UNIVERSIDAD POLITÉCNICA
DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS
MÁSTER UNIVERSITARIO EN INGENIERÍA DEL SOFTWARE –
EUROPEAN MASTER IN SOFTWARE ENGINEERING

Crowdsourced translation service for Multiplatform Application Localization

Master Thesis

Borys Makogonyuk Vasylev

Madrid, July 2017

This thesis is submitted to the ETSI Informáticos at Universidad Politécnica de Madrid in partial fulfilment of the requirements for the degree of Master of Science in Software Engineering.

Master Thesis
Master Universitario en Ingeniería del Software – European Master in Software Engineering
Thesis Title: Crowdsourced translation service for Multiplatform Application Localization

Thesis no: EMSE-2017-08
July 2017

Author: Borys Makogonyuk Vasylev
Student
Universidad Politécnica de Madrid

Supervisor:

Susana Muñoz Hernandez
Associate Professor
Universidad Politécnica de Madrid

Departmento de Lenguajes y Sistemas
Informáticos e Ingeniería de Software

Facultad de Informática
Universidad Politécnica de Madrid

ETSI Informáticos
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
28660 Boadilla del Monte (Madrid)
Spain

Abstract

Nowadays, applications are developed for a variety of platforms, and even though developers strive to cover as much of the potential market as possible, there are still a lot of problems when it comes to the localization of the applications. Usually, a developer will support only the majoritarian language in a country and will ignore other languages due to time and budget constraints. It does not matter if it is a website or a mobile application, the amount of effort that goes into the translation of them is equal. User requests on adding a particular language are usually given low priority due to the small size of the user base that would use that language as well as problems with the actual translation.

At the same moment we live in a time were crowdsourcing niche projects has become the norm and user contributions are highly valued for these low-demand features. Crowdsourcing is usually done for free, on a voluntary basis. Be it astronomy, ornithology or linguistics, crowd's contributions can be vital in different fields.

When it comes to software translation, although there are many services on the field that provide the developers a platform where users can contribute to the translation of the application, most of them are closed sourced, paid or have strict limitations.

This goal of this project is to kickstart and maintain an open source crowd-sourced translation platform. Going open source will not only allow any developer to use the service either as a hosted solution or self-host it themselves, but it will also provide a natural means of attracting new developers that might be interested in further developing the platform. The developers will share their resource files and with the help of their users and other enthusiasts the application will be translated to minority languages. The project's modular architecture allows extending the application without having to change any internals.

Table of Contents

Table of Figures

Table of Tables

Table of Code Snippets

1. Introduction

Today we live in a multicultural and globalized world where an application developed by somebody in one end of the world can be used by anyone on the other end. As it is with most media content, software also needs to be translated to be able to be utilized by the population of those markets. Most of the applications are unusable unless they are translated due to the sheer fact that people outside of the country where that language is familiar will most likely not be able to speak it.

Translation in itself is a laborious and time-consuming process. Although machine learning and automated translation system are getting better each year, it's not possible to rely on it entirely. If the quality of the translation is important, raw machine translations are never used. For the best results, it is better to have a translator that is a native speaker of the language that is being translated to and has high proficiency in the language he is translating from. Companies usually outsource the localization process in the respective countries or, the ones that can afford it, open offices whose sole job is the localization of the companies projects in that country.

Smaller developers do not have that opportunity. Upon releasing a new application, they may invest some amounts of money into translating it into majoritarian languages like English, Spanish, German, French, Russian. But at the same time other minoritarian and regional languages have to be left out due to it being not economically viable to translate the applications to a language that is not going to be used by 95% of the application's user base.

Therefore, a viable solution for smaller developers that want to support these languages is to crowdsource the localization process to their users. Most of the platforms that allow such translations are either paid, have strict limitations, or are free only for open source software. While the last condition may be viable for some developers, it’s not a solution for the majority.

Therefore, the idea that is right on the top of the stack is to develop an open-sourced solution that can be hosted, used and extended by anybody interested in it. The main idea of the platform is not different to other similar platforms. The developers are able to upload their localization files in one language and the platform then provides a means for the users of that application to contribute to the translation to a language they are interested in.

This document will further cover the following topics in the following chapters. Chapter 2 will cover the analysis of the problem of localization as well as discuss existing similar services and their shortcomings. In chapter 3 we will discuss the functional and non-functional requirements, data flows and user interactions, technical decisions as well as the coverage of the development within this thesis. Chapter 4 will focus on the development itself, on the technologies used, the architecture and the individual components of the platform. In chapter 5 we will review the developed platform, talk about its shortcomings and advantages. Chapter 6 will be devoted to conclusions.

2. Field background and reasoning

2.1. Translation industry overview

The translation industry has never been as big as it is now. In 2016 it amounted to a total of 40 billion US dollars according to research performed by Common Sense Advisory. This is almost 50% more than the market share in 2009 with 23.5 billion US dollars [1]. As we can see from Figure 1, the growth of the market has been steady for the period provided, and the forecasted rise to 45 billion US dollars does not seem that impossible.

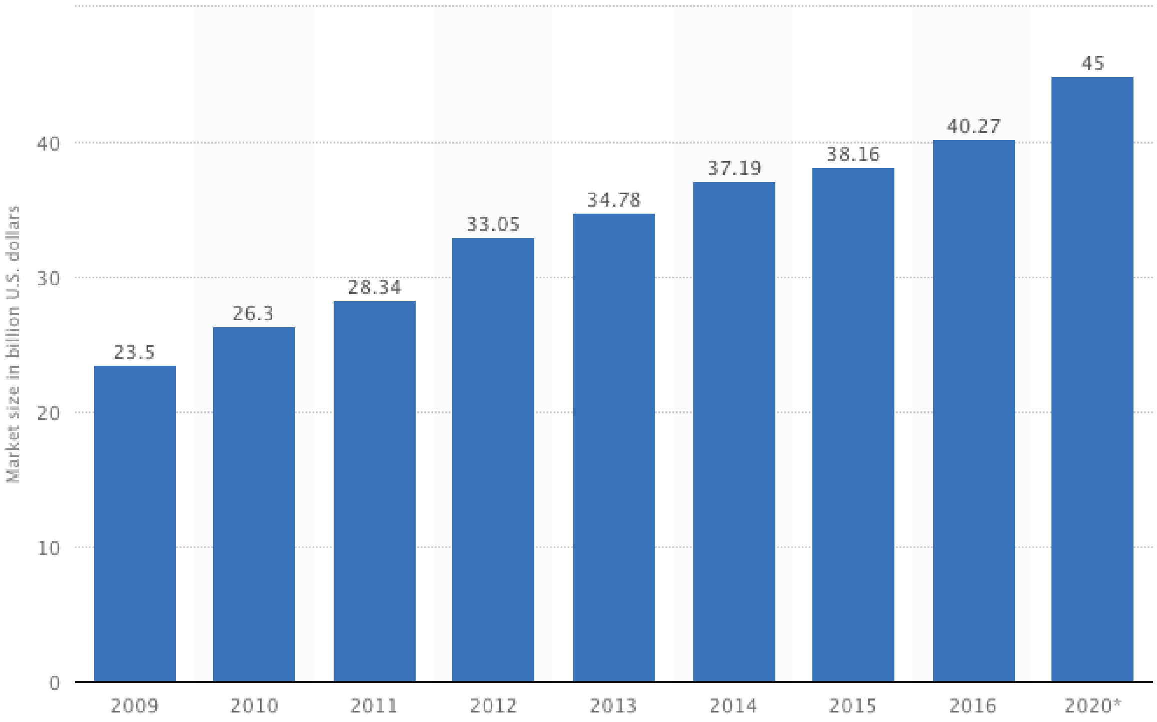


Figure 1. Market size of the global languages services industry from 2009 to 2020

This data helps us understand that the translation services industry is still experiencing growth, therefore, the demand is at least steady if not rising. And it is not surprising considering the speed at which globalization is currently going at and the urge for companies to reach to as many people as possible.

2.2. Minority languages

Minority languages are languages that are spoken by a minority of the population of a territory. There is a total of up to 7100 languages spoken worldwide, and 90% of them are considered minority languages due to the small amount of population that uses them in day to day life as can be seen on SIL International’s report [2]. Up to only 19% of languages have more than 100,000 people speaking them. The other 78%, although are considered living languages, have a meager amount of population speaking them. 3% of the languages are considered extinct.

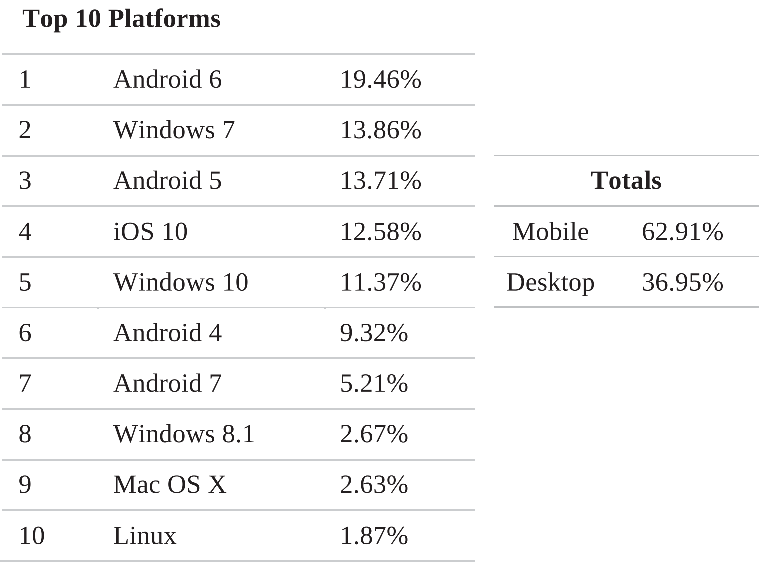
An important fact to take into account is that the number of speakers of a language while does have a strong correlation with how alive it is, it is not a direct one. Up to 65% of languages are considered sustainable by the EGIDS scale: the languages are used for face-to-face communication by all generations.

With the speed at which the world is globalizing, many languages are under severe risk of disappearance. People prioritize learning internationally popular languages to be able to have more opportunities. Languages that have strong cultural and traditional ties are at a lower risk due to the extra support that is provided by such interactions.

As such, being able to use applications in their native languages is a big selling point to a lot of people and if the users can moderate themselves for translations of said applications, the developers will be only happy to include those changes. Therefore what is required to further expand this process is a platform that facilitates the interaction between users and developers.

2.3. Platform share overview

In an article by Smart Insights [3], the data provided on the number of global internet users by platform extracted from a Morgan Stanley Research report indicates that in 2014 mobile internet usage surpassed desktop usage. As of 2016, 90% of consumers' mobile time is spent in applications and not in the browser. Although the number of users that mainly use applications on their smartphones has almost reached its absolute, the browsers are still the main way access the internet in general and not a specific service. Therefore we can use browser and platform market share data to determine the exact share.



Top 10 Platforms

1	Android 6	19.46%
2	Windows 7	13.86%
3	Android 5	13.71%
4	iOS 10	12.58%
5	Windows 10	11.37%
6	Android 4	9.32%
7	Android 7	5.21%
8	Windows 8.1	2.67%
9	Mac OS X	2.63%
10	Linux	1.87%

Totals	
Mobile	62.91%
Desktop	36.95%

Table 1. Browser platform market share as of May 2017 [4]

As we can see from Table 1, the mobile platform share is at almost 63% while desktop only occupies around 37%. Still, while the platform share is tilted towards the mobile market, and Android is the clear winner in that sphere, applications are still developed for all devices. One thing that does unite them all is the way that translation is handled. Therefore, it is a viable goal to be able to cover all platforms with one translation service.

2.4. Translation strategies

iOS, Android, and websites all use a similar approach to translations. They have hard coded keys of resource strings that then are loaded from the corresponding language resource files when the user interacts with the program. That resource file is a list of key-value pairs that includes a descriptive key for the needed field and has a corresponding text string.

Although different platforms do use different formats to achieve this, the core principle is not any different on any of them. It is a file with key-value pairs. This is an important fact as this allows us to map various types of resource files to a general structure in which they can be represented during translation and them converted back into their corresponding format.

