

# A Microservice Redesign of Search and Inference for the Linguistic Website Terraling

Shailesh Vasandani  
New York University  
shailesh@nyu.edu

Hannan Butt  
New York University  
hannan@nyu.edu

Dennis Shasha  
Professor  
New York University  
shasha@cims.nyu.edu

**New York University**  
Computer Science  
Technical Report TR2021-999  
Fall 2021

**Keywords**—Microservices, web development, linguistics, React, Golang, performance, software engineering

# Contents

<b>1: Abstract</b>	<b>3</b>
<b>2: What does Terraling do?</b>	<b>3</b>
<b>3: Functionality addressed</b>	<b>7</b>
Filter	8
Compare	8
Cross	9
Implication	10
Similarity	11
<b>3: The problem</b>	<b>13</b>
Maintainability	13
Availability	13
Architecture	14
Interface	14
Performance	14
<b>4: The solution</b>	<b>14</b>
<b>5: Results</b>	<b>16</b>
Maintainability	16
Availability	16
Architecture	16
Interface	16
Performance	22
Queries	23
<b>References</b>	<b>25</b>

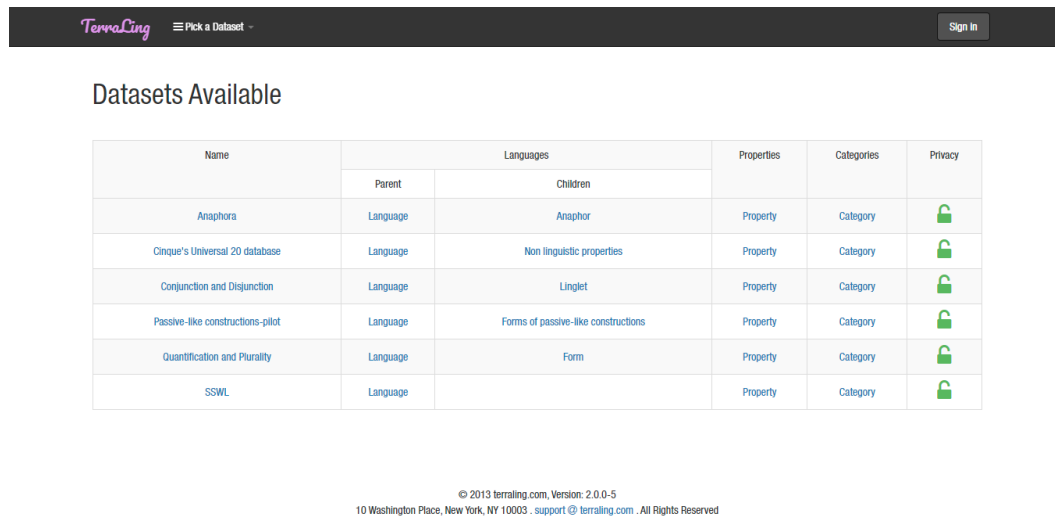
# 1: Abstract







The linguistics web application Terraling serves many useful functions for linguists. By extracting the critical path for linguistic analysis into microservices, we are able to improve user experience, optimize performance, and increase maintainability.

By using a modern stack with a React frontend and a Golang backend, performance was improved by 700 times. In addition, new features can be added with high velocity. The website can be accessed on any device on the Terraling website.

# 2: What does Terraling do?

Terraling is a linguistic web application that allows for the storage, retrieval and examination of linguistic data. The application is schema-agnostic, and can accept multiple types of linguistic data. In fact this is one of its key features; linguists can create groups that have custom formats for their research (Fig. 1).



Name	Languages		Properties	Categories	Privacy
	Parent	Children			
Anaphora	Language	Anaphor	Property	Category	
Cinque's Universal 20 database	Language	Non linguistic properties	Property	Category	
Conjunction and Disjunction	Language	Linglet	Property	Category	
Passive-like constructions-pilot	Language	Forms of passive-like constructions	Property	Category	
Quantification and Plurality	Language	Form	Property	Category	
SSWL	Language		Property	Category	

© 2013 terraling.com, Version: 2.0.0-5  
10 Washington Place, New York, NY 10003 . support @ terraling.com . All Rights Reserved

**Figure 1.** Terraling's groups

In a group, linguists define their own properties, and they and language experts set values for these properties for each language (Fig. 2). Linguists and language experts can also add examples that

illustrate these properties (Fig. 3). Properties are entirely determined by linguistic need, and the backend is designed such that any type of property can be accommodated.

