# Developing a Suite of Mobile Applications for Collaborative Language Documentation

**Mat Bettinson**[1] **and Steven Bird**[2,3]
[1]Department of Linguistics, University of Melbourne
[2]Department of Computing and Information Systems, University of Melbourne
[3]International Computer Science Institute, University of California Berkeley

## Abstract

Mobile web technologies offer new prospects for developing an integrated suite of language documentation software. Much of this software will operate on devices owned by speakers of endangered languages. We report on a series of prototype applications that support a range of documentary activities. We present ongoing work to design an architecture that involves reusable components that share a common storage model and application programming interface. We believe this approach will open the way for a suite of mobile apps, each having a specific purpose and audience, and each enhancing the quality and quantity of documentary products in different ways.

## 1 Introduction

Documenting a language calls for a substantial collection of transcribed audio to preserve oral literature, epic narratives, procedural knowledge, traditional songs, and so on. Carrying out this program at scale depends on effective collaboration with speech communities. This collaboration spans the documentary workflow, starting with raising awareness and recruiting participants, followed by the core work of recording, transcribing and interpreting linguistic events, and finally the processes of preservation and access. The collaboration goes beyond individual workflow tasks to collaborative management, whereby the language community – as the producers and consumers of the material – help to shape the work.

Our vision is for a suite of applications supporting a variety of workflows, and contributing to a common store of language documentation, designed to support the kinds of remote and long-distance collaborations that arise when working with endangered languages. Emerging web technologies and proliferating mobile devices open the door to this future. Creating reusable components and a common application programming interface (API) will accelerate the process.

Instead of seeking consensus about a single documentary workflow enshrined in monolithic software, we envisage diverse workflows supported by multiple applications, built and rebuilt from shared components as local requirements evolve. Linguists should be able to customise an app by tinkering with a top level page, to make changes like moving a consent process from before to after a recording, or replacing the language selection component with a fixed choice, or requiring not just a portrait of the speaker but a second landscape view of the context in which a recording was made. However, rather than design the whole infrastructure, we have taken a bottom-up approach, developing components and specialised apps that help to clarify the requirements of the API.

This paper is organised as follows. First, we discuss the state of the art in section 2. Then in section 3 we describe the evolution of our thinking through a series of prototype applications. Next, in section 4, we discuss the data requirements for apps and their constituent components based on our prototyping experience, and suggest a common interface of components. Finally, in section 5 we describe further work in support of our goal of establishing a suite of interconnected language documentation applications and lay out our vision for a common language data API.

## 2 Mobile Apps for Language Documentation

Digital tools are widely used in documentary workflows, but they often require specialised training and are platform specific. In recent years there has been growing awareness of the importance of collaborating with language maintenance
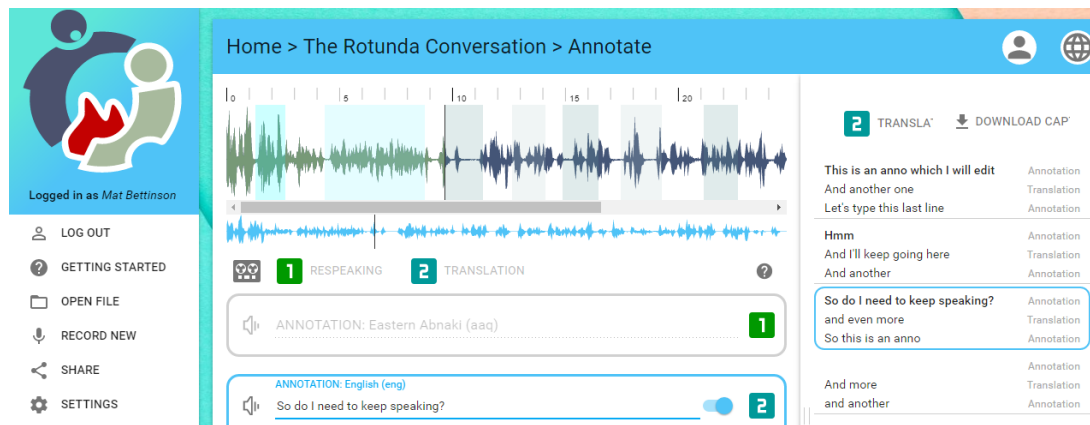
Figure 1: Aikuma-NG beta transcription web app, available from the Chrome Web Store

and revitalisation groups, particularly as we look to sustain activity and relationships beyond the 3-5 year window of a sponsored research project (Austin, 2010). The shift to mobile technologies is enabling this collaboration, and may ultimately transform the practice of documentary linguistics (Chatzimilioudis et al., 2012; Birch et al., 2013).

