render cut-off 1

LAYOUT
Bubble chart [✓ Tree map]

SIZE
File size [✓ Commits] Line changes Equal Last changed

COLOR
File type Commits [✓ Line changes] Top contributor Last changed

Figure 7 Zeeguu api files ordered by commits and coloured by line changes using GitTruck.

The file with most commits is *language.py* in the *endpoints* folder which is also in top three with the other metrics except file size. However, the pyan3 graph does not show any dependencies for the file as seen in Figure 8.

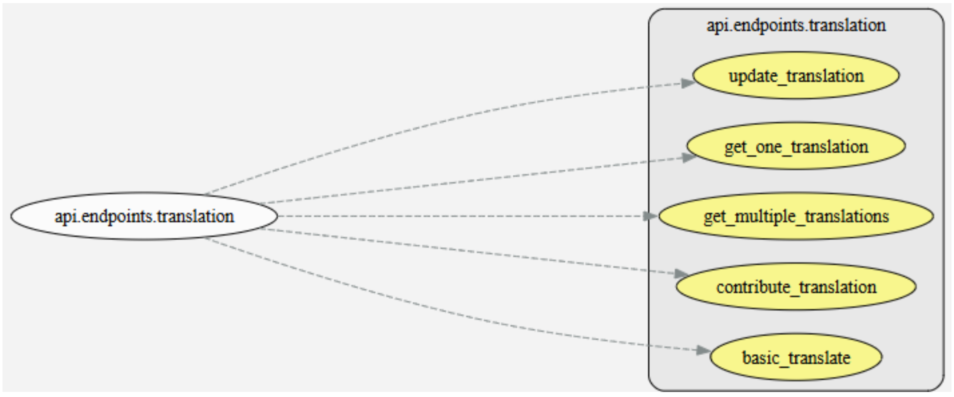

api.endpoints.translation
update_translation
get_one_translation
get_multiple_translations
contribute_translation
basic_translate
api.endpoints.translation

Figure 8 The translation.py file from pyan3 call graph.

Discussion

Tool flaws and limitations

The pyan3 tool had some major flaws that affects the usefulness of the results.

Fails to detect some method/function calls

```
@api.route("/user_preferences", methods=["GET"])
@cross_domain
@requires_session
def user_preferences():
    preferences = {}
    user = User.find_by_id(flask.g.user_id)
    for each in UserPreference.all_for_user(user):
        preferences[each.key] = each.value
```

Figure 9 Code section from user_preferences file in endpoints folder.

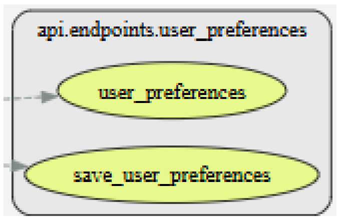


Figure 10 The user_preference file in the pyan3 call graph.

As seen in Figure 9 the ***user_preferences*** function calls the ***all_for_user*** function. But as seen in Figure 10 the pyan3 graph has no arrow from the ***user_preference*** node to ***all_for_user***. In general, there are no edges between files in the *api* folder and the *core* folder. This has led to a graph with a lot of disconnected subgraphs, which forced me to focus on a small part of the core and the api separately, and even here it is uncertain if dependencies are missing.

