



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

NEW GENERATION OF OPEN-SOURCE TOOL STRANGER STRINGS

NOVÁ GENERACE OPEN-SOURCE NÁSTROJE STRANGER STRINGS

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

MATEJ KŇAZÍK

SUPERVISOR

VEDOUCÍ PRÁCE

prof. Ing. ADAM HEROUT, Ph.D.

BRNO 2020

Bachelor's Thesis Specification



Student: **Kňazík Matej**
Programme: Information Technology
Title: **New Generation of Open-Source Tool Stranger Strings**
Category: User Interfaces

Assignment:

1. Study and describe the history and the current version of the tool Stranger Strings.
2. Identify opportunities and needs for radical rebuilding of Stranger Strings.
3. Propose necessary changes and implement them.
4. Test the created solution on users and improve it iteratively.
5. Evaluate the properties of the modified solution in a testing or preferably production use.
6. Evaluate the achieved results and propose possibilities for continuing the project; create a poster and a short video for presenting the work.

Recommended literature:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN-13: 978-0321657299
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- Nitish Singh: Localization Strategies for Global E-Business, Cambridge University Press, 2011, ISBN 9780511920226
- Dan Jurafsky and James H. Martin: Speech and Language Processing, <https://web.stanford.edu/~jurafsky/slp3/>

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Herout Adam, prof. Ing., Ph.D.**
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: May 28, 2020
Approval date: June 23, 2020

Abstract

The aim of this bachelor's thesis is to describe the development process of a web-based application called Stranger Strings. This project got its name from in-house software but apart from the name everything was changed. The purpose of Stranger Strings shifted, the whole code-base was rewritten and turned into an open-source project. The same applied to the user-interface which was redesigned to target a much broader spectrum of users. This paper provides in-depth insight into the aforementioned process and also explains how user testing was used during the development process to identify main problems. The outcome of this work is an open-source localization analyzing tool that helps users to improve the quality of their localization.

Abstrakt

Cieľom tejto bakalárskej práce je opísať vývoj webovej aplikácie s názvom Stranger Strings. Projekt si ponechal svoje meno po internom firemnom nástroji, avšak vo všetkom ostatnom prešiel kompletnými zmenami. Účel Stranger Strings sa pozmenil, kód bol od základu prepísaný a celý projekt bol zmenený na open-source. Rovnako bolo redizajnované aj užívateľské prostredie za účelom rozšírenia cieľovej skupiny užívateľov. Táto práca podrobne popisuje tento proces a tiež vysvetľuje, akú úlohu zohralo užívateľské testovanie pri identifikácii hlavných problémov aplikácie. Výsledkom tejto práce je open-source nástroj na analýzu lokalizácie, ktorý pomáha užívateľom zlepšovať jej kvalitu.

Keywords

Stranger Strings, localization, quality, inconsistencies, mistakes, analysis, QA, testing, style, translations, visualization, web, app, Firebase

Klíčová slova

Stranger Strings, lokalizácia, kvalita, nekonzistentnosti, chyby, analýza, QA, testovanie, štylistika, preklady, vizualizácia, internet, aplikácia, Firebase

Reference

KŇAZÍK, Matej. *New Generation of Open-Source Tool Stranger Strings*. Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Adam Herout, Ph.D.

Rozšírený abstrakt

Lokalizácia sa v globálne prepojenom svete stala neoddeliteľnou súčasťou mnohých produktov, spoločností či služieb. Lokalizácia má významný dopad na užívateľov a preto je dôležité ju nezanedbať a urobiť ju správne. Je pomerne bežné naraziť na chybu v obsahu alebo v lokalizácii pri prehliadaní webových stránok.

Táto bakalárska práca popisuje vývoj aplikácie Stranger Strings, ktorá je určená na analýzu a správu lokalizácie. Stranger Strings dokáže detekovať rôzne formy chýb a nekonzistentností. Prostredníctvom ďalších nástrojov upozorňuje na dodržiavanie overených postupov, aby sa predišlo lokalizačným chýbam aj v budúcnosti. Nástroj poskytuje aj nástroj na nahlasovanie nájdených chýb, čím zjednodušuje a urýchľuje celý proces opravy chýb v lokalizácii.

Vznik Stranger Strings bol inšpirovaný interným nástrojom, určeným na vyhľadávanie a nahrávanie prekladov do lokalizačnej databázy, po ktorom si ponechal názov. Pôvodný Stranger Strings bol určený len pre úzku skupinu programátorov, ktorí si chceli zjednodušiť prácu súvisiacu so lokalizáciou webstránky. Bol vyvinutý na mieru pre špecifické požiadavky užívateľov. To sa odzrkadlilo ako na zdrojovom kóde, tak aj na samotnom užívateľskom rozhraní.

Nová verzia Stranger Strings je určená pre omnoho širšiu skupinu používateľov. Cieľová skupina už nie sú len samotní programátori, ale aj menej technicky zdatní užívatelia ako tester, prekladatelia, produktoví či lokalizační manažéri. Účelom tejto aplikácie už nie je len poskytnutie nástroja na prehľadávanie lokalizačnej databázy, ale aj jej analýza za účelom detekcie chýb a nekonzistentností. Od Stranger Strings sa očakáva, že dokáže pomôcť zvýšiť kvalitu lokalizácie. Cieľom projektu bolo poskytnúť túto aplikáciu komukoľvek formou open-source tak, aby bola využiteľná pre ľubovoľný lokalizačný projekt.

Pri tvorbe novej verzie Stranger Strings boli využité poznatky z praxe, týkajúce sa častých chýb v lokalizácii. Na ich základe boli odvodené možné príčiny a pravidlá, ktoré zamedzujú výskytu najčastejších chýb v lokalizácii. Na implementáciu bolo využitých množstvo najnovších technológií z oblasti webového vývoja. Stranger Strings je navrhnutý tak, aby dokázal spoľahlivo fungovať aj pri veľkých lokalizačných projektoch. Celý projekt bol od samého začiatku vyvinutý a zdokumentovaný tak, aby sa verejnosť mohla podieľať na jeho ďalšom vývoji.

Pri návrhu a tvorbe užívateľského rozhrania bola využitá iterácia užívateľských testov. Vďaka nim boli odhalené zásadné užívateľské problémy a odvodené rozhodnutia pri vývoji jednotlivých funkcií. Aplikácia bola následne nasadená do firmy, kde bola sledovaná analytickými nástrojmi, ktoré zbierali dáta o jej užívaní.

Zobierané záznamy o užívaní ukazujú, že projekt Stranger Strings napĺňa účel, ku ktorému bol vytvorený. Meranú aplikáciu navštívi v priemere 7.97 jedinečných užívateľov za týždeň a 16.53 jedinečných užívateľov za mesiac. V rámci jedného lokalizačného projektu Stranger Strings odhalilo 51 problémov, ktoré boli cez túto aplikáciu nahlásené na okamžitú korektúru.

Výsledkom bakalárskej práce je nástroj, ktorý je úspešne overený v praxi. Samotný nástroj je k dispozícii komukoľvek ako open-source projekt na stránke <https://github.com/kiwicom/stranger-strings>.

New Generation of Open-Source Tool Stranger Strings

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of prof. Ing. Adam Herout, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Matej Kňazík
June 23, 2020

Acknowledgements

I want to express great gratitude to my supervisor prof. Ing. Adam Herout, Ph.D. for giving me the opportunity to make Stranger Strings my bachelor's thesis and also for guidance and much valuable advice in the last 2 years. My sincere thanks also go to Kiwi.com, which provided me with resources and support in this project. A big thank you goes to my mentor and friend Pavel 'Strajk' Doleček, who introduced me to Stranger Strings and mentored me during the whole course of its development. Last but not the least, I am grateful to my family and friends for unceasing support and encouragement.

Contents

1	Introduction	2
2	Localization	3
2.1	Why is Localization Important	4
2.2	Common Problems in Localization	5
2.3	Previous Version of Stranger Strings	5
3	Design of the New Stranger Strings	8
3.1	Requirements & Key Features of Stranger Strings	10
3.2	UI Design	11
3.3	Technologies Used in Stranger Strings	14
3.4	Indications of Success	16
4	Implementation	18
4.1	Open-Source Project Overview & Installation	21
4.2	Inconsistencies Computation	24
4.3	Database	36
4.4	Front-end of Stranger Strings	38
5	User Testing	46
6	Conclusion	52
	Bibliography	53

Chapter 1

Introduction

This thesis is about the transformation of project Stranger Strings. Stranger Strings began as a small internal tool that provided an overview of localization data taken from an external translation management system (TMS). Seeing the whole localization at one place led to a realization that there are many mistakes and inconsistencies with repeating occurrence. This brought an idea about detecting those mistakes and inconsistencies and that is exactly when this project started.

After a thoughtful discussion with previous Stranger Strings authors, we concluded that old Stranger Strings is not suitable for this kind of purpose and it requires the development of a whole new project. The creation of a brand new tool from scratch provided new opportunities, such as a purely Open-source approach with modularity and customization in mind for future development. But before the whole development process, I needed to get some insight into the topic of localization and properly analyze all the possible mistakes and their causes. Another challenge was to learn how to create the app itself. What technologies should be used and also how to design Stranger Strings to be used by numerous people on a daily basis. Especially knowledge about the user testing later turned out as really valuable and changed the way how Stranger Strings was developed. The challenge was to display a lot of data and information without too much distraction that will discourage the user from using the tool. Also, the development itself brought many challenges such as performance issues and implementation difficulties. This thesis describes all of them and also provides explanations of how Stranger Strings solves them.

At the end of this thesis, you can learn about the usage stats of Stranger Strings and if the tool delivers the expected results.

Chapter 2

Localization

Many businesses reach a certain point, when they need to expand to foreign markets in order to grow further. To do so, they first need to know the market, the culture, local regulations, legal requirements, and of course local language. Only then they can bring their products or services closer to their client or customers. This process is also called localization.

According to the Globalization and Localization Association (GALA) term localization can be defined as a process of adapting a product or content to a specific locale or market. Localization aims to give a product the look and feel of having been created specifically for a target market, no matter their language, culture, or location [6].

Localization is more than just translating string from one language to another. The translation is only one of the many localization processes like units of measure conversion, currency conversion, date formats, design adaptation, and many more. A great example of localization is in Figure 2.1 which demonstrates localization of Pixar Animation Studios movie „Inside out“ for the Japanese audience.

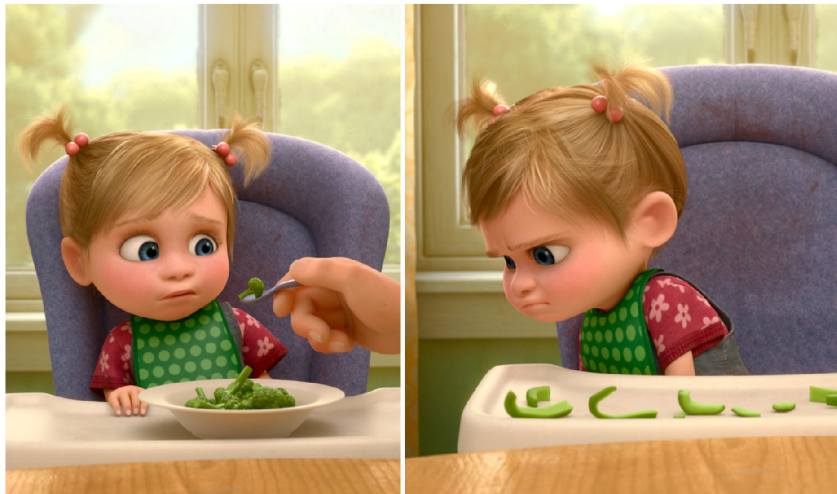


Figure 2.1: In one scene, Riley’s dad struggles to feed his daughter broccoli. Like most toddlers, she is disgusted by the vegetable, refusing to eat it. In Japan’s cut for the film, broccoli is replaced by bell peppers. This is because kids in Japan usually love broccoli, but instead they don’t like bell peppers. (source: Business Insider [1])

Another important requirement of a good localization is a good knowledge of the product or service itself. If someone doesn't know the purpose, meaning, or purpose of their product then they cannot properly localize it. Some companies emphasize on high quality copywriting, which can be easily ruined by improper localization. That is because for certain businesses the wording behind their products is very sensitive and even small differences in wording can cause a different perception of their brand.

Therefore it is important to periodically verify whenever the localization is being done properly. For this purpose there are many content management systems (CMS) or translation management systems (TMS) that groups all the localization in one place and helps users navigate through it.

2.1 Why is Localization Important

Is it enough to localize just for an English speaking audience? Is it worth bothering about other than non-native English markets? Do even users care about other localization once English localization is available to them?

First of all, according to a survey from 2006 with 2400 participants from 8 different non-Anglophone countries [4]. About 72.1% of internet users are more likely to stay on the website if the content is in their native language. More than half (52.4%) buys only at websites where the information is presented in their language. Even though two-thirds (67.4%) visit English-language sites frequently, just quarter (25.5%) regularly purchase goods or services at those properties. Even with information available in the local language, the inability to use their own credit cards or currency is enough for discouragement. The vast majority (85.3%) of respondents feel that having pre-purchase information in their own language is a critical factor in buying important purchases such as insurance or other financial services. Three out of four respondents (74.7%) agreed that they are more likely to repeat the purchase of the same brand again if the after-sales care is in their own language.

To put previous data into perspective, we need to look at Internet user distribution across the world. Figure 2.2 shows that vast majority of Internet users are from non-Anglophone countries. More than half of Internet users come from Asia. What is even more surprising, only 7.6% of Internet users are coming from the North America region.

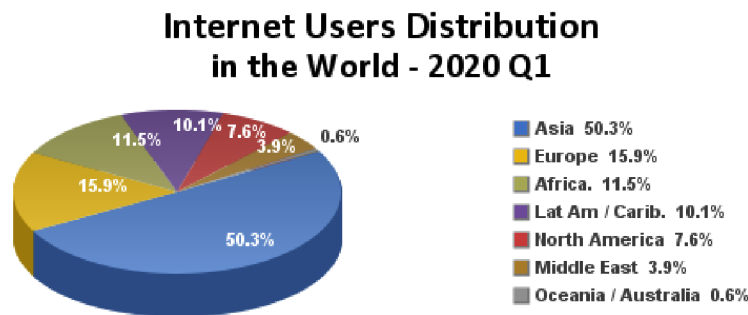


Figure 2.2: Distribution of Internet Users in the world by continents. (source: Internet World Stats: <https://www.internetworldstats.com/stats.htm>)

Undoubtedly, localization can bring significant value. The value can be in the form of higher sales, bigger audiences, more satisfied users, or even bigger global influence.

2.2 Common Problems in Localization

The easiest and cheapest way how to localize something is by using some form of machine translation. Even though services like *Google Translate* made a lot of progress in recent years, they are still far from perfect. Machine translation can get the job done for some use cases, however, it still cannot be considered as a proper localization solution. For now, machine translation is not able to produce translation while taking into consideration variables like the context of translation, intended mood, targeted audience, culture, legal affairs, etc.

Therefore most businesses decide to use human translations for their localization projects. Usually, each locale requires a different translator, as it requires deep knowledge of both original and targeted languages and respective cultures. This means that localization is usually outcome from the collaboration of multiple people. A collaboration of multiple translators can result in inconsistencies between translations. For example, each of the translators can choose different approaches in translating the same thing, while one of the approaches might be undesired. Localization teams try to avoid these scenarios by introducing some internal guidelines or glossaries but they can never cover all edge cases that eventually happen.

It is not easy to hire translators for each country, so the majority of companies tend to outsource localization to companies that are specializing solely on localization. This can introduce other issues, caused by unfamiliarity with the product that is being translated.

Another important thing to take into consideration is that human translators simply make mistakes because of their human nature. Translators daily translate a lot of sentences so it would be surprising if they never create some typo or just forget some colon.

It is not uncommon to find malformed HTML tags being part of the displayed content on some websites. The reason is that sometimes it is unavoidable for developers to include some HTML tags in the translation itself. Combined with the fact that it is usually not expected from translators to have deeper computer knowledge. Then it is not surprising that this kind of mistake can happen.

And it is not just an HTML tag that can be malformed. It is good practice to use placeholders in translation for dynamically changing values such as currencies, price, amounts, etc.. They are usually in some form of a variable that follows a certain pattern that can be recognized by computer via regular expressions or parser. This means that placeholders are equally sensitive to changes like HTML tags.

2.3 Previous Version of Stranger Strings

Stranger Strings was initially developed as an internal TMS tool. It was developed with an in-house software development approach. In-house software development means leveraging company resources to develop or implement software according to the company-specific requirements [15].

Even though the company used external TMS applications, it was limited by the number of accounts that can access it. Therefore Stranger Strings was dependent on another TMS as it took all the data from it. The tool was intended to be used primarily by developers so they could search and upload new translation keys with it.

The landing page was just a simple banner with a navigation bar and login button. It can be seen in Figure 2.3. The access to Stranger Strings was limited in the app only for users with company domain email.



Figure 2.3: Landing page in the first version of Stranger Strings

After successful authentication, the site led to a scene containing a table with all of the translation keys representing strings that are being translated. Apart from that table contained the English translations and the progress bar indicating how many locales are covered. The table can be seen in Figure 2.4. There was also a possibility to do search some keys and filtering based on some metadata from external TMS.

Key (showing 2419 / 2419)	Suspected errors	Progress	English
SS_strings_1		33	- In general, our flights are non-refundable. If you'd like to try cancelling, visit Manage My Booking Cancellations form. Our Claims Team will get back to you.
SS_strings_10		33	A transfer visa may be required at check-in between any connecting flight.
SS_strings_11		33	Additional baggage may be added below. Your fare details will outline what's included in the price.
SS_strings_12	Last	33	Add Checked Baggage
SS_strings_13	Last	33	Collect and recheck your baggage
SS_strings_14		33	Terms and conditions
SS_strings_2	First	33	- The possible refund amount always depends on the airline's policy. We charge a handling fee of and/or refunds of any kind.
SS_strings_3		33	Kiwi.com Guarantee
SS_strings_4	Placeholders	33	Worried about missing your connection? The __companyName__ Guarantee has you covered.
SS_strings_5		33	More Info
SS_strings_6	Last	33	- All changes must be made at least 48h before departure and are dependent on the airline's policy. Manage My Booking.
SS_strings_7		33	- The airline may charge additional fees for checked baggage or other optional services.
SS_strings_8		33	- Flight schedules and terminals are subject to change at any time by the airlines.
SS_strings_9	Last	33	- Correct travel documents are required. It is your responsibility to check your documents before
account.activate_watchdog		34	Turn On
account.active_watchdog	Last	34	On
account.add_bags		33	Added checked baggage quantity: __amount__
account.add_bags_action		33	Add __amount__ bag(s) for __price__
account.add_bags_after		33	After receiving payment, we'll process your order and email you when it's complete. It may take se

Figure 2.4: Table of translation keys in the first version of Stranger Strings

Click on any translation key opened modal that contained another table with all of the translation under the selected translation key. Additionally there were displayed metadata from external TMS with links redirecting to the external TMS.

