Seliverstov Mark,
Summer semester 2022/2023
Advanced C# programming, NPRG038
Repository with App: “https://github.com/MarkSeliverstov/Echolalia"

Documentation (Echolalia)

1. Introduction

This document provides overview of **Echolalia** - the mobile application designed to help users practice and learn words in a foreign language. Echolalia is built on the principle of repeated repetition and training, making it an effective tool for language learners of all levels.

1.1 Problem Statement

Learning a new words in a foreign language can be challenging, especially without consistent practice and reinforcement. Many language learners struggle to retain new words and phrases, often leading to frustration and disengagement. Echolalia addresses this issue by leveraging the concept of echolalia, where users repetitively practice words and phrases to reinforce their memory.

1.2 Features Overview

Echolalia offers the following *key features* to facilitate effective language learning:

- *User Profile*: Personalised user profiles and user settings.
- *Dictionary*: Users can add custom words or memorise existing ones.
- *Exercises*: Various exercises are available to test language skills, including memorisation, multiple-choice tasks and practicing words using writing.
- *Statistics*: Users can view learning statistics in graphical form, helping them monitor their progress over time.
- *Settings*: The application offers customisable settings such as changing the language, adjusting exercise word count, and deleting the user account and associated data.

1.3 Project Goals

The primary goals of the Echolalia project include:

- Creating a mobile application for both Android and iOS platforms using the Xamarin framework.
- Creating backend for app (Tasks, database model, statistics, settings, dictionary).
- Creating frontend for app in Figma and implement it in Xamarin Forms.

2. User's Documentation

**See the screens section at the end of the documentation for a visual representation.*

2.1. Getting Started

Echolalia is available for both Android and iOS devices. To install the application

Android:
- Download **echolalia.apk** file from "https://github.com/MarkSeliverstov/Echolalia" on your phone and install it.

iOS:
- You must have an *Apple developer key* and you must build *release* it in VS. Or you can use an emulator in VS.

2.2 Main tabbed page

Navigation:
- The main page of the Application is designed with tabs at the bottom of the screen.
- These tabs allow you to navigate between different sections of the application.
- By default, you will see a home page.

Switching Tabs:
- To switch between tabs, you can tap on the desired tab at the bottom of the screen.
- Each tab represents a specific section of the application, such as Home page, Adding page, Dictionary, and Settings.

Using the Tabs:
- **Home page** displays count of day learned words and buttons with *Tasks (exercises).*
- **Adding page** allows allows you to add new words to the dictionary.
- **Dictionary page** shows your local dictionary and count of words in dictionary. By swiping to the right on any word, you can *delete* or *edit (edit word, mark as favorit, mark as learned)* it.
- **Statistics page** display statistics about words (and has 3 indicators: learned, in process, unknown), page also display buttons to *Settings page*.

2.3 Task pages

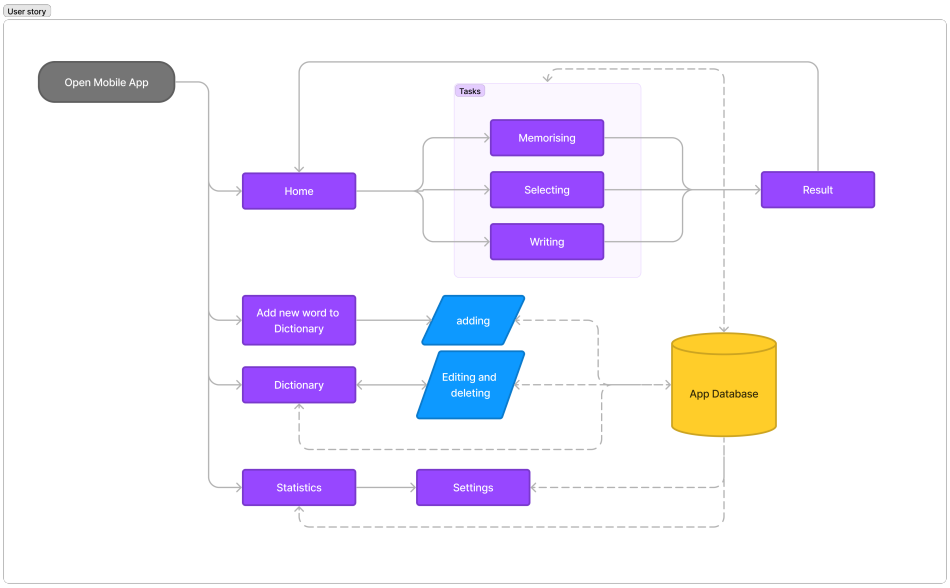
Currently, there are 3 tasks in the application with which you can practice new words by selecting on *Home page*:

- **Memorise** new words: the page where you can repeat unknown words.
- **Select** words: the page where you need to choose the correct answer from 4 optionsArchitecture and Technologies.
- **Writing** words: the page where you need to write the correct translation.

2.4 Settings Page

You can get to the settings page from the statistics page. On the settings page, you can choose several options:
- You can choose *Name* for yourself, which will be shown at the end of each task like this: "Good Job, <Your name>!". Default = "Guest"
- You can choose *how many words* each task should contain. Default = 20
- You can even choose the *interface language*. The truth is only to choose, because so far this choice does not affect anything. This choice is more for example. Default = "English"

3. Programmer’s Documentation



3.1 Architecture

The architecture of Echolalia is designed to ensure a robust and scalable application. It follows one of the best practices to separate application logic and user interface, making it easy to maintain, extend, and optimise the codebase. The key components of the Echolalia architecture include:

Model-View-ViewModel (MVVM) Design Pattern
Echolalia adopts the MVVM design pattern, a popular architectural pattern for developing cross-platform mobile applications. MVVM separates the application into three main components:

- *Model*: The model represents the data and business logic of the application. It includes data structures, data access, and other business-related components.
- *View*: The view is responsible for the user interface (UI) elements and layout. It communicates with the ViewModel to display data and receive user interactions.
- *ViewModel*: The ViewModel acts as an intermediary between the Model and the View. It manages the application's state, processes user inputs, and updates the Model or View accordingly.

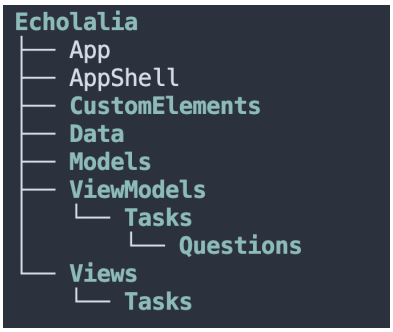
The *MVVM* pattern promotes a clean separation of concerns, enabling parallel development of UI and application logic while ensuring the application remains highly maintainable.

Xamarin and C# Language
Echolalia is built using the Xamarin framework, which allows the development of native Android and iOS applications using a single shared C# codebase.

SQLite Database
Echolalia uses SQLite as the local database for storing user-specific words and exercise data on the device. SQLite is a lightweight and self-contained database engine, making it an ideal choice for mobile applications. With SQLite, Echolalia supports offline functionality, enabling users to practice even without an internet connection.

3.2 Project structure

The project structure follows a modular and organized approach, making it easier for developers to navigate and maintain the codebase. It promotes separation of concerns and encourages the use of reusable components, contributing to a scalable and efficient application development process.



App
The App directory is the root of the application and it contains the main entry point some configuration and global settings. Also App navigate us to AppShell when app is launching.

AppShell
The AppShell directory is the root of the application too and it is contains configuration about ShellPage. It offers pre-defined navigation patterns and tabbed navigation.

CustomElements
The CustomElements directory contains custom user interface elements and controls. It include custom *Question and Controls* template.

Data
The Data directory deals with data-related functionalities, including data access, data storage, and data manipulation. It contain classes for interacting with local databases.