To date, dictionary and flashcard apps for language learning have been the most popular. For example, the suite of First Voices apps for iPhone in Canada, and the 'Ma' series of dictionary apps in Australia and the Pacific including Ma Iwaidja, Ma Gamilaraay and Ma Bena Bena (Carew et al., 2015). Taiwan's Council of Indigenous People's e-dictionary includes 16 Formosan languages, via a mobile-accessible website (Taiwan Indigenous Council, 2016). Apps have been used to conduct experiments on dialect variation, capturing linguistic judgements together with the location (Goldman et al., 2014; Leemann et al., 2016). The Android app Aikuma, a precursor of the work reported here, allows users to collect and translate spoken narrative (Bird et al., 2014).

Alongside these individual web apps there are web application suites for language documentation. LDC webann is an online annotation framework supporting remote collaboration (Wright et al., 2012). LingSync supports collaborative language documentation, and has been popular in North American linguistic fieldwork training (Cathcart et al., 2012). CAMOMILE is a framework for multilingual annotation, not specifically for linguistic research (Poignant et al., 2016).

These apps fall into two categories according to their audience and purpose: research apps for language documentation and 'community' apps for language development. The developer profile and funding sources are different, with research apps generally developed in academia, and community apps developed by commercial developers. This situation points to an opportunity for collaboration in the development of language apps that appeal to a broader audience.

## 3 Prototype Apps

This section reports on the development of three apps over the course of 2016. These apps represent an evolving understanding of methods to achieve modularisation through reusable components. One component in particular, for language selection, is required for all apps. We discuss this component in the context of each app to shed light on some options concerning web technologies and data models.

### 3.1 Transcription: Aikuma-NG

Aikuma-NG was developed to assess the feasibility of building mobile software using web technologies and delivering a similar feature set to desktop software (Figure 1). The app's audience is people who wish to transcribe speech, particularly community members and laypersons. The expected output is standard srt or vtt files for captioning YouTube videos. Aikuma-NG is a community app that requires no internet access post install.

Aikuma-NG incorporates basic metadata management and the ability to perform oral respeaking and translation, following the example of Say-More (Hatton, 2013).

Aikuma-NG's main feature is multi-tier transcription, making use of the additional audio from any respeaking or translation activities, in order

to create transcriptions of a source and its translations. The app exports to common video subtitling formats as well as ELAN, and has been localised into English, Korean and Chinese.

Aikuma-NG was built as a Chrome App, because this provided a means to deliver a desktop-like experience. Chrome apps open full screen with no URL bars and have unlimited local data storage. We adopted the technology stack based on the JavaScript Model-view-controller (MVC) framework *Angular* and the UI framework *Angular Material*. The capacity for Angular Material to deliver a high quality UI was key factor in this choice. The Wavesurfer waveform visualisation package was adopted to handle visualisation and time series-based data structures (WaveSurfer, 2017). We found limited support in the JavaScript ecosystem for multimedia and local storage, requiring us to gain a deep knowledge of emerging web standards that had yet to be widely adopted in third-party software. Key challenges arose from acquiring experience of asynchronous programming, poor support for modularisation in the ES5 JavaScript and, in particular, the weak inter-component communication model of Angular 1.