Usually, these files are reformatted and transferred to translators with context images so that the translators can do a better job of conveying the meaning of the original native text so that there is nothing lost in translation.

2.5. Proposed and developed solution

Due to the different ways that applications load translations but the similar way they store them, it was decided to develop a service that would be able to be used with the majority of platforms and could be easily extended to support any new coming ones.

Trowd (Translation Crowd) is an online service that allows developers to crowdsource the translation of their applications. The service can keep projects private if such is needed, but the main idea is that any user of the service can find projects that are interesting for him and help translate them. Once the translation to a language is finished, the developer can download the translated resource file and include it into his application or, if the platform allows it, download them from the web at runtime. In the latter, this allows updating the application localization without having the need to resubmit it to the corresponding application store or in the case of websites it removes the need for redeployment.

Trowd comes with a list of locales, regional parameters that define the language, region, and preferences for user interfaces. This allows people that speak a regional subset of a language to use what is more native to them. For example, Spanish has 23 different locales. Ecuadorian Spanish is different from Spain’s Spanish, as it is different from Colombian Spanish. Specifying languages in this way gives more freedom to the translators overall and allows to transfer a majoritarian language into it subset easily. If the application already has a Spanish translation, then a person from Uruguay can easily localize the parts that are different.

Going into technical details, developers will be able to upload their localization resource files in various formats: being it a JSON file which is usually used by websites, XML which is employed by Android and Windows applications, or Apple’s “strings” file. That file is then parsed and converted into database entries that represent the keys and values in those resource files. The developer can then provide a link to the users of his application by any means he seems viable. The users will then use the service to translate the values for the keys in the needed languages. Translations can be voted, discussed within the users. The developer or an assigned moderator can then confirm the translation.

At any given time the developer can download the localization resources for a language. If some strings are not translated, they will be pulled from the source language to be able to fill the gap.

Although this project aims at collaboration and openness, developers have the ability to make a project private and only allow invited users to collaborate. Although this is counter-productive to the idea of crowdsourcing the application, we do not want to limit the ways in which people may want to use the service.

The service will be open sourced and free to use. Trowd will be provided as an already hosted solution, but any person has the freedom to download and host it themselves. The fact that the software is open source will allow anybody to contribute to it and add functionality that he or she seems viable.

2.6. Existing services overview

What follows is a small summary of the existing platforms that have a similar approach in order to be able to better understand what solutions already exists and how they differ.

Transifex[1]

Transifex is a commercial state of the art translation platform. It empowers anybody that is related to the translation process to translate better and faster. Their service is mainly aimed at companies and not at open crowdsourcing per say. Their pricing plans depend primarily on the number of collaborators and the words in the resource files. While the number of words is not that big of a problem for crowdsourced translations, the number of collaborators is very strict starting at 170 US dollars for ten users. They also provide paid translators to aid in the translations of projects, a robust API, and next day support.

Transifex used to be an open sourced solution, but in 2012 they went commercial and stopped open sourced development in favor of a closed sourced company centered solution. They still offer free access to non-profit non-funded open source projects.

POEditor[2]

POEditor is one of the better alternatives for a small developer to use. While they do have a limit of only 1000 strings, there is no limit on the number of contributors or languages that can be utilized. They provide integrations with a variety of services to ease localization.

[1] https://www.transifex.com/
[2] https://poeditor.com/

Others

Smartling[3], Gengo[4], Get Localization[5], FoxTranslate[6], Acclaro[7] – all these services are crowdsourced, but not in the usual meaning of crowdsourcing. What these companies promise is a network of professional translators that will work on a project. Of course, due to the nature of these services they are paid-only and are not available for free. The prices vary from service to service, but their core idea is far from what we are looking for.

[3] http://www.smartling.com/
[4] http://gengo.com/
[5] http://www.getlocalization.com/
[6] http://www.foxtranslate.com/
[7] http://www.acclaro.com/

3. Technical analysis and specification

For the development of this project three possible ways of specifying requirements were considered:

IEEE 830-1998 [5]. It contains recommended practices on software requirements specification. These recommendations are very flexible and allow only to specify the needed categories. The standard can be well adapted and executed in projects that can have their functionality fully defined from the ground up, but in the real world, it hardly survives due to the dynamic nature of the field.

IEEE 29148:2011 [6]. This standard describes the life cycle processes of a software product and consists of the following:

- Stakeholder Requirements Specification (StRS)
- System Requirements Specification (SyRS)
- Software Requirements Specification (SRS)
- System Operational Concept (OpsCon)
- Concept of Operations (ConOps)

The problem with this standard is that it is very comprehensive and time-consuming to product and its level of detail is mainly applicable to large software product solutions as is CMMI.

Agile. The idea of iterative and incremental development works well in a real-world environment with a real customer. Constant communication with the client is a fundamental component to the success of this methodology.

Eventually, the decision fell toward IEEE-830 due to the sheer fact that the development would be performed by a team of one with a good knowledge of the initial product that is planned. A template written by Karl E. Wiegers will be used [7].

3.1. Overall description

This section is an overall description of the project in order to summarize all previous statements in regards to the system.

3.1.1. Product perspective

As can be seen in Figure 2, the service consists of two parts: the backend and the frontend client. The backend will provide an API that the client will consume. Such an approach will hide all the complexity from the user interface side and provide a guaranteed separation of concerns and modularity.

The backend interacts with the database and external services (if any). It does all the complex procedures and computations. Due to this project being data centered all operations must pass through the backend server to be validated.

The frontend part is independent of the backend apart from the reliance on the API. Its only job is to query the backend for information and render it.

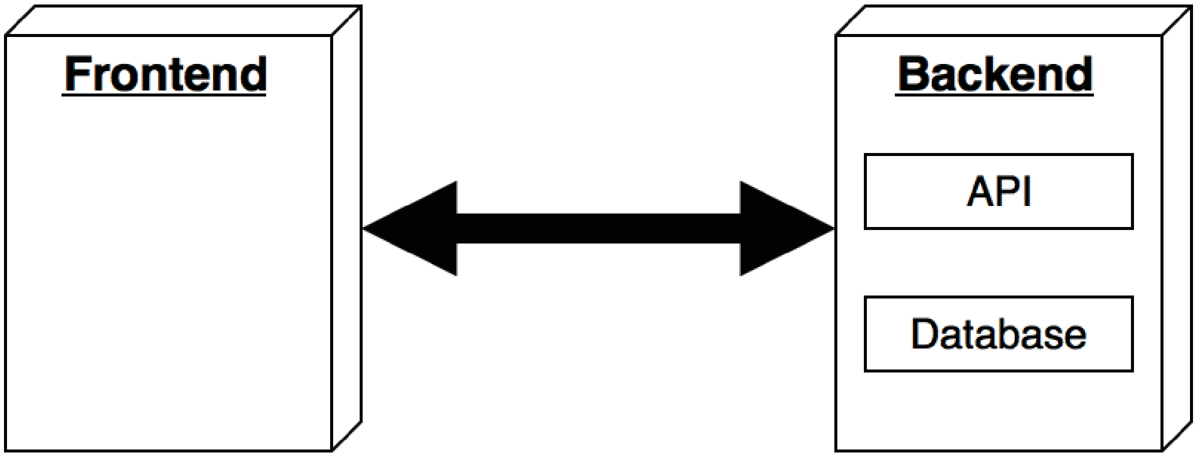


Figure 2. Product perspective. Context diagram

3.1.2. Service functions

The service allows developers create projects and upload resource files. The service then parses those resource files and provides a means to translate the contents of said resource files. In the process of translation, specific variants of a translation can be voted or commented on. The developer or a moderator assigned by the developer has the right to approve translations.

At any moment, the developer has the ability to generate and download the translated resource files. The downloaded translated resource will be based on the initially uploaded resource file by the user.

3.1.3. User classes and characteristics

The system contemplates four types of users that will interact with the system: administrator, developer, moderator, and translator. All of these users have different permissions in regards to the application and therefore will have access to varying parts of the application.

The translator role is applied to any registered user. These users can explore available projects, contribute to them and take part in discussions.

The developer rights are given to an account once a project is created and apply only to that project. The developer has full access to the management of the project. He can modify the project's details, upload new resource files and approve translations. He also can set the project to public or private, invite translators and assign moderators.

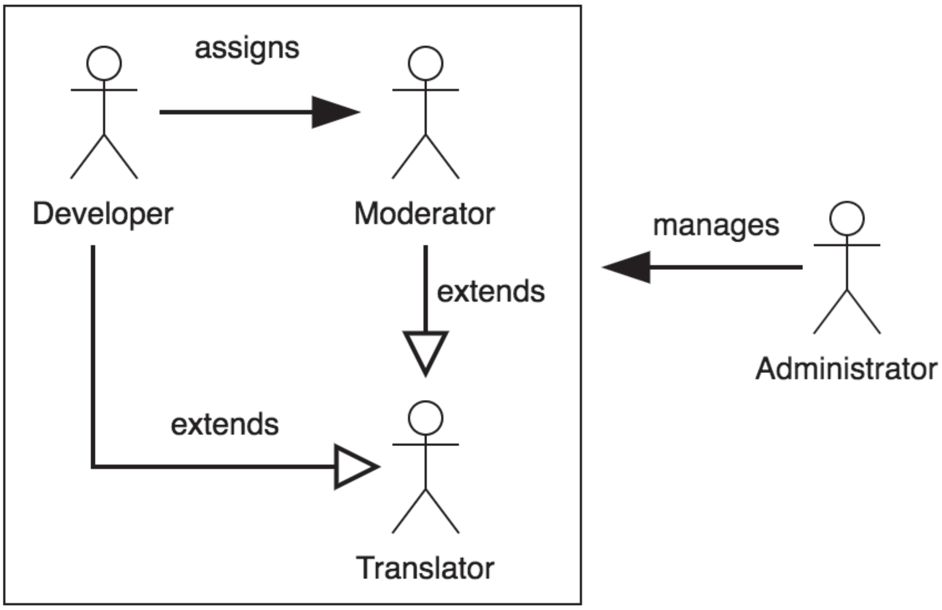


Figure 3. User characteristics. Actor relationship diagram

A moderator is an extension of the translator role that has the permissions to manage the translations within a project. To be able to administer these translations a moderator has to be invited by the creator of the project.

The administrator has access to the overall management of the translation system. This role can manage projects, users, languages, roll on updates, access logs and perform other administrative tasks. The main idea of the administrator role is to provide support in the big picture in the application.

3.1.4. Operating environment

The operating environment will not be of great importance as the service will be written in Python and therefore could be run on either Windows or Unix systems. An active internet connection will be needed to be able to access the service. It is also recommended to host the service behind a web server like Apache or Nginx. Nginx is the recommended solution due to the fact that it has inbuilt WSGI support.

The environment for the database is also not a concern at this point due to the usage of SQLite which can is handled completely by the adapter library of the language. If the need for a more robust database arises, the environment for such database should be discussed separately.

3.1.5. Constraints

This is an online only service. The service may be locally hosted for offline access, but the main idea of this collaboration service as such goes against such actions.

The frontend will cache user inputs in the case that the user loses the connection to the server in the process of translation. If the user resets his browser history before such data is synchronized with the server the data is considered lost forever.

3.1.6. Assumptions and considerations

The initially developed version of this project will not make a big effort on being secure. The service will implement basic authentication measures and do permission checks. CSRF attack counter measures will also be taken into account as well as user input preprocessing. Other possible security problems are out of the scope of the initial implementation of this service.

3.1.7. User documentation

This project includes a manual for the developer, the translator, and documents needed for maintenance and further development of this projects. All of which is to be made available in the wiki of the GitHub repository of this project.

3.2. External interface requirements

This section contains the description of the external inputs and outputs of the system.

3.2.1. User interfaces

Upon reaching the service, the user will be presented with a landing page containing a brief description of what the service is and what functionality it provides. Two calls to action will be shown to the user to either register, login or explore existing projects as illustrated in figure 4. Depending on the choice he makes he will be redirected to the corresponding page.

The login and registration page will be represented by a simple form that allows the user to either log in with an existing account or to register with his email address and a password. The login page will display an error message in the case that the credentials provided by the user are not valid. If the entered credentials are valid, the user will be redirected to the dashboard.

