2020

Language Vocabulary Builder

Vincenzo Pavano
Grand Valley State University

Language Vocabulary Builder

By
Vincenzo Pavano

A project submitted in partial fulfillment of the requirements for the degree of
Master of Science in
Computer Information Systems

at
Grand Valley State University
April, 2020

Dr. Zachary Kurmas 04/27/2020

YOUR PROFESSOR'S NAME HERE **Date**

Table of Contents

Abstract

For any student learning a foreign language, the process of looking up unfamiliar words and understanding their context can be tedious at times. The Language Vocabulary Builder aims to help self-learners conceptually understand more about the phrases they read in articles, newspapers, or lectures. Upon launching the app, a user is asked to copy and paste a body of text in the text field. Using Natural Language APIs provided by Google, known words are analyzed and are displayed in a simplistic, easy-to-understand format. Wiktionary, one of the largest open-source dictionaries, provides enhanced information about individual words.

The Android app is available on Google Play and was written natively using Kotlin. Firebase Cloud Functions and Cloud Storage were used as a back-end platform. Google Natural Language APIs and Wiktionary were both used to analyze and provide information about words inputted by users.

Introduction

Since the beginning, my life has been filled with an endless amount of curiosity regarding my heritage. As a first-generation Italian-American, I have worked my entire life to increase my understanding of the Italian language. A large portion of this includes building my vocabulary. To achieve my goal of being fluent in Italian, I often read or listen to articles from a variety of sources (e.g. newspaper articles, books, and lectures). Part of the self-teaching process has taught me of the need to constantly be translating words I am unsure of. I may read a given sentence but lose context of the sentence's meaning because of a single word I'm not sure about. My goals for the Language Vocabulary Builder were to help alleviate these issues and give users more context around words in the phrases they read. By providing users with a variety of ways to interpret words and phrases, the chances of users understanding and retaining words in their original contexts are far greater.

Background and Related Work

Several different translation apps are currently available for download on mobile app stores (e.g. Google Translate and Microsoft Translator). The Language Vocabulary Builder never set out to be just a translator app; the app strives to provide greater context around words users search for. While translating is a small piece of the app, the larger idea is to provide enhanced context behind each word. This includes, but is not limited to, a word's etymology, pronunciation, and functions as different parts of speak. Apps like Rosetta Stone and Duolingo provide users with exercises and methods of building their foundational skills in a language, but do not allow for quick, simple translations. The Language Vocabulary Builder strives to give users the best of both scenarios in a simplistic, easy-to-use way.

Program Requirements

The app begins by prompting users to enter a body of text. This can be as short as a few words to as long as a lecture. The user is also asked to select a source language from a list of languages. Currently, the Chinese, French, German, Italian, Japanese, Korean, Portuguese, Russian, and Spanish languages are supported. The source language will be used to convert the text into English.

Upon clicking the “Analyze” button, the user is presented with two different views: Based on proper nouns and all words. The “Proper Nouns” view analyzes the inputted text for known proper nouns (e.g. names and locations) and highlights them within the inputted text. For example, in the Italian sentence “Io vado a Catania stasera”, “Catania” (a location) would be bolded in the sentence. The user is presented with a bottom sheet asking to view a definition for Catania or a Wikipedia article on the location, if applicable.

The “All Words” view displays several other parts of speech (e.g. adverbs, adjectives, and verbs). Shown in a simple list, the word in the source language, definition, and part of speech are displayed in a row. Upon tapping on a word, the user is brought to the same definition screen as from the “Proper Nouns” view. This displays any known details of an individual word.

Implementation

The implementation of the project was produced from an entirely new architecture to me. The app was written on Android, but rather than using Java, which I was accustomed to, I tried using Kotlin. With a growing interest in the Android community, Kotlin is poised to be a modern language alternate to Java. Backed by Google, and fully supported in Android Studio, I was determined to try it out for myself. I enjoyed learning about Kotlin’s many benefits over Java, while still producing a fully native Android app. I was familiar with many of the libraries used in the project, as well as the Model-View-Presenter architecture pattern used, but this was my first take at the Kotlin programming language.