### 3.1.1 The language selector

MVC frameworks such as Angular allow us to specify an app view with a template containing custom HTML tags. Figure 2 depicts four components in Aikuma-NG. An audio file visualiser/player based on Wavesurfer, and three selector components based on the touch-friendly Material Design UI to select languages, people and customisable tags. These components are used in a view template with markup as follows:

```
<ng-player ...></ng-player>
<ng-language-selector ...></ng-language-selector>
<ng-person-selector ...></ng-person-selector>
<ng-tag-selector ...></ng-tag-selector>
```

We chose the Angular and Angular Material-based stack recognising that the well-documented UI component examples represented a professional implementation of a UI, one that was recognisable by millions of users of Google's web tools and Android platform. Material Design's "chip" UI component was ideally suited to display a list of arbitrary categories or labels such as languages and tags. Angular Material's documentation included an implementation binding chips with text input auto-complete. Thiw was a good fit to facil-
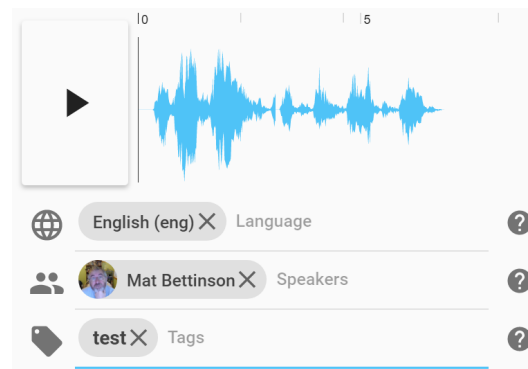


Figure 2: Aikuma-NG view from a template of sub-components

itate rapid selection from thousands of language names.

Experience from field testing showed that we must also allow users to create arbitrary names for their language. This is necessary to account for their language being unknown, or their preference to write the name of a language in another language such as the dominant language in the region. The consequence is that we must allow for auto complete over customised entries as well as ISO693 categorised labels.

Typical data flow involves passing an array of pre-populated languages to the component, if we are restoring a previous UI. The component accesses a data service to retrieve a list of ISO693 languages and custom languages for the auto-complete. Two-way data binding returned the data to a parent component.

### 3.1.2 Key findings

Web technologies can be used to build a full featured *desktop* app, as demonstrated by the ongoing popularity of Aikuma-NG. The Chrome App platform worked well for this application, but just months after the release, Google announced their intent to retire Chrome Apps. We believe Aikuma-NG can be implemented effectively as a progressive web app. Rapid iteratation of UI designs in the field is particularly valuable, allowing us to find the best approach for our target audience.
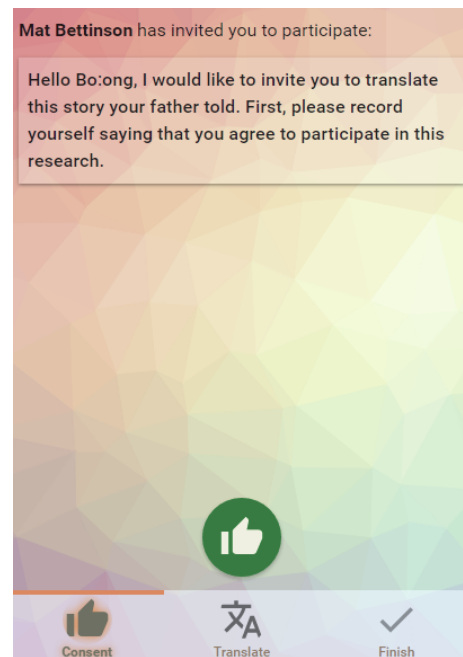
### 3.2 AikumaLink: Task management

Aikuma-Link (Figure 3) is an online only *research* app that allows the researcher audience to recruit a remote participant to perform an activity of the linguist's choosing.

The app was motivated by the observation that researchers often return from the field with data

(a) Task creation

(b) Receiving a task

Figure 3: Aikuma-Link: Linguists define documentary tasks with supporting materials, the task is dispatched to participants to perform with a mobile app

that requires further interaction with native speakers to become useful. The same app offers desktop and mobile views and employed a real-time back end as a service (BaaS) as a common data model. The app was intended to investigate the ability of web apps to virtually eliminate the 'on-boarding' cost of recruiting participants to use research software.

Aikuma-Link is based on a process where the researcher first defines a task such as translating or respeaking audio recordings. From this task, the app generates a URL link which the researcher sends to the remote participant, typically by social media. Clicking the link on a phone launches the Aikuma-Link mobile app, which invites their participation and allows them to perform the prescribed tasks. The resulting data is then returned to the linguist. A stretch design goal is to facilitate crowdsourced experiments by crafting a single link which can invite any number of participants to perform the same activity.

