Co-funded by the
Erasmus+ Programme
of the European Union

SHERPA

SELFIE HElpeR & Pedagogical innovation Assistant

European Commission

# D2.3 SELFIE Helper final release

| Document Control Page | |
|---|---|
| **Project Acronym** | SHERPA |
| **Project Full Title** | SELFIE HElpeR & Pedagogical innovation Assistant |
| **Project Number** | 612867-EPP-1-2019-1-EL-EPPKA3-PI-FORWARD |
| **Funding Sheme** | Erasmus+ KA3 Support for Policy Reform EACEA 36/2018 |
| **Project Coordinator** | AETMA Lab of IHU (Greece) |
| **Project Start Date / Duration** | 01-01-2020 / 24+6 Months |
| **WP/Result** | WP2 / D2.3 |
| **Title** | SELFIE Helper final release |
| **Result type** | Report and Open Source Code |
| **Lead Partner** | AETMA Lab of IHU (Greece) |
| **Due date** | 31 May, 2022 (M29) |
| **Submission date** | 31 May, 2022 |
| **Abstract** | This report describes and provide the SELFIE Helper KB and Chatbot along with the CBR Inference engine at their final released version after the pilot evaluation. |
| **Author(s)** | AETMA Lab of IHU (Greece) |
| **Contributor(s)** | All partners |
| **Reviewer(s)** | |
| **Dissemination level** | Restricted to other program participants (including Commission services and project reviewers) |

| Revision and History Chart | | | |
|---|---|---|---|
| **Version** | **Date** | **Modified by** | **Comments** |
| **V1** | 1 May 2022 | AETMA | V1 with introductory content was created |
| **V2** | 18 May 2022 | JYU, TLU, GFOSS | Input on various parts of the document |
| **V3** | 25 May 2022 | AETMA, JYU, TLU, GFOSS | Various changes and amendments were made |
| **V4** | 28 May 2022 | AETMA | V4 posted for review |
| **V5** | 30 May 2022 | AETMA | Final report |

## Executive Summary

SELFIE HElpeR & Pedagogical innovation Assistant (SHERPA) is a two-year Erasmus+ KA3 project with a mission to enhance innovation in schools by supporting self-assessment processes for making better use of digital technology in teaching and learning.

The SELFIE Helper is consisted by Knowledge Base (KB) and the Chatbot which are some of the most critical elements of the project. The implementation of these components is part of the Work-Package 2.

This document presents the SELFIE Helper final release (using data derived from T2.7) consisting of SELFIE Helper KB (using data derived from T2.2) and the Chatbot (using data derived from T2.4) and is developed as the third deliverable of the WP2 (WP2-D2.3.) of the SHERPA project of Erasmus+ EACEA/36/2018. In particular, this report presents the Chatbot interface, the Cased Based Reasoning (CBR) Inference engine which is part of the chatbot backend mechanism, the SELFIE Helper KB and the Backend Management System of the KB. It describes their functionality and provides their source code and the appropriate software prototypes.

These 3 modules will allow educators to make questions through the provided Chatbot interface and automatically retrieve appropriate feedback. The CBR Inference Engine will retrieve these questions as input from the Chatbot interface, it will contact the SELFIE Helper KB to find the most related set of question-answer and it will return a proper response for the user that will be provided through the Chatbot interface. In addition, special care has been taken for SELFIE experts in order to allow them to manage the provided questions with their related answers through the back-end management system of the KB which also presented at this report.

# Table of Contents

# 1. Introduction

The SELFIE Helper consists of 3 modules:

- The Chatbot Interface, where questions by the users are asked
- The Cased Based Reasoning (CBR) Inference Engine, where questions submitted to the Chatbot Interface are forwarded to
- The Knowledge Base (KB), where cases/answers are stored

In this report, "D2.3 SELFIE Helper final release", we describe each of these modules in their final released version after the results of the pilot evaluation. This report extends the document "D2.2 Title SELFIE Helper KB and Chatbot" providing updated information regarding SELFIE Helper final release.

The report is divided into 3 sections:

In Section 2, the Chatbot Interface is described. This section contains a screenshot of the User Interface and describes its basic functionality. This section includes appropriate instructions that can be used by a school in order to use the environment efficiently. It also contains technical information about the tools used in order to build the Chatbot Interface. A link to a Git repository is also provided for further technical instructions.

In Section 3, information regarding the CBR Inference Engine and the way it works is included. More technical details and justification of the implementation decisions taken during the design process of the engine can be found on report "D2.1. Design of the CBR Inference Engine and the SELFIE Helper KB". A link to a Git repository is also provided, containing the source code for the CBR Inference Engine.

In Section 4, the design of the Knowledge Base (KB) is depicted and the administration environment for accessing the KB. A detailed view of the general design of the KB can be found on the report "D2.1. Design of the CBR Inference Engine and the SELFIE Helper KB" and details regarding the initial implementation in "D2.3 Title SELFIE Helper KB and Chatbot".

Since minor changes were applied to the KB scheme, therefore was a need to apply all the latest changes to the database. All the migrations are handled by the platform itself and are applied by running commands on the terminal. The one used for database migrations is PHP Artisan Migrate. The updated SQL database scheme is added in APPENDIX A at section 1. Additional information could be found in the migrations (2022) part of the official Laravel documentation. A link to a GitHub repository is also provided for further technical instructions.

All the project code Git repositories are provided in APPENDIX A section 3 at Table 3.

## 2. Chatbot Interface

Chatbot Interface is a web application that serves as a publicly available front-facing interface to interact with CBR Inference Engine application programming interface (API) and Knowledge Base (KB) Service. CBR Inference Engine API is used for submitting questions and getting a response, provided that an algorithm is able to find a suitable answer from available data stores. The Chatbot Interface will suggest asking the question again in English, when no suitable answer can be found. The next step would be to offer a current user the possibility to submit the question as a suggestion that will be handled by Country Selfie Experts. Suggested questions would then be submitted to the KB API endpoint as a Pending Question type. Both Chatbot Interface and specific endpoint on the KB API side are using reCPTCHA[1] service to prevent misuse by the spam bots. The chatbot interface is available through the following link: http://helper.sherpa4selfie.eu/


**Figure 1. Chatbot Interface**

Figure 1 shows a screenshot of the Chatbot Interface in English. It consists of two main parts: a few example questions on the left of the screen and the conversation on the right. The left side content consists of the most popular questions asked to the system. Clicking on one of the questions will quickly ask that from the system. If the conversation has several questions and answers with scrolling enabled, then the interface will make sure that the conversation area is scrolled to the bottom with both answer and input becoming visible. That process is also demonstrated in Figure 1. The user can also change the language of the User Interface (UI),

[1] https://developers.google.com/recaptcha/

though that will not only change the interface but would also make sure that the system is being asked questions while providing the language context according to the currently selected language. The initial round of internal evaluation focused solely on using English content and would thus be limited to the UI in English with switching the language being disabled temporarily.

Updates from the previous versions include the option for the user to declare his/her satisfaction through specific face-icon buttons as displayed in Figure 1. Moreover at the left part of the interface the sample questions and the provided text has been updated. Additionally, the UI is translated and is available now in all partners' languages. Figure 2 presents interfaces in Estonian, Finnish, Greek and Italian.

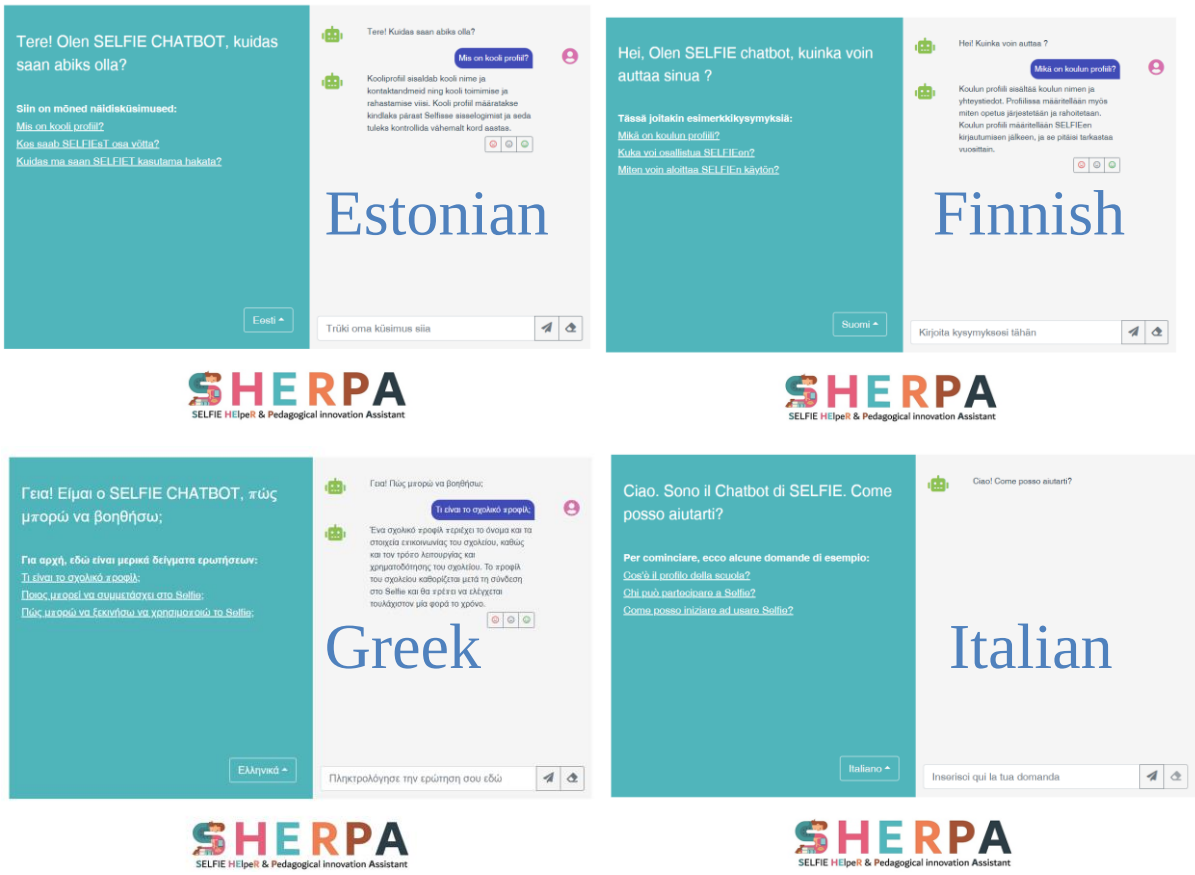


**Figure 2. Chatbot interface in partners languages**

The process of manual interacting with the Chatbot Interface can be seen in Figure 3. Users can type their questions into an input below the conversation and hit Enter or press the button to the right. If the chatbot cannot pair the question with one of the available answers then propose the user suggest this question as one that should be included in the Knowledge Base. If the user presses the “Yes” button this question is added as a suggested question into the database and SELFIE experts can manage the process of the addition to the Knowledge Base through the Backend Management System.

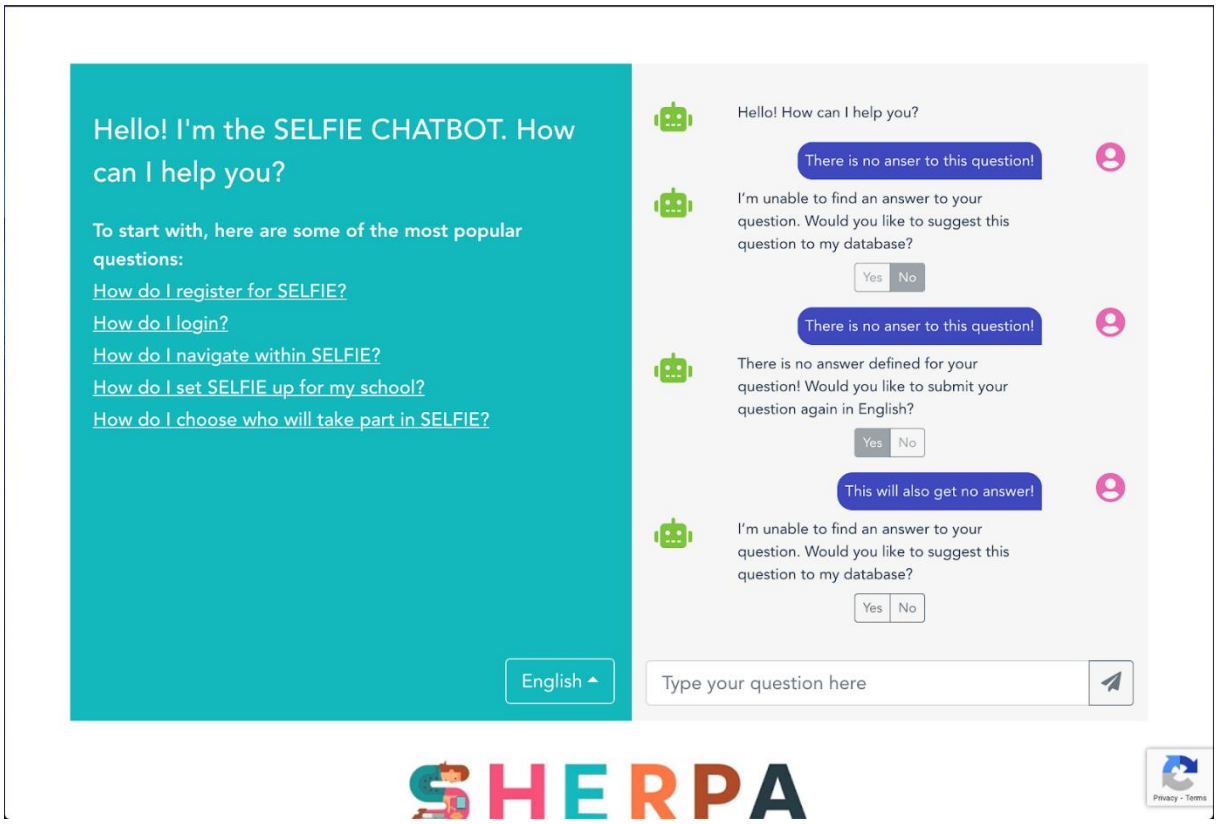


**Figure 3. Chatbot conversation with suggestions**

Chatbot Interface was generated with the use of a tool called Vue CLI (2020) and uses TypeScript (2020) instead of the standard JavaScript (Wikipedia 2020a) version. Vue.js (2020) provides a modern JavaScript Framework with built-in reactivity and a well-established ecosystem. Bootstrap User Interface (UI) framework (Bootstrap, 2020) provides some of the reusable components and styling. BootstrapVue (2020) is used to get Vue.js specific flavor of those components. The Chatbot Interface web application is a static asset package that is being built from the source code and includes configuration for services (CBR Inference Engine API, KB API and reCAPTCHA). Additional explanations and instructions could be found in the README file of the GitHub SELFIE Chatbot UI repository (SHERPA Team, 2020). Configurations are provided by an environment (.env) configuration file and included into the resulting page source code as meta tags, which allows for the application to use those values at runtime. This approach should later allow for the configurations to be changed even without rebuilding from source code to provide values for different sets of configuration options. Further technical instructions are provided in the SELFIE Helper GitHub repository (SHERPA Team, 2020a). The pre-built packages should be available once any releases[2] are tagged. The aggregation point of all SELFIE Helper Git repositories (SHERPA Team, 2020e) is provided at the APPENDIX A section 3 at Table 3.

[2] https://github.com/centre-for-educational-technology/sherpa-helper/tags

Repository has a Docker (2022) file to build a container with production ready built assets that are being served by the minimalistic nginx (2022) server, having an additional capability of providing configuration values that would be used at runtime. The process itself is pretty simple and replacement logic could be seen in the special shell script that will be triggered on each restart of the container, preceded by replacing existing assets with thce ones from the available distributive that has special placeholders which are being replaced with the values provided by the environment. A Docker Compose (2022) file serves as an example for predefined values for the container.

# 3. CBR Inference Engine

The CBR Inference Engine, is an application that is capable of answer selection based on user's questions. Its main core is based on the ChatterBot platform (Cox, 2019) which is a machine-learning-based conversational dialog engine built in Python. The CBR Inference Engine operation is based on the flow diagram of the ChaterBot provided in Figure 4.

The CBR Inference Engine works in a multilingual environment and due to the use of machine learning algorithms achieves high accurate results on user's questions. The implementation covers two main scenarios based on the existence of the appropriate response. Given a question, the CBR Engine determines the similarity with the predefined closed set of questions from the Knowledge Base. In case the similarity or confidence value is over a predefined threshold the answer to this question is given as a response to the user. In any other case, the engine returns an empty result and the Chatbot interface handles the following steps.

The issue of understanding users' queries and intent and providing appropriate responses was faced as a text and semantic similarity problem. Thus, the CBR Engine takes into account not only the surface closeness of two pieces of text but also their meaning. In order to create a more concrete solution and achieve accurate results, the implementation combines results from two different algorithms. Thus, the user response comes from the algorithm that achieved the highest score. The algorithms used are the following:

- Levenshtein distance (Wikipedia, 2020b), which is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.
- FastText algorithm (Wikipedia, 2020c), which is a library for learning of word embeddings and text classification created by Facebook's AI Research (FAIR) lab. The model represents questions as word vectors leveraging the fact that neighboring words in a sentence affect the semantic meaning of that word.

Each algorithm is trained based on the closed set of questions located on the Knowledge Base. As new questions are populated on the KB the models are retrained once a day via a cron job.

Both algorithms gather the Q&A pairs via REST calls, transform and apply them as an input. We need to mention that the FastText algorithm creates as many models as the languages supported by the system. Each model is located on the local file system and is used when needed. The CBR Engine exposes a REST API to communicate with the ChatBot Interface and based on the input parameters, language code and question, respond to the user. The CBR Inference Engine code written in python is available at the following Open Chatbot GitHub repository (SHERPA Team, 2020d).

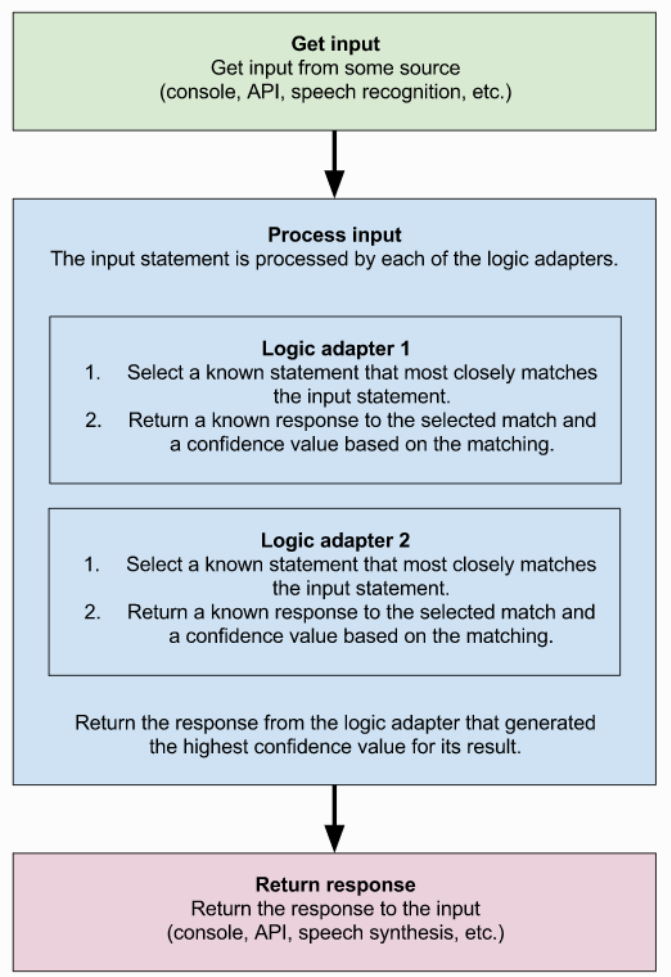


**Figure 4. ChaterBot flow diagram**

*(Source: https://chatterbot.readthedocs.io/en/stable/)*

Updates from the initial release were made in order to increase the system accuracy and performance. From a technical perspective, in order to make the system more reliable, we had to define a threshold value below which the system could return an empty response. To be more specific, a system with a high threshold value (0.8 - 0.9) would return empty results as it may not find answers with such a high confidence score. On the other hand, a system with a really low value (0.3) would return many wrong answers as the confidence value of the responses will be extremely low. Thus, through an experimental setup using many questions

and answer pairs in different languages the team decided that a value equal to 0.7 leads to promising results with an acceptable error number of wrong answers.