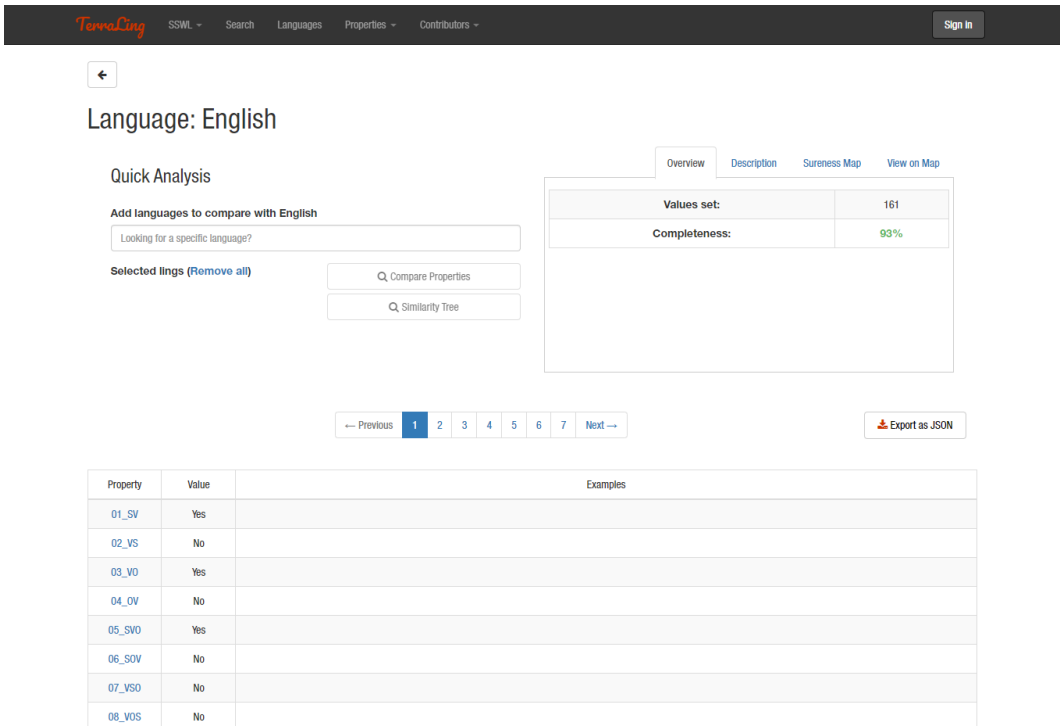


Figure 2. Properties in the “SSWL” group, for the language “English”

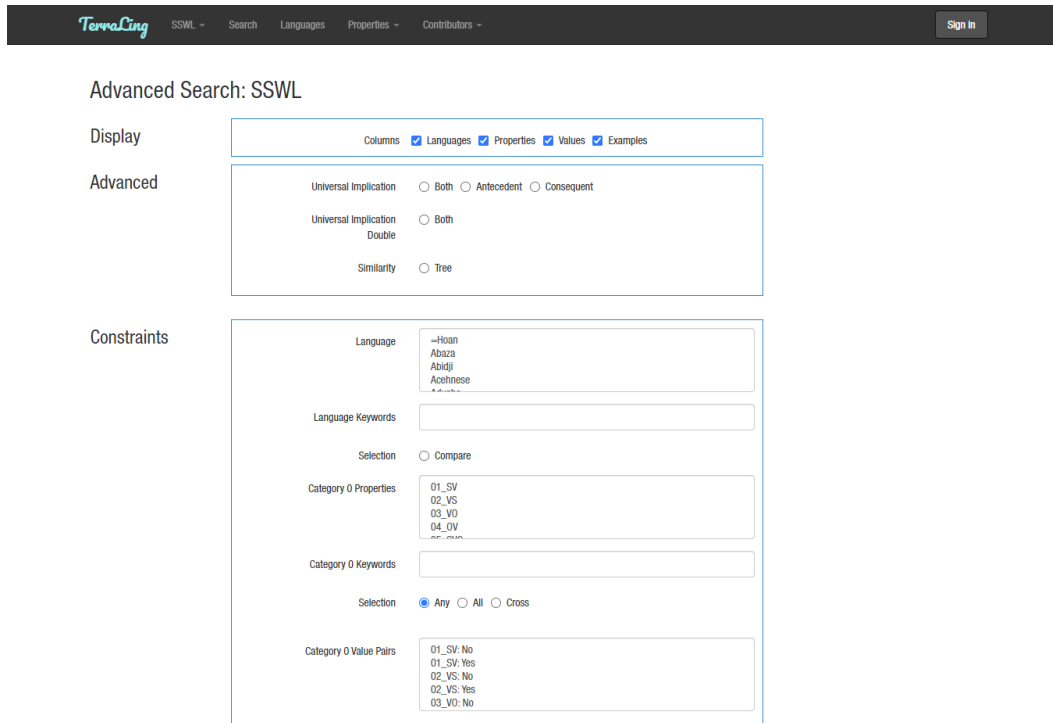
The screenshot shows the TerraLing web interface. At the top is a dark navigation bar with the TerraLing logo, links for SSWL, Search, Languages, Properties, and Contributors, and a Sign In button. The main content area displays 'Example\_139 Example'. Below this is a '← Back' button and a link 'Related to : English'. A table with two columns is shown, containing linguistic data for the example. At the bottom, there is a copyright notice for 2013 terraling.com, Version: 2.0.0-5, and contact information.

Words :	He is tall enough
Gloss :	He is tall enough
Translation :	He is tall enough

© 2013 terraling.com, Version: 2.0.0-5  
10 Washington Place, New York, NY 10003 . support @ terraling.com . All Rights Reserved

**Figure 3.** An example for a property in the “SSWL” group, for the language “English”

Once there is sufficient significant data for a group, linguists can search through this data using the search interface (Fig. 4). There are several types of search: filtering, comparing, crossing, implication, and similarity. These are described in Section 2.



