ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO BÀI TẬP LỚP
MÔN HỌC: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Tên bài tập	**: Phát triển ứng dụng học Tiếng Anh bằng Java**
Mã lớp	**: 2324I_INT2204_20**
Giảng viên	**: Tô Vân Khánh** **Nguyễn Thị Thu Trang**
Nhóm	**: Nhóm 2**
Thành viên	**- Hoàng Bảo Long** **- Nguyễn Đức Mạnh** **- Lê Tuấn Kiệt**

Mục lục

I. Tổng quan

Từ điển là một ứng dụng được thiết kế một cách tinh tế để phục vụ nhu cầu tra cứu từ vựng của người dùng. Nó không chỉ cung cấp các định nghĩa chính xác và rõ ràng, mà còn bao gồm các ví dụ minh họa và thông tin bổ sung, giúp người dùng hiểu sâu hơn về cách sử dụng từ trong ngữ cảnh cụ thể. Giao diện người dùng của Từ điển được thiết kế để đảm bảo tính trực quan và dễ dàng truy cập, làm cho việc tìm kiếm và học hỏi từ mới trở nên thuận lợi và hiệu quả. Đây là công cụ không thể thiếu cho những ai mong muốn cải thiện và mở rộng vốn từ vựng của mình, đồng thời nâng cao khả năng giao tiếp và hiểu biết về ngôn ngữ.

II. Cấu tạo và các tổng quan thành phần của chương trình

1. Phần Chức năng

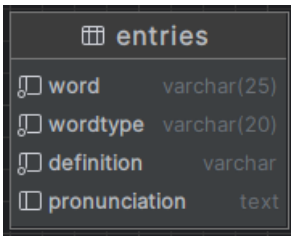
Phần chức năng của từ điển chính bao gồm tất cả các hàm chức năng được sử dụng trong phần mã nguồn

- Hàm tìm kiếm và tìm kiếm tương đồng, sử dụng cấu trúc dữ liệu Trie, tình toán độ tương đồng và xử lí xâu
- Các hàm xử lí thêm, sửa, xóa từ
- Các thuật toán tìm kiếm, cấu trúc dữ liệu

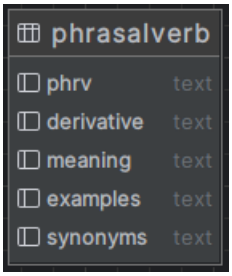
2. Phần Cơ sở dữ liệu

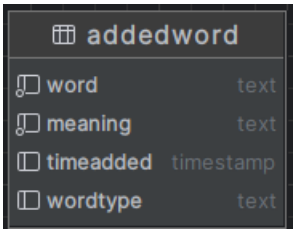
Phần cơ sở dữ liệu được sử dụng là SQLite với ba bảng dữ liệu riêng biệt:

- Bảng "entries" là cơ sở dữ liệu gốc, chứa tất cả các từ vựng của từ điển
- Bảng "phrasalverb" tương tự nhưng là các cụm động từ



- Bảng "addedword" là bảng chứa các từ người dùng thêm vào, phục vụ cho mục đích thêm, sửa, xóa của người dùng



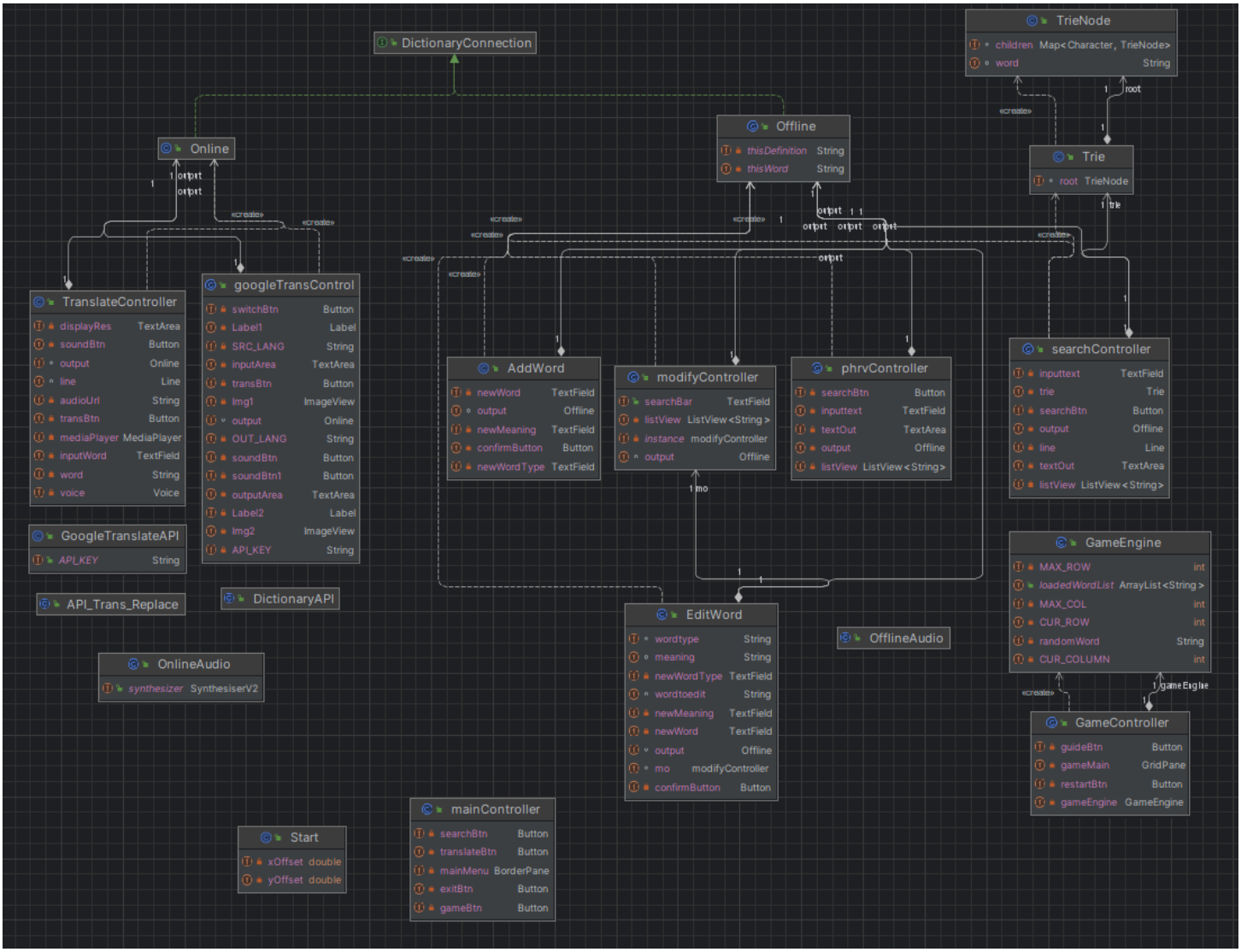


Cùng với đó là các câu câu truy vấn được viết để phục vụ cho phần chức năng

3. Phần Đồ họa chương trình

Phần đồ họa chương trình sử dụng JavaFX, CSS cùng với các hình ảnh, biểu tượng, màu sắc được tùy chỉnh theo cá nhân.

4. Sơ đồ quan hệ và kế thừa



III. Các package chính của chương trình và chi tiết

1. Package API

Đây là package chứa các API mà chương trình đã sử dụng, bao gồm:

- Free Dictionary API
- Google Translate API

a. Free Dictionary API

Dữ liệu trả về là dạng JSON, nên cần qua một phần xử lí để đưa tất cả về dạng xâu, và hiển thị kết quả, đây là phần xử lí trong DictionaryAPI.java

```
public static String searchWord(String newWord) {
    StringBuilder output = new StringBuilder();
    try {
        URL dictionaryAPI_URL =
                new
URL("https://api.dictionaryapi.dev/api/v2/entries/en/" +
newWord);

        HttpURLConnection newConnection =
                (HttpURLConnection)
dictionaryAPI_URL.openConnection();

        newConnection.setRequestMethod("GET");

        BufferedReader input =
                new BufferedReader(new
InputStreamReader(newConnection.getInputStream()));
        String inputLine;
        StringBuffer wordDetail = new StringBuffer();
        while ((inputLine = input.readLine()) != null) {
            wordDetail.append(inputLine);
        }

        input.close();
        newConnection.disconnect();

        JSONArray wordDetailArray = new
JSONArray(wordDetail.toString());

        try {
            if (wordDetailArray.length() <= 0) {
                throw new NegativeArraySizeException();
            }
        } catch (NegativeArraySizeException e) {
```

```
            System.out.println(e.getMessage());
            e.printStackTrace();
        }

        JSONObject wordObject =
wordDetailArray.getJSONObject(0);

        String wordWord = wordObject.optString("word", null);
        String wordPhonetic =
wordObject.optString("phonetic", null);

        output.append("Word:
").append(wordWord).append("\n");
        output.append(wordPhonetic).append("\n");

        JSONArray meaning =
wordObject.getJSONArray("meanings");
        for (int i = 0; i < meaning.length(); i++) {
            JSONObject meaningObject =
meaning.getJSONObject(i);

            String wordPos =
meaningObject.optString("partOfSpeech", null);
            output.append("Part of Speech:
").append(wordPos).append("\n");

            JSONArray definitions =
meaningObject.getJSONArray("definitions");
            for (int j = 1; j <= definitions.length(); j++) {
                JSONObject definitionsObject =
definitions.getJSONObject(j - 1);
                String wordDefinition =
definitionsObject.optString("definition", null);
                String wordExample =
definitionsObject.optString("example", null);

output.append("Definition(").append(j).append("):
").append(wordDefinition).append("\n");

                if (wordExample != null) {
                    output.append("Example:
").append(wordExample).append("\n");
                }
            }
        }

    } catch (Exception e) {
        output.append("Word not found");
    }
    return output.toString();
}
```

