http://install.diawi.com/h4zDDL

HI-FI PROTOTYPE REPORT

Alex Wu, Anna Yelizarova, Ishita Prasad

INTRODUCTION

Tongues is a crowdsourcing application that helps people find real time, accurate and colloquial translations. When automatic translators just can't get it right, Tongues enables users to ask the people around them what the real way to say something is. This project is being developed by Alex Wu (Team Manager and User Testing), Ishita Prasad (Visual/Interaction Designer and Documentation Coordinator), and Anna Yelizarova (Developer and Visual/Interaction Designer).

PROBLEM & SOLUTION

Modern computer generated translations are inaccurate, difficult to use, and give overly general translations that do not always match up to how the language is used in the real world. Our solution provides users with on demand, region-based translations. This system harnesses the crowd to make translations as accurate as possible, while also incorporating the nuances of colloquialism and natural speech.

TASKS

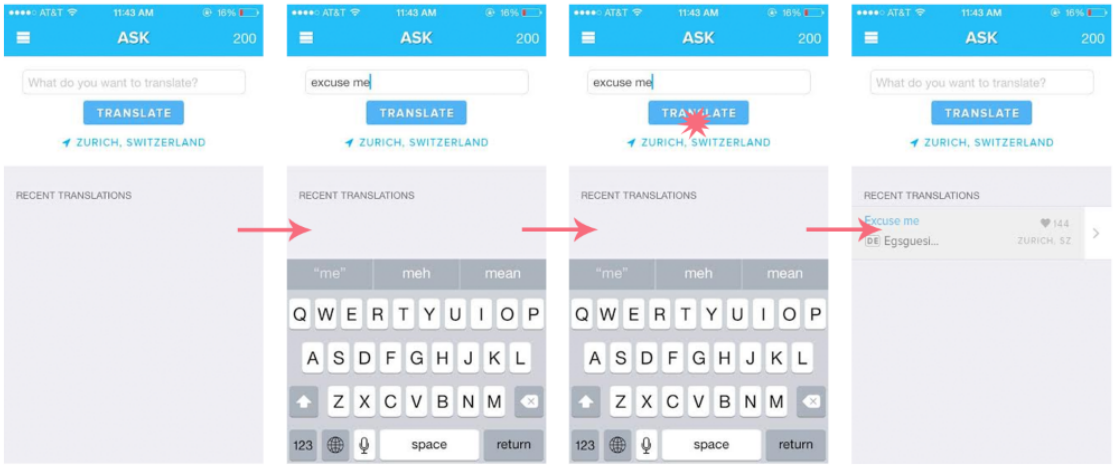
The three tasks that our prototype enabled ranged from simple to difficult, as indicated below:

1. **Ask for a translation:** a user could request a translation, and wait for and receive responses from the people in their location. Because our initial inquiries indicated that language use can differ from region to region, we wanted to give people a way to find translations by region as opposed to language, as most current systems work.

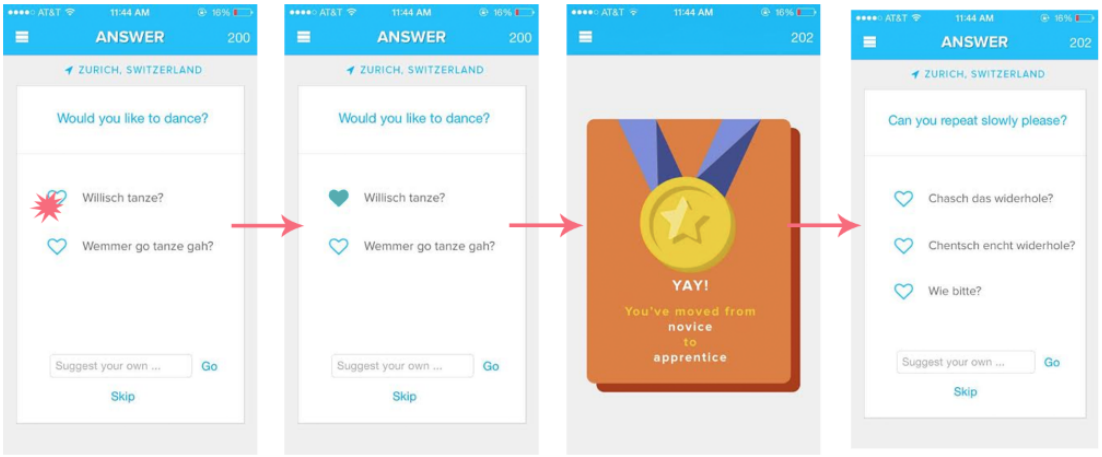
2. **Answer translations:** a user can participate in an upvoting-style game by upvoting or writing their own translation for requested translations. This process is intended to be a humorous insight into their culture or locality's colloquial quirks and phrases. In response to feedback, we changed our initial point system to one in which users could "level up" by upvoting or writing more translations, moving from, say, "Novice" to "Apprentice" and other fun levels. We chose this gamified task to engage users who can help out the people performing task (1).