**Terraling** SSWL Search Languages Properties Contributors Sign In

Advanced Search: SSWL

Display Columns ☒ Languages ☒ Properties ☒ Values ☒ Examples

Advanced

Universal Implication ☐ Both ☐ Antecedent ☐ Consequent

Universal Implication Double ☐ Both

Similarity ☐ Tree

Constraints

Language   
Abaza  
Abidji  
Acehnese

Language Keywords

Selection ☐ Compare

Category 0 Properties   
02\_VS  
03\_YO  
04\_OV

Category 0 Keywords

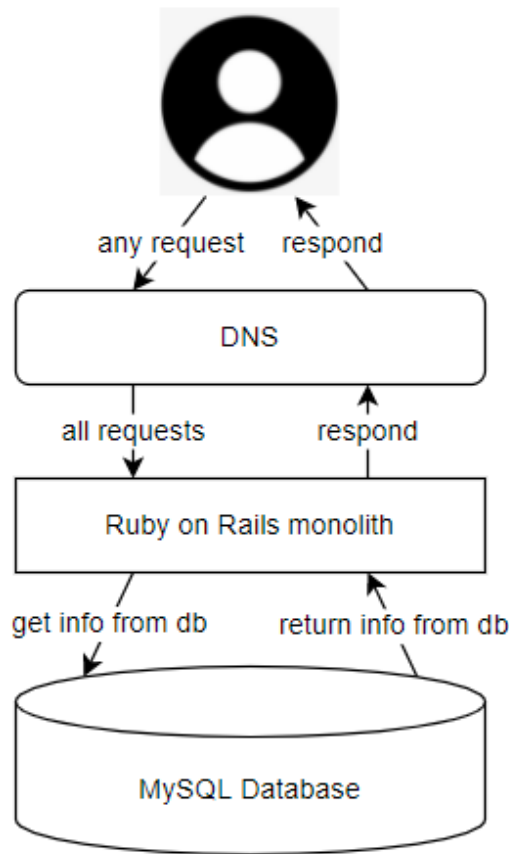
Selection ☒ Any ☐ All ☐ Cross

Category 0 Value Pairs   
01\_SV: Yes  
02\_VS: No  
02\_VS: Yes  
03\_YO: No

**Figure 4.** Terraling’s search interface

For more information on Terraling’s linguistic use, see [Koopman’s handbook](#).

From a technical perspective, Terraling is a Ruby on Rails monolith that uses the Model-View-Controller (MVC) architecture. All code is stored in one repository, and the entire website is deployed to a single server. All routing, database access, data transformation, data presentation, and user authentication are managed by a single multi-threaded process (Fig. 5).



**Figure 5.** The existing monolithic architecture

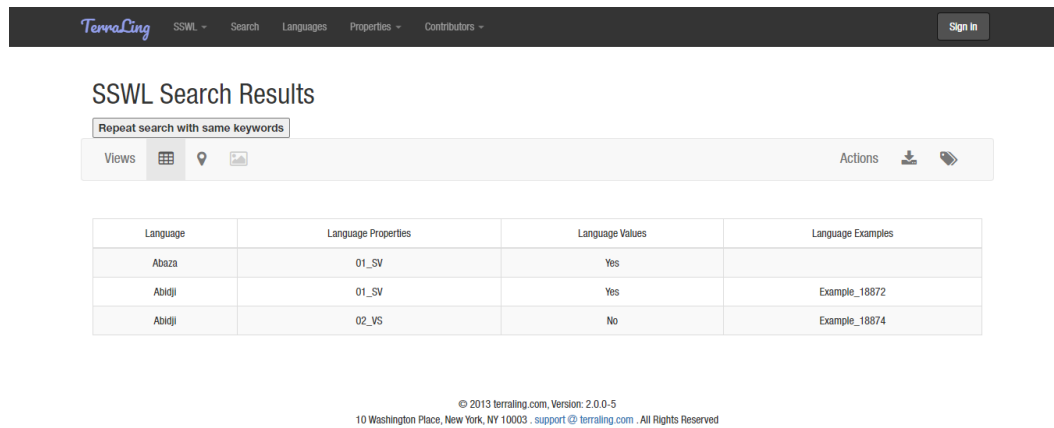
Data is rendered server-side, and pure HTML is sent to the browser. There was recently work done on allowing the monolith to function as a headless API, serving JSON when the client requested it and the standard HTML otherwise. This is discussed further in Section 3.

### 3: Functionality addressed

The focus of this paper is the search interface. As mentioned, there are five main types of searches available to linguists.

#### Filter

The filter function allows the user to select a subset of languages and display a selection of property values for these languages.



TerraLing SSWL Search Results

Repeat search with same keywords

Views Actions

Language	Language Properties	Language Values	Language Examples
Abaza	01_SV	Yes	
Abidji	01_SV	Yes	Example_18872
Abidji	02_VS	No	Example_18874

