Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Aplicativo para construção de vocabulário no Idioma Japonês

Rafael Simões Shimabukuro

Monografia Final

mac 499— Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. João Eduardo Ferreira

São Paulo
30 de Janeiro de 2021

Resumo

Rafael Simões Shimabukuro. **Aplicativo para construção de vocabulário no Idioma Japonês**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

O objetivo deste trabalho foi desenvolver um aplicativo de expansão de vocabulário da Língua Japonesa, com enfoque nas palavras com as quais o usuário tem mais contato no seu cotidiano. O contato de brasileiros com a língua japonesa acontece geralmente por meio do entretenimento, assim, será usado como vocabulário do aplicativo palavras que se mostram relevantes no contexto de filmes e animes japoneses.

Um dos aspectos importantes da construção de vocabulário é a memorização de longo prazo, que pode ser difícil, já que com o passar do tempo esquecemos de coisas que não são utilizadas com certa frequência. Para contornar esse problema, o aplicativo desenvolvido permite ao usuário selecionar, dentre uma lista, produções com que tem maior contato e mostra palavras relevantes na obra selecionada, assim, ao assisti-la o usuário reconhece o conteúdo estudado, isso auxilia a memorização de uma forma mais fácil e natural.

Ainda nesse aspecto, uma técnica que ganhou notoriedade, pelo auxílio na memorização, é a de repetição espaçada. Nela, palavras e suas traduções são colocadas em flashcards, os cards com palavras novas e/ou mais "difíceis" serão repetidos mais frequentemente, enquanto, os com palavras mais consolidadas terão uma frequência menor. Essa técnica de repetição espaçada será implementada no aplicativo para facilitar ainda mais a construção de vocabulário.

Palavras-chave: aprendizado. vocabulário. memorização. japonês. idioma.

Abstract

Rafael Simões Shimabukuro. **An application for japanese vocabulary building**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

The goal of this project was to develop a japanese vocabulary building app, focusing on words that the user has more contact in their daily lives. In general, the contact of Brazilians with the Japanese language occurs through entertainment, therefore, there will be selected, as vocabulary, words deemed important within the Japanese films and anime context.
One of the important aspects of vocabulary building is long-term memorization, which can be difficult, since as time passes we forget things that are not often used. To get around this problem, the developed application allows the user to select, from a list, films that he has the most contact with, then shows relevant words in the selected picture, thus, when watching it, the user recognizes the studied content, that supports memorization in an easier and more natural manner.
A technique that has gained notoriety in supporting memorization is spaced repetition. In this approach words and their translations are placed on flashcards, cards with new and/or more difficult words will be repeated more frequently, while those with consolidated words will have a lower frequency. The described spaced repetition technique will be implemented in the application to further facilitate the vocabulary building process.

Keywords: japanese. vocabulary. memorization. language. learning.

Lista de Figuras

Sumário

Capítulo 1

Introdução

1.1 Motivação e Objetivo

A motivação do desenvolvimento, teve origem no grande interesse popular pelas produções japonesas e, consequentemente, na língua japonesa. Embora existam aplicações que facilitam a expansão de vocabulário e o estudo do idioma, elas não consideram a origem do interesse pela língua e tampouco o contato cotidiano do usuário.

Entendendo esse contexto, o aplicativo alia entretenimento com a construção de vocabulário, tornando o aprendizado engajante e recompensador. Mais concretamente, espera-se que o entendimento, mesmo que de pequenas partes, de uma produção, além de ajudar a fixar o vocábulo, pode contribuir para uma experiência de aprendizado mais recompensadora. Enquanto decorar palavras e não vê-las em uso, pode ter um efeito contrário, que muitas vezes desencoraja o aprendizado.

Além da importância desse alinhamento do aprendizado com o entretenimento, outro fator relevante é a memorização do vocabulário a longo prazo. Esse tema é estudado há bastante tempo, e já existem diversas técnicas funcionais, com objetivo de maximizar a taxa de memorização. Tais técnicas, já eram usadas e testadas antes do surgimento do smartphone, mas agora, pode-se implementá-las em aplicativos, tornando-as muito mais práticas ao usuário, que pode se beneficiar mesmo sem conhecer a fundo as técnicas utilizadas. Essa praticidade aliado ao fato de que os smartphones estão cada vez mais presentes para a população geral podem permitir obter ótimos resultados com o uso regular do aplicativo, mesmo que em curtas janelas de tempo.

1.2 Organização dos capítulos

Este documento está dividido em quatro partes principais:

- Nos capítulos 2 e 3 são apresentados conceitos necessários e decisões pré desenvolvimento.
- Nos capítulos 4, 5 e 6 é mostrado o processo de obtenção e preparação de dados para o aplicativo.
- Nos capítulos 7, 8, 9 e 10 são explicadas decisões de desenvolvimento e funcionalidades do aplicativo.
- No capítulo 11 o trabalho é concluído e seu futuro é discutido.

Capítulo 2

Idioma Japonês

2.1 Conceitos necessários do idioma japonês

O idioma japonês contém dois alfabetos silábicos e um sistema de escrita visual e simbólica. O Kanji, de origem chinesa, foi adotado para ser usado no sistema de escrita da língua japonesa. Cada Kanji, possui um significado ou uma ideia por si só, e pode gerar outros significados quando junto a outros Kanjis ou sílabas. Kanjis também podem possuir diversas leituras diferentes, alguns continuam com a leitura de origem Chinesa, outros, adotaram a pronúncia da palavra da língua japonesa correspondente. A leitura de um Kanji, não é algo trivial, mas algo que deve ser memorizado em diversos contextos diferentes, já que a pode mudar de acordo com outros kanjis e sílabas pareados a ele.

O Hiragana e o Katakana, ambos são alfabetos silábicos, ou seja, cada "letra" simboliza uma sílaba. O Katakana é usado, principalmente, na escrita de palavras estrangeiras e em alguns casos específicos, como em onomatopeias. Já o Hiragana, é o alfabeto silábico padrão, usado em conjunto com Kanjis, na escrita de palavras que não possuem ideogramas e também para indicar a leitura de Kanjis menos comuns ou os presentes em livros infantis, cujo público ainda está aprendendo os significados e leituras.

Como o aplicativo desenvolvido é voltado ao aprendizado, será disponibilizado ao usuário, tanto a escrita em Kanjis, como em Hiragana e no alfabeto romano. Assim, com o tempo, pode-se aprender as leituras de diversos kanjis, e fixar qual sílaba cada Hiragana representa.

Capítulo 3

Obtenção de vocabulário

Um dos diferenciais do aplicativo é trazer vocabulário relevante em produções populares, com que o usuário terá mais contato no cotidiano. Isso pode ser dividido dois subproblemas:

Um dos diferenciais do aplicativo é trazer vocabulário relevante em produções populares, com que o usuário terá mais contato no cotidiano. Isso pode ser dividido dois subproblemas:

3.1 Seleção de quais produções serão abrangidas pelo aplicativo

Para definir quais produções são abrangidas, foram usados dados do site "myanimelist.net" (Myanimelist, 2020), uma enorme rede social de animes e mangás, que contém uma base de dados bastante completa, com avaliações de milhares de membros sobre cada produção. No site, os usuários também podem adicionar produções a suas listas de já assistidas e, com base nessas informações é gerada uma lista de animes e filmes mais populares. Essa lista foi usada como base para selecionar 10 filmes e 10 animes mais populares, pelos quais o usuário tem grande chance de se interessar.

