ETH zürich

Department of Computer Science
Algorithms and Didactics

cover image generated with DALLE/ChatGPT

Centralized documentation for TigerJython and WebTigerPython

Andri Hunkeler

Bachelor's Thesis

2024

Supervisors:	Prof. Dr. Dennis Komm Alexandra Maximova Clemens Bachmann

Abstract

This thesis is about the development of a centralized documentation for TigerJython and WebtigerPython, a solution to streamline the documentation which currently exists in multiple different versions. The main goal was to build a tool which allows for the easy distribution of the information, as well as making maintenance and additions to it simpler.

To make this happen I used a MySQL database which is made accessible through FastAPI, a Python based API framework. The user interface is written in TypeScript, utilizing Vue as the base JavaScript framework and extending its capabilities by adding Vuetify and Vueform.

Some of the key features of the documentation include multi-language support, flexible sorting of all parts of the information, tags for fine grained categorization and aids to keep the different languages consistent.

In the end, the solution provides a consistent documentation for all users and a streamlined process for maintaining it. It can be integrated in the currently existing versions of the documentation, as well as future endeavors such as the integrated documentation in WebTigerPython.

Acknowledgment

First and foremost, I would like to thank Prof. Dr. Dennis Komm for giving me the opportunity to conduct my thesis in his research group.

I want to thank Alexandra Maximova and Clemens Bachmann for supervising my project and their valuable support throughout the whole working process. Further, I would like to thank Andre Macejko for his help with the deployment and setup of the backend.

And of course I also want to thank my friends and family for their support.

Contents

1 Introduction **1**

1.1 Existing Documentations . . . 1

1.2 Contribution . . . 3

2 Related Work **5**

2.1 Design Inspiration . . . 5

2.2 Frameworks . . . 5

3 Design **7**

3.1 Workflow Requirements . . . 7

3.2 Hierarchy and Documentation Ordering . . . 8

3.3 Function Documentation . . . 9

3.4 Translation . . . 10

3.5 Small Changes . . . 10

3.6 User Interface . . . 11

4 Implementation **13**

4.1 Database . . . 13

Hierarchy . . . 14

Views . . . 14

4.2 API . . . 16

Open Code Snippets in WebTigerPython . . . 16

4.3 User Interface . . . 16

User Interface Language Changes . . . 16

Selection Dropdowns . . . 17

Translation . . . 18

Status Indication . . . 18

4.4 Security . . . 18

SQL-Injection Prevention . . . 18

Authentication . . . 20

5 Conclusion **21**

5.1 Future Work . . . 21

Evaluation . . . 21

Data Migration . . . 21

Integration in WebTigerPython . . . 22

Integration in TigerJython4Kids, Python-Online and WebTigerJython 22

Integration in TigerJython 22

Chapter 1

Introduction

The goal of my thesis was to create a centralized documentation for TigerJython [10] and WebTigerPython [3] that is maintainable and up to date. The need for it arose as there are multiple different versions that currently exist as well as plans to make the documentation directly accessible in WebTigerPython. While there were existing documentations [2, 9, 10, 16], they were not suitable for the intended use case.

Integrating the centralized documentation into WebTigerPython would benefit novices, as well as more experienced users, by providing them with up to date information next to where they are practicing their programming skills. Thereby reducing barriers and improving the learning experience of the students.

1.1 Existing Documentations

There exist multiple different versions of the documentation, such as Python-Online [9], TigerJython4Kids [2], WebTigerJython [16] and the locally installed TigerJython [10].

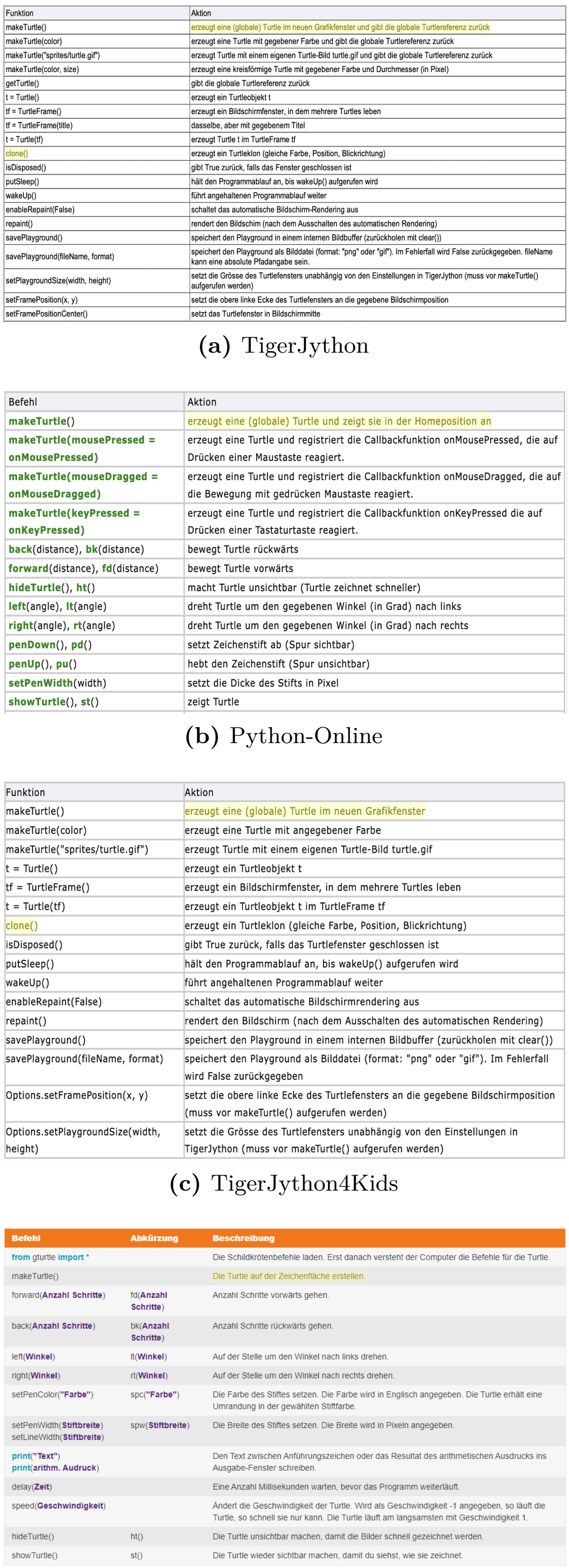
Comparing these versions, one can find that they differ in the function scope and function descriptions. These differences can either be attributed to functions not being supported by a version, the functions behaving slightly differently in a version, the documentation not being up to date or simply being written by a different person. Here are a few examples which can be seen in Figure 1.1:

- The function `clone()` does not exist in the Python-Online and WebTigerJython documentations,
- `makeTurtle()`, which is the first function in all four, has four different descriptions.

Only TigerJython and WebTigerJython support other languages than German.

Another notable difference is that Python-Online offers hyperlinks (green) which open WebTigerPython with the corresponding code snippet.

Apart from that, the Python-Online and TigerJython4Kids documentations exist as hard coded HTML elements which makes maintaining them a tedious process. It also prevents us from easily accessing this information to display it somewhere else unless we consider scraping the website content.