Models
The Models directory holds the data models that define the structure and properties of different data entities in the application. In this case, it includes the Word class.

ViewModels
ViewModels handle the application's logic and data presentation, acting as the intermediary between the Views and the Models. It is like the main “backend” of Echolalia.
- **Tasks** contains base Task and logics about each task.
  - **Questions** contains base Question and logic about each question.

Views
The Views directory contains the XAML files for the application's user interfaces. Each page or section of the app has a corresponding View file. The View is responsible for displaying data and user interface elements, following the structure defined in the ViewModel.
- **Tasks** contains the Views directory is further organized into subdirectories based on functional sections. In this case, there is the XAML files for different tasks-related view

3.3 Main data structure

Word model

The Word class defines the structure of individual words within the application. It contains several properties that store essential information related to each word:

```
public class Word
{
    [PrimaryKey, AutoIncrement]
    public int ID { get; set; }
    public string Original { get; set; }
    public string Translation { get; set; }
    public bool IsAddedByUser { get; set; }
    public LearningProgress Progress { get; set; }
    public DateTime LastPracticed { get; set; }
    public bool IsFavorite { get; set; }
}

public enum LearningProgress
{
    unknown,
    inProcess,
    learned
}
```

1. *ID*: This property serves as the primary key for the database and is automatically incremented when a new word is added to the database. Each word in the database will have a unique ID.

2. *Original*: Represents the word in the foreign language that the user wants to learn or practice.
3. *Translation*: Contains the translation of the foreign word in the native language. This property helps users understand the meaning of the word they are learning.
4. *IsAddedByUser*: A boolean flag indicating whether the word was added by the user. This property is useful for distinguishing between user-added words and pre-existing words in the application's database.
5. *Progress*: An enumeration representing the learning progress of the word. The "LearningProgress" enum defines three possible states: "unknown" (for when the learning progress is not determined yet), "inProcess" (when the word is currently being learned), and "learned" (when the user has successfully learned the word).
6. *LastPracticed*: Records the date and time when the user last practiced or interacted with the word. This information helps track the user's learning activity.
7. *IsFavorite*: A property indicating whether the user has marked the word as a favorite. This feature allows users to save and easily access words they find particularly significant or challenging.

3.4 Main solutions used in Echolalia

In Echolalia, several main solutions have been employed to create an easy-to-scale application for language learning. Here are the key solutions used:

1. **Base Task Page:**
   To ensure easy scalability, each Task page in Echolalia is based on the "Base Task Page" class. This approach allows developers to easily add new tasks to the application without rewriting common functionalities repeatedly.
   The BaseViewModel class in the Echolalia application is a fundamental class that implements the INotifyPropertyChanged interface. This base class provides a convenient and standardized way to handle property notifications and data binding within the ViewModel, ensuring a clean and organised codebase.

   - **`INotifyPropertyChanged` Implementation:**
     The BaseViewModel class implements the INotifyPropertyChanged interface. This interface is crucial for data binding in Xamarin.Forms applications. It allows the ViewModel to notify the View about changes in its properties, ensuring that the user interface is automatically updated when the underlying data changes.

   - **`OnPropertyChanged` Method:**
     The OnPropertyChanged method is responsible for raising the PropertyChanged event whenever a property's value changes. This event notifies the View of the property change, prompting the UI to update accordingly. The CallerMemberName attribute allows the method to automatically infer the property name when calling OnPropertyChanged, reducing the need for explicit property names in the calling code.

2. **Question Context:**
   Each Task in Echolalia has a "Question Context" class that facilitates the customization and addition of various question types. The "Question Context" class serves as a base class, enabling developers to seamlessly introduce new question types to the app. This approach fosters flexibility in handling diverse language exercises.

3. **Local Words Database and Preferences:**
   Echolalia uses a *local words database*, managed through the "LocalDB.cs" API in the Data folder. This local database allows efficient storage and retrieval of user-added words and learning statistics, contributing to a seamless and offline-capable learning experience.