3.2 Como obter os vocábulos de uma produção

Com as produções selecionadas, é necessário uma solução para obter quais são as palavras usadas na obra, uma opção bastante prática, é usar arquivos de legendas, de cada produção, na língua japonesa. No entanto, encontrar tais arquivos pode ser difícil. Assim, algumas das produções, cuja legenda não foi encontrada, tiveram que ser removidas. Quando encontradas estavam disponíveis em "kitsunekko.net" (Kitsunekko, 2020), um site que reúne legendas no idioma japonês de diversos filmes e animes.

Capítulo 4

Processamento de texto

4.1 Conceitos necessários da Computação: processamento de linguagem natural

O processamento de linguagem natural é a área da computação que visa entender a linguagem dos seres humanos, ou seja, como processar e analisar dados como áudios e textos. O objetivo é permitir que o computador "entenda" o contexto. Neste trabalho foram utilizadas algumas técnicas comuns nessa área. São elas:

4.1.1 Tokenização

A tokenização é a separação das palavras de um texto. Em idiomas como o portugues, é uma tarefa trivial, já que as palavras são separadas por espaços. Porém, em um idioma como o japonês, não ocorre essa separação. Para conseguir definir o começo e final de uma palavra é necessário ter um conhecimento da língua. O que torna essa tarefa bastante complexa.

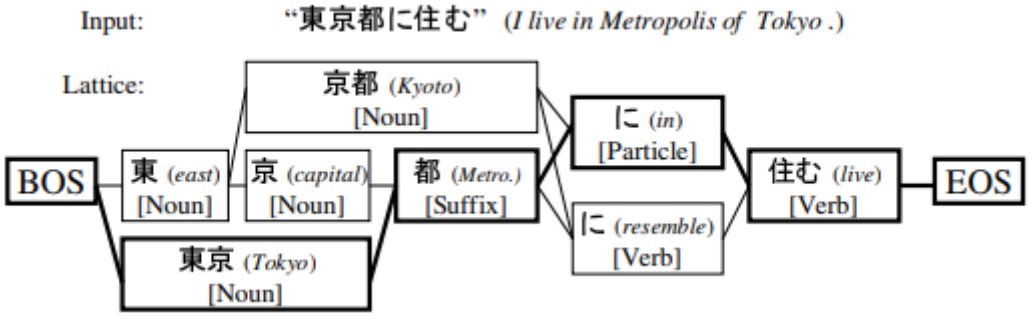

Figura 4.1: *Tokenização (Kudo, 2004)*

A Figura 4.1 acima ilustra o problema de tokenização da língua japonesa. Para definir o início e fim de cada palavra é preciso considerar todas as possíveis aglutinações de Kanjis e Hiraganas, e somente depois, definir qual o caminho correto (no caso, o caminho

em negrito). No exemplo da imagem podemos ver o Kanji '京' que poderia fazer parte da palavra '東京' (Tokyo) ou de '京都' (Kyoto).

4.1.2 Lematização

A lematização é a deflexão das palavras obtidas pela tokenização, retornando-as à forma base ou lema (encontrada em dicionários). Essa tarefa é essencial para este trabalho, já que, para definir a relevância de um vocábulo no contexto de construção de vocabulário, é necessário saber quantas vezes a palavra aparece, no anime ou filme, em qualquer uma das suas flexões. Como exemplo, as palavras come, comeu e comendo, devem todas contribuir para a relevância do verbo comer, que é o lema.

4.1.3 Classificação morfológica

Classificar morfologicamente as palavras, também é importante para definição de quais são relevantes, na construção de vocabulário. O motivo é que, alguns vocábulos, embora apareçam muito, podem não ter um significado claro, individualmente, não seriam bem traduzidos ou não são relevantes no contexto de aprendizado, como no caso de partículas e nomes próprios. Tal classificação, também abre um grande leque de futuras adições ao aplicativo, já que podemos tratar morfologias de forma individual, uma possibilidade por exemplo é, no caso de ser um verbo, mostrar algumas das possíveis conjugações.

4.2 MeCab

Uma biblioteca que facilitou o desenvolvimento, na área de processamento de linguagem natural, foi a ferramenta computacional MeCab (MeCab, 2013). Essa ferramenta provê a segmentação de texto e análise morfológica no idioma japonês.

A biblioteca utiliza Conditional random fields (CRFs) (Lafferty, 2001) e alguns dicionários como dados para treinamento. Tais dicionários são compilados de informações sobre as palavras e classificações morfológicas. Esse modelo preditivo baseado em CRFs é adequado quando a informação do contexto afeta a predição atual. É portanto muito utilizado na classificação morfológica de textos, pois permite definir a morfologia das palavras baseado no contexto (na frase que se encontra, palavras que vem antes e depois), não colocando muito peso em suposições individuais.

Porém, na língua japonesa há um obstáculo a mais. Enquanto em outras línguas como o português, dada uma frase, as palavras são separadas por espaços, é sabido a quantidade de palavras e a incógnita é apenas a morfologia. Na língua japonesa é necessário identificar tanto a segmentação das palavras, quanto a morfologia. Como é mostrado na Figura 4.1 podemos notar vários caminhos de possibilidades, que podem ter comprimentos diferentes. Mesmo assim, há soluções para uso de CRFs na análise morfológica no idioma japonês (Kudo, 2004), como o implementado no MeCab.

4.3 Resultado

Toda a parte de processamento de texto foi desenvolvida usando Python e o MeCab. Após o processamento, a partir de uma frase, com as técnicas e ferramentas citadas neste capítulo, foram obtidos, para cada palavra, seu lema e sua morfologia.

Exemplo de output gerado pela palavra 軍隊 (guntai):

軍隊 軍隊 名詞-普通名詞-一般

Os campos são separados por tabulações. O primeiro é a palavra na forma original, encontrada na produção. O segundo é o lema da palavra. E o terceiro a morfologia, com subclassificações separadas por traços “-”.

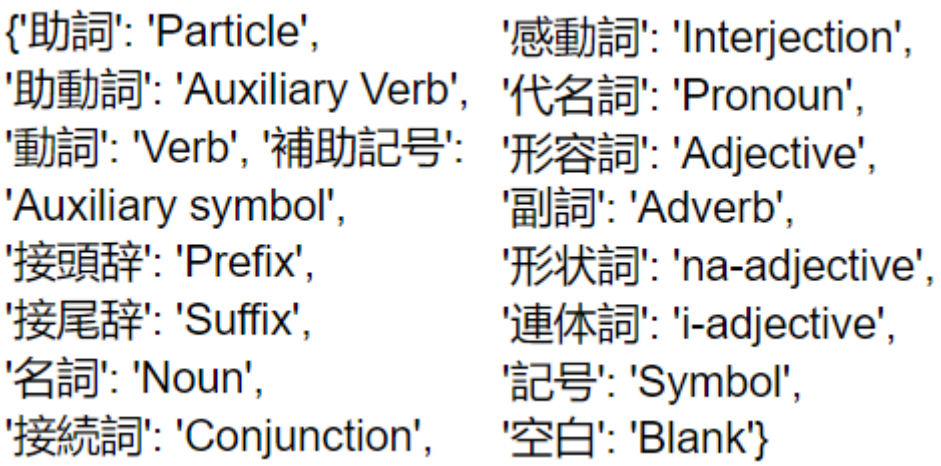

Figura 4.2: *Possíveis morfologias resultantes*

A imagem da Figura 4.2 apresenta as morfologias que podem ser identificadas e a suas traduções ao inglês. No caso de substantivos (nouns), é preciso também verificar sua subclassificação, caso seja ’固有名詞’ (koyūmeishi), significa que se trata de um substantivo próprio e, portanto, não é muito relevante no contexto de aprendizado, pois normalmente é um nome de algum personagem.