Funktion	Aktion
makeTurtle()	erzeugt eine (globale) Turtle im neuen Grafikfenster und gibt die globale Turtlereferenz zurück
makeTurtle(color)	erzeugt eine Turtle mit gegebener Farbe und gibt die globale Turtlereferenz zurück
makeTurtle("sprites/turtle.gif")	erzeugt Turtle mit einem eigenen Turtle-Bild turtle.gif und gibt die globale Turtlereferenz zurück
makeTurtle(color, size)	erzeugt eine kreisförmige Turtle mit gegebener Farbe und Durchmesser (in Pixel)
getTurtle()	gibt die globale Turtlereferenz zurück
t = Turtle()	erzeugt ein Turtleobjekt t
tf = TurtleFrame()	erzeugt ein Bildschirmfenster, in dem mehrere Turtles leben
tf = TurtleFrame(title)	dasselbe, aber mit gegebenem Titel
t = Turtle(tf)	erzeugt Turtle t im TurtleFrame tf
clone()	erzeugt ein Turtleklon (gleiche Farbe, Position, Blickrichtung)
isDisposed()	gibt True zurück, falls das Fenster geschlossen ist
putSleep()	hält den Programmablauf an, bis wakeUp() aufgerufen wird
wakeUp()	führt angehaltenen Programmablauf weiter
enableRepaint(False)	schaltet das automatische Bildschirm-Rendering aus
repaint()	rendert den Bildschim (nach dem Ausschalten des automatischen Rendering)
savePlayground()	speichert den Playground in einem internen Bildbuffer (zurückholen mit clear())
savePlayground(fileName, format)	speichert den Playground als Bilddatei (format: "png" oder "gif"). Im Fehlerfall wird False zurückgegeben. fileName kann eine absolute Pfadangabe sein.
setPlaygroundSize(width, height)	setzt die Grösse des Turtlefensters unabhängig von den Einstellungen in TigerJython (muss vor makeTurtle() aufgerufen werden)
setFramePosition(x, y)	setzt die obere linke Ecke des Turtlefensters an die gegebene Bildschirmposition
setFramePositionCenter()	setzt das Turtlefenster in Bildschirmmitte

(a) TigerJython

Befehl	Aktion
makeTurtle()	erzeugt eine (globale) Turtle und zeigt sie in der Homeposition an
makeTurtle(mousePressed = onMousePressed)	erzeugt eine Turtle und registriert die Callbackfunktion onMousePressed, die auf Drücken einer Maustaste reagiert.
makeTurtle(mouseDragged = onMouseDragged)	erzeugt eine Turtle und registriert die Callbackfunktion onMouseDragged, die auf die Bewegung mit gedrücken Maustaste reagiert.
makeTurtle(keyPressed = onKeyPressed)	erzeugt eine Turtle und registriert die Callbackfunktion onKeyPressed die auf Drücken einer Tastaturtaste reagiert.
back(distance), bk(distance)	bewegt Turtle rückwärts
forward(distance), fd(distance)	bewegt Turtle vorwärts
hideTurtle(), ht()	macht Turtle unsichtbar (Turtle zeichnet schneller)
left(angle), lt(angle)	dreht Turtle um den gegebenen Winkel (in Grad) nach links
right(angle), rt(angle)	dreht Turtle um den gegebenen Winkel (in Grad) nach rechts
penDown(), pd()	setzt Zeichenstift ab (Spur sichtbar)
penUp(), pu()	hebt den Zeichenstift (Spur unsichtbar)
setPenWidth(width)	setzt die Dicke des Stifts in Pixel
showTurtle(), st()	zeigt Turtle

(b) Python-Online

Funktion	Aktion
makeTurtle()	erzeugt eine (globale) Turtle im neuen Grafikfenster
makeTurtle(color)	erzeugt eine Turtle mit angegebener Farbe
makeTurtle("sprites/turtle.gif")	erzeugt Turtle mit einem eigenen Turtle-Bild turtle.gif
t = Turtle()	erzeugt ein Turtleobjekt t
tf = TurtleFrame()	erzeugt ein Bildschirmfenster, in dem mehrere Turtles leben
t = Turtle(tf)	erzeugt ein Turtleobjekt t im TurtleFrame tf
clone()	erzeugt ein Turtleklon (gleiche Farbe, Position, Blickrichtung)
isDisposed()	gibt True zurück, falls das Turtlefenster geschlossen ist
putSleep()	hält den Programmablauf an, bis wakeUp() aufgerufen wird
wakeUp()	führt angehaltenen Programmablauf weiter
enableRepaint(False)	schaltet das automatische Bildschirmrendering aus
repaint()	rendert den Bildschirm (nach dem Ausschalten des automatischen Rendering)
savePlayground()	speichert den Playground in einem internen Bildbuffer (zurückholen mit clear())
savePlayground(fileName, format)	speichert den Playground als Bilddatei (format: "png" oder "gif"). Im Fehlerfall wird False zurückgegeben
Options.setFramePosition(x, y)	setzt die obere linke Ecke des Turtlefensters an die gegebene Bildschirmposition (muss vor makeTurtle() aufgerufen werden)
Options.setPlaygroundSize(width, height)	setzt die Grösse des Turtlefensters unabhängig von den Einstellungen in TigerJython (muss vor makeTurtle() aufgerufen werden)

(c) TigerJython4Kids

Befehl	Abkürzung	Beschreibung
from gturtle import *		Die Schildkrötenbefehle laden. Erst danach versteht der Computer die Befehle für die Turtle.
makeTurtle()		Die Turtle auf der Zeichenfläche erstellen.
forward(Anzahl Schritte)	fd(Anzahl Schritte)	Anzahl Schritte vorwärts gehen.
back(Anzahl Schritte)	bk(Anzahl Schritte)	Anzahl Schritte rückwärts gehen.
left(Winkel)	lt(Winkel)	Auf der Stelle um den Winkel nach links drehen.
right(Winkel)	rt(Winkel)	Auf der Stelle um den Winkel nach rechts drehen.
setPenColor("Farbe")	spc("Farbe")	Die Farbe des Stiftes setzen. Die Farbe wird in Englisch angegeben. Die Turtle erhält eine Umrandung in der gewählten Stiftfarbe.

setPenWidth(Stiftbreite)
setLineWidth(Stiftbreite)
	spw(Stiftbreite)	Die Breite des Stiftes setzen. Die Breite wird in Pixeln angegeben.

print("Text")
print(arithm. Audruck)
		Den Text zwischen Anführungszeichen oder das Resultat des arithmetischen Ausdrucks ins Ausgabe-Fenster schreiben.

delay(Zeit)		Eine Anzahl Millisekunden warten, bevor das Programm weiterläuft.
speed(Geschwindigkeit)		Ändert die Geschwindigkeit der Turtle. Wird als Geschwindigkeit -1 angegeben, so läuft die Turtle, so schnell sie nur kann. Die Turtle läuft am langsamsten mit Geschwindigkeit 1.
hideTurtle()	ht()	Die Turtle unsichtbar machen, damit die Bilder schnell gezeichnet werden.
showTurtle()	st()	Die Turtle wieder sichtbar machen, damit du siehst, wie sie zeichnet.