   Additionally, the app employs a separate local database, "PreferencesDB" in the Data folder, using Xamarin.Essentials library for easy storing data. This file handles *preferences*, includes: user name, interface language, count of words per task.

4. **Used `System.Linq` library fro filtering Data from local DB.**

5. **Standard Settings in App and AppShell:**
   The application's initial settings, including colours, button configurations, and other global settings, are centralised in the App and AppShell classes. This centralisation simplifies app-wide customisations and ensures consistent design elements throughout the user interface.

6. **Dictionary Implementation**:

   - **Refreshing view**
     The DictionaryViewModel includes a RefreshCmdAsync command, which triggers the refreshing of the view. When users initiate a refresh action, this command is executed, and the ViewModel updates the view's content to reflect any changes made to the dictionary.

   - **CollectionView**
     To display the list of translations in the dictionary view, the *DictionaryViewModel* uses a *CollectionView*. The CollectionView is a flexible and powerful control in Xamarin.Forms that provides a scrolling list of items, supporting features like grouping, selection, and custom layouts.

     - **`ObservableCollection<Word>`**
       The ViewModel utilises the `ObservableCollection<Word>` to store and manage the translations. An ObservableCollection is a generic collection class in C# that automatically notifies the View whenever items are added, removed, or when the collection is refreshed. This real-time notification ensures that the dictionary view stays up-to-date with the latest changes in the data.

7. **Using `Microcharts` Library for dynamically shows beautiful “Donut” diagram**

   - **`GetStatsAsync` Method**
     Asynchronous method which calculates the number of *learned, in-process, and not-learned words* in the user's dictionary. It retrieves the word items from the local database (App.localDB) and performs *LINQ queries* to count the words falling into different learning progress categories.

4. Possible improvements

1. **External Database** allowing users to download word packages based on specific topics or themes. This enhancement will provide users with a broader range of content and foster a more comprehensive language learning experience.

2. **Automatic Suggestions**: To optimize the addition of words, it would be possible to implement automatic suggestions when users add new words. This feature will offer appropriate translations and context, simplifying the learning process. Add automatic suggestions when adding new words.

3. **Gamification and Rewards**: Users will earn points or rewards for completing exercises, achieving milestones, and staying consistent in their learning journey, fostering a sense of accomplishment and motivation.

4. **Personal Account for Synchronisation**: With this feature, users can create individual accounts that enable synchronisation across multiple devices. This ensures that their learning progress, custom words, and exercise history are accessible from any device, providing a seamless and personalised learning journey.

5. Conclusion

In conclusion, Echolalia is a powerful and user-friendly language learning application designed to practicing and learning words. Based on the principle of repeated repetition and training, Echolalia offers an effective and engaging way to improve vocabulary and language comprehension. Users can engage in various exercises, such as Repeating Words, Learning Words, and Training Words, to practice their language skills actively.

The Echolalia architecture, built using the Xamarin framework and the C# programming language, provides cross-platform compatibility and also provides scalability. Offline functionality ensures continuous learning even when there is no internet connection.

Happy Learning with Echolalia!

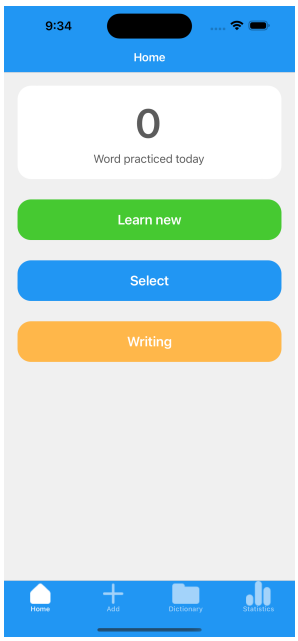
6. References to the materials used

Basic MVVM Base class / INotifyPropertyChanged implementation
- "https://onewindowsdev.com/2017/07/21/basic-mvvm-base-class-inotifypropertychanged-implementation/"