### 3.2.1 The language selector

Material Design was originally chosen specifically because it was touch-friendly for mobile devices. Nevertheless, components based on text input (and therefore virtual keyboards) and long lists of autocomplete choices represent UI challenges of a different order to desktop. Where the component is used for the desktop browser linguist view, it can be used inline with many others because desktop users have large displays and scrolling in acceptable. For mobile, a component is better realised as one step in a wizard-like approach of multiple actions.

We were keen to improve upon inter-component communication and migrating to the commercial real-time backend Firebase turned out to be a great boon. We experimented with an approach where components accepted Firebase objects as arguments, effectively passing the component the means to read and write data to a specific database schema defined by the parent component. The language selector component was modified to read the current state and bind UI elements directly to the database.

In the following snippet, the language selector controller is adding an object specifying a language id directly to a Firebase array provided via the Firebase SDK. The 'then' syntax is a JavaScript asynchronous 'promise' pattern where the following will be evaluated when the promise to save the data is complete.

```
ctrl.selectedLanguagesFb.\$add({id: langId})
.then(function(ref) {
    chip.saved = true
})
```
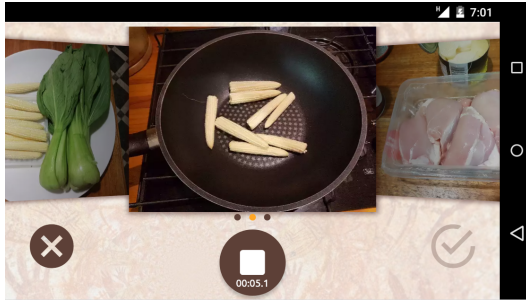
Figure 4: Zahwa Procedural Discourse App prototype: Users narrate while swiping through photos

The UI "chip" saved property is set to true when the Firebase database write is complete. The saved property is used in the component template to apply a CSS class with the result that the chip colour changes to provide a visual indication that a save is complete.

The obvious issue with this component communication pattern is that it locks in a particular data system, or third-party vendor SDK in this case. Ideally a component should accept and return stand-alone data structures and leave parent components to decide how to retrieve and save such data. This was a general problem with the the current generation of MVC frameworks (Angular 1.x). We describe this pattern in the discussion of the Zahwa app to follow.

### 3.2.2 Key findings

Aikuma-Link's showed that, with care, web technologies could deliver a performant native app-like experience on relatively low-end phones. The link-share method of onboarding is promising and opens up a number of possibilities for crowd-sourced research. We found Firebase to be an excellent solution for rapidly prototyping mobile apps with collaborative data. Accounting for different sized displays, orientation changes and virtual on-screen keyboards is a significant challenge for mobile software development.

### 3.3 Zahwa: Procedural discourse

Zahwa is a *community* app (Figure 4) that has users take a series of photos and short videos from their device, then swipe through them while recording a voice-over. The app as conceived and designed with cooking recipes in mind but it is broadly applicable to documenting of any procedural discourse. Users who view the recipe, or instructions for making craft, etc, are able to inter-

act with recipes, providing their own translation or reusing the media set for their own version.

Zahwa is a fully-featured mobile app built with web technologies (it is a progressive web app) and with robust offline capabilities. We first built a prototype out of a newer generation of our former framework but later adopted the Ionic 2 mobile framework, and with it Angular 2 and the Typescript variant of JavaScript. Migrating to this framework meant that all UI components would need to migrate from Angular Material to Ionic's native UI. Ionic offers both an iOS-like UI and Material Design (Android).
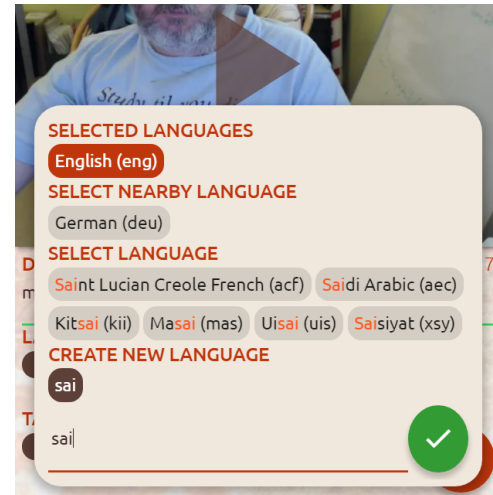
Broadly speaking, the reasons for adopting of Ionic 2 were threefold. First, Angular 2 brought substantial improvements in the methods to define components and views (pages), and inter-component communication. This virtually eliminated a slew of performance and reliability issues with Angular 1. Second, we realised that when building a full-scale mobile app, we are less interested in building common components for all mobile apps where we ought to concentrate on language documentation specific components. Finally, we found that Ionic had already demonstrated, through their user community, a realisation of our own goal to expand the base of potential app developers.