Capítulo 5

Leitura e significado dos lemas

5.1 Sentidos e tradução

No início do projeto, a tradução foi feita utilizando a *API do Google Translate* (Google, 2020) para obter as traduções de palavras. No entanto, três problemas surgiram. O primeiro, é que o *Google tradutor*, embora seja amplamente utilizado para tradução de textos e frases, quando se trata de palavras isoladas tem um resultado pior, e não muito confiável, já que não consegue saber o contexto em que a palavra foi usada. O segundo problema, é que apenas uma possibilidade de tradução é dada, e no aplicativo, queremos mostrar algumas alternativas, isso dá um entendimento mais completo da palavra. E, por fim, muitas palavras não tem uma tradução em nosso idioma, é necessário explicar seu sentido, algo que o *Google tradutor* não é capaz de oferecer.

A solução encontrada foi utilizar a versão digital de um dicionário, porém, não foi encontrado nenhum disponível do idioma japonês para o português. Como alternativa, é utilizado um dicionário digital do japonês para inglês, o *JMdict* (JMdict, 2020), para encontrar os sentidos e leituras da palavra e, em seguida, o sentido é traduzido ao portugues por meio da *API do Google Translate*. Com esse método o dicionário fornece, em geral, mais de um sentido com, muitas vezes, alguns sinônimos, como pode-se ver na Figura 5.1 abaixo:

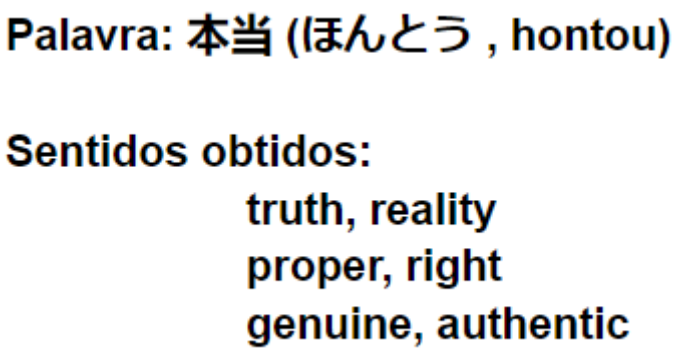

Figura 5.1: *Exemplo de definição do dicionário*

Assim, ao traduzir cada um dos três sentidos individualmente, o tradutor terá mais de uma palavra em que se basear e caso ocorram erros em algum dos sentidos, ainda existem outros para explicar a palavra ao usuário. Outro benefício desse método é que em palavras sem traduções diretas, ou que necessitam de uma definição mais detalhada, o dicionário possui a explicação enquanto o *Google translate* teria dificuldade na tradução.

5.2 Problemas na tradução

Embora o método de obtenção de sentido e sua tradução seja bastante eficiente, há alguns problemas que podem ocorrer. Como no caso de sufixos e prefixos, em que, muitas vezes, no dicionário é encontrado como sentido o sufixo/prefixo correspondente em inglês, tornando muito difícil a tradução do inglês para portugues pela *API do Google Translate*.

Exemplo: Lema: 不
Sentido: un-, non-

Para as pessoas que dominam o idioma em inglês é possível entender o significado desse prefixo no exemplo, porém, sua tradução para o português pelo *Google translate* é muito dificultada.

Em verbos, também foi encontrado um problema, o sentido de verbos dado pelo dicionário em inglês costuma começar com “to ”, por exemplo *To eat, To drink*. Isso está correto, é assim que se representa o infinitivo em inglês, porém, “to” também pode significar “para”, assim, muitas vezes, o *Google Translate* acaba traduzindo com a palavra “para” antes do verbo, por exemplo: “para comer” ao invés de “comer”. Para tratar esses casos foram consideradas duas alternativas: remover o “to” antes da tradução de verbos ou remover o “para” depois da tradução. A primeira opção não é boa pois sem o “to” a tradução acaba, muitas vezes, aparecendo em uma forma flexionada do verbo ao invés do infinitivo. Já a segunda opção, não apresentou significativos problemas e foi a utilizada.

5.3 Leituras

Por fim, com o dicionário também obtêm-se algumas leituras possíveis para a palavra em Hiragana, como também queremos a leitura no alfabeto romano é necessário realizar a conversão. A parte mais complexa da obtenção da leitura é a transformação dos Kanjis para Hiragana, que foi já obtida procurando a palavra no dicionário. A transição do Hiragana para o alfabeto romano foi efetuada com auxílio da biblioteca romkan (Romkan, 2013) em Python, utilizando as seguintes relações apresentadas na Figura 5.2.

Vogais e monógrafos (*Seion*)					Dígrafos (*Youon*)		
あ *a*	**い *i***	**う *u***	**え *e***	**お *o***	や (***ya***)	ゆ (***yu***)	よ (***yo***)
か *ka*	き *ki*	く *ku*	け *ke*	こ *ko*	きゃ *kya*	きゅ *kyu*	きょ *kyo*
さ *sa*	し *shi*	す *su*	せ *se*	そ *so*	しゃ *sha*	しゅ *shu*	しょ *sho*
た *ta*	ち *chi*	つ *tsu*	て *te*	と *to*	ちゃ *cha*	ちゅ *chu*	ちょ *cho*
な *na*	に *ni*	ぬ *nu*	ね *ne*	の *no*	にゃ *nya*	にゅ *nyu*	にょ *nyo*
は *ha*	ひ *hi*	ふ *fu*	へ *he*	ほ *ho*	ひゃ *hya*	ひゅ *hyu*	ひょ *hyo*
ま *ma*	み *mi*	む *mu*	め *me*	も *mo*	みゃ *mya*	みゅ *myu*	みょ *myo*
や *ya*		ゆ *yu*		よ *yo*			
ら *ra*	り *ri*	る *ru*	れ *re*	ろ *ro*	りゃ *rya*	りゅ *ryu*	りょ *ryo*
わ *wa*				を *wo*			
				ん *n*			
が *ga*	ぎ *gi*	ぐ *gu*	げ *ge*	ご *go*	ぎゃ *gya*	ぎゅ *gyu*	ぎょ *gyo*
ざ *za*	じ *ji*	ず *zu*	ぜ *ze*	ぞ *zo*	じゃ *ja*	じゅ *ju*	じょ *jo*
だ *da*	ぢ *dji*	づ *dzu*	で *de*	ど *do*	ぢゃ *dya*	ぢゅ *dyu*	ぢょ *dyo*
ば *ba*	び *bi*	ぶ *bu*	べ *be*	ぼ *bo*	びゃ *bya*	びゅ *byu*	びょ *byo*
ぱ *pa*	ぴ *pi*	ぷ *pu*	ぺ *pe*	ぽ *po*	ぴゃ *pya*	ぴゅ *pyu*	ぴょ *pyo*