Seeing all of the localization in one place revealed many inconsistencies and even errors that could be easily fixed by simple scripts. This led to an idea of detecting those inconsistencies with the Stranger Strings.

As an experimental feature there were added 4 checks. All of them were represented by „Suspected errors“ column in main table (Figure 2.4) and each of them had own column in table of translation key modal (Figure 2.5). For example Placeholder check was meant to detect whenever all of the translations under single translation key have the same number of placeholders. If not the placeholder badge was added to the main table. In the translation key detail table, column „Placeholders“ listed all of the placeholders inside respective translation. The same principle was applied to the other 3 checks, which were supposed to detect inconsistent character in the begging and end of translations and prohibited HTML tags.

Locale	Placeholders	First	Last	Tags	Translation	Badges
ar-AE	value	LETTER	LETTER		نظام المقاعد: __value__	
bg-BG	value	LETTER	LETTER		План на местата: __value__	
cs-CZ	value	LETTER	LETTER		Uspořádání sedadel: __value__	
da-DK	value	LETTER	LETTER		Sædeoversigt: __value__	
de-DE	value	LETTER	LETTER		Sitzanordnung: __value__	
el-CR					Not translated	
en-GB	value	LETTER	LETTER		Seat layout: __value__	
en-CU					Not translated	
en-US					Not translated	
es-AR	value	LETTER	LETTER		Disposición de asientos: __value__	
es-ES	value	LETTER	LETTER		Disposición de asientos: __value__	
es-MX					Not translated	
fi-FI	value	LETTER	LETTER		Istuimien aseteltu: __value__	
fr-FR	value	LETTER	LETTER		Disposition des sièges : __value__	
he-IL	value	LETTER	LETTER		מבנה המושבים: __value__	
hu-HU	value	LETTER	LETTER		Ülőhelyek elrendezése: __value__	
id-ID					Not translated	
it-IT	value	LETTER	LETTER		Disposizione posti: __value__	

Figure 2.5: Modal of selected translation key displaying all translations in the first version of Stranger Strings

As this tool was only used by developers these features were not being used. Apart from developers, nobody used Stranger Strings as they did not understand it. Therefore, the experimental features did not have any effect on how the Stranger Strings was used.

This bachelor thesis is about a complete transformation of this tool into one that will not just serve to another purpose but also will be available for everyone.

Chapter 3

Design of the New Stranger Strings

Previously Stranger Strings was intended to be used just by developers to find translations keys and add new ones. But new Strangers Strings will target primary translation managers, QA testers, and copywriters but still will be valuable for people who just want to search for translation keys and analyze translations. One of the main features of future Stranger Strings will be mistake and inconsistency detection. And not just that it will try to ensure that responsible users are seeing them and can easily resolve them. This is probably the biggest challenge for the front-end part of the application and will require careful UI/UX (user interface/user experience) design along with user testing. Finding mistakes in translations is quite hard and categorizing the severity of those mistakes is even harder especially since every user may have different needs even within the same localization project. To sum up, the new Stranger Strings aims to provide a tool that will help to manage translation projects and improve the quality of localization in multiple areas.

Intended redesign of Stranger Strings might sound like proposing the creation of another Translation Management System (TMS) when there are already many others TMS like Phrase¹ or Pontoon by Mozilla². The difference is that those tools are usually expensive or do not provide the detection of inconsistencies and mistakes in translations. Stranger Strings will be more like a translation analyzer tool and at the same time, it will offer a free and open-source alternative to other TMS. Additionally, anyone using some TMS with API or anyone with the possibility to export translation data will be able to insert Stranger Strings into their translation workflow with the existing module or create new and contribute to Stranger Strings project.

Old Stranger Strings was custom-tailored for specific needs which made the code impossible to reuse for new open-source versions. For instance, the architecture of the previous version was a bit chaotic. The company used a localization management tool Phrase. Additionally, *GitHub* was used for translation versioning and also as a cache to solve some problems with API restrictions of Phrase. Therefore front-end of the previous version of Stranger Strings was fetching data (mainly metadata) from TMS service (Phrase) and rest (translations, translation keys, and computed data) from Firebase Realtime Database. Firebase Realtime Database was filled with data by Google Cloud Functions which worked with data solely from *GitHub* repository and performed some basic computing what are the starting characters and what placeholders are represented in translation. Addition-

¹TMS by Phrase: <https://phrase.com/>

²TMS from Mozilla called Pontoon: <https://pontoon.mozilla.org/>

ally, there was also a possibility to report issues from Stranger Strings to *Slack*³. The old Stranger Strings architecture can be seen of Figure 3.1.

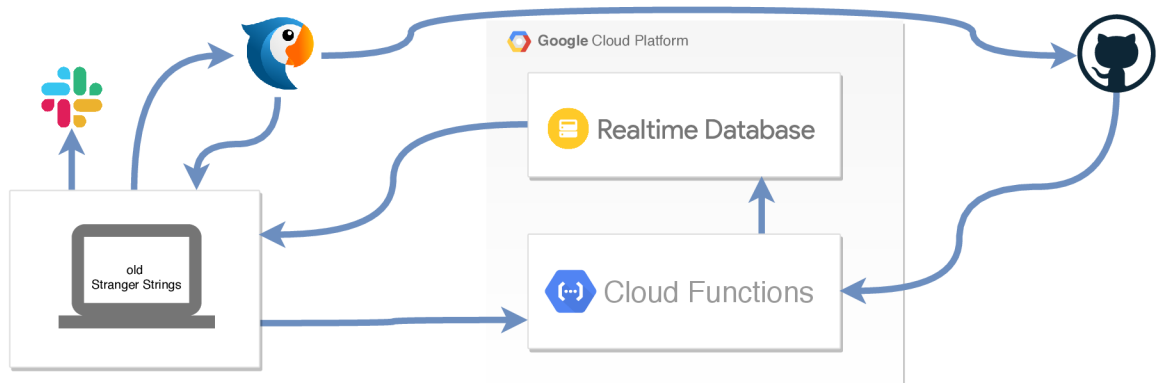


Figure 3.1: Old Stranger Strings architecture

At first sight, there is an evident problem that Stranger Strings are dependent on single TMS which is Phrase. Another problem could be the format of translation data stored in *GitHub*. Also *Slack* as a reporting destination could be useless for some users. The aim is to create an open-source tool that would be easily integrated into anyone’s workflow. Therefore new Stranger Strings required a more general and modular approach to architecture design.

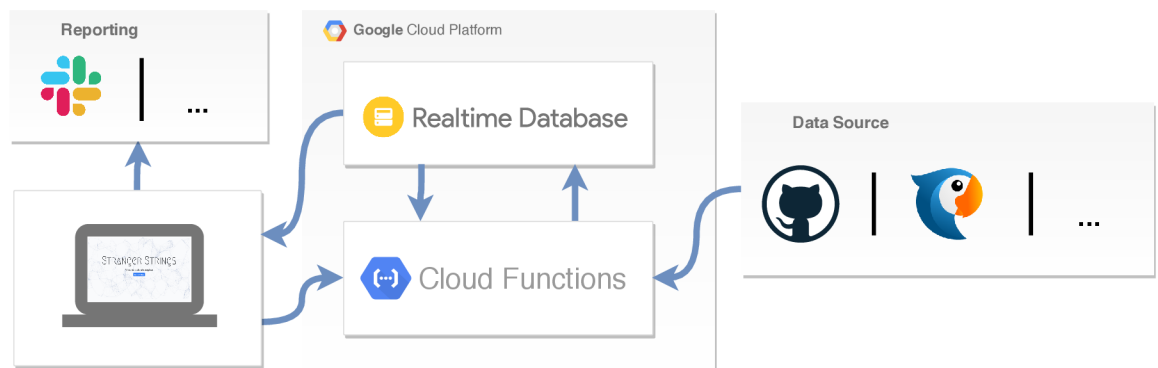


Figure 3.2: New Stranger Strings architecture

The new version of Stranger String will not be dependent on external services like *GitHub* or *Phrase*. Instead, it will always take data from a single data source selected by users. This approach will allow users to connect Stranger Strings to various external services providing translation data. This will be provided by data source modules which will create a clear interface for communication with Stranger Strings. Modules demonstrating this interface will be represented by the *GitHub* module and *Phrase* module. Those modules will provide an example for future open-source contributors on how to implement other data-source modules. Another module (e.g. *FTP* module, *Mozilla Pontoon* module, etc...) may follow in the future. Reporting translation bugs will be also solved with a modular interface. As a demonstration for this modular interface, there will be implemented *Slack* module. Also, an email module will be considered in the close future. The front-end and

³Slack: <https://slack.com/>

back-end of the application will be the same for every Stranger Strings instance. Front-end will display data taken only from its back-end database and apart from that, it will only be generating reports submitted by users. Back-end will consist of two parts, computation engine, and database. The computation engine will compute all inconsistencies and errors found in translations and write them into the database. It will also be responsible for detecting new versions of translations and keeping the database up-to-date. The proposed modular approach is visually represented in Figure 3.2.

3.1 Requirements & Key Features of Stranger Strings

As already mentioned the key features of the new Stranger Strings can be identified as a simple translation overview with the ability to search and mistake and inconsistency detection with an interface that helps to resolve those issues. Old Stranger Strings was able to search through translation keys but could not search for translation themselves at the same time. New Stranger Strings will have this ability to help the user easily map searched translation to its translation key and vice versa. The mistake and inconsistency detection will be performed by features called checks. In total there will be 9 different checks:

- **Placeholders check** detects missing / excess / inconsistent placeholders
- **HTML tags check** detects invalid / inconsistent / prohibited HTML tags
- **Values check** detects values, that could be replaced by placeholders, because values are likely to change over time and is easy to forget update all occurrences of specific value
- **First character check** detects inconsistencies of first character (e.g. translation begins with letter instead of dash)
- **Last character check** detects inconsistencies of last character (e.g. translation end with question mark instead of exclamation mark)
- **Length check** detects suspicious variations in length between same translations
- **Spelling check** detects spelling mistakes
- **Style check** detects usual stylistic issues (e.g. weasel words, repeated words, cliches...)
- **Insensitiveness check** detects words that might be considered as offensive or insensitive (e.g profanities, gender favouring, race related...)

In the previous version of Stranger Strings, there were experimentally implemented 4 of these checks (Placeholder check, HTML tags check, and first/last value check). Placeholder check was implemented quite well but it was hard-coded to only detect placeholders with double underscore notation. However, there are many other notations widely used in other translation projects. New Stranger Strings will not limit users who want to use this feature to only one notation but instead it will provide a fully configurable interface to fully customize this feature to their needs. An HTML tags check was implemented differently from the previous specification. It only detected HTML tags that were prohibited. And prohibited HTML were as well hard-coded. Similarly, as in the case with placeholders, this check will make selecting undesirable HTML tags configurable and will expand this check

with detecting missing or invalid tags. The First and Last character check was implemented quite well but still had one issue. It was detecting too many false alarms (6/10 false alarms for Last character check). The new version will try to solve this problem by reducing false alarms as much as possible.

Broadly speaking, some checks like Insensitiveness check or Style check are more useful for copywriters but other checks like Placeholder check or HTML tags check might be preferred by QA or translation managers. New Stranger Strings should allow users to personalize this tool to their specific needs. Additionally, every translation project will require a different setting. This means that the tool should be personalizable on multiple levels. Stranger Strings will distinguish three different levels of access:

- **Maintainer level** – represented usually by software engineer or IT-support etc.
- **Administrator level** – represented usually by project managers, localisation managers, platform team members etc.
- **User level** – can be anyone but usually translators, QA testers, front-end developers

Maintainer level will usually be assigned to only one person. This person will be responsible for creating the instance of Stranger Strings and will be able to choose who will have access to it as users or as administrators. The maintainer will also have responsibility for setting up data sources or loading custom spellchecking dictionaries. Administrators will be able to configure project-related settings such as modifying spellchecking dictionaries, white-listing allowed HTML tags or choosing reporting channels. And all users will be able to turn off checks they do not want or configure their importance.

3.2 UI Design

The shift in the purpose of Stranger Strings required a completely different approach to designing new UI. The first step was to change the landing page. Old one (see Figure 2.3 in Section 2.3) was rather dull with a ton of unused space and at the same time it contained too much unnecessary information. These issues are highlighted in Figure 3.3. I wanted to change it to something much more simple but also intrigue users about what this tool can offer.

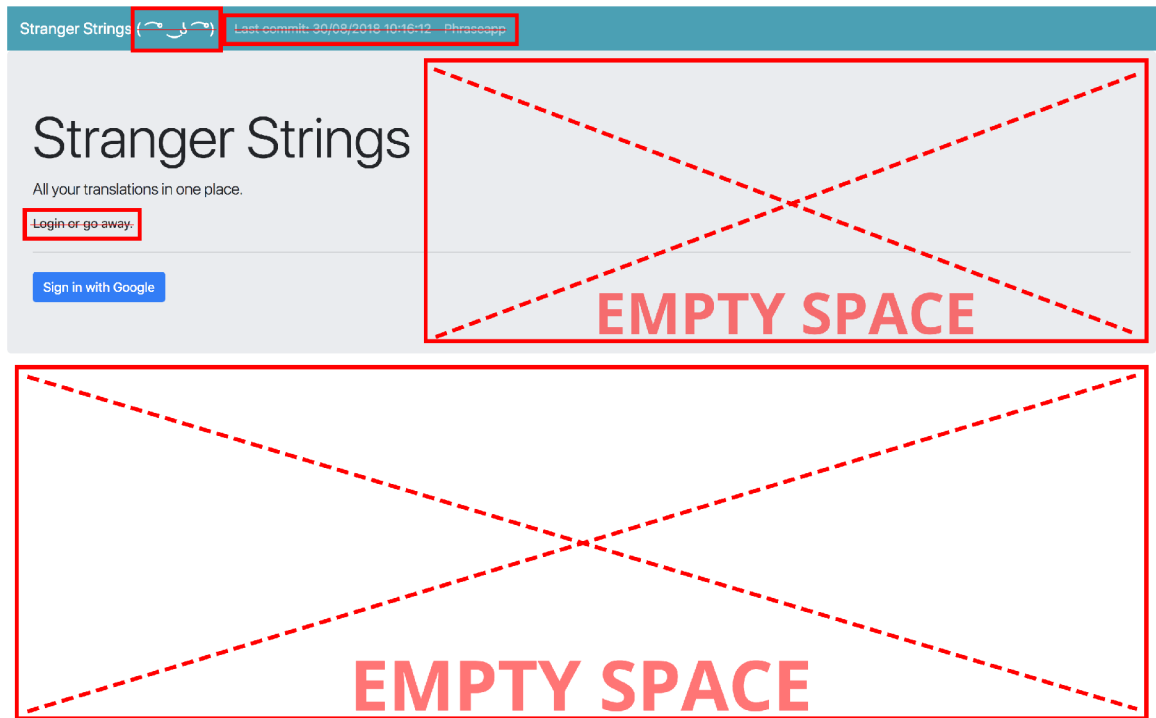


Figure 3.3: Screenshot with the landing page of old Stranger Strings with highlighted design issues. Most of the screen is unused and half of the displayed information is unnecessary.

Another important part is the view that I named Items view. This view shows after sign in to Stranger Strings and contains a list of translations keys with its attributes (e.g. progress, English translation, detected inconsistencies...). The previous version of the 'items' view provided an overview of all translations and allowed the user to search in them. It also showed some badges as indications that checks detected some issues in translation. At first sight, there was too much information and this can be confusing to users. Moreover, when I added more checks, the page became perplexing and its purpose unclear. These conclusions were not done by me personally, but instead, they were concluded by repeated user testing performed during the development process. More information about user testing can be found in Chapter 5.

As I was planning to add even more checks that problem would only get worse. Based on this information and consultation with my mentors I figured out that the table in the items view needs to be reorganized and the functionality of checks is not intuitive therefore needs to be explicitly explained on the Items view page. For this purpose, there were many created many mock-ups to see how the possible approaches might look like. One of them can be seen in Figure 3.4.

Keys	Translations	First	Last	Length	Spelling	Dynamic	Style	XML	Default translation
actions.add	██████████	3	1	1	1	1	2	2	Add _what_
common.button	██████████		1		5				Button
common.click_on_button	██████████	2	1		2	1			Click on Button
common.continue	██████████	2	1		3	1			Continue
common.error	██████████			1	2	2	3		Sorry, an error occurred.
company.guarantee	██████████	2	1	3	1	1	3		Worried about something? The ___companyName___ Guarantee has you covered.
help.not_good	██████████		3						Doesn't answer my question
personal.first_name	██████████	1	1		2	1			First name
personal.last_name	██████████	2			3	3			Last name
test.spelling	██████████	4	1	1	1	1			Very wrong spelling

Figure 3.4: One of the early mock-ups for the new Stranger Strings items page, showing the reorganized table. The main difference is showing each check-in single column instead of putting all check inside a single column as badges. This approach enables showing more checks more simply.

There are 2 possible approaches to solve this problem. The first one is to create a quick tutorial for new users consisting of several steps that explain different attributes on-page and their functionality. The other one is to insert a description to the page itself. Stranger Strings is supposed to be a lightweight simple tool so providing tutorials would not feel like a step in the right direction. Additionally, in my own experience, I usually skip those tutorials and want to figure out everything on my own. Therefore I decided to go with the other solution. The only challenge with the description solution is to figure out how to provide this information simply and without spamming users with more unnecessary information. The idea is to put this brief description inside the tool-tip alongside example (image or even interactive component).

