Project German Rosetta Tales

GROUP NUMBER: 23

TEAM MEMBERS

Jaesung Lee
Dave Liu
Randy Shi
David Au

TABLE OF CONTENTS

Team Contribution

David Au:

David and Dave will be working together using the Proto.io in order to redesign the prototype in order to include some new features given from the feedbacks from the interviewees. In addition, they will also be in charge of making the storyboard for the task for the application. Lastly, David will be mainly organize the structure of the report document. He adds the book titles, pictures, and the quick summary into the dynamoDB table called Book.

Dave Liu:

Dave and David will be working together using the Proto.io in order to redesign the prototype in order to include some new features given from the feedbacks from the interviewees. In addition, they will also be in charge of making the storyboard for the task for the application. Lastly, Dave will be checking on the overall content of the report document. He adds the couple of sentences for each book and demonstrates the translation in the dynamoDB table called BookMetaData.

Randy Shi:

Randy and Jaesung will both be working on the implementation aspect of the prototype. They both will go over the latter part of the grading spec and provide other half of the group the needed screenshots of their implemented tasks. Randy will be doing the recording that is required to demonstrate the features being implemented. Partnering up with Jaesung, Randy will help Jaesung with implementing the architecture Jaesung came up with. Furthermore, Randy will do the database management as well as Lambda and DynamoDB integration.

Jaesung Lee:

Jaesung is the coder and engineer in our group. In terms of implementing our product, Jaesung will come up with software architecture including database schema, API protocol, as well as overall structure of the web application. So he will be doing the most of coding. Jaesung will build our GUI component using ReactJS, configuring the product from end to end using WebPack and Redux. By using the latest technology stack, Jaesung will also introduce us with trending web development principles and patterns. Having fairly frequent experiences with using AWS, Jaesung will also help out on building the VUI.

Problem and Solution Overview

Problem: Learning through apps often requires significant effort on the user's part to commit to learning as well as implementing the "lessons". Additionally, many apps teach through words, which might be good for reading/writing, but poor techniques compared to a voice-interface, which would train conversational adeptness in a language.

Solution: Create a method for passive language learning by replacing some of the English words in engaging stories with words of a foreign language and then giving translations for those words. This is highly similar to early education language development techniques used on TV shows (Dora the Explorer) and in classrooms with English as a Second Language (ESL) children. Additionally, by reading a fully-English sentence first, and then replacing parts of it, users are exposed to the original English context before being exposed to another language. Then after exposure to a foreign word and told its translation, the switching between two languages reinforces memorizing the word. Not only will users memorize words, but memorize them in an active context, while not actively engaged themselves, since the text is being read to them. Active learning means the ability to apply knowledge in different scenarios, which contrasts passive learning, which would only develop the skills to recognize specific patterns. The difference between active and passive is the difference between conversational and reading abilities within a language. We are currently focused on proper grammar, so we want to implement mainly noun translations to minimize potential errors in translation. Additionally, we have a GUI that can be used for supplementary read-along to the VUI text. It can also be used to review replaced words.

Tasks

Task 1: (Easy) [Select a Book]

Selecting a book is easy for the user. They're provided a list of options, and simply need to choose the one they'd like.

Representative Task

Context Reader - "The list of available books is as follows: Animal Farm, Game of Thrones, The Book Thief."
User - "Read The Book Thief"
Context Reader - "Book Successfully Loaded."

Task 2: (Middle) [Set the Speed of the book at any time: reading, main menu, etc.]

This requires a bit more commitment on the part of the user. They would need to consult our help functionality, which will prompt them with the ability to adjust speed. Fortunately, this functionality works even without an 'active' book, but can also work in the midst of reading.

Representative Task

User - "Set speed to slow"
Context Reader - "I have set the speed to slow"
(From now on, the app pauses .6 seconds between each word, until back to Normal.)

Task 3: (Hard) [Setting Replacements for languages]

This is the most tedious to use because it requires the user to learn, through either the main prompt or help functionality, that there are two options: one word or two word replacement. Upon activating the prompt, they would need to specify how many words replaced, and told that they can either choose one or two. However, they might incorrectly try to choose a non-valid number or percentage.

Representative Task

User - "Replace two words"
Context Reader - "I will replace two words in each sentence for you. Say main menu to return to the main menu."

Revised Interface Design

Changes and Rationale as a Result of Testing (screenshots below)

For the final GUI modification of the project, we decide not to have the help page since it will now be covered in the VUI component (Figure 1.a).