(d) WebTigerJython

Figure 1.1. Screenshots of different Turtle documentations

These differences are not breaking issues per se. However, it can cause frustrations for students if they by chance find a different documentation than they were supposed to. Additionally, keeping all of them up to date results in more effort as work has to be repeated over the different versions.

By centralizing the information one can reduce the workload and make sure the information is as consistent as possible throughout the different versions. It would also open up more possibilities which with the current system are hard to achieve, for example, the integrated documentation in WebTigerPython.

1.2 Contribution

My contribution towards achieving this goal consisted of the implementation of a backend, comprising of a MySQL [5] database and an API utilizing the Python FastAPI framework [15], and a user interface to maintain the documentation, which is based on the Vue framework [17].

The documentation information is stored in the database which in turn is made accessible and editable through an API.

The user interface is a browser based tool which provides the means for maintaining and expanding the documentation.

Chapter 2

Related Work

In order to design a centralized documentation that can later be embedded in WebTigerPython, we had to decide what information is necessary and needs to be stored for every function. We considered the existing documentations as well as other IDEs with an integrated documentation which we took inspiration from.

One IDE we looked at is the micro:bit Python Editor [13] which not only has documentation about the micro:bit's functionality but also more basic programming knowledge such as data types.

We also looked at the documentation in the Eclipse IDE [7] which is more verbose than what we had in mind. This in no way should discredit the Java Documentation in Eclipse. However, the information provided there would likely be more confusing than helpful to people learning the basics.

2.1 Design Inspiration

From the existing documentations we can take the descriptions, code snippets and what is referred to as "Befehl" as an identifier or title. Using this as our baseline we included a second description which can be used to go into more detail if necessary. Similar to the micro:bit Python Editor this more detailed description is not strictly required to understand the function and therefore is hidden unless needed.

Another feature we wanted to carry over from the Python-Online documentation is the ability to open the code snippet directly in WebTigerPython.

When it came to displaying the documentation information, neither the existing documentations nor the micro:bit Python Editor offered us any concrete ideas. This is due to the fact that the micro:bit editor is focused on one specific library and because the existing documentations have the different parts disjoint. For this reason we pivoted to the Java Docs [4] which feature a tree view akin to a file directory.

2.2 Frameworks

For the API we chose FastAPI which is a performant and straight forward API framework written in Python.

For the database we originally wanted to use PostgreSQL [8], as it is widely used. There was no specific feature it offers that we needed. In the end we went with a

MySQL database because ISG [1], who hosts the database, does not yet support PostgreSQL and MySQL is the only database management system they currently support.

For the user interface we used Vue which was already used for WebTigerPython. To simplify the designing process of the user interface, we also included Vuetify [11] which offers a collection of components which provide a lot of functionality and exhibit a consistent style. We also used Vueform [12] to expand the form capabilities of Vue. Vueform adds a lot of functionality to forms while sticking to the basic style ideas of the Vue framework. Vueform also comes with an online form builder tool which aids the design process greatly.

Chapter 3

Design

During the design process we evaluated the requirements for the centralized documentation. To that end we came up with a workflow which showed us what was needed to achieve the desired functionality.

After getting this overview, we worked out the details of the individual requirements until they met our expectations.

3.1 Workflow Requirements

Part of maintaining the documentation is the addition new functions. To facilitate that, a form is required where the editors can enter the function information. The repetitive parts of the function information, such as the language, tags and the library it is part of should be assignable through dropdown menus to prevent typos. Once all the function information has been entered it is sent to the backend where it is buffered until it has been reviewed. Only after it has been accepted by a moderator it becomes part of the documentation.

In case there was something that requires changing, the function can be rejected, which requires a short text explaining why. This allows editors to perform the necessary changes. These changes once again go through the review process.

After the function has been accepted by a moderator it needs to be positioned in relation to the other functions of the library. This should be done through a drag and drop interface to make it intuitive. The positioning is necessary because it allows us to have functions that perform similar actions close to each other. Otherwise we would rely on having the functions sorted alphabetically or in the order they were added.

As TigerJython is not only used in German-speaking countries, it is important to provide the documentation in several languages. To that end the function has to get translated and it is important that the same information is conveyed in every language. To assist the editors in translating, a different language can be displayed next to it, which can be used as a reference.

The edited function also goes through the review process where the moderators have the option to view the function in a different language too.

In case a mistake has been found with one of the functions, there are two courses of action, depending on the kind of error. If it is one that fundamentally changes the conveyed information, the other languages should reflect those changes accordingly.

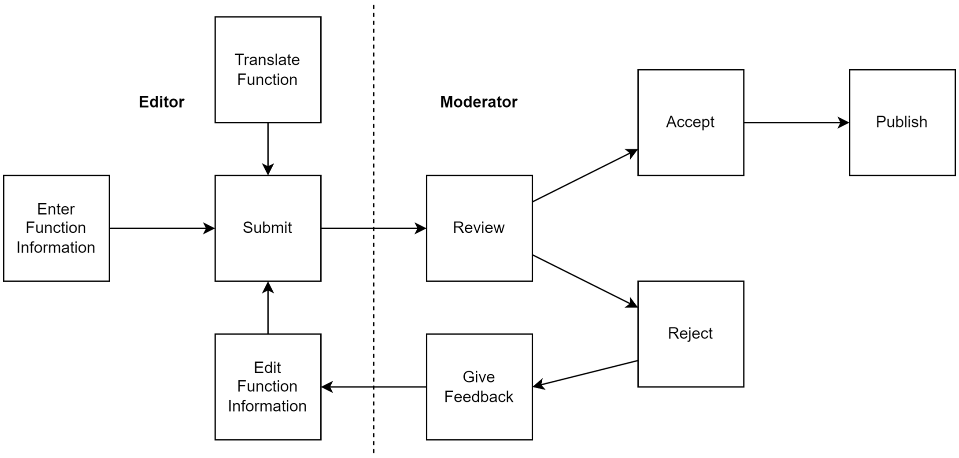


Figure 3.1. Diagram of the function workflow

Once these edits have been accepted by a moderator, the other languages of this function get marked as requiring changes.

If however it is a simple typo, the other languages are likely not affected by it and will not be marked.

Figure 3.1 features a diagram of what the workflow for functions looks like.

To make the documentation easier to navigate, it should not just have functions assigned to libraries. Similar to what exists currently, it should be organized by topic. Therefore the libraries need to be assignable too. This permits bunching libraries together when they belong to the same topic. This forms a hierarchical ordering, which allows for a quick traversal of the documentation when looking for something specific.

This hierarchy has to be translated too and will follow a similar process to what has been discussed for functions. With a notable difference, which is that only functions require moderation. Additionally libraries do not require translation, as they have the same name regardless of the language.