After clicking on the translation key, a key detail view will show up. The main purpose of a key detail view is to show the content of all translations for the selected translation key along with additional information about inconsistencies and mistakes. Old Stranger Strings also displayed a lot of data that were company-specific and also contained features that were useless for the majority of users. The same problems as previously mentioned in items view were relevant for the key detail view. There was too much information that needed to be removed and at the same time, I needed to inform users about detected inconsistencies and mistakes. Figure 3.5 shows elements that should be removed and highlights important elements that need to be more noticeable. The solution I choose has similarities with the previous one. The idea is to highlight problems directly in translations and provide tool-tips that would further explain the problems. The key detail view would only display languages and translation content, which is the main purpose of this view. Everything else should be displayed through highlighting and with tool-tips.

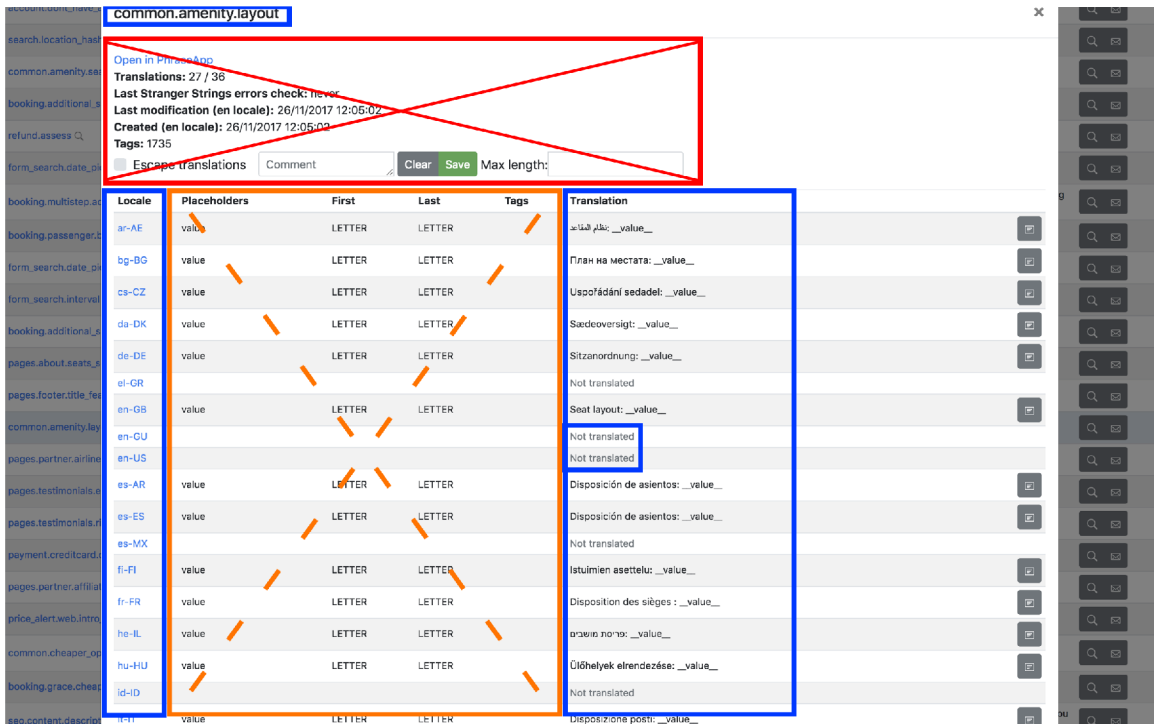


Figure 3.5: Screenshot with the key detail view of old Stranger Strings with highlighted design issues. The red square highlight elements that are unnecessary and should be removed. The orange square highlight secondary elements (checks) that should not take a lot of space. Blue square highlights important elements (translation key, translations, locales, missing translations) that should be visible at first sight.

At the beginning of creating the new version of Stranger Strings I was playing with an idea about the collection view with a collection detail view (as equivalent to items view with key detail view). The point of this view was to create a collection of keys with the same attributes and check consistency across these translations keys. But to fulfill the possibilities of this idea this project would require machine learning. Therefore when I found out during testing that users are not able to understand this feature and even cannot grasp the idea behind it when I explained to them, I decided to drop this feature.

The last part of the design is the configuration settings. As mentioned before Stranger Strings will be configurable on 3 different levels. Two of them (administrator and user levels) will have settings directly in the Stranger Strings app. I decided to go with two different setting pages one for users and another for administrators. To keep transparency in possibilities of Stranger Strings, everyone will be able to see admin settings but changes will be locked for them. This will be useful mainly when providing demo Stranger Strings to the public.

3.3 Technologies Used in Stranger Strings

This application is quite simple and making it easily adoptable by everyone is in the main requirements of this project. Therefore web platform was a clear choice. As this project requires some form of database, hosting, and remote server for lightweight computation, I

decided to learn how to work with *Google Firebase*⁴ platform which provides a solution for all of these requirements.

Firebase offers two types of databases: *Firebase Realtime Database*⁵ and *Firebase Cloud Firestore*⁶. Although it is recommended to use a more robust NoSQL *Cloud Firestore database* for the majority of new projects, I decided to go with *Realtime Database*. The reason was that *Realtime Database* is based on JSON and *Stranger Strings* would only need to visualize this JSON, which made me conclude that the implementation would be easier. Additionally, with expected use cases of *Stranger Strings*, this option will be eventually cheaper for the majority of *Stranger Strings* instances.

*Google Cloud Functions*⁷ will be used for server-side computation. As a runtime environment, I would like to use *Node.js*⁸ because that would allow reusing some parts of the code with front-end part which will use JavaScript as well. And not just that. This also introduces the possibility to set up front-end and back-end parts under a single package manager and share dependencies and overall configuration. This will not just improve the developer experience for future contributors but hopefully will simplify deployment and installation flow.

As acknowledged earlier, JavaScript will be used for the front-end part too. The old version of *Stranger Strings* used *Vue.js*⁹ as JavaScript framework. Since I decided to create new *Stranger Strings* from scratch I could choose any other JavaScript framework. However, I decided to stick with *Vue.js*. The main reason behind this choice is that *Vue.js* is known for being a very lightweight framework and lately is getting quite popular [2]. This is proven by many libraries created by community around *Vue.js*, including those that are used by the new version of *Stranger Strings*. For example *BootstrapVue*¹⁰ was chosen as design framework to provide *Stranger Strings* with uniform design and help keeping basic responsiveness. Another important library in the new *Stranger Strings* is *Vuefire*¹¹ that provides a binding between *Vue.js* app and *Firebase Database* (both *Realtime Database* and *Firestore*) which ensures data synchronization in real-time. *Vue Material Design Icons Components*¹² and *Vue Octicons*¹³ were used as libraries for icons and *Vue Country Flag*¹⁴ for flag icons. There are even more *Vue.js* exclusive community maintained libraries used in new *Stranger Strings*, but those will be described later in chapter 4 about Implementation.

*Jest*¹⁵ will be used as a unit testing framework across this project and will be crucial to keep this maintainable in future and might help provide potential contributors. Old *Stranger Strings* used different bundlers for back-end part (*Parcel*¹⁶) and front-end part (*Webpack*¹⁷). The reason behind this choice was due to some problems that occurred during the *Webpack* configuration for *Node.js*. However, this is not aligned with planned

⁴*Google Firebase* platform: <https://firebase.google.com/>

⁵*Firebase Realtime Database*: <https://firebase.google.com/products/realtime-database>

⁶*Firebase Cloud Firestore*: <https://firebase.google.com/products/firestore>

⁷*Google Cloud Function*: <https://firebase.google.com/docs/functions>

⁸*Node.js*: <https://nodejs.org/en/>

⁹*Vue.js* JavaScript Framework: <https://vuejs.org/>

¹⁰*BootstrapVue* framework: <https://bootstrap-vue.org/>

¹¹*Vuefire* library: <https://vuefire.vuejs.org/>

¹²*Vue Material Design Icons Components* library: <https://www.npmjs.com/package/vue-material-design-icons>

¹³*Vue Octicons* library: <https://www.npmjs.com/package/octicons-vue>

¹⁴*Vue Country Flag* library: <https://github.com/P3trur0/vue-country-flag/>

¹⁵JavaScript testing framework *Jest*: <https://jestjs.io/>

¹⁶Web application bundler *Parcel*: <https://parceljs.org/>

¹⁷Web application bundler *Webpack*: <https://webpack.js.org/>

unification between all parts of this project. According to (*Webpack*) documentation, it should be possible to have a single configuration for both parts. Therefore new Stranger Strings will tackle this problem and will only use (*Webpack*) across the whole project. To be able to write both front-end and back-end part in the newest ECMAScript standard, JavaScript compiler *Babel*¹⁸ will be used. This should be especially helpful for cross-browser compatibility and Google Cloud Functions limitations regarding supported *Node.js* versions. To improve developer experience, even more, the whole project will have configured its *ESLint*¹⁹ setup.

New Stranger Strings aims to be a proper open-source contribution-friendly project. For this purpose *GitHub*²⁰ will be used as a public repository. Repository will have integration with *CircleCI*²¹ pipeline to ensure all that all contributions are in line with code style rules and all unit tests are working.

3.4 Indications of Success

To consider project successful it would need to fulfill multiple expectations. Some of those are quite hard to track and evaluate but others can be objectively assessed when correctly specified.

The most important expectation of this project is its usefulness. This means that new Stranger Strings should be able to help users improve the localization quality of their websites or other similar products. It should be clear and simple enough to transfer all of that information to users in a straightforward way. It should prove its value to future users that might consist of QA testers, translators, translation managers, developers, project managers, etc... As new Stranger Strings will be open-source projects accessible for anyone to create their instance it would be hard to keep track of all users using this project. Another thing to take into consideration is the fact that translation content might be considered very sensitive for some users. Therefore I decided not to put any kind of tracking into this project. However, I have the possibility to set up Stranger Strings in a Czech company *Kiwi.com*. Their product is oriented to the travel industry therefore, it is localized to 35+ different languages. For this specific instance of Stranger Strings, I will set up *Google Analytics*²² to track usability data. It is important to mention that I will not take any steps to enforce or even push people to use this tool. The reason is that enforcing this tool would introduce bias to usage stats and users will not use this tool because they want to, but because they have to. Those measures should help to only see users that use Stranger Strings because they see the value in it a helps them do their job better.

To put those expectations into the perspective of measurable metrics, the aim for weekly users in the tracked instance of Stranger Strings is 10 different users/week. About 20 users will visit this tool once in a month. If there will be a possibility to check the type of users based on their position in the company, the expectation is that there will be a representation of at least 3 different positions across the company (e.g. developers, QA testers, product managers or translation team members). Additionally, I will integrate Stranger Strings with the company *Slack* channel dedicated to reporting translation bugs. In the end I will provide statistics about how many translation bugs were reported via Stranger Strings to

¹⁸JavaScript compiler *Babel*: <https://babeljs.io/>

¹⁹JavaScript linting utility *ESLint*: <https://eslint.org/>

²⁰*GitHub*: <https://github.com/>

²¹*CircleCI*: <https://circleci.com/>

²²*Google Analytics*: <https://marketingplatform.google.com/about/analytics/>

this *Slack* channel. Success would be considered when there will be more than 20 unique bugs reported with Stranger Strings by at least 5 different users.

Another expectation from new Stranger Strings is its code quality, easy adaptability, and code-base setup for good developer experience. Evaluation of these aspects would go above the scope of this bachelor thesis because it requires external propagation and building community around open-source projects takes years.

Chapter 4

Implementation

From the perspective of used technologies, there is not too much difference between old and new Stranger Strings. The composition of both technological stacks might be even the same apart from dependencies and libraries. However, that is not the case for implementation. Both front-end and back-end parts were re-implemented from scratch. The main reasons for not reusing code from old Stranger Strings were a shift in the purpose of new Stranger Strings and the fact that old Stranger Strings was basically in-house software indented for internal use only. The problem of turning in-house software into open-source software was not just about the removal of credentials and tokens. Rather it was about changing the way of thinking and solving different challenges that popup along with many possible edge cases that future users might require. In other words, the implementation in old Stranger Strings was not scalable for the needs of the open-source project with a different purpose.

The previous chapter mentioned changes in the architecture (Chapter 3) that consisted of separating parts of applications into individual modules and taking into consideration that each user might prefer different configurations. The same applied to the implementation. Meaning that whenever there was a possibility to introduce configurable variables instead of constant value Stranger Strings should go with a configurable variable. Stranger Strings is meant to be an open-source application that anyone can set up for their specific needs. From the implementation point of view, it would be much easier to provide Stranger Strings as a web service where users just log in to the application and choose the data source of their translations. Unfortunately, many companies consider their localization data as sensitive and would never provide access to third parties. Hence the first challenge for new Stranger Strings was to create user-friendly setup experience that would be close to website service experience and would avoid discouraging future possible users. All of this is needed to be done without compromises on security.

Few things deny automation of the installation process into a single step/command and are needed to be done manually. First of all, the project is heavily dependent on Google Firebase and each instance of Stranger Strings requires its own Firebase project which requires ownership of Firebase account. Firebase offers 2 pricing plans¹ Spark plan (free) & Blaze plan (billed monthly based on data usage)². Spark plan which is free does not allow API calls outside of other Google services³. New Stranger Strings depends on outbound

¹At the time of implementation there was a 3rd price plan called Flame plan that is not available since January 2020. More info: <https://firebase.google.com/support/faq#flame-plan-legacy>

²Firebase pricing plans: <https://firebase.google.com/pricing>

³Docs mentioning outbound API calls restrictions: <https://firebase.google.com/docs/functions/get-started>

API calls because its data source modules are basically interfaces between different servers containing translation data. That means that the project requires the monthly billed Blaze plan, which might be discouraging for some users. However, the good news is that Stranger Strings is a very lightweight application and is also very good optimized, which means that costs of running are usually below minimum billed limits so the application has zero or close to zero costs for a majority of users ⁴. Another thing that needs to be set up manually is translation data-source. Currently, there are 2 data-source modules available:

- **Phrase data-source module** gets translation data from specified translation project through Phrase API
- **GitHub data-source module** gets translation data from *GitHub* repository containing JSON files in form of keys and values. The repository needs to have certain structure which can be seen in Figure 4.1. Supported are both flat (Figure 4.2) and nested (Figure 4.3) JSON formats.

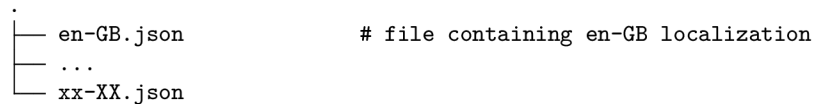


Figure 4.1: The required directory tree structure for repositories containing translation for Stranger Strings. The name of JSON files must be a combination of an ISO 639 two-letter lowercase code associated with a language and an ISO 3166 two-letter uppercase code associated with a country or region separated by a dash.

```
{
  ...
  "translation.key": "Translation content 1.",
  "translation.keyTwo": "Translation content 2.",
  "another.translation.key": "Another translation content.",
  ...
}
```

Figure 4.2: Supported flat JSON format of translations in xx-XX.json file.

⁴The billing for Stranger Strings instance with 6000 translation keys and with 140 database updates per month cost 0.15 \$


```

{
  ...
  "translation": {
    "key": "Translation content 1.",
    "keyTwo": "Translation content 2."
  },
  "another": {
    "translation": {
      "key": "Another translation content."
    }
  },
  ...
}

```

Figure 4.3: Supported nested JSON format of translations in xx-XX.json file.

Localization data-sources are usually based on servers that require some form of authentication to access their data. To be specific in case of *GitHub*, it is username and password (or generated token with specified access restrictions)⁵. For Phrase API it is project ID and access token⁶. An access token is highly sensitive data and security is very important for Stranger Strings. Stranger Strings stores all token as environment variables inside */.env* file. This ensures that tokens are always separated from code. Therefore it is expected that during installation each user create their own */.env* file. The template for creating */.env* file can be found in */.env.example* (Figure 4.4).

⁵GitHub API documentation regarding authentication: <https://developer.github.com/v3/auth/>

⁶Phrase API documentation regarding authentication: <https://developers.phrase.com/api/#authentication>

```

#####
##### DATA SOURCE #####
#####

### note: must uncomment one data source to use it

##### \textit{GitHub} #####
#VUE_APP_GITHUB_USER=""
#VUE_APP_GITHUB_PASSWORD=""
#VUE_APP_GITHUB_REPO=""

##### PhraseApp #####
#VUE_APP_PHRASEAPP_PROJECT_ID=""
#VUE_APP_PHRASEAPP_TOKEN=""

#####
##### FIREBASE PROJECT #####
#####

VUE_APP_FIREBASE_MESSAGING_SENDER_ID=""
VUE_APP_FIREBASE_API_KEY=""
VUE_APP_FIREBASE_AUTH_DOMAIN=""
VUE_APP_FIREBASE_DATABASE_URL=""
VUE_APP_FIREBASE_PROJECT_ID=""
VUE_APP_FIREBASE_STORAGE_BUCKET=""
VUE_APP_FIREBASE_APP_ID=""

```

Figure 4.4: Content of `./env.example` file which contains template for user's `.env` file. User needs to complete missing authentication tokens for Firebase project along with the authentication tokens for either *GitHub* or *Phrase* data-source.

Based on those restrictions and requirement it is supposed that the person responsible for creating an instance of Stranger Strings will have advanced computer skills and will be considered the maintainer for Stranger Strings. In the installation, process maintainer will also be able to restrict access rights to a specific email address or email domain. There is also a possibility to choose users with administrator right by their emails or choose sign in methods. Currently, Stranger Strings support 3 different sign in methods: Google signs in, sign in by email link, and anonymous access (access for everyone without restrictions). These configurations can be made by adjusting the `/common/config.js` file before each deployment.

4.1 Open-Source Project Overview & Installation

Project Stranger Strings can be found on *GitHub*⁷ under MIT licence. Everyone with *GitHub* account is welcomed to contribute to this project via Pull Requests.