3. **Change location:** a user can change their location to request or answer translations in regions other than their own. We added this as a task because it is central to our idea of "region-based" translation, and expands the user base (from just people in foreign countries to anybody).

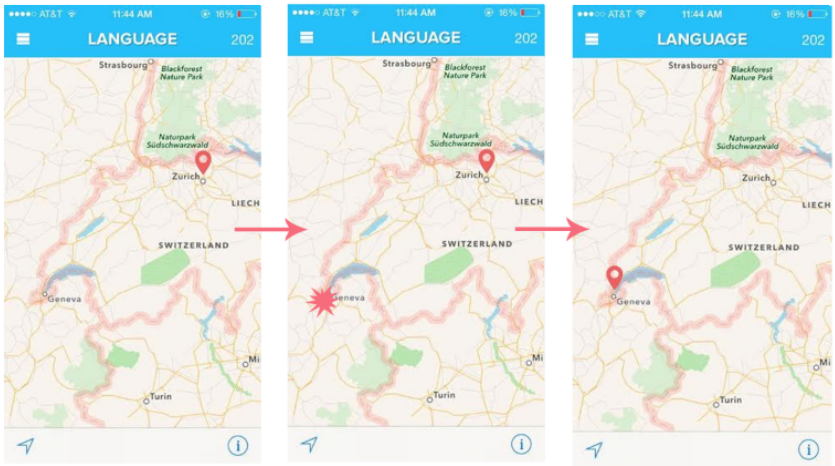
Here are how the tasks are carried out in our final design:



ASK STORYBOARD



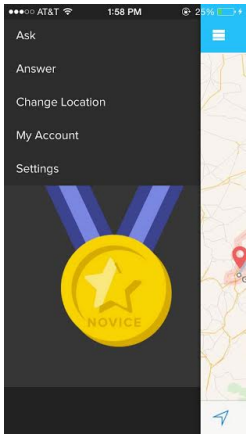
ANSWER STORYBOARD



CHANGE LOCATION STORYBOARD

MAJOR USABILITY PROBLEMS ADDRESSED

1. [H2-1 Visibility of System Status][Severity 3][A,B,C,D]
The Ask/Translate buttons of the “French Translator” screens are too small and on the upper right corner of the screen. Users may miss that as the main status indicator of the screen. Remove the “TONGUES” text and make those two buttons larger and centered.

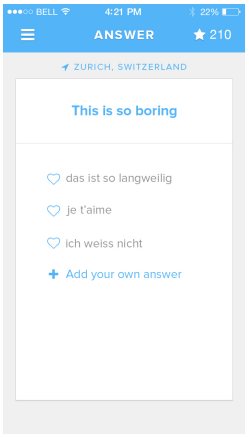


Solution:
We moved the moved the ask/translate options to their own slide out navigation bar. This freed up clutter from the interface and helped simplify our design overall.

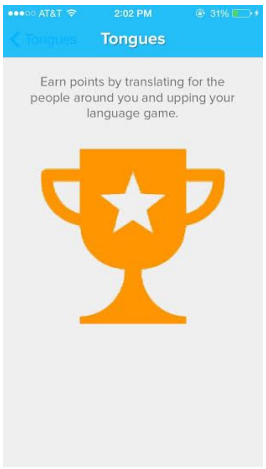
2. [H2-4 Consistency and Standards][Severity 4][A,B,C,D]
The interactive portion of the interface of the Translator screens is misleading as it seems to resemble a messaging app. The user might mistake this as a conversation with someone else. Fix by providing a clear flow with a “Translate” button or with distinct portions of the screen reserved for input/output.

Solution: We changed the layout of the translator screen to a text box on top of the screen and a clear translate button, with previously translated phrases underneath. This eliminated the problem of the confusing “messaging app” style.

3. [H2-4 Consistency and Standards][Severity 3][A,B,C,D]
The “upvote” button for phrases doesn’t seem to resemble the function of upvoting. Rather, it looks like moving or a traffic sign. Replace with something like thumbs up that is more indicative of voting up.

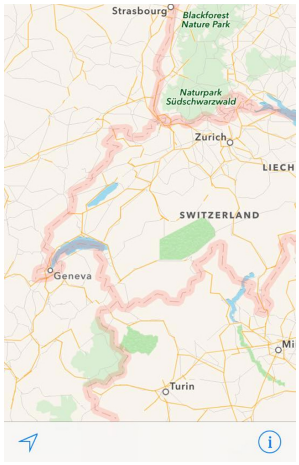


Solution: We ended up using a heart button rather than a “upvote” button. This was to make it clearer to the user that they are choosing the answer that they prefer.



6. [H2-10 Help and documentation][Severity 3][A,B,C,D]
The “German Translator” page is completely blank with no instructions or content. Fix this by adding elements. Furthermore, there is no clear home page with guidance to the tasks, and there is no documentation while using the app.