Furthermore, through experiments, we tried to find the most accurate hyper parameters for the fastText algorithm. The criteria for such a decision were the learning rate, the minimum n-grams and the loss function. Thus, the algorithm had high accuracy scores and in almost all cases responded impressively. Finally, our initial lack of dataset was solved with all team's effort who created almost one thousand questions using the SELFIE guide as a source and paraphrasing tools. So, we end up having a promising number of question-answer pairs for all the needed cases.

The CBR technical documentation is available at APPENDIX 7.2 CBR Technical Documentation and further technical details are provided in the SHERPA KB GitLab repository (SHERPA TEAM, 2020d). The Sherpa CBR Gitlab repository contains all the code needed to build and deploy the inference engine. Specifically, it contains a complete README file with instructions for both docker and native python execution. Also, contains a requirements txt file which contains all the necessary dependencies to install in any setup.

# 4. Knowledge Base (KB)

SELFIE Helper, includes a database that contains a set of questions and answers, described as Knowledge Base (KB). At the end of the project, the Knowledge Base comprised 4695 questions and 373 corresponding answers available via SELFIE Helper in 5 languages.

A Web application was built in order to manage questions and answers. This data would later on be used as a basis for CBR Inference Engine training data set. An additional functionality is an API endpoint for submitting suggestions (Pending Questions) that will go through a review process and could potentially become questions that are used by the SELFIE Helper. The KB database is structured according to “D2.1. Design of the CBR Inference Engine and the SELFIE Helper KB” document while information regarding the first version are available to “D2.2 SELFIE Helper KB and Chatbot”.

The final release of Knowledge Base application contains several new features and improvements, with most important one being listed below:

- Improved statistics table that also shows the number of translated Questions that have relation to translated Answers (Figure 28).
- Real-time updates functionality for all of the available data types that enable multiple users to use the system at once and get all of the data updates without ever needing to refresh the data set (Figure 33).
- Questions and answers without translations to the current language have buttons with the number of translations shown in red and fully translated to all the languages shown in green (table rows of figures 10 and 12).
- A standalone column in the answers table will not allow the user to see the list of all related questions (Figure 20). Any question from the list could be edited without closing the dialog and navigating to the questions tab (Figure 21).

- Answer select has a new input that allows both searching from the existing ones and creating the new one in place (Figures 9, 10, 12, 22 and 24).
- Questions, answers and pending questions can now be deleted with the click of a button located at the bottom of the edit form (Figures 13, 13, 14, 15 and 17). The system will show an additional dialog to make sure that the user really wants the deletion to take place.
- Categories can now be managed by Master Selfie Experts and the new feature is located in the corresponding tab (Figures 29, 30, 31, and 32).
- Chatbot Interface will now be sending data about usage to the newly added API endpoints. That includes the cases of:
    - question asked, answer received and a language key
    - question asked, answer received, language key and a user rating
- System will log actions taken with the data models, also logging changes made to the attributes. An example of that would be a new question being added, an existing question being changed and
- that same question being removed.
- Previous version did not allow adding questions or answers purely in English, without providing translation to at least one more language. New version has that capability and no longer treats English as just a baseline language for making translations.
- Latest version includes numerous upgrades to the base platform and dependencies on both the back- and front-end side. This is important for the long term sustainability of the system and would allow the developer to easily apply the updates in future versions.

KB service is based on a Laravel Web framework (Laravel, 2020) and is closely following the best practices for the structure of the codebase itself and internals of the application logic. Web application is using the default UI package that provides basic user registration and authentication flow and UI. The package is based on a Bootstrap UI framework, with Vue.js JS framework being used on the client side with an addition of components provided by BootstrapVue. The resulting UI is not a single-page application (Wikipedia, 2020d), though it heavily relies on Vue.js components with most of the complex views relying on components instead of server-side templates. More information about requirements, installation and development process could be found from Laravel Documentation and SHERPA Knowledge Base GitHub repository (SHERPA Team, 2020b) README file. The latter would hold any data that is specific to current implementation.

The Web application supports the Master Selfie Expert and the Country Selfie Experts categories of users, as described in "D2.1. Design of the CBR Inference Engine and the SELFIE Helper KB" document. Overall, the Web application supports several roles, with some of those being explicit and some implicit. The roles are as follows:

- Anonymous - anonymous user that will only be able to see the landing page with a link to login page (Figure 5), not an explicit role or a role as such. Most of the API endpoints are also publicly available
- Administrator - a user role that should mostly deal with user account management tasks, though there also is a full access for the KB specific content

- Authenticated - a user with an account that will be able to authenticate and access the User Home with no content, not an explicit role
- Country Selfie Expert - a user that has ability to review Pending Questions, add new Questions and Answers, translate existing Questions and Answers, mark Questions and Answers as translated and send those for review. This role requires the language field to have a value, making a user an expert for that specific language
- Master Selfie Expert - a user that has the ability to perform Country Selfie Expert actions for any of the languages, reviews Questions and Answers before those could be published

Users with credentials can login in order to gain access to the Web application's features (Figure 5). User Management could be accessed through the dropdown menu at the top right of the page where the name of the current user is shown. Corresponding menu entry will only be available to users that have sufficient rights for user management. As the system does not allow self-registration, the only way to create new accounts would be for one of the administrators to do that through User Management. User roles could be assigned at creation time or later on through the user account edit dialog.

There also is an API that allows Pending Question to be submitted, currently used by the Chatbot Interface application. This is the only API endpoint that uses reCAPTCHA for providing protection against spam bots. The rest of the API is only meant for fetching the Question and Answer data and would be used by the CBR Inference Engine to update the dataset while running the training routines. Current implementation would allow the API to return data for both Published and Translated content. Initial design only assumed the published content to be exposed through the API. The main difference is that Published content has to be reviewed by the Master Selfie Expert and will have translations for all the languages present.

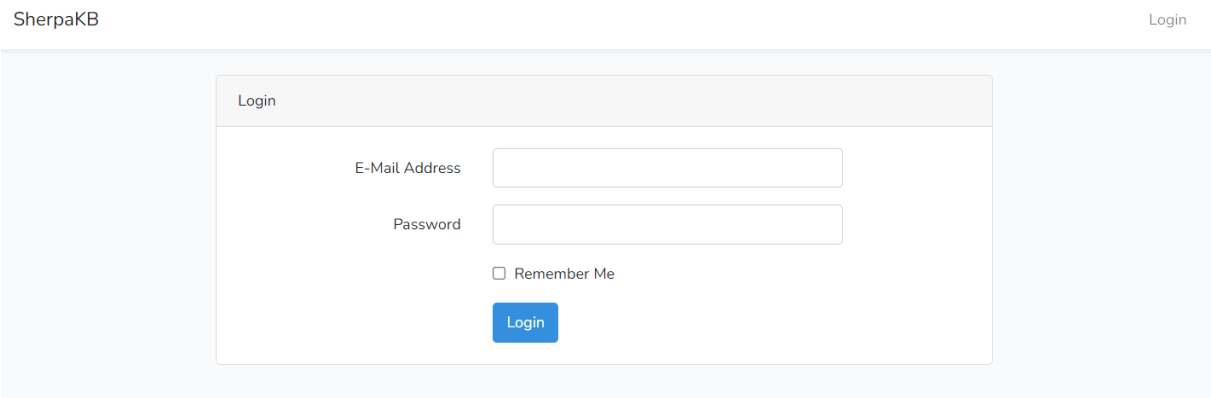


**Figure 5. Login screen**

The Administrator can perform tasks regarding user management, such as adding a new user and assign roles (Figure 6), viewing all existing users (Figure 7) and editing existing users' settings (Figure 8).

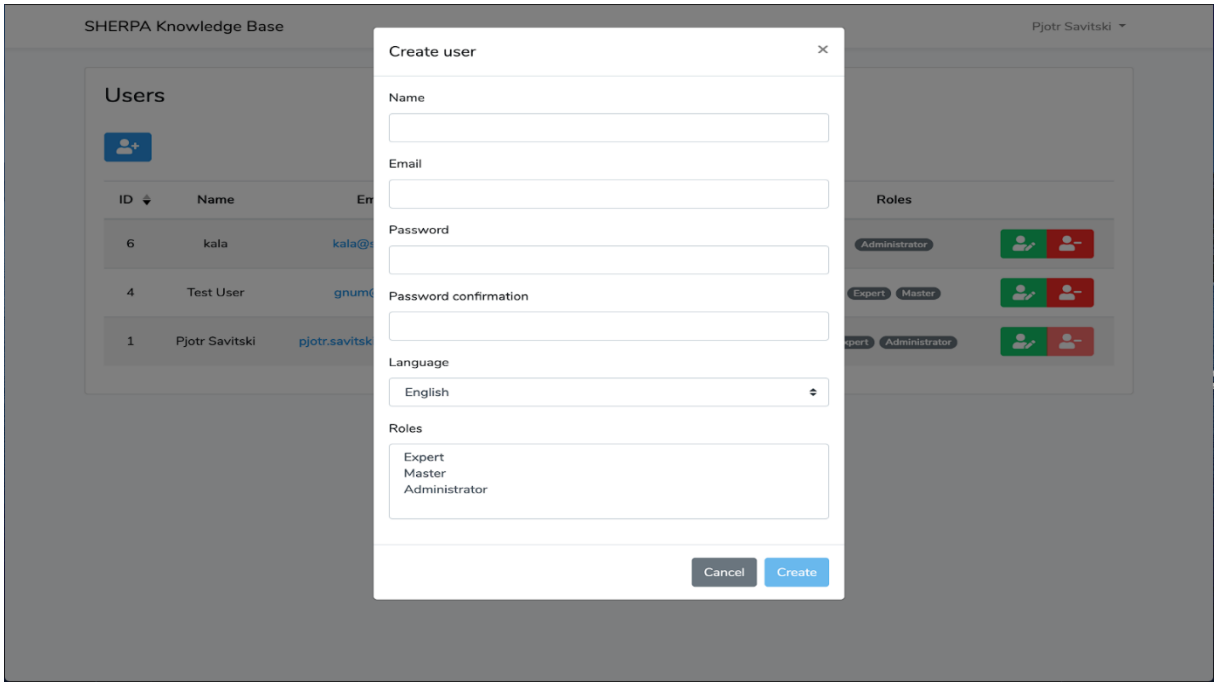


**Figure 6. Administrator adds a new user and assigns role**

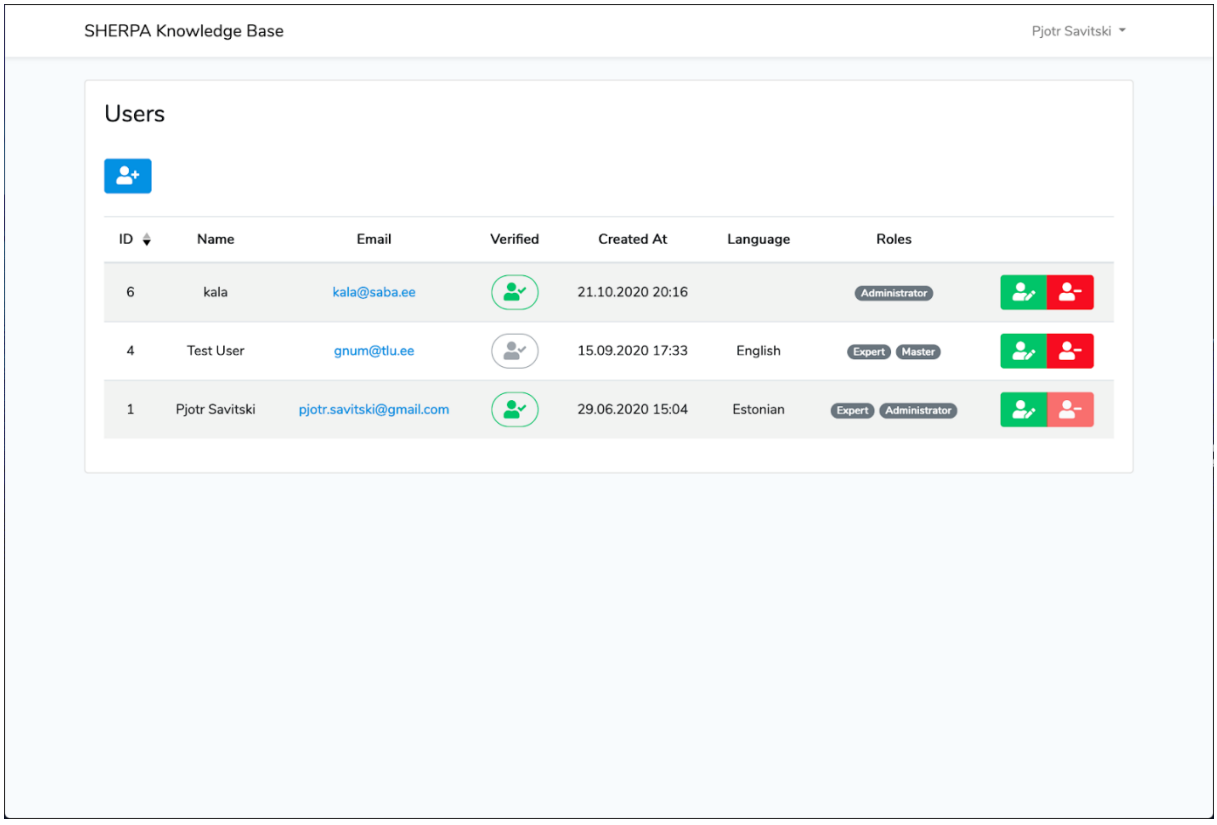


**Figure 7. Administrator user management**

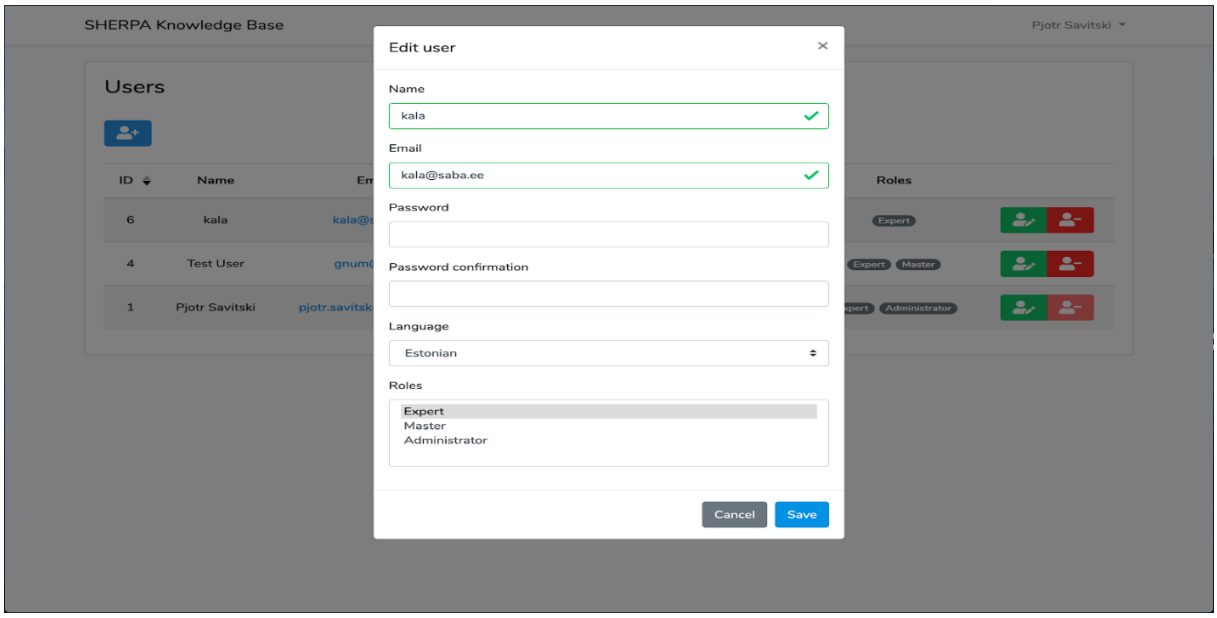


**Figure 8. Administrator edits an existing user's settings**

The Country Selfie Expert can add new Questions (Figure 9) and Answers (Figures 10 and 11) or edit existing Questions (Figure 12) and Answers (Figure 14). Both question-(Figure 13) and answers (Figure 15) can be deleted. Country Selfie Expert views for Questions and Answers also include a creation functionality that can be used by pressing the button with plus icon and filling the required data in the opened dialog. The system will also allow the table with data for any of the content types to be searched/filtered by both text in English and one for the currently selected language or language of the expert.