Instead of a cover page (Figure 1.b) and main menu of buttons for different functionalities (Figure 1.c), we chose to design a homepage that serves a list of books, so that the user can quickly immerse in the app (Figure 2-3). Here, we replaced alphabetical listing with a search bar for books, to reduce search time. Alphabetical order would still take up a significant amount of time to scroll through if there are numerous books. Interactive prototype testing revealed that this would be optimal timewise.

Originally we were going to put the book summary in a separate page right after the user clicks ‘Read’ from the GUI. However, interactive prototype testing revealed that it would be more convenient for the user to be able to browse through the entire book list at the front page while seeing the summaries at the same time (Figure 3).

We added the ability to go through each sentence for each book and listed translations for each sentence (Figure 4).

We also developed “read from beginning” and “main menu” on our GUI (Figure 5).

We decided to revamp the [About Us] page and include a Linkedin account instead of having emails to contact us (Figure 6). Interactive prototype testing revealed that this would be optimal for minimizing spam and maximizing valuable communication.

Originally, we had planned on implementing a settings option for the GUI. However, based on interactive prototype testing, we found that it was actually more convenient to have the user be able to configure the settings in the VUI while he/she is exposed to the dictation. You can view the original settings page GUI design in Figure 1.d.

Screenshots of Unimplemented UI

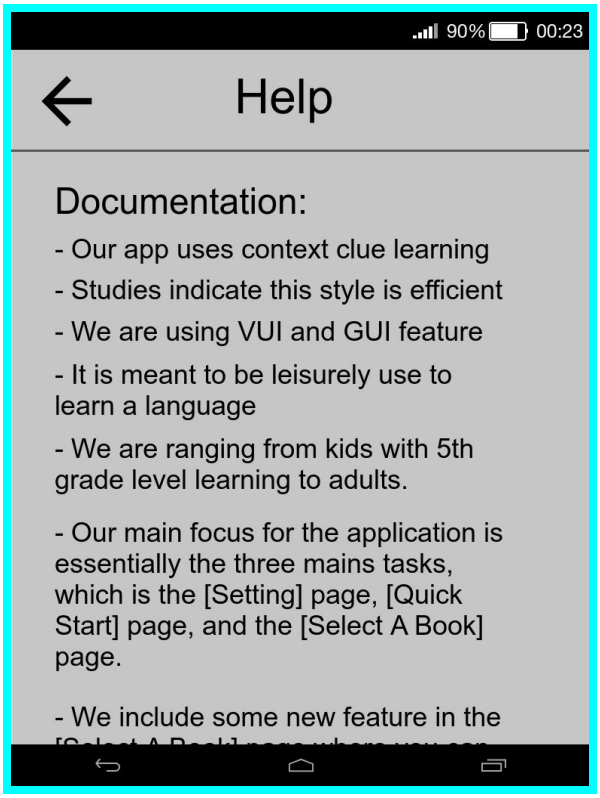


Figure 1.a: List of Instructions & Documentation

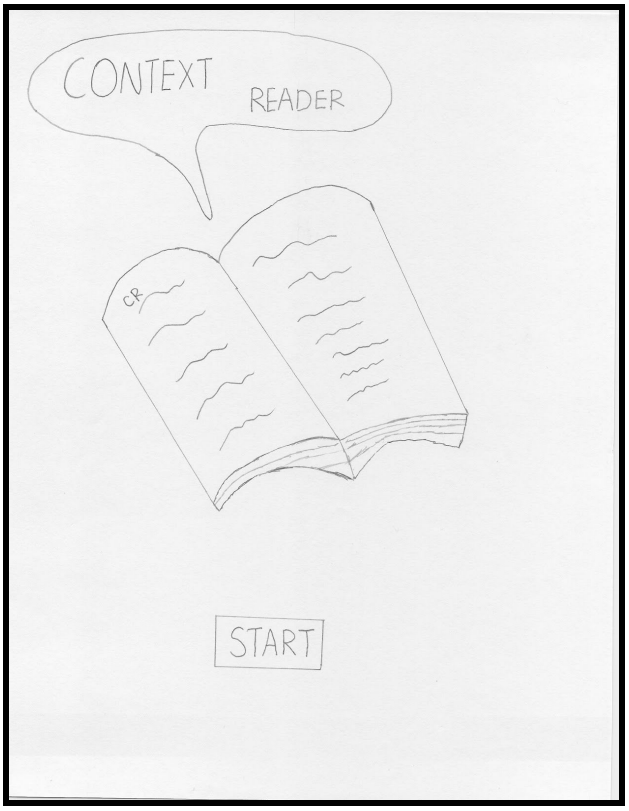


Figure 1.b: Cover Image for our Application

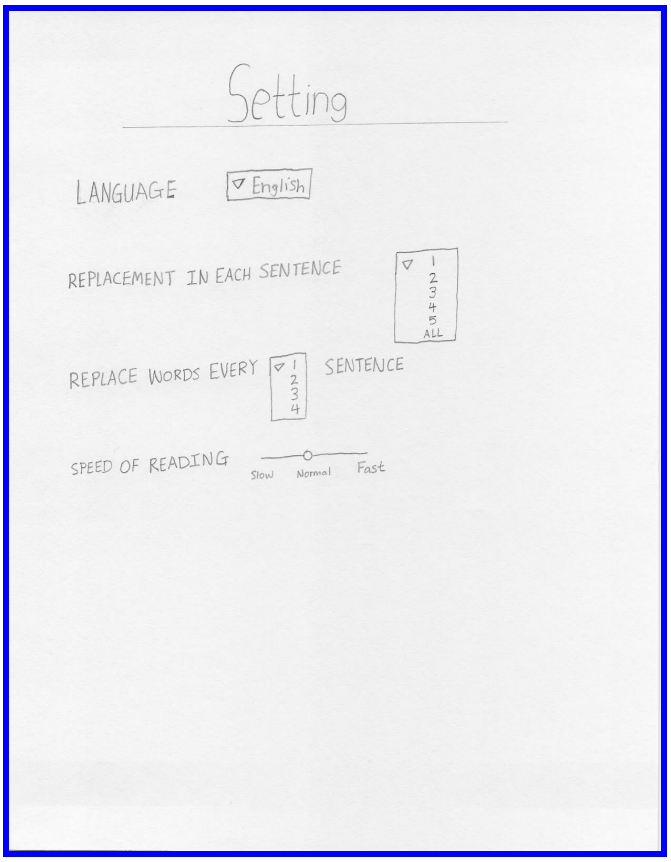


Figure 1.c: Original Main Menu

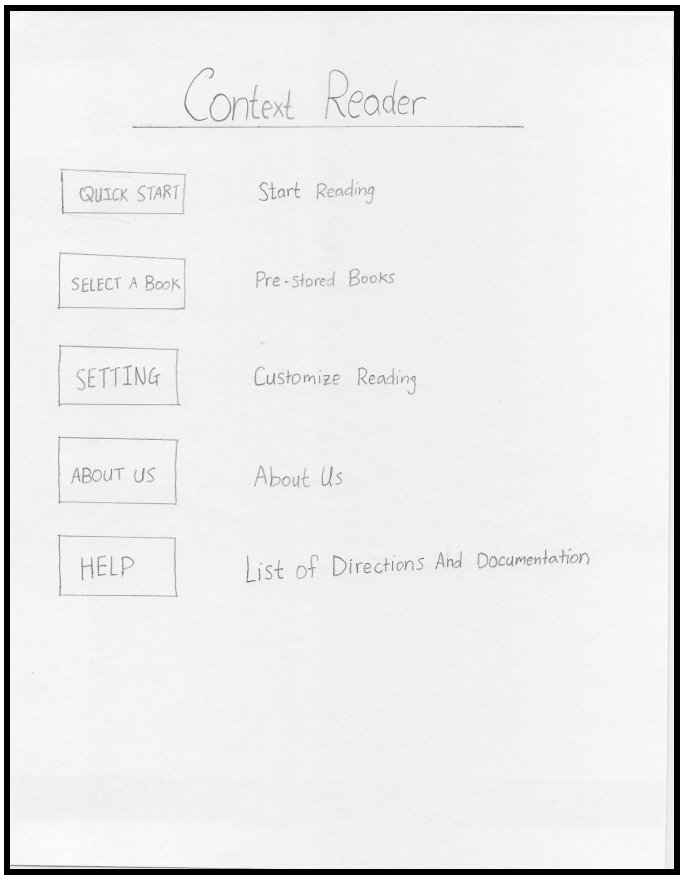


Figure 1.d: Intended Settings Page

Screenshots of Implemented UI

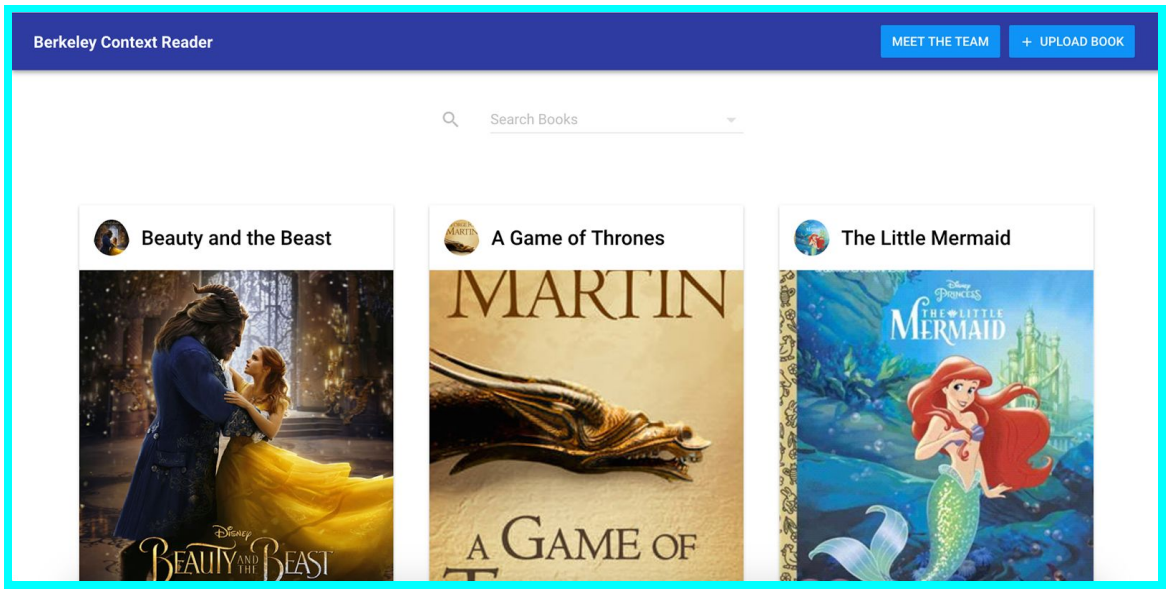


Figure 2: Main Page with Book List

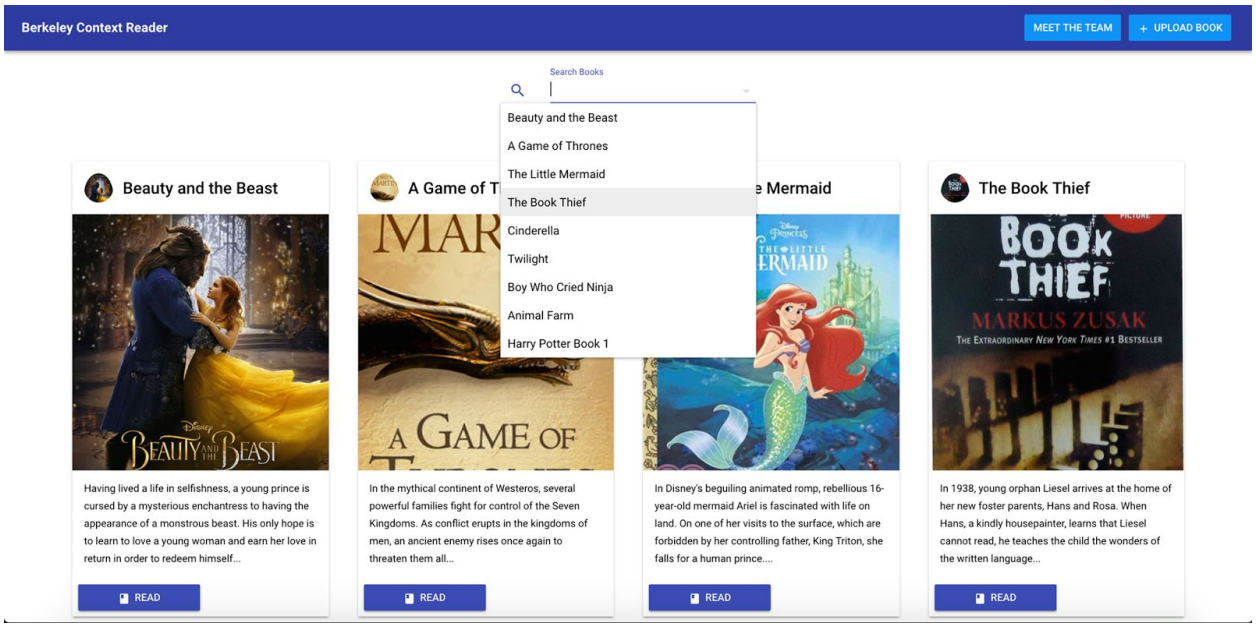


Figure 3: Main Page with Book List (Search Option)