Dạng file JSON lưu dữ liệu dưới dạng Objects (các đối tượng), với API như trên, dữ liệu trả về cho một từ cụ thể, giả sử từ “example” sẽ có dạng như sau:


```
[
  {
    "word": "example",
    "phonetic": "/əgˈzæmpl̩/",
    "meanings": [
      {
        "partOfSpeech": "noun",
        "definitions": [
          {
            "definition": "Something that is representative of all
such things in a group.",
            "synonyms": [],
            "antonyms": []
          },
          {
            "definition": "Something that serves to illustrate or
explain a rule.",
            "synonyms": [],
            "antonyms": []
          },
          {
            "definition": "Something that serves as a pattern of
behaviour to be imitated (a good example) or not to be imitated (a
bad example).",
            "synonyms": [],
            "antonyms": []
          },
          {
            "definition": "A person punished as a warning to
others.",
            "synonyms": [],
            "antonyms": []
          },
          {
            "definition": "A parallel or closely similar case,
especially when serving as a precedent or model.",
            "synonyms": [],
            "antonyms": []
          },
          {
            "definition": "An instance (as a problem to be solved)
serving to illustrate the rule or precept or to act as an exercise
in the application of the rule.",
            "synonyms": [],
            "antonyms": []
          }
        ],
```

```
        "synonyms": [],
        "antonyms": []
      },
      {
        "partOfSpeech": "verb",
        "definitions": [
          {
            "definition": "To be illustrated or exemplified
(by).",
            "synonyms": [],
            "antonyms": []
          }
        ]
      }
    ]
  }
]
```

Dạng JSON trả về gồm 1 mảng, trong đó chứa 1 Object

- Trong Object chứa 2 xâu đầu tiên với khóa là `"word"` và `"phonetic"` tiếp đó là 1 Object với khóa là `"meanings"`
- Trong Object `"meanings"` chứa 2 Object nữa, cả 2 Object đều chứa đầu tiên là 1 xâu với khóa `"partOfSpeech"`, sau đó là 1 mảng với khóa `"definitions"`
- Trong mảng `"definitions"` lại tiếp tục chứa các phần từ là các Object, và trong các Object chứa 1 xâu và 2 mảng với khóa lần lượt là `"definition"`, `"synonyms"`, và `"antonyms"`
- `"synonyms"`, và `"antonyms"` là 2 mảng rỗng với không có phần từ nào
- Cứ sau mỗi lần xử lí tại các Object, StringBuilder được khởi tạo sẽ nối lại các chuỗi vừa xử lí, kết quả trả về là một xâu hoàn chỉnh hiển thị trên trang ứng dụng

Phần xử lí JSON trên yêu cầu sự hỗ trợ của các thư viện JSON, và phần request API thì cần sự hỗ trợ từ các thư viện .net với giao thức kết nối HTTPS

```
import org.json.JSONArray;
import org.json.JSONObject;

import java.net.HttpURLConnection;
import java.net.URL;
```

b. Google Translate API

Sử dụng chính API từ Google Cloud Translate, ta sẽ cần sử dụng một API Key từ Google và từ đó sẽ có quyền sử dụng thư viện này:

- Trong file GoogleTranslateAPI.java:

```
public static String API_KEY = "AIzaSyBOti4mM-
6x9WDnZIjIeyEU21OpBXqWBgw";

public static String getAPI_KEY() {
    return API_KEY;
}
```

Tuy nhiên sử dụng API Key để dịch sẽ có ưu điểm là sẽ dịch được văn bản với một số lượng lớn từ, tuy nhiên khi API Key chẳng may hỏng thì từ điển sẽ không thể nào dịch được văn bản nữa.

- Để khắc phục lỗi này ta có thể sử dụng API từ một nguồn không chính thức của Google, cụ thể:

```
String st = "en";
String tl = "vi";
String text = "hello";

String GOOGLE_TRANSLATE_URL =
"https://translate.googleapis.com/translate_a/single?client=gtx&
sl=" + st + "&tl=" + tl + "&dt=t&q=" + text;
```

Với đường link này ta có format như sau:
- st: là ngôn ngữ của nguồn văn bản cần dịch với “en" là tiếng anh.
- tl: là ngôn ngữ mà ta cần dịch văn bản kia thành với “vi" là tiếng việt.
- text: là đoạn văn bản mà ta cần dịch.

Kết hợp tất cả các yếu tố lại ta sẽ có một đường link sau:

https://translate.googleapis.com/translate_a/single?client=gtx&sl=en&tl=vi&dt=t&q=hello

Khi ta truy cập đường link nó sẽ trả về một file json chứa kết quả văn bản sau khi dịch. Theo cách này ta đã bỏ được nhược điểm của Google API là bị phụ thuộc vào API Key còn hoạt động được hay không và có thể sử dụng mọi lúc miễn là có kết nối mạng, tuy nhiên với cách thức này sẽ bị giới hạn độ dài văn bản mà ta có thể dịch được.

Để triển khai 2 phương pháp trên vào ứng dụng:

- Đối với Google API thì ta dùng chính thư viên của Google Cloud Translate.
- Đối với nguồn không chính thức, ta dùng một thư viện ngoài của maven tại nguồn https://mvnrepository.com/artifact/com.github.goxr3plus/java-google-speech-api/8.0.0 , thư viện này đã hỗ trợ xử lý json mỗi khi đường link kia trả về.

Trong file googleTransControl:

```
// import API Key
import API.GoogleTranslateAPI;
// import thu vien xu ly nguon API khong chinh thuc
import com.darkprograms.speech.translator.GoogleTranslate;
// import thu vien translate cua Google Cloud Translate
import com.google.cloud.translate.Translate;
import com.google.cloud.translate.TranslateOptions;
import com.google.cloud.translate.Translation;

GoogleTranslateAPI api = new GoogleTranslateAPI();
private final String API_KEY = output.connectionType();
// Key API connection
private String SRC_LANG = "en";
private String OUT_LANG = "vi";

try {
    // Dich bang Google API
    Translate translateTest =
TranslateOptions.newBuilder().setApiKey("API_KEY").build().getServ
ice();
    Translation translation = translateTest.translate(input,
Translate.TranslateOption.sourceLanguage(SRC_LANG),
Translate.TranslateOption.targetLanguage(OUT_LANG));
    String translatedText = translation.getTranslatedText();
    outputArea.setText(translatedText);
} catch (Exception e1) {
    try {
        // Dich bang unoffical google API
        String translatedText =
GoogleTranslate.translate(SRC_LANG, OUT_LANG, input);
        outputArea.setText(translatedText);
    } catch (Exception e2) {
        System.out.println("Khong tim thay tu");
    }
}
```

Đầu tiên sau khi import thư viện và set up các biến cần thiết, ta sẽ thử sử dụng Google Cloud API để dịch trước. Nếu API kia bị hỏng, thì ta sẽ chuyển sang

nguồn API không chính thức. Việc ta sử dụng 2 cách dịch API này sẽ đảm bảo cho API dịch của ta sẽ luôn luôn hoạt động trong hầu hết mọi trường hợp với điều kiện máy tính có kết nối với mạng.

2. Package Audio

Đây là phần chứa các phương thức phát âm của chương trình:

- Online Audio
- Offline Audio

a. Online Audio

Đối với Online Audio, để có audio thật chính xác và đa ngôn ngữ ta phải sử dụng chính API từ Google Cloud Text to Speech điều này có nghĩa là ta bắt buộc phải có API Key để cho có thể hoạt động được, chúng ta có 2 cách để sử dụng:

- Ta có thể sử dụng trực tiếp qua thư viện Google Cloud Text to Speech.
- Hoặc ta có thể sử dụng một link từ chính Google Cloud Text to Speech.

Trong ứng dụng này thì ta sử dụng theo phương pháp sử dụng một link từ chính Google Cloud Text to Speech, cụ thể:

```
String st = "en";
String text = "hello";
String API_KEY = "AIzaSyBOti4mM-6x9WDnZIjIeyEU21OpBXqWBgw";

String audio = “https://www.google.com/speech-
api/v2/synthesize?enc=mpeg&client=chromium&key=” + API_KEY +
“&text=” + text + "&lang=" + st;
```

Với đường link này ta có format như sau:

- st: là ngôn ngữ của nguồn văn bản cần lấy audio với “en" là tiếng anh.
- text: là đoạn văn bản mà ta cần lấy audio.
- API_KEY: là API Key để cho phép ta sử dụng Google Cloud Text to Speech.

Kết hợp tất cả các yếu tố lại ta sẽ có một đường link sau:
https://www.google.com/speech-api/v2/synthesize?enc=mpeg&client=chromium&key=AIzaSyBOti4mM-6x9WDnZIjIeyEU21OpBXqWBgw&text=hello&lang=en

Khi ta truy cập vào đường link này thì sẽ trả về cho ta một file audio dưới dạng mp3.

Để cài đặt trong ứng dụng, trong file OnlineAudio.java:

```
import com.darkprograms.speech.synthesiser.SynthesiserV2;
import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.advanced.AdvancedPlayer;


// Khai bao API_Key de lay file audio
public final static SynthesiserV2 synthesizer = new
SynthesiserV2("AIzaSyBOti4mM-6x9WDnZIjIeyEU21OpBXqWBgw");

public static void usingOnlineSpeak(String text) {
    Thread soundThread = new Thread(() -> {
        try {
            AdvancedPlayer audioPlayer = new
AdvancedPlayer(synthesizer.getMP3Data(text));
            audioPlayer.play();
        } catch (IOException | JavaLayerException e) {
            System.out.println("Word not exist");
            e.printStackTrace();
        }
    });
    soundThread.setDaemon(false);
    soundThread.start();
}
```

Để xử lý file audio này, ta sử dụng thư viện https://mvnrepository.com/artifact/com.github.goxr3plus/java-google-speech-api/8.0.0 và Jlayer.

Đầu tiên ta import các thư viện ngoài, tiếp sau đó ta dùng SynthesiserV2 từ https://mvnrepository.com/artifact/com.github.goxr3plus/java-google-speech-api/8.0.0 để nhập một API Key của Google sau đó sử dụng nó để xử lý tải về file mp3 từ đường link kia thành dạng InputStream để từ đó dùng thư viện Jlayer chạy được trực tiếp audio đó trong một thread mới. Trong code gốc của thư viện SynthesiserV2 đã set sẵn ngôn ngữ là "auto" nên ta không cần chỉ định ngôn ngữ audio cần lấy mà chính Google Cloud sẽ tự động lấy dựa theo ngôn ngữ input của chúng ta.

Lý do ta chọn cách xử lý này thay vì Google Cloud vì thư viện bên thứ ba hỗ trợ tải và xử lý thành dạng InputStream để Jlayer có thể chạy audio trực tiếp luôn. Và sau khi đóng chương trình sẽ không lưu lại gì vào đoạn code gốc. Còn nếu ta xử dụng theo cách Google Cloud thì ta phải tạo một file mp3 trung gian để mỗi lần sử dụng nó sẽ tải về và lưu file mp3 vào đó để chạy audio, như vậy sẽ tồn tại một file audio trung gian để lưu trữ mp3 kể cả sau khi đóng chương trình.