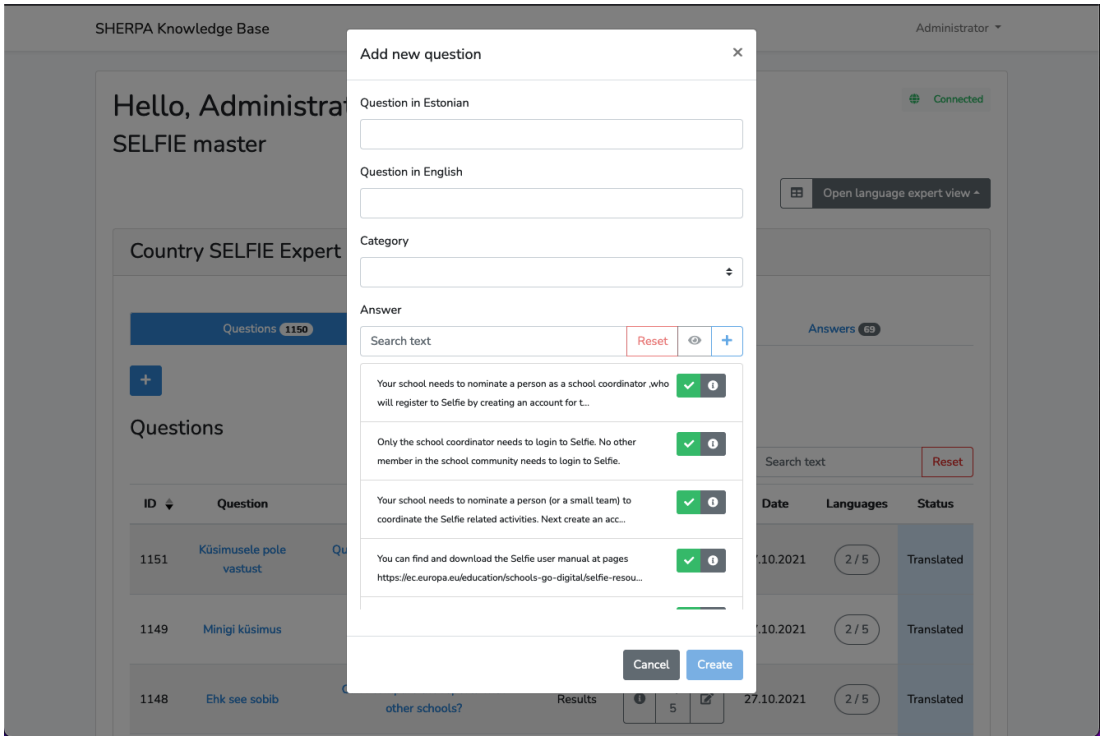

**Figure 9. The Country Selfie Expert adds a new question**

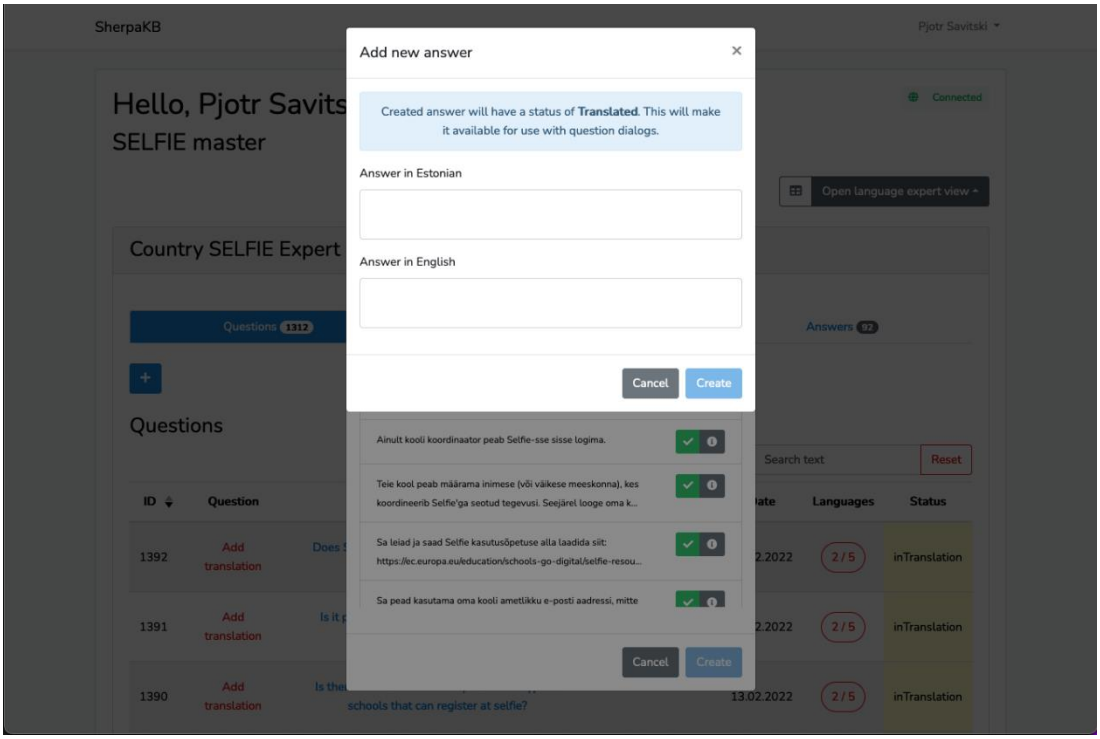


**Figure 10. The Country Selfie Expert adds a new answer while creating a question**

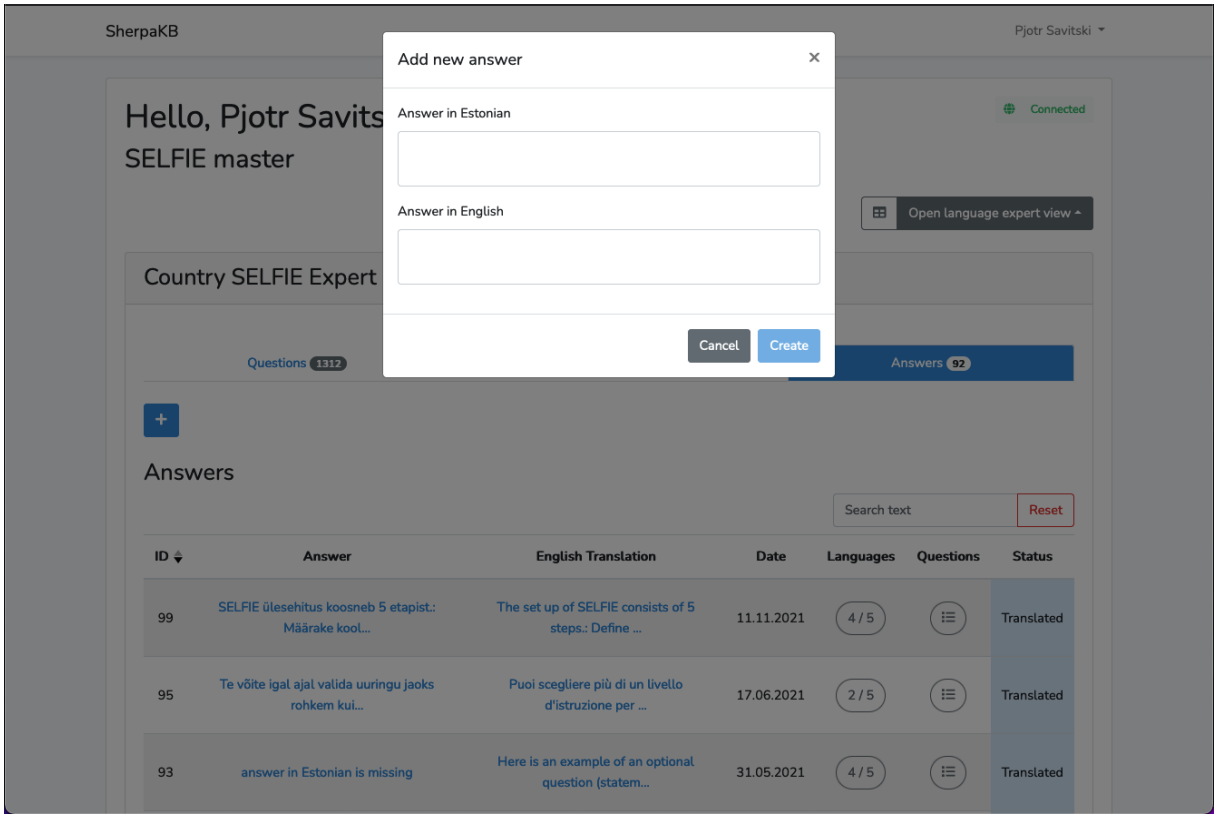


**Figure 11. The Country Selfie Expert adds a new answer**

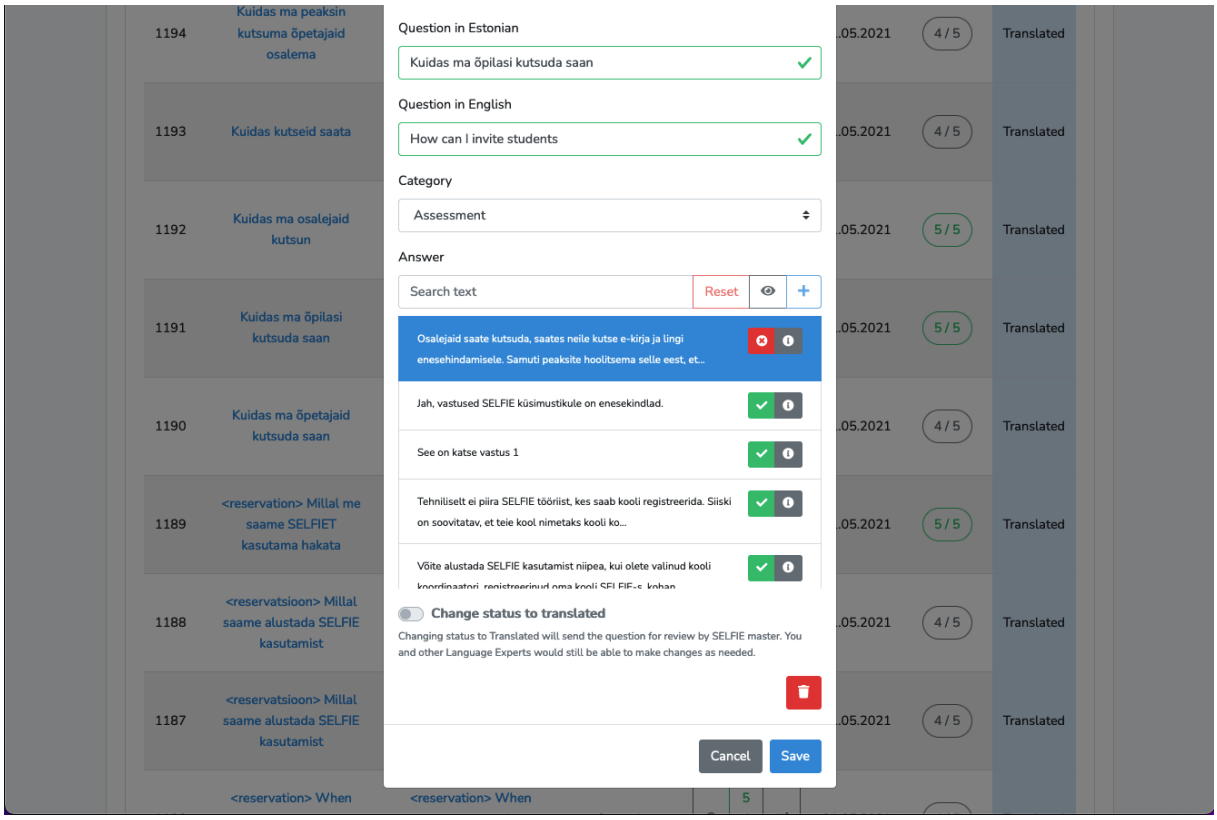


**Figure 12. The Country Selfie Expert edits a question**

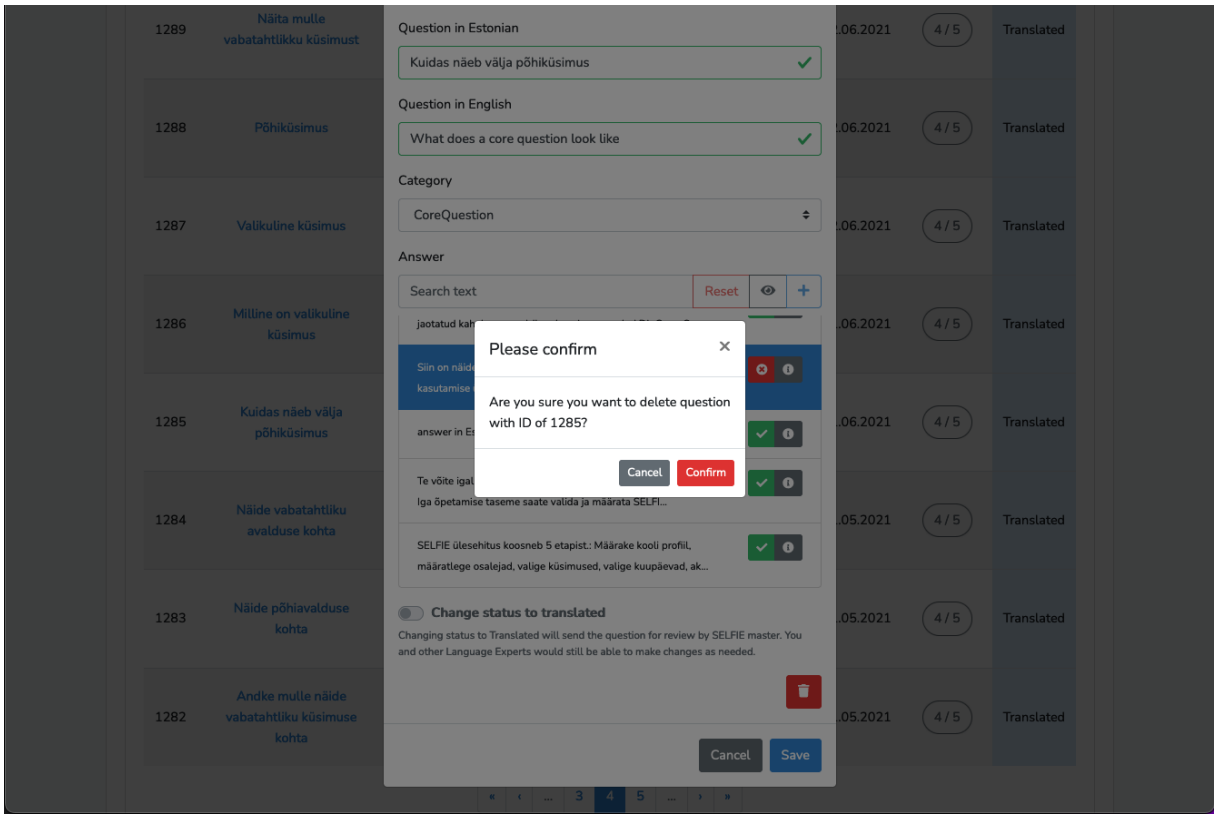


**Figure 13. The Country Selfie Expert deletes an existing question**

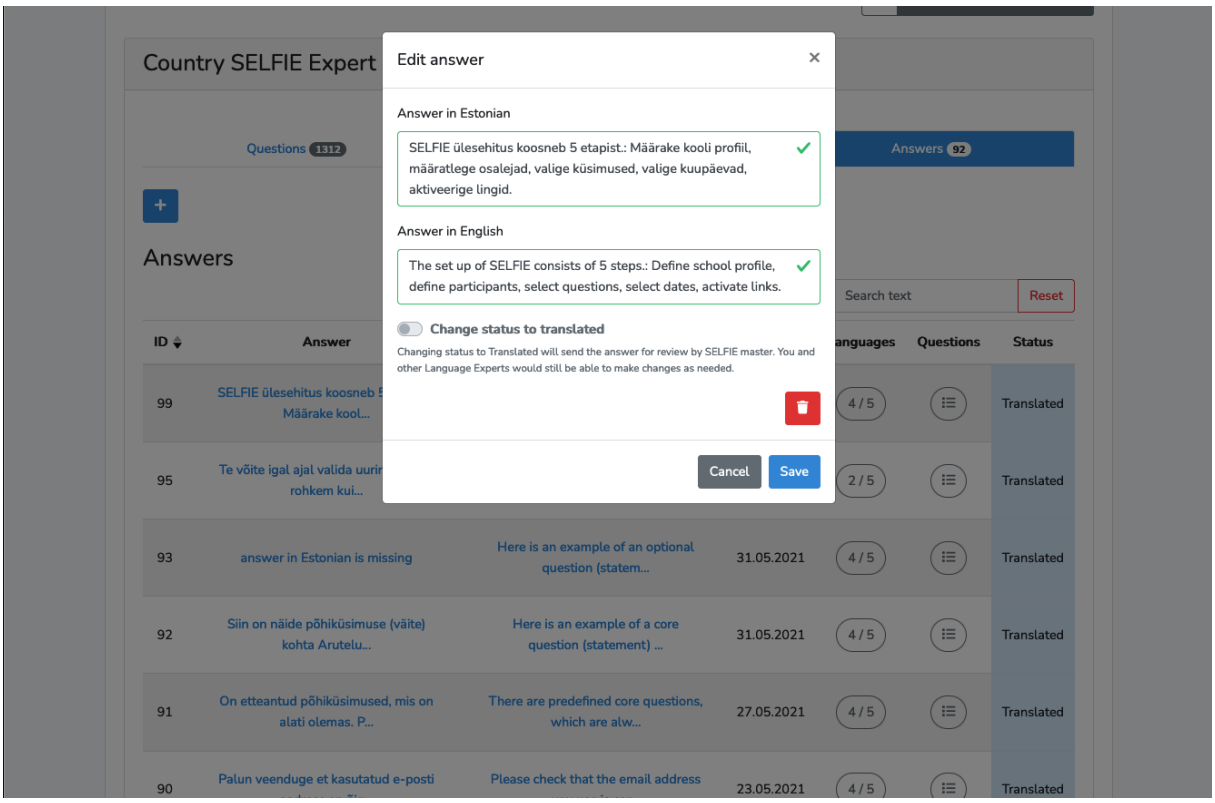


**Figure 14. The Country Selfie Expert edits an answer**

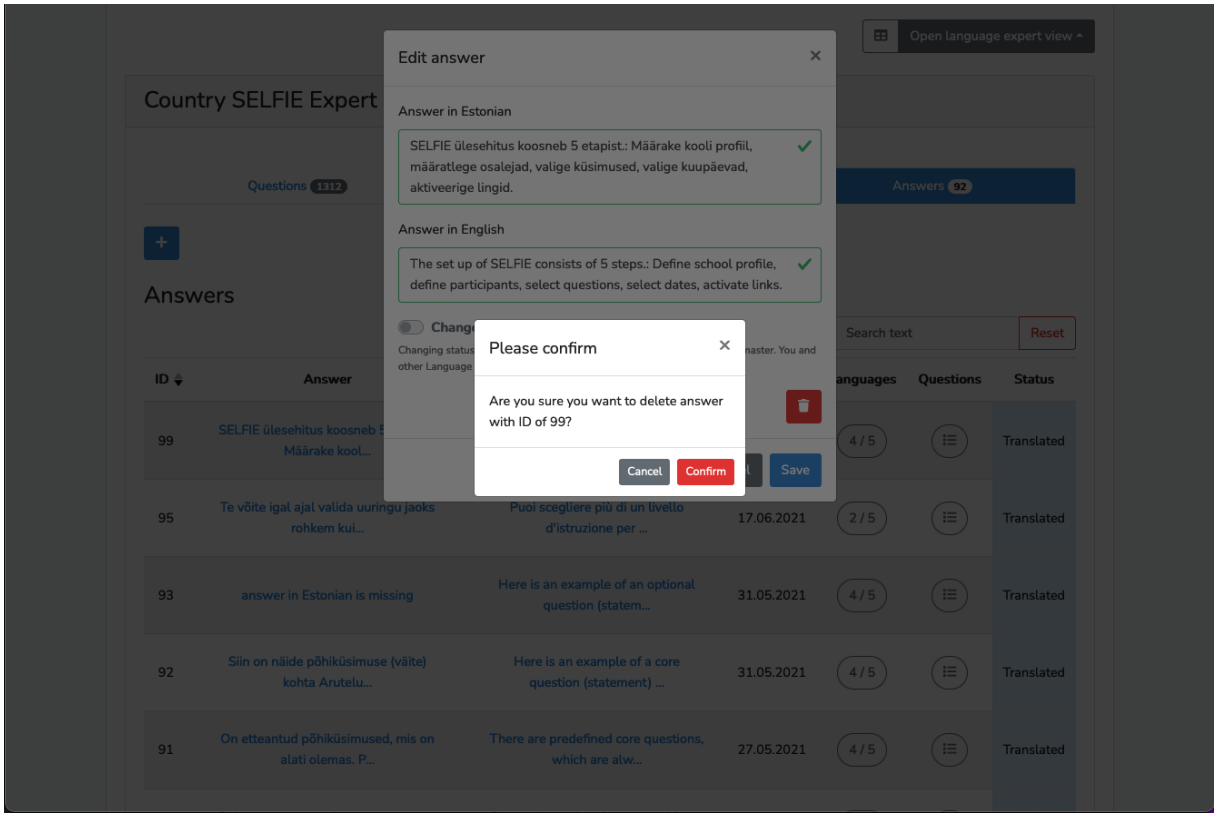


**Figure 15. The Country Selfie Expert deletes an existing answer**

The Country Selfie Expert can also view Pending Questions (Figure 16) and edit pending questions in order to provide translations (Figure 17). Pending Question status could be changed to Propagated and thus send it to Master Selfie Expert for review.