When requirements for setting up Stranger Strings instance are met (user has *Firebase* project with *Realtime* database on *Firebase Blaze* account and owns *Phrase* project or has *GitHub* repository with translation data in the correct format), then after downloading or cloning repository can proceed to installation. The installation process consists of the following steps.

⁷Stranger Strings repository on *GitHub*: <https://github.com/kiwicom/stranger-strings>

1. **Configure .env file** with authentication tokens for Firebase along with authentication tokens for either *GitHub* or Phrase.

2. **Install project dependencies** with the following command:

```
$ yarn install --all && yarn --cwd ./functions
```

3. **Login to Firebase and select a project** where you want to host your Stranger Strings instance:

```
$ firebase login  
$ firebase use <your_Firebase_project_ID>
```

4. **(Optional) Adjust access rights and sign-in method in /common/config.js.** The file also contains some other default configurations (f.e. there is a possibility to enable spellchecking for various languages⁸)

5. **Deploy:**

```
$ yarn deploy
```

6. **Enable sign-in providers** accordingly to configuration constant *SIGN_IN_METHOD* in */common/config.js* in Firebase Console (default is Anonymous sing-in provider)

For more detailed installation description with images see README on Stranger Strings's *GitHub*⁹. Once the application is deployed, the front-end part can be run locally as well. This can be done with the following command:

```
$ yarn start
```

This might be especially handy for development or debugging purposes. Running back-end modules locally is a little bit more tricky. The whole back-end part is located under */functions* folder. The files in */functions* can be run under *Node.js* environment. Most files there also contain conditions that recognize that file is run locally and then adjust setup regarding environment variables for easier debugging.

To avoid bugs or unintended behavior in code-base, Stranger Strings project uses Jest¹⁰ as a unit testing framework. Although this project does not have very high test coverage, the crucial and error-prone parts of code should be covered. To run all of the unit tests use the command:

⁸By default spellchecking is allowed for en-GB, sk-SK, and cs-CZ. The reason is that many dictionaries used in other languages have poor quality and cause many false alerts and I was only able to verify those 3 languages properly. There is also a possibility to use custom dictionaries, more info at <https://github.com/kiwicom/stranger-strings/tree/master/functions/dicts>

⁹Stranger Strings' *GitHub* README with detailed installation description: <https://github.com/kiwicom/stranger-strings#stranger-strings>

¹⁰More info about JavaScript testing framework Jest can be found here: <https://jestjs.io/>

```
$ yarn test
```

For checking bugs that are caused by syntax errors, Stranger Strings uses *ESLint*¹¹. JavaScript, being a dynamic and loosely-typed language, is especially prone to developer error. Without the benefit of a compilation process, JavaScript code is typically executed to find syntax or other errors. Linting tools like *ESLint* allow developers to discover problems with their JavaScript code without executing it [12]. Many modern code editors have *ESLint* plugins available for real-time code linting or automatic fixes. This hugely improves the developer experience. Another big feature of *ESLint* is that apart from static syntax check it can also enforce certain code style across code base and even allow creation of custom rules¹². Stranger Strings does not have any custom *ESLint* rules but has a configuration that will enforce a specific code style. To run *ESLint* static check use command:

```
$ yarn lint
```

Stranger Strings repository uses CI/CD from *CircleCI*¹³ to make sure that repository stays free from bugs which can be detected by static analysis or by unit tests. *CircleCI* also runs *ESLint* to ensure that the same code-style is kept in the git master branch across the codebase. The complete configuration of *CircleCI* can be seen in Figure 4.5.

```
version: 2
jobs:
  test:
    docker:
      - image: circleci/node:10.11.0
    working_directory: ~/repo
    steps:
      - checkout
      - run: yarn
      - run: yarn lint
      - run: yarn test
      - run: yarn build
workflows:
  version: 2
  master:
    jobs:
      - test
```

Figure 4.5: Content of `/.circleci/config.yml` file that contains configuration for *CircleCI* pipeline. To allow the merging of some pull requests into the master branch, each pull request must first pass few commands without errors. Firstly pipeline installs all dependencies and then runs *ESLint* check. Then trigger, all unit tests and finally tries to build the whole project. When everything passes successfully without errors PR can be merged into the master branch.

¹¹More info about JavaScript linting utility *ESLint*: <https://eslint.org/>

¹²More info about creation of custom *ESLint* rules: <https://eslint.org/docs/developer-guide/working-with-rules>

¹³More info about *CircleCI*: <https://circleci.com/>

4.2 Inconsistencies Computation

It is not unusual that localization projects contain hundreds or even thousands of translation keys and tens of different languages. Doing operations big data-sets takes a significant amount of time and memory usage. Especially when translations are in the form of strings and the majority of operations are some form of regex search. The only logical choice was to do all computations on translations only when necessary and on the server-side. There are only two scenarios where computation is necessary. First is when the content of data-source is changed and the second is when some adjustable variable that is used for computations is changed.

Therefore all computations that Stranger Strings needs are done on Google Cloud Functions that serve as the back-end for this project. On deployment, *functions/index.js* is uploaded on Firebase. In this file, it can be seen that Stranger Strings has 3 different functions that provide computations:

- **update**
- **inconsistenciesUpdate**
- **collectionsUpdate**

When **update** function is triggered, firstly it checks whenever there is some change on data-source (for *GitHub* data-source it compares commit ID (also known as SHA) of the last commit and for *Phrase* it compares last update date). If no change in data-source is detected the function ends. Otherwise the function will continue with downloading whole content from data-source.

Each data-source module takes care of data downloading independently and the duration can vary as there might be API limitations. In the case of the *Phrase* data-source module there were limitations regarding the maximum number of concurrent (parallel) requests and the requests for translations were paginated¹⁴. This meant that per request only 100 translation can be obtained from the server and only 4 requests¹⁵ can be served at once. To be able to download all the data at once, Stranger Strings uses *bluebird*¹⁶ library for concurrency coordination. The full implementation of the *Phrase* data-source module is located in */functions/loaders/PhraseappLoader.js*.

After downloading translations Stranger Strings performs all of the computations. Finally after computations, it uploads all the translations along with its computed metadata to Stranger Strings database. Similarly old Stranger Strings used to update as the only function, but after introducing checks feature there was a need to recompute everything without downloading data from data-source. And that is exactly what **inconsistenciesUpdate** does. It takes data that is already in Stranger Strings database and recompute inconsistencies or errors of every translation and translation key and then update the metadata in the database. For example, this is helpful when there is an update of the dictionary for spellchecking, there is a change in settings of computation sensitivity for some checks. Similar to *inconsistenciesUpdate* works the **collectionsUpdate** but instead of doing computations for translations and translations keys it computes inconsistencies across a collection of multiple translation keys. Unfortunately this feature called collection was later hidden

¹⁴Phrase API documentation: <https://developers.phrase.com/api/#intro>

¹⁵At the time of implementation the limit for the concurrent request was just 2 requests at once.

¹⁶More info about bluebird library: <http://bluebirdjs.com/>

on front-end because it was not fulfilling its potential and was just confusing users. Anyway it is still persistent on the back-end side and can be easily revisited later.

All of the 3 functions are using HTTP triggers¹⁷. This means that each function can be invoked with an HTTP request. Each function has its HTTP endpoint which is generated during deployment. From the caller's perspective, HTTP invocations are synchronous, meaning that the result of the function execution will be returned in the response to the HTTP request. However none of those 3 functions use the response to send data back to the caller. In this case HTTP endpoints are only supposed to be used for invocations. Because there is no point passing information about running or finished function to the only caller when multiple users are using Stranger Strings at the same time. Instead the information about running function is directly uploaded to the database so every user can see that some update is running in real-time. Another advantage of chosen approach is that it serves as a semaphore to avoid running multiple functions at once. Because each of these functions is first deleting the data and then updating them. Upon invocation each function checks the database whenever there is another running function and if yes the function ends. The implementation of this semaphore can be found in `/functions/dbMutex.js`, thanks to this the database always remains stable.

As previously mentioned in Section 3.1 Stranger Strings features 9 different checks. Some of them are checking whenever there is a problem in some translation or whenever there are inconsistencies between translations under one specific translation key. And some of those checks are checking both at the same time. Implementations of each check consist of two steps. In the first step, there are computed metadata¹⁸ for each translation under single translation key. In the second step, each translation key is evaluated based on metadata of its translation, whenever it contains some inconsistencies or if there are translations with some problems. To see implementation check file `functions/dbUpdates.js`.

4.2.1 Placeholders Check

Placeholders check is a feature that detects errors regarding placeholders. This check is especially valuable for a bigger localization project. Usually, there are some values (f.e. numbers, dates, currencies) that are appearing multiple times across many pages where they are part of different contexts and translations keys. It is a good practice to keep them represented under some variables. Particularly when those values are prone to change over time. In localization those variables are usually represented by placeholders. Unfortunately there are multiple notations used to mark placeholders in translations. Many companies use different localization services or providers which use their own custom notation. However they always follow patterns that can be parsed by some localization tool with some form of a regular expression. Placeholder check relies on this. Stranger Strings expects that its maintainer or administrator will enter placeholder pattern via regex.

Once regex for placeholder pattern is known it will be then used in all computations to detect placeholders in translations. A list of placeholders is stored in the database as translation metadata in the form of an array. In the second step of computation the translations under each translation key are compared whenever all of them contain the same

¹⁷Documentation of Google Cloud Function HTTP triggers: <https://cloud.google.com/functions/docs/calling/http?hl=en>

¹⁸Metadata of translations contains information such as present placeholders, present HTML tags, type of starting/ending character, letters that are unknown for spellchecking, length of translation, etc.

placeholders. The result is then stored in the form of Boolean value inside translation key metadata.

	bg-BG		Плщане на price_
	cs-CZ		Zaplatit price_
	da-DK		Placeholder error
	de-DE		bezahlen
	el-GR		Πληρώστε
	en-GB		Pay price_
	es-AR		Pagar price_
	es-ES		Pagar price_
	es-MX		Pagar
	fi-FI		Maksa price_

Figure 4.6: Demonstration of placeholders checks functionality. It can be seen that placeholders are not consistent for all translation under one translation key (placeholders are missing in translations for el-GR and es-MX locales).

The idea about detecting inconsistencies in placeholders was inspired by reoccurring translation bug reports caused by placeholders. This check served as inspiration for the future of Stranger Strings that new Stranger Strings follows. In old Stranger Strings, this was the first implemented check, still, it had some limitations. As expected from the in-house tool it was hardcoded to detect certain placeholder patterns. There were even some problems that went under the radar of this check. It was detecting placeholders correctly but the consistency comparison of placeholders under translation key was only based on whenever each translation has the same number of placeholders. In reality, this meant that when two translation under the same translation key had two different placeholders, the check did not consider it as inconsistent.



Figure 4.7: Example of placeholder inconsistency that old Stranger Strings was not able to detect. This kind of mistake is quite normal for a big translation project with placeholders. It is usually caused when translators are not completely understanding the concept of placeholders.

4.2.2 HTML Tags Check

It is not rare to spot messed up HTML tags as part of some content on websites. It is obvious when someone looks at it on the website, but that not always the case from the content management system's point of view. Usually, it is detected by QA testers or end-users and it can be seen on many popular websites. When wondering how to detect broken HTML tags in websites the first thing that usually comes in mind is to use machine learning. But it can be done much more easily by comparing translation that should have the same content. And that is exactly what this check does. The principle is the same as for Placeholders check. In the first step, it computes HTML tags via regex for all translations. Then in the second step, it compares a list of HTML tags for all translations under the same translation key. If there is at least one translation with correctly used HTML tags under some translation key than Stranger Strings will be able to detect whenever there is some broken, missing, or excess HTML tag under this translation key.

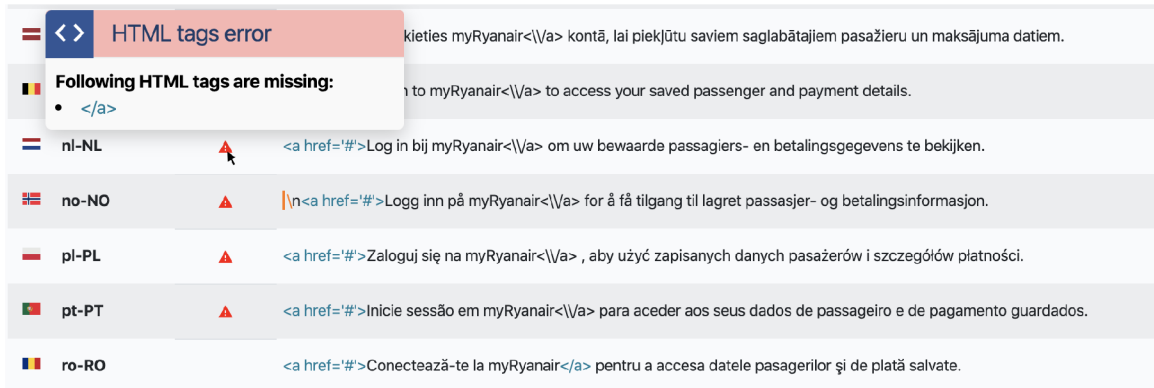


Figure 4.8: Demonstration of ability to detect broken HTML tags in translations. The displayed translation is taken from a production website belonging to one of the biggest low-cost airlines in the world.

When we look closer to what is the root cause of broken HTML tags, we realize that it is a result of mixing scopes of front-end development and localization. Knowledge of HTML is out of the scope of localization and usually it is not required from translators to know HTML. Ideally HTML tags should not be part of localization and should be avoided if possible. In practice, not including HTML tags into localization is usually impossible or would bring unnecessary complications. Even when it is decided to use HTML tags in localization it should be used consistently. For example when styling text to bold is part of localization it should be done with the same HTML tag across whole localization project (e.g. use only `` and than never use ``). And these recommendations are represented in another feature this check offers. By default HTML tags checks only allow usage of few HTML tags¹⁹ and the rest is considered as disallowed. Allowed HTML tags can be adjusted in the UI of Stranger Strings by users with administrator rights.

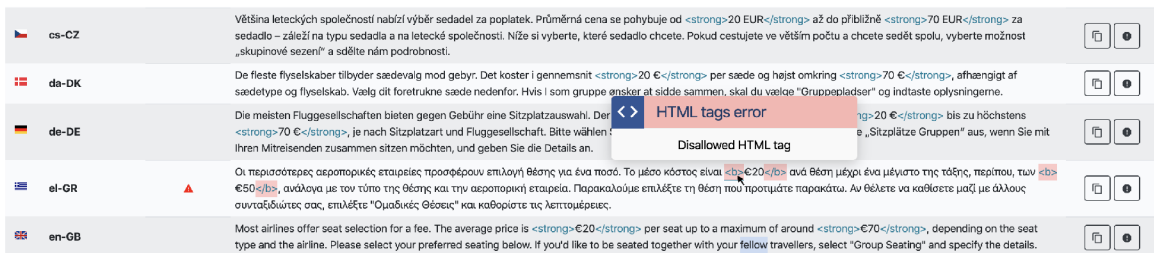


Figure 4.9: It is not a good practice to use multiple HTML tags for the same purpose across the localization project. Stranger Strings offers a solution that helps to enforce consistent usage of HTML tags across the project.

The old version of Stranger Strings featured similar check but it only warned about the usage of certain hard-coded HTML tags. So it was not detecting any broken, missing, or excess HTML tags.

¹⁹Currently by default only allowed HTML tags are `
`, `<a>`, ``, ``, `` and `<i>` (together with their closing equivalents)

4.2.3 Values Check

Subsection 4.2.1 about Placeholder check already described the value of placeholders and why it is a good practice to use them over values that are likely to change over time. Since Stranger Strings is supposed to be a tool that helps to improve the quality of localization, it makes sense to warn or even enforce good practices. And that is exactly what Values check does. It detects dynamic values in translations and recommends replacing them with placeholders. Basically, all occurrences of numbers can be considered as dynamic values. Even when the numbers are considered to be never changing, this approach can save money and time that eventual change might require.

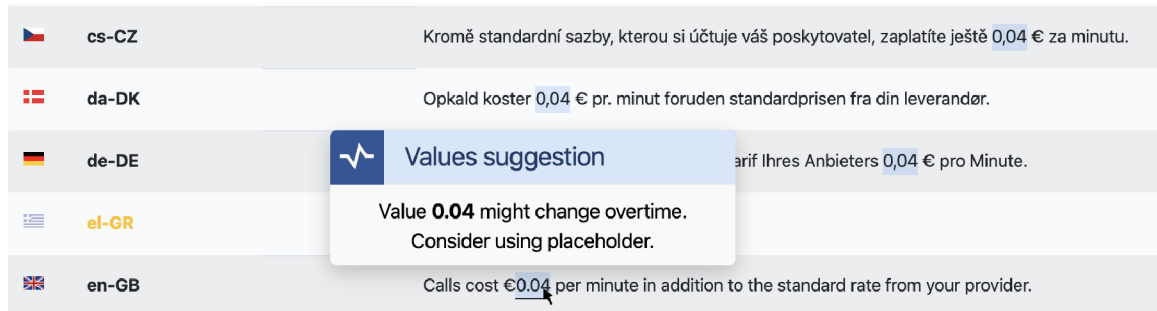







Figure 4.10: Demonstration of values check. This check detects dynamic values (numbers) and recommends usage of placeholder. For this particular case, it is very likely that some business decisions will affect this value in the future. This would mean changes in multiple translations which can cause some other mistakes, takes time, and usually cost money.

Values check was introduced in the new version of Stranger Strings. In the future, this check can be improved by detecting another dynamic value apart from numbers (e.g. currencies, names, URLs, etc.). Upon consultation of this feature with translation manager from Kiwi.com, I realized that values check might not be ideal for everyone as the changing of values without noticing translators might lead to wrong declensions of words as many languages are very sensitive for just minor changes in numbers. However, there was a mutual agreement that this check is still valuable as a suggestion rather than an error.

4.2.4 First and Last Character Check

The first character check and Last character check are two different checks, however, they have similar functionality. It can be apparent from the name, that these features check the consistency of character in the beginning and end of each translation under the same translation key.








	de-DE	+ __price__
	el-GR	Not translated
	en-GB	
	es-AR	Por __price__
	es-ES	+ __price__

< First character warning

First character is **letter** but expected **plus**

Figure 4.11: Example of inconsistency detected by First character check. The translator responsible for es-AR translation decided to use the word „plus“ instead of plus sign character „+“. As plus sign character was used in all of the other languages, it was probably an intentional copywriter’s choice. In some edge cases this kind of inconsistency can result in distorted UI.