Triển khai trong ứng dụng, trong file googleTransControl.java

```
import static Audio.OnlineAudio.usingOnlineSpeak;

@FXML
public void playAudioIn() {
    String tmp = inputArea.getText();
    usingOnlineSpeak(tmp);
}

@FXML
public void playAudioOut() {
    String tmp = outputArea.getText();
    usingOnlineSpeak(tmp);
}
```

Sau khi import phương thức static từ file OnlineAudio.java, ta có thể sử dụng trực tiếp hàm usingOnlineSpeak để nhận một tham số là text và có thể chạy được audio sound luôn.

b. Offline Audio

Sử dụng thư viện FreeTTS, cụ thể:

```
import com.sun.speech.freetts.Voice;
import com.sun.speech.freetts.VoiceManager;

public class OfflineAudio {
    public static void usingOfflineSpeak(String text) {
        System.setProperty("freetts.voices",
"com.sun.speech.freetts.en.us.cmu_us_kal.KevinVoiceDirectory"
);
        // Khoi tao sound
        // set nguoi noi la kevin16
        VoiceManager voiceManager =
VoiceManager.getInstance();
        Voice voice = voiceManager.getVoice("kevin16");
        if (voice == null) {
            System.out.println("Cannot find the specified
voice.");
```

```
            }
            // Synthesize speech
            voice.allocate();
            voice.speak(text);
        }
    }
```

Sau khi import thư viện, ta sẽ tạo một hàm để phát audio như trên với giọng đọc là kevin16 của thư viện FreeTTS. Khi sử dụng thì chỉ cần gọi hàm usingOfflineSpeak(text); với tham số đầu vào là đoạn văn bản cần đọc là sẽ phát được audio. Do FreeTTS chỉ hỗ trợ đọc tiếng anh nên nó được sử dụng cho trang Search từ Offline và trang Phrasal Verb cũng chạy Offline.

3. Package DictionaryMethod

Đây là phần triển khai các phương thức của từ điển với 2 loại kết nối

- Online
- Offline

a. Online

Tại đây là các phương thức có sử dụng đến API tại package trên

```
@Override
public String connectionType() {
    return GoogleTranslateAPI.getAPI_KEY();
}

@Override
public String searchWord(String newWord) {
    return DictionaryAPI.searchWord(newWord);
}
```

b. Offline

Tại đây là phần sử dụng đến cơ sở dữ liệu, trước hết sẽ là phần két nối

```
public Connection connection() {
    String dbPath = "./Dictionary.db";
    Connection connection = null;
    try {
        connection =
DriverManager.getConnection("jdbc:sqlite:" + dbPath);

    } catch (SQLException e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
    return connection;
}
```

Phần còn lại sẽ là các phương thức gọi truy vấn đến cơ sở dữ liệu (phần này có code khá dài nên em xin phép chỉ đưa ra những phương thức chính)

```
- public String searchWord(String word) {
        thisWord = word;
        StringBuilder output = new
   StringBuilder(word).append("\n");
        String searchQuery = "SELECT definition, wordtype FROM
   entries WHERE word = ?";
        String phoneticsQuery = "SELECT DISTINCT pronunciation
   FROM entries WHERE word = ?";
```

```
    Map<String, String> definitions = new HashMap<>();
    String phonetic;

    try (Connection newConnection = this.connection();
PreparedStatement query =
newConnection.prepareStatement(phoneticsQuery)) {
        query.setString(1, word);
        ResultSet outputString = query.executeQuery();
        phonetic =
outputString.getString("pronunciation");
        output.append(phonetic).append("\n");

        try (PreparedStatement subquery =
newConnection.prepareStatement(searchQuery)) {
            subquery.setString(1, word);

            try (ResultSet outputMap =
subquery.executeQuery()) {
                while (outputMap.next()) {
                    String wordtype =
outputMap.getString("wordtype");
                    String definition =
outputMap.getString("definition");
                    definitions.put(definition, wordtype);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }
    } catch (Exception e) {
        e.printStackTrace();
        System.out.println(e.getMessage());
    }

    if (definitions.isEmpty()) {
        throw new NullPointerException("Word not
available");
    }

    for (Map.Entry<String, String> entry :
definitions.entrySet()) {
        String def = entry.getKey().replaceAll("\\n\\s+",
" ").replaceAll("--", "");
        String wordtype = entry.getValue();
        output.append("(" + wordtype + ")" +
"\n").append(def + "\n");
    }

    thisDefinition = output.toString();
```

```
        return output.toString();
    }

- public void insertWord(String newWord, String newWordType,
    String newMeaing) {
        String insertQuery = "INSERT INTO addedword (word,
    wordtype, meaning) VALUES (?, ?, ?)";

        try (Connection newConnection = this.connection();
             PreparedStatement query =
    newConnection.prepareStatement(insertQuery)) {
            query.setString(1, newWord);
            query.setString(2, newWordType);
            query.setString(3, newMeaing);
            query.executeUpdate();
            System.out.println("Add new word successfully!");
        } catch (SQLException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }

- public void deleteWord(String deleteWord) {
        String deleteQuery = "DELETE FROM addedword WHERE word
    = ?";

        try (Connection newConnection = this.connection();
             PreparedStatement query =
    newConnection.prepareStatement(deleteQuery)) {
            query.setString(1, deleteWord);

            int affected = query.executeUpdate();

            if (affected > 0) {
                System.out.println("Delete word
    successfully");
            } else {
                System.out.println("Word does not exist");
            }

        } catch (SQLException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }

- public void editWord(String word, String newWord, String
    newWordType, String newMeaning) {
        String editQuery = "UPDATE addedword SET word = ?,
```

```
    wordtype = ?, meaning = ? WHERE " +
                "word = ?";

        try (Connection newConnection = this.connection();
             PreparedStatement query =
    newConnection.prepareStatement(editQuery)) {
            query.setString(1, newWord);
            query.setString(2, newWordType);
            query.setString(3, newMeaning);
            query.setString(4, word);

            int affected = query.executeUpdate();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }


- public List<String> getAllWordsOffline() {
        String sql = "SELECT DISTINCT word FROM entries ORDER
    BY word";

        List<String> words = new ArrayList<>();

        try (Connection conn = this.connection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                String word = rs.getString("word");
                words.add(word.toLowerCase());
            }
        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }
        return words;


- public List<String> getAllPhrasesOffline() {
        String sql = "SELECT phrv FROM phrasalverb ORDER BY
    phrv";

        List<String> words = new ArrayList<>();

        try (Connection conn = this.connection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                String word = rs.getString("phrv");
```

```
                words.add(word.toLowerCase());
            }
        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }
        return words;
    }
- public List<String> showAddedWord() {
        String sql = "SELECT word FROM addedword";

        List<String> output = new ArrayList<>();

        try (Connection conn = this.connection();
             PreparedStatement pstmt =
    conn.prepareStatement(sql)) {
            try (ResultSet words = pstmt.executeQuery()) {
                while (words.next()) {
                    String word = words.getString("word");
                    output.add(word);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }
        return output;
    }
```

Các phương thức cần phải yêu cầu việc viết truy vấn đề tương tác với cơ sở dữ liệu đều có các câu truy vấn được viết dưới dạng PrepareStatement để đảm bảo một số điều sau:

- ***Ngăn chặn SQL Injection***: PreparedStatement giúp tránh được các tấn công SQL Injection bằng cách liên kết các tham số với truy vấn trước khi thực thi. Điều này ngăn chặn việc nhập dữ liệu động trực tiếp vào câu lệnh SQL.
- ***Tối ưu hóa truy vấn***: Điều này có thể dẫn đến hiệu suất tốt hơn cho truy vấn được thực thi nhiều lần.
- ***Thay đổi truy vấn dễ dàng***: Có thể thay đổi các tham số hoặc nội dung của truy vấn mà không cần phải thay đổi cấu trúc cơ bản của truy vấn SQL.
- ***Tái sử dụng được***: PreparedStatement cho phép tái sử dụng các truy vấn với các tham số khác nhau, giúp tái sử dụng mã code một cách linh hoạt

- ***Loại bỏ escape kí tự đặc biệt***: PreparedStatement tự động xử lý việc escape ký tự đặc biệt trong dữ liệu, giúp tránh lỗi khi nhập các giá trị có chứa các ký tự đặc biệt (như dấu nháy đơn).
- ***Dễ đọc và bảo trì***: Cú pháp của PreparedStatement thường dễ đọc và hiểu hơn so với việc tạo truy vấn SQL bằng cách nối chuỗi.

4. Package DicFX

Đây là package chứa toàn bộ phần sử dụng các phương thức và đưa nội dung hiển thị lên ứng dụng. Package này bao gồm nhiều các package con là các trang riêng biệt của chương trình

- Game : phần game trong ứng dụng
- GoogleTranslate : phần dịch sử dụng GoogleAPI
- Modify : phần để thêm, sửa, xóa từ cho người dùng
- PhrasalVerb : phần tra các cụm động từ
- Search : phần tra tử điển thông thường
- Translate : phần tra từ qua sử dụng Free Dictionary API

a. Game

Ý tưởng: Game Wordle, một trò chơi giáo dục thú vị và hấp dẫn, được thiết kế để hỗ trợ việc học tiếng Anh. Trò chơi này giúp người chơi nâng cao vốn từ vựng, khả năng phân tích và suy luận, cũng như kỹ năng ngôn ngữ chung.

Cách chơi:

Trong Wordle, người chơi sẽ được đưa ra một từ tiếng Anh bí mật. Nhiệm vụ của họ là đoán từ đó bằng cách đưa ra những từ đề xuất. Mỗi từ đề xuất sẽ được kiểm tra và trả về kết quả cho người chơi biết các chữ cái nào trong từ đó chính xác và ở đúng vị trí, và các chữ cái nào không thuộc từ đó.

Người chơi phải sử dụng kiến thức từ vựng và khả năng phân tích ngôn ngữ để suy luận và tìm ra từ bí mật trong số lượt đoán giới hạn. Mục tiêu cuối cùng là tìm ra từ bí mật trong số lượt đoán ít nhất.

Cơ chế hoạt động:

- Trong file GameController.java khởi tạo GameEngine, từ tiếng anh được chọn ngẫu nhiên và xử lý các sự kiện từ bàn phím và các nút để đưa vào các hàm chức năng của GameEngine để xử lý.