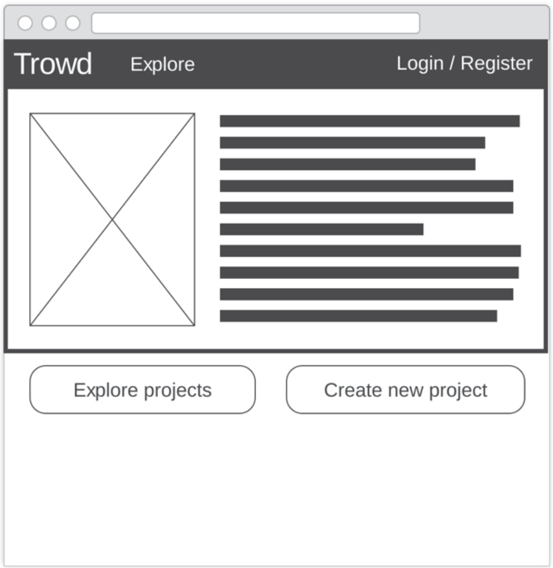


Figure 4. Landing page

The “explore” page will contain a paginated list of projects created by developers that are marked as public. As can be seen on Figure 5, each project will be represented as a card containing the name of the project, a short description, the number of likes and contributors as well as the percentage of text translated based on the languages that the developer has marked as destination languages. Upon clicking onto such card the user will be taken to the “project details” page of the corresponding project.

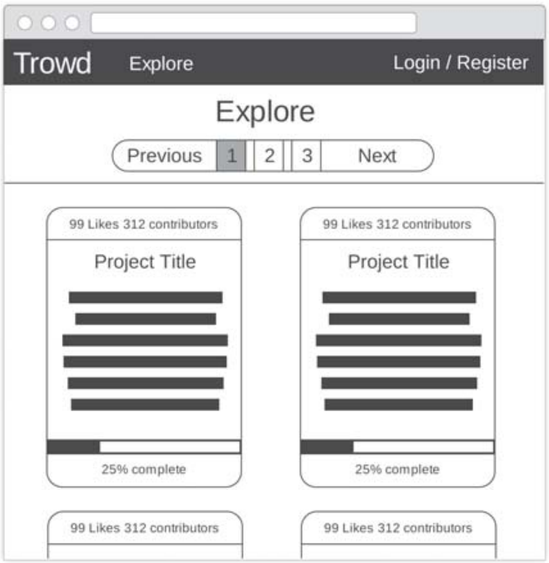

Trowd
Explore
Login / Register
Explore
Previous
1
2
3
Next
99 Likes 312 contributors
Project Title
25% complete
99 Likes 312 contributors
Project Title
25% complete
99 Likes 312 contributors
99 Likes 312 contributors

Figure 5. Explore projects page

The dashboard is the home page for translators and developers. The dashboard shows a summary of owned projects of the user as well as the status of the projects which he has marked as liked. Clicking on a project on this page will open the project’s detail page.

As seen in Figure 6, The detail page shows details about the project such as its title, publicity, number of contributors, number of likes, a full rich text description that is generated from Markdown, and a URL to where the project is distributed through. In the case that this page is accessed by the owner of the project, he can also edit these details. From this page the users can also access the translation page.

The translation page contains a list of resource files that were uploaded by the developer of the project. The developer can add new resource files from this page. The process of adding a new resource file requires the user to enter the source language of the resource file that will be uploaded, the type, and the languages that the developers want to mark as “destination” languages to be able to track their translation process. After filling those details, the user will be prompted to upload a file to finalize the creation of the new resource set. Once the uploaded file has been parsed, and the corresponding entries in the database have been created, the developer will be able to access to the resource editor view.

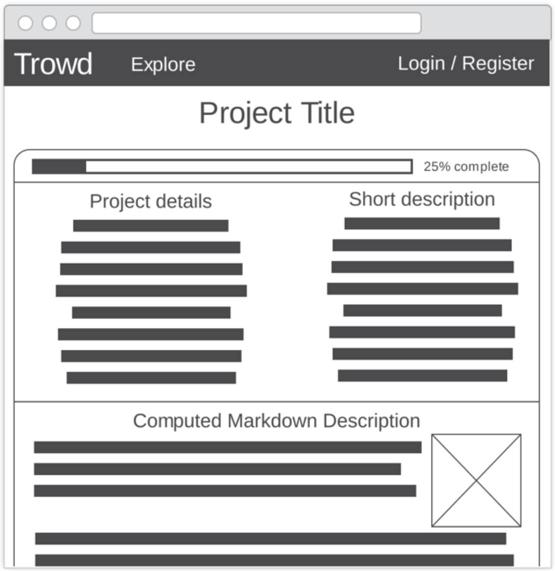


Figure 6. Project details page

The resource editor page allows users to vote, comment, discuss and add translations to existing translation keys. As seen in Figure 7, the user is presented with a table of the translation keys, the string in the source language and the state of a specific string. At the top of the page, the user can change the origin and destination translation languages via two searchable dropdowns.

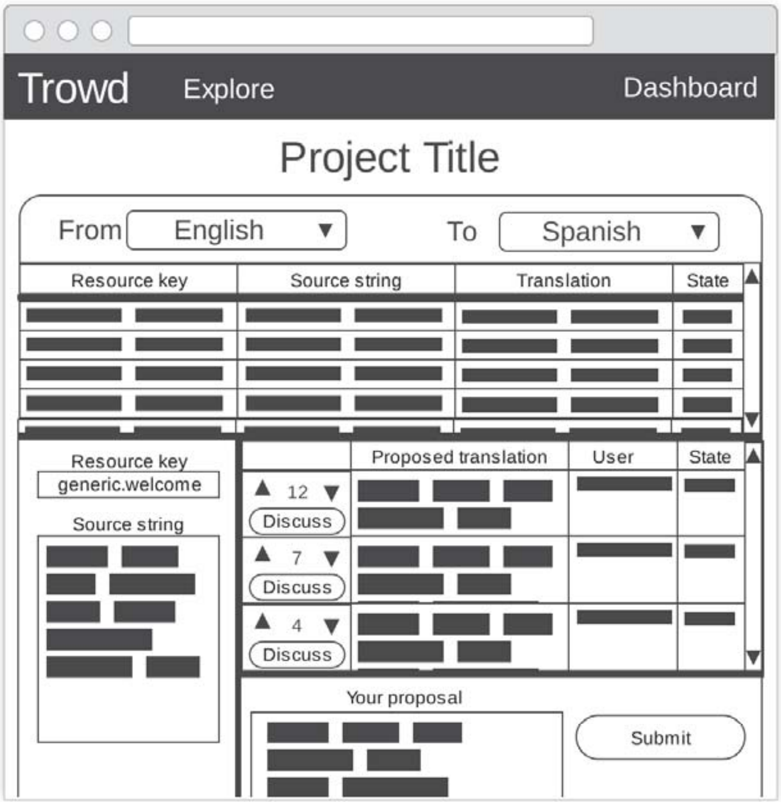


Figure 7. Translation editor page

Once the user selects a specific entry, in the lower part of the window a list of proposed translations is presented, allowing the user to vote and comment on existing translations. The user also has the option to submit his proposal. Only one proposal per user is allowed.

The owner of the project, as well as users whom the developer has assigned as moderators, can also lock any entry so that it is the one that ends up in the final translation file. Voting and discussion on locked strings will still be available.

Upon clicking the discuss button, the user will be presented with a modal window with the current discussions for that specific string. The user can then submit a new comment in that window.

The initial prototype will not be completely responsive and mobile friendly because perfecting the UI and UX is not a goal within this project.

3.2.2. Hardware interfaces

There is no designated hardware neither for the frontend or backend part of the application. The only expected interface is an internet connection between the user and the server.

3.2.3. Software interfaces

The server needs to have full permissions to access and modify the underlying database, therefore a viable adapter has to be selected that supports such operations and does not bring a performance limitation.

One of the bigger mentions in terms of software interfaces in the project is the API hosted on the backend that allows the consumption of the service via any client. This implementation will have a web client developed for the application, but this does not limit in any way the possibilities of developing new clients that interact with the same API.

The API will be implemented as a RESTful service that provides means of performing CRUD operations on the various types of data. The default client implementation will be focused on consuming JSON data, although other types of formats will be available if requested.

3.2.4. Communications interfaces

As for communication, all data between the client and the server will be transferred via HTTP. SSL may be enabled optionally, although recommended.

3.3. System features

3.3.1. General backend requirements

3.3.1.1. Backend API

ID:	FR-BACK-1
Description:	The backend should provide an API that is available for public access that allows interaction with the application in the corresponding flows.
Rationale:	In order to be able to decouple the frontend implementation from the backend implementation.
Dependency:	None

3.3.1.2. Backend API documentation

ID:	FR-BACK-2
Description:	The backend API should have accessible documentation that can be publicly accessed by any user.
Rationale:	In order to be able to provide better integration capabilities.

Dependency:	None

3.3.1.3. Backend API logging

ID:	FR-BACK-3
Description:	The backend API should have the ability to save all incoming requests to a database.
Rationale:	In order to provide interaction traceability.
Dependency:	None

3.3.2. General access

These requirements cover general application access.

3.3.2.1. Default client

ID:	FR-GEN-1
Description:	The service should provide a default UI client available to the users that visit the service.
Rationale:	Due to the main component of the system being the webserver with an exposed API, an initial consumer of said API is needed in order to be able to give access to the users.
Dependency:	None

3.3.2.2. Web browser access

ID:	FR-GEN-2
Description:	The user should be able to access the website that acts as the client for the service from any modern web browser.
Rationale:	In order to be able to interact with the service.
Dependency:	None

3.3.2.3. Webpage interaction

ID:	FR-GEN-3
Description:	The user should be able to interact with the webpage by using a mouse and keyboard, or by using a touch enabled screen.
Rationale:	In order to be able to interact with the service.
Dependency:	FR-GEN-2

3.3.2.4. Bad connectivity accessibility

ID:	FR-GEN-2
Description:	Users with slow internet connections should be able to interact with the website regardless.
Rationale:	In order to be able to interact with the application.
Dependency:	None

3.3.2.5.

ID:	FR-GEN-3
Description:	A user should be notified when he commits an error or there is an error on the backend side of the application.
Rationale:	In order to be able to better navigate the service.
Dependency:	None

3.3.3. Authentication features

These requirements cover user registration, authentication, and security requirements.

3.3.3.1. Registration

ID:	FR-AUTH-1
Description:	A user should be able to create a new account on the service.
Rationale:	In order to be able to login into the service.
Dependency:	None

3.3.3.2. Login

ID:	FR-AUTH-2
Description:	A user should be able to log in with an existing account.
Rationale:	In order to be able to interact with the protected features of the service.
Dependency:	FR-AUTH-1

3.3.3.3. Password reset

ID:	FR-AUTH-3
Description:	A user should be able to reset his account password.

Rationale:	In order to be able to gain access to his account in the case that the aforementioned password is forgotten
Dependency:	FR-AUTH-1

3.3.3.4. Protected sections access

ID:	FR-AUTH-4
Description:	A user should be able to access protected parts of the website once he is authenticated in the service.
Rationale:	In order to be able to interact with parts of the servicethat are available only to specific roles.
Dependency:	FR-AUTH-2

3.3.3.5. Session persistence

ID:	FR-AUTH-5
Description:	A user should be able to return to an opened session once the user closes the web browser or reloads the page.
Rationale:	In order to remove the need to re-authenticate the user each time that he closes the website.
Dependency:	FR-AUTH-2

3.3.3.6. Logout

ID:	FR-AUTH-6
Description:	A user should be able to close his session on the website removing all credentials that gives permission to sections accessible by the user.
Rationale:	In order to be able to use multiple accounts or prevent unauthorized usage on shared devices.
Dependency:	FR-AUTH-2

3.3.4. Exploration features

These requirements cover the exploration functionality of the service.