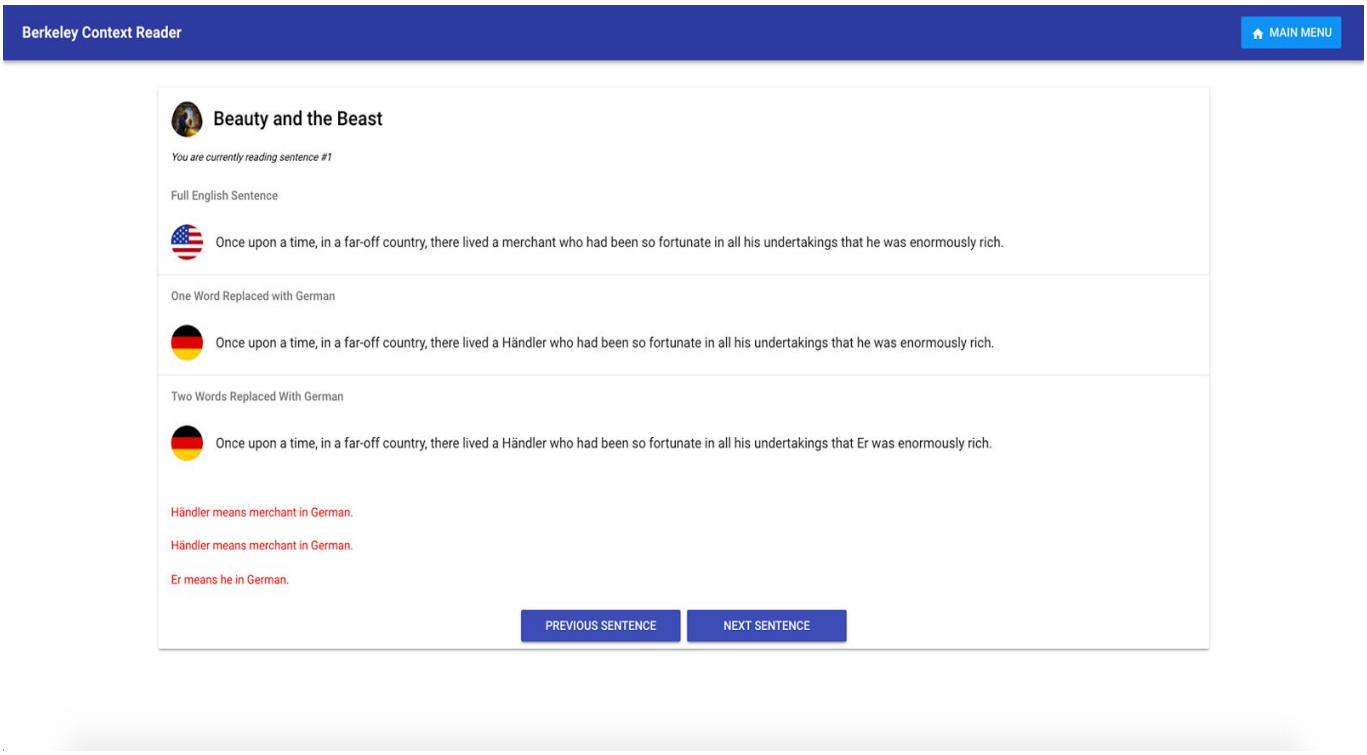


Figure 4: Beginning to Read Each Sentence of a Book

Figure 5: Completion