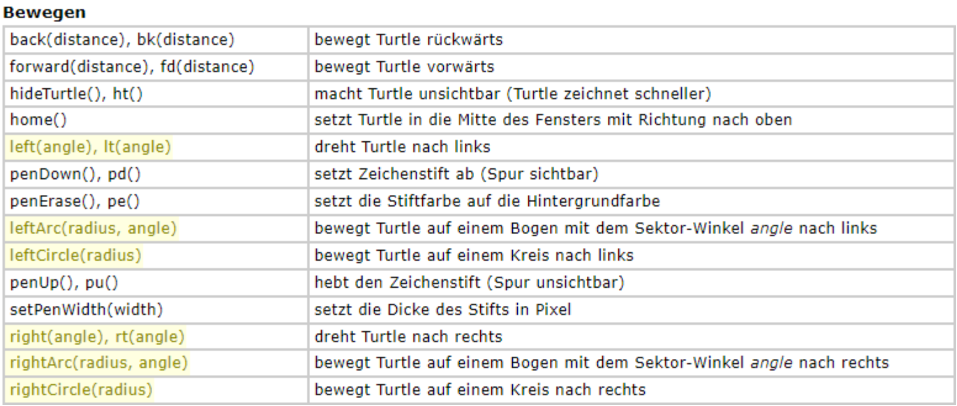
3.2 Hierarchy and Documentation Ordering

The currently existing documentations roughly group functions based on what they do and within these groups functions performing similar actions are usually adjacent to each other. Figure 3.2 presents some of the turtle movement functions documented on TigerJython4Kids. Highlighted are the functions moving the turtle to the right or the left, where the functions for the leftward movement are interrupted.

To be able to sort and order the documentation properly, we introduced four levels to organize the documentation. These are Meta Category, Category, Library and Function. Table 3.1 contains an example using the function `forward()` from the `callibot` Python library. It is used to make the Calliope robot move forwards.

There are cases where not all levels of the hierarchy are necessary, for example, when there is one single library. In such cases we will artificially extend the hierarchy by using the library or category name respectively in the next higher level.

Each of these levels gets its own indexing, allowing the entire documentation to be sorted from the highest level in the hierarchy (Meta Categories) down to the lowest



Bewegen

back(distance), bk(distance)	bewegt Turtle rückwärts
forward(distance), fd(distance)	bewegt Turtle vorwärts
hideTurtle(), ht()	macht Turtle unsichtbar (Turtle zeichnet schneller)
home()	setzt Turtle in die Mitte des Fensters mit Richtung nach oben
left(angle), lt(angle)	dreht Turtle nach links
penDown(), pd()	setzt Zeichenstift ab (Spur sichtbar)
penErase(), pe()	setzt die Stiftfarbe auf die Hintergrundfarbe
leftArc(radius, angle)	bewegt Turtle auf einem Bogen mit dem Sektor-Winkel *angle* nach links
leftCircle(radius)	bewegt Turtle auf einem Kreis nach links
penUp(), pu()	hebt den Zeichenstift (Spur unsichtbar)
setPenWidth(width)	setzt die Dicke des Stifts in Pixel
right(angle), rt(angle)	dreht Turtle nach rechts
rightArc(radius, angle)	bewegt Turtle auf einem Bogen mit dem Sektor-Winkel *angle* nach rechts
rightCircle(radius)	bewegt Turtle auf einem Kreis nach rechts

Figure 3.2. Screenshot of the TigerJython4Kids Turtle movement functions

Meta Category	Category	Library	Function
Robotik	Calliope	callibot	forward()

Table 3.1. Example of the documentation hierarchy in the databse

level (Functions). This is crucial because otherwise we would rely on having the functions sorted alphabetically or in the order they were added. As these orderings would be more or less random, functions that belong together because they perform similar actions would be spread out and therefore hard to find.

3.3 Function Documentation

A lot of information is to be stored for functions. This includes a title, a short description, a code snippet and optionally a longer description, all of which should exist in every language supported by the documentation. Translating the code snippet might appear odd; however, having the variable names in a language the students understand can aid them in understanding how a function works. This becomes even more important for the integrated documentation in WebTigerPython, because the code snippet will be draggable to the editor where it can be used directly.

To keep the information consistent in all languages, we have a "requires changes" boolean which can be set for all other language versions of a function if something has been changed substantially and these changes need to be propagated to the other languages too.

Tags were introduced to allow an even more fine grained selection of the information in case only a subset of functions from a library is required. One use case for this would be a task where students have to make a turtle move across the canvas and only require the turtle movement functions.

The number of tags that can be given to a function is not limited so they can be used for various collections of functions.

Another issue that was mentioned in the introduction is that currently not every version (i.e. WebTigerPython or TigerJython) supports the same function scope. Functions that do not work should ideally not be displayed which is why the ability to filter for a specific version is added. It allows editors to set booleans depending on whether a function is supported by a version or not.

If functions behave differently depending on the version, they can be added multiple times with the corresponding correct definition and will then only be enabled for a specific version.

The function information stored is also the live accessible version. To prevent edits from being immediately accessible, they are buffered until a moderator has reviewed them.

3.4 Translation

Being able to support an arbitrary amount of languages posed the challenge of how the different languages are kept up to date and how we can make sure they convey the same information.

Ensuring they convey the same information can be addressed by providing a reference of different languages while translating to make sure everything has been covered.

However this does not solve the case where a language is not yet up to date because something has not been translated yet. For these cases we had to add some backend logic.

We considered taking a different language as a fallback in that case. However then the issue arose which language do we pick and what if that language does not exist either. This solution would involve a language hierarchy which defines in what order languages are taken to replace something that is missing. However the problem with this is that there is a good chance the user does not understand the fallback language.

The solution we settled on was using dummy values which read "This has not been translated to [language] yet" in the given language.

If a function, tag, category or meta category is added, an entry for every other language is created containing the dummy value.

If a new language is added, every function, tag, meta category and category receives a dummy value in the new language.

Thus all languages have a database entry for every part of the documentation.

3.5 Small Changes

To make sure that the different languages are conveying the same information, functions that have been edited in another language are marked in order to let editors know which need to be looked at. However, there are changes where this process is not required, for example, if a typo is fixed or the description is reworded. In such cases the other languages do not need to be adapted.

In these cases the changes can be marked as "minor change" which means when the changes have been accepted, they get written to the database but the other language versions do not are marked as requiring changes.

3.6 User Interface

The initial idea was that the website, which editors and moderators use to maintain the documentation, can feature one page dedicated to each area of the documentation. That is, all the functionality for Meta Categories would be on one page, the ones for Categories on another one and so on.

When I began sketching some ideas to figure out what it might look like, I realized that combining adding, editing, deleting and ordering would become disorganized and very unintuitive to use.

This is why we decided to split the adding, editing and ordering functionalities into their own pages, while also still being separated from the other areas. Deleting was less of an issue and could be added both to ordering and editing. As there seemed to be no downside in having the option to delete in both, we settled on adding it to both.

Chapter 4

Implementation

The backend is written in Python. I used FastAPI to set up the API and mysql-connector-python [14] to interact with the MySQL database which has been provided by ISG.

The API documentation can be found at `«api url»/docs`.

The user interface is written in TypeScript and uses Vue as the base JavaScript framework. To aid the process I used Vuetify and Vueform.

Vuetify is an open source UI component framework for Vue, whose components make up the majority of the design elements. Vueform is a framework to extend the functionality of forms for Vue projects. It comes with an online builder which streamlines the creation of forms and their implementation.

Both the backend and the user interface have been dockerized to streamline setup and deployment.

Figure 4.1 features a diagram, of how the user interface, the backend and the database interact with each other.