3.3.4.1. Unauthorized access

ID:	FR-EXPL-1
Description:	An unauthenticated user should be able to access and interact with the exploration functionality of the service.
Rationale:	In order to be able to give unregistered users a preview of what the service is about

Dependency:	None

3.3.4.2. Public projects only

ID:	FR-EXPL-2
Description:	The exploration list presented to users should only include projects that have been marked as public by their owners.
Rationale:	In order to be able to maintain private projects available only to their invitees.
Dependency:	None

3.3.4.3. Project details access

ID:	FR-EXPL-3
Description:	A user should be able to navigate to the detail page of a project selected in the explore page.
Rationale:	In order for the user to be able to familiarize himself with a specific project.
Dependency:	None

3.3.5. Projects requirements

These requirements cover the creation and management of projects

3.3.5.1. Project publicity

ID:	FR-PROJ-1
Description:	A project should be only available for public access if it is marked as such. If the project is private, only invitees can access the project.
Rationale:	In order to be able to maintain projects private.
Dependency:	None

3.3.5.2. Project management access

ID:	FR-PROJ-2
Description:	Any authenticated user should be able to create a new project and manage it.
Rationale:	In order for users to be able to take part in the platform with their projects
Dependency:	None

3.3.5.3. Project creation and modification

ID:	FR-PROJ-3

Description:	A user should be able to create a new project by providing descriptive characteristics of the project.
Rationale:	In order to be able to get visibility and contributions by other users of the platform.
Dependency:	None

3.3.5.4. Markdown description

ID:	FR-PROJ-4
Description:	A user should be able to provide a project description by using the Markdown language. Such descriptions will be accordingly parsed and formatted for other users to see.
Rationale:	In order to be able to use advanced formatting options while keeping the editor purely textual.
Dependency:	None

3.3.5.5. Project voting

ID:	FR-PROJ-5
Description:	A user should be able to vote for a project.
Rationale:	In order to be able to indicate his support for the project and make it more visible on the service’s explore page
Dependency:	None

3.3.5.6. Changing publicity

ID:	FR-PROJ-6
Description:	An owner of the project should be able to change the publicity of the project at any given time and the changes should be instant.
Rationale:	In order to be able to revert user errors in the case that the owner accidentally changes the projects publicity
Dependency:	None

3.3.5.7. Project fields constrains

ID:	FR-PROJ-7

Description:	The backend should validate the values provided by the users into the project’s fields with the following value constraints: - Title. Should not exceed 30 characters. - Short description. Should not exceed 140 characters - Project link. Should be a valid URL. Should not exceed 140 characters - Full description. Should not exceed 2000 characters.
Rationale:	In order to be able to control the size of the content provided by the users.
Dependency:	None

3.3.5.8. Preview of details modification

ID:	FR-PROJ-8
Description:	The user should be able to get preview the modified details of a project before submitting the changes.
Rationale:	In order to be able to edit the project details with more robustness, specifically the Markdown enabled description.
Dependency:	None

3.3.6. Project resource requirements

These requirements cover the project’s resource set requirements.

3.3.6.1. Resource file uploading

ID:	FR-RESS-1
Description:	The owner of a project should be able to upload a new resource file for a specific project.
Rationale:	In order to be able to add that resource file to the list of the project’s resources.
Dependency:	None

3.3.6.2. Resource set parsing

ID:	FR-RESS-2
Description:	An uploaded resource file should be parsed by the server and create the appropriate database entries.
Rationale:	In order to be able to initiate the translation process via the service.
Dependency:	FR-RESS-1

3.3.6.3. Resource file management

ID:	FR-RESS-3
Description:	The owner of a project should be able to manage an existing resource file
Rationale:	In order to be able to change the resource file properties and change prioritized languages.
Dependency:	FR-RESS-1

3.3.6.4. Translated resource file export

ID:	FR-RESS-4
Description:	The owner of a project should be able to download a translated resource file.
Rationale:	In order to be able to export the translated data for further interactions.
Dependency:	FR-RESS-1

3.3.6.5. Resource file deletion protection

ID:	FR-RESS-5
Description:	A user is required to wait an amount of time before data associated to a resource file is completely deleted from the server.
Rationale:	In order to prevent malicious intent of people that gained access to the user's account as well as to prevent human error.
Dependency:	None

3.3.7. Translation editor requirements

These requirements specify the user interactions with the main component of the system – the translation editor.

3.3.7.1. Authorized editor access

ID:	FR-EDTR-1
Description:	Only authenticated users should be able to access the translation editor.
Rationale:	In order to be able to protect the modification of the project by unauthorized users.
Dependency:	None

3.3.7.2. Resource key-value pairs view

ID:	FR-EDTR-2

Description:	The editor page should provide the user with a list of available resource keys, their original language strings and the translation status of said pairs.
Rationale:	In order to give the user an overview of the translation of the resource file to a specific language.
Dependency:	None

3.3.7.3. Language switching

ID:	FR-EDTR-3
Description:	A user should be able to change the source and destination language of translation at any given time.
Rationale:	In order for the user to be able to translation between different languages.
Dependency:	None

3.3.7.4. Fallback resource string values

ID:	FR-EDTR-4
Description:	The resource key-value pairs view should display the original language string with such indication in the case that the language pair selected by the user does not have a translated origin string.
Rationale:	In order to be able to continue with the translation without the need of switching between different languages.
Dependency:	None

3.3.7.5. Resource key selection

ID:	FR-EDTR-5
Description:	A user should be able to see the details on the translation of a specific key by selecting it.
Rationale:	In order to be able to interact with a specific resource key.
Dependency:	FR-EDTR-2

3.3.7.6. Resource key translation details

ID:	FR-EDTR-6
Description:	The service should display the proposed translations, their status, votes, and discussions in regards to a specific resource key.
Rationale:	In order to be able to view the details of a resource key.
Dependency:	FR-EDTR-5

3.3.7.7. Resource key translation voting

ID:	FR-EDTR-7
Description:	A user should be able to vote on a proposed translation for a specific translation key. A user can only vote for one translation per resource key, voting on a different translation removes the previous vote.
Rationale:	In order to be able to endorse valid translations.
Dependency:	FR-EDTR-6

3.3.7.8. Resource translation discussion

ID:	FR-EDTR-8
Description:	A user should be able to discuss a specific translation with other participating users.
Rationale:	In order to be able to discuss the correctness of the translation.
Dependency:	FR-EDTR-6

3.3.7.9. New proposal submission

ID:	FR-EDTR-9
Description:	A user should be able to submit a new translation proposal for a resource key in a specific language.
Rationale:	In order to be able to add new proposals to the process of translation.
Dependency:	FR-EDTR-6

3.3.7.10. Resource key locking

ID:	FR-EDTR-10
Description:	An owner should be able to lock a resource key in such a way that it cannot be translated in any language
Rationale:	In order to mark terms like the application name untranslable
Dependency:	FR-EDTR-2

3.3.7.11. Resource translation locking

ID:	FR-EDTR-11
Description:	An owner or project moderator should be able to lock a specific translation therefore choosing the final version of it for that language
Rationale:	In order to be able to confirm a translation and make it unchangeable.
Dependency:	FR-EDTR-6

3.3.8. General administration requirements

This section covers the requirements for the administration part of the website that is only available to staff members.

3.3.8.1. Administrator access only

ID:	FR-ADMN-1
Description:	Only users that have administrator permissions should be able to access the administration part of the website.
Rationale:	In order to limit access to people not related to the service
Dependency:	None

3.3.8.2. Event logs

ID:	FR-ADMN-2
Description:	An administrator should be able to review the API calls logs.
Rationale:	In order to be able to resolve problematic situations
Dependency:	FR-BACK-2

3.3.8.3. Access everywhere

ID:	FR-ADMN-4
Description:	An administrator should be able to modify access and modify any project, resource or other entity of the service as if it belonged to him.
Rationale:	In order to be able to moderate the service.
Dependency:	FR-BACK-2

3.3.9. Language administration

These requirements are related to the language administration by the administrator role.

3.3.9.1. Language management

ID:	FR-LANG-1
Description:	An administrator should be able to add, modify, enable and disable specific languages.
Rationale:	In order to be able to control the languages available to the users.
Dependency:	FR-ADMN-1

3.3.9.2. Language disabling

ID:	FR-LANG-2
Description:	An administrator should be able to disable a language site-wide without deleting it.
Rationale:	In order to better adapt the service's functionality to specific needs.
Dependency:	FR-LANG-1

3.3.9.3. Language addition

ID:	FR-LANG-3
Description:	An administrator should be able to add new languages to the service which would allow the users to use that language site-wide.
Rationale:	In order to be able to incorporate more languages than the service provides initially.
Dependency:	FR-LANG-1

3.3.9.4. Initial language set

ID:	FR-LANG-4
Description:	The service should provide an initial language set for new deployments.
Rationale:	In order to provide an initial set of languages for the users of the service to use.
Dependency:	None

3.3.9.5. Language data structure

ID:	FR-LANG-5
Description:	A language entry should be structured in such a way that they represent a specific locale rather than just a language itself. Therefore, a language representation should at least include the following fields: - Language name in English. - Language country. - Language locale code. Therefore an entry for a language would have the following format, i.e.: Spanish – Spain – es_ES Spanish – Mexico – es_MX
Rationale:	In order to be able to cover an amount of languages as broad as possible, the format used for language storage should be robust and flexible.
Dependency:	None

3.4. Nonfunctional requirements

3.4.1. Security requirements

Due to the application being developed in a API connected backend-frontend fashion, the security of the information transferred from the client to the server and vice-versa is always critical. Therefore, a solution that allows to guarantee the authenticity of the request as well as protect the information transferred to both sides from tampering is needed.

3.4.2. Software quality attributes

The main focus within the development of this project is to be able to provide a high maintainability, flexibility and portability to the project. Due to the specter of possible usages of the system is so broad, we require the system to be able to run under almost any environment and have the smallest amount of platform dependencies as possible.

4. Development of the solution

4.1. Project architecture

The project consists of two parts: the backend, running on the server, and the frontend that is initially served from the server but then is executed only on the client’s side. The communication between the backend and the frontend is done via a RESTful API therefore further decoupling the implementations.

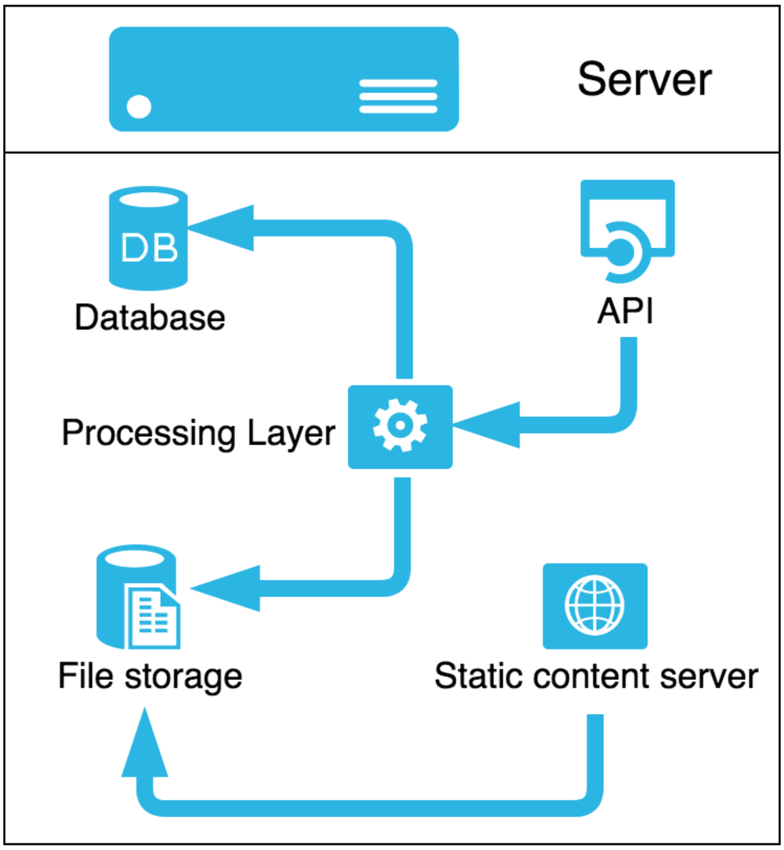


Figure 8. Project architecture

As seen on Figure 8, the database, file store and API are all running on one server. Although this is the initial setup for this specific implementation, every component could be potentially isolated on a separate server and served from there. The API calls to processing layer, which in its case accesses the file storage or the database as needed.