The implementation is quite simple. In the first step, the first and last characters from each string are evaluated and categorized into groups such as letters, colons, exclamation marks etc.. To see how the categorization works see the *determineCharType* function inside */functions/utills.js* file. In the second step, the characters are compared while multiple exceptions are applied.

	ar-AE	
	bg-BG	Има
	cs-CZ	Podívejte se na výhodné letenky
	da-DK	Vi har nogle fantastiske tilbud til dig!
	de-DE	Wir haben ein paar tolle Angebote für Sie!
	el-GR	Not translated
	en-GB	Have we got some deals for you!

> Last character warning

Last character is **letter** but expected **exclamation mark**

Figure 4.12: Example of Last character inconsistency. Translation for cs-CZ was the only one missing exclamation mark. Most likely the exclamation mark has been simply forgotten by the translator responsible for cs-CZ language.

Those two checks were also part of the initial 4 checks in old Stranger Strings. But in most cases they were reporting false alarms. Those alarms were caused by syntax differences between languages, which caused uncertainty whenever those checks are valuable. After experimenting with real translation data and adding multiple exceptions to the evaluation the false alarms were drastically reduced. For instance Thai language does not have sentence-ending punctuation [9] or Japan language does not need question mark [3]. Another repeating pattern was falsely reported inconsistency caused by brackets or digits that resulted from syntax differences across languages. Including all of these exceptions inside Last and First character checks made them reliable for most of the cases.

4.2.5 Length Check

Length check is a feature that detects an inconsistency in the length of the translation. Big length differences can mean that the context of translations can slightly differ or even have a completely different meaning. Translating requires ingenuity as you cannot always translate something word to word. Sometimes there are no equivalent words and translators must use several different words to describe one specific thing. The skilled translators can usually cope with this and translate it without using much more character than the original text. Using too much character compared to the original text can result in unintended behavior and a broken UI. Another common situation that happens is that some translation is changed and the rest of the translation under the same translation key is not. The length check is able to detect both of these problems.



Language	Warning	Text
bg	Length warning	денни авиокомпания предлагат възможността да си поръчате предварително храна за вашия полет. Таксата зависи от избраните ястия. За да можем
	This translation looks suspiciously long	яската ви, се нуждаем от допълнителна информация: Моля, посочете своите изисквания към храната, като например дали сте вегетарианец или
cs-CZ	⚠	Většina klasických přepravních leteckých společností nabídne možnost pro let předobjednat občerstvení. Výše poplatku závisí na zvoleném občerstvení. Abychom mohli požadavek dokončit, potřebujeme několik dalších informací: uveďte všechny své stravovací požadavky, například žádost o vegetariánskou nebo celiakální stravu.
da-DK	⚠	De fleste Legacy Carrier-flyselskaber tilbyder mulighed for at forudbestille måltider til flyturen. Gebyret afhænger af det valgte måltid. Vi skal bruge yderligere oplysninger for at kunne gennemføre din anmodning: Oplys os venligst om særlige kostbehov, såsom vegetarisk eller celiaki.
de-DE	⚠	Ältere Fluggesellschaften bieten eventuell die Möglichkeit, Mahlzeit für Ihren Flug vorher zu bestellen. Die Gebühr hängt von der ausgewählten Mahlzeit ab. Damit wir Ihre Anfrage bearbeiten können, benötigen wir einige Zusatzinformationen: Bitte lassen Sie uns wissen, welche Ernährungsbedürfnisse Sie haben, z. B. ob Sie Vegetarier sind oder Zöliakier.
el-GR	Not translated	
en-GB		Once you select your preferred meal, we will contact the airline and get back to you. Some low-cost airlines may not provide specific meals.

Figure 4.13: Example of big translation length difference detected by Length check. The English translation is clearly smaller and less wordy compared to others. Upon further investigation we can see that even when the context might be similar the content of translation is different. The translation in Czech can be translated back to English following: „The Majority of traditional airlines offer an option to pre-order snacks for your flight. The price depends on snack selection. To be able to complete your order, we need a few additional information. Please write down all your dietary requirements, like a vegetarian on a gluten-free diet.“

The implementation is quite simple. This check just compares the expansion ration between the English translation and any other translation and determines whenever it is higher than allowed. The function that calculates the maximum allowed expansion ratio can be found in `/common/maxExpansionRatio.js`. This function is inspired by data provided by IBM about additional space required for translations in relation to number of characters in text [8] (see Table 4.1). Using cubic spline interpolation, I created a function that approximates this data. Additionally after experimentation with real data, this function was optimized to ignore texts that consist of less than 3 characters and further adjusted sensitivity to allow even more expansion.

Number of characters in text	Additional space required	Additional expansion allowed by Length check
<3	100 to 200%	-
3 to 10	100 to 200%	1360 to 180.6%
11 to 20	80 to 100%	175.5 to 148.4%
21 to 30	60 to 80%	146.4 to 133.1%
31 to 50	40 to 60%	132 to 116.2%
51 to 70	31 to 40%	115.6 to 106.1%
71 to 500	30%	105.6 to 58.5%
501 to 1000	30%	58.4 to 45.3%
1001 to 5000	30%	45.3 to 20.4%

Table 4.1: Table showing translation length expansion based on character length of U.S. English translation. Column „Additional space required“ is maximum expansion estimation provided by IBM Knowledge Center [8]. Column „Additional expansion allowed by Length check,“ tells how much expansion is allowed by the Length check.

The final function for estimating maximum character length expansion ration in relation to character length is the following:

$$f(x) = \frac{14.1}{1 + \left(\frac{x-3}{0.002}\right)^{0.2}} + 0.5 \quad (4.1)$$

4.2.6 Spelling Check

Spelling check is one of the most complex features of Stranger Strings. One of the most famous open-source spellcheckers is Hunspell²⁰ created by László Németh. It is used by LibreOffice, Mozilla Firefox, Google Chrome, or even can be found on operating systems like macOS or Linux distributions. If Hunspell is provided with right dictionaries it can do spellchecking for any language. But Hunspell is just a command-line tool implemented in C++/C which is not suitable for this project. Luckily Titus Wormer created Hunspell compatible spell-checker in plain-vanilla JavaScript called nspell²¹. Author of nspell also maintains repository²² that groups 90 open-source dictionaries accessible as npm packages. Unfortunately some of those dictionaries do not have the quality to provide reliable spellchecking as this would result in marking too many valid words as unrecognized by the spellchecker. Spelling check uses nspell along with those dictionaries, but by default it only checks en-GB, sk-SK, and cs-CZ. This can be changed before deployment in ‘./common/config.js’. If someone owns access to better dictionaries they can update them inside ./functions/dicts/ folder²³. Stranger Strings always prioritizes custom dictionaries over default ones.

Before each spellchecking the translation is stripped from HTML tags and placeholders based on provided placeholder regex. Spelling check also counts with the possibility that users often use product-related terms that are not part of dictionaries. Therefore after each

²⁰More information about Hunspell: <http://hunspell.github.io/>

²¹More info about nspell: <https://github.com/woorm/nspell>

²²Repository that groups 90 dictionaries in one place can be found here: <https://github.com/woorm/dictionaries>

²³To see more information about how to incorporate custom dictionaries in Stranger Strings for spellchecking see [./functions/dicts/README.md](#) or visit following link: <https://github.com/kiwicom/stranger-strings/tree/a577be404a71215f3ccfcf55aead903bcc6cee62/functions/dicts#custom-dictionaries>

spellchecking the unrecognized words are compared to those that are in database marked by the user as white-listed words. This means that in each instance of Stranger Strings users can build their dictionary expansion for any language with custom words based on their needs.

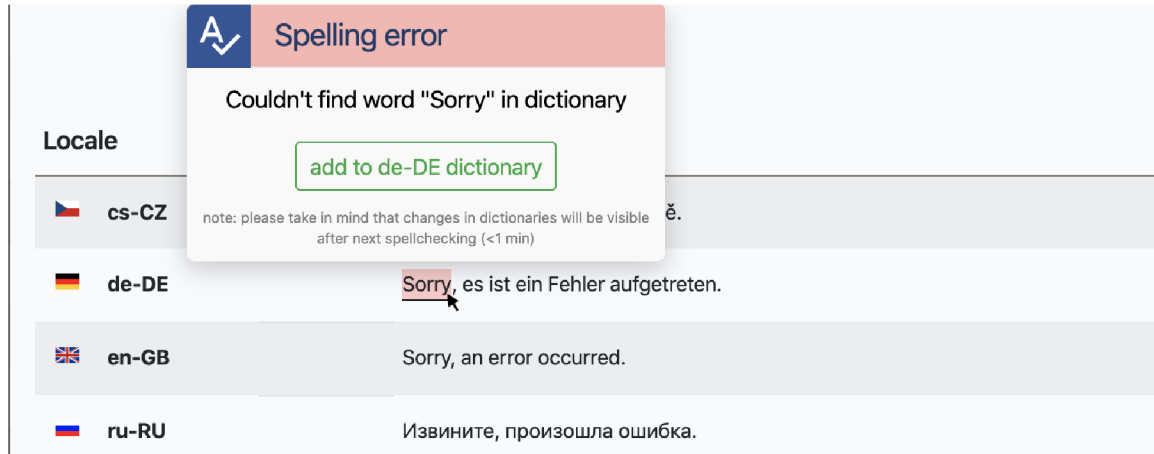


Figure 4.14: Apart from detecting possible typos Spelling check also allows users to expand used dictionary with used terms.

4.2.7 Style Check

Style check is inspired by extension called *write-good linter* used in multiple code editors like Atom²⁴ or Visual Studio Code²⁵. As *write-good* is also available as open-source npm package²⁶, Stranger Strings uses it to check correct writing style. However this check is only available for English language and partially for German.

Style check helps to avoid usage of passive voice, lexical illusions, „so“ at the beginning of the sentence, „there is“ or „there are“ at the beginning of the sentence, „weasel words“, common cliches, „to-be“ verbs, wordy phrases and unnecessary words and lastly it helps to avoid adverbs that can weaken meaning (e.g. really, very, extremely...). Style check also uses the German extension for *write-good* called *schreib-gut*²⁷. This German extension is only able to detect wordy phrases and unnecessary words along with „weasel words“.

²⁴Write-good linter for Atom: <https://atom.io/packages/linter-write-good>

²⁵Write-good linter for Visual Studio Code: <https://marketplace.visualstudio.com/items?itemName=travisthechie.write-good-linter>

²⁶More info about write-good: <https://github.com/btford/write-good>

²⁷More info about *schreib-gut*: <https://github.com/timkam/schreib-gut>

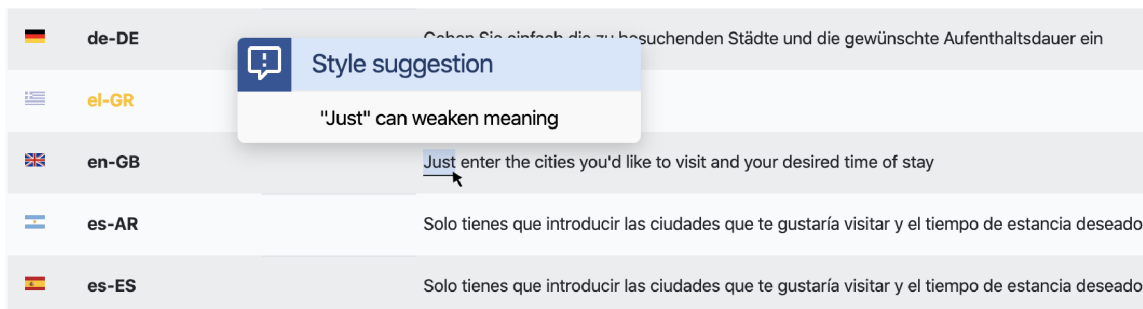


Figure 4.15: Example of style suggestion provided by style check. The check might be more useful for copywriters to help them in more user-friendly and simpler writing.

All of the checks provided by *write-good* can be turned off. Before each computation, style check uses *write-good* based on settings in the database so the computation is always according to user options. Style check is not suitable for everybody, but still might be helpful for some users.

4.2.8 Insensitiveness Check

Functionality of insensitiveness check is similar to style check, but instead of *write-good* it is based on library called *alex*²⁸. In this case only the English language is supported. Insensitiveness check can find gender favoring, polarizing, race-related, religion inconsiderate, or other texts containing unequal phrasing.

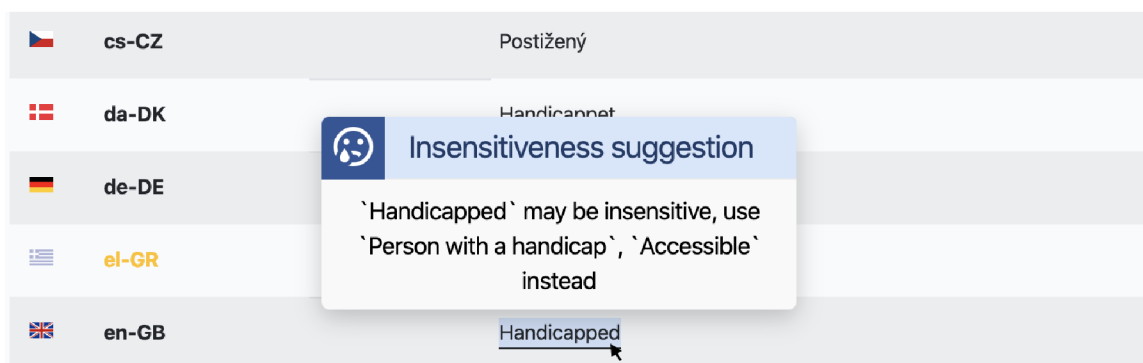


Figure 4.16: Insensitiveness check not only detects words that can be considered as insensitive but also suggests alternatives that should be used instead.

The sensitivity can be adjusted on 3 levels (from detecting words that are considered profanity only in certain cases to words that are very likely to be a profanity). The sensitivity of each computation is based on the setting retrieved from the database. Usage recommendations for insensitiveness check are the same as for style check. Most likely it will be useful for copywriters to avoid usage of words that might hurt the feelings of their customers.

²⁸More info about alex: <https://alexjs.com/>

4.2.9 Optimisation

After a successful initial implementation of the new Stranger Strings, there was still a big question regarding the usefulness of this tool in different localization projects. One of the options on how to answer the question was to try Stranger Strings on external data. With the help of my mentor Pavel „Strajk“ Doleček from Kiwi.com, we scraped localization data from an external website owned by one of the biggest European low-cost airlines. External localization data contained more than 6400 translation keys translated to 41 languages. This was a much bigger data sample than what was used for development. The scraped data were in nested JSON format which is supported by Stranger Strings, so no additional parsing was required.

After running the computation on these data, Stranger Strings run into a problem. As the data was too big the computation took a lot of time. The maximum time of computation for a Google Cloud Function is 540 seconds per invocation²⁹. The computation of checks for scraped data took even more so it never finished because the process under which the function ran was terminated.

This was quite a big problem which meant that Stranger Strings is unusable for bigger localization projects. After debugging the core problem was identified in *computeInconsistenciesOfTranslations* function located inside */functions/dbUpdates.js* file. This function serves for inconsistency computation in each translation. Computations in each check are usually based on regex search which is one of the most performance demanding operations in JavaScript. Usage of regex in JavaScript has a lot of pitfalls and if they are implemented wrongly it can result in 20 times slower performance [13]. Following the best practices in regex implementation did not help much because most of the regular expression are used inside of external libraries (e.g. *nspell*, *write-good*, *alex* etc.). Even removing functions from *lodash.js* that just add abstraction on top of native JavaScript function for better developer experience did not help.

Looking at the localization data closely, revealed an interesting pattern. A lot of times content of English translation is identical to other translations. The pattern was occurring in both external and internal data. The reason is that a lot of localization project uses English as they fallback alternative to missing languages. Also, 10 countries have English as official languages with just minor grammar variations (e.g. en-GB, en-US, en-CA, en-AU, etc.). Another reason is that many times the translation key contains just a single word that is product-related or just some international abbreviation and is intentionally untranslated (e.g. ISO, JavaScript, IATA, Reddit, etc.). Occurrences of this kind of words are surprisingly very frequent.

Reflecting the pattern in code meant doing computation for English translation first. During iterations of other languages, firstly content is compared to see whenever it is the same as in English translation. If yes then the results from computations done for English computation are used. Only if the content is different the iteration continues to do its own computations.

²⁹Quotas of Google Cloud Function: <https://cloud.google.com/functions/quotas>

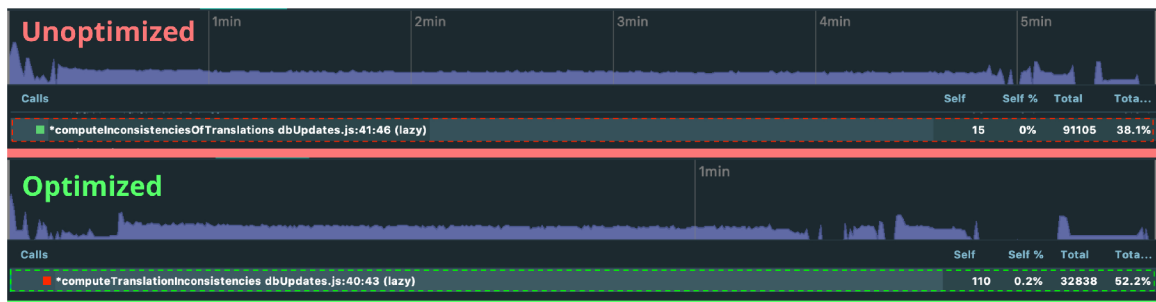


Figure 4.17: After optimization the computing done inside *computeInconsistenciesOfTranslations* function was moved inside new function called *computeTranslationInconsistencies*. *computeInconsistenciesOfTranslations* called this new function only for English translation and for translations that had different content. As can be seen from the picture, before optimization the computations were done 91 105 times. After optimization the function that performed all of the computations was called only 32 838 times (the data was exactly the same). The difference was evident also on the total execution time that went down from almost 6 minutes to less than 2 minutes.