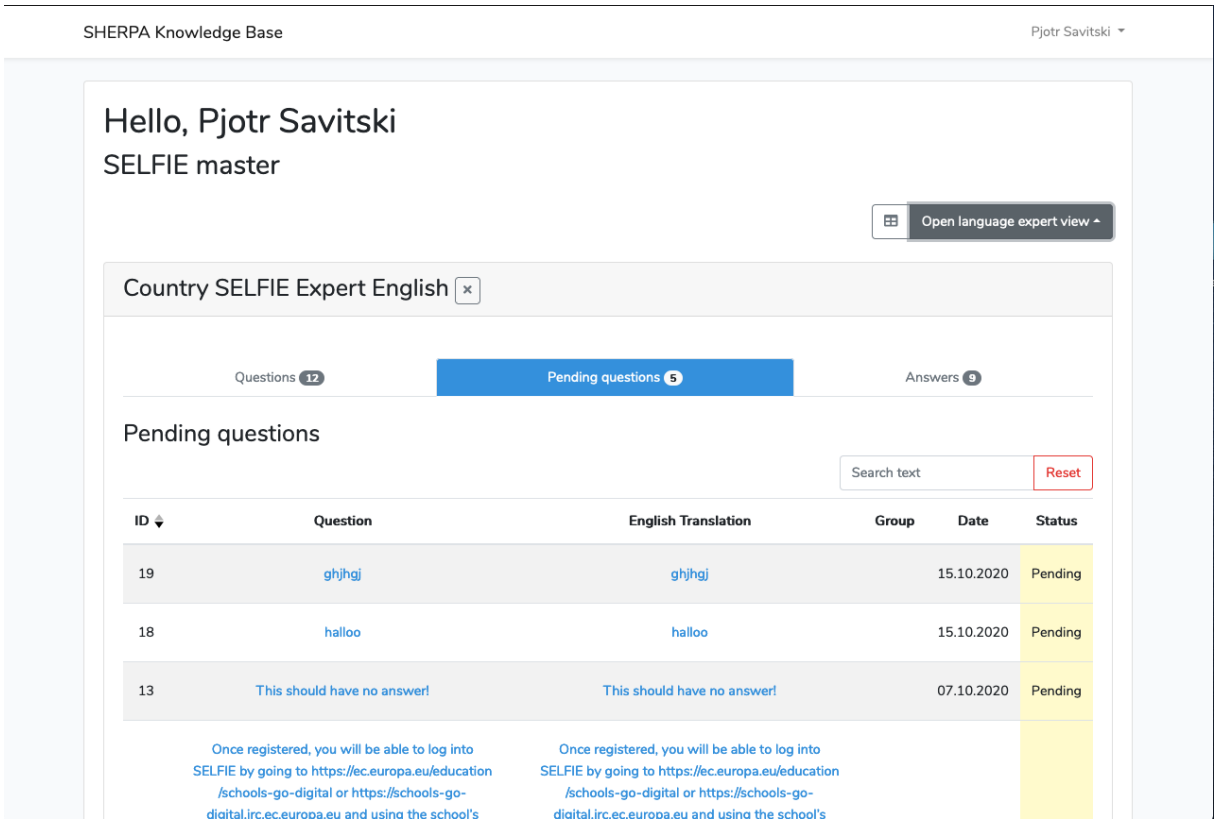


**Figure 16. Country Selfie Expert view for pending questions in English**

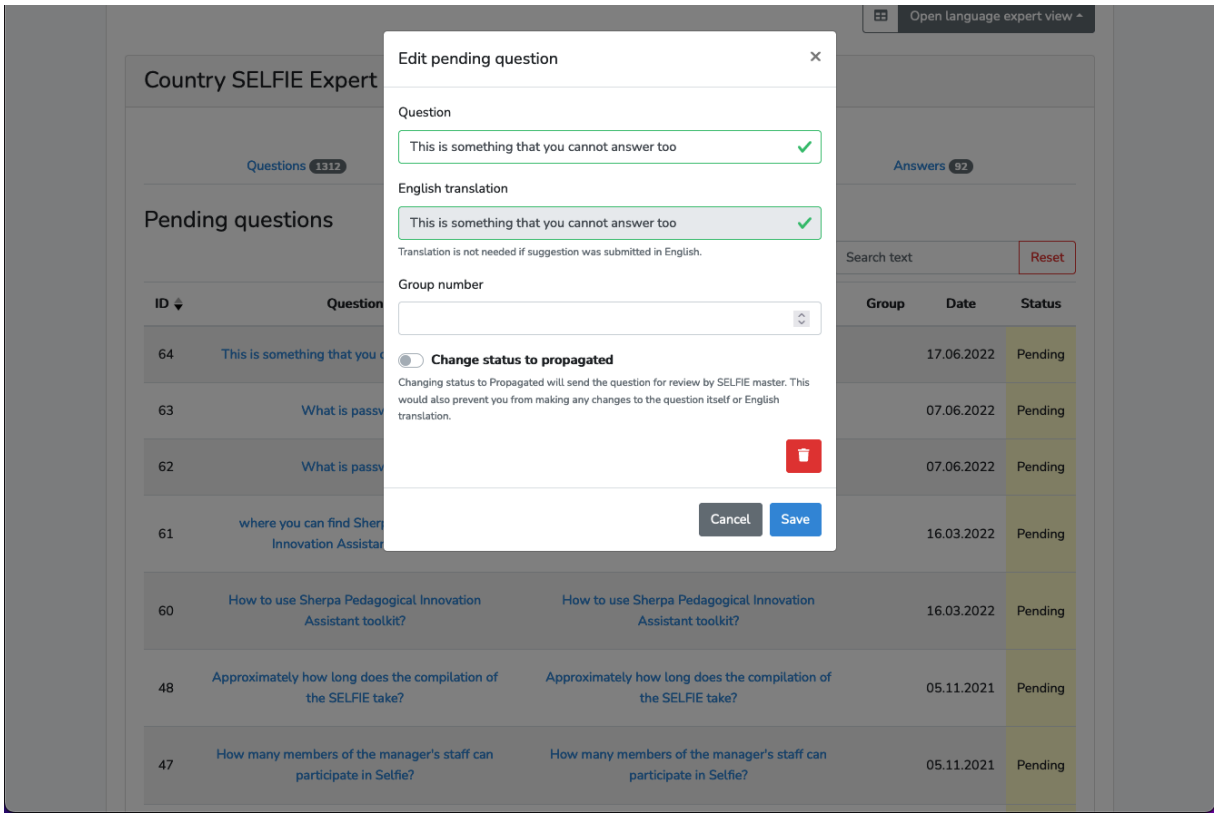


**Figure 17. The Country Selfie Expert edits a pending-question**

Master Selfie Expert can view the same content as a Country Selfie Expert (Figure 18), see a list of all questions (Figure 19) and answers (Figure 20) for a specific language and see a list of pending-questions for a specific language (Figure 23).

The listing view for Answers also contains a column (Figure 20) that allows one to view all the related Questions (Figure 21). This dialog allows editing any of those questions without leaving the view (Figure 22).