- Trong file GameEngine.java khởi tạo các biến và các hàm chức năng xử lý sự kiện chính của game:

Các biến khởi tạo của game:

```
public static final ArrayList<String> loadedWordList = new
        ArrayList<>();
private final int MAX_COL = 5;
private final int MAX_ROW = 6;
private int CUR_ROW = 1;
private int CUR_COLUMN = 1;
private String randomWord;
```

Tải và nạp danh sách từ vào game:

```
public void initGameWord() {
    InputStream wordList =
getClass().getResourceAsStream("game_Data.txt");

    if (wordList != null) {
        try (BufferedReader winningWordsReader = new
BufferedReader(new InputStreamReader(wordList))) {

winningWordsReader.lines().forEach(loadedWordList::add);
        } catch (IOException e) {
            System.out.println("Error reading winning
words.");
        }
    } else {
        System.out.println("Error loading winning words
file.");
    }
}
```

Chọn một từ ngẫu nhiên trong danh sách từ:

```
public void generateRandomWord() {
    randomWord = loadedWordList.get(new
Random().nextInt(loadedWordList.size()));
}
```

Khởi tạo các Label cho GridPane làm UI chính cho game:

```
public void createGameUI(GridPane gameMain) {
    for (int i = 1; i <= MAX_ROW; i++) {
        for (int j = 1; j <= MAX_COL; j++) {
            Label label = new Label();
            label.getStyleClass().add("label-default");
            gameMain.add(label, j, i);
        }
    }
}
```

Xử lý sự kiện game:

```
public void onKeyPressed(GridPane gameMain, KeyEvent
keyEvent) {
    Platform.runLater(() -> {
        KeyCode keyCode = keyEvent.getCode();
        if (keyCode == KeyCode.BACK_SPACE) {
            onBackSpacePressed(gameMain);
        } else if (keyCode.isLetterKey()) {
            onLetterPressed(gameMain, keyEvent);
        } else if (keyCode == KeyCode.ENTER) {
            onEnterPressed(gameMain);
        }
    });
}
```

Xử lý khi yêu cầu reset game:

```
public void resetGame(GridPane gameMain) {
    generateRandomWord();
    Label label;
    for (Node child : gameMain.getChildren())
        if (child instanceof Label) {
            label = (Label) child;
            label.getStyleClass().clear();
            label.setText("");
            label.getStyleClass().add("label-default");
        }
    CUR_COLUMN = 1;
    CUR_ROW = 1;
}
```

Đặc biệt, do danh sách từ nạp tải vào game kích thước khá lớn đồng thời danh sách từ của ta được sắp xếp theo thứ tự theo bảng chữ cái. Do đó để tăng tốc độ trong game đã cài đặt một hàm BinarySearch để hỗ trợ tìm kiếm cho game:

```
private boolean binarySearch(String string) {
    int i = 0;
    int j = GameEngine.loadedWordList.size() - 1;

    while (i <= j) {
        int mid = i + (j - i) / 2;
        int cmp =
string.compareTo(GameEngine.loadedWordList.get(mid));

        if (cmp == 0) {
            return true;
        } else if (cmp < 0) {
            j = mid - 1;
        } else {
            i = mid + 1;
        }
    }

    return false;
}
```

Đó là tổng quan qua cách hàm chức năng chính và cơ chế của game. Các file còn lại thi có chức năng như sau:

- Alert.java: File chương trình hiện lên alert cảnh báo khi người chơi nhập một từ không hợp lệ, không trong danh sách từ được nạp tải vào game.
- EndWindow.java: File chương trình hiện lên cửa sổ khi ta kết thúc trò chơi, khi ta thắng hoặc thua game thì game sẽ gọi cửa sổ này ra để tổng kết lại kết quả chơi của ta.

b. GoogleTranslate

Set up các biến và API cần thiết:

```
Online output = new Online();
GoogleTranslateAPI api = new GoogleTranslateAPI();
private final String API_KEY = output.connectionType();
private String SRC_LANG = "en";
private String OUT_LANG = "vi";
```

Dịch văn bản:

```
@FXML
public void translateTextInput() throws IOException {
    String input = inputArea.getText();
    if (input != null) {
```

```
            try {
                // Dich bang Google API
                Translate translateTest =
    TranslateOptions.newBuilder().setApiKey("API_KEY").build().ge
    tService();
                Translation translation =
    translateTest.translate(input,
    Translate.TranslateOption.sourceLanguage(SRC_LANG),
    Translate.TranslateOption.targetLanguage(OUT_LANG));
                String translatedText =
    translation.getTranslatedText();
                outputArea.setText(translatedText);
            } catch (Exception e1) {
                try {
                    // Dich bang unoffical google API
                    String translatedText =
    GoogleTranslate.translate(SRC_LANG, OUT_LANG, input);
                    outputArea.setText(translatedText);
                } catch (Exception e2) {
                    System.out.println("Khong tim thay tu");
                }
            }
        }
    }
```

Phát âm thanh:

```
@FXML
public void playAudioIn() {
    String tmp = inputArea.getText();
    usingOnlineSpeak(tmp);
}

@FXML
public void playAudioOut() {
    String tmp = outputArea.getText();
    usingOnlineSpeak(tmp);
}
```

Đổi ngôn ngữ:

```
@FXML
public void SwitchTrans() {
    String tmp = SRC_LANG;
    SRC_LANG = OUT_LANG;
    OUT_LANG = tmp;

    // Hoán đổi nội dung của hai nhãn Label
    String lb1 = Label1.getText();
```

```
    String lb2 = Label2.getText();
    Label1.setText(lb2);
    Label2.setText(lb1);

    // Kiểm tra xem outputArea có rỗng hay không
    if (!outputArea.getText().isEmpty()) {
        // Hoán đổi nội dung của hai vùng văn bản inputArea và
outputArea
        String in = inputArea.getText();
        String out = outputArea.getText();
        inputArea.setText(out);
        outputArea.setText(in);
    }
}
```

c. Modify

- Thêm từ

```
@FXML
void addWord(ActionEvent event) throws IOException {
    // Create a new Stage (popup window)
    Stage popupStage = new Stage();
    popupStage.setTitle("Add Word");
    FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("AddWord/addWord.fxml"));
    Parent root = fxmlLoader.load();
    Scene scene = new Scene(root);
    popupStage.setScene(scene);
    popupStage.show();
    popupStage.setResizable(false);
}

public void addedword() {
    String newWord1 = newWord.getText();
    String newWordType1 = newWordType.getText();
    String newMeaning1 = newMeaning.getText();

    if (newWord1 == null || newWord1.trim().isEmpty() ||
            newMeaning1 == null || newMeaning1.trim().isEmpty())
{
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Warning!");
        alert.setHeaderText(null);
        alert.setContentText("No word has been added or the word
has no meaning!");
        alert.showAndWait();
    } else {
        output.insertWord(newWord1, newWordType1, newMeaning1);
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Done!");
        alert.setHeaderText(null);
```

```
            alert.setContentText("Add new word successfully!");
            alert.showAndWait();
            modifyController.getInstance().updateListView();
        }
    }
```

- Sửa từ

```
    @FXML
    void editWord(ActionEvent event) throws IOException {
        String searchWord = searchBar.getText().trim();
        if (searchWord.isEmpty()) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Wait!");
            alert.setHeaderText(null);
            alert.setContentText("Please enter a word");
            alert.showAndWait();
        } else {
            // Create a new Stage (popup window)
            Stage popupStage = new Stage();
            popupStage.setTitle("Edit Word");
            FXMLLoader fxmlLoader = new
    FXMLLoader(getClass().getResource("EditWord/EditWord.fxml"));
            Parent root = fxmlLoader.load();
            Scene scene = new Scene(root);
            popupStage.setScene(scene);
            popupStage.show();
            popupStage.setResizable(false);
            popupStage.setResizable(false);
        }
    }

    public void editWord() {
        newWord.setEditable(true);
        newWordType.setEditable(true);
        newMeaning.setEditable(true);
    }

    public void setConfirmButton() {
        String newWord1 = newWord.getText();
        String newWordType1 = newWordType.getText();
        String newMeaning1 = newMeaning.getText();
        output.editWord(wordtoedit, newWord1, newWordType1,
    newMeaning1);
        newWord.setEditable(false);
        newWordType.setEditable(false);
        newMeaning.setEditable(false);
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Done!");
        alert.setHeaderText(null);
        alert.setContentText("This word has been edited
    successfully!");
```

```
        alert.showAndWait();
        modifyController.getInstance().updateListView();
    }
```

- Xóa từ

```
    @FXML
    public void deleteWord() {
        output.deleteWord(wordtoedit);
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Done");
        alert.setHeaderText(null);
        alert.setContentText("This word has been deleted
    successfully!");
        alert.showAndWait();
        modifyController.getInstance().deleteView();
    }
```

d. PhrasalVerb

```
    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {

        textOut.setEditable(false);
        textOut.setWrapText(true);

        inputtext.setPromptText("Enter a word");

        searchBtn.setOnAction(event -> search(null));

        listView.setCellFactory(lv -> {
            ListCell<String> cell = new ListCell<>();
            cell.textProperty().bind(cell.itemProperty());
            cell.setStyle(
                    "-fx-background-color: transparent; " +
                            "-fx-border-radius: 15%;" +
                            "-fx-font-family: 'Comic Sans MS';" +
                            "-fx-font-size: 14px;");
            return cell;
        });


        Set<String> allPhrasesOfflineSet = new
    TreeSet<>(output.getAllPhrasesOffline());
        listView.getItems().addAll(allPhrasesOfflineSet);

        inputtext.textProperty().addListener((observable, oldValue,
    newValue) -> {
            listView.getItems().clear();
            String searchWord = newValue.trim();
```

```
        if (searchWord.isEmpty()) {

listView.getItems().addAll(output.getAllPhrasesOffline());
        } else {
            try {
                List<String> searchResults =
output.searchList(searchWord,
                        allPhrasesOfflineSet);
                if (searchResults.isEmpty()) {
                    performFuzzySearch(searchWord,
allPhrasesOfflineSet);
                } else {
                    listView.getItems().addAll(searchResults);
                }
            } catch (NullPointerException e) {
                listView.getItems().clear();
                Label noword = new Label("Word not available");
                noword.setStyle("-fx-font-size: 14; -fx-text-
fill: black;");
                listView.setPlaceholder(noword);
            }
        }
    });

    listView.setOnMouseClicked(event -> {
        String selectedText =
listView.getSelectionModel().getSelectedItem();
        try {
            if (selectedText != null) {

textOut.setText(output.searchPhrase(selectedText));
            }
        } catch (Exception e) {
            listView.getItems().addAll(allPhrasesOfflineSet);
        }
    });
}
```

e. Search

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {

    textOut.setEditable(false);
    textOut.setWrapText(true);

    inputtext.setPromptText("Enter a word");

    searchBtn.setOnAction(event -> search(null));