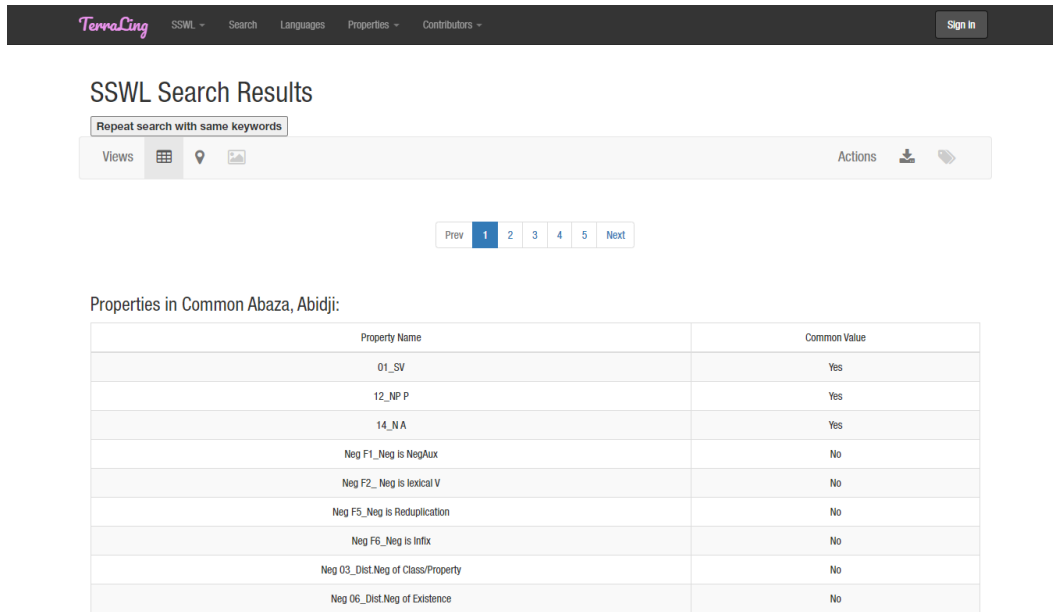
© 2013 terraling.com, Version: 2.0.0-5  
10 Washington Place, New York, NY 10003. support @ terraling.com. All Rights Reserved

**Figure 6.** The results of a filter search in “SSWL” on languages “Abaza” and “Abidji” and properties “01\_SV” and “02\_VS”

## Compare






The compare function allows the user to select multiple languages and view which property values are common for all selected languages, and which are distinct.





SSWL Search Results

Repeat search with same keywords

Views    Actions  

Prev 1 2 3 4 5 Next

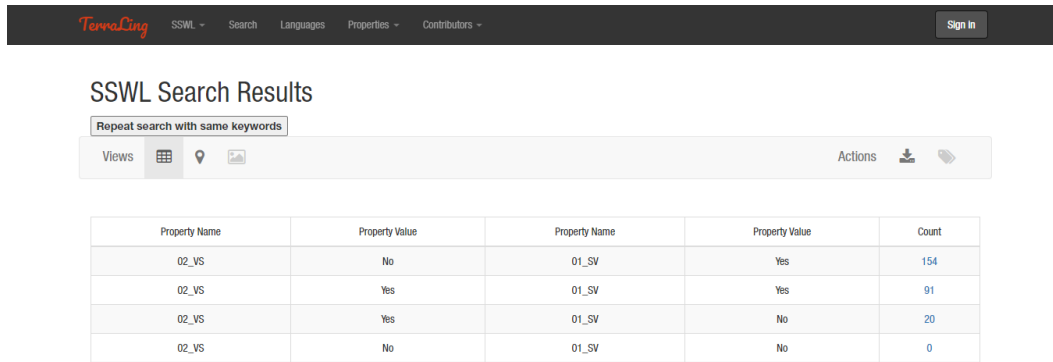
Properties in Common Abaza, Abidji:

Property Name	Common Value
01_SW	Yes
12_NP P	Yes
14_NA	Yes
Neg F1_Neg is NegAux	No
Neg F2_Neg is lexical V	No
Neg F5_Neg is Reduplication	No
Neg F6_Neg is Infix	No
Neg 03_Dist.Neg of Class/Property	No
Neg 06_Dist.Neg of Existence	No

**Figure 7.** The results of a compare search in “SSWL” on languages “Abaza” and “Abidji”

## Cross

The cross function allows the user to select multiple properties and conduct a cross product of all the values of the selected properties. That is, for every combination of possible property values, the cross function finds which and how many languages have that combination of values.



The screenshot shows the TerraLing SSWL Search Results page. At the top, there's a navigation bar with the TerraLing logo, links for SSWL, Search, Languages, Properties, and Contributors, and a Sign In button. Below the navigation bar, the page title is "SSWL Search Results". There's a button to "Repeat search with same keywords". Below that, there's a "Views" section with a grid icon selected, and an "Actions" section with download and share icons. The main content is a table with 5 columns: Property Name, Property Value, Property Name, Property Value, and Count. The table contains 4 rows of data.

Property Name	Property Value	Property Name	Property Value	Count
02_VS	No	01_SV	Yes	154
02_VS	Yes	01_SV	Yes	91
02_VS	Yes	01_SV	No	20
02_VS	No	01_SV	No	0

**Figure 8.** The results of a cross search in “SSWL” on properties “01\_SV” and “02\_VS”

## Implication

The implication function allows the user to select a property and a value for that property, and shows all other property values that either imply or are implied by that property.

There are three types of implications: antecedent, consequent, and both. Formally, given a property value of  $p$ , the antecedent implication function shows all property values  $Q$  defined by:

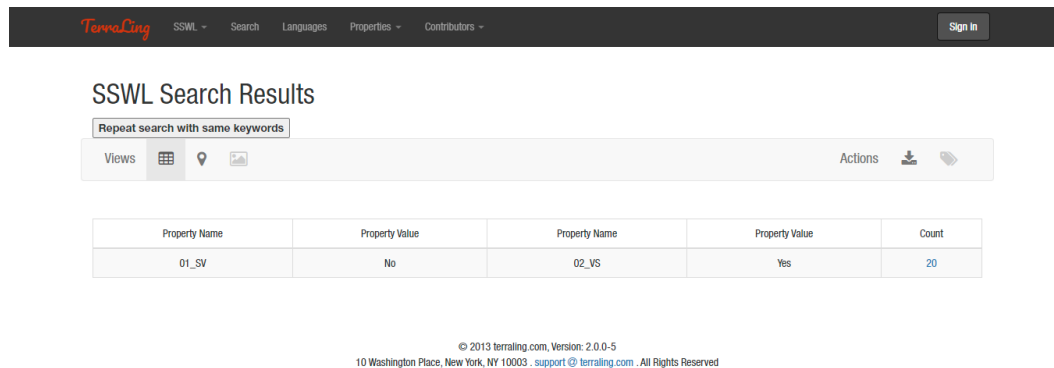
$$Q = \{ q \mid p \in L \Rightarrow q \in L \text{ for all languages } L \}$$