Geminados (*Sokuon*)				
(a)	***(i)***	***(u)***	***(e)***	***(o)***
っか *kka*	っき *kki*	っく *kku*	っけ *kke*	っこ *kko*
っさ *ssa*	っし *sshi*	っす *ssu*	っせ *sse*	っそ *sso*
った *tta*	っち *tchi (cchi)*	っつ *ttsu*	って *tte*	っと *tto*
っば *bba*	っび *bbi*	っぶ *bbu*	っべ *bbe*	っぼ *bbo*

Figura 5.2: *Relações do Hiragana com alfabeto romano, (WIKIPEDIA, 2020)*

Capítulo 6

Estrutura base do Software e Exportação de dados

Em ambiente Python, as legendas de cada produção foram quebradas em linhas e enviadas para o processamento, é retornada então, uma lista de lemas das palavras e seus respectivos dados. A estrutura principal do programa é um dicionário em que ficam guardados para cada lema, o sentido em inglês, a tradução, a leitura, um contador geral de ocorrências do lema, suas morfologias com contadores associados à elas, e as produções em que a palavra aparece com contadores associados. Pode-se observar essa estrutura na Figura 6.1.

```
dicionario [lema] = [   sentidos da palavra,
                        leituras da palavra,
                        {morfologia1: contador1, morfologia2: contador2},
                        {produção1: cont1, produção2: cont2},
                        contador geral,
                        tradução da palavra]
```

Figura 6.1: *Estrutura do dicionário*

Com esses dados é possível determinar qual a morfologia mais comum de uma palavra e quantas vezes ela aparece em uma produção específica.

6.1 Exportação de dados

Os dados gerados em python, pelo processamento e tradução de palavras presentes nas produções selecionadas, serão utilizados pelo aplicativo para celular. Assim, é preciso exportá-los. Serão exportados 3 arquivos *JSON*, com a estrutura mostrada em Figura 6.2.

- Dados das palavras(lemas), no formato**:**

{

lema: [*leituras, morfologias, produções, tradução*],
lema2: [*leituras, morfologias, produções, tradução*],
…

}

-Dados de cada produção, no formato:

{

produção: [*Titulo da produção, grupo da produção*],
produção2: *[Titulo da produção, grupo da produção*],
…

}

-1000 lemas mais relevantes por produção, no formato:

{

produção :{**lema**: *número de aparições*, **lema2**: *número de aparições* ....},
produção2 :{**lema**: *número de aparições*, **lema2**: *número de aparições* ....},
.
.
.

}

Figura 6.2: *Estrutura dos arquivos JSON exportados*

Somente serão exportados os dados de palavras relevantes, ela deverá pertencer a uma das morfologias aceitas. As morfologias aceitas são verbo, substantivo comum, interjeição, pronome, adjetivo e advérbio. Serão rejeitados nomes próprios, partículas, verbos auxiliares, prefixos, sufixos e símbolos. Essas morfologias são rejeitadas pois não tem um significado próprio, não fazem sentido no contexto de expansão de vocabulário ou por problemas que podem ocorrer no processo de tradução.

Capítulo 7

Estrutura do aplicativo mobile

7.1 React Native

O React Native (Native, 2020) tem como objetivo a criação de aplicativos mobile multi plataformas (iOS e Android). Ele é um framework implementado com a abordagem de software livre, escrito em JavaScript e desenvolvido pelo Facebook. Os aplicativos são programados usando o *Cascading Style Sheet* (CSS) e o JavaScript XML (JSX) que mescla características do JavaScript com *Extensible Markup Language* (XML). Uma das principais vantagens do seu uso é que, alternativamente a outros frameworks, que utilizam WebViews para a renderização, o JavaScript do *React Native*, é utilizado como um meio de comunicação com os componentes nativos do celular, o que resulta para o usuário final uma experiência mais fluida e agradável.

7.2 Realm

O Realm (Realm, 2020) é uma base de dados construída para desenvolvimento de computação móvel que utiliza objetos do JavaScript que são mapeados em uma base de dados. Ele consegue prover uma linguagem simples com ótima performance. Quando comparado a base de dados alternativas como SQLite, o *Realm* consegue uma performance muito melhor em plataformas de computação móvel , com mais capacidade de consultas por segundo. Podemos ver um exemplo simples de uso do *Realm* em Figura 7.1

```
class Dog {}
Dog.schema = {
    name: 'Dog',
    properties: {
        name: 'string',
        age: 'int',
    }
};

let realm = new Realm({schema: [Dog]});

realm.write(() => {
    realm.create('Dog', { name: 'Rex', age: 3 });
});
```

Figura 7.1: *Exemplo de schema de base de dados Realm, realm.io*

7.3 Base de dados e organização

Os dados gerados e exportados pelo software em Python são importados para uso pelo aplicativo. Após a importação e interpretação dos dados, eles serão armazenados em uma base de dados Realm, como mostrado nas Figuras 7.2 e 7.3.

```
const Cards_DBSchema = {
    name: 'Words',
    primaryKey: 'key',
    properties: {
      key:  'string',
      translation: 'string',
      reading: 'string',
      morph: 'string',
      level: 'int',
      dificulty: 'float',
      last_interval: 'float',
      iteration: 'int',
      review_time: 'int',
    }
```

Figura 7.2: *Estrutura do realm*

Como estrutura principal do aplicativo, temos a tabela com as palavras mais relevantes encontradas nas produções, mostrada em Figura 7.2, em Realm, cada objeto nessa tabela representa um *flashcard* e tem diversos campos. *Key* é o lema da palavra, assim, com o lema podemos obter informações como a tradução do sentido, a leitura e a morfologia. Os campos *level*, *dificulty*, *last-interval*, *iteration* e *review-time* serão usados no algoritmo de repetição espaçada para mostrar os *flashcards* e serão melhor explicados no capítulo 9.

```
const Freq = {
  name: 'Freq',
  properties: {
    word: 'string',
    freq: 'float'
  }
```

```
const Prods_DBSchema = {
    name: 'Prods',
    primaryKey: 'key',
    properties: {
      key:  'string',
      prod_type:  'string',
      prod_name:  'string',
      freqs: 'Freq[]'
    }
};
```

Figura 7.3: *Estrutura do realm*

Também é construída uma tabela com informações de cada produção, que contém objetos no formato mostrado em *"Prods-DBSchema"* na Figura 7.3, nela é armazenado o id da produção como key, seu título, seu tipo (animes ou filmes), e uma lista com pares de palavras e suas respectivas frequências *"Freq"*. Assim, será possível ordenar as palavras por frequência em que aparecem em uma produção específica.

