FARSI TO FINGLISH

CSC 492 Senior Project
Technical Specifications Document
Aryan Zaferani-Nobari

Table of Contents

BACKGROUND

As of right now there is a lack of a mobile application tool which allows users to translate Farsi based languages into English based characters. Currently users have to rely on web based platforms to translate Farsi based idioms to their literal English meanings, this often times is problematic as many words in the Farsi based idioms lack the literal English translations. The accuracy of these translators often times cannot be trusted due to the many discrepancies in the syntax's of both idioms. Often times many people who are trying to translate Farsi idioms into English are able to speak and understand the Farsi language however they are unable to read and write the language because of their unfamiliarity with the Farsi based letters. In order to solve this current barrier in the two languages I propose to create an iOS application for mobile devices which allows users of the application to write or copy Farsi sentences/words/paragraphs into the application, the application will then translate the sentence/word/paragraph into its Finglish version.

1. SPECIFICATIONS

1.1 <u>Users</u>

This system is tailored to an audience that is a native speaker of both the Farsi idiom and English idiom, but are unable to read or write using Farsi characters. This platform is designed to allow people like me to communicate with their friends and family that are not able to write using English letters. There is also an Admin mode that is designed for an admin user to finalize certain translations.

1.2 <u>Initial Goals</u>

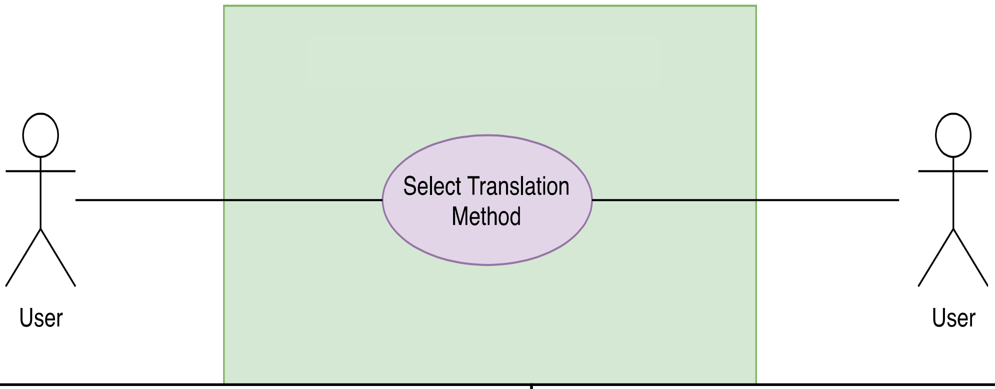
Initially I wanted to just create a translator for converting Farsi letters into Finglish, but I decided I could create a tool that could be much more useful with a few modifications. I decided I would create a system that also allowed users to translate Finglish to Farsi. I also added a game aspect that allowed users to test their abilities and how to spell new words.

1.3 Stretch Goals

In the future I would like to be able to add an aspect to the application that allows users to take pictures of signs, and other text and have the application recognize the characters and convert the spelling of the words.

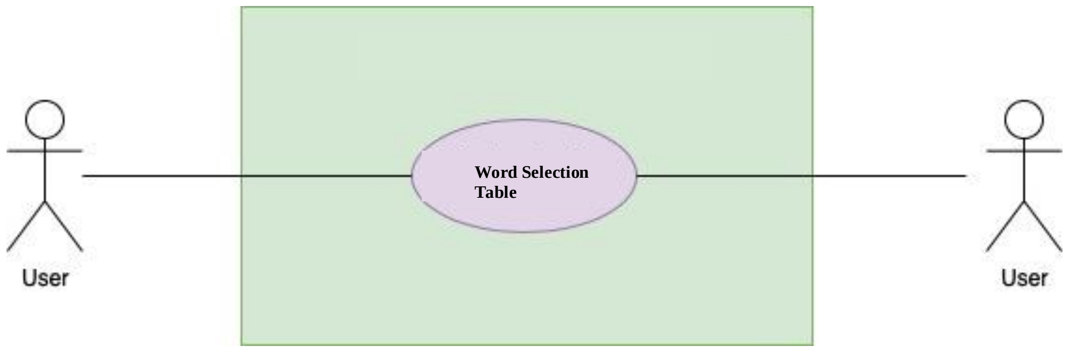
1.4 Use Cases

1.4.1 Application Mode Selections



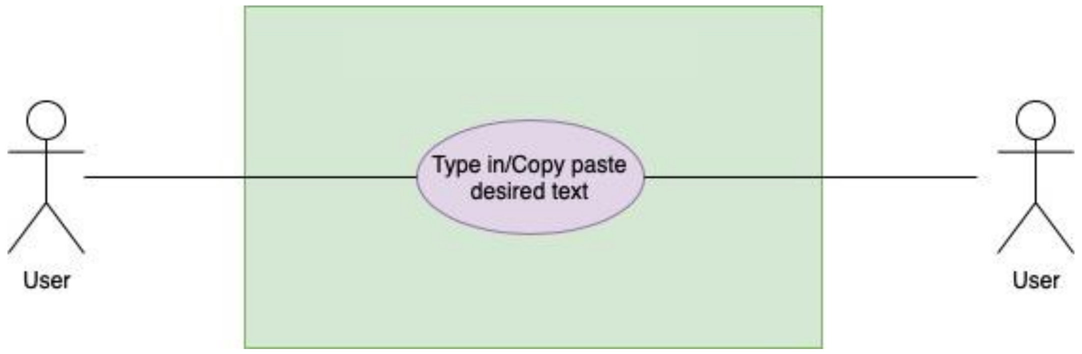
Users	**Steps**
• Application User	• Open application on iOS device • Wait for application to load • Select Farsi to Finglish option or Finglish to Farsi option, or game, or admin option.