The consequent implication function shows the set  $R$  defined by:

$$R = \{ r \mid r \in L \Rightarrow p \in L \text{ for all languages } L \}$$

The both implication function shows the set  $S$  defined by:

$$S = \{ s \mid s \in L \Leftrightarrow p \in L \text{ for all languages } L \}$$



**Figure 9.** The results of an antecedent implication search in “SSWL” on property “01\_SV” with value “No”

## Similarity

The similarity function allows the user to select multiple languages and displays a phylogenetic tree of those languages grouping them together based on how many identical property values they share.



**Figure 10.** The results of an similarity search in “SSWL” on all languages

### 3: The problem

There are several issues with the existing search interface. Since many linguists depend on being able to explore and transform their data for their research, it is important that users are able to easily and quickly obtain results. This is impacted by five factors: maintainability, availability, architecture, interface, and performance. On all counts, the existing implementation offers room for improvement.

#### Maintainability

As previously mentioned, the Terraling application is a Ruby on Rails monolith. In addition, all code exists on a single repo. This poses several challenges.

First, use of Ruby is decreasing in favor of JavaScript, Python, and Golang, amongst others. This makes it harder to find developers well-versed in Ruby.

Second, the MVC architecture is also losing popularity, in favor of a microservice-based architecture that uses several orchestrated services. This makes it harder to find developers who understand the architecture, and Ruby on Rails in particular.

Third, and perhaps the cause for the change above, is that a monolithic architecture in a single repo means that contributions necessarily affect the entire application and the entire codebase. This requires more effort to be allocated towards reviewing contributions to ensure they do not adversely affect some other part of the application. It also means contributors must have a deeper knowledge of the codebase than the part they wish to contribute to.

Fourth and finally, an MVC architecture makes it difficult to separate concerns effectively. Canonically, presentation logic should remain in views and business logic should remain in controllers. Models should only contain class methods and properties. Since the entire application is a monolith, logic frequently gets mixed up between sections, making debugging and adding features extremely difficult.

## Availability

The existing implementation already has good availability. However, there is always room for improvement.

## Architecture

Since the MVC architecture allows only for sending HTML from the server to the client, it is difficult to build new interfaces. Either the new interface must parse the data from the HTML, or some change must be made to the monolith to allow the usage of an open standard data format such as JSON.

The current implementation has already had the necessary changes made such that some core endpoints are exposed via JSON API. While not strictly necessary for this project, it made it significantly easier.

Still, the search endpoints in particular return a complicated and unintuitive object, and there is much room for improvement.

## Interface

The existing user interface is poor. Even experienced linguists often struggle with finding the correct functions. There is minimal separation of concerns.

While the interface was conceived with the idea of offering every combination of filtering and advanced functions to the end user, most of these combinations are never used. The result is a confusing and unintuitive interface with poor usability.

## Performance

Another consequence of the MVC architecture is that database access is often abstracted away by the framework. As such, poorly constructed calls to the models can cause multiple repeated queries or missed opportunities for optimization.

This can (and does) result in disastrously bad performance. Raw numbers will be displayed in Section 5, but the current

implementation forces users to wait for minutes in order to view only tens or hundreds of results.

## 4: The solution

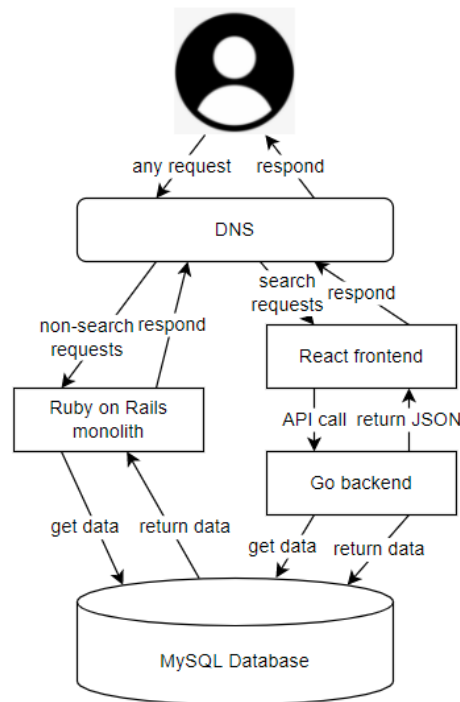
To solve the problems detailed above, we extracted any search related logic into two new services: a frontend service for creating searches and displaying their results, and a backend service for accessing the database and transforming the data in response to search queries (Fig. 11).

The frontend service is written in React, which is a popular JavaScript framework that enjoys significant community and developer support. It uses minimal dependencies, with a new design language that offers better usability.

The backend service is written in Golang, which is a popular C-like language that has significant community support, extensive tooling, and extremely fast performance. Most required packages are already built into Golang, which means that there are no external dependencies to go out of date.