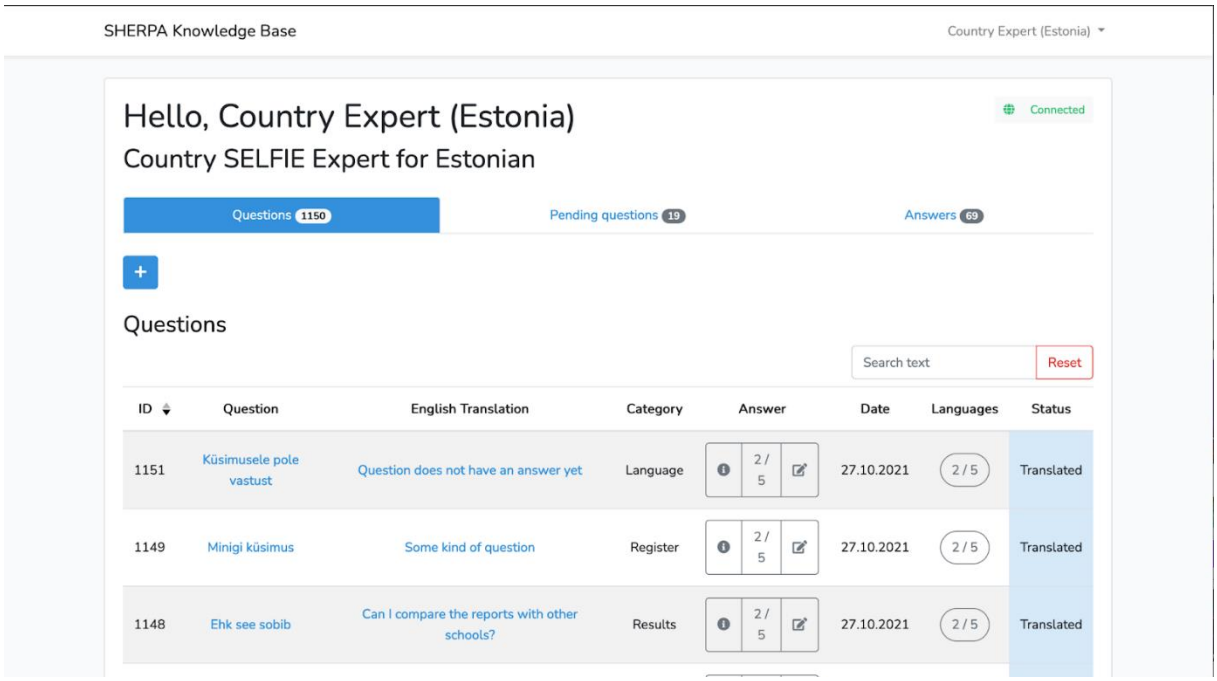


**Figure 18. Country Selfie Expert view accessed by Master Selfie Expert**

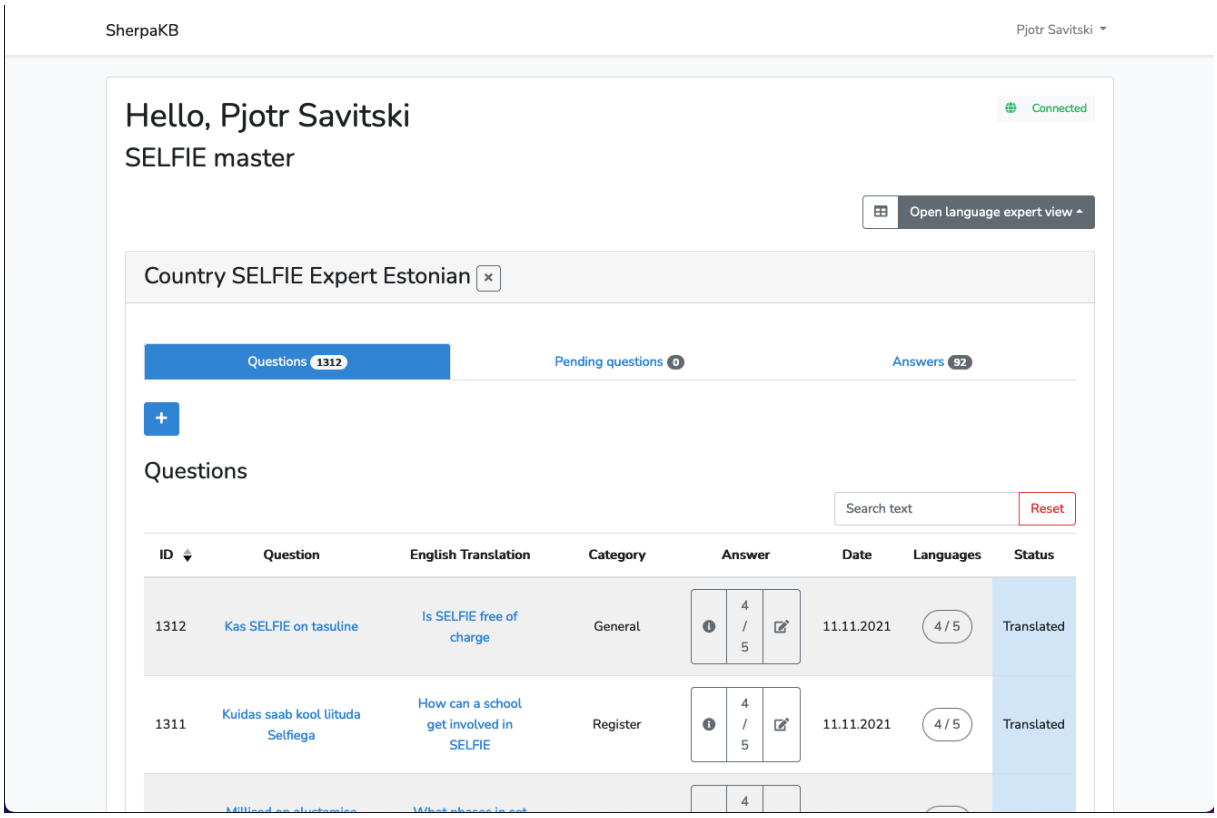


**Figure 19. The Master Selfie Expert view of questions (Country Selfie Expert view for Estonian)**

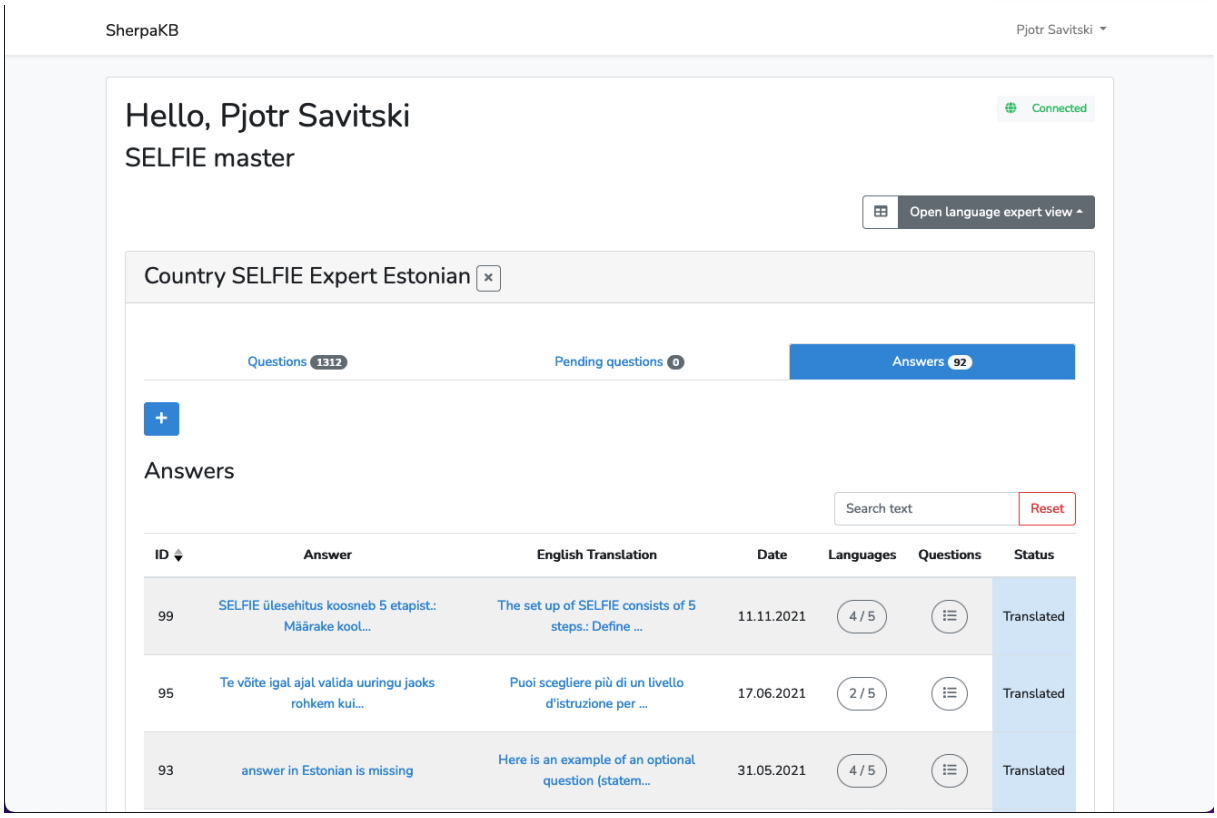


**Figure 20. Master Selfie Expert view of answers (Country Selfie Expert view for Estonian)**

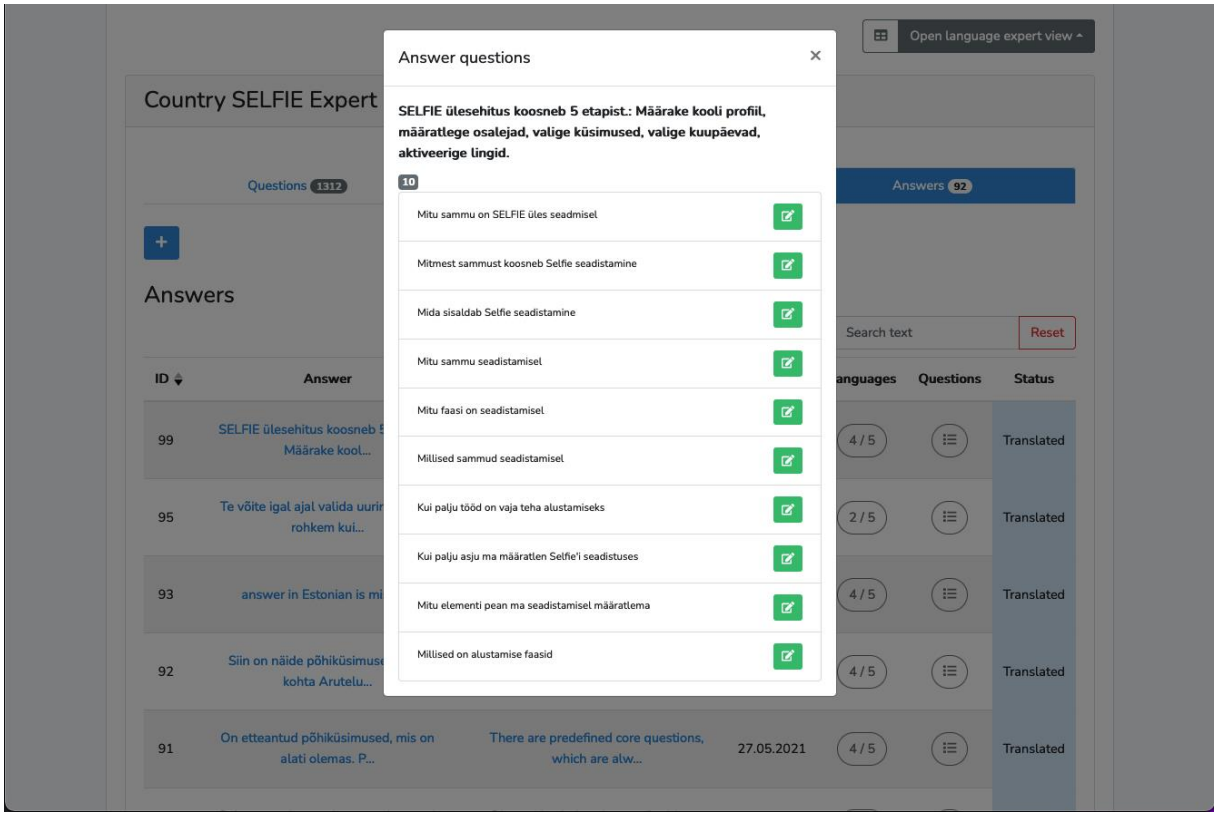


**Figure 21. List of all Questions related to a certain Answer**

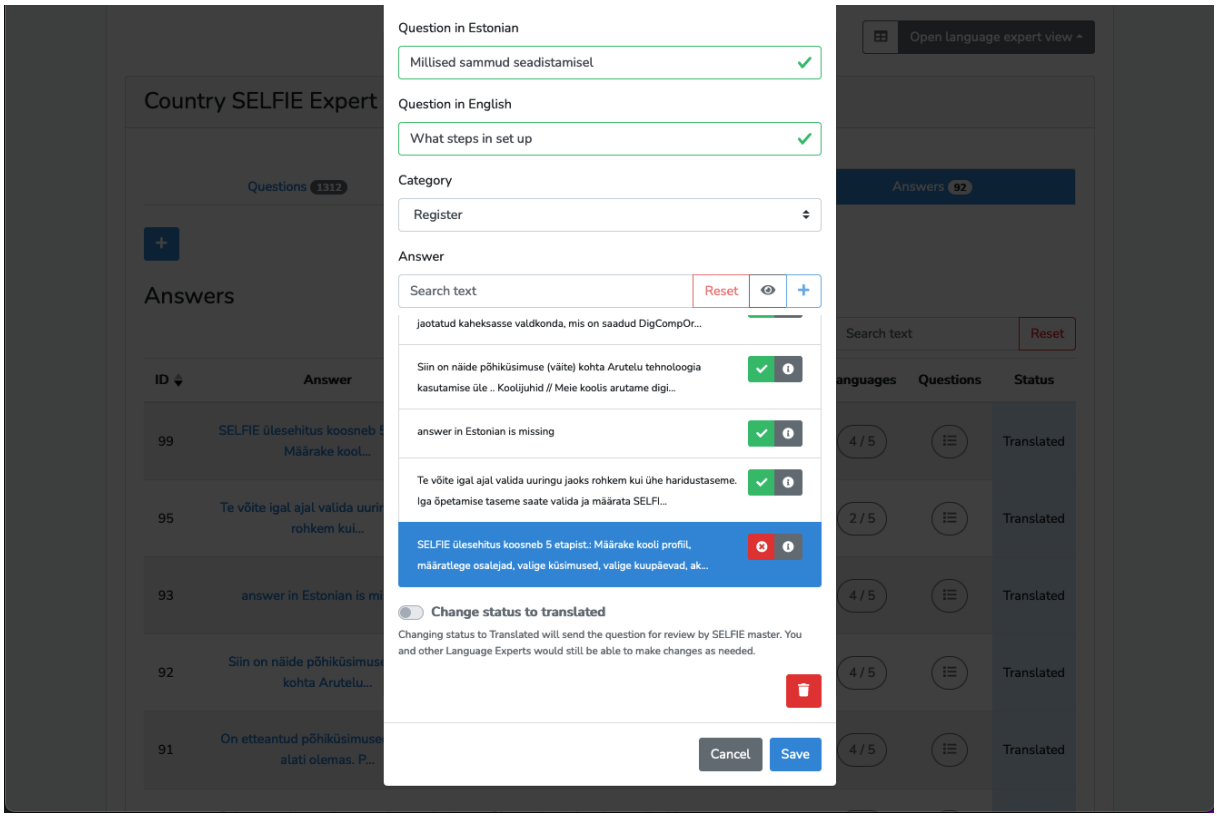


**Figure 22. Editing a question directly from the list of related questions**

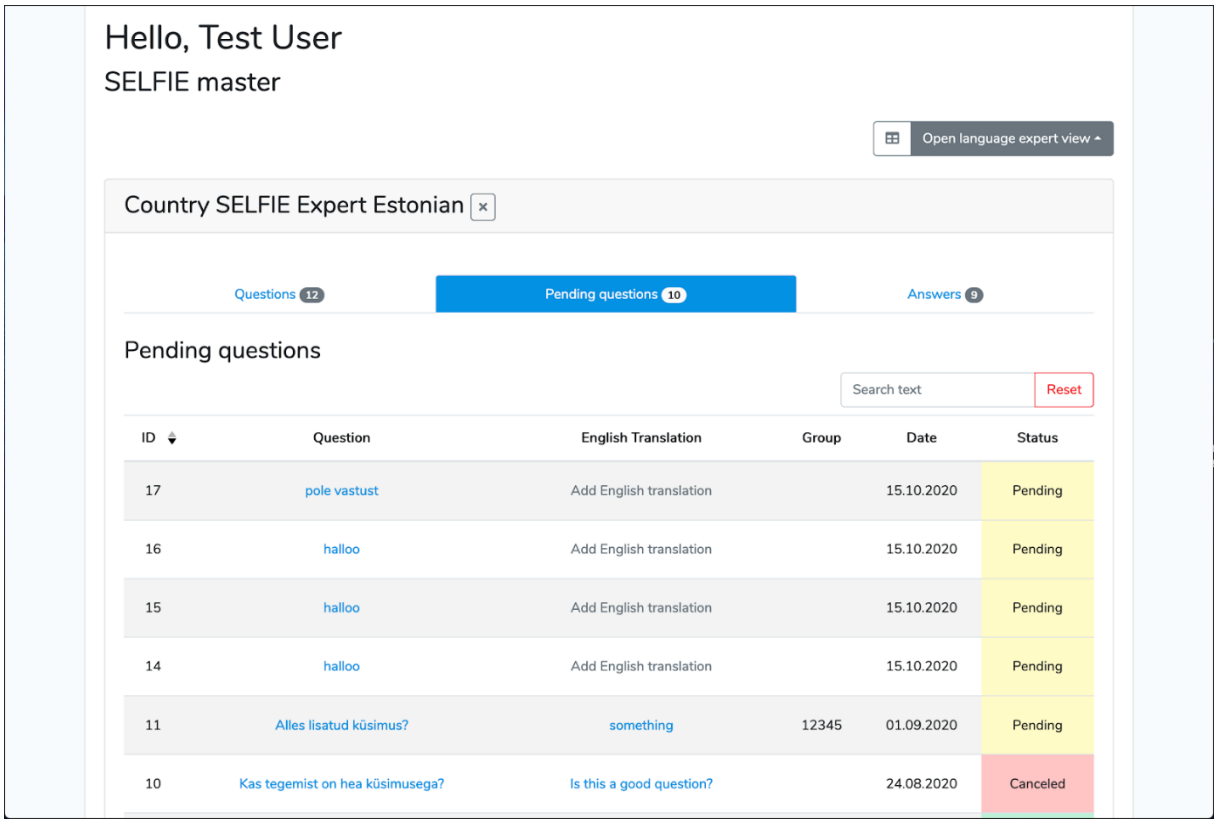


**Figure 23. Master Selfie Expert list of pending-questions (Country Selfie Expert view for Estonian)**

The Master Selfie Expert can also review Questions (Figure 24) and Answers (Figure 25), before those could be published. Finally, he or she can view (Figure 26) and review (Figure 27) all pending questions and access statistics data about the KB (Figure 28).