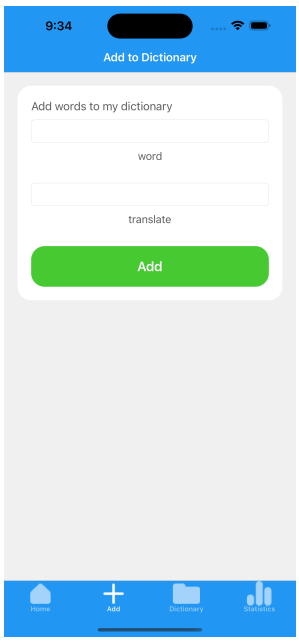
Xamarin documentation
- “https://learn.microsoft.com/en-us/xamarin/"

YouTube channels | playlists about Xamarin
- "https://www.youtube.com/@Mikaelson08"
- “https://www.youtube.com/@JamesMontemagno"
- "https://www.youtube.com/@XamarinDevelopers"

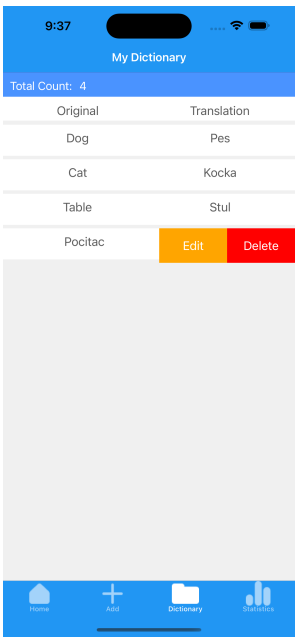
7. Screens



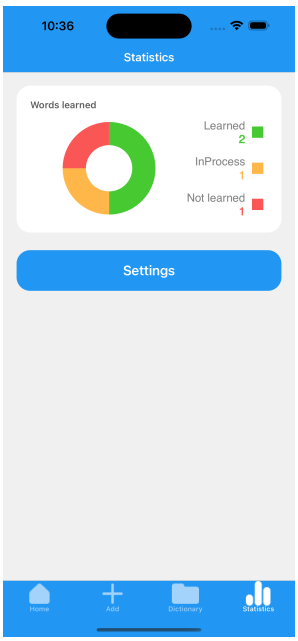
Home



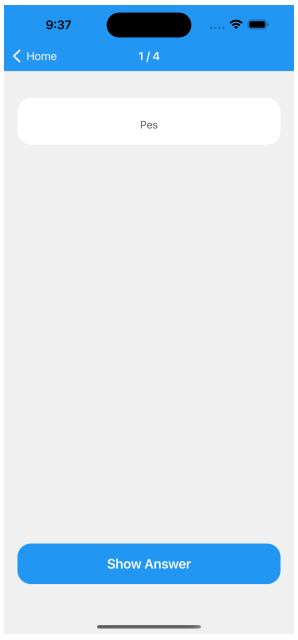
Add word to dict