Surprisingly after implementing this caching approach resulted in tremendous performance improvement which can be seen in Figure 4.17. The performance was profiled with *V8 profiler*³⁰. Benchmarks after optimization showed reduced execution time by $\approx 64\%$. Thanks to the optimization the function execution time was comfortably below 540second and it would work even for localization data-sets that would be 3 times larger.

4.3 Database

Stranger Strings uses Firebase Realtime Database. The advantage of this database is that data is synchronized across all clients in realtime, and remains available even when the app goes offline. Because of this, it is important to think about how users need to access data and then structure it accordingly. The Realtime Database is a NoSQL JSON database.

NoSQL databases are not using a relational model and operate without a schema. For example this allows more freedom when adding fields to database records without having to define any changes in structure first. The possible reason why Google Firebase offers only NoSQL based databases might be that they are strongly oriented towards running on clusters. So they can better utilize their cloud servers with parallelism. Another reason can be that it just offers a more convenient data interaction style for most of the modern mobile and web applications [14].

In the case of the *Google Realtime* database, this means, that it has different optimizations and functionality compared to a relational database. The old Stranger Strings structured the database rather in a logical way with multiple nested levels. Meaning that all translation keys were nested under key called items (named after items view) which can be considered as the root of the whole structure. Each translation key contained a list of locales that contained its translation content with the rest of the computed metadata.

This structure was wrong for the following reason. According to *Firebase* documentation the best practices regarding data structure is to avoid nesting data [5]. After the HTML request is sent for data at a location in the database, the HTML response will contain all

³⁰How V8 profiler works can be found here: <https://v8.dev/docs/profile>

of the child nodes under the requested location. In reality this means that the database structure should be structured based on the queries not the other way around and the redundancy should be secondary.

Even though it made sense for old Stranger Strings because it only relied on single data fetch after each app initialization on the client-side. New Stranger Strings displays and store much more data. Because of this the structure of the database was changed to be flat as possible. The data was split based on request calls. The database structure can be seen in Figure 4.18.

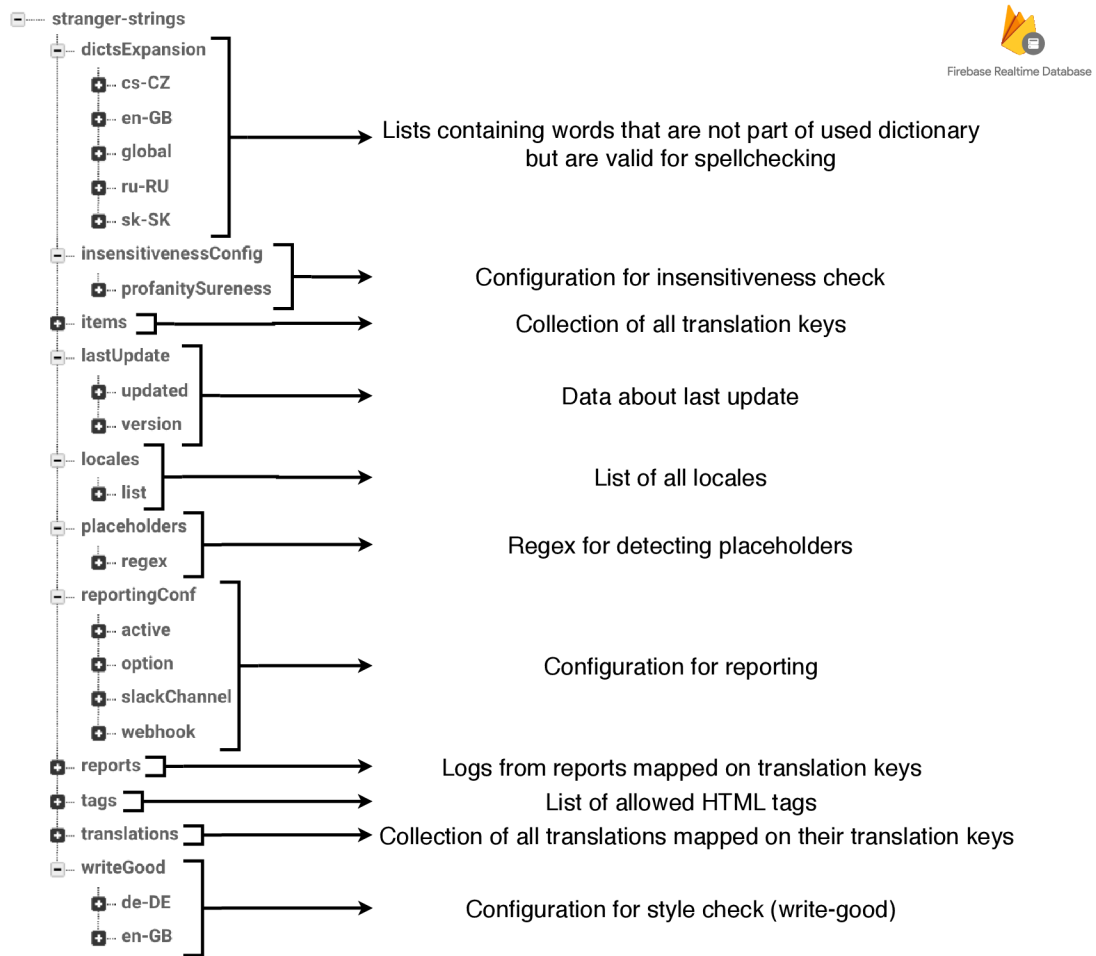


Figure 4.18: Database structure of new version of Stranger Strings in Firebase Realtime Database

The biggest weaknesses of the old Stranger Strings from a database point of view was its security. The database was completely unprotected. Even non-authenticated users had assigned read and write access. In reality this meant that anyone was able to obtain whatever data were in the database or completely delete it. Obviously this was unacceptable in the new Stranger Strings as security is one of its goals.

Firebase provides the possibility to distinguish between authenticated users, non-authenticated users, and administrators accessing with Firebase SDK³¹ (in case of Stranger Strings those are Google Cloud Functions). New Stranger Strings restricts writing access to all users and only allows it for parts of the database that are connected to configurations. For example this means that nobody has writing access to translations or translation keys except Google Cloud Functions. Apart from that all reading access rights and some writing right regarding configurations are only available to authenticated users.

4.4 Front-end of Stranger Strings

As was mentioned earlier several times, the front-end part of Stranger Strings is implemented in JavaScript *Vues.js* framework. In previous version of Stranger Strings the whole front-end consisted of only 4 Vue files³² and few JavaScript files containing helper functions. In other words, the front-end was a really small part of the whole project. From the beginning, this was expected to change which should reflect on a more thoughtful code structure. The UI/UX was quite poor as the application missed a specified purpose and was intended to be used by only a few users. From the code-side perspective, the better choice was not to reuse the old implementation and rather start from scratch with a new purpose in mind. It was hoped that this will not just make the code clearer, the transformation to open-source easier and future maintenance undemanding. But will also result in better UI/UX. Lately, this expectation turns out as partially wrong. No significant UI/UX improvement was revealed during further user testing, however, this topic is covered in the following chapter 5.

As a first step I requested feedback from others how they use old Stranger Strings and what would they like to have in new Stranger Strings. The answers were put together with personal observations of usage. This was a good first stepping stone towards building new Stranger Strings.

The client application was setup via *Vue CLI*³³. The core */src/App.vue* the file contains the root of the DOM tree and everything regarding authentication. The authentication method is chosen based on the configuration in */common/config.js*. The possible options are accessible for everyone (useful for demonstration purposes), login by an email link, and log in with google account. The landing page with the authentication button was redesigned. In comparison to old landing page (Figure 2.3) the new one (Figure 4.19) is simpler and more visually appealing at the same time. The background contains moving particles effect created with *particles.js* library³⁴.

³¹More info about security in Firebase Realtime Database: <https://firebase.google.com/docs/database/security/securing-data>

³²*App.vue* (core file), *Welcome.vue* containing landing page, *Navbar.vue* containing top navigation bar and *Items.vue* that contained the rest of the page

³³Vue CLI: <https://cli.vuejs.org/>

³⁴Particles.js library: <https://github.com/VincentGarreau/particles.js/>

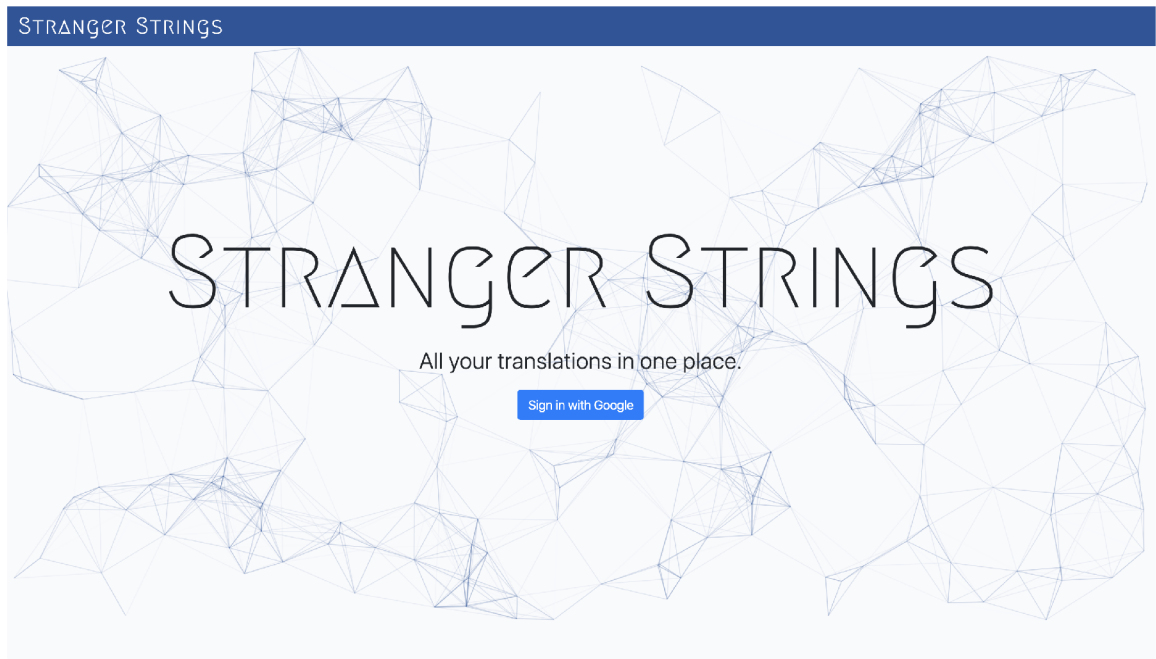


Figure 4.19: The landing page of the new Stranger Strings. The small particles in the background are moving and are even interactive on click. This effect is created with *particles.js* library. The implementation of landing page can be found in `/src/components/Welcome.vue`.

Even though Stranger Strings is intended for desktops and tablets only, basic responsiveness was one of the objectives in new Stranger Strings. Previously it was impossible to interact with applications on mobile devices. This is not the case in the new version.

After logging in the application users arrive at to so-called 'items' view displayed in Figure 4.20. This view contains a table with translation keys and their additional information. From a UI perspective the concept is very similar to the previous version. However all of the filtering options were removed because nobody needed them and only the search bar persisted. The searching was improved because previously it only worked on searching in keys and not its English content. The searching in new Stranger Strings is done on both keys and its English contents at the same time and is performed with a fuzzy searching technique provided by *Fuse.js*³⁵ library.

³⁵Fuzzy-search library Fuse.js: <https://fusejs.io/>

Key (showing 6308 / 6308)	Progress	Actions	English
booking.trip_survey.thank_you	28 / 10	{}	__name__, thank you for your response.
booking.trip_survey.undo	28 / 10		Undo
booking_unchecked-agreement	35 / 3	>	To continue with this booking, you must agree with our terms and conditions.
booking_user_booking_changes.card_title	28 / 10		Requests
booking_completion.boarding_passes.app_promo_link	27 / 11	>	Get the Kiwi.com app today
booking_completion.boarding_passes.app_promo_text	28 / 10		Traveling is easier with electronic boarding passes.
booking_completion.boarding_passes.know_more	27 / 11		Learn more
booking_completion.boarding_passes.notes.already_flow	27 / 11	>	Online check-in was available on our site until __availableUntil__
booking_completion.boarding_passes.notes.at_the_airport	27 / 11	>	You must check in at the airport (a fee may be required)- Online check-in on our site was available
booking_completion.boarding_passes.notes.available	27 / 11		Available:
booking_completion.boarding_passes.notes.boarding_pass_never_available	27 / 11	>	We don't do online check-in for this flight. Please check in at the airport.
booking_completion.boarding_passes.notes.call_to_action	27 / 11	>	Add your details for online check-in
booking_completion.boarding_passes.notes.download	27 / 11		Download
booking_completion.boarding_passes.notes.error	28 / 10		Check-in status unavailable. Try a bit later or contact our Customer Support.
booking_completion.boarding_passes.notes.nonflight_ticket_processing	27 / 11		Your tickets are being processed
booking_completion.boarding_passes.title	27 / 11		Boarding passes
booking_completion.button_label	27 / 11	>	Back to Manage My Booking
booking_completion.checkin.button_save	27 / 11		Save

Figure 4.20: Items view of new Stranger Strings. To compare with previous version see Figure 2.4. The implementation of items view can be found in `/src/views/Items.vue`

Another big difference between both versions is how data are loaded from the database. Previously, data were fetched only on 'items' view load and not anymore. This was not wasting the potential of *Firebase Realtime Database*, as it offers the possibility to display data from the database in real-time. New Stranger Strings uses *Vuefire*³⁶ library. One of the features that real-time database made possible was a notification about ongoing computation or data-source update. After the update, all of the data on the client-side are immediately updated including a timestamp of the last update. This is especially helpful for big localization projects.

One of the tasks that resulted from observation was the task to improve UX in sharing translations keys. When users wanted to share some translation key in old Stranger Strings they simply copied the URL. This is intuitive however the link contained long and confusing queries that were very hard to read and users could not immediately spot where the link will redirect. The example of such a link can be seen in Figure 4.21. The idea was to clean the URL so that once users see the link, they can immediately recognize that it belongs to Stranger Strings and even identify which translation key will show up after clicking.

³⁶Realtime database binding library *Vuefire*: <https://vuefire.vuejs.org/vuefire/>

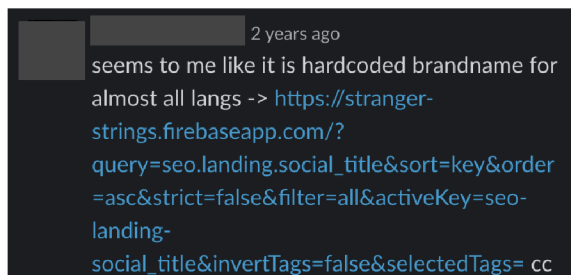


Figure 4.21: Example of how translation keys were shared in old Stranger Strings. In new Stranger Strings the same link looks following `https://stranger-strings.firebaseio.com/items/seo-landing-social_title`

This was accomplished with implementing *Vue Router*³⁷. The previous URL was cleaned from the query string and mapped to views and translation keys. So now when someone wants to share translation key the URL will be `<domain>/<view>/<translation_key>`. The view can be either 'items' view or 'collections' however the latter is still in development. The *Vue Router* also added the possibility to integrate HTML5 history mode which is very useful since Stranger Strings is a single page application. The full implementation of *Vue Router* can be found in `/src/router.js`.

From the beginning, the new Stranger Strings was meant to offer customization for each user. The previous version lacked state management patterns because there was no need for it. Since customization requires keeping certain states across the whole application, *Vuex*³⁸ library was used as a centralized store for all of the components in the application. The only downside of this state management tool is that it is not persistent after closing. Storing each user setting in the database was intentionally avoided to keep the database size as possible, but keeping the user setting locally was still desired. The solution was keeping the Vuex state in the browser via *localStorage*. Plugin for Vuex called *vuex-persist*³⁹ helped to achieve this desired feature. The whole implementation of store in *Vuex* can be found in `/src/store.js`.

Once Stranger Strings was able to keep state across the whole application, the implementation of settings could follow. During the project setup Stranger Strings separated maintainer level configuration from user and administrator access levels. Now in the application Stranger Strings separates user access level from administrator access level.

Each user can assign custom importance to each check (available are 3 levels of importance: suggestion, warning, error). The difference between importance levels is in color highlighting of the text. Each of the checks can be turned off at any time, so the corresponding warning is not shown anywhere. Apart from checks configuration users can configure their language importance preferences and there is also an option to chose how to wrap the text in the English preview column in the 'items' view. Additionally, users that are considered administrators can change settings that directly affect computations. Both administrator and user configuration are displayed in Figure 4.22. Whenever some administrator configuration changes, the change is immediately written in the database, and *computeInconsistencies* function on Google Cloud Functions is triggered via HTML trigger hook.