Additionally, using Golang allows for more targeted optimization of the SQL. By manually controlling database access, we can collapse multiple queries into a larger, more performant query. This allows extremely large searches to perform almost as well as small searches.

The two services communicate exclusively via JSON, and are deployed separately but on the same server. This had the added benefit of allowing maintainers to gracefully degrade the existing service; there were no interruptions of service. In addition, we also were able to perform A/B tests; i.e. selectively turning on the new interface and backend for specific users. This allowed for a completely seamless testing period and rollout.



**Figure 11.** The new architecture with the search microservices extracted.

## 5: Results

The choice of languages, interface, and architecture were designed with the five key components in mind. They represent improvements over the existing architecture in every way.

### Maintainability

The design for the services was chosen to ensure that developers could easily and quickly contribute to the application. It is easier to find developers who understand React and Golang, and contributions should be overall of higher quality.

Additionally, the new services exist on two separate repositories. Changes to each can be made without affecting the rest of the application. In addition, developers contributing to each service do not need to have knowledge outside of the specific domain they are



working on. The only consideration is that the interface remains the same (or is versioned, such that there are no breaking changes).

Further research can be done to ensure that the interface between the two is more strictly defined in code. For example, [protocol buffers](#) offer a language-agnostic way to define schema in code and reduce bundle size at the same time.

## Availability

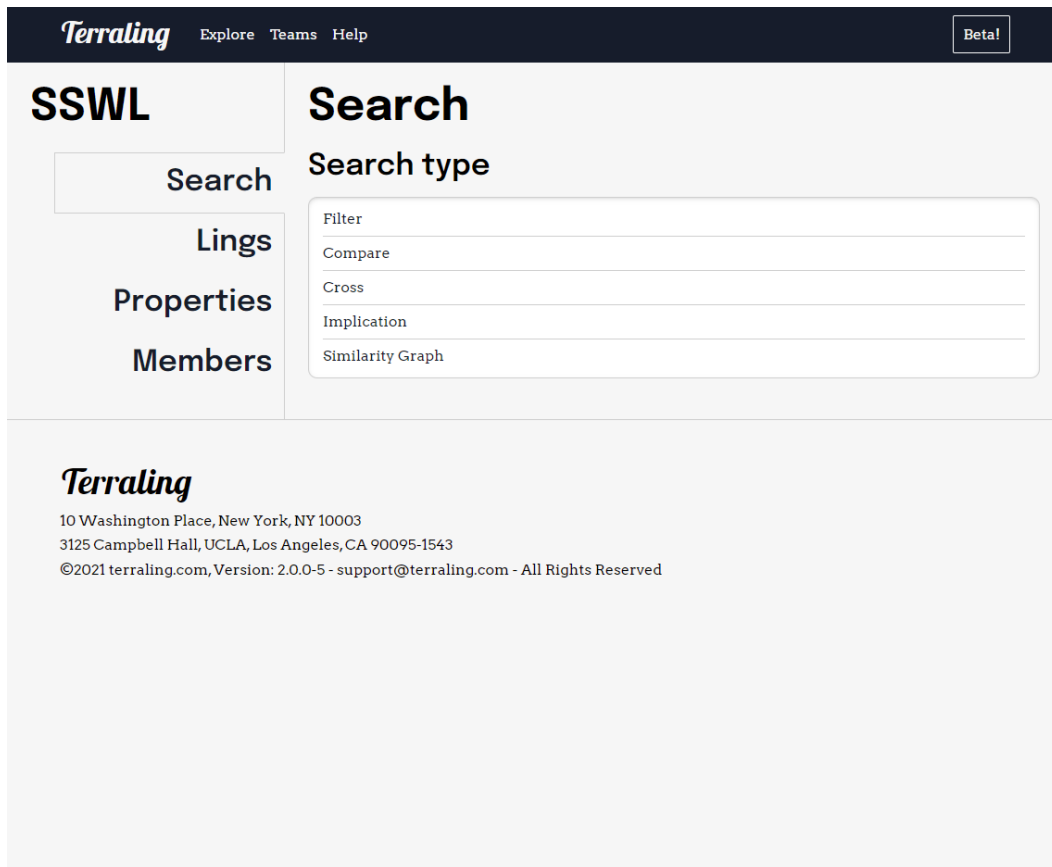
While availability was already excellent on the existing implementation, the use of a microservice architecture means that system-wide outages are significantly less frequent. If one service fails, the rest of the application can continue to function.

## Architecture

As well as being able to define the schema in code, the format of the JSON being transferred across the two services is significantly more intuitive. This has the benefit of allowing other developers to more easily develop a way to interface with the search should the need arise, which is important for an open-source application such as Terraling.

## Interface

The new interface is also more intuitive, at the expense of culling certain unused combinations from the search interface. Linguists choose the type of search they are creating (Fig. 12) and are dynamically presented with the appropriate selection box when they do (Fig. 13).