    Set<String> allWordsOfflineSet = new
```

```
TreeSet<>(output.getAllWordsOffline());

    if (trie != null) {
        allWordsOfflineSet.forEach(trie::insert);
    } else {
        System.out.println("Trie has not been initialized");
    }

    listView.setCellFactory(lv -> {
        ListCell<String> cell = new ListCell<>();
        cell.textProperty().bind(cell.itemProperty());
        cell.setStyle(
                "-fx-background-color: transparent; " +
                        "-fx-border-radius: 15;" +
                        "-fx-font-family: 'Comic Sans MS';" +
                        "-fx-font-size: 14px;");
        return cell;
    });

    Trie allWordsOfflineTrie = new Trie();
    for (String word : output.getAllWordsOffline()) {
        allWordsOfflineTrie.insert(word);
    }
    listView.getItems().addAll(allWordsOfflineSet);

    inputtext.textProperty().addListener((observable, oldValue,
newValue) -> {
        listView.getItems().clear();
        String searchWord = newValue.trim();
        if (searchWord.isEmpty()) {

listView.getItems().addAll(output.getAllWordsOffline());
        } else {
            try {
                List<String> searchResults =
allWordsOfflineTrie.searchPrefix(searchWord);
                if (searchResults.isEmpty()) {
                    performFuzzySearch(searchWord,
allWordsOfflineTrie);
                } else {
                    listView.getItems().addAll(searchResults);
                }
            } catch (NullPointerException e) {
                listView.getItems().clear();
                Label noword = new Label("Word not available");
                noword.setStyle("-fx-font-size: 14; -fx-text-
fill: black;");
                listView.setPlaceholder(noword);
            }
        }
    });
```

```
        listView.setOnMouseClicked(event -> {
            String selectedText =
    listView.getSelectionModel().getSelectedItem();
            if (selectedText != null) {
                textOut.setText(output.searchWord(selectedText));
            } else {
                for (String word : output.getAllWordsOffline()) {
                    allWordsOfflineTrie.insert(word);
                }
            }
        });
    }
```

f. Translate

Đây chính là phần sử dụng API nằm trong package API đã được khởi tạo và cài đăt bên trên

```
@FXML
private void handleTranslateButton() {
    word = inputWord.getText().trim();
    if (word != null) {
        String translationResult = output.searchWord(word);
        displayRes.setText(translationResult);
    }
}
```

```
import static Audio.OnlineAudio.usingOnlineSpeak;
@FXML
private void playAudio() {
    String text = word;
    if (text != null) {
        usingOnlineSpeak(text);
    }
}
```

Phần audio cho phần phát âm này được sử dụng qua import phương thức tĩnh từ package Audio đã được cài đặt từ trước đó

g. Cấu trúc dữ liệu và giải thuật

** Phần PhrasalVerb và Search thực hiện đưa tất cả các từ đã có trong cơ sở dữ liệu vào một Trie, từ đó bắt đầu sử dụng thuật toán tìm kiếm tương đồng. Với*

mỗi kí tự mà người dùng nhập trên thanh tìm kiếm, các từ liên quan sẽ hiện ra theo bảng chữ cái

```
private void performFuzzySearch(String wordTarget, Trie trie) {
    List<String> similarWords =
trie.getAllWords().parallelStream()
            .filter(word -> output.calculateSimilarity(wordTarget,
word) < 0.56)
            .sorted((a, b) ->
Double.compare(output.calculateSimilarity(wordTarget, b),
                    output.calculateSimilarity(wordTarget, a)))
            .collect(Collectors.toList());

    if (!similarWords.isEmpty()) {
        listView.getItems().addAll(similarWords);
    } else {
        listView.getItems().clear();
        Label noword = new Label("Word not available");
        noword.setStyle("-fx-font-size: 14; -fx-text-fill:
black;");
        listView.setPlaceholder(noword);
        listView.setPrefHeight(360.0);
    }

}
```

→ Ý tưởng: Với các chuỗi kí tự được nhập từ bàn phím, trong trường hợp các kí tự đó không xuất hiện trong các từ của từ điển, thuật toán này sẽ tính độ tương đồng và đề xuất ra những từ có liên quan với độ tương đồng đáng tin cậy và sắp xếp chúng theo thứ tự giảm dần của độ tương đồng(ở đây đang lấy độ tương đồng phù hợp là nhỏ hơn 0,56).

Trong đoạn code trên, parallelStream() được sử dụng để thực hiện các thao tác tìm kiếm mờ một cách song song trên tất cả các từ trong Trie. Sử dụng luồng song song giúp tăng hiệu suất bằng cách phân chia công việc xử lý lọc và sắp xếp các từ tương tự vào nhiều luồng khác nhau, cho phép chúng được xử lý đồng thời trên các lõi CPU khác nhau.

```
public double calculateSimilarity(String s1, String s2) {
    int distance = levenshteinDistance(s1, s2);
    return 1 - Math.pow(1 - (double) distance /
Math.max(s1.length(), s2.length()), 2);
}
```

+) Sử dụng công thức đặc biệt để tính toán độ tương đồng giữa hai chuỗi ký tự, s1 và s2, dựa trên khoảng cách Levenshtein. Công thức này được thiết kế để chuyển đổi khoảng cách Levenshtein thành một giá trị tương đồng, nằm trong khoảng từ 0 đến 1, với 1 là hoàn toàn giống nhau và 0 là hoàn toàn khác biệt.

+) Ở đây, distance là khoảng cách Levenshtein giữa hai chuỗi, được tính bởi hàm levenshteinDistance. Giá trị này được chia cho độ dài lớn nhất giữa hai chuỗi để chuẩn hóa khoảng cách, đảm bảo rằng nó nằm trong khoảng từ 0 đến 1. Sau đó, giá trị này được trừ đi từ 1 và bình phương để tăng cường sự phân biệt giữa các cặp chuỗi có độ tương đồng cao và thấp. Khi khoảng cách Levenshtein nhỏ (tức là hai chuỗi tương đồng cao), giá trị tương đồng sẽ tiệm cận về 1. Ngược lại, khi khoảng cách lớn (hai chuỗi khác biệt nhiều), giá trị tương đồng sẽ tiệm cận về 0.

```
public int levenshteinDistance(String s1, String s2) {
    int n = s1.length();
    int m = s2.length();
    int[][] dp = new int[n + 1][m + 1];

    for (int i = 0; i <= n; i++) {
        dp[i][0] = i;
    }

    for (int j = 0; j <= m; j++) {
        dp[0][j] = j;
    }

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
                dp[i][j] = dp[i - 1][j - 1];
            } else {
                dp[i][j] = 1 + Math.min(dp[i - 1][j - 1],
Math.min(dp[i - 1][j], dp[i][j - 1]));
            }
        }
    }
    return dp[n][m];
}
```

+) Ở đây, dp là một ma trận hai chiều, với dp[i][j] là khoảng cách Levenshtein giữa chuỗi con s1[0..i-1] và s2[0..j-1]. Ma trận được khởi tạo sao cho dp[i][0] và dp[0][j] tương ứng với số lượng thao tác cần thiết để biến đổi chuỗi con rỗng thành chuỗi con s1[0..i-1] và s2[0..j-1].

+) Trong vòng lặp, chúng ta so sánh từng cặp ký tự của hai chuỗi. Nếu chúng giống nhau, không cần thao tác nào và khoảng cách không thay đổi. Nếu chúng khác nhau, chúng ta cần thực hiện một trong ba thao tác (chèn, xóa, hoặc thay thế) với chi phí thấp nhất để tiếp tục quá trình biến đổi.

=> Kết quả trả về là giá trị của dp[n][m], tức là khoảng cách Levenshtein giữa hai chuỗi đầy đủ s1 và s2. Đây là số lượng thao tác tối thiểu cần thiết để biến đổi s1 thành s2.

** Xây dựng cấu trúc dữ liệu Trie*

```
public class TrieNode {
    Map<Character, TrieNode> children;
    String word;
    TrieNode() {
        this.children = new HashMap<>();
        this.word = null;
    }
}

public class Trie {
    TrieNode root;

    public Trie() {
        this.root = new TrieNode();
    }

    public void insert(String word) {
        if (word == null || word.isEmpty()) {
            return; // Skip insertion for empty or null words
        }
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node == null) {
                System.out.println("DEBUG: Node is null for word:
" + word);
                return;
            }
            if (node.children == null) {
                node.children = new HashMap<>();
            }
            node.children.putIfAbsent(c, new TrieNode());
            node = node.children.get(c);
        }
        node.word = word;
    }
```

```
    public List<String> searchPrefix(String prefix) {
        TrieNode node = root;
        for (char c : prefix.toCharArray()) {
            node = node.children.get(c);
            if (node == null) {
                return List.of();
            }
        }
        return findAllWords(node);
    }

    private List<String> findAllWords(TrieNode node) {
        if (node == null) {
            return new ArrayList<>();
        }
        List<String> words = new ArrayList<>();
        if (node.word != null) {
            words.add(node.word);
        }
        for (TrieNode child : node.children.values()) {
            words.addAll(findAllWords(child));
        }
        return words;
    }

    public List<String> getAllWords() {
        return findAllWords(root);
    }
}
```

→ Ý tưởng: Cấu trúc dữ liệu Trie trong đoạn code trên đóng vai trò quan trọng trong việc lưu trữ và tìm kiếm các từ một cách hiệu quả. Trie, còn được gọi là cây tiền tố, là một cấu trúc dữ liệu dạng cây mà mỗi nút đại diện cho một ký tự từ một từ cụ thể. Trie cho phép tìm kiếm nhanh chóng các từ có chung tiền tố. Trie cung cấp hiệu suất tìm kiếm nhanh hơn so với việc sử dụng danh sách hoặc tập hợp thông thường, đặc biệt khi làm việc với số lượng lớn từ.

- Lớp Trie bao bọc cấu trúc dữ liệu trie và cung cấp các phương thức để tương tác với nó.
- Biến root là một thể hiện của TrieNode phục vụ như điểm nhập vào trie.
- Phương thức insert nhận một từ String làm đầu vào và chèn nó vào trie. Nó lặp qua từng ký tự trong từ, điều hướng qua trie và tạo nút mới khi cần thiết. Khi kết thúc từ được đạt tới, biến word của nút cuối cùng sẽ được thiết lập là từ đã chèn.

- Phương thức searchPrefix tìm kiếm tất cả các từ trong trie bắt đầu với một tiền tố cho trước. Nó điều hướng qua trie sử dụng các ký tự của tiền tố. Nếu kết thúc của tiền tố được đạt tới trong trie, nó sẽ gọi phương thức findAllWords để lấy tất cả các từ phân nhánh từ nút đó.
- Phương thức findAllWords là một hàm trợ giúp đệ quy đi qua trie từ một nút cho trước và thu thập tất cả các từ nó tìm thấy. Nó thêm từ được lưu trữ tại nút hiện tại vào một danh sách và sau đó gọi đệ quy chính nó trên tất cả các nút con.
- Phương thức getAllWords là một phương thức tiện ích trả về tất cả các từ được lưu trữ trong trie bằng cách gọi findAllWords bắt đầu từ nút gốc.
- Việc sử dụng HashMap cho các nút con cho phép ánh xạ ký tự-nút hiệu quả, làm cho các thao tác nhanh chóng và có thể mở rộng.

Dưới đây là phần đưa Trie vào sử dụng trong phần PhrasalVerb và Search