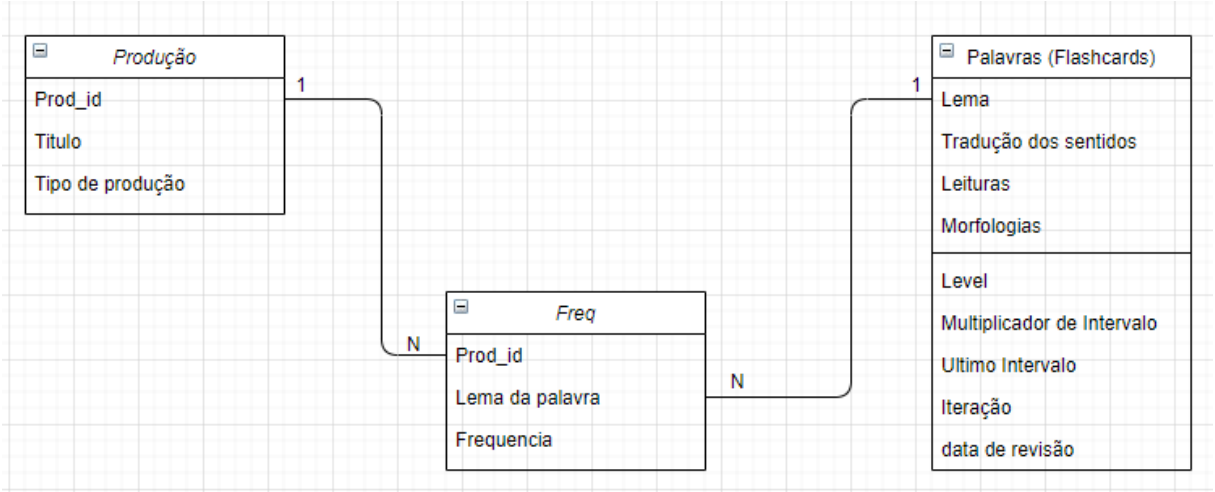

Figura 7.4: *Representação da base de dados*

A tabela representada como Freq, faz a ligação entre uma produção e uma palavra, e contém informação sobre a frequência. Cada produção pode estar ligada a diversos *flashcards*, e cada *flashcard* pode ter diversas produções associadas. Reunir as informações do *flashcard* em uma tabela separada, ao invés de, cada produção possuir uma lista de *cards*, é importante, já que, no aplicativo, interações com um *flashcard* vinculadas a uma produção, devem afetar próximas interações inter relacionadas .

Exemplo: Se um usuário já "masterizou"um *flashcard* com a palavra “分かる” pertencente a produção “Your Name”, é correto afirmar que ele terá conhecimento da palavra em qualquer outro Anime ou Filme e caso o *flashcard* continue se repetindo ao usuário, a sua experiência será prejudicada.

Capítulo 8

Interface do aplicativo mobile

8.1 *Navigator*

Em React Native o *Navigator* é o responsável pela transição de telas, neste aplicativo será utilizado o *Stack Navigator*, que organiza os componentes na estrutura de pilha, permitindo voltar à tela anterior, se houver. Por meio dele também são passados os parâmetros para comunicação entre diferentes componentes.

8.2 Telas

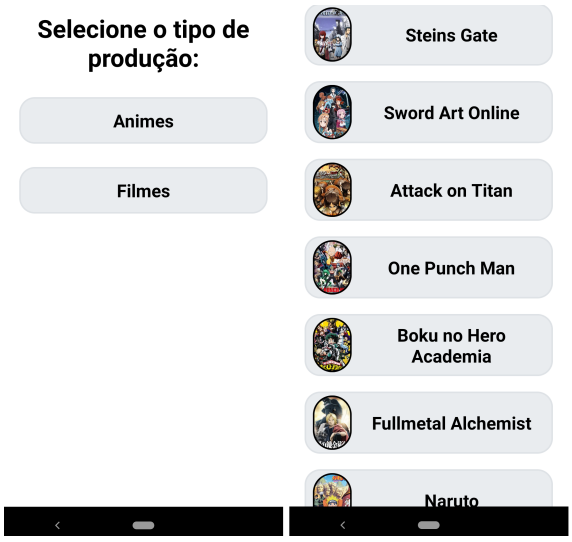

Figura 8.1: *Duas primeiras telas de seleção*

As duas telas iniciais (Figura 8.1) são listas que permitem a rolagem e cliques para seleção do tipo e da produção. A primeira abre a base de dados Realm e solicita as produções disponíveis, depois, ela lista quais tipos diferentes existem, assim ao serem adicionados novos tipos ou novas divisões, como músicas, a tela se adapta facilmente.

Com o clique em algum dos itens, é chamado o Navigator para transacionar à segunda tela, dando como parâmetro o tipo de produção selecionado.

A segunda tela recebe o tipo de produção, o solicita ao Realm e obtém uma lista que é mostrada ao usuário. Com o título da obra, também é mostrada sua respectiva imagem, previamente baixada e endereçável tendo o nome da produção. Ao selecionar algum item ocorre a transição para a tela específica da obra.

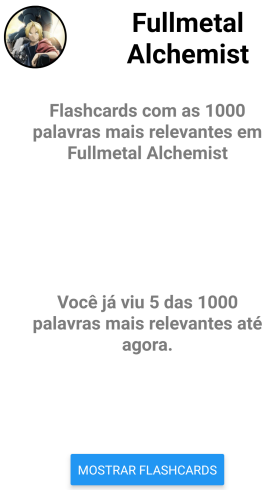

Figura 8.2: *Tela de informações da produção*

Na tela ilustrada pela Figura 8.2 é solicitado ao Realm a lista de palavras e frequências na produção, para cada palavra é solicitado o seu *flashcard* e verificado seu estado (se o *card* já foi visto). Em seguida mostra-se ao usuário a quantidade de palavras disponíveis e quantas ele já viu anteriormente. Ao acionar o botão de mostrar *flashcards* somos direcionados para tela central do aplicativo.

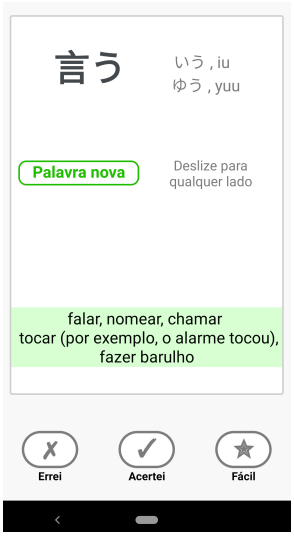

Figura 8.3: *Tela de exibição de flashcards*

Na tela ilustrada pela Figura 8.3 são mostrados os *flashcards* pertencentes a produção, a ordem em que são mostrados é explicada em 9. Há duas possibilidades nessa tela, uma é aparecer um *card* que já foi visto anteriormente, outra, um com vocabulário novo. Caso seja uma palavra nova, a tradução será mostrada ao usuário, que poderá deslizar o *card* para qualquer um dos lados ou pressionar qualquer botão para seguir a próxima palavra conforme ilustra a Figura 8.4.