Page After Reading a Book

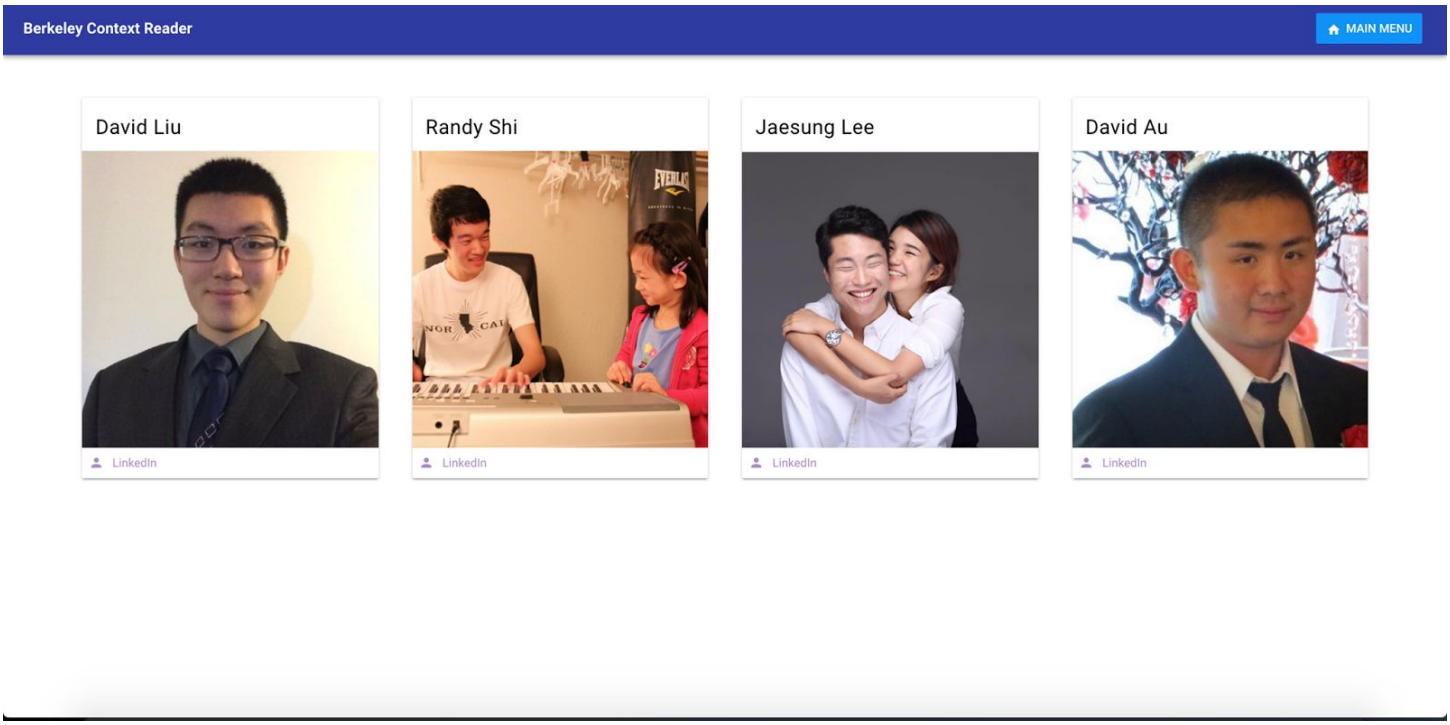
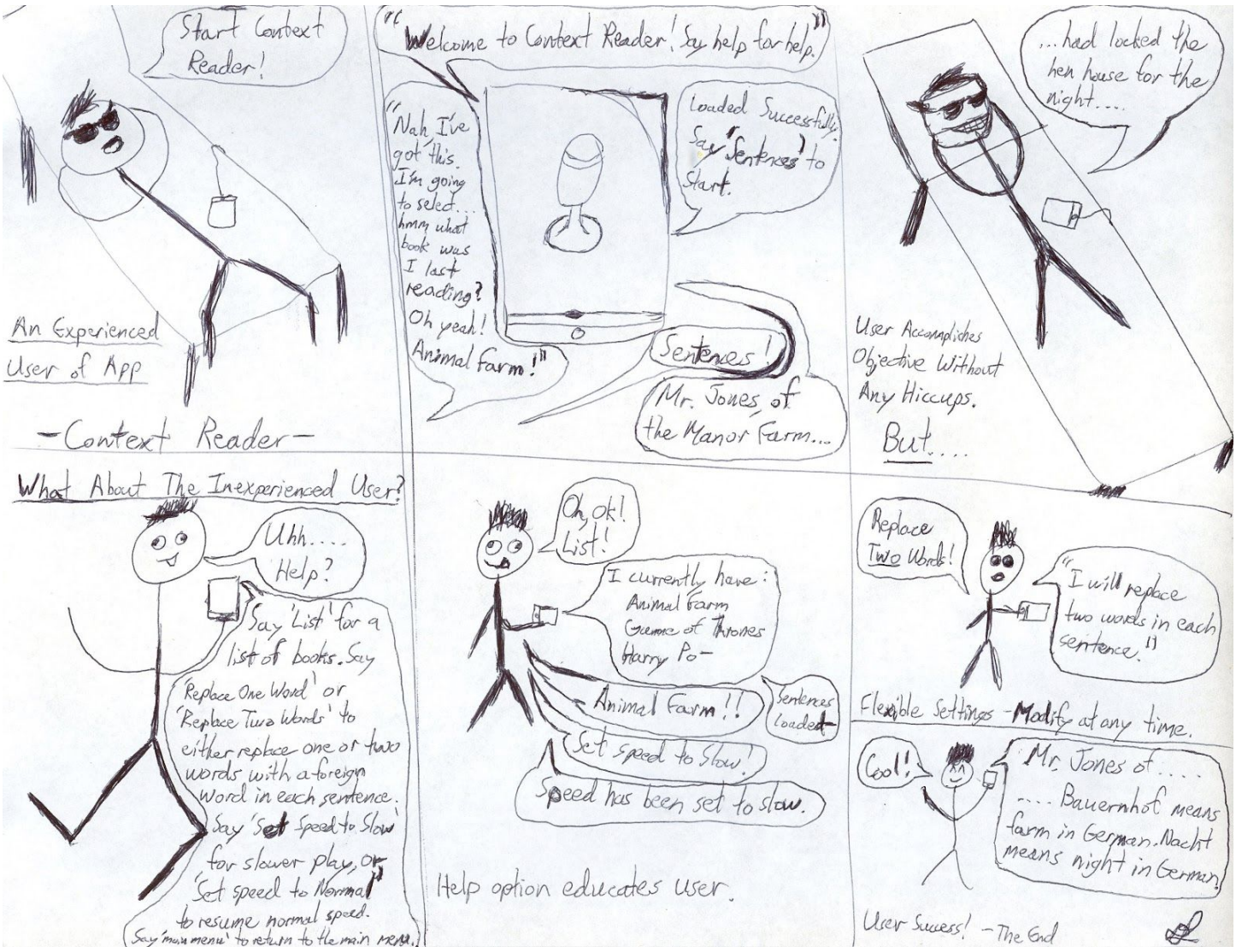