```
private Trie trie = new Trie();

Set<String> allWordsOfflineSet = new
TreeSet<>(output.getAllWordsOffline());
if (trie != null) {
    allWordsOfflineSet.forEach(trie::insert);
} else {
    System.out.println("Trie has not been initialized");
}

// Other section …

Trie allWordsOfflineTrie = new Trie();
    for (String word : output.getAllWordsOffline()) {
        allWordsOfflineTrie.insert(word);
    }
    listView.getItems().addAll(allWordsOfflineSet);

// Start searching section …
```

- Một tập hợp allWordsOfflineSet được tạo để lưu trữ tất cả các từ từ nguồn output.getAllWordsOffline() dưới dạng TreeSet (để có thứ tự theo bảng chữ cái).
- Một đối tượng trie được sử dụng để chứa cây trie của các từ từ tập hợp allWordsOfflineSet.
- Đoạn mã sau đó kiểm tra xem trie đã được khởi tạo chưa và chèn từng từ từ allWordsOfflineSet vào cây trie.

- Một đối tượng allWordsOfflineTrie khác được tạo và mọi từ từ output.getAllWordsOffline() được chèn vào cây trie này.
- Một ListView được sử dụng để hiển thị các từ và có một trình lắng nghe để xử lý tìm kiếm dựa trên nội dung nhập.
- Sau đó sẽ là phần sử dụng giải thuật tìm kiếm nêu trên để bắt đầu công việc tìm kiếm từ vựng

h. Dữ liệu và cơ sở dữ liệu

- Tại bảng addedword, phần từ vựng được thêm vào với việc tạo bảng và sử dụng truy vấn với sự hỗ trợ của Console trong MySQL Workbench, phần truy vấn thêm dữ liệu được tham khảo tại

  MySQL English Dictionary download | SourceForge.net

- Tại bảng phrasalverb, phần dữ liệu được sử dụng tham khảo tại

  WithEnglishWeCan/generated-english-phrasal-verbs: [public][generated-english-phrasal-verbs] (github.com)

  Do dữ liệu nằm trong một file JSON có cấu trúc, việc truy xuất và thêm dữ liệu vào cơ sở dữ liệu cần một chương trình con riêng để xử lí "UpdatePhrasal.java"

```
public class UpdatePhrasal {
    public static void main(String[] args) {
        UpdatePhrasal app = new UpdatePhrasal();
        app.readJSONAndInsertToDatabase();
    }

    public Connection connection() {
        String dbPath = "./Dictionary.db";
        Connection connection = null;
        try {
            connection =
DriverManager.getConnection("jdbc:sqlite:" + dbPath);

        } catch (SQLException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
        return connection;
```

```
    }

    public void readJSONAndInsertToDatabase() {
        try {
            BufferedReader input = new BufferedReader(new
FileReader("src/phrasal.verbs.build.json"));
            String inputLine;
            StringBuffer wordDetail = new StringBuffer();
            while ((inputLine = input.readLine()) != null) {
                wordDetail.append(inputLine);
            }

            JSONArray jsonArray = new
JSONArray(wordDetail.toString());

            Connection conn = this.connection();
            String insertQuery = "INSERT INTO phrasalverb (phrv,
derivative, meaning, examples, " +
                    "synonyms) " +
                    "VALUES " +
                    "(?, ?, ?, ?, ?)";
            PreparedStatement pstmt =
conn.prepareStatement(insertQuery);

            JSONObject inside = jsonArray.getJSONObject(0);

            Iterator<String> keys = inside.keys();

            while (keys.hasNext()) {
                String key = keys.next();
                JSONObject wordObject = inside.getJSONObject(key);
                pstmt.setString(1, key);


                if (wordObject.has("derivatives")) {
                    JSONArray derivatives =
wordObject.getJSONArray("derivatives");
                    StringBuilder derive = new StringBuilder();
                    for (int i = 0; i < derivatives.length(); i++)
{

derive.append(derivatives.get(i)).append(",");
                    }
                    pstmt.setString(2, derive.toString());
                } else {
                    pstmt.setString(2, "Now available now");
                }

                if (wordObject.has("descriptions")) {
                    JSONArray descriptions =
wordObject.getJSONArray("descriptions");
```

```
                    StringBuilder des = new StringBuilder();
                    for (int i = 0; i < descriptions.length();
i++) {

des.append(descriptions.get(i)).append("\n");
                    }
                    pstmt.setString(3, des.toString());
                } else {
                    pstmt.setString(3, "Now available now");
                }

                if (wordObject.has("examples")) {
                    JSONArray examples =
wordObject.getJSONArray("examples");
                    StringBuilder ex = new StringBuilder();
                    for (int i = 0; i < examples.length(); i++) {
                        ex.append(examples.get(i)).append("\n");
                    }
                    pstmt.setString(4, ex.toString());
                } else {
                    pstmt.setString(4, "Now available now");
                }

                if (wordObject.has("synonyms")) {
                    JSONArray synonyms =
wordObject.getJSONArray("synonyms");
                    StringBuilder syn = new StringBuilder();
                    for (int i = 0; i < synonyms.length(); i++) {
                        syn.append(synonyms.get(i)).append(",");
                    }
                    pstmt.setString(5, syn.toString());
                } else {
                    pstmt.setString(5, "Now available now");
                }

                pstmt.executeUpdate();
            }
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();

        }
    }
}
```

- Mở đầu là phần khởi tạo một kết nối với cơ sở dữ liệu mà chương trình sử dụng, sau đó mới phần đọc file và đưa dữ liệu và cơ sở dữ liệu

- Với cấu trúc các Object con cùng cấp với nhau trong một Object lớn (một từ), việc xử lí chỉ cần gọi ra các khóa tương ứng với Object để lấy dữ liệu: `"derivatives”`, `"descriptions"`, `"examples"`, và `"synonyms"`

```
"abide by": {
  "derivatives": [
    "abides by",
    "abiding by",
    "abided by"
  ],
  "descriptions": [
    "to follow a rule, decision, or instruction"
  ],
  "examples": [
    "Ah, surely,' Maram said, `this is the time to abide by the
letter of his rule?",
    "They promised to abide by the rules of the contest."
  ],
  "synonyms": [
    "agree to",
    "carry out",
    "follow",
    "obey"
  ]
},