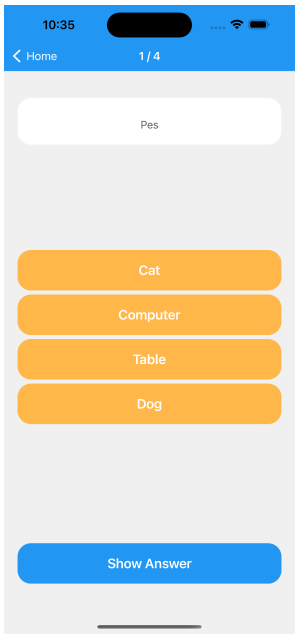
Dictionary



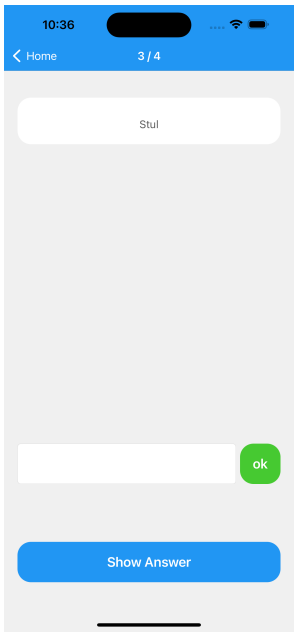
Statistic



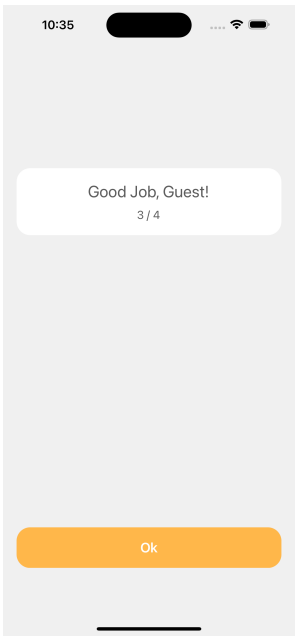
Learning new words



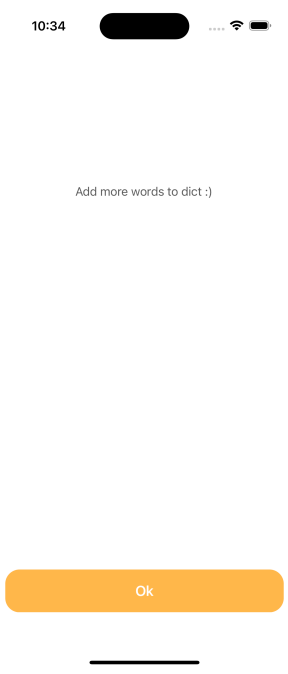
Select right answer



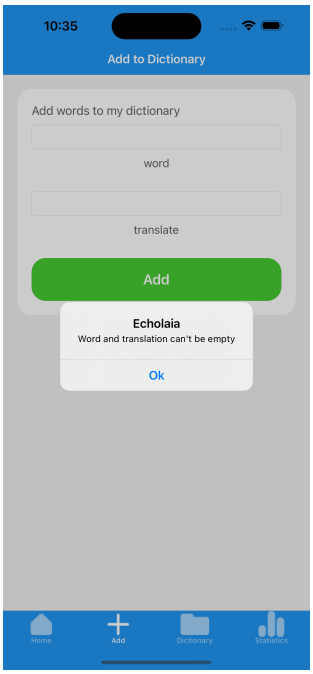
Write right answer



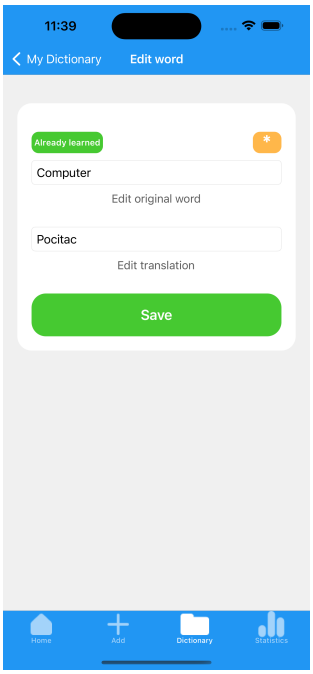
Result



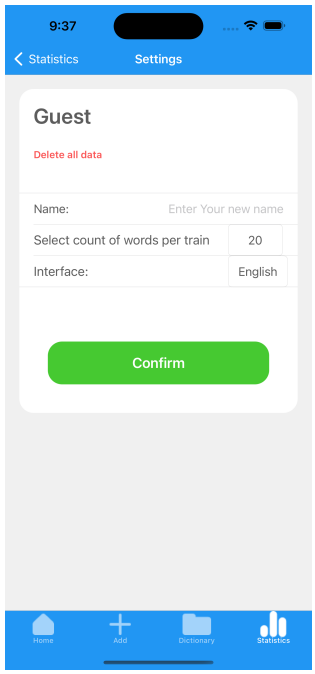
Warning if user haven't any word



Warning if user adds incorrect words



Editing word



Settings