Solution: We added a “tutorial” that plays the first time the user uses the app. This tutorial is a series of images that explain to the user how the app works, and provides all of the necessary instructions for operating the app.



8. [H2-7 Flexibility and efficiency of use][Severity 3][A,B,C,D]
When selecting a region for localization, the user has to click on the location on a map. It would be nice to also allow users to type in the name of the city/region/country they are in (similar to the functionality available for weather apps).

Solution: On the region selection screen, we added the option for the user to type in the city/region/country they want a translation from.

10. [H2-7 Flexibility and Efficiency of Use][Severity 3][A,B,C,D]
Right now, languages are taken directly from the area; the user finds the area he/she is searching in, and the language for that area pops up. But the same area could have multiple languages, so the user should be able to tailor the language he/she wants to translate instead of just the area. Add a button somewhere that allows the user to specify his/her language of choice regardless of geography.

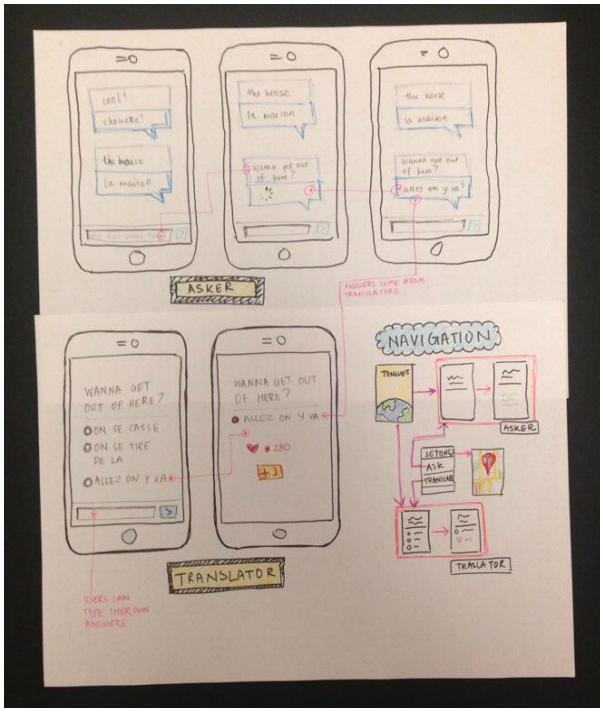
Reason for not fixing: We didn’t fix this violation because we felt the app should produce a result reflective of the most commonly used language in the region. Our app gives users translations based on the region, not the language, and thus produces a result that is understandable given a region, no matter what language This also allows the app to support regions where the most commonly spoken phrases are combinations of different languages. Instead of fixing this violation, we changed the way we present/“market” our app and phrase the wording within it to reflect this idea.

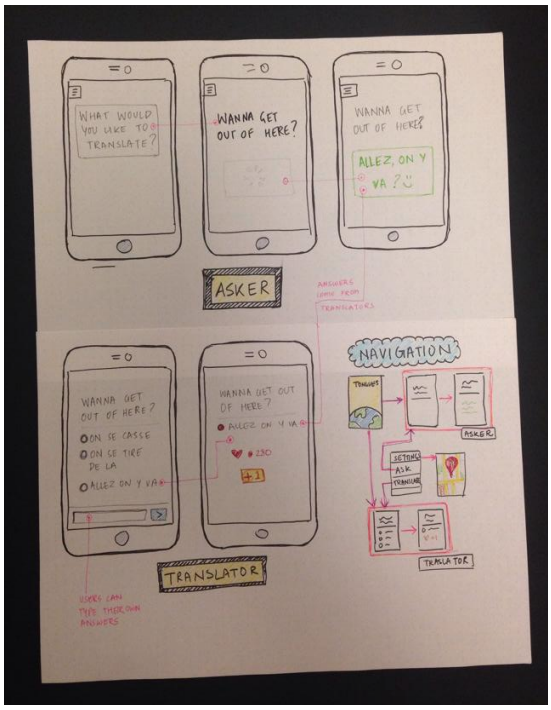
Design Evolution

Initial UI Sketches/Concept video:

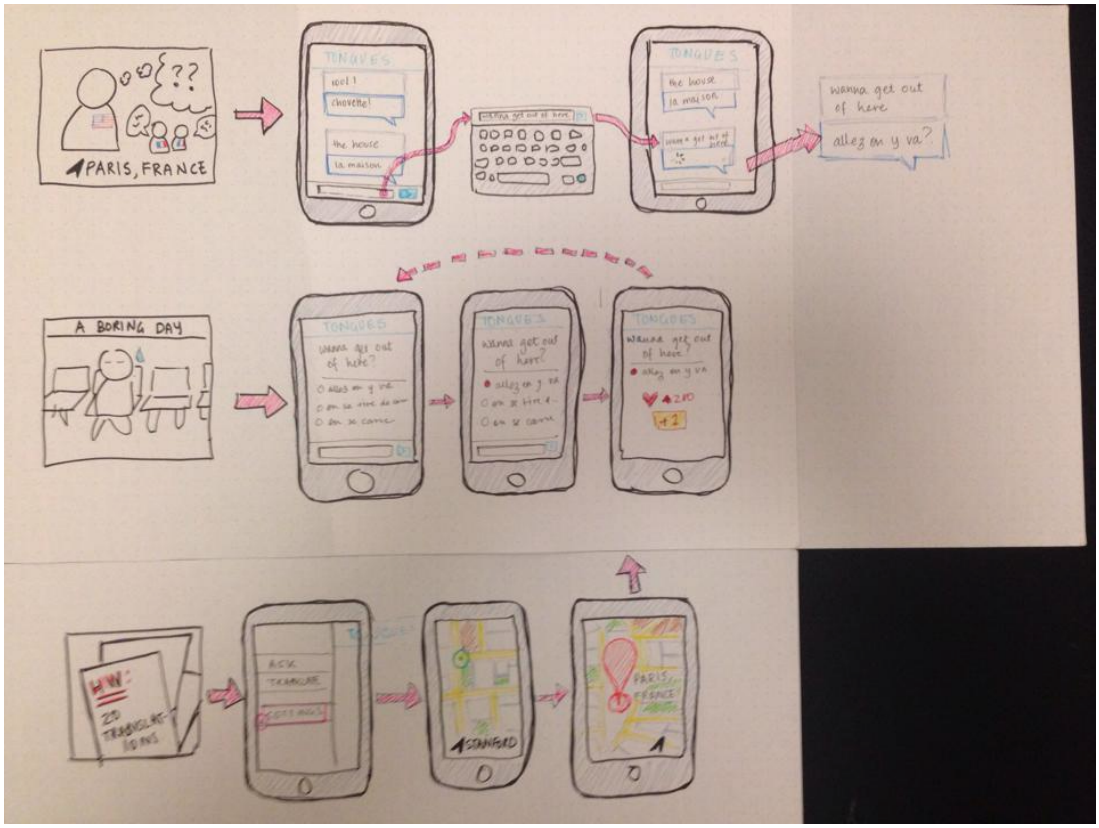
Our concept video ended up being a compilation of four different ways we imagined our app being used. We felt that different types of people would use the app in different ways, prompting us to use different actors for each of the four scenarios (on the bus, at the market, at a party, and while studying).
Our UI sketches and video planning storyboards reflected our imagined use cases for our app.

UI Sketches:

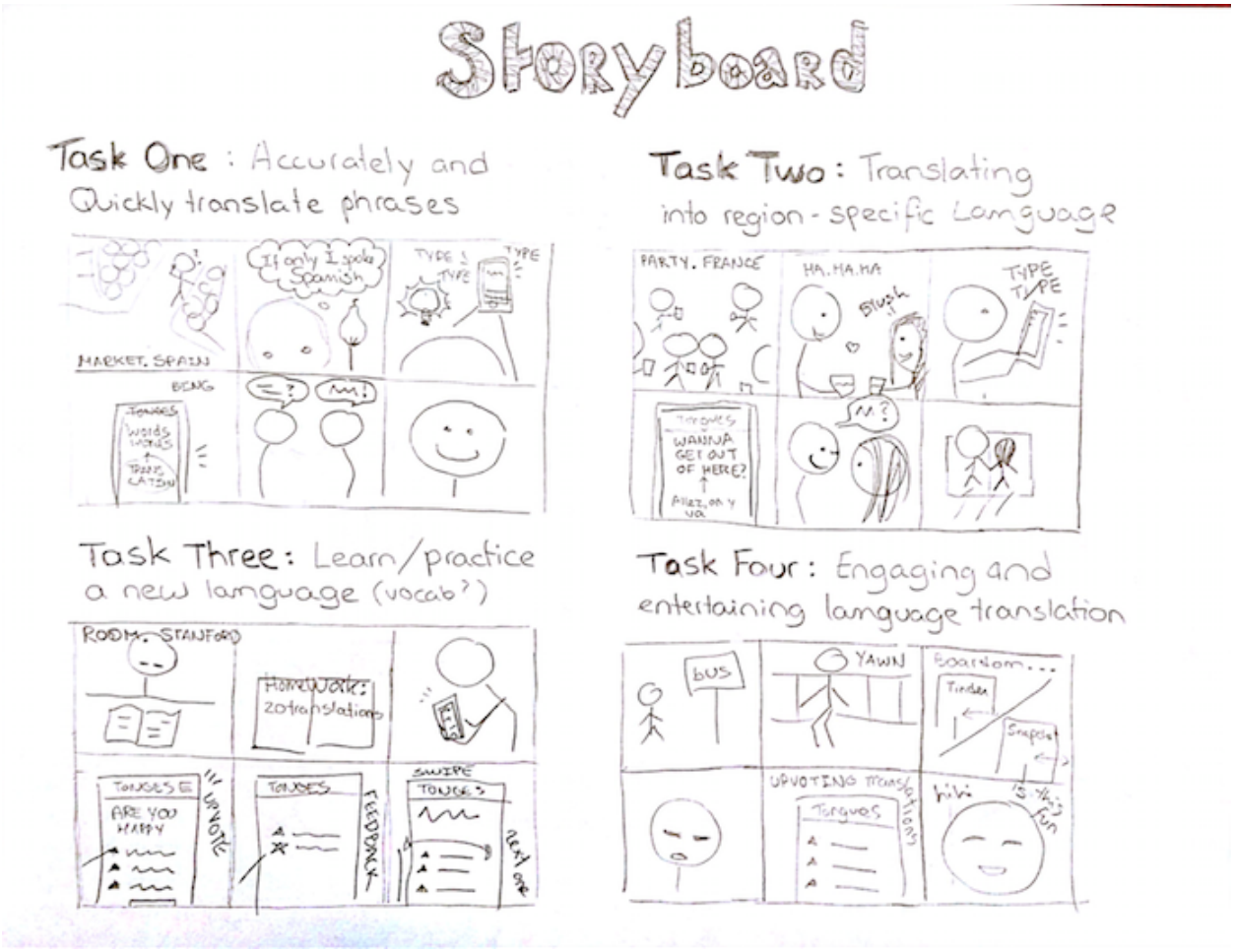




UI Storyboards:

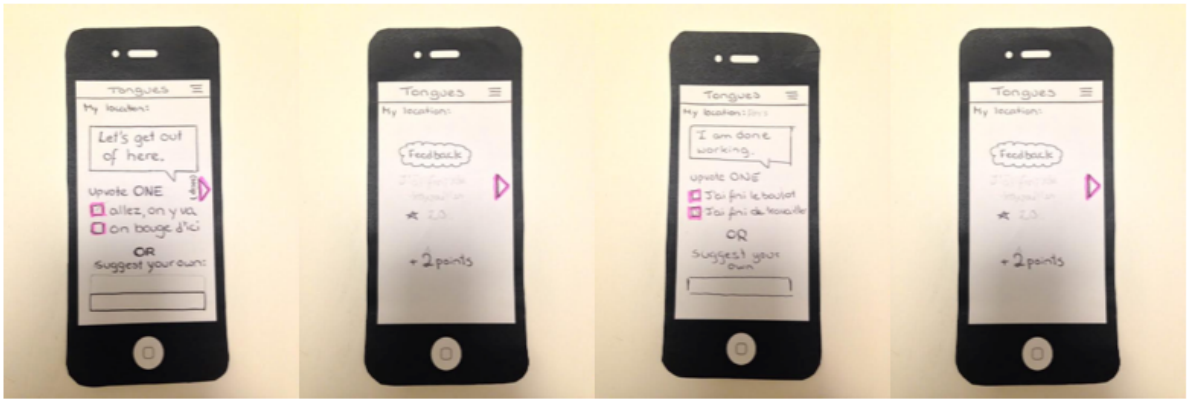


Video Planning Storyboards:



Lo-Fi Prototype:

Our Lo-Fi prototype was composed of three sets of paper screens, one for each “task” that the user could perform with our application (the tasks are described in more detail in a later section of this paper). The user interacted with the screens with a pencil either “typing” by writing, or “clicking” by hitting a button with the pencil. When an interaction prompted a screen change or customized response, the human computer would swap out the screens or write on the screen, respectively.