// Other Objects …
```

- Trong mỗi Object con, dữ liệu được lưu ở dạng mảng gồm một hoặc nhiều các xâu, nên để lấy được đầy đủ, chỉ cần sử dụng một StringBuilder để append lần lượt các xâu nằm trong mảng đó, sau cùng sẽ truyền tham số vào truy vấn là xâu kết quả tử StringBuilder đã tạo
- Các Object lớn có thể không chứa Object con cần tìm để đưa vào cơ sở dữ liệu, với những trường hợp như vậy, để tránh giá trị null (khó xử lí và có nhiều vấn đề) xuất hiện trong cơ sở dữ liệu, dùng

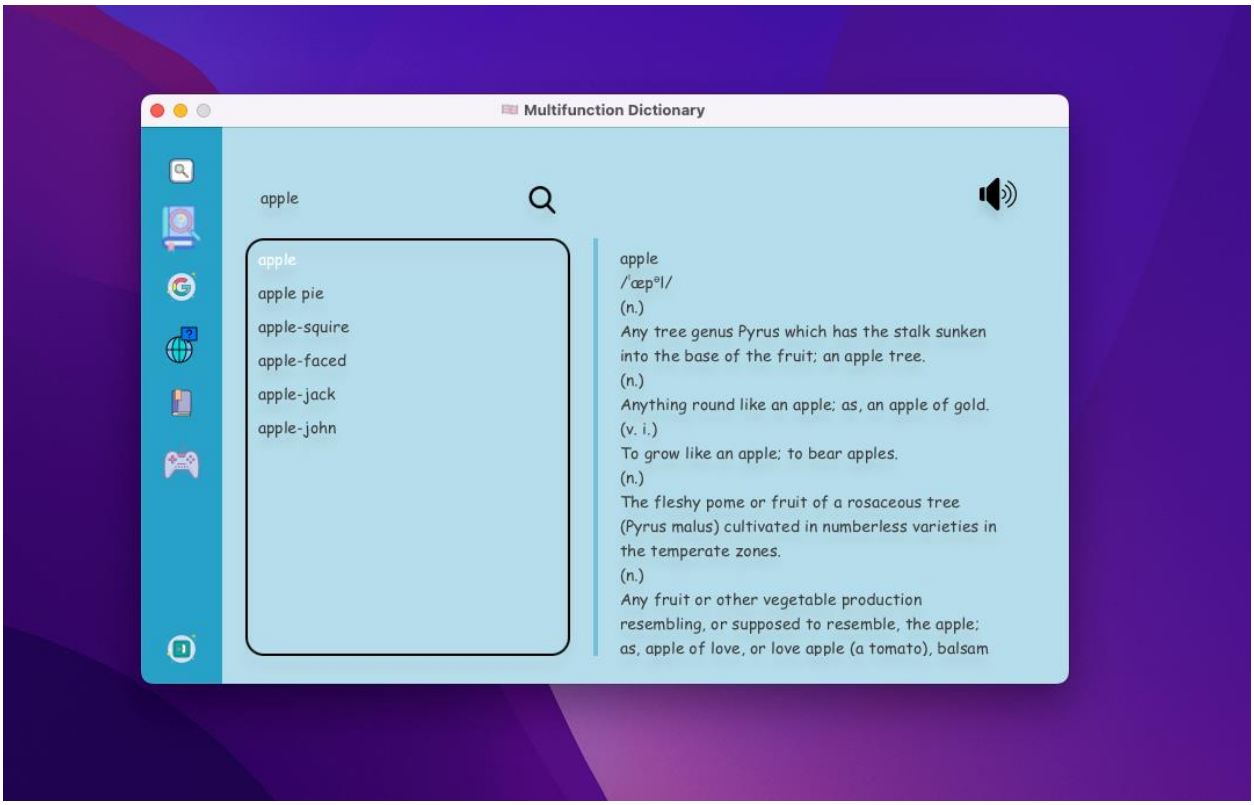
```
pstmt.setString(index, "Now available now");
```

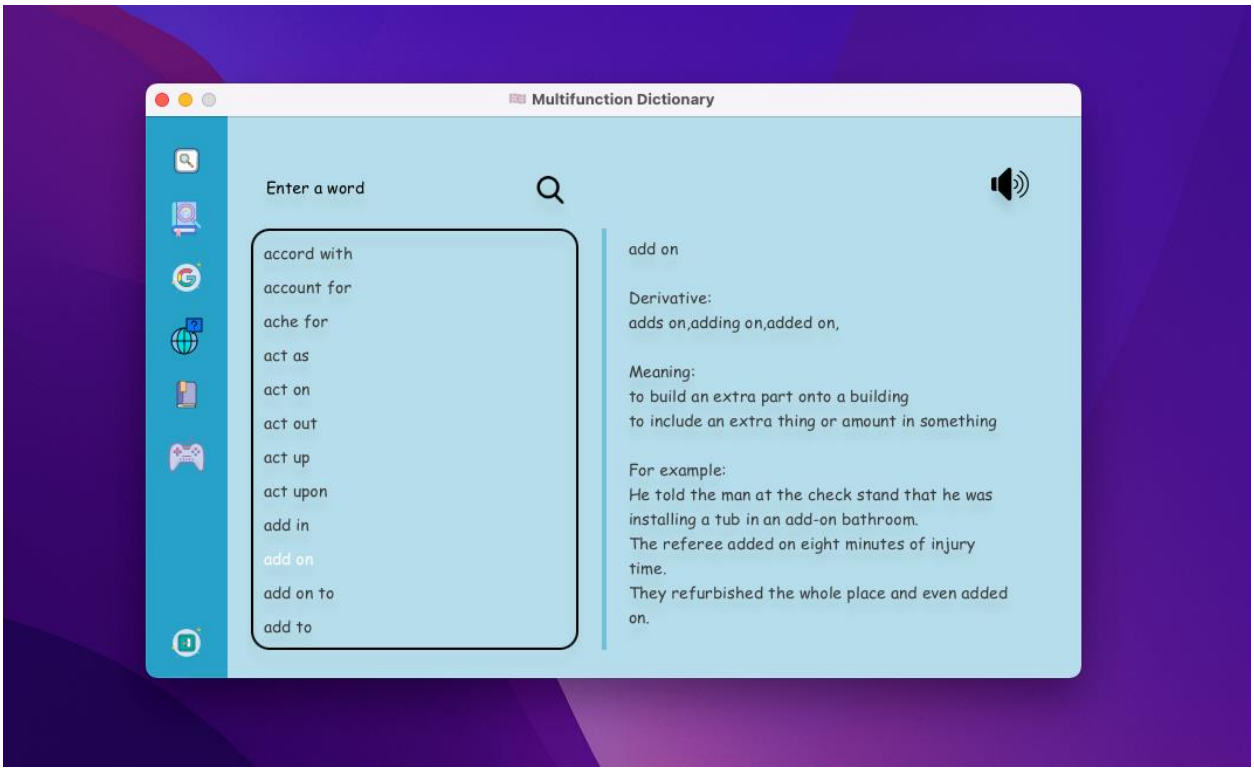
để đưa tham số đầu vào cho truy vấn là `"Now available now"`, vẫn đảm bảo thông báo sự không có mặt của dữ liệu, và tránh sự xuất hiện của giá trị null trong cơ sở dữ liệu

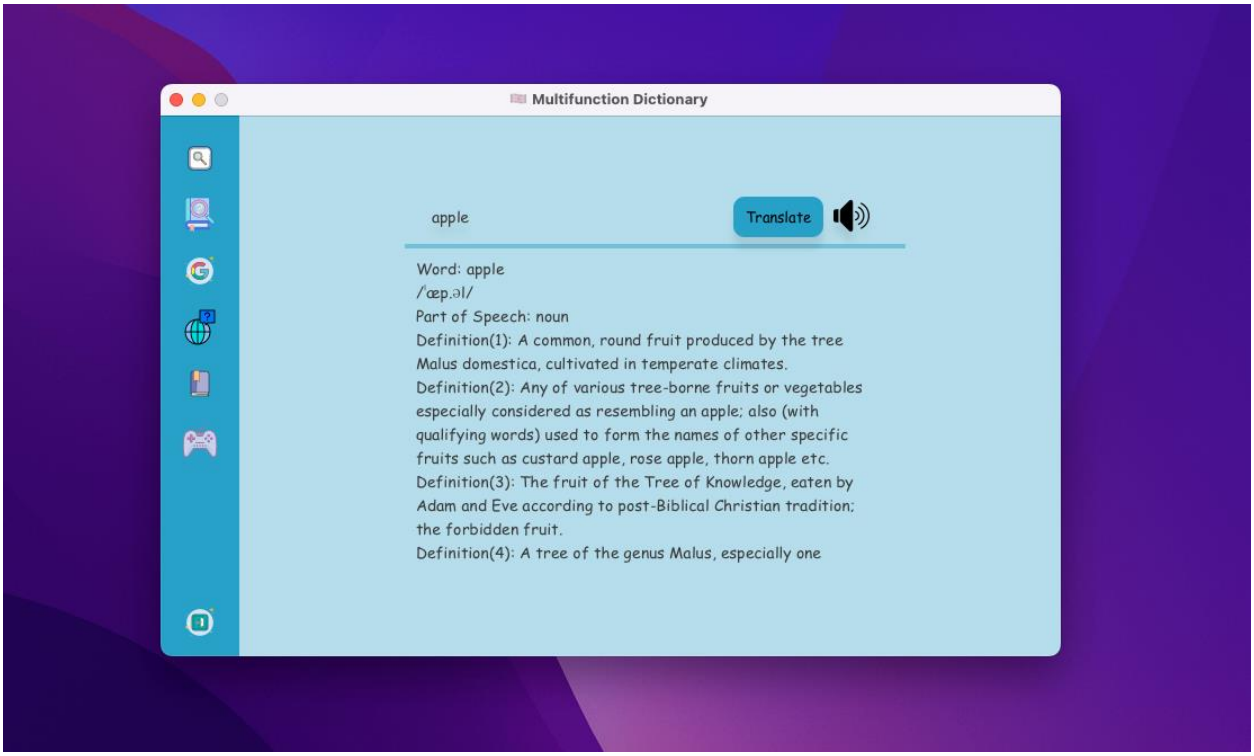
- Sau khi đã xử lí xong, dữ liệu được đưa vào bảng phrasalverb:

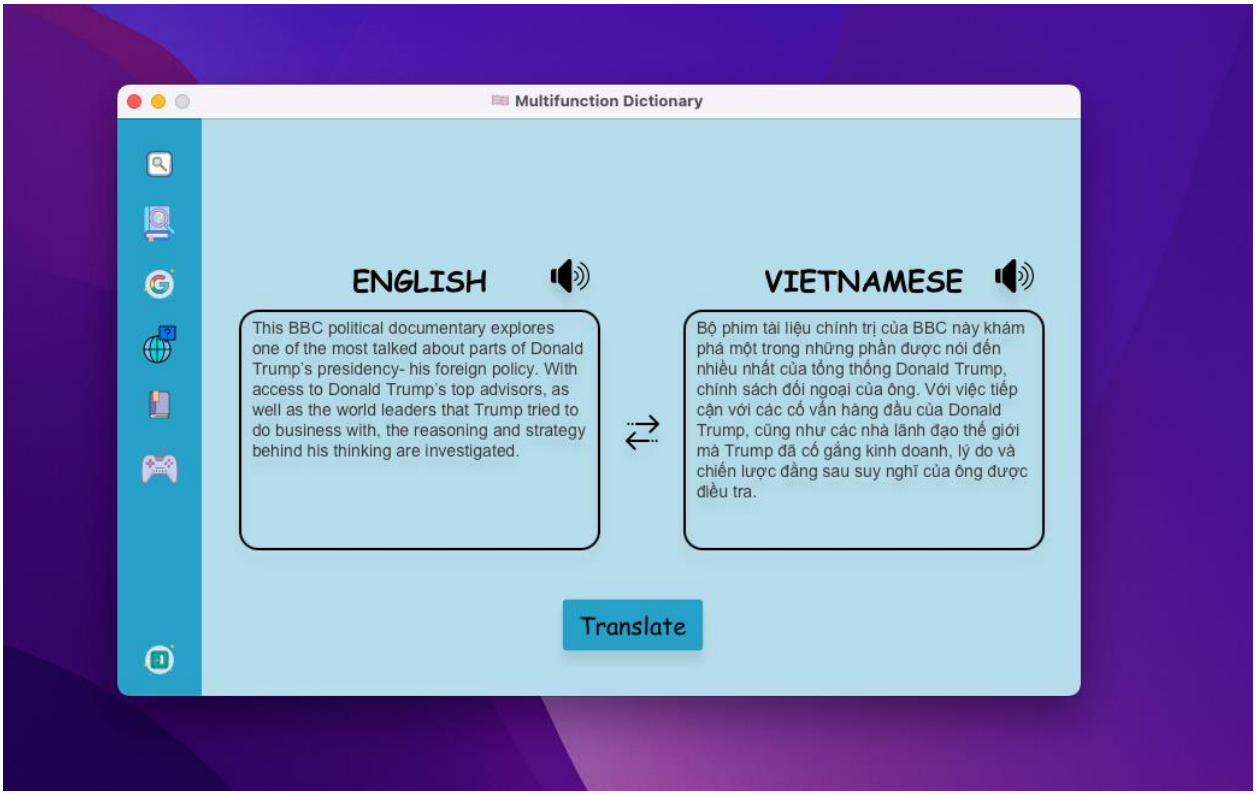
	phrv	derivative	meaning	examples	synonyms
1	throw back on	throws back on,throwing …	to force someone to use…	When the business failed he was thrown b…	(Not available now)
2	pass round	passes round,passing rou…	to collect money from p…	The photos were passed round for each of…	(Not available now)
3	keep down	keeps down,keeping down,…	to control people in su…	Even if you're intelligent, they still t…	burden,depress,harass,oppress,

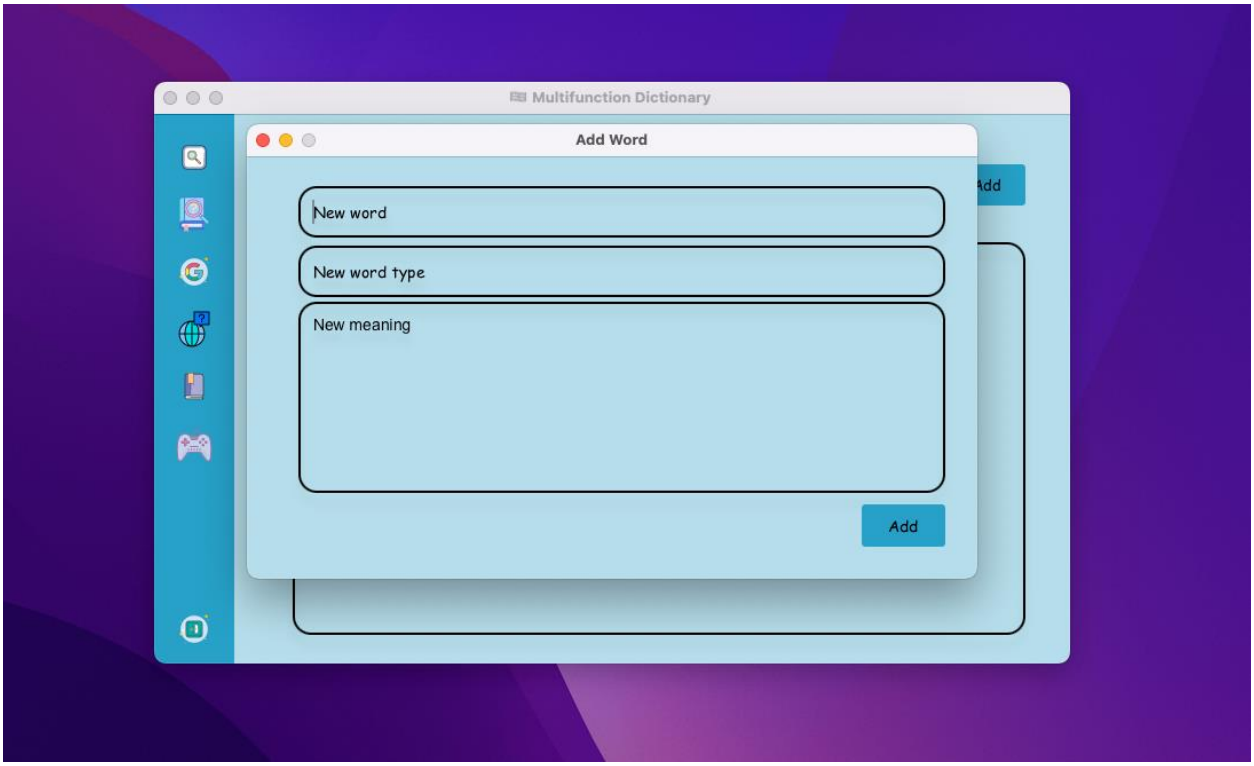
IV. Phần DEMO cho ứng dụng

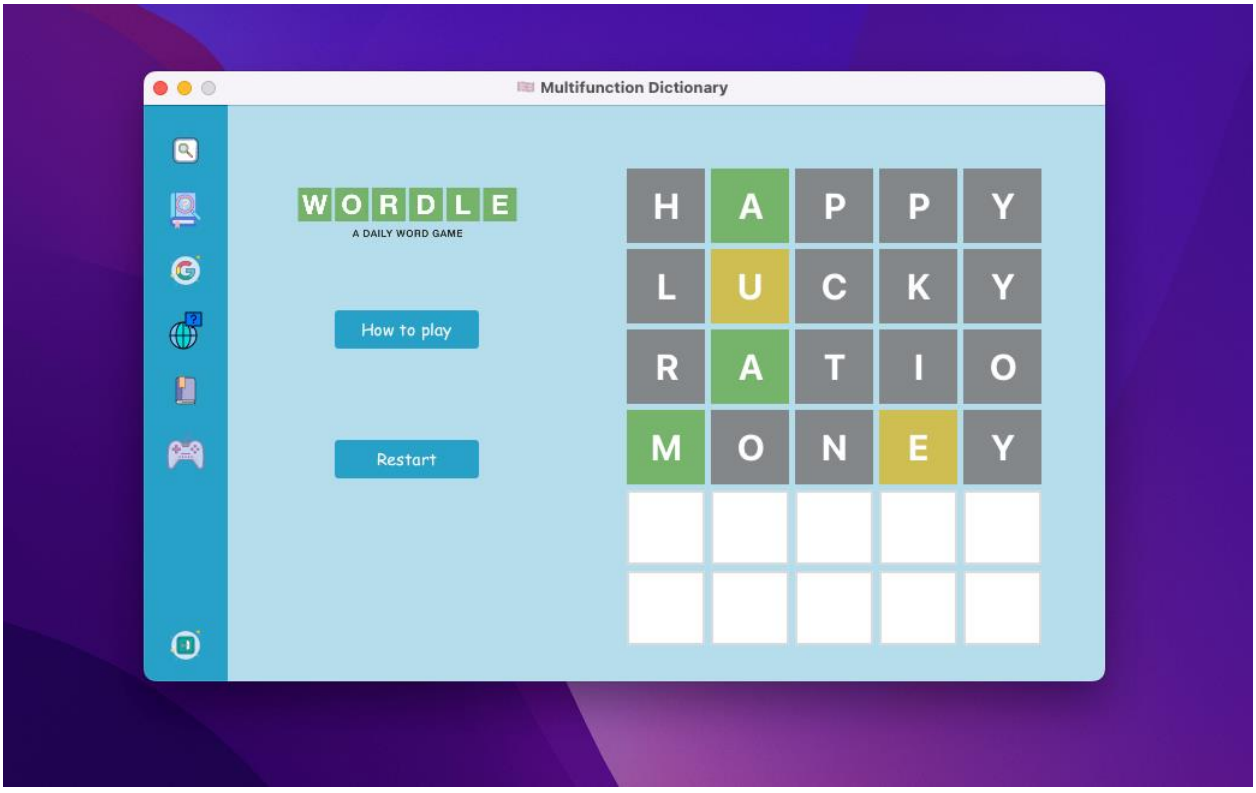





Multifunction Dictionary
apple
Translate
Word: apple
/ˈæp.əl/
Part of Speech: noun
Definition(1): A common, round fruit produced by the tree Malus domestica, cultivated in temperate climates.
Definition(2): Any of various tree-borne fruits or vegetables especially considered as resembling an apple; also (with qualifying words) used to form the names of other specific fruits such as custard apple, rose apple, thorn apple etc.
Definition(3): The fruit of the Tree of Knowledge, eaten by Adam and Eve according to post-Biblical Christian tradition; the forbidden fruit.
Definition(4): A tree of the genus Malus, especially one


Multifunction Dictionary
ENGLISH
VIETNAMESE