Ultimately, adopting Ionic allowed us to be more ambitious, and focus on technology components and user interfaces specific to our domain without needing to reinvent functionality that is common across mobile apps. This win turned out to be fortuitous because there was a substantial engineering challenge ahead. We had previously built offline-only and online-only apps, but had yet to combine the two, to craft a collaborative app that would use a network where available, while allowing for meaningful app usage offline.

After determining that there are no good off-the-shelf solutions, we retrofitted Zahwa with: offline storage based on PouchDB, a new service to synchronise local storage with Firebase, and a caching service to support offline behavior. These allow Zahwa to provide offline users with features to find salient recipes, e.g. geographically nearby, or use search such as languages and tags. The user can indicate that they would like to download a recipe when they have a connection. Creating new recipes and translating cached recipes can be performed offline. The work is sychronised when a

(a) Searching by Language      (b) Editing Languages with Popover

Figure 5: Zahwa: The language selector component used in different contexts. A simplfied in-line search mode and a more comprehensive popover mode to support the full range of the language selector's enhanced capabilities.

network reappears. Users may specify to limit this to WiFi rather than cellular data.

Zahwa has been a helpful vehicle to prototype a UI based on progressive enhancement of the data around a core activity. Given an existing recipe, community members may be motivated to translate it into another language, or record a new version of the same recipe by reusing the media. They may be motivated to tag the images of a recipe, contributing to an evolving lexicon. These enhancements may not be attractive to all users, but one could explore gamification as a way to encourage users to perform such tasks.

### 3.3.1 The language selector

Zahwa offers a more comprehensive demonstration of the ways in which a component may be utilised in different contexts within the same app (Figure 5). Users can discover recipes by language, even if offline, and mark the recipes for retrieval. The language selector in these cases simply presents chips to touch to select. With intelligent context, most users do not need to use the keyboard at all.

Mobile UIs should tend towards the minimal until the user has indicated they wish to engage in further detail. In Zahwa, the language selector offers a minimal list of languages but upon user interaction the component launches a pop up modal UI that is able to utilise most of the displays real-estate for the task at hand.

With there now being separate storage systems at play, child components ought now to act on pure data and let parent components load or save. Angular 2's component communication paradigm urges one-way binding for data inputs and an event-driven schema for data output as seen in this example from Zahwa:

```
<lang-edit [languages]="recipe.languages"
    (langsUpdated)="langsChanged(\$event)">
</lang-edit>
```

Zahwa's atomic components are children of activity components, usually represented as a page view. An example of an activity/page component is 'new recipe' and 'recipe edit'. Activity components create or modify higher order types, or documents, assembled or edited from child components. In the given example, the variable recipe is of type Recipe, which must have a property 'languages' of type Language. The function call langsChanged() would update the 'languages' property of the recipe document and execute a service call to persist the recipe to local and remote storage. While these were prototype services in Zahwa, this pattern corresponds well to more generalisable API calls as we'll discuss in section 5.