At the same time there is a static content server whose sole work is to serve files from the file storage. Those files will include static files representing the compiled VueJS frontend on the application.

4.2. Technological stack

Taking into account the constraints dictated by the requirements and the architectural decisions, the technology stack was chosen to be as platform independent as possible and as maintainable as possible.

Main language – Python 3.6.1[8]. Python is one of the most known languages in the world due to the large amount of institutions that initiate the teaching process with it. The big number of people that are familiar with Python, it's simplicity, syntax, number of available libraries and cross-platform capabilities made it one of the best candidates possible for the job.

Other languages that were evaluated for this role include, but not limited to:

- Ruby[9]. The Ruby on Rails[10] framework to be exact. Ruby on Rails is a great contender and shares many properties with Python in its simplicity, cross-platform and ease of learning. Due to me being more familiar with Python, it did not take the upper hand.
- C#[11]. .NET Core[12] to the specific. C# in itself is a mature language with very robust tools, it's main problem is its dependency on the .NET Framework and therefore the Windows operating system. .NET Core, being the cross-platform alternative to .NET Framework, although gaining traction and becoming more mature every year, is still very young. There are some "youth bugs" that are still being fleshed out, the number of libraries written or ported for it is not as big as the ones available for the main platform and the tooling is subject of constant breaking changes.

Web Framework – Django[13]. Python has a great variety of available frameworks, but when it comes to full-stack framework Django is the one that has the biggest community and therefore the biggest amount of people familiarized with it. Django is considered to be a framework with "batteries included". It has object-relational mapping out of the box, entity administration tools, a number of different database adapters out of the box and other features that are well integrated into the framework itself.

Due to the need to use Django as an API endpoint, it was decided to use the Django Rest Framework (DRF)[14]. DRF is a mature framework that has highly robust and configurable serialization layer that allows to easily expose the needed parts of the underlying system. It's great documentation and support are also a key factor in choosing it.

Another considered web framework was Flask[15]. Flask is a very simple and slick web framework that allows an easy way of developing a RESTful API. Flask has an extension being developed for it called Flask API which basically replicates what DRF does for Django. Sadly, it is in its early stages of development as is not yet ready for production.
Database – SQLite[16]. As the official SQLite website states, "SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine" [8]. One of the main benefits of SQLite for this specific project is the lack of configuration and lack of hosting the database on a server, it can be just embedded in the running process itself. Apart from that, Django comes with native support for SQLite.

[8] https://www.python.org/
[9] https://www.ruby-lang.org/
[10] http://rubyonrails.org/
[11] https://docs.microsoft.com/en-us/dotnet/csharp/csharp
[12] https://www.microsoft.com/net/core
[13] https://www.djangoproject.com/
[14] http://www.django-rest-framework.org/
[15] http://flask.pocoo.org/
[16] https://www.sqlite.org/

While other database solutions like MySQL were considered, their advantages over SQLite are not of great importance within the boundaries of this development cycle. As stated before, due to Django's great integration with popular databases, the migration to other databases would not be an issue.

Frontend – VueJS[17]. Again, taking into account that the foreseen architecture of the project will not depend on server rendered pages, but rather on an API, a framework that would allow rendering pages just based on incoming data was needed. There are many possible solutions on the market such as AngularJS and ReactJS. The main problem with these frameworks is that they have a steep learning curve and while the concepts themselves are not hard to grasp, the time it takes to get comfortable with the specifics of the frameworks is considerable.

VueJS on the other hand aims at being easy to learn, simple to use, but at the same time has all the needed tools. VueJS unlike AngularJS focuses only on the presentation layer and takes inspiration from ReactJS by providing such functionalities as state and route management through optional extensions.

Again, the decoupling of the frontend and the backend allow for the easy replacement of VueJS with another framework or even full-fledged application if such need arrives.

4.3. Considerations

Although the initial architectural diagram represents the processing and the API layers as separate components, by using Django with the DRF framework we are coupling them into one monolithic handler. Still, the disadvantage of bending the architecture in such a way is not something of big concern as the performance loss is not that critical. If the traffic amount that the server has to process becomes critical, a load balancer could be placed before the entry API point.

4.4. Database schema and entity relations

The database schema representing the entities used in this project is as follows:

Language

This entity represents the general concept of a language. As described in FR-LANG-5, the language entity forms a locale based on a number of properties that define it: the locale code, suffix, name and country. The language also has an "enabled" field to comply with FR-LANG-2.

Project

Accommodates the concept of a project as an existing or developing application or website that requires translation. The properties that define a project are: title, short description, full description, project URL, project, publicity, owner, invitees and moderators. The latter three being references to a entity and the latter two being a many to many relationship.

[17] https://vuejs.org/

ProjectVote
Represents a user’s vote on a specific project.

Profile
Extends the user entity with a one to one relationship. Contains general language related details about a user such as his displayed name, country, known languages and biography. All fields except for his displayed name are optional.

ResourceSet
Transfers the concept of a resource file. Has a reference to the location of the originally uploaded file in the file storage, generated resource keys, a display name for the file, it’s source and owner prepicked destination languages.

ResourceKey
A resource key represents an entry in the originally uploaded translation file. It does not contain the original translation per say, but rather represents the keys that were parsed from the file. Has a Boolean flag that makes the key untranslatable.

ResourceValue
Represents the value of a resource key in a specific language. Therefore contains a reference to the resource key, the value of said key and the language in which it represents said key. Also has a Boolean flag that confirms the entity as being the chosen translation for this language.

ResourceVote
Represents a user’s vote on a specific translation value.

ResourceComment
Represents a user’s comment on a specific translation value.

More detailed information as for the contents of the entities, the names of the fields and the relationships between the entities can be found in the appendix chapter in the form of an entity relationship diagram.

4.5. General project structure

According to the designed architecture, the project is structured in the following way:

- backend folder. Contains all the backend running on Django and DRF.
- frontend folder. Contains the frontend client running on VueJS.
- docs folder. Contains documentation regarding the project.

Such an approach allows us to have a mono repository on GitHub instead of having to split everything into a different organization with three sub-repositories.

During development, we will be executing the backend separately using Django’s development server. The frontend will also be executed separately, but using Webpack’s server. As for deployment, once the frontend is built it will need to be included either inside the backend or a separate server should be used. While Django has the ability to serve static files without any problems, we want to live open the opportunity to be able to be able to host the client separately. Therefore, a the deployment of the client as part of the backend will be in the deployment manual, but will stay separated at a project level.

4.6. Backend implementation

4.6.1. Django and DRF

Following the Django documentation, it was decided to separate translations into a separate independent module in the form of a “Django Application” [9]. The usage of Django Rest Framework forces a structure to the application.

The flow shown in Figure 9 represents the request-response cycle of a Django application. As we can see, the Django architecture provides a means of extending any component via middleware software that gets loaded upon startup. In the case of this project, the addition of DRF extends the View middleware. The upper part of the flow (request and response pre-processing, URL resolution) are left untouched, instead the middleware manipulates the view level where the data is parsed, analyzed, processed and a value for a response is created.

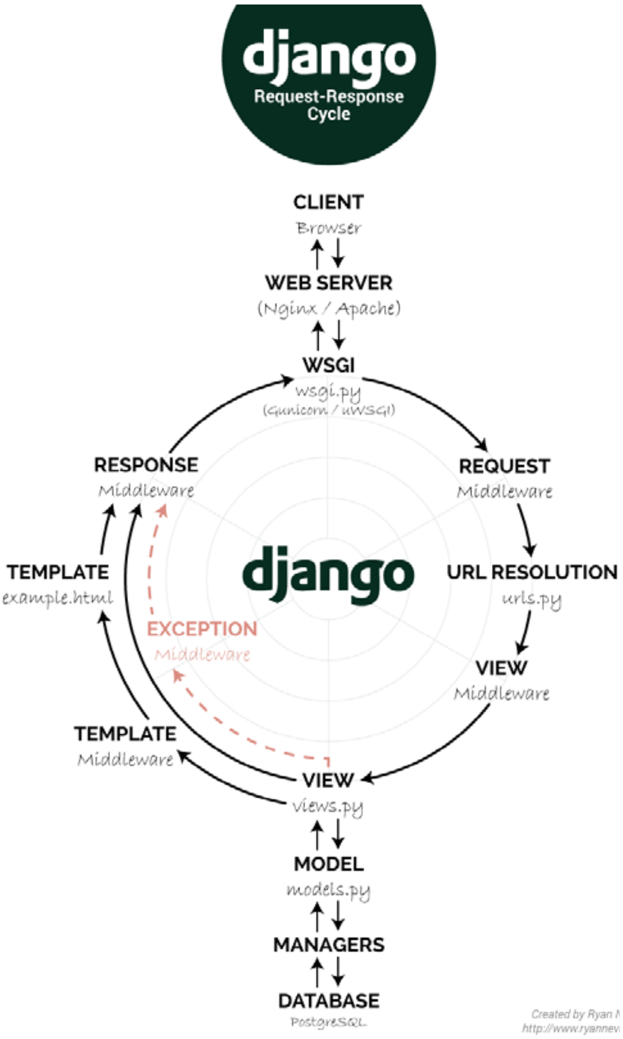


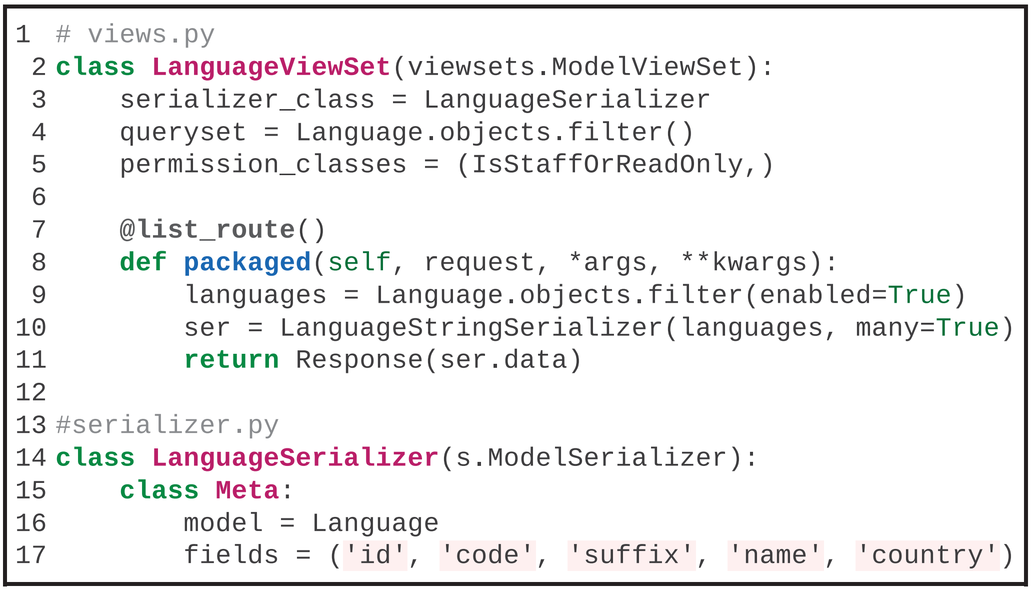
Figure 9. Django cycle

The DRF middleware consists of the following components:

- Views. These are the main handlers for the requests. At this level permission checks are performed, content parsers are applied based on the headers, database queries are executed.
- Serializers. This component converts the data from and to the format indicated in the request. The conversion can be to a custom object or directly to a database entity. The latter follows the Django cycle and invokes the model component with its responding object managers that generate the queries, execute them against the database and return a response.

Overall, as was stated during the technological evaluation, DRF is an extension to Django that comes with full batteries included and requires little to no reimplementation for simple CRUD operations.