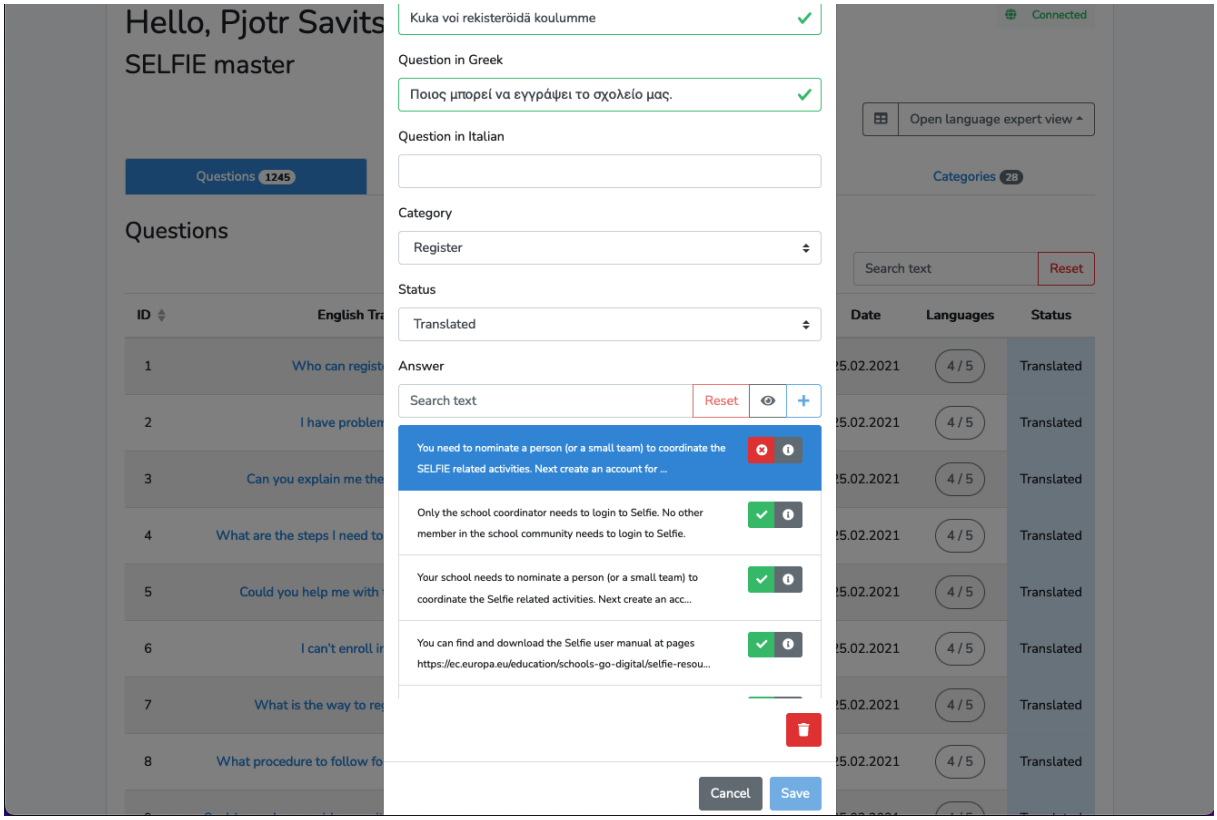


**Figure 24. The Master Selfie Expert reviews a question**

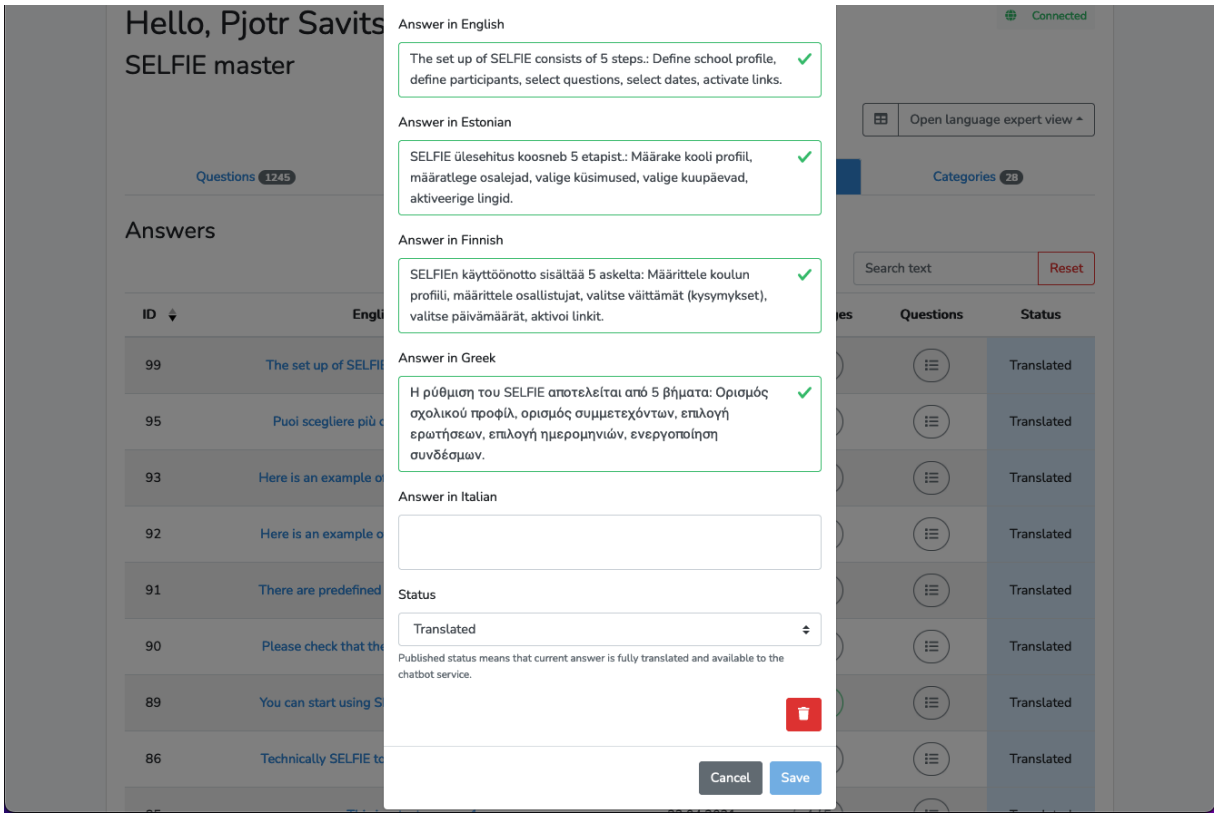


**Figure 25. The Master Selfie Expert reviews an answer**

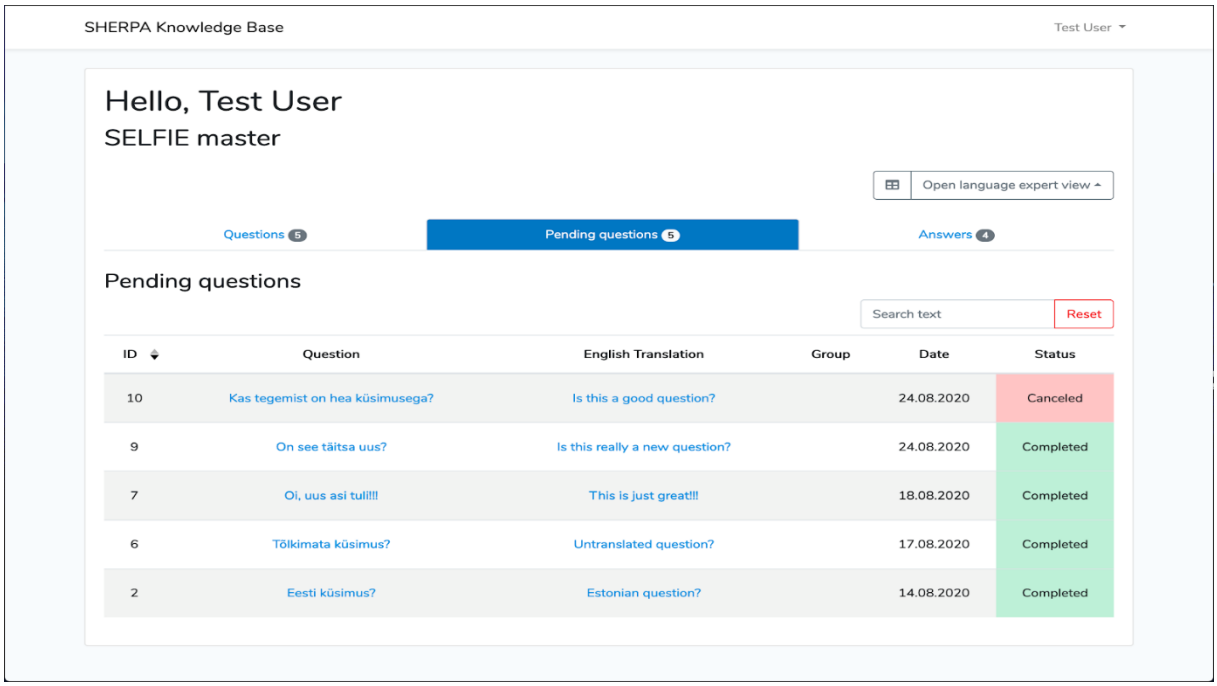


**Figure 26. The Master Selfie Expert views pending questions (propagated, canceled and completed)**

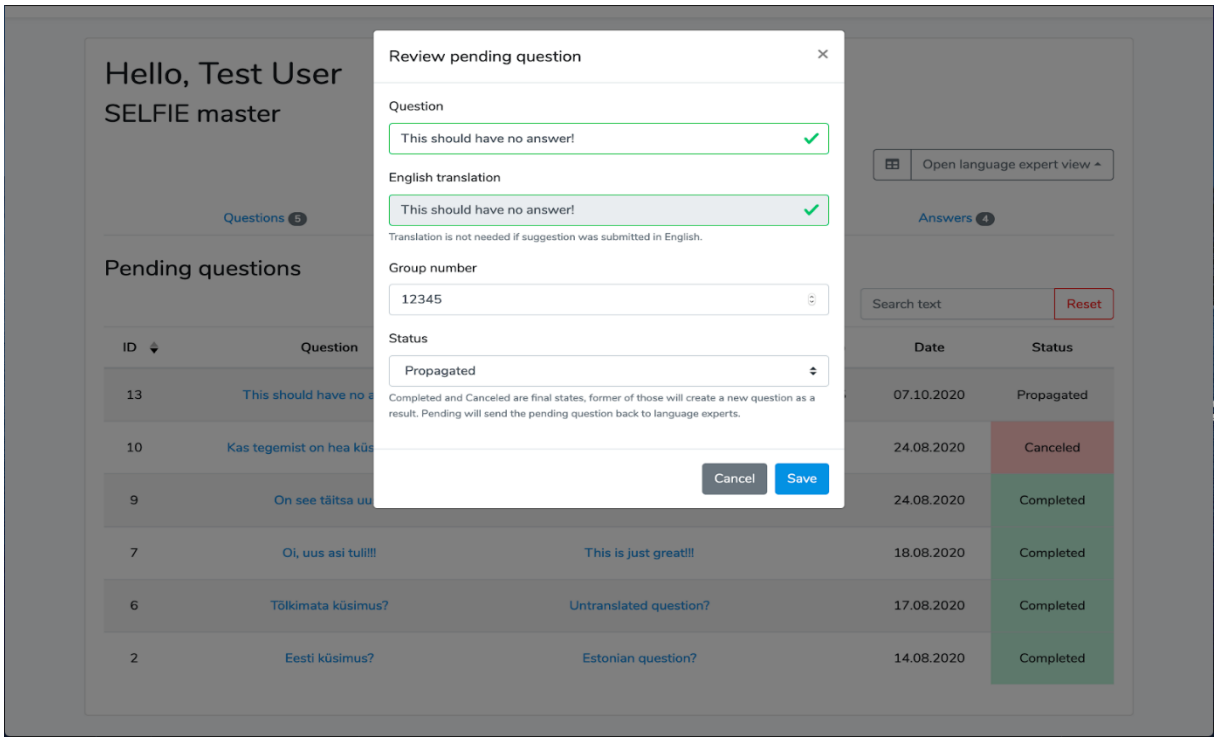


**Figure 27. The Master Selfie Expert reviews a pending question**

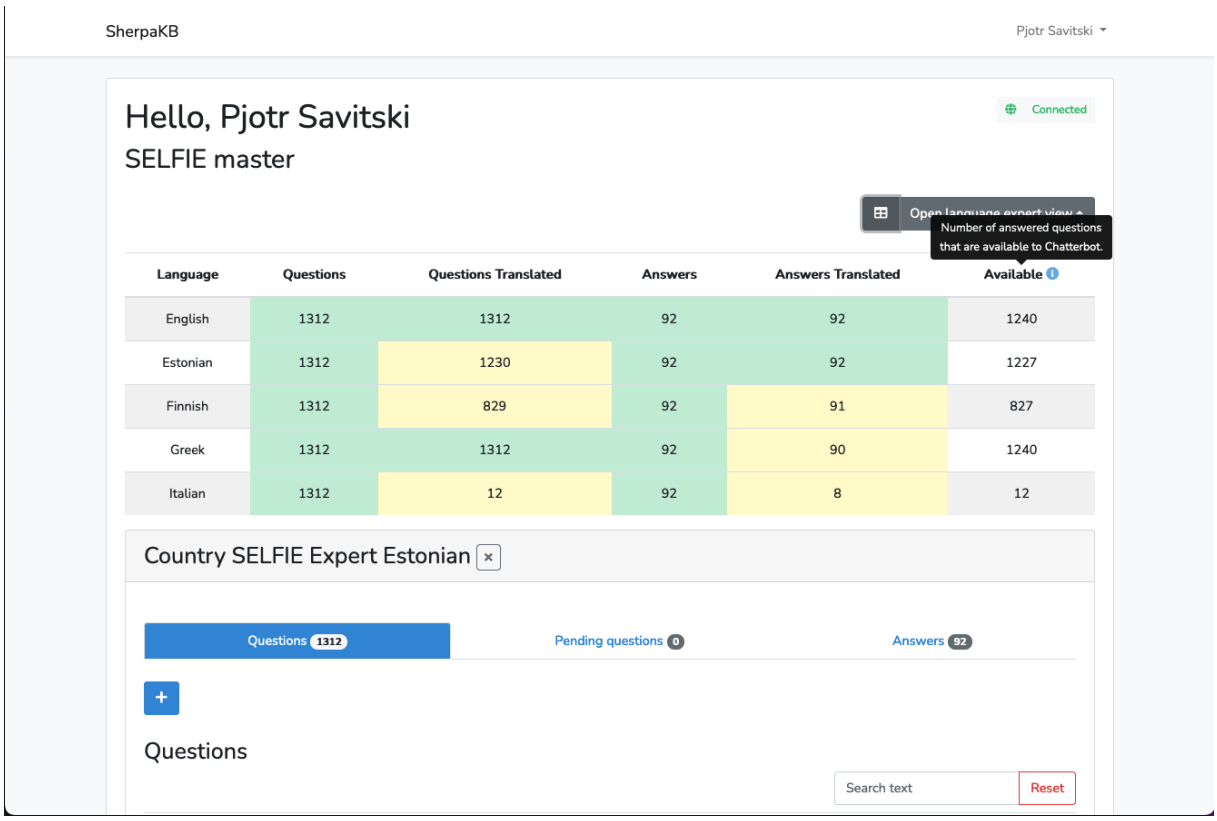


**Figure 28. Master Selfie Expert statistics page**

Master Selfie Experts are also allowed to access the categories management tab (Figure 29). Category management solution is quite simple and includes adding new (Figure 30), editing (Figure 31) and deleting (Figure 32) existing ones.

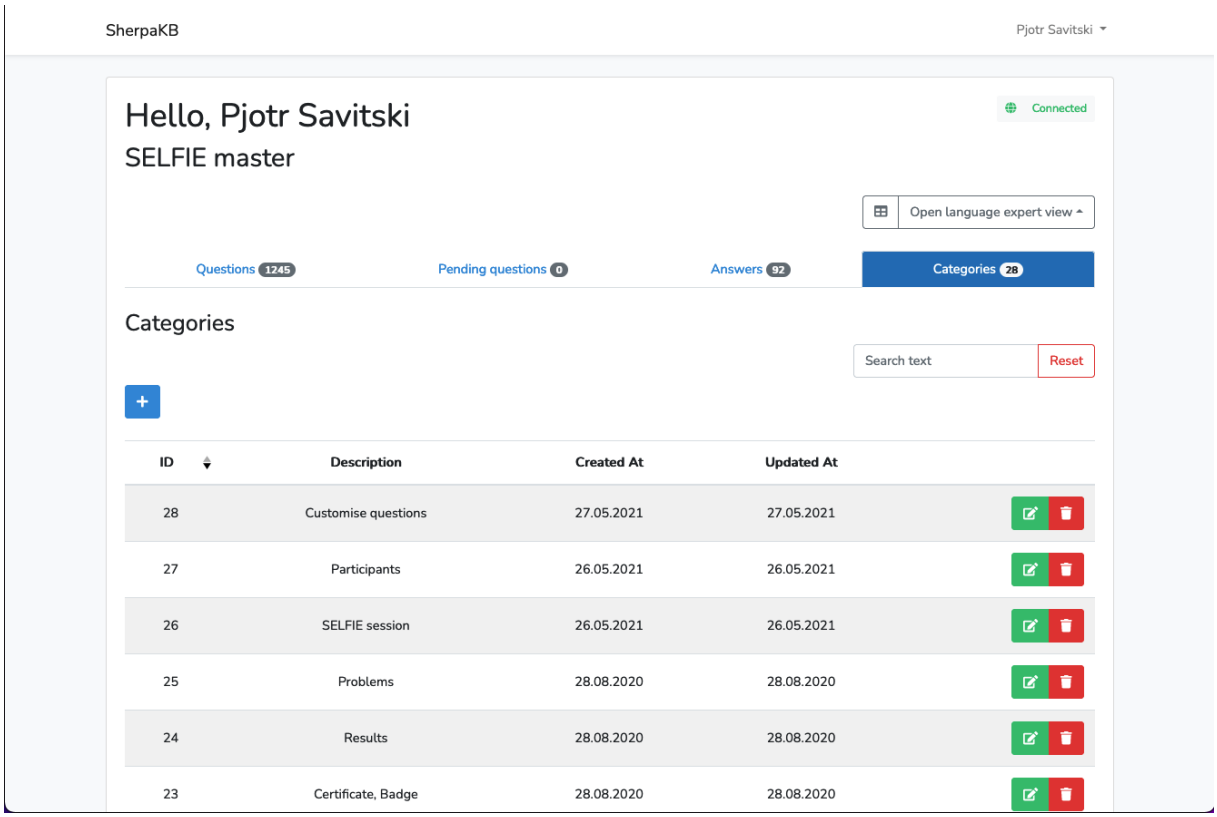


**Figure 29. Master Selfie Expert categories management page**

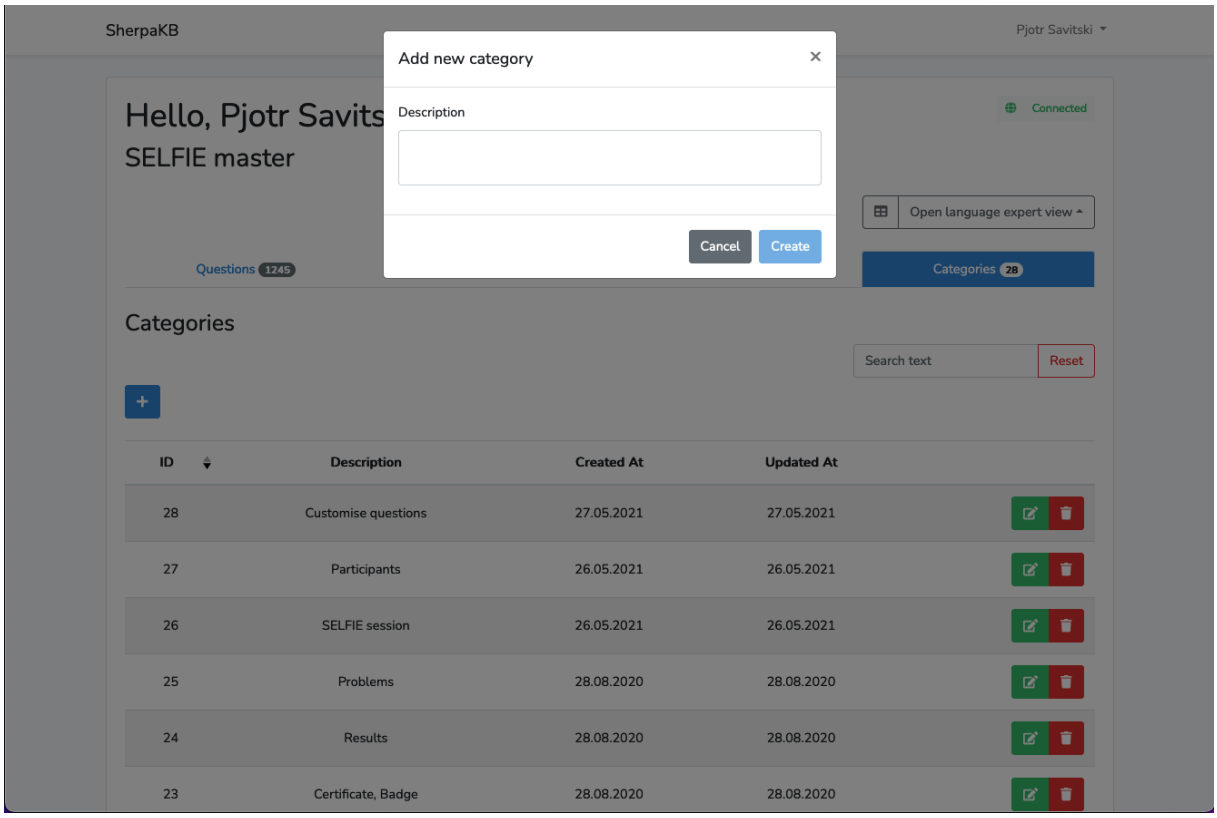


**Figure 30. Master Selfie Expert adding a new category**

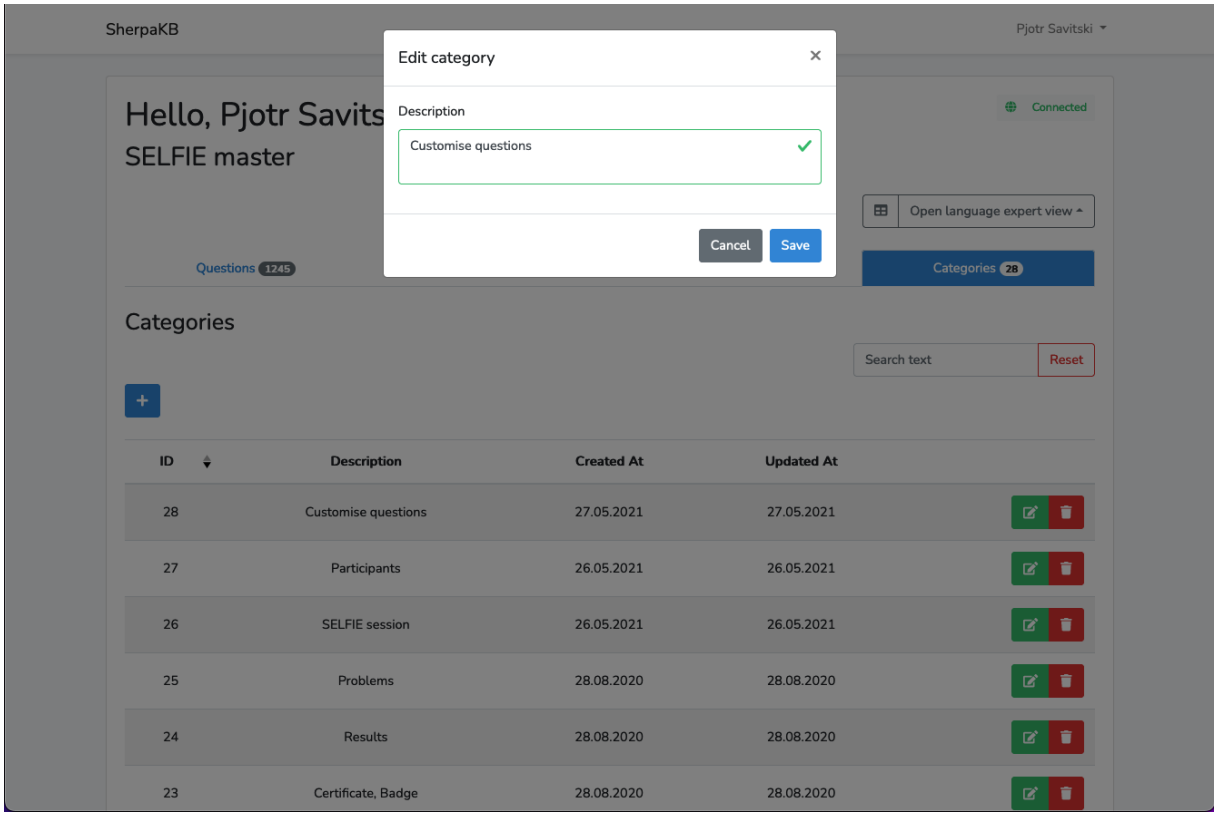


**Figure 31. Master Selfie Expert editing an existing category**

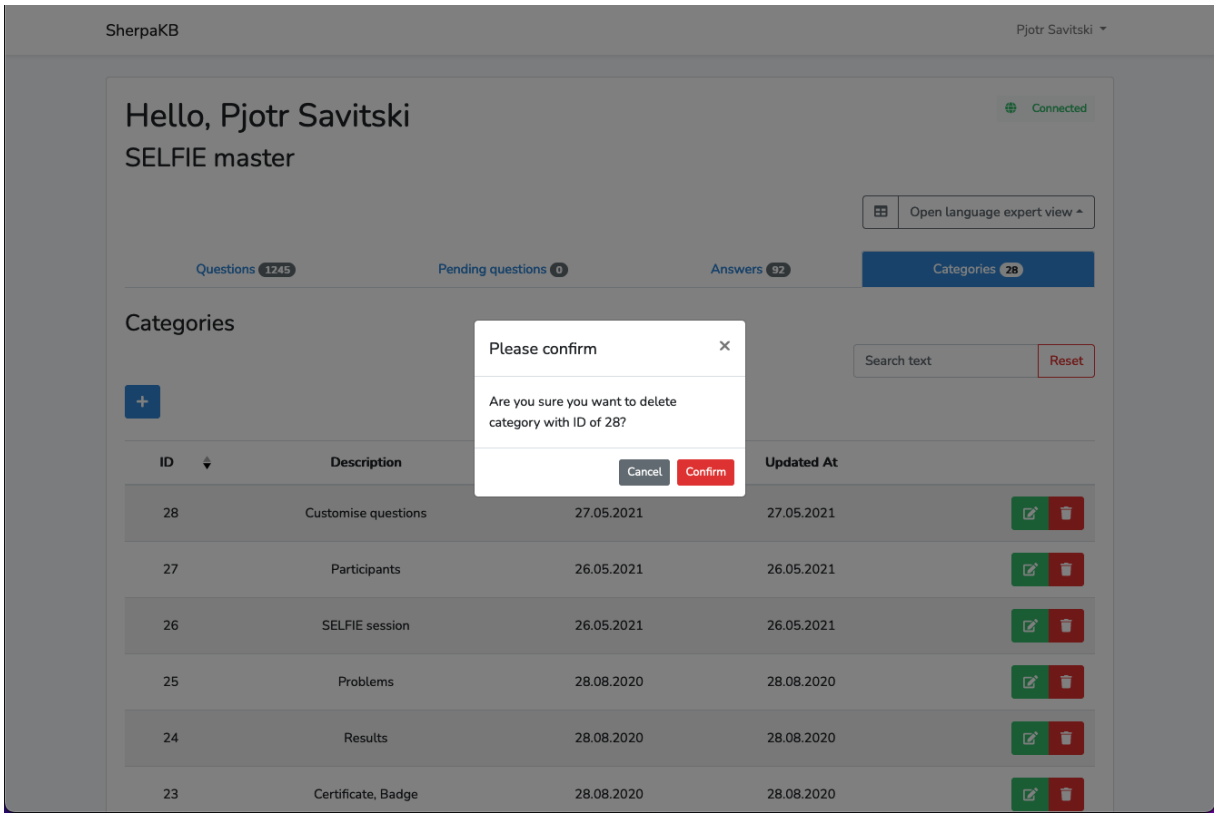


**Figure 32. Master Selfie Expert deleting an existing category**

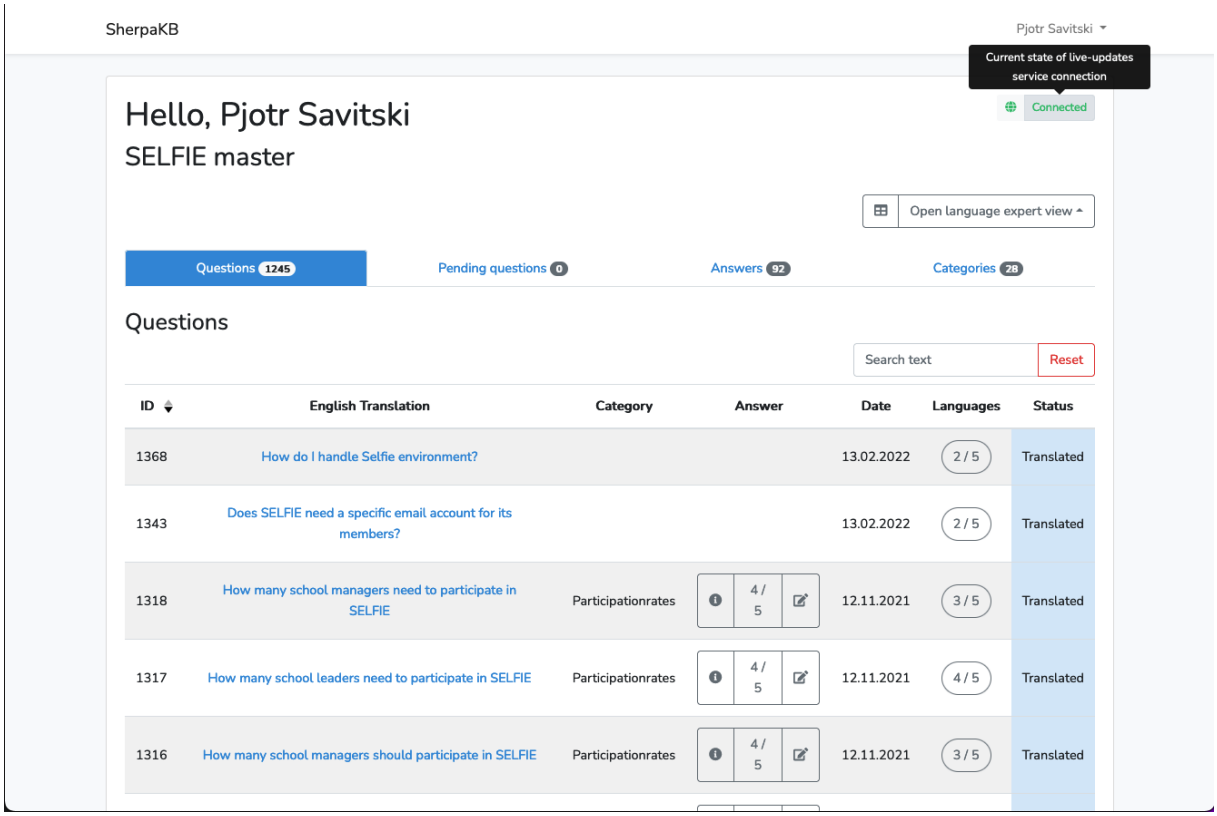


**Figure 33. Real-time updates for all the data**

Application includes capability of using real-time updates, allowing different users to get the latest data without having to refresh the browser (Figure 33). This is a separately configurable service, which is optional yet strongly suggested. In case the real-time updates service configuration is missing, the interface will just show the refresh button that would allow the user to fetch the fresh copy of the data from the server. This is just a fallback solution and administrators are highly discouraged from using the service without that feature enabled.
Further technical details are provided in the SHERPA KB GitHub repository (SHERPA TEAM, 2020b). Please make sure that you use the production (SHERPA TEAM, 2020c) branch outside of development as that contains static assets that have been specifically built for use in a live environment.