4.1 Database

The database was set up to simplify updates. This was done by moving data to a new table in case of duplicate data in the original table. For example, the translations of names were moved out of the tables which contain the position index, which is used to order the documentation, as it would have been saved for every language. This allows us to update one position instead of making sure that the position is the same in all languages.

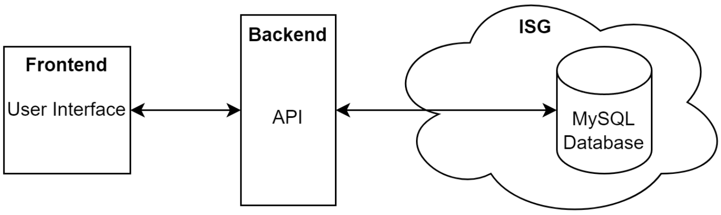


Figure 4.1. Diagram of how the different components communicate with each other

A side benefit of this is that we reduced the required storage by not storing redundant data.

Foreign key constraints have also been used to enforce consistency between tables. They prevent the usage of a key (an ID in this case) which is not defined in the parent table. Foreign keys are indicated by lines between the tables in Figure 4.2, showing the database layout.

Here, one special case is the `tags` table which only contains `tag_ids`. This is due to the fact that we did not order tags and wanted them to be translated. To make use of foreign key constraints we needed a parent table which contains the keys.

Hierarchy

Each of the four levels (Meta Category, Category, Library and Function) have their own table which keeps track of the IDs and position of the specific element. Initially the position was supposed to be global and only assigned to functions. However, we noticed that this would be impractical to maintain, therefore we changed the position to be correlated to the parent level. For example, every Meta Category has a Category with position 0.

The IDs and positions are stored in the main tables of the corresponding levels while the names and language codes are broken out into their own table, linked by their unique IDs.

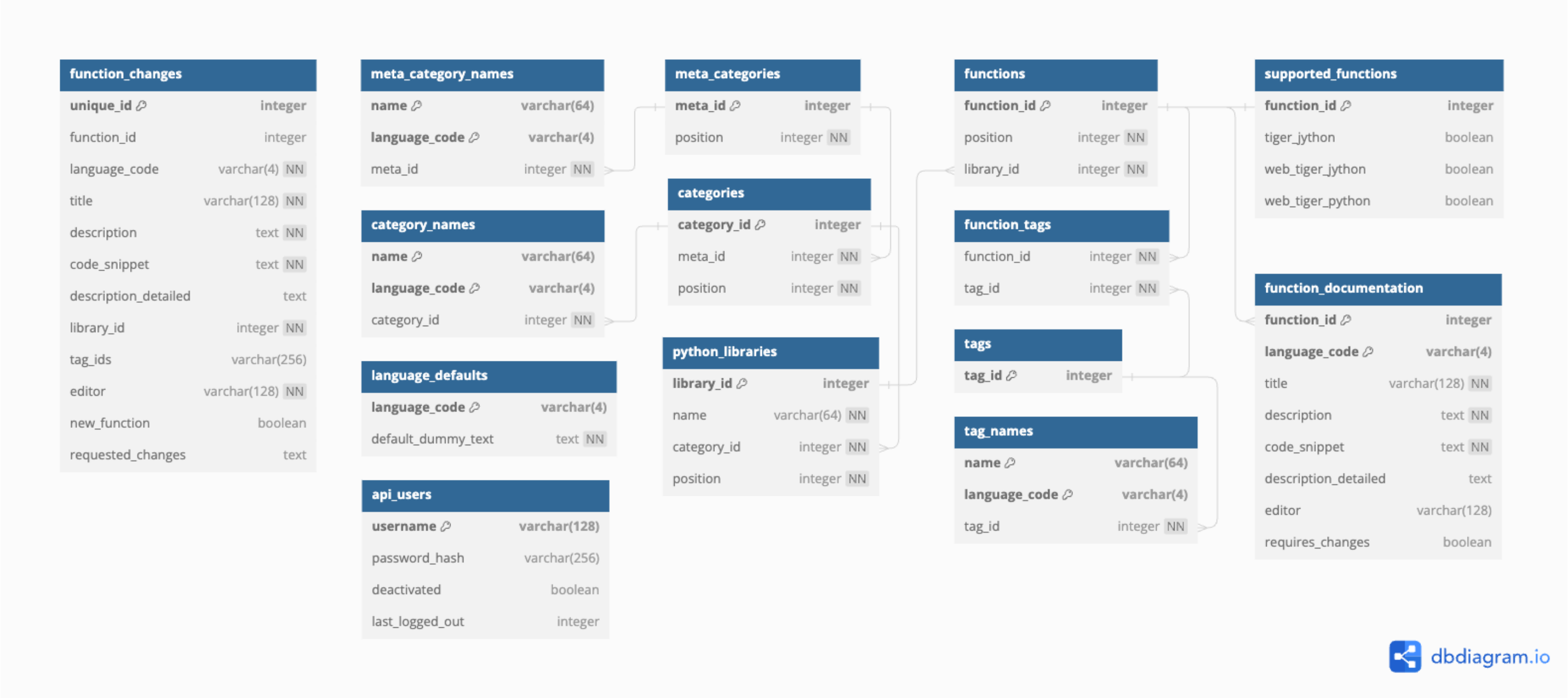
Views

The API endpoint to request the documentation from can be used in a variety of ways. It can return the entire documentation in every language and at the same time only return a couple of functions which share the same tag in one language. In brief, the selection was intended to be very flexible to fit as many use cases as possible.

I soon encountered obstacles that made the task much harder than originally expected. I attempted to build the SQL query in a smart way by adding various things only if they were needed. When discussing this with my supervisors we came to the conclusion that views [6] are going to provide the most straight forward solution.

The selection process starts with the `function_view` view, which combines various information from different tables building the basis of the selection. In successive steps each provided parameter is used to either create a subset of the previous view or join more information to the previous result which is stored in a new view. Once the selection has been made, the views that were temporarily created for this process are dropped to not clutter up the database.

My goal was to keep the views as small as possible and only keep the necessary information for each call. Assuming every parameter is provided (version, meta category ID, category ID, library ID, tag IDs and language code), the first thing that happens is that functions which are not supported by a specific version are removed. In the next step we reduce to meta categories, categories or libraries where we only take the lowest level of the hierarchy that has been provided. Following that the tag IDs are used to select only the functions with the provided tags. The last step consists of joining the left over functions with their information in the language that has been chosen.



function_changes		
unique_id	**integer**	
function_id	integer	
language_code	varchar(4)	NN
title	varchar(128)	NN
description	text	NN
code_snippet	text	NN
description_detailed	text	
library_id	integer	NN
tag_ids	varchar(256)	
editor	varchar(128)	NN
new_function	boolean	
requested_changes	text	

meta_category_names		
name	**varchar(64)**	
language_code	**varchar(4)**	
meta_id	integer	NN

category_names		
name	**varchar(64)**	
language_code	**varchar(4)**	
category_id	integer	NN

language_defaults		
language_code	**varchar(4)**	
default_dummy_text	text	NN

api_users		
username	**varchar(128)**	
password_hash	varchar(256)	
deactivated	boolean	
last_logged_out	integer	

meta_categories		
meta_id	**integer**	
position	integer	NN

categories		
category_id	**integer**	
meta_id	integer	NN
position	integer	NN

python_libraries		
library_id	**integer**	
name	varchar(64)	NN
category_id	integer	NN
position	integer	NN

functions		
function_id	**integer**	
position	integer	NN
library_id	integer	NN

function_tags		
function_id	integer	NN
tag_id	integer	NN

tags		
tag_id	**integer**	

tag_names		
name	**varchar(64)**	
language_code	**varchar(4)**	
tag_id	integer	NN

supported_functions		
function_id	**integer**	
tiger_jython	boolean	
web_tiger_jython	boolean	
web_tiger_python	boolean	

function_documentation		
function_id	**integer**	
language_code	**varchar(4)**	
title	varchar(128)	NN
description	text	NN
code_snippet	text	NN
description_detailed	text	
editor	varchar(128)	
requires_changes	boolean	

Figure 4.2. Database layout used in the backend to store the documentation.