Figura 8.4: *Fluxo de flashcards e Botões disponíveis*

Caso a palavra já tenha sido vista anteriormente, a resposta não será mostrada. O usuário deve pensar em qual é a tradução/significado e pressionar o botão para ver a resposta correta. Em seguida, ele deve informar ao aplicativo se acertou (deslizando para direita), errou(deslizando para esquerda), e, caso tenha acertado com facilidade, deve deslizar para cima. Essas interações também podem ser executadas pelo botão correspondente.

8.3 Deslize de um *card*

Para possibilitar essa interação, foi utilizada a biblioteca Animated (Animated, 2020) do React Native. A biblioteca foi desenvolvida para tornar animações fluídas mais fáceis de serem construídas. Ela permite a declaração de relacionamentos entre inputs e outputs, transformações configuráveis e métodos que controlam animações baseadas em tempo.

O *flashcard* mostrado em primeiro plano é uma Animated View com um "*PanResponder*"e uma posição "Animated.ValueXY" associados. O "*PanResponder*" possui funções que são executadas quando ocorrem ações como mover ou soltar. Essas ações são transformadas em um gesto que é dado como parâmetro para as funções.

Tais gestos possuem diversos campos. Para esse aplicativo será usado **dx** e **dy**, que são as distâncias entre onde o toque começou e onde se encontra no momento. O método "*onPanResponderMove*" é executado toda vez que **dx** ou **dy** mudam. Assim, se determinarmos **dx** e **dy** as novas posições do *flashcard* em relação a posição inicial, teremos como resultado um *card* acompanhando o movimento do dedo.

Outro método que será usado é o "*onPanResponderRelease*" executado sempre que o usuário distancia o dedo da tela após um movimento, ou seja, no fim de um gesto de deslizar. Para interpretar se o usuário teve a intenção de deslizar para algum dos lado, vamos nos basear nos valores de **dx** e **dy**:

- caso **dx** seja maior que um quarto da largura tela, significa que o usuário deslizou uma distância considerável. Assim podemos considerar um movimento de deslizar para direita, ou seja, acertou a resposta;
- caso **dx** seja negativo, com módulo maior que um quarto da largura tela, significa que o usuário deslizou uma distância considerável. Assim podemos considerar um movimento de deslizar para esquerda, ou seja, errou a resposta;
- como o aplicativo será usado no modo retrato, com altura maior do que largura, se usarmos a distância necessária para deslizar para cima como um quarto da altura, o movimento necessário será muito longo e inconveniente, assim, caso **dy** seja maior que um 17% da altura tela, significa que o usuário deslizou uma distância considerável. E podemos considerar um movimento de deslizar para cima, ou seja, acertou a resposta com facilidade;
- no caso das posições de **dx** e **dy** significarem mais de um movimento, será considerado apenas o movimento horizontal.

8.4 Fluidez e visualização

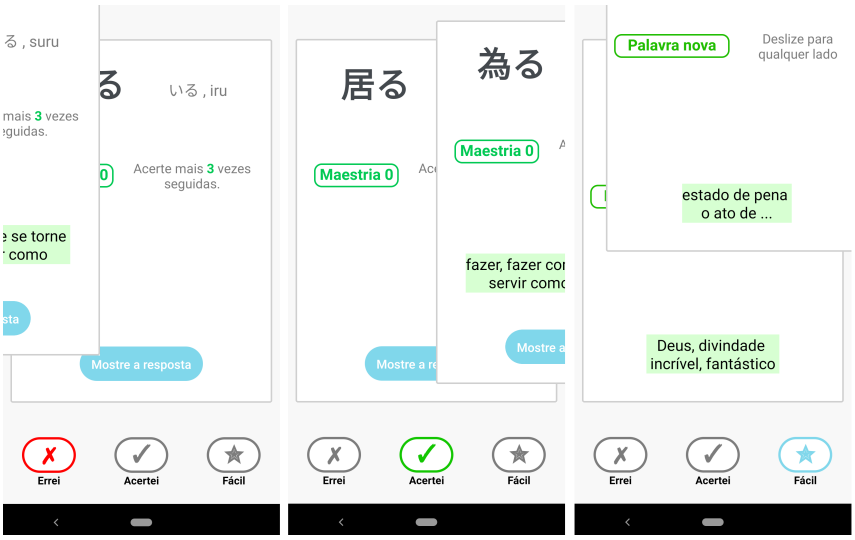

Figura 8.5: *Interfaces para delizamento*

As telas ilustradas na Figura 8.5 mostram que, caso a posição atual do gesto signifique completar um movimento para algum lado, mesmo que o usuário ainda não tenha distanciado o dedo da tela, o botão correspondente ao movimento terá sua cor modificada, indicando ao usuário que se soltar o *card* naquela posição ele efetuará aquela ação.

Para obter o efeito de *cards* infinitos, mostrado nas imagens, é sempre renderizado o *card* atual e o próximo em uma camada abaixo. Assim, ao deslizar o *flashcard* principal o próximo já estará visível, dando uma fluidez interessante ao aplicativo e removendo pequenos travamentos relacionados ao carregamento de um novo *card*.

Outra funcionalidade que contribui para a fluidez são animações, se após um movimento, o *card* apenas some da tela e o próximo aparece, temos uma usabilidade prejudicada, já que o usuário não consegue visualizar que o *flashcard* anterior já foi descartado e já está sendo mostrado um novo. Para solucionar esse problema, após o acionamento de um botão ou após um movimento de deslizar, será executada uma animação que faz o *card* se mover na direção correspondente à ação até este sumir da tela. A animação também é feita com a ajuda da biblioteca *Animate*. Como resultado, fica mais claro ao usuário a troca de *cards* que aconteceu e, no caso do uso de botões, indica que interações por deslize também podem ocorrer.

Capítulo 9

Exibição de cards

9.1 Algoritmo de repetição espaçada

A repetição espaçada é uma técnica usada em conjunto de *flashcards*. Ela tem como objetivo aumentar a taxa de memorização. É usada em contextos em que é necessário a memorização e retenção de diversos itens na memória, por tempo indeterminado. Assim, se encaixa perfeitamente no problema de aumento de vocabulário, que será visado neste aplicativo. Sua premissa é que com o passar do tempo esquecemos boa parte do que aprendemos, e quanto mais enraizada está uma memória, mais tempo demora para a esquecermos. Assim, conceitos novos e menos fixados devem ser repetidos com maior frequência do que conceitos em que foi demonstrada maior proficiência.

O sistema mais básico de repetição espaçada é o sistema de Leitner (Leitner, 1970). Ele é baseado em caixas e *cards*, ao responder um *card* corretamente ele vai para próxima caixa, ao responder incorretamente ele volta uma caixa. *cards* em caixas anteriores são mais repetidos e em caixas posteriores menos. O sistema é mostrado na Figura 9.1 .