While we will not cover all the specific implementations for all API methods, an extraction of the Language view is provided in Code Snippet 1 below.



```
1  # views.py
 2 class LanguageViewSet(viewsets.ModelViewSet):
 3     serializer_class = LanguageSerializer
 4     queryset = Language.objects.filter()
 5     permission_classes = (IsStaffOrReadOnly,)
 6
 7     @list_route()
 8     def packaged(self, request, *args, **kwargs):
 9         languages = Language.objects.filter(enabled=True)
10         ser = LanguageStringSerializer(languages, many=True)
11         return Response(ser.data)
12
13 #serializer.py
14 class LanguageSerializer(s.ModelSerializer):
15     class Meta:
16         model = Language
17         fields = ('id', 'code', 'suffix', 'name', 'country')
```

Code Snippet 1. DRF Language view and serializer

On line 2 the LanguageViewSet is defined as a class that inherits from ModelViewSet. ModelViewSet is a helper base class that has predefined workflows for CRUD operations on a model. On line 2 a serializer is defined, the LanguageSerializer. The language serializer, as noted before, is a critical part of connecting the existing model to DRF. We use a special LanguageSerializer that is defined separately.

On line 4 a query set is defined. A queryset is a base definition for all the queries that will be done via the Django ORM. Line 5 defines the permission classes. Permission classes allow fine-grained control on the access to different views. In this example, the permission classes are set to only the “IsStaffOrReadOnly” class. This class checks that the coming request was done by an authenticated user that is part of the staff (administrator group) or that the request has a read only verb such as GET, OPTION, or HEAD.

Custom API operations can also be defined. Lines 7-9 define a “packaged” API method on the language subset. This method returns all languages that have the enabled status in a single response.

Lines 14-17 define the language serializer that is used for the Language view. Lines 16 defines the model that is used for this serializer, while on line 17 a set of fields to serialize is provided. The “fields” field purposefully excludes an existing field on the model named “enabled” and only extracts data from the ones provided.

Such a small amount of code allows us to completely define and fine-grain the CRUD operations we want to be available through the API. A small addition to the URL resolver will provide three possible paths for the API:

- /api/trans/languages/. Upon GET requests, returns a list of all available languages. POST requests allow to create a new language.

- /api/trans/languages/<pk>/. GET requests return a language with the provided ID, PUT requests allow to modify the language with said ID and DELETE requests will remove that language form the database
- /api/trans/languages/packaged/. GET requests will return the list of all available enabled languages in a single query.

The other API paths do not differ much from the Language example. The only big difference is that they are nested as such: /api/trans/projects/<project_pk>/sets/<set_pk>/. The provided path corresponds to a resource set with the id set_pk as part of a project with id project_pk. Nesting paths in this way allows for better navigation of an API as well as enforces a structured approach to design.

Sequence diagrams as well as workflows describing the interaction with the API can be seen in the Appendix chapter, “API paths” part.

4.6.2. Backend structure

In order to be able to stitch everything together on the backend side, we are going to use the recommended directories by the Django documentation. Table 2 represents the general structure of the backend. The *config* folder is the main Django project folder and therefore contains the settings and WSGI handler. The *urls.py* file in this folder defines the high-level routes available.

The *apps* folder is where all of the application logic will be placed. In order to be able to better separate concerns, we will be splitting the account management and translations parts of our application into two different Django applications named accordingly. This way we can replace the accounts module without any problems to adapt to the needs of other developers.

```
├──── Makefile
├──── Procfile
├──── apps
│   └──── api
│       ├──── accounts
│       ├──── shared
│       └──── translations
├──── config
│   ├──── settings.py
│   ├──── static
│   ├──── urls.py
│   ├──── views.py
│   └──── wsgi.py
├──── data
│   └──── db.sqlite3
├──── manage.py
├──── package-lock.json
├──── requirements-dev.txt
├──── requirements.txt
├──── static
└──── tools
```

Table 2. Backend structure

4.6.3. File storage and processing

Due to the need to extract data from resource files, a file processor is needed. Due to the amount of different file types available on different platforms, the decision to make a base parser and extend it with specific implementations is not viable. Instead, the application will provide an abstract class. Each parser will have to implement that abstract class and will be loaded dynamically. Same goes for the reverse processing when a file is downloaded.

The abstract resource file handler defines two methods: one importing and one for exporting resource files. The importing method accepts Django File object while the output provides the resource set model. This gives complete freedom as to the implementation of the importer and exporter methods. It is expected that the parser class will be interacting with the model directly, we do not want to gateway that part due to the different possible necessities that a parser may have.

As for file storage, the application completely relies on Django's file storing capabilities. Due to Django's modularity, we do not have to worry about the possible extension of that storage with some sort of third party middleware as our calls will be high level enough to be handled by that middleware.

4.6.4. Authorization and authentication

Due to the decision of the frontend being a single page application, JSON Web Tokens[18] (JWT) were chosen as the best possible choice. On the backend side of things JWT has standard compliant DRF extension libraries that allow for the authentication through sole use of JWT tokens. Therefore we will take off the weight of having to reimplement the authentication process and use an industry tested solution.

The use of JWT also allows us to store any kind of information in the "payload" part of the token. In our implementation we will be passing the user identifier, email and username. This way there is no need to request them separately and they are always available as soon as the user authenticates in the application.

4.6.5. Initial language population

The initial language population, due to the constraints to locales, was decided to be done via a list provided by 2xlibre.net[19]. The *tools* folder of the *backend* part contains a locale parser tool that will parse all available locales on the website and add them to the database through Django.

A link to a gist containing a generic JSON extractor can be found in the Appendix chapter in the "External Links" part.

4.7. Frontend client implementation

4.7.1. VueJS and extensions

VueJS follows a forces a simple approach code structuring. The application is split into different components which then are combined to show different pages. Due to this being a highly hierarchical approach, all implementations must have a base "App.vue" component which serves as the main layout of the application.

A .vue in itself, represents an isolated Vue component by consisting of three parts: template, scripts and style. As can be noted from the names of those parts, they correspond to the HTML code that is rendered, the scripts that are executed and the styles for the HTML code. Components can include other components and render them inside.

[18] https://jwt.io/
[19] https://lh.2xlibre.net/locales/

Vue’s main recommendation as for building the client into a self-contained application is to use Webpack. Upon issuing a build command, Webpack will analyze and collect all used dependencies in all referenced components and libraries. Once that is done, Webpack will copy, compress and link those dependencies in a recursive manner making sure that all referenced libraries and components are being included in the build. This enables worry-free development as once configured the developer does not need to repeat tedious build tasks.

Having selected VueJS as the frontend client, therfore the web client being a single page application, we have to take into account that VueJS on its own represents only the presentation level. The framework itself does not have path routing or state management. Both of this problems can be solved with middleware that is officially supported by the VueJS project: vuex and vue-router.

vuex allows us to manage an application state for VueJS. As is the framework itself, vuex is reactive and propagates events when the underlying values change while at the same time giving complete freedom on what information to store and how to store it. In our implementation we will be using vuex coupled with local storage APIs to be able to add persistence to our client.

vue-router enables the application to link different URL paths to different components within our client, deny access and redirect upon entering restricted zones and other functionalities that are expected from a router.

4.7.2. Frontend client structure

As a starter boilerplate, we used an example from GitHub[20]. This example project comes with a basic Webpack configuration tailored to be used with VueJS. Although, this boilerplate was edited in order to update to the latest Webpack version and fix some minimization libraries issues.

The *config* and *build* folders are related to webpack. The *dist* folder will contain the files once they are built. The *src* folder is where the we develop code resides. Inside of it we have the following subfolders:

- *api*. Contains the clients needed to interact with the backend API. Structured into different files per functionality.
- *assets*. Contains CSS files, images and other content.
- *components*. Contains the actual components of our application: pages, individual reusable elements, etc.
- *router*. Contains our router configuration and defined router for the application.
- *store*. Contains the configuration, store definitions, actions, getters and mutators that vuex uses.

```
├──── README.md
├──── build
├──── config
├──── dist
├──── index.html
├──── package-lock.json
├──── package.json
├──── src
│     ├──── api
│     ├──── assets
│     ├──── auth.js
│     ├──── components
│     ├──── eventbus.js
│     ├──── main.js
│     ├──── router
│     ├──── store
│     └──── utils.js
├──── static
└──── test
```

Table 3. Frontend structure

[20] https://github.com/prograhammer/example-vue-project

4.7.3. Components

We structure our components into pages and reusable components. Reusable components include things like loading spinners, form error messages, page errors, not found errors, custom buttons and empty pages that are used by routing components.

Code Snippet 2 contains the code for the loading spinner component as an example. As we can see, the template part of the component is simple HTML with special tags. The script part also contains conventional Javascript code. There is a requirement for components to implement *export default*, so that VueJS can setup this component appropriately. The style tag can contain CSS, but in our case we are using SCSS.

Other components and pages are defined in a similar way and only vary in size and complexity.

```
<template>
  <div>
    <div class="...">
      <div class="spinner-content">
        <svg class="spinner" ... >
          <circle class="path" ... />
        </svg>
        <br>
        <div class="spinner-message">
                {{ message }}
        </div>
      </div>
    </div>
  </div>
</template>
<script>
  export default {
    name: 'spinner',
    props: {
      'message': {
        type: String,
        default: 'Loading...'
      }
    }
  }
</script>
<style lang="scss" scoped>
  ...
</style>
```

Code Snippet 2. Spinner component

4.7.4. States and routing

We split the vuex states into two stores: one for authentication information and one for translation.

The authentication store contains the JWT token and the user information extracted from the token in order to facility easier access to it. The JWT token is saved to the local storage of the browser every time it gets updated.

The translation storage contains the information specific to the translation workflows like the list of available languages, temporary page states and object cache. This state also makes some of the data available in it persistent via synchronizing it with the local storage. Languages are persistent as they rarely change, but we do provide them with an expiration date so that the client re-queries them as needed. Starting the resource creation flow will also reset the list of languages.

Routing is dependent on the authentication store. Depending on whether the JWT is present, the user either has or not access to modification actions. A similar approach is used to block off the administration area. Even if the user tampers with the stored authentication information and gets access to the administration page, due to the JWT being required for every API request, he will not be able to interact with that part of the service in any way.

The client route map and available pages are available in the Appendix chapter, in the “Client Route Map” part.

5. Developed project review

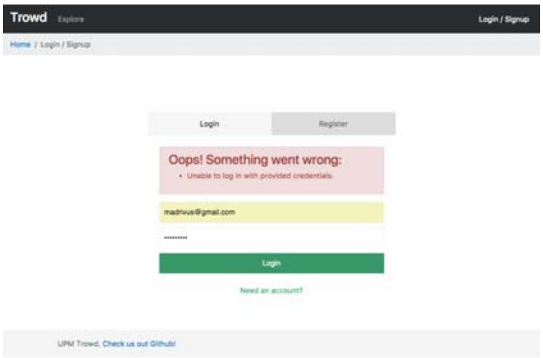
5.1. Resulting project

The result of the development is hosted on Heroku[21]. You can also check other links related to this project in the Appendix chapter, “Client Route Map” part.

On Figures 10 and 11, we can see examples of the modular components client interacting with our backend. In the presented Login page the navigation bar as well as the footer were loaded only once from the server and then only the contents were updated. The error form also was not rendered upon the initial loading of the login page and only rendered upon the page’s update. An important part is that this is not a mere difference of toggling the styles on the element: it is not physically present during the first render of the page. This is an important fact this also has an impact on the DOM tree.

On Figure 11 you may note that the upper navigation bar has changed from a “Login / Register” label to buttons leading to the dashboard and logging out. Again, this was done without reloading the whole page. In fact, as soon as all the initial assets are received, the only interactions with the server that remain are loading styles for specific styles. The network footprint of the application is therefore minimal. Once the user has been on a page it should be cached by the browser and therefore the content will not be reloaded unless there is a change server-side or the cache expires.