4.2 API

By making use of the FastAPI tags, the API endpoints are grouped together based on their purposes. This not only makes the API documentation more manageable but also allows new people to get familiar with it quicker.

In an effort to keep the code defining the API endpoints as clean as possible, all the code that is responsible for interacting with the database has been moved to a separate file.

Open Code Snippets in WebTigerPython

Currently Python-Online allows the user to open code snippets in WebTigerPython. The main functionality for this is on WebTigerPython's side and we only have to provide the code in a file, in a way that is accepted by WebTigerPython.

The first attempt used files that are created in memory to prevent writing them to disc. However, these files were not accepted by WebTigerPython. It turned out that the files I created were downloaded immediately, while the files from Python-Online were opened in the browser directly.

To open the files in the browser I had to add a static access point to the API for these files. When a request is sent to the API for a specific code snippet, the file is generated and the API returns the URL under which the file can be found. They are regenerated for every access to make sure it is always the most up to date version of the code snippet. The endpoint returns a URL to the file which is then passed to WebTigerPython which extracts the code snippet.

4.3 User Interface

The user interface consists of three general parts. The navigation bar on the left where the different functionalities can be selected, the menu bar at the top where the login button is located and the preferred language can be selected, as well as the main body which fills the remaining area as shown in Figure 4.3.

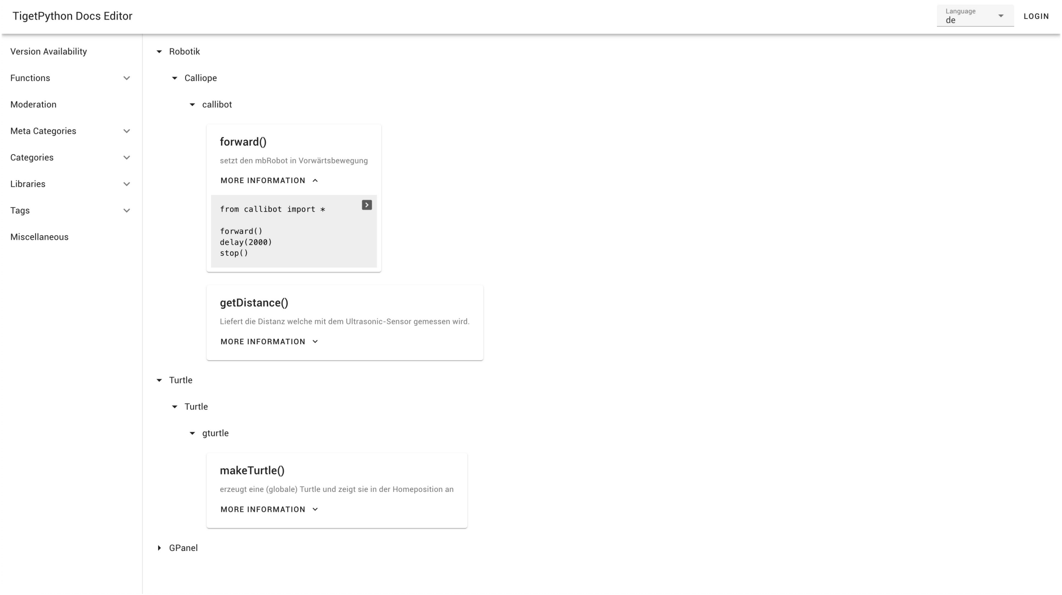
The remaining area is used to display the contents of the various pages which handle different functionality.

All the links in the navigation drawer require the user to be logged in to view the contents of the page as they all allow editing the database contents. This is not strictly necessary as there is no sensitive information being displayed, however it is easier to manage access control at that point instead of when accessing the API.

The selected language is persistent through reloads and the same applies to the authentication token. This information is stored in the local storage.

User Interface Language Changes

Users input the information through forms which I built with Vueform. One of the functionalities offered by Vueform is to assign an API endpoint to drop down menus where it can get the selection options from. This also allows you to make a call dependent on a previous selection to dynamically adjust the content by adding the previously selected values as parameters.



forward()

setzt den mbRobot in Vorwärtsbewegung

```
from callibot import *

forward()
delay(2000)
stop()
```

getDistance()

Liefert die Distanz welche mit dem Ultrasonic-Sensor gemessen wird.

makeTurtle()

erzeugt eine (globale) Turtle und zeigt sie in der Homeposition an

Figure 4.3. Screenshot of the home screen of the user interface

These updates, however, are only triggered when a field in the form is changed, which posed a problem as I wanted the form content to change based on the language selected in the top right. Basically, it initially took the right language when the page was loaded. But if the language was changed while on the page, it did not fetch the information in the correct language again.

My first attempt to fix it was somewhat hacky. I added a hidden field to the form where the language changes were mirrored to, meaning I added a function which changed the value of that field when the language was changed. Using this field as the language parameter in the API calls did result in the requests being sent again when the language was changed.

However, a different problem became apparent. Despite the data being fetched again, the values in the fields did not update. The options in the dropdown menus were in a different language, even the one that was selected, but what was displayed was still in the previous language.

The solution turned out to be to have the API requests completely detached from the form and store the results in variables that are then used by the form to display the contents in the dropdown menus. This way the options as well as the displayed selected option correctly change to the new language.

Selection Dropdowns

When adding or editing functions, libraries or categories some dependent information is required such as what it is part of or in the case of functions which tags to assign to a function. To input this information I could have used text inputs where the users write the corresponding information. This would require the users to know everything by heart and typos could cause serious issues which would not just be hard to fix but also hard to spot.

Therefore I used dropdown menus instead where the users can select the options or even start writing them and then select the matching suggestion. Thus, the internal handling gets a lot more straight forward as there is almost no room for error and we can work with IDs instead of the names. However, one downside is for example if one wants to assign a tag to a function that does not yet exist, it is not possible until the tag has been added separately and the form has been reloaded.

In the backend I expanded the API with endpoints that fetch the information required for the dropdown menus from the database and format it to match the layout that Vueform expects for the items in the dropdown menu.

Translation

To allow users to see other languages when editing, translating or moderating changes, there is a button to the right of the input fields which toggles a window. It displays the same information that has been selected on the left hand side in the language that has been chosen in the top right hand corner. This is just the default value and any of the other languages can be selected in the window, as can be seen in Figure 4.4.