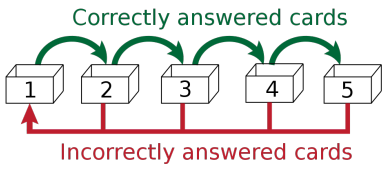

Figura 9.1: *Sistema de Leitner Leitner*

É um sistema simples que pode ser utilizado sem computadores ou celulares modernos. Com o avanço da tecnologia, novas possibilidades surgem, entre elas: usar *flashcards* digitais, definição individual de dificuldade de cada *flashcard* (não é mais necessário pertencer a um grupo de dificuldade como uma caixa), definição individual de quando cada *flashcard* deve ser mostrado novamente, mais de duas opção de resposta (certo, certo com facilidade, certo com dificuldade, errado, errado sem ideia nenhuma da

resposta etc.). Este aplicativo fará uso de algumas dessas novas possibilidades visando melhores resultados.

Os pilares do aplicativo são permitir uma boa memorização a longo prazo, mostrar *cards* relevantes à peça de entretenimento selecionada e ter uma usabilidade simples. O uso de *flashcards* digitais com algoritmos de repetição espaçada já foi implementado em softwares como Anki (Anki, 2020) e Supermemo (Supermemo, 2020). Porém, esses algoritmos de repetição definem por si só a ordem de exibição de *cards* e no caso deste aplicativo a frequência de uma palavra na produção também deve ter um grande impacto nessa ordem. Outro ponto é que, como os *flashcards* foram gerados programaticamente, tem-se uma quantidade enorme de palavras que devem ser introduzidas aos poucos ao usuário.

Além disso, temos um conjunto de *flashcards* (deck) diferente para cada produção, que devem interagir entre si, de modo que interações com uma palavra em um deck reflita em todos os outros. Isso também deve ser feito de uma forma escalável, para possibilitar aumentar cada vez mais a lista de animes, filmes e outros tipos de produções no futuro, como músicas. Portanto, foi necessário desenvolver um algoritmo de exibição de *flashcards* próprio ao aplicativo.

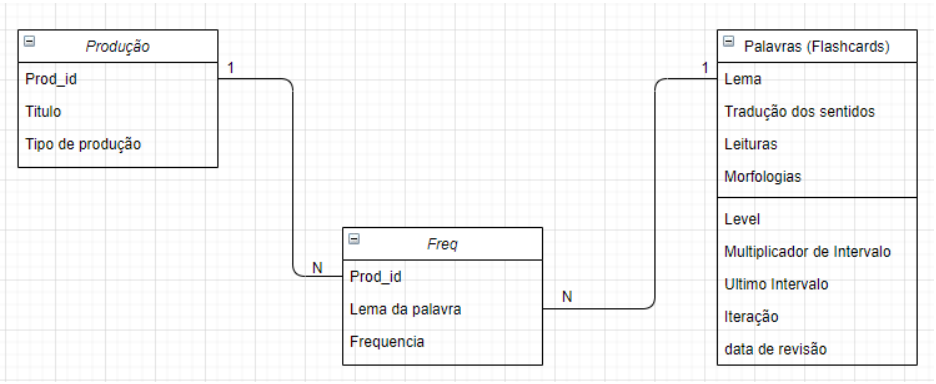

Figura 9.2: *Base de dados Realm utilizada no aplicativo.*

A Figura 9.1 apresenta o esquema da base de dados Realm usada, que como já mencionado possui ótima performance em dispositivos móveis comparado às alternativas. No esquema dos *flashcards*, há cinco variáveis que serão usadas para definir a ordem de exibição. A frequência de cada palavra na produção selecionada também será importante.

9.1.1 Aplicação da repetição espaçada

O algoritmo de repetição espaçada precisa definir qual o intervalo de tempo até que o *card* possa ser mostrado novamente. Para isso ele usa seguintes os conceitos:

- Data de revisão: Quando o *card* estará disponível para ser visto novamente. A data é representada como quantidade de milisegundos desde 1 de Janeiro de 1970 00:00:00. Inicialmente vale 0.
- Multiplicador de intervalo: O multiplicador calculado de um *card*, é usado para obtenção de um novo intervalo de tempo. A cada acerto o multiplicador aumenta e a cada erro ela diminui.
- Último Intervalo: Um intervalo determina o tempo entre a última exibição do *card* e quando ele ficará disponível novamente. Esta variável guarda qual foi o último intervalo de tempo calculado para o *card*. Inicialmente tem valor 0.
- Iteração: Quantas vezes foram calculados intervalos para esse *card*, ou seja, por quantos ciclos de exibição e espera ele já passou. Inicialmente tem valor 0.

Na primeira e segunda iteração, o intervalo até a próxima revisão é fixo, e será de 1 dia e 7 dias respectivamente. Isso ocorre, pois, nas primeiras iterações, ainda não se tem uma boa ideia do multiplicador ideal para o usuário com aquele *flashcard*. Nas próximas iterações é calculado o novo intervalo de acordo com o produto do anterior e o multiplicador, esse intervalo é somado com a data atual obtendo a Data de revisão. Assim *cards* mais difíceis (valor do multiplicador menor) serão mais repetidos enquanto mais fáceis (valor do multiplicador maior)

Intervalo(Iteração = 0) = 1 Dia Intervalo(Iteração = 1) = 7 Dias Intervalo(Iteração > 1) = Intervalo(Iteração - 1) * multiplicador

Os valores de intervalo fixos, o impacto de acertos e erros no multiplicador de intervalo e outros valores fixos citados neste capítulo podem ser ajustados no futuro caso testes com usuários mostrem melhores resultados algumas mudanças.

9.2 Introdução de cards

Para introduzir *cards* aos poucos, são usados subconjuntos de *cards* com tamanho N = 10, cada *flashcard* será repetido até atingir o level = 4. O level de um *card* é inicialmente 1, a cada acerto, o level sobe e, a cada erro, diminui para um valor mínimo 1. Assim, o usuário deverá acertar o significado de um *flashcard* algumas vezes antes dele parar de ser repetido. Quando um *card* atinge nível 4 é removido do subconjunto, seu level é resetado e entra em ação o algoritmo de repetição espaçada.

O subconjunto de tamanho N = 10 é obtido a partir da classe *cards*, para isso, deve-se iniciar a classe com o id da produção em contexto. Após iniciada, o componente pode executar a função "*cards*.getCards", com número de *cards* desejados "N" como parâmetro.

A função encontra a lista com pares de palavras e frequências, relacionadas à produção, na base de dados. Em seguida, obtém os *cards* relacionados a essas palavras. Os *cards* são então ordenados, de acordo com a frequência e os N primeiros *cards* com Data de revisão anterior a data atual são retornados.

Foi adotado o valor N=10, pois de acordo com feedbacks de usuários de teste, foi um bom tamanho. Se N é muito baixo, temos subconjuntos muito pequenos e a memorização de curto prazo fica muito facilitada. Se N é muito grande, cada subconjunto tem muitos *cards* dificultando muito a memorização de curto prazo frustrando o usuário.

Exemplo: Com N=2 apenas 2 palavras ficarão sendo repetidas por ciclo. O usuário só precisará memorizar 2 *cards* por vez , embora isso faça ele acertar muito mais vezes, os acertos não vão significar melhorias na memorização de longo prazo.

9.3 Remoção e adição de novos *cards*

Quando o *card* atinge o level 4 significa que já houve uma quantidade razoável de acertos comparado a de erros, ou seja, o *card* já está suficientemente memorizado. Portanto, será removido, e o algoritmo de repetição espaçada decidirá quando mostrá-lo novamente. O level do *card* é resetado, para que, quando voltar a disponibilidade, ainda seja necessário alguns acertos, antes de que volte a espera.