Figure 6: About Us Page (with link to our Linkedin profiles)

StoryBoard of Tasks:



Prototype Overview

Overview of Implemented UI

Let's first pretend we're using the GUI. We look at the homepage (Figure 2), and see several books to chose from, but we want to listen to a cool book whose title we know. We search the book in the search bar (Figure 3), and now we've found it. After clicking the "read" button, we're brought to a page that looks like Figure 4.

Switching to the VUI, we call up Context Reader on the Fire Tablet by saying, "Start Context Reader". Then we say the title of the book we're interested in, since we already know it in this case and don't need to list it. However, we want to change the speed setting after the book loads, but we forget how to do this, so we say "Help" to consult the VUI's assistance dialogue. We learn that to slow down the text, we say, "Set speed to slow." We also want a challenge, so we say, "Replace two words!" Two words will be replaced in each sentence, and the story will be read slowly. Now we return to our story, and begin our audio journey.

On the GUI, we follow along to the story (Figure 4), reading the words and sounding them out when we're not sure. We can tell the GUI to "repeat sentence" in order to reiterate the translation, and help reinforce our pronunciation. Finally, we reach the end (Figure 5). At this point, we decide it was so good, that we'll click the "Read From Sentence" again.

<For a more detailed voice overview, see "Appendix: Sample User Dialog">

What was left out:

Unnecessary Pages

We originally left out Help and Documentation and About Us because it was not part of the 3 main tasks that we have to implement. As a result, we decide not to include a GUI component of these pages page and only include them in the VUI component.

Android WebView

We thought about mounting our web app to Android via Android WebView to support both web and mobile platforms, but the new Android WebView had a horrendous UI design, so we decided it would be better to drop it. Our fiddling with webview was still able to produce our web app and present it via Android simulator, but the quality was so poor, we chose not to continue developing this aspect of our prototype, and instead focus on important core-components.

Upload Book to Database

We weren't able to implement the “upload a book” functionality, because when we were uploading a large text file like a book, we needed to be able to upload the text in a batch fashion. For a batch upload, we needed to split the text file into sentences, make buckets of at most 25 sentences (25 is the maximum number of items for batch addition for DynamoDB) and then send them over, but batch-adding was a technical challenge that consumed a significant amount of our time. We instead chose to focus on the important general design of our application.

An Actual Translation Implementation

We initially spent a lot of time trying to connect our code to API’s that would help us translate from English to German such as using the Pos library to detect nouns and the RapidAPI library to connect us to Google Translate. In short, we did get the translations and API functionalities to work, but there were still issues with getting the interface to run smoothly for a good user experience. This is when we decided to scrap this idea and black box it instead so that users would get the most interactive experience possible.

Having the VUI handle every single book we have listed

This should be straight-forward since we only really need to have a couple books working in the VUI to provide a prototype experience since all of them would work the same way. We may have about 9 books stored in the database but that was just to make the web GUI look prettier when it renders all of the available books on the screen. Otherwise, this would be a lot of unneeded redundancy in the VUI coding when this is

just for a prototype that should focus mainly on providing the proper user experience. As of now the VUI code should properly handle reading Game of Thrones and The Book Thief.

Black Magic (WOZ) :

1) We manually added the books to DynamoDB, but we DO utilize DynamoDB for accessing books and cover images within our application. We were originally planning on having an upload function from the GUI which would automatically pre-translate the uploaded .txt files (as mentioned in the 'What was left out' section), but we discovered that the pre-loaded books state would benefit us the most since it still achieves the effect that we want for the user to experience without spending too much time on the technicalities of our own code.
2) Translations - this goes along with our manual adding of the books since we are also manually adding translations along with the books in the database.