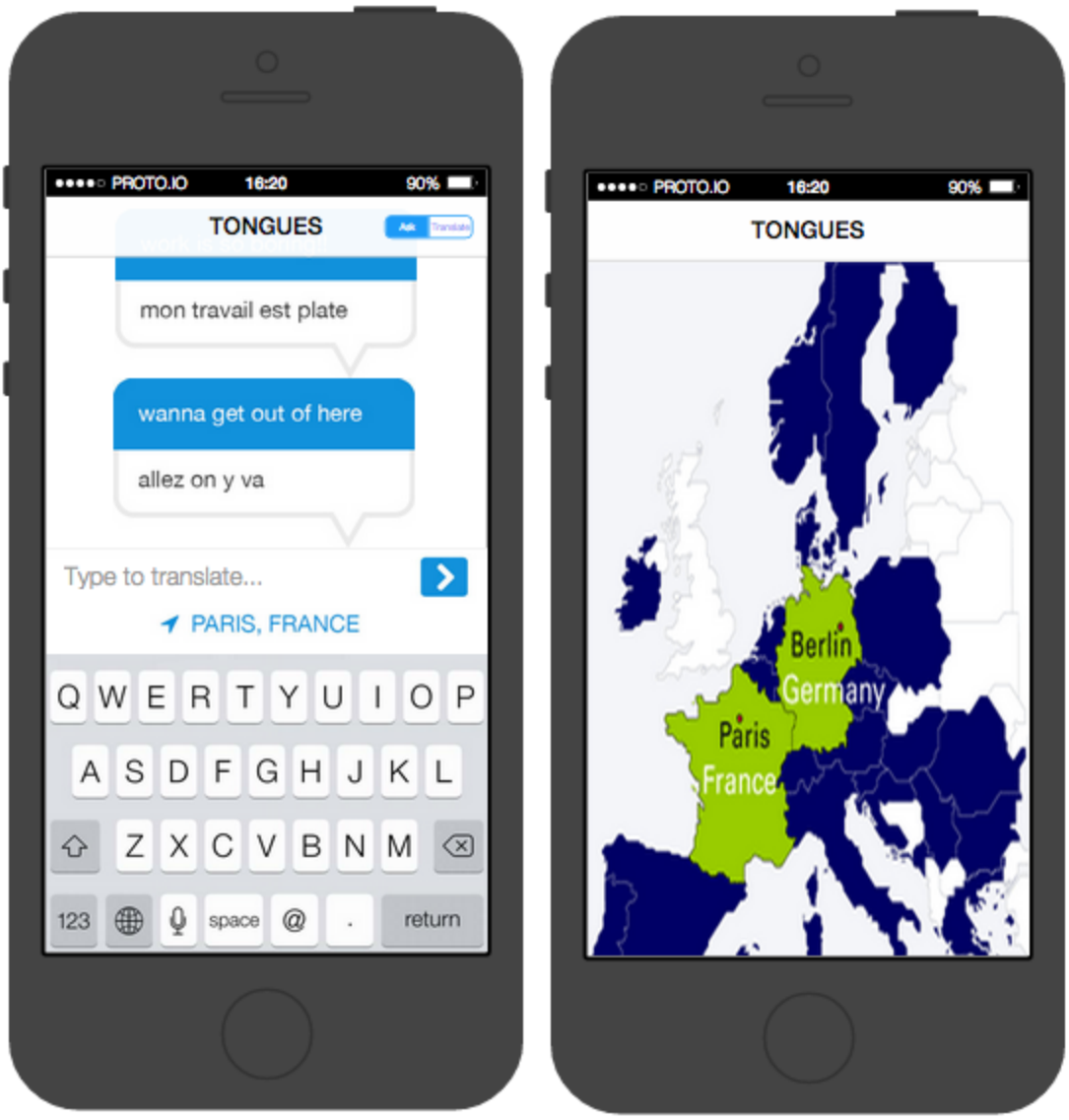


Design Changes:

We ended up going with a more simplistic, functional design. From our interviews, we discovered that people valued simplicity, ease of use, and speed when it came to translations.

Medium-Fi Prototype:

To construct our medium-fidelity prototype, we used the tool *Proto.io*. We surveyed a number of prototyping tools, including Marvel and InVision. Ultimately, we chose Proto.io because we felt that it had a stronger capability to create forked storyboards based on different user inputs, and it also had the feature of letting the user type - a crucial piece of interaction for our application. Additionally, we used Photoshop to create the graphics. Although Photoshop has a heavy functionality, we chose to use it as it gave us more freedom to create a unique design, and our team knew the application well.