The API layer was written using Node.js and hosted on the Firebase platform. The API calls the Google Natural Language and Wiktionary APIs to retrieve data, while storing results in a Firebase Cloud Storage database. I was new to both Node as an API and Firebase as a platform. Being a Web/Mobile Developer in my full-time job, I am familiar with JavaScript and decided to include TypeScript in the Node app. This was a great idea in theory, however much of Firebase’s documentation is written in straight JavaScript. This presented several challenges while trying to follow documentation, but I was able to resolve all issues. We typically use C#/.NET as our back-end technology at work, so that was very strongly considered for this project’s API. My reasons for not picking .NET Core primarily revolved around the exploratory nature of the new project and costs for hosting the app in the long-term. Despite being able to host .NET Core apps on Linux servers, the pricing for hosting a .NET Core app is far higher than a Firebase project. If the app saw any future beyond the semester, it would be wise to start with a Firebase project.

Results, Evaluation, and Reflection

Overall, the I was happy with the results of the project. I received lots of praise from professors and colleagues around the final results of the app shown during the presentation. However, I do believe the app could have gone further throughout the semester with more concise planning. We ran the project in a Scrum-based fashion, which benefitted us by given us bi-weekly deadlines. There were a couple of sprints that produced in low results, which put the project behind towards the middle of the semester. This was difficult to overcome and was attributed to several reasons:

Planning/SDLC

Following a proper Software Development Lifecycle (SDLC), I planned out work at the beginning of the project during the "planning" phase. The final results of the app were not conceived during the planning phase of the project. While this is the nature of Agile Development, I do believe I should have had more concrete architecture and implementation ideas at the end of the planning phase. It would have also behooved me to study the new architecture more in-depth first, rather than just jumping into the projects.

Firebase

For most of the semester, I struggled with Firebase and Firebase Cloud Storage (a NoSQL-based database). With this being my first Firebase project, as well as NoSQL database implementation, the back-end layer of the project took several sprints to iron out. It would have been far easier to implement a .NET Core API, which I am accustomed to, but I wanted challenged myself with technology that I know is well-respected in the development community today. At a couple points in the sprint, I highly considered rewriting the API layer to be a .NET Core app. I was able to overcome some of the challenges in the Node app by introducing proper architecture patterns (Controller, Service, Subscriber) to separate out logic. This helped pave the way forward for easier feature implementation.

Caching

To optimize network calls, I decided to implement a caching strategy in the API layer. Rather than calling Google's Natural Language and Wiktionary's APIs, which charge per network call, I wanted to save as much information as I could in the Cloud Firestore database. Consuming cached data would also result in quicker network calls for the mobile app, which was reliant on the availability of all services. I initially approached the caching solution with that of a relational database. Since I had never worked with NoSQL before, it took a mind shift adjustment to fully understand the benefits of NoSQL and how my implementation was wrong. While simple, I did not consolidate data in an effective way optimized for NoSQL. This resulted in data being selected and inserted in an inconsistent way, which took time each sprint to understand and follow the logic. Towards the end of the project, I halted the caching strategy and decided to postpone all new caching until after the end of the semester.

Conclusions and Future Work

In addition to several performance-related bugs found in the app, there are several features which were not implemented due to the time constraints in the semester.

Data Sources

Despite the app supporting nine languages, several other popular languages were omitted from the initial list. This is due to constraints in Google's Natural Language APIs. If additional data sources were introduced, more in-depth, reliable descriptions of words and phrases could be achieved. The Collins Dictionary is a multilingual dictionary which includes a REST API, and initial research proves to return consistent data. This could potentially be a successor to Wiktionary.

Word and Phrase Analyzation

Additionally, the engine which drives word analyzation could be improved. Google's Natural Language APIs are helpful, but an ideal app would include a word's meaning *in the context of the sentence*, rather than individually. The app does this currently, to an extent, but it could be greatly improved.

Caching

With all I learned from trying to implement a NoSQL-based caching strategy, I would delete the entire database and retry it from scratch. The idea of caching is beneficial but trying the implementation again would result in a more logically organized database.

iOS App

The decision to write the app as a native Android app comes with many advantages, but also with disadvantages. One of the disadvantages of choosing a native app architecture came the inability to produce an iOS app during the semester. Failing to launch an iOS in the future would alienate much of the population, especially in the United States. It would not be difficult to launch an iOS app, but it would take time to write once a more solidified plan was implemented.

Bibliography

Dr. Kurmas was a large driving force behind the initial idea for the app. I approached him with my own idea for a language-based app, but many of his initial ideas (e.g. using Wiktionary to translate large bodies of text) were used in the final app. I took the project and made it my own, but combining our ideas together produced the final app.

Additionally, Google's Natural Language and Wiktionary's APIs were instrumental to the success of the app. They handled the heavy lifting behind the analyzation and translation services the app provides.

Appendices

- Firebase
- Google Natural Language API
- Language Vocabulary Builder (pending release on Google Play)
- Wiktionary API