Status Indication

When working with the various endpoints, there should be some status indication to let the user know if the action they tried to perform worked or if something went wrong. To achieve this, I added a small notification pop-up in the top right hand corner which is triggered by the responses received from the API. While this worked with successful API calls, it did not work with errors as in these cases a simple JSON with an error message was returned. To make sure that the errors also trigger the pop-up, I had to replace the JSONs with HTTP exceptions that then carried the right error codes.

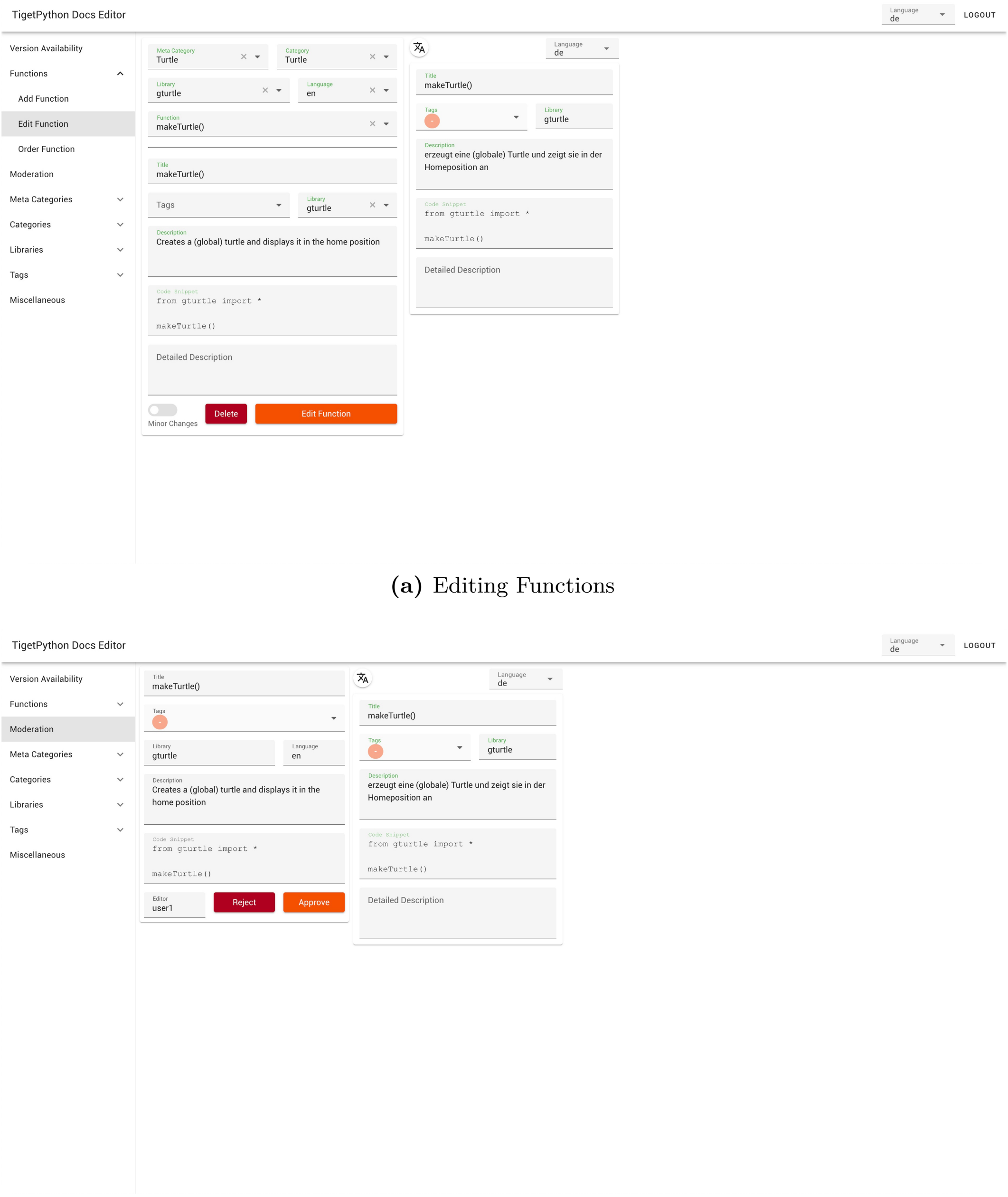
In the user interface the HTTP exceptions are filtered before the pop-up color is set and it is displayed. The filtering makes sure to ignore login exceptions which have a different indicator already and provides a way to handle unexpected exceptions from the backend. This was necessary as the actual internal server error (which is what the unexpected exception from the backend is called) is preceded by a different one which gets caught by the try-catch block. Afterwards, the following internal server exception was simply printed to the console and therefore not visible.

4.4 Security

While some security related issues, such as SQL-injections, were a topic from the beginning, others arose later on. For example, when the backend was ready for deployment, the question was raised how we prevent unauthorized access to a database hosted by ISG.

SQL-Injection Prevention

The majority of the SQL-injection prevention relies on the functionality provided by the MySQL connector library for Python [14].



TigetPython Docs Editor

Language: de

LOGOUT

Meta Category: Turtle

Category: Turtle

Library: gturtle

Language: en

Function: makeTurtle()

Title: makeTurtle()

Tags

Library: gturtle

Description: Creates a (global) turtle and displays it in the home position

Code Snippet:

```
from gturtle import *

makeTurtle()
```

Detailed Description

Minor Changes

Language: de

Title: makeTurtle()

Library: gturtle

Description: erzeugt eine (globale) Turtle und zeigt sie in der Homeposition an

Code Snippet:

```
from gturtle import *

makeTurtle()
```

Detailed Description

(a) Editing Functions

TigetPython Docs Editor

Language: de

LOGOUT

Title: makeTurtle()

Tags

Library: gturtle

Language: en

Description: Creates a (global) turtle and displays it in the home position

Code Snippet:

```
from gturtle import *

makeTurtle()
```

Editor: user1

Language: de

Title: makeTurtle()

Tags

Library: gturtle

Description: erzeugt eine (globale) Turtle und zeigt sie in der Homeposition an

Code Snippet:

```
from gturtle import *

makeTurtle()
```

Detailed Description

(b) Moderation

Figure 4.4. Screenshots of two use cases for the translation pop-up

However there are cases that are not covered by this library. One example is the column name being a parameter. To make sure that this cannot be abused for SQL-injections, I first query if the column name exists, which makes use of the sanitization functions provided by the library and only if this is successful, I use the column name unsanitized.

Authentication

To prevent unauthorized people from making changes to the database content, an authentication method is required. Due to the fact that the official login has not been set up yet, I implemented a temporary solution.

To add a user, one has to hash the password and insert it into the `api_users` table. Both of these steps are manual and do not come with an API endpoint. While one could have been added easily, the idea was to prevent the temporary solution to be too convenient as this could delay the implementation of the official login.

After the username and password have been verified in the backend, a JSON Web Token is returned to the user which will be used for further authentication. This token is valid for roughly 17 hours, after which the user has to sign in again. Currently there is no refreshing of the token to extend its expiration date.