Figure 12 shows the concept of reactive changes. Due to the data introduced in the textbox is considered a model for the title on top, once the data in the textbox changes the title changes as well. Such a reactive approach is very valuable due to the fact that



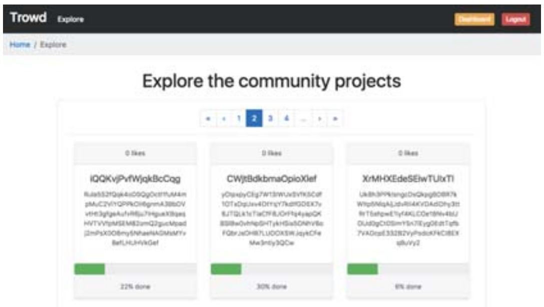
Oops! Something went wrong:

- Unable to log in with provided credentials.

madrivus@gmail.com

****

Figure 10. Login page



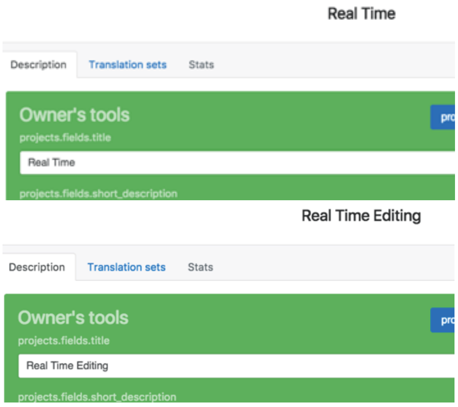
Explore the community projects

iQQKvjPvfWjqkBcCqg

CWjtBdkbmaOpioXlef

XrMHXEdeSElwTUlxTl

Figure 11. Exploration page



Real Time

Real Time

Real Time Editing

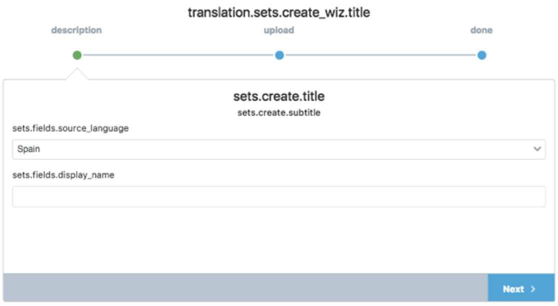
Real Time Editing

Figure 12. Real time details editing

[21] https://upm-trowd.herokuapp.com/

it allows the user to preview the changes before actually submitting them and modifying the database entries.

After the creation of a project the user will be prompted to upload a new resource file. This involves a wizard that will step by step provide forms to fill in as displayed on Figure 13.



translation.sets.create_wiz.title

sets.create.title

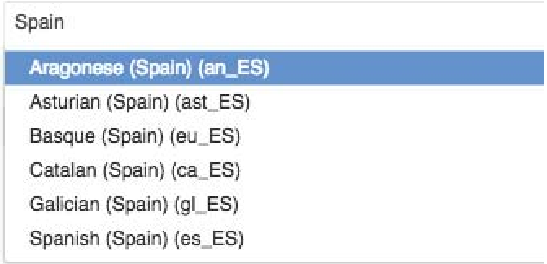
sets.create.subtitle

sets.fields.source_language: Spain

sets.fields.display_name:

Figure 13. Project creation wizard

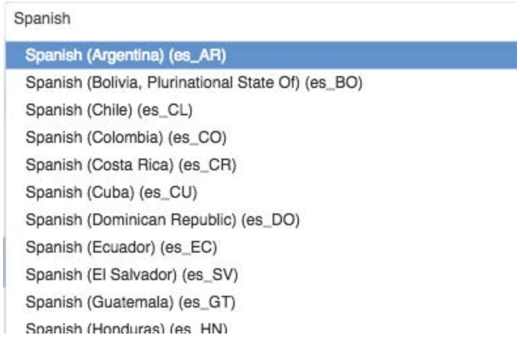
All language forms on the website use an isolated component that has filtering due to the list of languages being very long. An example of using said component can be seen on Figure 14. This is where it is important to note that the original idea with implementing locales instead of languages played well. As it is seen, introducing "Spain" into the search bar found us all languages that are used in Spain. On Figure 15 we see similar results upon searching for "Spanish", we get all regional Spanish variations. Again, this list is extendable through the administration part of the service.



Spain

- Aragonese (Spain) (an_ES)
- Asturian (Spain) (ast_ES)
- Basque (Spain) (eu_ES)
- Catalan (Spain) (ca_ES)
- Galician (Spain) (gl_ES)
- Spanish (Spain) (es_ES)

Figure 14. Project languages. Spain



Spanish

- Spanish (Argentina) (es_AR)
- Spanish (Bolivia, Plurinational State Of) (es_BO)
- Spanish (Chile) (es_CL)
- Spanish (Colombia) (es_CO)
- Spanish (Costa Rica) (es_CR)
- Spanish (Cuba) (es_CU)
- Spanish (Dominican Republic) (es_DO)
- Spanish (Ecuador) (es_EC)
- Spanish (El Salvador) (es_SV)
- Spanish (Guatemala) (es_GT)
- Spanish (Honduras) (es_HN)

Figure 15. Project languages. Spanish

The main part, of course, is the translation editor. Due to the qualities described above, the translation page is smooth. Even after loading 1000 resource key entries at once, the page felt fluid and stable. The payloads being sent between the server and the client are very small in size. When a user is loading a translation key that has tens of different proposals, the once the loading spinner was hidden the interactions where instantaneous.

All of this is possible thanks to the well-structured backend implementation. The client consumes the API without any problem, interacting with the service in a transparent manner. The API documentation is also available directly via the API itself.

The resource file parsers for importing and exporting ended up being pretty self-contained and at the same time allow for further extension. As of now, the service allows the importation and exportation of the following formats: JSON, Apple Strings file, Android-centric XML localization files, Windows-centric XML localization files.

5.2. Positives and negatives

The backend and frontend sides ended up working very well together, even though they are completely decoupled. The backend can have different clients, the frontend could attach to different backends. Altogether, the service ended up being close to what was initially expected. It is fast, simple, and is lightly coupled. Most of the positives are already covered in the previous section.

We ended up with a service that could potentially be used by anyone. The service will be self-hosted via Heroku, so that developers that do not want to invest in hosting the service can also use it.

As for other negatives, it's not like they are nonexistent. We will go through them point by point:

- Languages. We lack translations for many languages and we lack the languages themselves, luckily, that part is easily expandable.
- Frontend client. Due to having a small amount of experience in developing front end parts of the applications, it ended up being pretty generic. Bootstrap without any theming was used for the styling. There are many areas in which the user experience and the user interface in general could be better.
- Resource file parsers. The file parsers ended up being a part of the Django application itself, therefore the application may take a hit when a big number of files is parsed at the same time.
- Single sign on authentication. Right now the platform requires the user creating a new account on the platform when he signs in.
- Socialization. At the current state the application lacks some sort of socialization features. We want to users of our platform to interact on things that interest them.

These may seem like small negatives, but each of them has a sufficient weight that makes the platform not up to par.

5.3. Documentation

Documentation on setting up a local instance, populating the database with initial languages, deployment and ways to contribute to the project can be found on the project's wiki page on Github[22]. For other URLs related to the project, please refer to the Appendix chapter, "External URLs" part.

[22] https://github.com/lances101/UPM-Trowd/wiki

6. Conclusions

6.1. Objectives

The main objective of this project was to create an open source platform that would facilitate the interactions between software developers and translators. The applications already have big fan bases that could contribute to the translation of their loved application. What our service needs to be is something that allows connecting people.

The service provides a platform to which developers can attract their users in order to able to translate their projects. The resulting product's aim is not to gain money, it is to make a place were people can make applications they use better. The platform is free, open and available for everybody to use as they wish. Applications do not need to be open source in order to be translated. The translation itself can also be exclusively private and allow access only to invitees.

I consider that the goal has been reached. The main concern about the platform as a hosted service and as a self-hosted tool, being it an open source project is "how popular will it be?".

6.2. Future work

As was stated in the positives and negatives section, there are a series of improvement that could be done in order to make the project even better.

More included languages

As of the initial release of this project, there are 330 languages that are included in it. While that is a big number, out of those only 195 entries are unique. This can be expanded. GitHub contributions as well as user contributions are the options.

Language requests

Users and developers should be able to request new languages to be added directly to the platform. While languages added via GitHub may eventually end up in the deployed version, it would be of great help to be able to submit new languages via the service itself.

UI / UX improvements

The user interface as well as the overall user experience can and should be improved. While the initial client provided with the application does cover all the basic interactions, a contribution of a specialist that could actually design a better flow would be good for the project.

Distributed tasks

Right now the resource files are parsed as soon as they are uploaded to the service. While this does not have a big effect on functionality, it does have a negative one on performance. A distributed task queue would be a perfect solution to this project. Celery[23] is an option.

[23] http://www.celeryproject.org/

Third party authentication

Integration with Google, Facebook, GitHub, Twitter and other services would facilitate the authentication of the users.

Socialization

While the service does provide with comments, votes and exploring, integrating discussions for projects would be a step in the right direction.

7. Bibliography

[1] Statista, "Market size of the global language services industry 2009-2020," [Online]. Available: https://www.statista.com/statistics/257656/size-of-the-global-language-services-market/. [Accessed June 2017].

[2] SIL International, "Summary by language size," [Online]. Available: https://www.ethnologue.com/statistics/size. [Accessed 06 2017].

[3] Smart Insights, "Mobile Marketing Statistics compilation," 05 2017. [Online]. Available: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/. [Accessed 06 2017].

[4] AWIO Web Services LLC, "Browser & Platform Market Share June 2017," 05 2017. [Online]. Available: https://www.w3counter.com/globalstats.php?year=2017&month=5. [Accessed 06 2017].

[5] Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Software_requirements_specification.

[6] IEEE, "IEEE 29148," [Online]. Available: http://sebokwiki.org/wiki/ISO/IEC/IEEE_29148.

[7] K. E. Wiegers, "Software Requirements Specification Template," 1999. [Online]. Available: http://www.cse.msu.edu/~chengb/RE-491/Papers/SRSExample-webapp.doc.

[8] SQLite, "SQLite," [Online]. Available: https://www.sqlite.org/.

[9] Django, "Django Applications," [Online]. Available: https://docs.djangoproject.com/en/1.11/ref/applications/.

[10] International Telecomunications Union, "ICT Facts and Figures 2016," June 2016. [Online]. Available: http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2016.pdf. [Accessed June 2017].

[11] Chalmers Technical University, "Software Requirements Specification," [Online]. Available: http://www.cse.chalmers.se/~feldt/courses/reqeng/examples/srs_example_2010_group2.pdf.

8. Appendix

8.1. Project related URLs

Note: the URLs hosted on Heroku may be unavailable from time to time due to automated redeployment