³⁷Vue Router is the official router for *Vue.js*. More info:<https://router.vuejs.org/>

³⁸State management pattern + library Vuex: <https://vuex.vuejs.org/>

³⁹Vuex plugin *vuex-persist* for saving state of app on persisted browser storage: <https://github.com/championswimmer/vuex-persist>

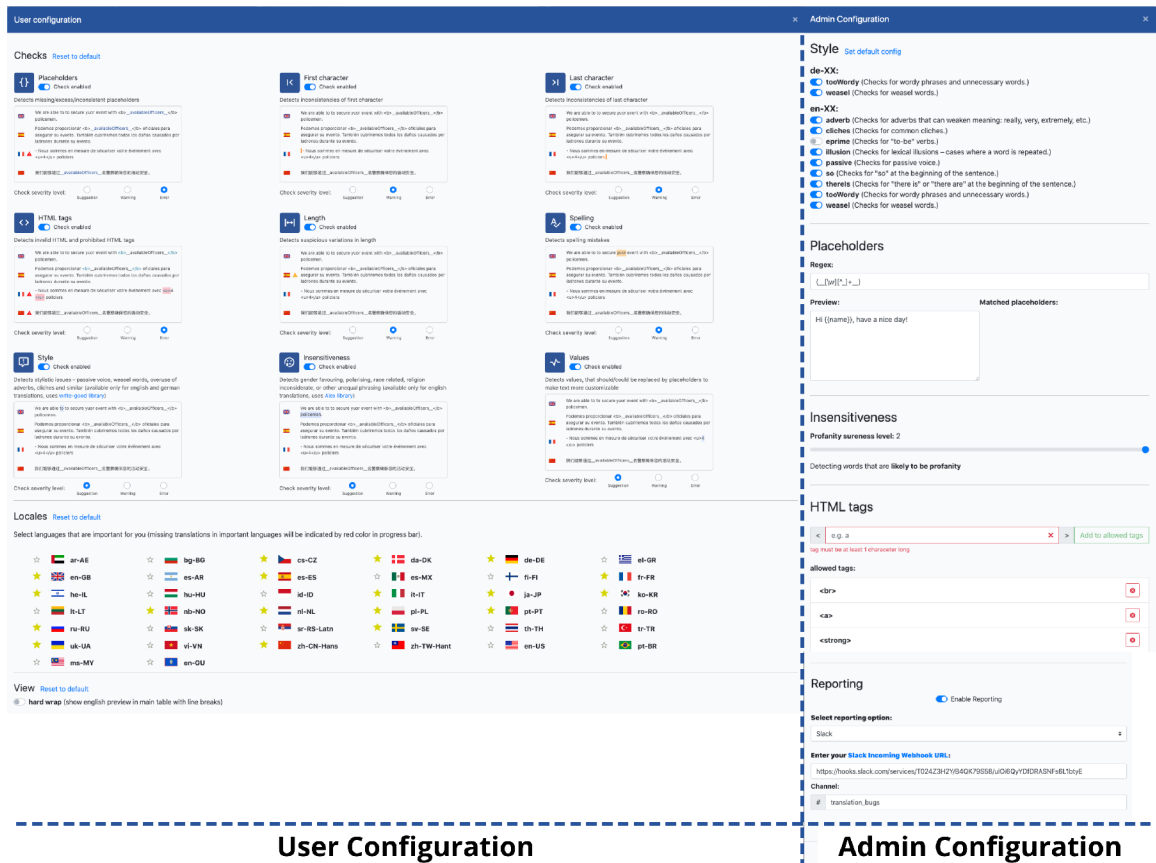


Figure 4.22: Screenshots of user configuration (left) and admin configuration (right). Screenshots were taken from the administrator point of view, therefore changes are allowed even in the admin configuration modal.

Each translation key in items view is clickable and opens modal that contains the translations of the selected translation key. Previous Stranger Strings simply displayed all of the translations along with their metadata in a simple table. New Stranger Strings displays only relevant data. For example previously the table contained columns where each row listed all placeholders that are in corresponding translation. New Stranger Strings first computes which languages are missing placeholders with the same algorithm that is used in *Google Cloud Functions*. Than displays warning about missing placeholders only for relevant translations.

Another way how new Stranger Strings displays inconsistencies in translation key detail modal is directly inside text. Whenever some inconsistency or error can be highlighted directly in the text, Stranger Strings does it. For example, missing placeholder or excessive translation length cannot be highlighted inside text but that is not the case for spellchecking error or missing question mark at the end of the sentence. How Stranger Strings highlights issues directly inside the text can be seen in Figure 4.23.



Figure 4.23: Screenshot of the key detail view. All of the red and orange marks are issues that on mouse hover show further description. Additionally, the red triangles icons before text indicate other issues that cannot be marked directly inside text.

The component for highlighting is the most complex part of the whole UI. The first problem was to parse translation into tokens based on computed inconsistencies from the database. The parser needed to identify each inconsistency (spellchecking error) or entity (HTML tag) and separate it from the rest of the string while keeping the same order as in original translation. The initial string was tokenized, and each token got assigned parameters like content, its position in the order, and its type. It was important to keep in mind that some languages are written from right to left. Stranger Strings uses *rtl-detect*⁴⁰ library to detect RTL languages. There were many edge cases that this component needed to handle. For instance, it turned out that once the translation is larger than tokenization approach causes problems with wrapping. This is because, in parsing, Stranger Strings splits the string containing translation to tokens that are later represented as nodes in the Document Object Model (DOM) tree. However users should not see that because the text should behave the same as if it was a single block of text. The solution was to slice the tokens that did not contain any other information apart from its string content because those were the largest tokens and caused wrapping problems. These tokens were sliced by white-spaces in all languages that are by nature segmented by white-spaces. The nature of the tokenization task in unsegmented languages like Chinese, Japanese, and Thai is fundamentally different from tokenization in space-delimited languages like English. The lack of any spaces between words makes it much harder to correctly slice strings into segments by word boundaries [9]. Since Stranger Strings doesn't need to tokenize this words by strictly by word boundaries, it segments these languages character by character.

Even before implementation it was evident that this part will be very complex. Therefore it was developed with a Test-driven development (TDD) method. TDD helps reduce the defects in code increases the maintainability of code [11]. TDD method undeniably helped in this specific case and probably even sped up the whole development process of

⁴⁰Library for detecting RTL languages: <https://github.com/shadiabuhilal/rtl-detect>

this feature. The tests can be found in `/src/tests/highlighting.spec.js` and the highlighting component in `/src/components/Highlighting.vue`. The highlighting component is also used in checks configuration where it serves as a demonstration of each check functionality.

Only downside that resulted from highlighting inconsistencies inside text was that it was no longer possible to copy the content of translation. Because the copied string will be malformed with excessive white-spaces. Partially, this was solved by „copy to clipboard“ button implemented with `vue-clipboard2`⁴¹ library which provides binding for `clipboard.js`⁴² library.

Right next to the „copy to clipboard“ button, there is another button for reporting bugs. Clicking this button opens the reporting modal displayed in Figure 4.24. After users spot some translation mistakes that should be immediately fixed, they can use this reporting feature to easily report the error. Reporting modal contains a partially predefined report form. After filling necessary information the user just submits the report and Stranger Strings will format the whole report and send it. The reporting destination can be configured by an administrator in the settings.

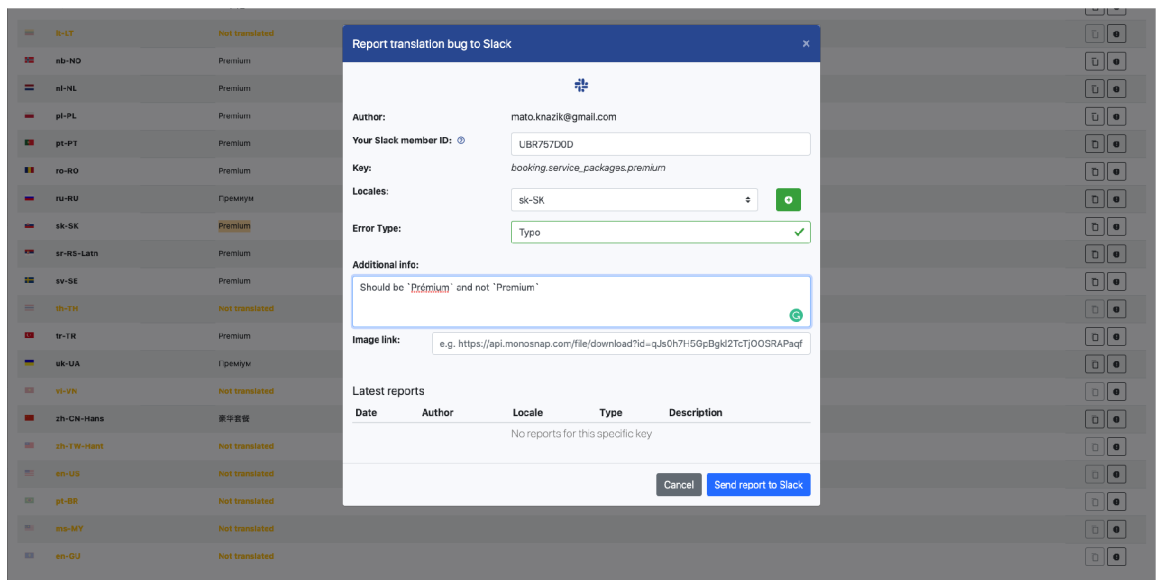


Figure 4.24: Reporting modal for sending translation bugs reports to *Slack*.

At the time of writing only supported reporting module was *Slack* module. *Slack* reporting module was implemented via *Webhooks* provided by *Slack* API⁴³. Users only need setup their *Slack* app in their desired *Slack* channel and then generate *Webhooks* for this specific *Slack* app. Once *Slack* *Webhooks* is generated, the administrator inserts the URL of this *Webhook* in Stranger Strings' reporting configuration, and everything is set. The reporting feature in old Stranger Strings worked on the same principle however the formatting of the report was changed in the same way as the reporting form modal. The comparison of generated **Slack** reports of new and old Stranger Strings can be seen in Figure 4.25.

⁴¹ *Vue.js* binding for *clipboard.js*: <https://github.com/Inndy/vue-clipboard2>

⁴² *clipboard.js* library: <https://clipboardjs.com/>

⁴³ *Slack* *Webhooks*: <https://api.slack.com/messaging/webhooks>

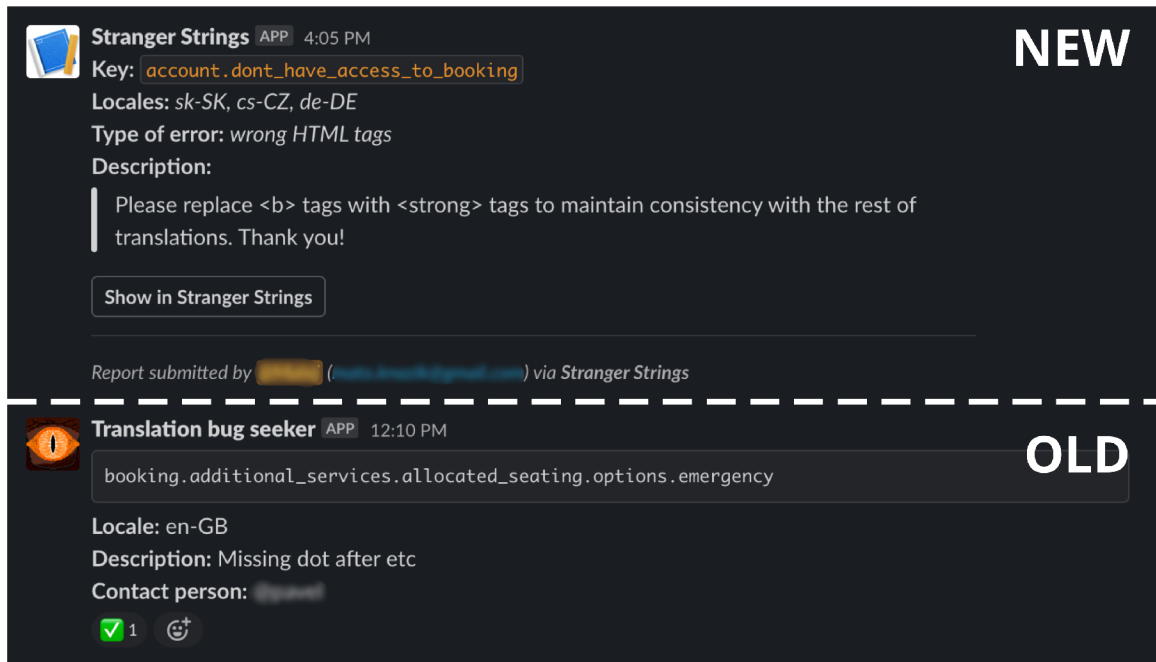


Figure 4.25: Comparison of *Slack* reports generated by new Stranger Strings (top) and old Stranger Strings (bottom). The new one is better structured and allows the reporting of multiple locales at once. Additionally, the new *Slack* report contains a button that opens the translation key directly in the Stranger Strings.

4.4.1 Optimisation

One of the problems repeated in feedback from users of old Stranger Strings was its performance. The loading and response time was unusually long. This problem kept growing in a new version as multiple components and libraries were added to the application. It was still somehow tolerated as the translation data were just too big.

Similarly as the back-end part was tested on external data (mentioned in Subsection 4.17) the front-end part was as well. After external localization data were successfully processed by function and uploaded to the database, the application loaded them. Unfortunately, the app kept crashing on each load so the initial 'items' view was never shown. The much larger external data caused that app tried to render so many components that it simply runs out of memory.

Stranger Strings solved this issue by not rendering the whole table in the 'items' view. Instead it only renders as many translation key rows as can be displayed on the screen. When users scroll the table, the rendered DOM nodes are being reused for other rows of translation keys. This means that even though only a small subset of rows is rendered on the screen, it does not affect the user because it is exactly the subset of rows that are currently visible on the screen. This solution was implemented thanks to *vue-virtual-scroller*⁴⁴ library created by one of the *Vue.js* core team members Guillaume Chau.

After implementing virtual scrolling in Stranger Strings the app was able to display data of any size. As a bonus Stranger Strings become faster than ever before.

⁴⁴vue-virtual-scroller library: <https://github.com/Akryum/vue-virtual-scroller>

Chapter 5

User Testing

During the early stages of development, almost all of the design choices were based on intuition. Objectively this was not ideal and to continue further with development, the project needed some metrics or data that design decisions will rely on. The decision was to introduce both quantitative and qualitative research into the Stranger Strings development process. Quantitative research is designed to gather data points in measurable, numerical form. Qualitative research relies on the observation and collection of non-numerical insights such as opinions and motivations [7].

Once the prototype version of Stranger Strings was ready, the project was ready for its first qualitative research iteration in form of usability testing. The idea was to volunteer who would participate as a testing subject. We were looking for someone who would be considered as a targeted future user (e.g. QA tester) and had no prior experience with Stranger Strings. We found exactly someone that matched those expectations and as a bonus our test subject was familiar with the basics of localization.

The whole user testing including, preparation, testings process itself, and evaluation were inspired by book *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems* by author Steve Krug [10].

The testing was done in meeting room, and the user interaction with Stranger Strings was screened on the big television screen in front of us. The testing process itself could be summarized into these steps:

1. Very brief introduction of Stranger Strings and its purpose.
2. Share the goal of user testing with participant so it is clear we are not testing the participant, but instead the participant helps us with testing the application.
3. Once the participant was comfortable with talking we proceeded to user testing.
4. The participant was asked to freely explore the application while explaining verbally their interpretation of what they see. (without interruptions)
5. The participant was assigned a few tasks to perform on his own. These tasks were designed to imitate real-world tasks. (without interruptions)
6. In the end participant was asked to explain some particular choices they made in previous tasks
7. Debriefing that results in a list of observed usability problems along with a plan on how to fix them.

In total, 2 user testing iterations were made during the development process. The first user testing was a total shock as it revealed that participants had no clue what the checks are supposed to do. Apart from a complete misunderstanding of checks users did not what reporting feature does or was confused by the 'collection' view. An interesting observation was, that the first direction user took to learn about Stranger Strings was in user settings. However, at that time, each little configuration had its own modal so there was no place to get the bigger picture over whole Stranger Strings. The user was testing Stranger Strings in a state that is illustrated in Figure 5.1.

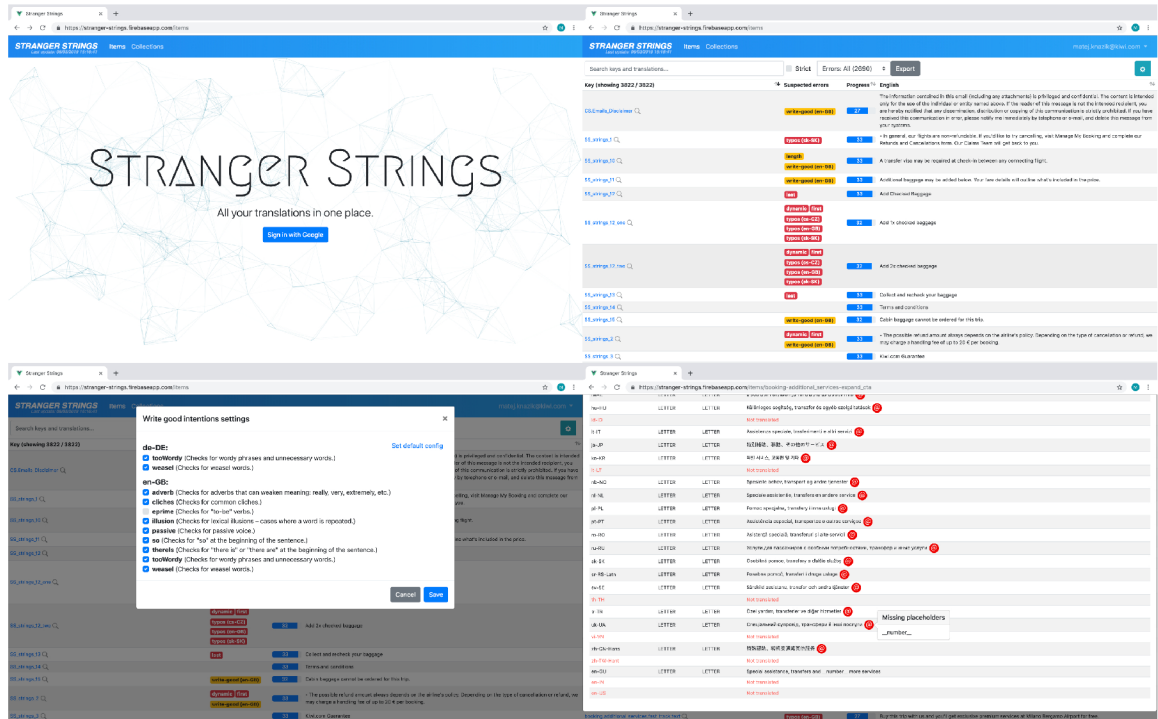


Figure 5.1: Collage of screenshots from Stranger Strings at the time of the first user testing iteration. In the top left corner is the landing page, to the right is the items view. The translation key detail view can be seen in the bottom right corner and in the bottom left corner is configuration modal for the style check.

In response to the mentioned findings, the settings were grouped into 2 different modals. One for user settings and another for administrator settings. Even that non-administrator users cannot change administrator settings they still remained read-only access to these settings, so they can better understand all of the features. Additionally, the 'items' view table was changed by replacing the general error column with multiple columns for every single check. There was added ability to turn off each check by clicking in the header of its column. The translation key detail modal was cleaned to only contain relevant information. The outcome of changes can be seen in Figure 5.2, which illustrates the state of Stranger Strings before the second iteration of user testing.