The token is verified every time a restricted page is loaded. Once a user logs out, the token is removed locally from the local storage and invalidated in the backend. This is done by storing the time when the user last logged out and comparing that to the time the token was issued. If the log out time is after the token was issued, the token is rejected.

Chapter 5

Conclusion

I implemented a backend and a user interface to store, manage and distribute a centralized documentation for WebTigerPython and TigerJython. The user interface allows the documentation to be edited and expanded in an intuitive way and provides aids for translating function documentations to other languages. The backend consists of a database and an API through which the database content is accessible and editable. Part of the database are aids that help keeping information consistent throughout languages.

The reason why this project came to be is because there are currently multiple different versions of the documentation which have a different function scope, different function descriptions and capabilities as well as varying degrees of support for multiple languages.

By using the centralized solution, the time required to keep the documentation up to date is reduced and the various places, which can access the information through the API, have consistent and up to date information.

5.1 Future Work

The next steps involve evaluating the performance of the user interface and transferring the documentation. Once that is completed it can be integrated into various places to display the information.

Evaluation

To evaluate the performance of the user interface, people who have not yet been exposed to it should be given tasks to complete. Tracking what they do and say, and asking them to fill out a survey at the end will provide data that can be used to improve the usability.

It can also lead to bugs being found due to unintended actions which are difficult to predict. Those have to be fixed afterwards.

Data Migration

To begin the process of importing the documentation information, I would pick the most up to date and complete currently existing documentation and import it directly into the database with a script. Once that is done, one language should be completed as the reference before translating the information to other languages.

Integration in WebTigerPython

The integration into WebTigerPython should be straight forward as there already exists a prototype of it, which would have to be coupled to the API serving the function documentation as JSONs.

It currently uses YAML files to serve the documentation, however, they are translated to JSONs for internal processing. This translation step can be removed to accept the API return value and directly process it to match the established layout and functionality of the prototype. This includes the drag-n-drop functionality.

Integration in TigerJython4Kids, Python-Online and WebTigerJython

If the currently existing tabular layout persists, the integration of the API served documentation should not be too much effort. Once the required data is returned from the API, a function can iterate through it and generate the table.

The hyperlink to open the code snippet in WebTigerPython could pose a challenge as the files used do not exist by default and therefore one cannot just add a URL. To make that possible the links have to be coupled to a function that sends the request to the API and then opens the returned address in a new window.

Currently they also have functions grouped based on what they do. While that can be achieved with the tags we introduced, the functions would need additional processing to sort them based on tags. An alternative approach would be to request only certain tags from the API.

Integration in TigerJython

The process for the locally installed version of TigerJython documentation should be mostly identical to what was discussed for TigerJython4Kids and Python-Online. One thing to consider is that this version might be used on devices without internet connection.

One solution may be to add a snapshot of the documentation while the installer is being created. This would be the fall back version in case there is no internet connection and if there is a connection, the fall back version could be updated while the regular fetch is run.

Bibliography

[1] ISG. https://inf.ethz.ch/people/administration/it-service.html. Accessed 2024-08-12.

[2] Jarka Arnold and Aegidius Plüss. TigerJython4Kids. https://tigerjython4kids.ch/. Accessed 2024-07-31.

[3] Clemens Bachmann. WebTigerJython 3 A Web-Based Python IDE Supporting Educational Robotics. Master's thesis, ETH Zurich, 2023.

[4] Oracle Corporation. Java Docs. https://docs.oracle.com/javase/8/docs/api/. Accessed 2024-08-15.

[5] Oracle Corporation. MySQL. https://www.mysql.com/. Accessed 2024-08-12.

[6] Oracle Corporation. MySQL Views. https://dev.mysql.com/doc/refman/8.4/en/views.html. Accessed 2024-08-16.

[7] Eclipse Foundation. Eclipse. https://eclipseide.org/. Accessed 2024-08-15.

[8] PostgreSQL Global Development Group. PostgreSQL. https://www.postgresql.org/. Accessed 2024-08-15.

[9] TJ Group. Python-Online. https://www.python-online.ch. Accessed 2024-07-31.

[10] Tobias Kohn. *Teaching Python Programming to Novices: Addressing Misconceptions and Creating a Development Environment*. Doctoral thesis, ETH Zurich, Zürich, 2017.

[11] John Leider et al. Vuetify. https://vuetifyjs.com/. Accessed 2024-08-07.

[12] FormTools Ltd. Vueform. https://vueform.com/. Accessed 2024-08-07.

[13] Micro:bit Educational Foundation. Micro:bit Python Editor. https://python.microbit.org/v/3. Accessed 2024-08-12.

[14] Oscar Pacheco et al. mysql-connector-python. https://pypi.org/project/mysql-connector-python/. Accessed 2024-08-12.

[15] Sebastián Ramírez et al. FastAPI. https://fastapi.tiangolo.com/. Accessed 2024-08-12.

[16] Nicole Trachsler. Webtigerjython - a browser-based programming ide for education. Master thesis, ETH Zurich, Zurich, 2018.

[17] Evan You et al. Vue. https://vuejs.org/. Accessed 2024-08-12.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Eigenständigkeitserklärung

Die unterzeichnete Eigenständigkeitserklärung ist Bestandteil jeder während des Studiums verfassten schriftlichen Arbeit. Eine der folgenden drei Optionen ist in Absprache mit der verantwortlichen Betreuungsperson verbindlich auszuwählen:

○ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Es wurden keine Technologien der generativen künstlichen Intelligenz[1] verwendet.

☒ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Als Hilfsmittel wurden Technologien der generativen künstlichen Intelligenz[2] verwendet und gekennzeichnet.

○ Ich bestätige, die vorliegende Arbeit selbständig und in eigenen Worten verfasst zu haben, namentlich, dass mir niemand beim Verfassen der Arbeit geholfen hat. Davon ausgenommen sind sprachliche und inhaltliche Korrekturvorschläge durch die Betreuungsperson. Als Hilfsmittel wurden Technologien der generativen künstlichen Intelligenz[3] verwendet. Der Einsatz wurde, in Absprache mit der Betreuungsperson, nicht gekennzeichnet.

Titel der Arbeit:

Centralized documentation for TigerJython and WebTigerPython

Verfasst von:
Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich.

Name(n):	**Vorname(n):**
Hunkeler	Andri

Ich bestätige mit meiner Unterschrift:

- Ich habe mich an die Regeln des «Zitierleitfadens» gehalten.
- Ich habe alle Methoden, Daten und Arbeitsabläufe wahrheitsgetreu und vollständig dokumentiert.
- Ich habe alle Personen erwähnt, welche die Arbeit wesentlich unterstützt haben.

Ich nehme zur Kenntnis, dass die Arbeit mit elektronischen Hilfsmitteln auf Eigenständigkeit überprüft werden kann.

Ort, Datum	**Unterschrift(en)**
Zürich, 19.08.2024	

Bei Gruppenarbeiten sind die Namen aller Verfasserinnen und Verfasser erforderlich. Durch die Unterschriften bürgen sie grundsätzlich gemeinsam für den gesamten Inhalt dieser schriftlichen Arbeit.

[1] z. B. ChatGPT, DALL E 2, Google Bard
[2] z. B. ChatGPT, DALL E 2, Google Bard
[3] z. B. ChatGPT, DALL E 2, Google Bard