API Root
https://upm-trowd.herokuapp.com/api/

API Documentation
https://upm-trowd.herokuapp.com/api/docs/

Client
https://upm-trowd.herokuapp.com/

Trowd GitHub Repository
https://github.com/lances101/UPM-Trowd

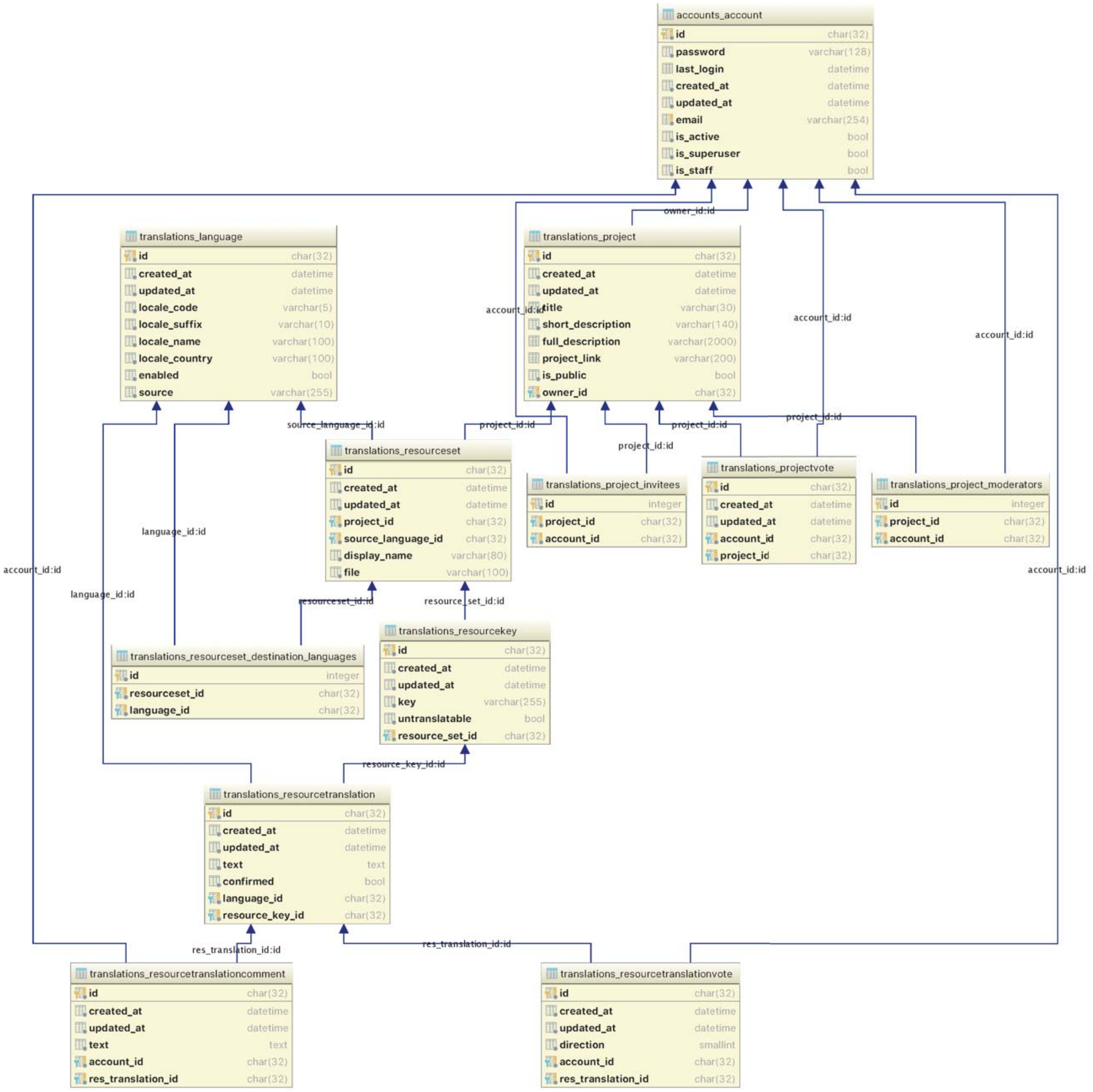
Trowd Wiki
https://github.com/lances101/UPM-Trowd/wiki

Trowd Wiki – Development guide
https://github.com/lances101/UPM-Trowd/wiki/Development

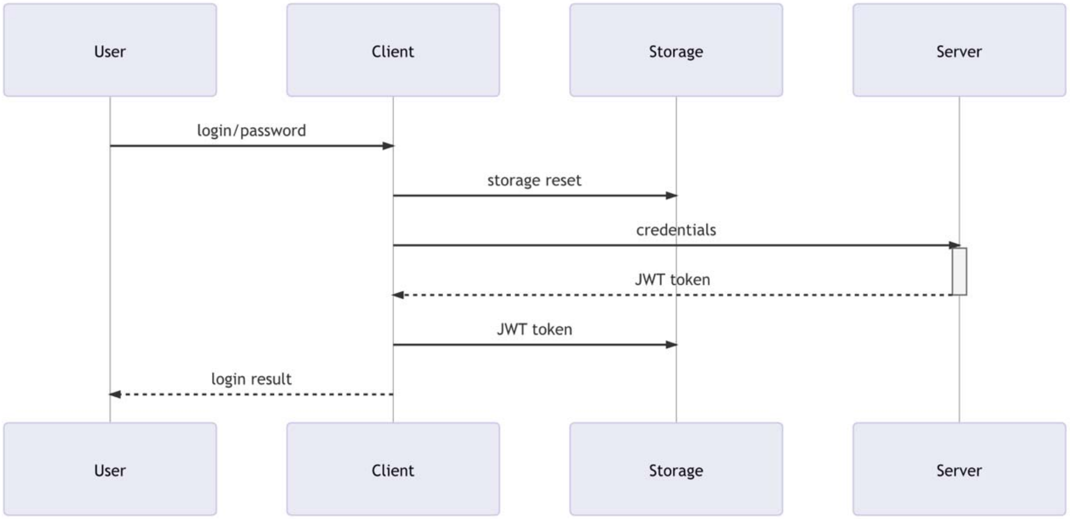
Trowd Wiki – Deployment guide
https://github.com/lances101/UPM-Trowd/wiki/Deployment

Trowd Wiki – Contribution guide
https://github.com/lances101/UPM-Trowd/wiki/Contribution

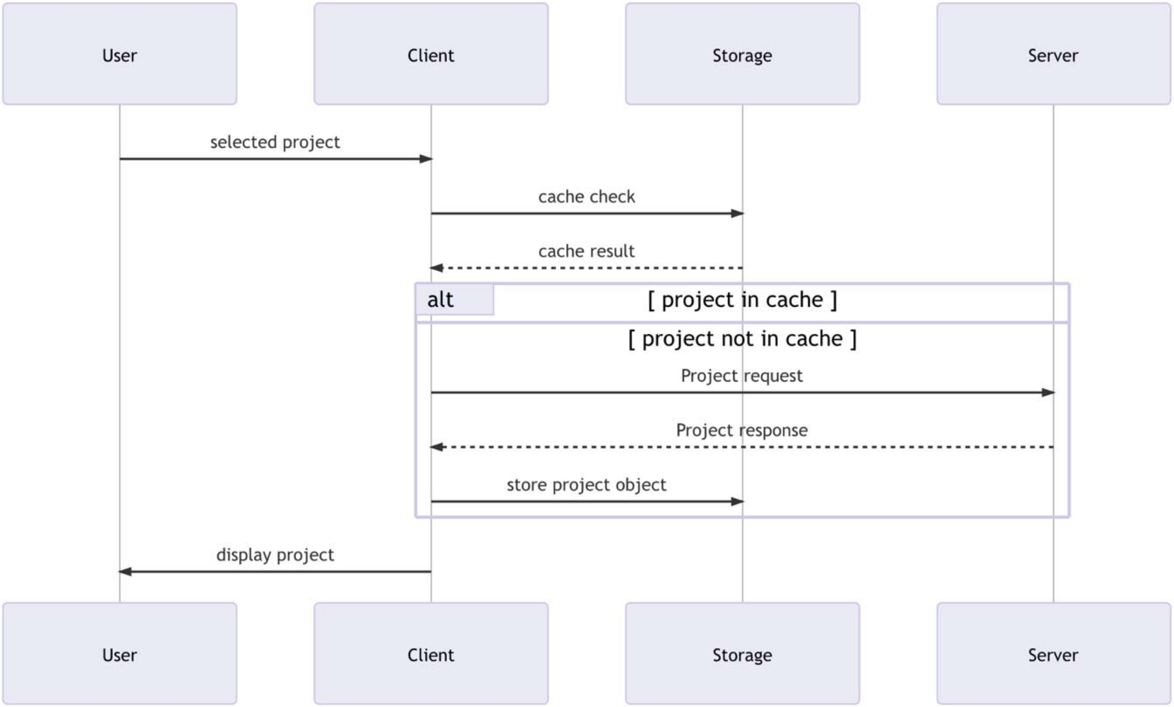
8.2. Entity Relationship Diagram (ERD)



8.3. Authorization sequence diagram



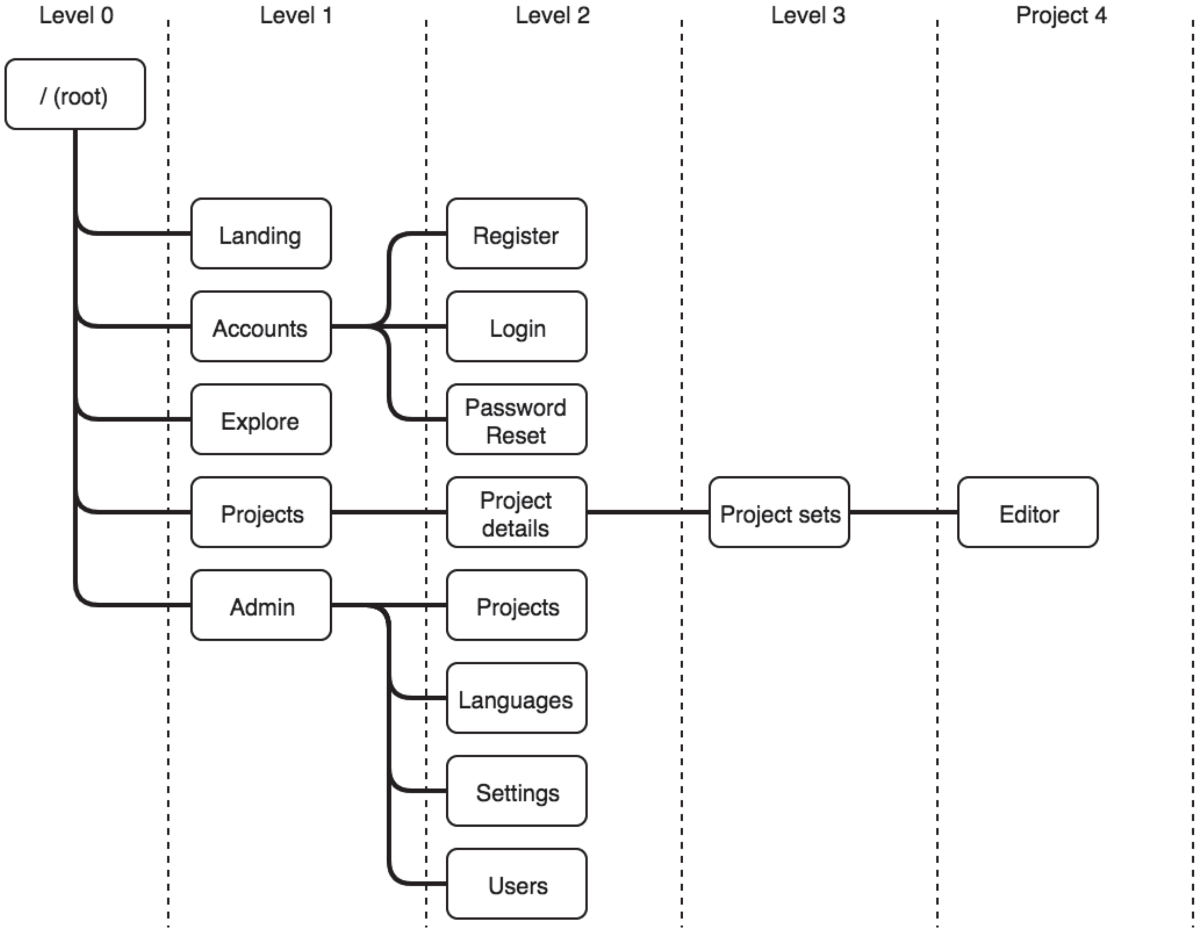
8.4. Project details sequence diagram



8.5. API paths

```
/api/docs/                                                    drfdocs
/api/accounts/                                                accounts:create
/api/accounts/change_password                                 accounts:change_password
/api/accounts/jwt/login/                                      accounts:jwt-login
/api/accounts/jwt/refresh/                                    accounts:jwt-refresh
/api/trans/languages/                                         translations:language-list
/api/trans/languages/<pk>/                                    translations:language-detail
/api/trans/languages/packaged/                                translations:language-packaged
/api/trans/projects/                                          translations:project-list
/api/trans/projects/<parent_lookup_project>/sets/             translations:projects-set-list
/api/trans/projects/<parent_lookup_project>/sets/<pk>/        translations:projects-set-detail
/api/trans/projects/<parent_lookup_project>/sets/<pk>/upload/ translations:projects-set-upload
/api/trans/projects/<pk>/                                     translations:project-detail
/api/trans/projects/explore/                                  translations:project-explore-list
/api/trans/users/<account__id>/                               translations:profile-detail
/api/trans/users/<account__id>/owned/                         translations:profile-owned
```

8.6. Client route map


Level 0
Level 1
Level 2
Level 3
Project 4
/ (root)
Landing
Accounts
Explore
Projects
Admin
Register
Login
Password Reset
Project details
Projects
Languages
Settings
Users
Project sets
Editor