1.4.2 Admin Confirmation



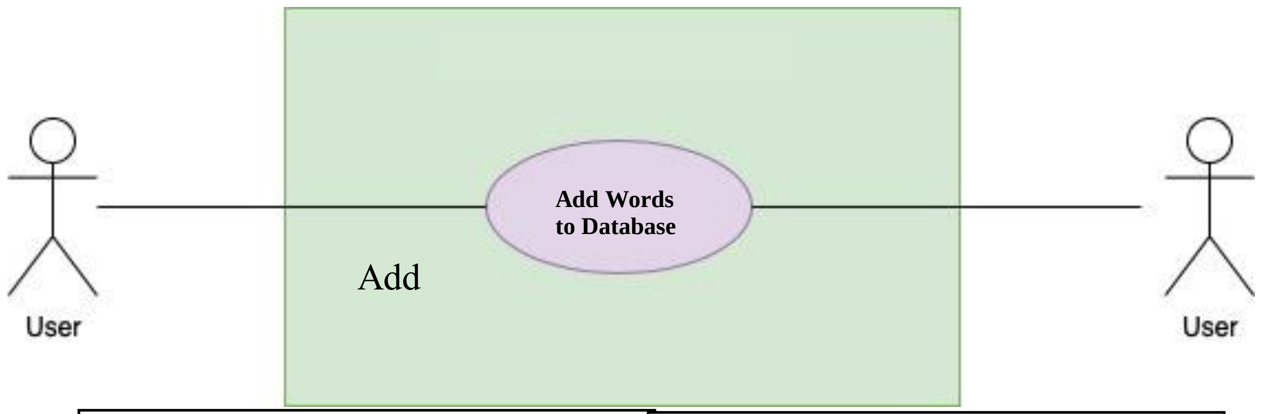
Users	**Steps**
• Application User	• Select Admin button in main menu. • Enter Admin password. • Select desired word. • Confirm spelling for desired word or edit the translation. • Hit the enter button on the screen.

1.4.3 Farsi to Finglish Input



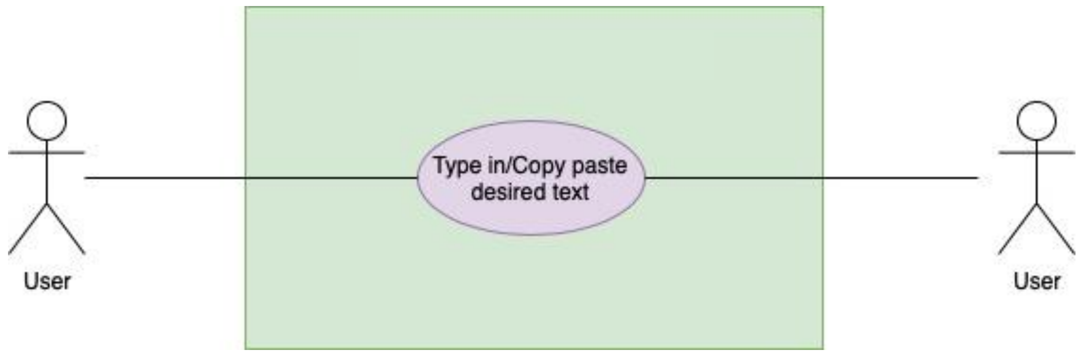
Users	**Steps**
• Application User	• Navigate to dialogue box then select it. • Type desired text via keyboard or paste text into dialogue box. • Hit translate button.

1.4.4 Add New Words



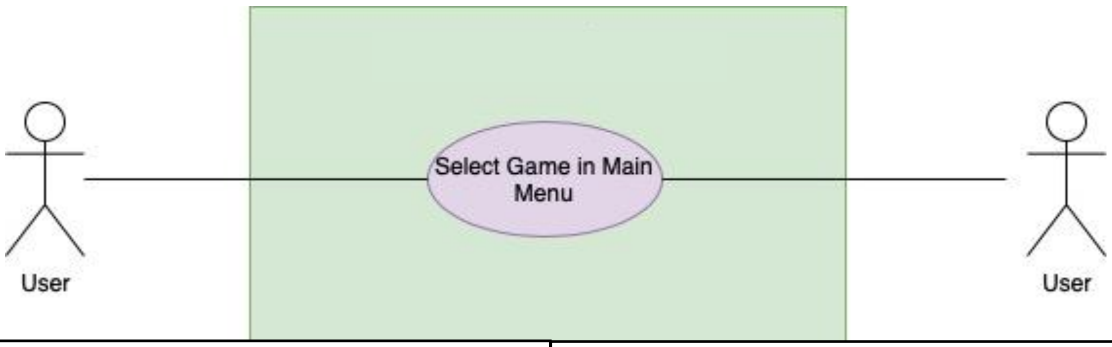
Users	• **Steps**
• Application User	• If desired click add new words button in order to add translation mappings to the database. • In the table view select desired word. • Type in the desired translation. Then Submit