**Figure 12.** Selection box prompting the user to select a search type

The screenshot shows the Terraling web application. The top navigation bar includes the logo 'Terraling', links for 'Explore', 'Teams', and 'Help', and a 'Beta!' badge. On the left, a sidebar contains the main heading 'SSWL' and a list of categories: 'Search', 'Lings', 'Properties', and 'Members'. The 'Search' category is selected. The main content area is titled 'Search' and contains two sections: 'Search type' and 'Search target'. The 'Search type' section has a dropdown menu with 'Filter' selected, and other options: 'Compare', 'Cross', 'Implication', and 'Similarity Graph'. The 'Search target' section has a dropdown menu with 'Filter by Ling' selected, and another option: 'Filter by Ling property'. Below these sections is a 'Lings' section with a 'Reset' link. It contains a list of language names: '=Hoan', 'Abaza', 'Abidji', 'Acehnese', 'Adyghe', 'Afrikaans', and 'Anni (Rini)'. Each name is followed by a horizontal line, suggesting a table structure.

Search type
Filter
Compare
Cross
Implication
Similarity Graph

Search target
Filter by Ling
Filter by Ling property

Lings
=Hoan
Abaza
Abidji
Acehnese
Adyghe
Afrikaans
Anni (Rini)

**Figure 13.** More selection boxes dynamically appearing as the user selects a search type and target

In addition, results are more easily viewable, with sticky tables and smart headings that react to the search. For example, filtering by Language shows Language headings (Fig. 14), whereas filtering by Language property shows Language property headings (Fig. 15).

**Terraling**

Explore Teams Help

Beta!

**SSWL**

Search

Lings

Properties

Members

**Filtering by Lings**

**Results**

[Abaza](#)

Property	Value
<a href="#">01_SV</a>	Yes

[Abidji](#)

Property	Value
<a href="#">01_SV</a>	Yes
<a href="#">02_VS</a>	No

**Terraling**

10 Washington Place, New York, NY 10003  
3125 Campbell Hall, UCLA, Los Angeles, CA 90095-1543  
©2021 terraling.com, Version: 2.0.0-5 - support@terraling.com - All Rights Reserved