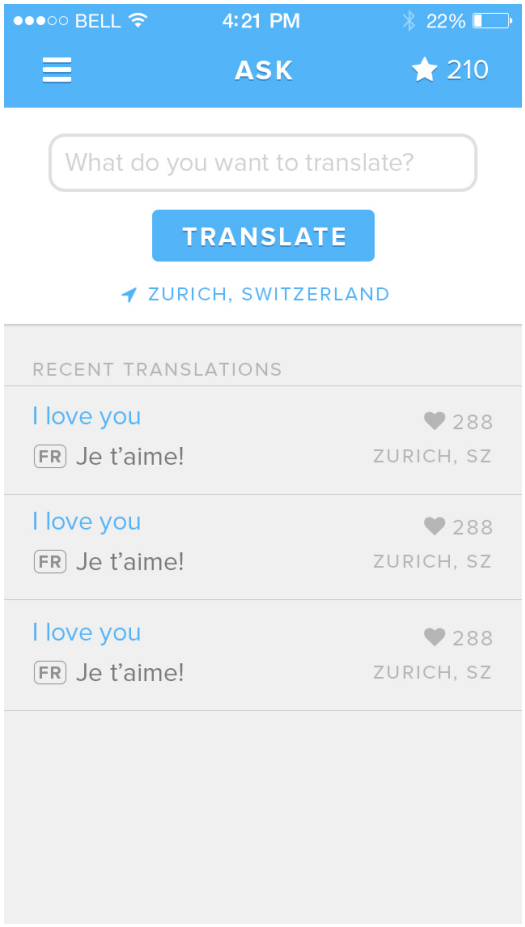


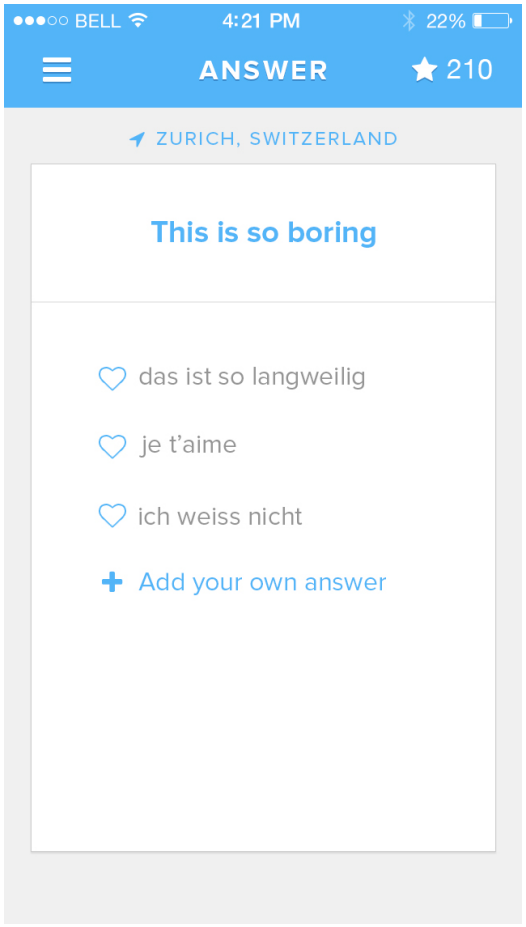
Design Changes:

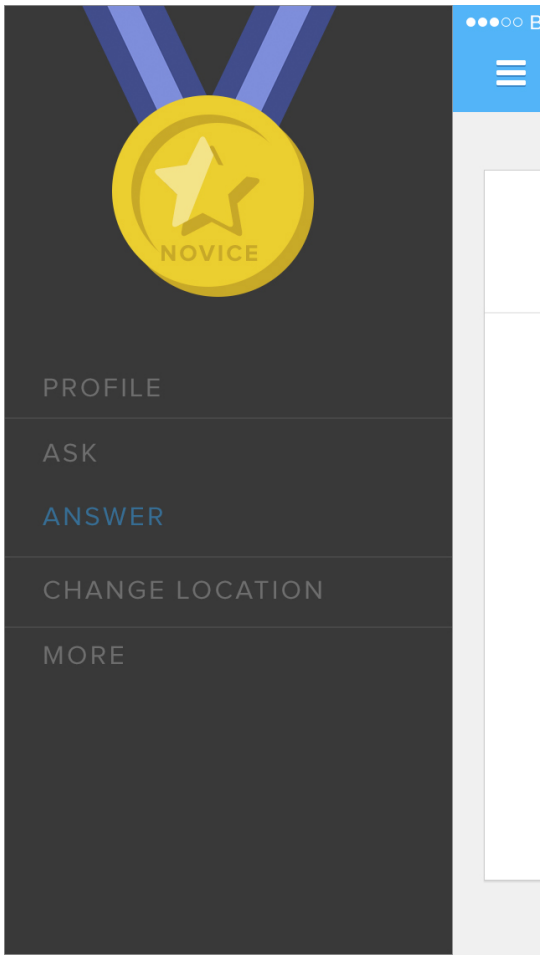
Based on the feedback we received from our previous prototype, we made 2 major changes to our UI. First, we changed our transitions on the translator side. In our low-fi prototype, we had a feedback page after each upvote. However, according to our participants, this was slow and confusing to them. Instead we opted to have the feedback and rewards appear on the same page as an animation. We ended up having a star appear next to the saying if the top pick was chosen. This ties into our next major change - better gamification. Instead of having points pop up like in our old prototype we decided to replace it with an icon. Users were confused by what the points meant so instead we wanted to have something more intuitive like thresholds and badges.

Hi-Fi Prototype:

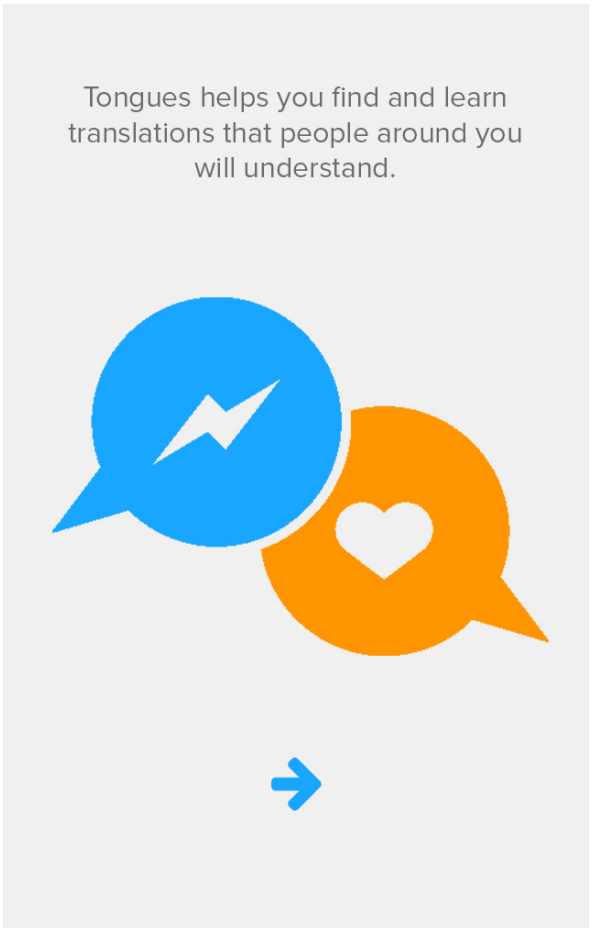
Our Hi-Fi prototype is the culmination of all of the previous iterations of our design. We created a simple, yet functional (and aesthetic) design, that accomplishes all three our initial tasks.


BELL
4:21 PM
22%
ASK
210
What do you want to translate?
TRANSLATE
ZURICH, SWITZERLAND
RECENT TRANSLATIONS
I love you
288
FR Je t'aime!
ZURICH, SZ
I love you
288
FR Je t'aime!
ZURICH, SZ
I love you
288
FR Je t'aime!
ZURICH, SZ


BELL
4:21 PM
22%
ANSWER
210
ZURICH, SWITZERLAND
This is so boring
das ist so langweilig
je t'aime
ich weiss nicht
Add your own answer


NOVICE
PROFILE
ASK
ANSWER
CHANGE LOCATION
MORE


YAY!
You've moved from
novice
to
apprentice



Design Changes:

For our Hi-Fi prototype, we made most of our design changes off of the heuristic violations listed in the previous section. Most of the changes relied on increasing/improving the gamification, simplifying the app, and creating documentation (the initial tutorial).

PROTOTYPE IMPLEMENTATION

We implemented our app in iOS using Xcode in Objective C. Apple's integrated development environment is really good - it offers features like a graphical interface builder and a syntax-aware and compiler-aware source editor. The greatest advantage however is the quantity and quality of resources found online for iPhone programming. If you are trying to do something, odds are someone else has done it before you and there are a multitude of related tutorials and stack overflow posts. That is one great advantage iOS has over Android. We also chose Objective C over Swift for the same reason - more documentation available online.

The disadvantage of working in this environment was that our team didn't have strong iOS experience and couldn't push our idea as far as our imagination. Unlike with Android, where people

are more likely to have Java experience, the number of people who have encounter Objective C is smaller. We were limited by our skills and what we were and weren't able to implement in Objective C. A lot of it had to do with finding good tutorials, experimenting and learning as we went. The other major problem we had was effective collaboration. We didn't figure out how to share our Xcode project on github and were forced to email each other updated versions. This made us waste time as we would be waiting for a team member to finish something before being able to add to it ourselves.

We used a couple of "Wizard of Oz" techniques in this prototype. The main technique we used was related to our translations. Since we did not implement a backend (nor had the user base to make our translations behave as intended) we had a local set of phrases that were pre-translated. During our demonstration, we memorized the order in which the pre-translated phrases would appear when the translate button was pressed, and typed in those phrases so it would appear our app was actually doing the translating. Another technique we used was related to our translation data. Since we had no actual users of the app, we used preset numbers for all of the data displayed (point totals, "heart" totals, etc.). This was to show how the app would look if many other people were using it.

The biggest problem we could foresee in our project was how to simulate the crowd in our Hi-fi prototype. Because our app relies on crowdsourcing to provide translations, we had to make a small dictionaries of sayings in the two languages we chose to work with. In the asker side and the answer side, all the translations provided were hard-coded data.

Because we were limited by our translations and kept to two languages, we had to go about our third task (changing region) accordingly. Ideally what our real product would function as is google maps, but that's something that was missing from our Hi-fi prototype. We tried to make it look as close as possible with various design elements, but ultimately our page was a static image whereas the designed solution has a scrollable world view and search bar. Since we only made a translation libraries for the cities of Zurich and Geneva, we had no choice but to restrain it, Another thing we would spend much more time on would be gamification. Since that was not one of our three tasks, we made it but didn't focus too much on it. We would have a better feedback system if we were to implement our full solution as a real life app. For instance when someone upvotes an answer, there would be some kind of animation or pop up which indicated if you picked the most popular option or not and how the other ones did. We would also implement an account page to go along with it to better track your progress. With an account we would also need a login/password screen. This version would obviously need a real database instead of hardcoded arrays.