### 3.3.2 Key findings

Many of the challenges we faced over a year of development were challenges inherent in the state of web technologies, particularly MVC frameworks. Engineering software with the Ionic 2, Angular 2 and Typescript stack represents a dramatic
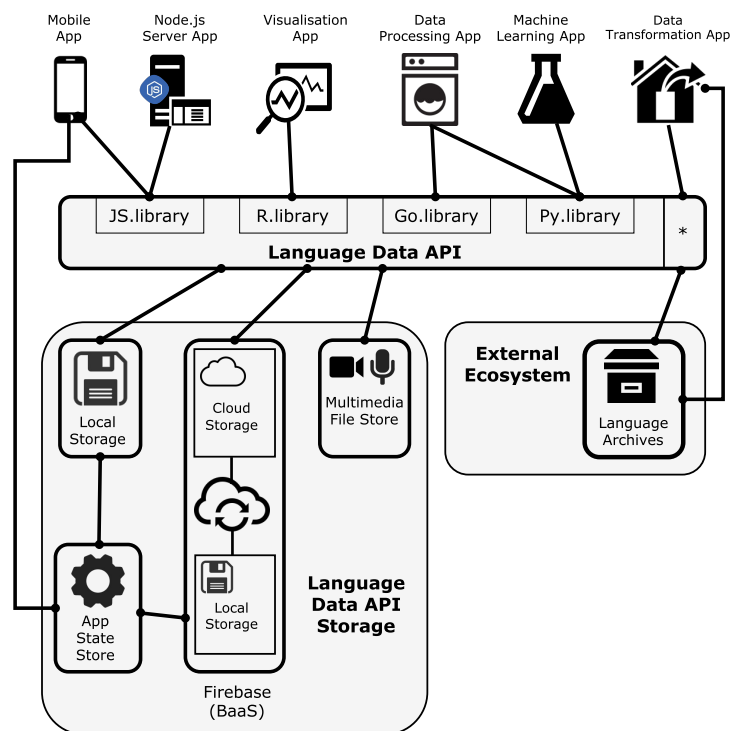
Figure 6: The ecosystem of a proposed Language Data API. Firebase is just one potential backend.

improvement at nearly every level, with fewer 'gotchas' requiring expert diagnostics. Ionic 2's definition of a 'page' as a type of component is a helpful counterpart for the notion of a language documentary *activity* combining several components toward one goal. A shift away from UI implementation details allowed us to focus on interaction design for higher level workflows.

## 4 A simple web component schema

We hope that the developers of language documentation technology will collaborate on a library of documentary components specific to the domain of language documentation. Adopting clean APIs on web-based components opens the door for others to modify open source apps to meet their requirements. Defining interfaces and providing a library of implementations is an effective way to build an open source community.[1] Current-generation JavaScript MVC frameworks offer a robust pattern for component communication based on attributes and event handelers passed as attributes on HTML templates. This can be demonstrated with this Angular 2 syntax example:

```
<type-select [input]="var" (output)="func($event)">
</type-select>
```

We suggest a simple selector name constructed from the name of a data type such as Language, and an implementation specific label based on the verb for the action such as 'select'. A example selector name is language-edit. [input] represents a one-way data binding from JavaScript variable 'var' of the type, e.g. Language. (output) specifies a local function to be executed when the component emits data. The function 'func' is passed an argument ($event) of schema {type: [Type...]}, e.g. a single property of the named type, with a value of an array of objects of this type. Occasionally components need to emit data other than the raw data type and those may be safely added as custom properties of the object.

With Typescript, we define a type Language as follows, noting that only 'name' is obligatory in this type:

```
export interface Language {
    name: string
    id?: string
    iso693?: string
}
```

A consequence of collaboration on shared components will be less duplication of effort, more reliable implementations, and ultimately, better user experiences and wider uptake of the software. Simultaneously, we lower the barrier to entry for would-be language app developers.

---

[1]This has been the model used in the computational linguistics community which has developed the *Natural Language Toolkit* (Bird et al., 2009).

## 5 Further Work

As we develop increasingly sophisticated apps, we require increasingly sophisticated manipulation of linguistic data. Supporting this in offline apps leads to a requirement for a JavaScript implementation. The API should also be implemented outside of the JavaScript ecosystem to facilitate data exchange and mobilising of data intensive capabilities into mobile applications.

Implementation of an API can be seen as an extension of defining language data types as the common interface for components as discussed earlier. We will develop the API as a JavaScript library initially, for use across mobile apps and server instances utilising NodeJS. Figure 6 illustrates the language data API in an ecosystem including research tools in other domains.

Our aim is for this API to encourage collaboration by facilitating data interchange between an array of language documentation apps for different audiences. A common API provides a gateway for other ecosystems to collect linguistic data and to mobilise existing data to new audiences via mobile devices.