**Figure 14.** Filtering by Language shows Languages as headings

**Terraling** Explore Teams Help Beta!

**SSWL**

Search

Lings

Properties

Members

## Filtering by Lings properties

### Results

01\_SV

Ling	Value
<a href="#">Abaza</a>	Yes
<a href="#">Abidji</a>	Yes

02\_VS

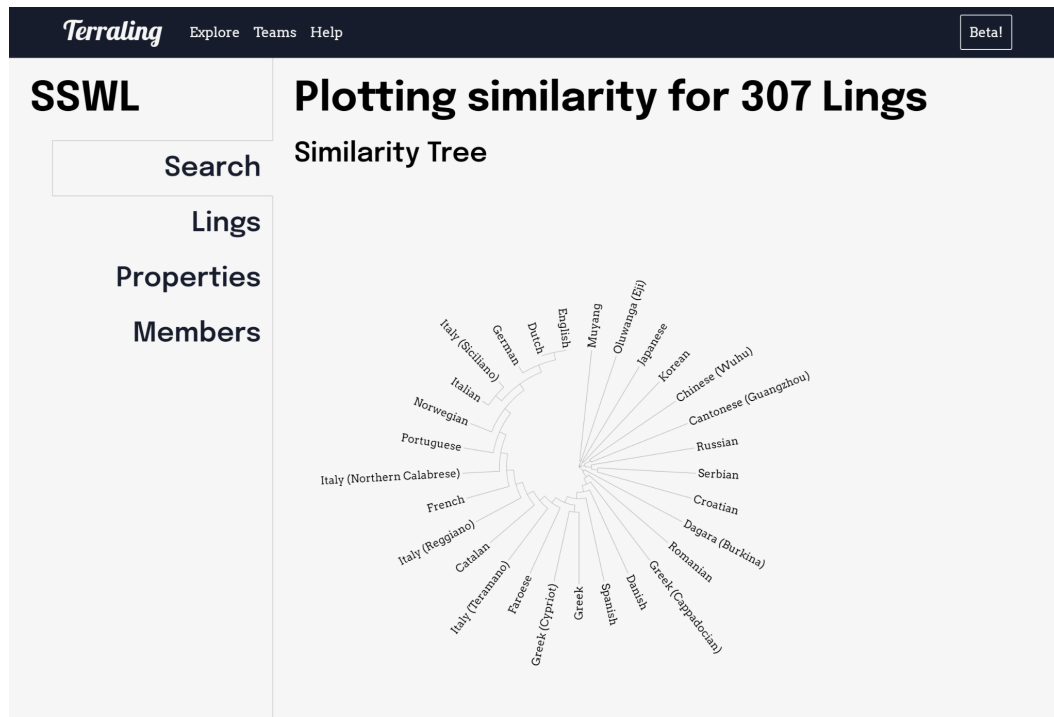
Ling	Value
<a href="#">Abidji</a>	No

**Terraling**

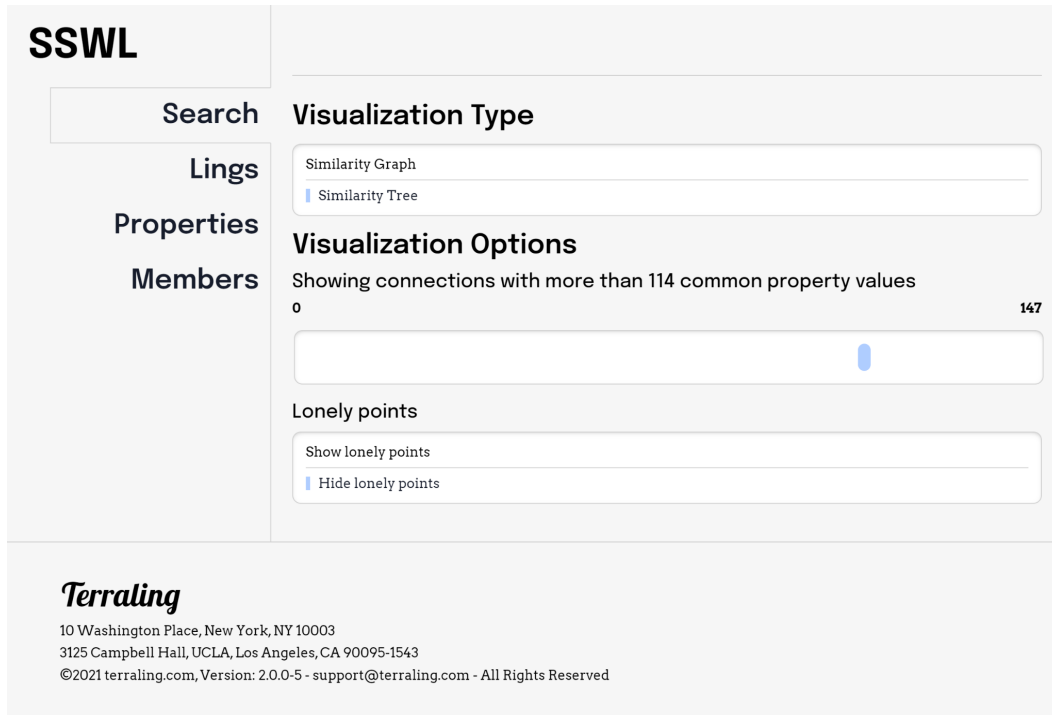
10 Washington Place, New York, NY 10003  
3125 Campbell Hall, UCLA, Los Angeles, CA 90095-1543  
©2021 terraling.com, Version: 2.0.0-5 - support@terraling.com - All Rights Reserved

**Figure 15.** Filtering by Language property shows properties as headings

For more complex visualizations, using React allows for more modern browser support. For example, we can use D3 to display a phylogenetic tree of languages (Fig. 16) with customizable parameters and highlighting (Fig. 17).



**Figure 16.** A phylogenetic tree of a subset of languages



**Figure 17.** Customizable parameters for the phylogenetic visualization

The full interface is viewable at [this address](#).

## Performance

To test performance, five queries were chosen and performed 10 times for each implementation. They were then averaged. To ensure a fair test, initial benchmarks were performed locally on development servers, with a local database, and on the same data.

The results are as follows:

Query	Old (s)	New (s)	Speedup (x)
Filter (all)	37.071275	0.486178	76.2
Compare (2)	1.143203	0.050646	22.6
Compare (6)	1.916712	0.073384	26.1

<b>Cross (2)</b>	4.150662	0.067625	<b>61.4</b>
<b>Cross (6)</b>	7.998947	0.093085	<b>85.9</b>

**Figure 18.** Performance improvement results across five different searches

Clearly, the new implementation represents a significant improvement in performance. There is likely still room for more optimization, but it is clear that the use of Golang and the more granular database access it afforded allowed for faster searches.

For the sake of interest, the tests were performed once more on the production server, with compiled HTML pages and binaries (i.e. optimal conditions). The results are even more striking:

<b>Query</b>	<b>Old (s)</b>	<b>New (s)</b>	<b>Speedup (x)</b>
<b>Filter (all)</b>	34.34378	0.050624	<b>678</b>
<b>Compare (2)</b>	1.121114	0.002990	<b>375</b>
<b>Compare (6)</b>	2.136508	0.004673	<b>457</b>
<b>Cross (2)</b>	3.910758	0.005830	<b>671</b>
<b>Cross (6)</b>	6.459189	0.009677	<b>667</b>

**Figure 19.** Performance improvement results across five different searches in optimal (production) conditions

The new implementations are clearly significantly faster, over 500 times so in most cases.

## Queries

The queries were chosen at random, and were not cherry picked for performance. For reference, the queries performed were:

- **Filter (all)** - A filter query on SSWL selecting all languages
- **Compare (2)** - A compare query on SSWL selecting two languages: Abaza and Abidji



- **Compare (6)** - A compare query on SSWL selecting six languages: Abaza, Abidji, Bosnian, Danish, Dutch, English
- **Cross (2)** - A cross query on SSWL selecting two properties: 01\_SV and 11\_P NP
- **Cross (6)** - A cross query on SSWL selecting six properties: 01\_SV, 03\_VO, 11\_P NP, 13\_A N, 15\_Num N, and 17\_Dem N

## 6: Future work

Some future work remains to be done. There are still several optimizations that can be made to the backend service to improve performance. These include merging queries and reducing memory usage. Since the SQL is now manually controlled, queries can be tweaked and tested to find the most performant solution.

In addition, the increased maintainability also enables future work on new features. For example, more diverse visualizations for the similarity feature can be created. Different types of searches can also be built.

The frontend will also be improved. Since the new interface is drastically different, user feedback will be collected by first directing key users to the staging servers, then by performing A/B tests on production. The feedback collected here will allow for minor but important improvements to usability and user experience.

## References

- [1] Koopman, Hilda, and Cristina Guardiano. Managing data in TerraLing, a large-scale cross-linguistic database of morphological, syntactic, and semantic patterns. "https://linguistics.ucla.edu/wp-content/uploads/2020/08/KoopmanGuardiano-handbookarticle-revised08172020-.pdf."