Com um *flashcard* removido, é necessário obter outro, assim continuaremos com N=10 *cards*. Para isso, a função *cards*.getCards é executada, passando 1 como parâmetro, já que queremos apenas mais 1 *flashcard*, que será então adicionado ao subconjunto.

Capítulo 10

Testes com usuários e outras funcionalidades

O aplicativo foi desenvolvido visando a entrega de software funcional e, a partir daí, realizar iterações de teste com usuários com evoluções sucessivas. Esses ciclos permitem entender rapidamente as funcionalidades que realmente importam para o usuário, e evita gastar tempo com ideias e funcionalidades que poderiam ser irrelevantes.

A partir desses ciclos algumas funcionalidades foram implementadas, tais como:

- Botão de "Fácil" (acerto com facilidade) Alguns usuários já tinham um conhecimento prévio de japonês, assim, já tinham o significado de algumas palavras fixadas. Porém, não tinham uma forma de demonstrar isso, assim, essas palavras continuavam aparecendo novamente. Com o novo botão, e gesto de deslizar para cima, o *flashcard* atinge diretamente nível = 4, ou seja, não será mais mostrado naquele ciclo, e tem seu multiplicador de intervalo aumentado significativamente, para demorar a ser mostrado novamente.
- Informações sobre quantidade de *flashcards* e maestria. Em versões iniciais, o aplicativo transiciona diretamente da tela de listagem de produções para a visualização de *cards*. Como feedback, um usuário comentou que havia falta de senso de progresso. Ele gostaria de saber quantos *cards* já foram estudados de um anime específico, e em que nível ele se encontra para determinadas palavras. Assim, foi introduzida uma tela para visualização de informações como quantidade de *cards* disponíveis e quantos já foram vistos. Também foi introduzido o conceito de maestria que indica por quantos ciclos de espera um determinado *card* passou.

10.1 Futuras modificações necessárias

Em futuras versões do software, o conceito de maestria precisa ser modificado para considerar também a dificuldade (dada pelo multiplicador de intervalo) de um *card*. Assim, *flashcards* com que o usuário tem mais dificuldade, embora repetidos mais vezes, terão maestria menor. A tela de visualização também deverá mostrar a quantidade de palavras por maestria. Exemplo: Você possui [maestria 5] em 13 *flashcards*

Capítulo 11

Conclusão e futuro

11.1 Conclusão

Neste projeto, conseguimos atingir o objetivo de desenvolver uma plataforma com usabilidade simples e fluida para a expansão de vocabulário do idioma japonês em um contexto de entretenimento. O desenvolvimento, buscando fluidez e usabilidade, foi guiado com a ajuda de testes com usuários. Já a seleção de produções de entretenimento foi baseada em sites que classificam sua popularidade. Também foram implementadas técnicas para o auxílio da memorização de longo prazo. Porém, é difícil mensurar sua efetividade no aplicativo, seria necessário testes com usuários por longos períodos de tempo. Pode-se apenas esperar a eficácia baseada em aplicações prévias das técnicas em outros contextos.

Não foi desenvolvido apenas o aplicativo, mas todo um método para transformação de simples arquivos de texto, em um sistema de aprendizado, que utiliza *flashcards* interativos com palavras, seus significados e leituras. Esse método, possibilita que o escopo de produções do aplicativo aumente, podendo incluir futuramente músicas, mangás, entre outros. Também há a possibilidade de uso do método em outros idiomas com algumas modificações no código e nos dicionários utilizados.

Conforme o desenvolvimento avançou, e testes com usuários foram sendo feitos, foi notado um grande leque de possíveis adições, ao aplicativo e ao método, que podem melhorar ainda mais a usabilidade e aprendizado. A importância de entregas frequentes do software, com feedback, foi provada. Surgem não só sugestões dos usuários como também insights interessantes sobre possíveis funcionalidades.

11.2 Futuro

Algumas funcionalidades possíveis que podem beneficiar o aprendizado, porém devido a limites de tempo não foram implementadas são:

- Inclusão de trechos, com sua tradução, em que a palavra aparece em uma produção específica. Essa funcionalidade, além de possibilitar o melhor entendimento do contexto em que a palavra é utilizada, também deixa mais fácil seu reconhecimento em um filme ou anime.
- Para *flashcards* com verbos, mostrar conjugações diferentes que foram utilizadas na produção, com suas respectivas traduções. Isso além de facilitar o entendimento do significado de um verbo, ensina indiretamente a como conjugar.
- Criação de decks de *flashcards* com Hiraganas/Katakanas e suas respectivas leituras.

Essas e possíveis adições permitem ao aplicativo não somente viabilizar uma plataforma para construção de vocabulário, mas principalmente oferecer um aprendizado mais completo do idioma japonês aliado ao entretenimento.

Referências

[Anki 2020] Anki. *Anki*. 2020. URL: https://apps.ankiweb.net/ (citado na pg. 28).

[Google 2020] Google. *Cloud Translation*. 2020. URL: https://cloud.google.com/translate (citado na pg. 11).

[JMDict 2020] JMDict. *jmdict*. 2020. URL: https://www.edrdg.org/jmdict/j_jmdict.html (citado na pg. 11).

[Kitsunekko 2020] Kitsunekko. *Kitsunekko*. 2020. URL: https://kitsunekko.net/ (citado na pg. 5).

[Kudo 2004] Taku Kudo. *Applying Conditional Random Fields to Japanese Morphological Analysis*. 2004. URL: http://chasen.org/~taku/publications/emnlp2004-2.pdf (citado nas pgs. 7, 8).

[Lafferty 2001] John Lafferty. *Conditional random fields: Probabilistic models for segmenting and labeling sequence data*. 2001. URL: https://repository.upenn.edu/cgi/viewcontent.cgi?article=1162&context=cis_papers (citado na pg. 8).

[Leitner 1970] Leitner. *Leitner system*. 1970. URL: https://en.wikipedia.org/wiki/Leitner_system (citado na pg. 27).

[MeCab 2013] MeCab. *MeCab: Yet Another Part-of-Speech and Morphological Analyzer*. 2013. URL: https://taku910.github.io/mecab/ (citado na pg. 8).

[Myanimelist 2020] Myanimelist. *Myanimelist*. 2020. URL: https://myanimelist.net/ (citado na pg. 5).

[Native 2020] React Native. *React Native*. 2020. URL: https://reactnative.dev/docs/getting-started (citado na pg. 17).

[Realm 2020] Realm. *What is Realm Platform?* 2020. URL: https://docs.realm.io/sync/what-is-realm-platform (citado na pg. 17).

[Romkan 2013] Romkan. *Romkan*. 2013. URL: https://pypi.org/project/romkan/ (citado na pg. 12).

[Supermemo 2020] Supermemo. *Supermemo*. 2020. url: https://www.supermemo.com (citado na pg. 28).

[wikipedia 2020] wikipedia. *Tabela de Hiraganas*. 2020. url: https://pt.wikipedia.org/wiki/Hiragana (citado na pg. 13).