# 5. Reference List


Bootstrap (2020). Build fast, responsive sites with Bootstrap, Retrieved from https://getbootstrap.com/

BootstrapVue (2020). Retrieved from https://bootstrap-vue.org/

Cox, G. (2019). ChatterBot: Machine learning, conversational dialog engine, Retrieved from https://chatterbot.readthedocs.io/en/stable/

EELLAK (2020). Open Chatbot GitHub repository, Retrieved from https://github.com/eellak/openchatbot

Docker (2022). Docker makes development efficient and predictable, Retrieved from https://www.docker.com/

nginx (2022). nginx [engine x] is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server, originally written by Igor Sysoev, Retrieved from https://nginx.org/en/

Docker Compose (2022). Compose is a tool for defining and running multi-container Docker applications, Retrieved from https://docs.docker.com/compose/

Laravel (2020). Laravel: The PHP Framework for Web Artisans, Retrieved from https://laravel.com/

Laravel documentation (2022). Official documentation for the Laravel framework, Retrieved from https://laravel.com/docs/8.x/

Microsoft (2020). TypeScript, Retrieved from https://www.typescriptlang.org/

SHERPA TEAM (2020a). SELFIE Helper GitHub Repository, Retrieved from https://github.com/centre-for-educational-technology/sherpa-helper/tree/master/

SHERPA TEAM (2020b). SHERPA Knowledge Base GitHub Repository, Retrieved from https://github.com/centre-for-educational-technology/sherpa-kb/tree/master

SHERPA TEAM (2020c). SHERPA Knowledge Base GitHub Repository, Retrieved from https://github.com/centre-for-educational-technology/sherpa-kb/tree/production

SHERPA TEAM (2020d). SHERPA CBR Inference Engine GitLab Repository, Retrieved from https://gitlab.com/aetma/sherpa-project/sherpa_inference_engine

SHERPA TEAM (2020e). SHERPA Project Aggregation point GitLab Repository, Retrieved from https://gitlab.com/aetma/sherpa-project

Vue CLI (2020) Retrieved from https://cli.vuejs.org/

Vue.jd (2020) Retrieved from https://vuejs.org/
Wikipedia (2020a). JavaScript, Retrieved from https://en.wikipedia.org/wiki/JavaScript
Wikipedia (2020b). Levenshtein distance, Retrieved from https://en.wikipedia.org/wiki/Levenshtein_distance
Wikipedia (2020c). FastText, Retrieved from https://en.wikipedia.org/wiki/FastText
Wikipedia (2020d). Single-page application, Retrieved from https://en.wikipedia.org/wiki/Single-page_application

# 6. List of Abbreviations

**Table 1. List of Abbreviations**

| Abbreviation | Meaning |
|---|---|
| AETMA Lab - IHU | Advanced Educational Technologies and Mobile Applications Lab of International Hellenic University |
| API | Application Programming Interface |
| CBR | Case Based Reasoning |
| CNR-ITD | National Research Council of Italy, Institute for Educational Technology |
| CPI | Cyprus Pedagogical Institute |
| D | Deliverable |
| EC | European Commission |
| JYU | University of Jyvaskyla |
| KA | Key Action |
| KB | Knowledge Base |
| M | Month |
| P | Partner |
| Q&A | Questions and Answers |
| REST | Representational State Transfer |
| SHERPA | SELFIE HElpeR & Pedagogical innovation Assistant |
| TLU | Tallinn University |
| UI | User Interface |
| WP | Work Package |

# Appendix A: Open Source Code and documentation

## 1. Code for generating the KB schema

```
# ************************************************************
# Sequel Ace SQL dump
# Version 20033
#
# https://sequel-ace.com/
# https://github.com/Sequel-Ace/Sequel-Ace
#
# Host: 127.0.0.1 (MySQL 8.0.23)
# Database: sherpakb
# Generation Time: 2022-06-21 13:50:53 +0000
# ************************************************************



/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
SET NAMES utf8mb4;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE='NO_AUTO_VALUE_ON_ZERO',
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;



# Dump of table activity_log
# ------------------------------------------------------------

DROP TABLE IF EXISTS `activity_log`;

CREATE TABLE `activity_log` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `log_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFAULT
NULL,
  `description` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `subject_type` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
DEFAULT NULL,
  `subject_id` bigint unsigned DEFAULT NULL,
  `causer_type` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
DEFAULT NULL,
  `causer_id` bigint unsigned DEFAULT NULL,
  `properties` json DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `subject` (`subject_type`,`subject_id`),
  KEY `causer` (`causer_type`,`causer_id`),
  KEY `activity_log_log_name_index` (`log_name`)
) ENGINE=InnoDB AUTO_INCREMENT=6329 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;




# Dump of table answer_language
# ------------------------------------------------------------

DROP TABLE IF EXISTS `answer_language`;

CREATE TABLE `answer_language` (
```

```
  `answer_id` bigint unsigned NOT NULL,
  `language_id` bigint unsigned NOT NULL,
  `description` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`answer_id`,`language_id`),
  KEY `answer_language_language_id_foreign` (`language_id`),
  CONSTRAINT `answer_language_answer_id_foreign` FOREIGN KEY (`answer_id`)
REFERENCES `answers` (`id`) ON DELETE CASCADE,
  CONSTRAINT `answer_language_language_id_foreign` FOREIGN KEY (`language_id`)
REFERENCES `languages` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;




# Dump of table answers
# ------------------------------------------------------------

DROP TABLE IF EXISTS `answers`;

CREATE TABLE `answers` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `status` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=70 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;




# Dump of table failed_jobs
# ------------------------------------------------------------

DROP TABLE IF EXISTS `failed_jobs`;

CREATE TABLE `failed_jobs` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `uuid` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFAULT '',
  `connection` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `queue` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `payload` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `exception` longtext CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `failed_at` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `failed_jobs_uuid_unique` (`uuid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;




# Dump of table helper_activity_log
# ------------------------------------------------------------

DROP TABLE IF EXISTS `helper_activity_log`;

CREATE TABLE `helper_activity_log` (
  `question` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `answer` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `language_code` varchar(5) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
```

```
  `ip` varchar(45) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` timestamp NOT NULL,
  KEY `helper_activity_log_language_code_index` (`language_code`),
  KEY `helper_activity_log_created_at_index` (`created_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;



# Dump of table helper_response_user_ratings
# ------------------------------------------------------------

DROP TABLE IF EXISTS `helper_response_user_ratings`;

CREATE TABLE `helper_response_user_ratings` (
  `question` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `answer` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `language_code` varchar(5) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  `rating` tinyint unsigned NOT NULL,
  `ip` varchar(45) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` timestamp NOT NULL,
  KEY `helper_response_user_ratings_language_code_index` (`language_code`),
  KEY `helper_response_user_ratings_rating_index` (`rating`),
  KEY `helper_response_user_ratings_created_at_index` (`created_at`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;



# Dump of table language_pending_question
# ------------------------------------------------------------

DROP TABLE IF EXISTS `language_pending_question`;

CREATE TABLE `language_pending_question` (
  `pending_question_id` bigint unsigned NOT NULL,
  `language_id` bigint unsigned NOT NULL,
  `description` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`pending_question_id`,`language_id`),
  KEY `language_pending_question_language_id_foreign` (`language_id`),
  CONSTRAINT `language_pending_question_language_id_foreign` FOREIGN KEY
(`language_id`) REFERENCES `languages` (`id`) ON DELETE CASCADE,
  CONSTRAINT `language_pending_question_pending_question_id_foreign` FOREIGN KEY
(`pending_question_id`) REFERENCES `pending_questions` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;



# Dump of table language_question
# ------------------------------------------------------------

DROP TABLE IF EXISTS `language_question`;

CREATE TABLE `language_question` (
  `question_id` bigint unsigned NOT NULL,
  `language_id` bigint unsigned NOT NULL,
  `description` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
```

```
 PRIMARY KEY (`question_id`,`language_id`),
 KEY `language_question_language_id_foreign` (`language_id`),
 CONSTRAINT `language_question_language_id_foreign` FOREIGN KEY (`language_id`)
REFERENCES `languages` (`id`) ON DELETE CASCADE,
 CONSTRAINT `language_question_question_id_foreign` FOREIGN KEY (`question_id`)
REFERENCES `questions` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;




# Dump of table languages
# ------------------------------------------------------------

DROP TABLE IF EXISTS `languages`;

CREATE TABLE `languages` (
 `id` bigint unsigned NOT NULL AUTO_INCREMENT,
 `code` varchar(5) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
 `name` varchar(30) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
 `created_at` timestamp NULL DEFAULT NULL,
 `updated_at` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`id`),
 UNIQUE KEY `languages_code_unique` (`code`),
 UNIQUE KEY `languages_name_unique` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;




# Dump of table migrations
# ------------------------------------------------------------

DROP TABLE IF EXISTS `migrations`;

CREATE TABLE `migrations` (
 `id` int unsigned NOT NULL AUTO_INCREMENT,
 `migration` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
 `batch` int NOT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=50 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;




# Dump of table model_has_permissions
# ------------------------------------------------------------

DROP TABLE IF EXISTS `model_has_permissions`;

CREATE TABLE `model_has_permissions` (
 `permission_id` bigint unsigned NOT NULL,
 `model_type` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
 `model_id` bigint unsigned NOT NULL,
 PRIMARY KEY (`permission_id`,`model_id`,`model_type`),
 KEY `model_has_permissions_model_id_model_type_index` (`model_id`,`model_type`),
 CONSTRAINT `model_has_permissions_permission_id_foreign` FOREIGN KEY
(`permission_id`) REFERENCES `permissions` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
# Dump of table model_has_roles
# ------------------------------------------------------------

DROP TABLE IF EXISTS `model_has_roles`;

CREATE TABLE `model_has_roles` (
  `role_id` bigint unsigned NOT NULL,
  `model_type` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  `model_id` bigint unsigned NOT NULL,
  PRIMARY KEY (`role_id`,`model_id`,`model_type`),
  KEY `model_has_roles_model_id_model_type_index` (`model_id`,`model_type`),
  CONSTRAINT `model_has_roles_role_id_foreign` FOREIGN KEY (`role_id`) REFERENCES
`roles` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;



# Dump of table password_resets
# ------------------------------------------------------------

DROP TABLE IF EXISTS `password_resets`;

CREATE TABLE `password_resets` (
  `email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `token` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  KEY `password_resets_email_index` (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;



# Dump of table pending_questions
# ------------------------------------------------------------

DROP TABLE IF EXISTS `pending_questions`;

CREATE TABLE `pending_questions` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `status` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `group_no` int DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=31 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;



# Dump of table permissions
# ------------------------------------------------------------

DROP TABLE IF EXISTS `permissions`;

CREATE TABLE `permissions` (
```

```
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `guard_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;




# Dump of table questions
# ------------------------------------------------------------

DROP TABLE IF EXISTS `questions`;

CREATE TABLE `questions` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `status` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `topic_id` bigint unsigned DEFAULT NULL,
  `answer_id` bigint unsigned DEFAULT NULL,
  `pending_question_id` bigint unsigned DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `questions_topic_id_foreign` (`topic_id`),
  KEY `questions_answer_id_foreign` (`answer_id`),
  KEY `questions_pending_question_id_foreign` (`pending_question_id`),
  CONSTRAINT `questions_answer_id_foreign` FOREIGN KEY (`answer_id`) REFERENCES
`answers` (`id`) ON DELETE SET NULL,
  CONSTRAINT `questions_pending_question_id_foreign` FOREIGN KEY
(`pending_question_id`) REFERENCES `pending_questions` (`id`) ON DELETE SET NULL,
  CONSTRAINT `questions_topic_id_foreign` FOREIGN KEY (`topic_id`) REFERENCES
`topics` (`id`) ON DELETE SET NULL
) ENGINE=InnoDB AUTO_INCREMENT=1152 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;




# Dump of table role_has_permissions
# ------------------------------------------------------------

DROP TABLE IF EXISTS `role_has_permissions`;

CREATE TABLE `role_has_permissions` (
  `permission_id` bigint unsigned NOT NULL,
  `role_id` bigint unsigned NOT NULL,
  PRIMARY KEY (`permission_id`,`role_id`),
  KEY `role_has_permissions_role_id_foreign` (`role_id`),
  CONSTRAINT `role_has_permissions_permission_id_foreign` FOREIGN KEY
(`permission_id`) REFERENCES `permissions` (`id`) ON DELETE CASCADE,
  CONSTRAINT `role_has_permissions_role_id_foreign` FOREIGN KEY (`role_id`)
REFERENCES `roles` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;




# Dump of table roles
# ------------------------------------------------------------
```

```
DROP TABLE IF EXISTS `roles`;

CREATE TABLE `roles` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `guard_name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;



# Dump of table topics
# ------------------------------------------------------------

DROP TABLE IF EXISTS `topics`;

CREATE TABLE `topics` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `description` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=53 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;



# Dump of table users
# ------------------------------------------------------------

DROP TABLE IF EXISTS `users`;

CREATE TABLE `users` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `email_verified_at` timestamp NULL DEFAULT NULL,
  `password` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  `remember_token` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
DEFAULT NULL,
  `language_id` bigint unsigned DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `users_email_unique` (`email`),
  KEY `users_language_id_foreign` (`language_id`),
  CONSTRAINT `users_language_id_foreign` FOREIGN KEY (`language_id`) REFERENCES
`languages` (`id`) ON DELETE SET NULL
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;



/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

```
/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

# 2. CBR Technical Documentation

The main part of the SHERPA Helper is the chatbot inference engine. Its main responsibility is answer selection to user's questions based on a predefined closed set of Q&A pairs. This document aims to describe such choices through analytical explanation and examples where necessary.

## 2.1. System Architecture

ChatBot implementation was based on ChatterBot platform, which is a machine-learning based conversational dialog engine built in Python which makes it possible to generate responses based on collections of known conversations. It uses a selection of machine learning algorithms to produce different types of responses.

The given input is provided using a REST API implementation via an endpoint. The user needs to provide the proper question on a string format as long as the language code. In our current implementation our multilingual system supports English, Greek , Italian, Estonian and Finish. One can ask a question using the following language codes: {en,gr,it,et,fi} respectively.

## 2.2. Code description

Our implementation is organized into packages based on their functionality. We explain each package and main purpose in the Table 1.

**Table 1. Description of ChatterBot packages**

| Package name | Description |
|---|---|
| config | It contains the app_config.ini file where the environment variables (local db host/port/password , external db api url etc ) are assigned values. |
| core | The directory contains the core logic of our application.<br>• adapters: This package contains two custom logic adapters, the fastTextLogiAdapter and the logicAdapter. As it described in the official documentation the logic adapters are used to determine the way we select an output to an input statement. So, the CustomLogicAdapter (logicAdapter.py) selects the response with the closest match using the levenshtein distance algorithm whereas the FastTextLogicAdapter (FastTextLogicAdapter.py) uses the proper fastText model based on the input language. In both cases we need to implement the process function with our custom logic.<br>• http: This package contains an HttpClient class which is used to communicate with an external db. It has a method per endpoint such as to retrieve all topics, languages etc.<br>• model: The folder contains two subfolders. The wordSimilarity subfolder contains a python class with the proper trainer. The **ChatterBotWordSimilarityTrainer** extends the base Trainer class and implements the train function with our own logic. So, we retrieve, from the external db, all question-answer pairs per language which are stored in our local db in the Statement format (described in the documentation). The **fastTextTrainer** extends the base Trainer class and implements the train function. In order to create a unique model per language with retrieve each language code from the external db. For every code, we create a model, response_data and train_data file as described in the files section below. Once we create the proper files we train the model with some custom parameters described in a dictionary named hyper_params. In case we need to modify the algorithm's accuracy we can change those parameters. Each one is described in an analytical way on the official fastText documentation. |
| files | This directory contains 3 subdirectories which are used by the fastText algorithm. |

| | |
|---|---|
| | • models: Once the fastText algorithm is executed this folder contains all .bin files where all models are described.<br>• response_data: The folder contains a json file per language. Each json is a dictionary with (label, answer) pairs.<br>• train_data: The folder contains one txt file per language. Each row contains a (label,question) pair. We need to mention that a label contains many questions but only an answer. |

In the above Table 1 we described all the internal python classes. Our core logic is orchestrated by an app.py file which is capable of reading the external configuration from app_config.ini file, to initiate the ChatBot engine and to provide the REST endpoint for all external users, such as an API.
The endpoint description is presented in Table 2.

**Table 2. Endpoint Description**

| Endpoint name | Description |
|---|---|
| /applyQuestion | The endpoint gets a json input containing the question and language code. Then, it executes the generaResponse method which is capable of running the chatbot and selects the answer with the higher accuracy score. |

## 2.3. Code location

The application code is located in AETMA lab github account under SHERPA project directory and can be accessed through the GitLab link.

## 2.4. Code execution

The application can be executed both as a native python app or via docker. In the following sections we will show in an analytical way the commands we need to use for both execution types.

### 2.4.1. Native python application execution

```
foo@bar:~ git clone https://gitlab.com/aetma/sherpa-project/sherpa_inference_engine.git
foo@bar:~ cd sherpa_inference_engine
foo@bar:~ pip install -r requirements.txt
foo@bar:~ python app.py
```

Once the execution starts the logs are shown on the command line. The requirements file contains all the application dependencies regarding the external and internal services needed to deploy the app. Such services are a mysql connector, a restful engine, the fastText library, the chatterbot core code as well as json libs.

### 2.4.2. Docker environment execution

Another way to execute the application is the use of the docker environment. In the following lines we show the instructions needed to install docker on a native Ubuntu machine and how to deploy the application.

```
foo@bar:~ sudo apt-get update
foo@bar:~ sudo apt-get install \
            apt-transport-https \
            ca-certificates \
            curl \
            gnupg \
            lsb-release
foo@bar:~ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
foo@bar:~ echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
foo@bar:~ sudo apt-get update
foo@bar:~ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Once the installation is completed we can use the following command to deploy our app.