Limited customization and filtering

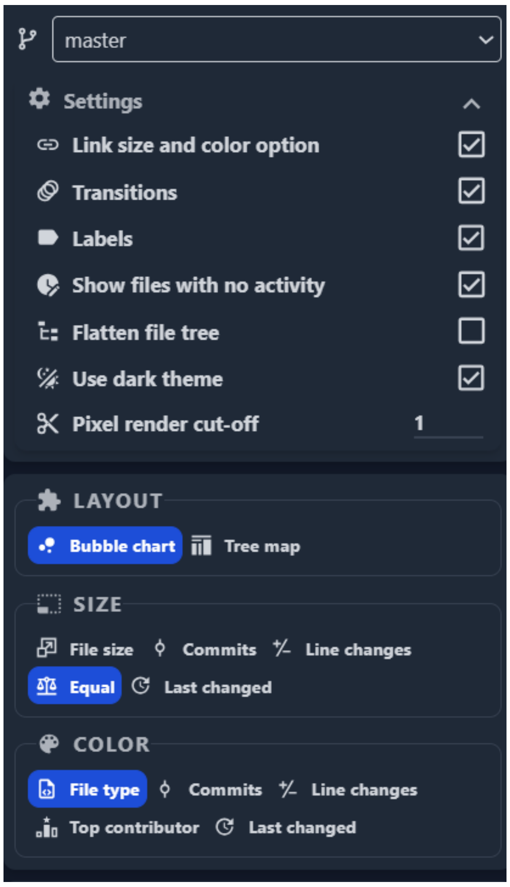
The pyan3 tool takes paths as arguments which it analyses, but it does not support exclusion of folders or files. For example, I had to delete the *tests* folder to avoid including it. It was also not possible to edit the graph with the tool, like removing nodes without in or outgoing edges or rearranging nodes to better fit them in a frame.

Improvements if I had more time

The results helped me get an overall idea of what the Zeeguu app contains in terms of functionality. However, pyan3's major flaws made it hard to tell how much important information was missing. The lack of freedom for graph customization and filtering also lead to a lot of manual work. For future analysis I would spend more time finding better tools with more filtering and customization options to avoid a lot of the manual work that went into boiling down a huge graph.

Appendix

A GitTruck settings



master

Settings

- Link size and color option ☑
- Transitions ☑
- Labels ☑
- Show files with no activity ☑
- Flatten file tree ☐
- Use dark theme ☑
- Pixel render cut-off 1

LAYOUT

[✓ Bubble chart] Tree map

SIZE

File size Commits Line changes [✓ Equal] Last changed

COLOR

[✓ File type] Commits Line changes Top contributor Last changed

Figure 11 GitTruck settings used for browsing the folder/file structure of Zeeguu's api.

B Pyan3 arguments explained

Taken from the output of running ***pyan3 -h***:

-c, --colored *color nodes according to namespace*

-G, --grouped-alt *suggest grouping by adding invisible defines edges*

-g, --grouped *group nodes (create subgraphs) according to namespace*

--dot-rankdir RANKDIR *specifies the dot graph 'rankdir' property for controlling the direction of the graph. Allowed values: ['TB', 'LR', 'BT', 'RL']*

C Boiled down file structure of Zeeguu

```
.
├───Tools
│   ├───Sql
│   ├───Migrations
│   ├───Report generator
│   ├───Shell scripts
│   ├───Mail sender
│   └───etc.
├───Workflows
│   ├───Test
│   ├───Docker
│   ├───Shell scripts
│   ├───Mail sender
│   └───etc.
└───Zeeguu
    ├───Api
    │   ├───Endpoints
    │   │   ├───Helpers
    │   │   └───Teacher dashboard
    │   ├───Tests
    │   └───Utility
    │
    └───Core
        ├───Language
        ├───Sql
        ├───User stats
        ├───Emailer
        ├───Account manager
        ├───Test
        ├───Model
        ├───…
        └───Word filter (proper nouns, profanity and city/person names)
            └───Data (list of naughty words and readme with source)
```

Figure 12 Abstracted file/folder structure for the Zeeguu api repository.

D Time allocation

I spent a significant amount of time trying out different tools that I could not get to work, or did not give me a result that I could use (for example *prospector*[7] that turned out to be more like a linter).

pycallgraph[8] and *pyreverse*[9] gave the same error:
AttributeError: 'TreeRebuilder' object has no attribute 'visit_typealias'.

After approximately two working days I had two tools that worked, pyan3 and GitTruck. Then I spent most of a working day on playing around with these until I felt I had a good overview of the Zeeguu

[7] https://prospector.landscape.io/en/master/
[8] https://pycallgraph.readthedocs.io/en/master/
[9] https://pylint.readthedocs.io/en/latest/additional_tools/pyreverse/index.html

system. I spent another two working days putting my mind on paper - Creating abstractions for file structure and the module view as well as writing down notes and collecting it in a document for the report.