This BBC political documentary explores one of the most talked about parts of Donald Trump's presidency- his foreign policy. With access to Donald Trump's top advisors, as well as the world leaders that Trump tried to do business with, the reasoning and strategy behind his thinking are investigated.
Bộ phim tài liệu chính trị của BBC này khám phá một trong những phần được nói đến nhiều nhất của tổng thống Donald Trump, chính sách đối ngoại của ông. Với việc tiếp cận với các cố vấn hàng đầu của Donald Trump, cũng như các nhà lãnh đạo thế giới mà Trump đã cố gắng kinh doanh, lý do và chiến lược đằng sau suy nghĩ của ông được điều tra.
Translate





Do phần audio không thể đưa trực tiếp vào đây với phần video ghi màn hình, nhóm em xin phép gửi link video đã lưu trên Google Drive, phần này bao gồm tất cả các phần DEMO của ứng dụng

V. Đóng góp của các thành viên

Các đề mục	Hoàng Bảo Long	Nguyễn Đức Mạnh	Lê Tuấn Kiệt
Package API			
Dictionary API	X		X
Google Translate API			X
Package Audio			
Online Audio			X
Offline Audio			X
Package DictionaryMethod			
Online	X		X
Offline	X	X	
Package DicFX			
Game Package			X
Google Trans Package			X
Modify Package		X	
PhrasalVerb Package	X		
Search Package	X	X	
Translate Package	X		X

Các mục khác			
Thuật toán chương trình		X	
Cơ sở dữ liệu	X		
Đồ họa chương trình	X	X	X

Phần bên trên là phần tổng quát về các phần mà mỗi thành viên tham gia và thực hiện, tùy theo chuyên môn và mức độ nặng nhẹ yêu cầu, sẽ có những phần chỉ có một thành viên đảm nhiệm, và có những phần cần nhiều thành viên góp sức.

Về tổng quan, các thành viên đều có đóng góp hết mình với khả năng cho bài tập lớn, nên đánh giá về đóng góp sẽ là như nhau với mỗi thành viên theo tỉ lệ đồng đều nhau.

VI. Nguồn tham khảo và các công nghệ đã sử dụng:

- https://stackoverflow.com/
- https://www.geeksforgeeks.org
- https://vnoi.info
- MySQL English Dictionary download | SourceForge.net
- WithEnglishWeCan/generated-english-phrasal-verbs: [public][generated-english-phrasal-verbs] (github.com)
- Free Dictionary API
- FreeTTS
- Google Cloud Translate
- Java Google Speech API 8.0.0

Cùng với đó là sự hỗ trợ của các thư viện và phần mềm:

- IntelliJ
- Scene Builder
- Java SDK
- JavaFX
- CSS
- Json Library
- SQLite JDBC Library
- Jlayer Library