```
foo@bar:~ docker-compose build && docker-compose up -d
```

Finally, the logs can be accessed through the next command.

```
foo@bar:~ docker logs -f chatbot
```

## 2.5. Use case scenarios

In the following section we will describe two use case scenarios along with the system logs. The first scenario will contain a question from the training set whereas the second one will have a new one. Our goal is to clarify the way the system selects the most proper answer based on the algorithm with the highest score.

### 2.5.1. Predefined question

```
Question (cURL format)


curl -X POST "http://helper.sherpa4selfie.eu:5000/chatterbot/applyQuestion"
-H "accept: application/json"
-H "Content-Type: application/json"
-d "{
      \"text\": \"What is a school profile?\",
      \"languageCode\": \"en\"
    }"


Answer


[
  "A school profile contains the name and contact information of the school, and
also how the school is operated and funded. School profile will be defined after
logging into Selfie, and should be checked at least once a year."
]
```

In the following table, via the system logs, we can see the process of choosing the correct answer. As we have already mentioned the system uses two approaches, the closest match and the fastText algorithm. The system will try to find the statement that mostly matches the input questions. We can see that using MyLogicAdapter the system chooses the statement with a confidence score 0.98. Using the FastTextLogicAdapter the system chooses the statement with score 0.99 so finally the response comes from the algorithm with the highest confidence value score.

```
INFO:chatterbot.chatterbot: Beginning search for close text match
INFO:chatterbot.chatterbot: Processing search results
INFO:chatterbot.chatterbot: Similar text found:
Only the school coordinator needs to login to Selfie. No other member in the school community needs to login to Selfie. 0.21
INFO:chatterbot.chatterbot: Similar text found: I don't have a school account! Can I sign up with my mail account? 0.35
INFO:chatterbot.chatterbot: Similar text found: How does the use of SELFIE can benefit a school? 0.38
INFO:chatterbot.chatterbot:Similar text found: How can selfie help a school? 0.44
INFO:chatterbot.chatterbot:Similar text found: What is a school coordinator 0.72
INFO:chatterbot.chatterbot:Similar text found: What is a school supervisor? 0.79
INFO:chatterbot.chatterbot:Similar text found: What is a school profile 0.98
INFO:chatterbot.chatterbot: Using "What is a school profile" as a close match to "What is a school profile?" with a confidence of 0.98
INFO:chatterbot.chatterbot: Selecting response from optimal responses.
INFO:chatterbot.response_selection: Selecting response from list of options.
INFO:chatterbot.chatterbot:Response selected. Using "A school profile contains the name and contact information of the school, and also how the school is operated and funded. School profile will be defined after logging into Selfie, and should be checked at least once a year."
INFO:chatterbot.chatterbot:MyLogicAdapter selected "A school profile contains the name and contact information of the school, and also how the school is operated and funded. School profile will be defined after logging into Selfie, and should be checked at least once a year." as a response with a confidence of 0.98
INFO:chatterbot.chatterbot:FastTextLogicAdapter selected "A school profile contains the name and contact information of the school, and also how the school is operated and funded. School profile will be defined after logging into Selfie, and should be checked at least once a year." as a response with a confidence of 0.9997645020484924
```

### 2.5.2. Unknown question

In the following section we will also apply a question related to the definition of the school profile.

```
Question (cURL format)


curl -X POST "http://helper.sherpa4selfie.eu:5000/chatterbot/applyQuestion"
-H "accept: application/json"
-H "Content-Type: application/json"
-d "{
      \"text\": \"Which is the definition of the school profile?\",
      \"languageCode\": \"en\"
    }"


Answer


[
  "A school profile contains the name and contact information of the school, and
also how the school is operated and funded. School profile will be defined after
logging into Selfie, and should be checked at least once a year."
]
```

As we can see in the following table the closest match algorithm is not able to find a statement as all similarity scores are pretty low, so it generates an alternative one with a confidence score equal to zero. On the other hand, the fastText algorithm generates the correct response with a high confidence value. In that case we understand the effectiveness of text similarity that an algorithm such as fastText is able to locate.

```
INFO:chatterbot.chatterbot:Processing                                    search
results
                                                                    INFO:chat
terbot.chatterbot:Similar text found: Only the school coordinator needs to login
to Selfie. No other member in the school community needs to login to Selfie.
0.24
INFO:chatterbot.chatterbot:Similar text found: I don't have a school account! Can
I         sign         up         with         my         mail         account?
0.27
  INFO:chatterbot.chatterbot:Similar text found: How does the use of SELFIE can
benefit                                a                                school?
0.47
                        INFO:chatterbot.chatterbot:Similar text found: What is the
```

```
supervisor for a school?
0.54
INFO:chatterbot.chatterbot:Similar text found:
What is the coordinator of a school?
0.66
INFO:chatterbot.chatterbot:Similar text found:
What is the meaning of a school profile?
0.77
INFO:chatterbot.chatterbot:Similar text found: What
is the definition of a school leader
0.78
INFO:chatterbot.chatterbot:Using "Only the school
coordinator needs to login to Selfie. No other member in the school community
needs to login to Selfie." as a close match to "Which is the definition of the
school profile?" with a confidence of
0.24
INFO:chatter
bot.chatterbot:No responses found. Generating alternate response
list.
INFO:chatterbot.chatterbot:No known
response to the input was found. Selecting a random
response.
INFO:chatterbot.chatterbot:MyLogicAdapter selected
"Teachers are defined as persons who are qualified teachers and other personnel
directly involved in teaching students (whole class and/or small groups or
individuals). Please do not include teachers who are primarily in a management
role - they fall into the school leader group." as a response with a confidence
of 0 INFO:chatterbot.chatterbot:FastTextLogicAdapter
selected "A school profile contains the name and contact information of the school,
and also how the school is operated and funded. School profile will be defined
after logging into Selfie, and should be checked at least once a year." as a
response with a confidence of 0.988141655921936
```

## 2.6. Implementation details

### 2.6.1. Hyper-parameter tuning

As it is described in the code description the fastTextTrainer contains a set of parameters which affects its efficiency. Our set up came up with a number of experiments and by following the official documentation. To be more specific, in the fastTextTrainer.py file (located in core/model/fastText path) we have the following code.

```
hyper_params = {"lr": 1.0,
                "epoch": 50,
```

```
                "wordNgrams": 2,
                "minn": 3,
                "maxn": 5,
                "dim": 100}
model = fasttext.train_supervised(input=train_file_name, **hyper_params)
```

We can see that we set the hyper_params dictionary which is passed as a parameter in the new model. Anyone can change these parameters in order to achieve higher accuracy. We will describe in the following lines in a nutshell the use of each parameter.

- lr: a way to change the learning speed of our model is to increase (or decrease) the learning rate of the algorithm. This corresponds to how much the model changes after processing each example. A learning rate of 0 would mean that the model does not change at all, and thus, does not learn anything.
- epoch: the number of times each example is seen.
- wordNgrams: the way to use bigrams or more in our model. It is important in our case as the word order is critical.
- minn: min length of char ngram.
- maxn: max length of char ngram.
- dim: size of word vectors.

### 2.6.2. Model training

In the following tables we will describe the training code that both models are following which is located in fastTextTrainer.py and in wordSimilarityTrainer.py for fastText and word similarity algorithms respectively.

```
class ChatterBotWordSimilarityTrainer(Trainer):
    def __init__(self, chatbot, **kwargs):
        super().__init__(chatbot, **kwargs)
        self.httpClient = None


    def setHttpClient(self, httpClient):
        self.httpClient = httpClient
```

```
    def train(self, *corpus_paths):
        languages = self.httpClient.getAllLanguages()


        for language in languages:
            statements_to_create = []
            languageCode = language['code']
            questions = self.httpClient.getQuestionsPerLanguage(languageCode)
            for question in questions:
                answer = question['answer']
                response = answer['description']
                question = question['description']
                statement_search_text                                          =
self.chatbot.storage.tagger.get_bigram_pair_string(response)
                search_in_response_to                                          =
self.chatbot.storage.tagger.get_bigram_pair_string(question)
                statement = Statement(
                    text=response,
                    search_text=statement_search_text,
                    in_response_to=question,
                    search_in_response_to=search_in_response_to,
                    conversation='training'
                )
                statement = self.get_preprocessed_statement(statement)
                statements_to_create.append(statement)


                pattern_pair                                                   =
self.chatbot.storage.tagger.get_bigram_pair_string(question)
                statement = Statement(
                    text=question,
                    search_text=pattern_pair,
                    conversation='training'
                )
                statement = self.get_preprocessed_statement(statement)
                statements_to_create.append(statement)
            self.chatbot.storage.create_many(statements_to_create)
```

As we can see, using an http client we receive all question/answer pairs for every language from the external database. For each question/answer pair we create a Statement, which is an internal representation of the chatterbot engine, and we store it in our mysql database.

```
class ChatterBotFastTextTrainer(Trainer):
```

```
    def __init__(self, chatbot, **kwargs):
        super().__init__(chatbot, **kwargs)
        self.httpClient = None


    def setHttpClient(self, httpClient):
        self.httpClient = httpClient


    def preprocess(self, line):
        filtered_line = ''
        line_token = word_tokenize(line)
        remove_sw = [word for word in line_token if not word in STOP_WORDS]
        filtered_line += ' '.join(remove_sw)
        filtered_line += '\n'
        return filtered_line.replace(' ', '\t', 1)


    def train(self):
        languages = self.httpClient.getAllLanguages()
        for language in languages:
            languageCode = language['code']
            train_data = ''
            response_data = {}
            train_file_name = os.path.join(os.getcwd(),
                     'files/train_data/' + languageCode + '.txt')
            response_file_name = os.path.join(os.getcwd(),
                    'files/response_data/' + languageCode + '.json')


            createFile(train_file_name)
            createFile(response_file_name)


            train_file = open(train_file_name, 'w+')
            response_file = open(response_file_name, 'w+')
            questions = self.httpClient.getQuestionsPerLanguage(languageCode)
            for question in questions:
                answer = question['answer']
                train_data += '__label__' + str(answer['id']) + ' '
                    + question['description'] + '\n'
                response_data['__label__'
                    + str(answer['id'])] = answer['description']


            train_file.write(train_data)
            response_file.write(json.dumps(response_data))
            train_file.close()
            response_file.close()
            hyper_params = {"lr": 1.0,
                            "epoch": 50,
```

```
                                "wordNgrams": 2,
                                "minn": 3,
                                "maxn": 5,
                                "dim": 100}
            model = fasttext.
                    train_supervised(input=train_file_name, **hyper_params)
            model_file_name = os.path.join(os.getcwd(),
                    'files/models/' + languageCode + '.bin')
            createFile(model_file_name)
            model.save_model(model_file_name)
```

The above code describes the way the fastText trainer is initialized. The system contains as many models as the system languages and each model is saved in the proper .bin file. Thus, anytime a question response is needed the proper model is loaded. As word similarity trainer all question/answer pairs are received from the external database using an http client. In the fastText trainer case, a transformation is necessary for each pair in order to format them in the proper representation. Thus, for every language a train file is created which contains a label/question pair and a response file containing a label/answer pair. Finally, the training file is used as model trainer input for the training process.

### 2.6.3. Model response

In the following tables we will describe the code that both models are following in order to respond to an input question. The code is located in fastTextClassifier.py and in logicAdapter.py for fastText and word similarity algorithms respectively.

```
class MyLogicAdapter(LogicAdapter):


    def __init__(self, chatbot, **kwargs):
        super().__init__(chatbot, **kwargs)
        self.excluded_words = []


    def can_process(self, statement):
        return True


    def process(self, input_statement,additional_response_selection_parameters):
        search_results = self.search_algorithm.search(input_statement)
        # Use the input statement as the closest match
        # if no other results are found
        closest_match = next(search_results, input_statement)
```

```
        # Search for the closest match to the input statement
        for result in search_results:
            # Stop searching if a match that is close enough is found
            if result.confidence >= self.maximum_similarity_threshold:
                closest_match = result
                break


        self.chatbot.logger.info('Using "{}" as a close match to
           "{}" with a confidence of {}'.format(
            closest_match.text, input_statement.text, closest_match.confidence
        ))


        response_selection_parameters = {
            'search_in_response_to': closest_match.search_text,
            'exclude_text_words': self.excluded_words
        }


        alternate_response_selection_parameters = {
            'search_in_response_to': self.chatbot.storage.tagger.get_bigram_pair_string(
                input_statement.text
            ),
            'exclude_text_words': self.excluded_words
        }


        if additional_response_selection_parameters:
            response_selection_parameters
                 .update(additional_response_selection_parameters)
            alternate_response_selection_parameters
                 .update(additional_response_selection_parameters)


        # Get all statements that are in response to the closest match
        response_list = list(self.chatbot.storage
                      .filter(**response_selection_parameters))


        alternate_response_list = []


        if not response_list:
            self.chatbot.logger.info('No responses found.
                             Generating alternate response list.')
            alternate_response_list = list(self.chatbot.storage
                    .filter(**alternate_response_selection_parameters))


        if response_list:
            self.chatbot.logger.info(
                'Selecting response from {} optimal responses.'.format(
                    len(response_list)
                )
            )
```

```
            response = self.select_response(
                input_statement,
                response_list,
                self.chatbot.storage
            )


            response.confidence = closest_match.confidence
            self.chatbot.logger.info('Response selected.
                        Using "{}"'.format(response.text))
        elif alternate_response_list:
            '''
            The case where there was no responses returned for the selected match
            but a value exists for the statement the match is in response to.
            '''
            self.chatbot.logger.info(
                'Selecting response from {} optimal alternate responses.'.format(
                    len(alternate_response_list)
                )
            )
            response = self.select_response(
                input_statement,
                alternate_response_list,
                self.chatbot.storage
            )


            response.confidence = closest_match.confidence
            self.chatbot.logger.info('Alternate response selected.
                         Using "{}"'.format(response.text))
        else:
            response = self.get_default_response(input_statement)


        return response
```

The above code describes the way the embedded chatterbot's mechanism is used to choose the best response. In a nutshell, using the search algorithm in the first line of the process method, in our case the Levenstein distance, all possible matching cases along with their confidence score are retrieved. The one with the best match score is chosen as the closest one and the response is the answer which is in response to that question. In case all search results have low confidence scores the process method returns a randomly selected answer.

```
class FastTextClassifier:


    def __init__(self, languageCode):
        self.languageCode = languageCode
        if os.path.exists(os.path.join(os.getcwd(),
               'files/models/' + self.languageCode + '.bin')):
```

```
            self.model = fasttext.load_model(os.path.join(os.getcwd(),
                'files/models/' + self.languageCode + '.bin'))


    def clean_up_sentence(self, sentence):
        stemmer = LancasterStemmer()
        # tokenize the pattern
        sentence_words = nltk.word_tokenize(sentence)
        # stem each word
        sentence_words = [stemmer.stem(word.lower()) for word in sentence_words]
        return sentence_words


    # return bag of words array: 0 or 1
    #for each word in the bag that exists in the sentence
    def bow(self, sentence, show_details=False):
        # tokenize the pattern
        sentence_words = self.clean_up_sentence(sentence)
        # bag of words
        bag = [0] * len(self.data['words'])
        for s in sentence_words:
            for i, w in enumerate(self.data['words']):
                if w == s:
                    bag[i] = 1
                    if show_details:
                        print("found in bag: %s" % w)


        return np.array(bag)


    def classify(self, sentence):
        sentence = self.preprocess(sentence)
        prediction = self.predict(sentence)
        utterance = self.utter(prediction)
        return utterance


    def predict(self, query):
        texts = [query]
        prediction = json.dumps(self.model.predict(texts, k=-1), cls=NumpyArrayEncoder)
        return prediction


    def utter(self, prediction):
        response_file = open(os.path.join(os.getcwd(),
           'files/response_data/' + self.languageCode + '.json'), 'r')
        responses = json.loads(response_file.read())
        labels = json.loads(prediction)[0][0]
        confidences = json.loads(prediction)[1][0]
        return [(responses[labels[0]], confidences[0])]


    def preprocess(self, sentence):
        filtered_sentence = ''
        line_token = word_tokenize(sentence)
        remove_sw = [word for word in line_token if not word in STOP_WORDS]
```

```
        filtered_sentence += ' '.join(remove_sw)
        return filtered_sentence
```

The above code describes the way the fastText algorithm is used to select the best match response. The intuition behind the answer selection is that via both the preprocess and prediction steps the algorithm tries to labelize, from the predefined closed set of labels, the user's question. The label/questions pair with the highest confidence score is the best matching. Thus, the algorithm's response equals the label/answer pair which is in response to that question.

### 2.6.4. Best response selection

As it is already described the chatterbot engine uses two algorithms for answer selection to achieve better performance and accuracy. The code below shows the last step which includes the selection of the most accurate answer, the one with the highest confidence value, between the best algorithms' responses.

```
def generateResponse(chatbotLevenshtein, chatbotFastText,
                     question,languageCode, threshold):

    s1 = chatbotLevenshtein.get_response(question)
    args = {'additional_response_selection_parameters':
               {'languageCode': languageCode}}

    s2 = chatbotFastText.get_response(question, **args)


    if s1.confidence <= s2.confidence:
        response = ("FastText similarity", s2.text, s2.confidence)
    else:
        response = ("Levenshtein similarity", s1.text, s1.confidence)


    if response[2] <= float(threshold):
        return []
    return [response[1]]
```

It is obvious from the code above that in case all answers are lower than the threshold score, which was experimentally selected, the system returns no answer to the user.

## 3. Project Code repositories

The project code is available through the repositories presented at Table 3. All the code is available in an aggregation point at SHERPA Helper Project (SHERPA Team, 2022e) while there has been mirroring with the git-hub account of Centre for educational Technology.

**Table 3. Project code repositories**

| Project Module | Repository URL |
|---|---|
| SHERPA Helper Project Aggregation point | https://gitlab.com/aetma/sherpa-project |
| SHERPA CBR | https://gitlab.com/aetma/sherpa-project/sherpa_inference_engine |
| SHERPA HELPER KNOWLEDGE BASE REPOSITORY | https://github.com/centre-for-educational-technology/sherpa-kb<br>https://gitlab.com/aetma/sherpa-project/sherpa-knowledge-base |
| SHERPA CHATBOT USER INTERFACE | https://github.com/centre-for-educational-technology/sherpa-helper<br>https://gitlab.com/aetma/sherpa-project/sherpa-chatbot |

# Appendix B: SHERPA Consortium Partners

The SHERPA consortium comprises five (5) partners (P) from 5 different European countries (Greece, Finland, Cyprus, Estonia, and Italy) and 1 affiliated entity depending from P01. Table 3 includes information of the SHERPA project consortium.

**Table 4. SHERPA Consortium Information**

| P# | Full Official Name | Acronym | Country |
|---|---|---|---|
| P01 | Advanced Educational Technologies and Mobile Applications Lab, International Hellenic University | AETMA-IHU | Greece |
| P02 | University of Jyvaskyla | JYU | Finland |
| P03 | Cyprus Pedagogical Institute, of the Ministry of Education and Culture | CPI | Cyprus |
| P04 | Tallinn University | TLU | Estonia |
| P05 | National Research Council of Italy, Institute for Educational Technology | CNR-ITD | Italy |
| AF01 | Open Technologies Alliance (GFOSS) | GFOSS | Greece |