## 6 Conclusion

In this paper, we have reported on our investigation of web technologies to craft a series of web and mobile apps in language documentation. We have shown with Aikuma-NG that it is feasible to use these technologies to migrate the well-established genre of audio transcription with waveforms to platform-independent web technologies. Aikuma-Link provides a glimpse of new capabilities arising from the low onboarding cost of mobile web apps, and the potential for a new generation of crowdsourcing applications. Finally with Zahwa, we developed a complete mobile app for a narrowly-defined linguistic task, and supported the online-offline requirement of many fieldwork situations.

Despite the inherent productivity gains of web technologies, our prototyping experience was occasionally frustrating due to the lack of maturity of some common technologies. We initially struggled to find a suitable pattern for component modularisation, data interchange and online-offline storage. However these common problems were and remain the target of significant engineering efforts by major players and the current situation is already much improved. A significant benefit of this prototyping work was reaching the point where we could collaborate with our target audience and deliver software people want to use.

There are many opportunities for collaboration in this space of app development, to unite existing initiatives and communities, and to share implementations. The work reported here has already served others as an effective starting point for quickly developing new mobile apps. For those who prefer to use other technologies, we nevertheless hope to collaborate on the design of a shared implementation-independent language data API. Our ultimate goal is to employ the web platform to connect tools outside of the web genre, improving the flow of data and the production of language documentation, while gaining rich new capabilities we have yet to explore.

## References

Peter K Austin. 2010. Current issues in language documentation. In *Language Documentation and Description*, volume 7, pages 12–33. SOAS.

Bruce Birch, Sebastian Drude, Daan Broeder, Peter Withers, and Peter Wittenburg. 2013. Crowdsourcing and apps in the field of linguistics: Potentials and challenges of the coming technology.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media.

Steven Bird, Florian R Hanke, Oliver Adams, and Haejoong Lee. 2014. Aikuma: A mobile app for collaborative language documentation. In *Proceedings of the 2014 Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 1–5.

Margaret Carew, Jennifer Green, Inge Kral, Rachel Nordlinger, and Ruth Singer. 2015. Getting in touch: Language and digital inclusion in australian indigenous communities. *Language Documentation and Conservation*, 9:307–23.

MaryEllen Cathcart, Gina Cook, Theresa Deering, Yuliya Manyakina, Gretchen McCulloch, and Hisako Noguchi. 2012. Lingsync: A free tool for creating and maintaining a shared database for communities, linguists and language learners. In *Proceedings of FAMLi II: workshop on Corpus Approaches to Mayan Linguistics*, pages 247–50.

Georgios Chatzimilioudis, Andreas Konstantinidis, Christos Laoudias, and Demetrios Zeinalipour-Yazti. 2012. Crowdsourcing with smartphones. *IEEE Internet Computing*, 16:36–44.

Jean-Philippe Goldman, Adrian Leemann, Marie-José Kolly, Ingrid Hove, Ibrahim Almajai, Volker Dellwo, and Steven Moran. 2014. A crowdsourcing smartphone application for Swiss German: Putting language documentation in the hands of the users. In *Proceedings of the 9th International Conference on Language Resources and Evaluation*, pages 3444–47.

John Hatton. 2013. SayMore: language documentation productivity. In *Proceedings of the 3rd International Conference on Language Documentation and Conservation*. University of Hawaii.

Adrian Leemann, Marie-José Kolly, Ross Purves, David Britain, and Elvira Glaser. 2016. Crowdsourcing language change with smartphone applications. *PloS one*, 11(1):e0143060.

Johann Poignant, Mateusz Budnik, Hervé Bredin, Claude Barras, Mickael Stefas, Pierrick Bruneau, Gilles Adda, Laurent Besacier, Hazim Ekenel, Gil Francopoulo, et al. 2016. The CAMOMILE collaborative annotation platform for multi-modal, multi-lingual and multi-media documents. In *Proceedings of the 10th International Conference on Language Resources and Evaluation*. European Language Resources Association.

Taiwan Indigenous Council. 2016. Taiwan Indigenous Council e-dictionary.

WaveSurfer. 2017. WaveSurfer.js. `https://wavesurfer-js.org`.

Jonathan Wright, Kira Griffitt, Joe Ellis, Stephanie Strassel, and Brendan Callahan. 2012. Annotation trees: Ldc's customizable, extensible, scalable, annotation infrastructure. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*, pages 479–85. European Language Resources Association.