Code that we did not write : Dynasty Database Library for DynamoDB
Google Translate JavaScript Library
LoDash Library for Modularity
Alexa-App Javascript Package
POS Javascript Library
Node.js :)

Appendix: Technical Details

The way our GUI interacts with our VUI is very similar to what was done in Project 3, but we will still provide instructions on how to set everything up.

Since we are using React.js to create our website, we have provided you a zip file (webCode.zip) that contains the directory to launch the site. To start it, “cd” into the cs160-final-proj directory and run “npm install” to set it up and then run “npm start” to start the server. The terminal should then provide a localhost address for you to copy paste into the browser to view the website. Please email us if you have problems with this directory, so we can provide you a git clone link to the repository or another fresh directory.

Our Alexa code uses dynasty to access DynamoDB so we had to create two lambda functions for the whole setup to work since we need to zip up the code to have dynasty as a dependency. The first one would be for the database code in order for the web app to connect to the database and the second one would be the Alexa Skills Kit where we upload the zip file with all of our Alexa code. We are using Node.js for both lambda functions. Let us know if the zip file for the Alexa code does not work, so we can email you a new one. The database code should be in action.TYP and the Alexa code should be in contextread.zip

For setting up DynamoDB, we essentially have 3 Tables: **BookMetaData**, **Books**, and **Team**. The web GUI should be able to just access the data from one of our Amazon accounts without having the need to create a new table. However, for testing the VUI, the Books table would need to be in the same account, and that need it to be created. In any case, here are the schemas for each table:

- **BookMetaData** contains 3 columns: BookName, ImageURL, and Summary. BookName is the primary key which contains the name of the book. This table is used to store the images of the book covers as well as their summaries, and our web GUI will use it to display all of the available books on the main page.
- The **Books** table is rather complex, for we store books by sentences rather than entire texts. We have BookTitle as a primary key and it stores the name of the book, and we have SentenceIdx as a sort key. We need this SentenceIdx column as a sort key so that we can differentiate between different sentences for the same book (the first sentence will be indexed at 0, the second at 1, etc). The rest of the columns would be: EnglishSentence, OneWordReplaced, TwoWordsReplaced, WordOne, WordTwo, and WordThree. Thus for each

sentence we would have EnglishSentence as the sentence by default, OneWordReplaced as the sentence with one word translated, and TwoWordsReplaced as the sentences with two words translated. WordOne corresponds to the word that was translated in OneWordReplaced, and it will tell you what the translated word is in English. Likewise, WordTwo and WordThree will be the 2 words translated in TwoWordsReplaced, and they will also provide the English word for the translated word. So with this, the full schema of the table would be: BookTitle, SentenceIdx, EnglishSentence, OneWordReplaced, TwoWordsReplaced, WordOne, WordTwo, WordThree.

- **Team** contains 3 columns: Name, ImageURL, and LinkedInURL. Name is the primary key which is the name of the person. ImageURL would be the picture of the person and LinkedInURL is just the URL for the person's LinkedIn profile. This table is used by our web GUI when someone clicks 'Meet the Team'.

Appendix: Sample User Dialog

Note: <> notation refers to Alexa's actions instead of actual dialogue

User: *Start Context Reader*
Alexa: *Welcome to Context Reader! For a list of books, say list. To choose whether you want to replace one or two words, either say replace one word OR say replace two words. To set the speed of the reader, say either, set speed to slow or say set speed to normal. To quit the application, say quit or exit.*

User: *Read Book Thief*
Alexa: *Loaded successfully. Say sentences to start.*

User: *Sentences*
Alexa: *<Says the first sentence in English>. <Repeats the sentence with either one or two translations>. <Mentions which words were translated and their corresponding English words.>*

User: *Set speed to slow*
Alexa: *I have set the speed to slow. Say help for help.*

User: *Previous*
Alexa: *<Repeats everything from before but just slower>.*

User: *Set speed to normal*
Alexa: *I have set the speed to normal. Say help for help.*

User: *Replace two words*
Alexa: *I will replace two words in each sentence for you. Say help for help.*

User: *previous*

Alexa: *<Repeats everything again but just with 2 translations instead of the defaulted 1. Also the speed will be reset to normal>.*

User: *next*
Alexa: *<Will say the next sentence in the loaded book along with all the translations. The next command will continue to do this until the end of the book is reached. >*

User: *start again*
Alexa: *<Will start over from the first sentence.>*

User: *main menu*
Alexa: *Welcome to Context Reader! For a list of books, say list. To choose whether you want to replace one or two words, either say replace one word OR say replace two words. To set the speed of the reader, say either, set speed to slow or say set speed to normal. To quit the application, say quit or exit.*

Appendix: Video (7 min, 29 sec)

https://www.youtube.com/watch?v=lGw_ClRQJFk&t