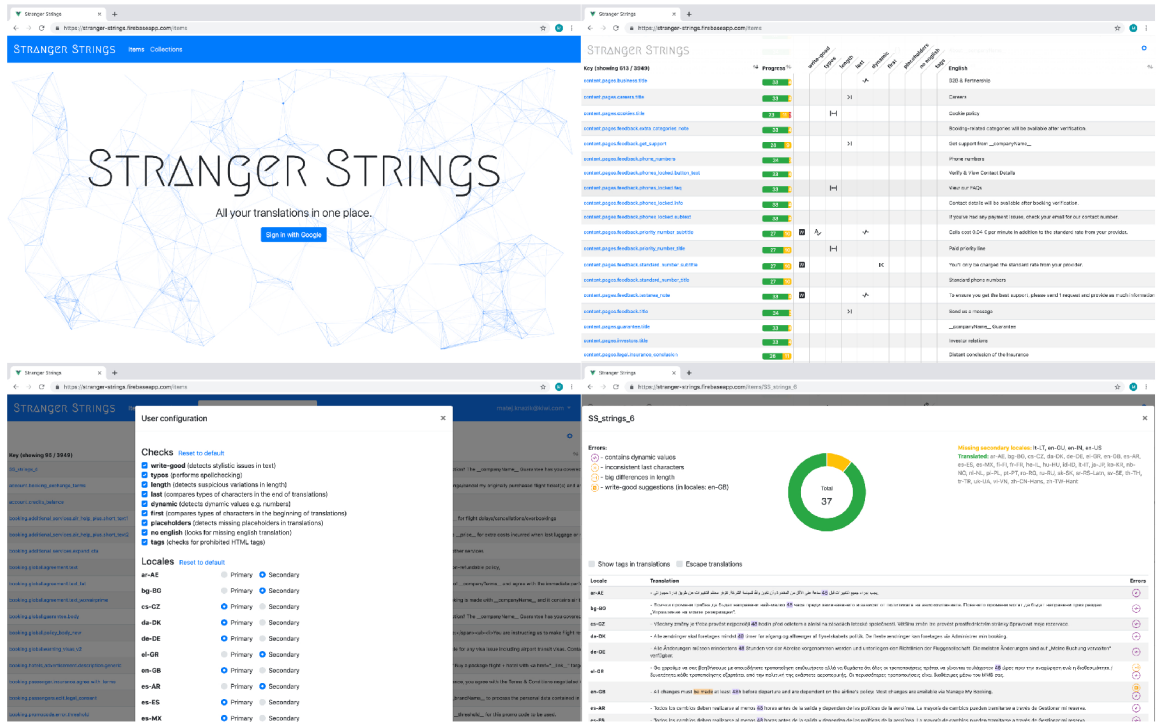


Figure 5.2: Collage of screenshots from Stranger Strings at the time of the second user testing iteration. The main differences, compared to the version before are the checks indications in key detail and items view.

The second user testing iteration took place 2 months after the previous one. The whole process was almost the same. This time the participant immediately understood the concept of checks, which was a sign of improvement from the previous session. Even though there were still some misunderstandings regarding some checks, importance levels, or reporting. The whole list of notes from the second user testing session can be seen in Figure 5.3.

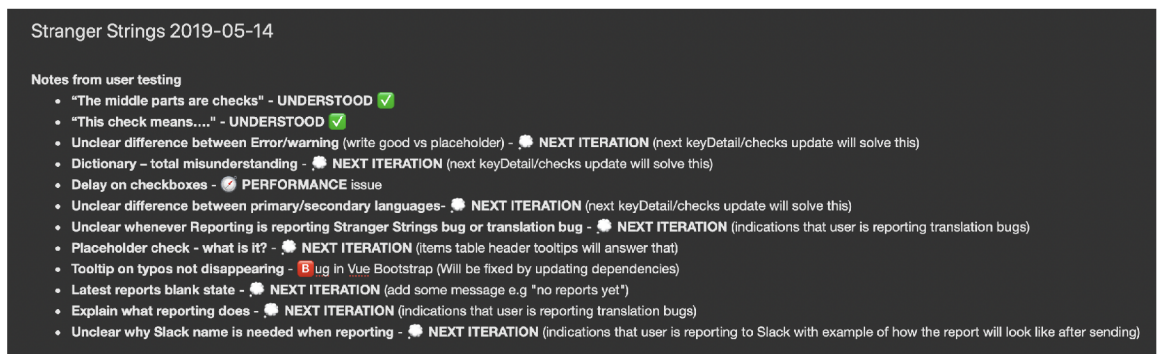


Figure 5.3: Notes taken during the second iteration of user testing.

After the second user testing, all of the observed issues were rather small. So the proposed fixes were inspired by Steve Krugs' advice to fix the usability problem with the easiest and smallest possible changes [10]. So each of the checks got its own description and example of how it works, so once the user changes the importance level the change

immediately reflects in the example. To include these checks' explanations with examples directly in the 'items' view, it was simply reused in tool-tip of each check which showed on mouse hover (see Figure 5.4). Importance settings for languages were solved simply by added description. The same applied to the report modal, which was solved by adding multiple tool-tip descriptions and examples in the form of images. After this testing session, there was a big decision taken regarding the 'collection' view. As it still an experimental feature I decided to hide it in UI to not confuse users.

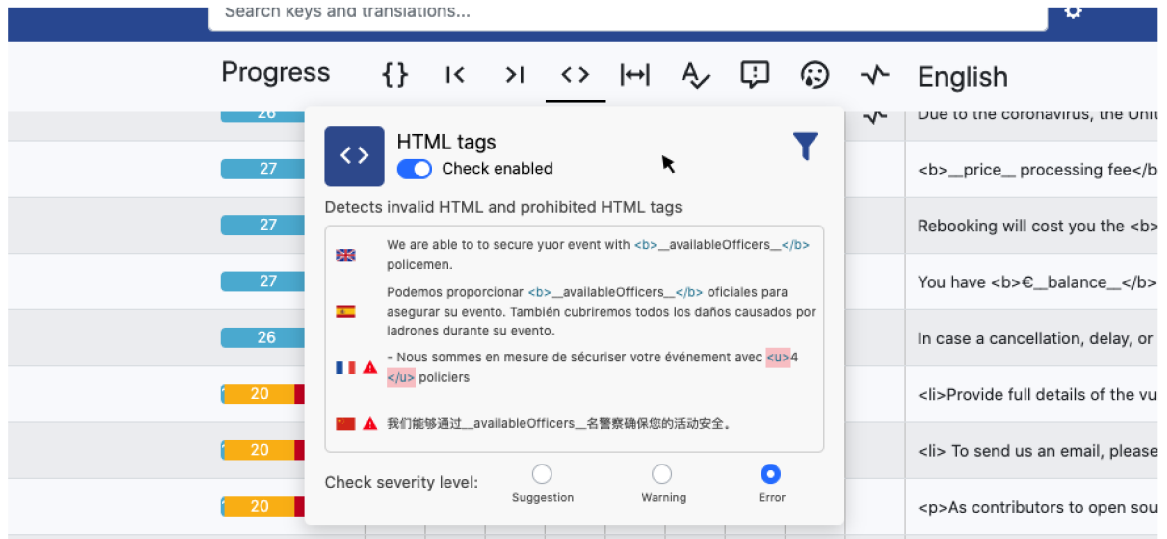


Figure 5.4: Illustration of how Stranger Strings describes the functionality of checks directly in items view, by reusing components from check configuration modal. The check description along with its settings and interactive example can be seen on mouse hover in the main table header.

For a quantitative part of the research, I choose to track usage stats of Stranger Strings instance that I maintain at Kiwi.com. So with the release of the first changed version of Stranger Strings in 20.3.2019 all of the usage stats were tracked via *Google Analytics* for this specific instance. The analytic metrics were meant to provide quantitative feedback on how many and how often users interact with this Stranger Strings instance.

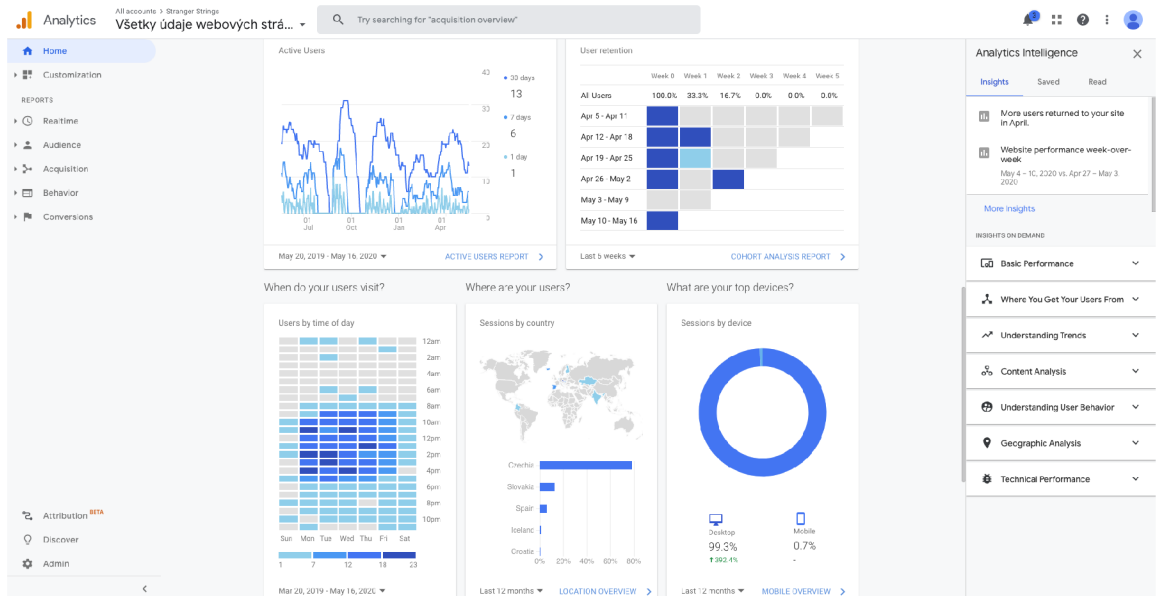


Figure 5.5: Screenshot from *Google Analytics* that tracks usage data of Stranger Strings in Kiwi.com. Graph showing usage stats by the time of the day clearly indicates the correlations working hours. The pie chart shows that almost all of the devices accessing Stranger Strings are desktops.

The usage was tracked in a period longer than one year except for two months during which the Stranger Strings was in maintenance mode therefore not tracked. This data can be seen in Figure 5.5. The biggest spike in active users was recorded around 17.9.2019 which correlates with a release that drastically improved performance.

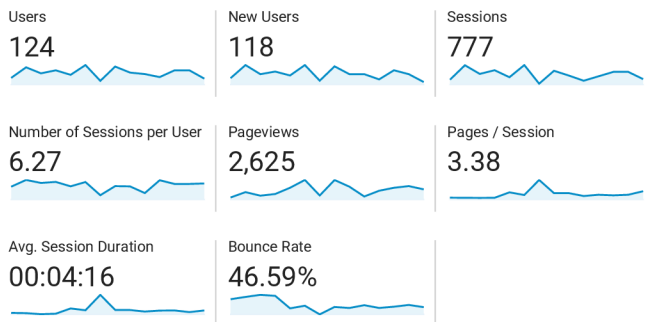
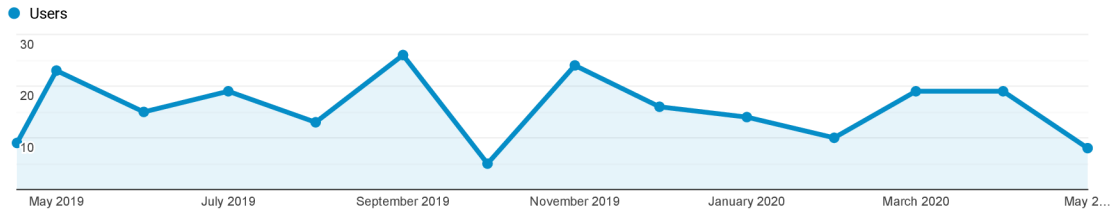
The final statistics numbers were adjusted by this not tracked period. On average 16.53 different users used Stranger Strings at least once in a month. From a weekly perspective it was around 7.97 unique users per week. On average each user had 6.27 recorded sessions with an average duration of 4 minutes and 16 seconds per session. The top 3 countries from which users visited Stranger Strings correlate with the location of company offices. This means that Stranger Strings was used by multiple teams. From my personal survey I have received confirmation, that it is being used by developers, QA testers, and even by some people from product management and members of translation management. In company *Slack* channel dedicated to translation bugs, there were in total 51 reports from Strange Strings reported by 10 different users. The whole *Google Analytics* report on audience can be seen of Figure 5.6.

Audience Overview

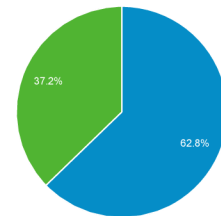
Apr 17, 2019 - May 16, 2020

All Users
100.00% Users

Overview



New Visitor Returning Visitor



Country	Users	% Users
1. Czechia	107	78.68%
2. Spain	10	7.35%
3. Slovakia	10	7.35%
4. Croatia	2	1.47%
5. Colombia	1	0.74%
6. Finland	1	0.74%
7. India	1	0.74%
8. Iceland	1	0.74%
9. Kazakhstan	1	0.74%
10. Netherlands	1	0.74%

Figure 5.6: Audience overview report from Google Analytics for monthly usage stats over year.

Chapter 6

Conclusion

This bachelor thesis described development process of tool called Stranger Strings that is able to improve localisation quality. Firstly it explained problems that are quite common in localisation project and how it inspired idea to create open-source tool that would help to fix them. Later this thesis enumerated the resources and design changes that are needed for development of such tool. It also explained, the approach that was taken for implementing solutions of localisation problems and revealed the caveats that appeared during the development. Lastly it provided insights in user testings that shaped this project to its final form.

The aim of this bachelor thesis was to create tool that would prove its value which can be verified by statistics about it usage and numbers of fixed localisation issues. The expectations were placed quite high in form of 10 unique users per week and 20 unique users per month. After an year of monitoring the real usage stats showed on average 7.97 unique users per week and 16.53 unique users per month. Those numbers are definitely not meeting the expectations, but are not too far off. It needs to be reminded that usage of Stranger Strings was only voluntary and people used it because it helped them do their job better or more efficiently. The fact that this project fulfills its purpose can be seen on 51 observed reports generated by Stranger Strings, which is 31 more than initially expected. Stranger Strings is currently being used by multiple teams across various profession including QA testers, developers, product managers and localisation team members. In comparison old Stranger Strings was used by just small number of developers. Based on this, the whole project can be considered as success, because it proves its usefulness.

All of these usage stats were measured only on single localisation project, however as Stranger Strings became open-source project anyone can use it for themselves. It can be found on *GitHub* <https://github.com/kiwicom/stranger-strings>, and is very much open for anyone's contributions. There is also website that demonstrates the possibilities that Stranger Strings offers: <https://stranger-strings-showcase.firebaseio.com/items>. Hopefully, it will find its way into other localisation workflows and will help improve the localisation quality of many other products.

Bibliography

- [1] ACUNA, K. Why Pixar changed several scenes in ‘Inside Out’ for foreign audiences. *Business Insider: Tech Insider*. july 2015.
- [2] AHUVIA, Y. React vs. Vue (vs. Angular). *Medium.com: Fundbox Engineering*. june 2018.
- [3] CROES, J. K. and DEXTER, K. *THE ART OF JAPANESE PUNCTUATION* [online]. Tofugu, march 2016 [cit. 2020-05-21]. Available at: <https://www.tofugu.com/japanese/japanese-punctuation/>.
- [4] DEPALMA, D. A., SARGENT, B. B. and BENINATTO, R. S. *Can’t read, won’t buy: Why language matters on global websites*. 1st ed. Common Sense Advisory, Inc., 2006. ISBN 1-933555-30-0.
- [5] DEVELOPERS, G. Structure Your Database. *Firebase Realtime Database* [online]. [cit. 2020-05-21]. Available at: <https://firebase.google.com/docs/database/web/structure-data>.
- [6] GALA. What is Localization? *Globalization and Localization Association* [online]. [cit. 2020-05-22]. Available at: <https://www.gala-global.org/industry/intro-language-industry/what-localization>.
- [7] GLEASON, D. *Qualitative vs. quantitative user research: the answers you will (and won’t) get from each* [online]. hotjar, march 2019 [cit. 2020-05-21]. Available at: <https://www.hotjar.com/blog/qualitative-vs-quantitative-user-research/>.
- [8] IBM. Text translation design. *IBM Knowledge Center* [online]. [cit. 2020-05-21]. Available at: https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/nls/rbagstextdatatransdesign.htm.
- [9] INDURKHYA, N. and DAMERAU, F. J. *Handbook of natural language processing*. 2nd ed. Chapman & Hall/CRC, 2010. ISBN 978-1-4200-8593-8.
- [10] KRUG, S. *Rocket surgery made easy: the do-it-yourself guide to finding and fixing usability problems*. 1st ed. New Riders, 2010. ISBN 0321657292.
- [11] MAKINEN, S. and MÜNCH, J. Effects of Test-Driven Development: A Comparative Analysis of Empirical Studies. *Lecture Notes in Business Information Processing*. january 2014, vol. 166, p. 15. ISSN 1865-1348.
- [12] NGUYEN, V. *ESLint: The Essential Facts About Essential Front End Tools* [online]. freeCodeCamp, august 2019 [cit. 2020-05-19]. Available at: <https://www.freecodecamp.org/news/the-essentials-eslint/>.

- [13] RZAŞA, M. Performance of Regular Expressions. *Medium.com: TextMaster Engineering*. october 2018.
- [14] SADALAGE, P. J. and FOWLER, M. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. 1st ed. Addison-Wesley, 2013. ISBN 978-0-321-82662-6.
- [15] SAMANTA, J. Outsourcing vs in-house development. *Medium.com: Hacker Noon*. may 2019.