1.4.5 Finglish to Farsi Input



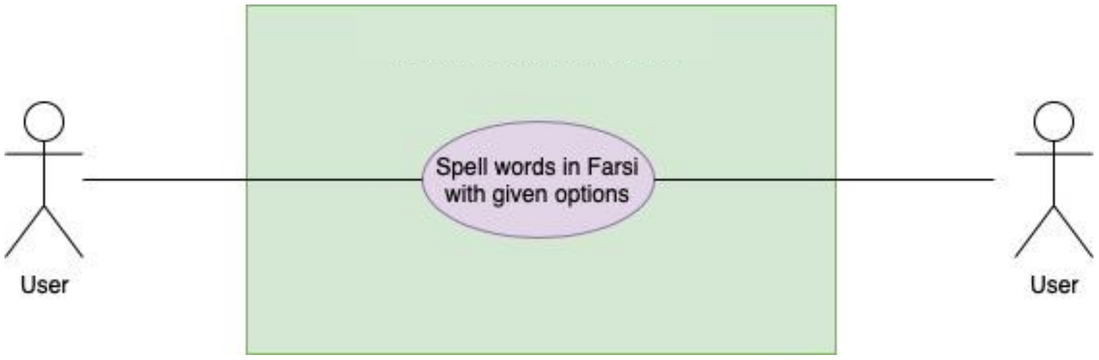
Users	**Steps**
• Application User	• Navigate to dialogue box then select it. • Type desired text via keyboard or paste text into dialogue box. • Hit translate button.

1.4.6 Enter Game mode



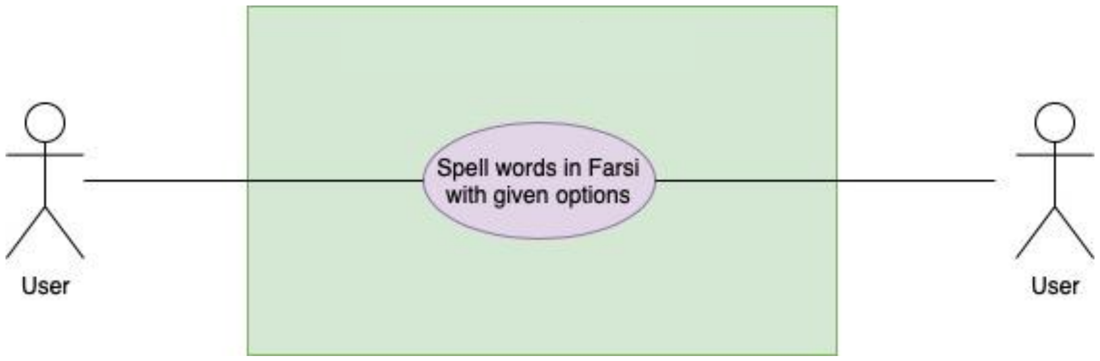
Users	**Steps**
• Application User	• Navigate to Main Menu of the application. • Select Game button in the main menu. • Select difficulty of level based on a scale of 1-5.

1.4.7 Game Level Easy



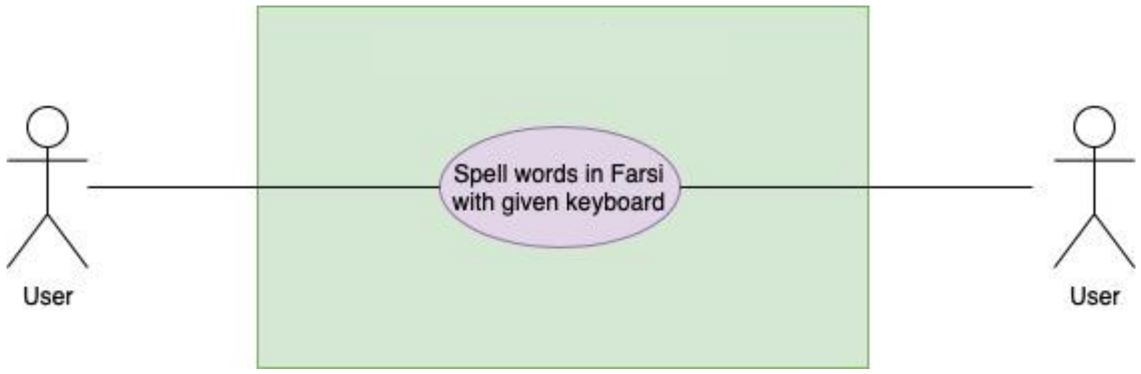
Users	**Steps**
• Application User	1. Match the word written on top of the screen with one of the given options. 2. Press the correct option, do this for 10 rounds.

1.4.8 Game Level Medium



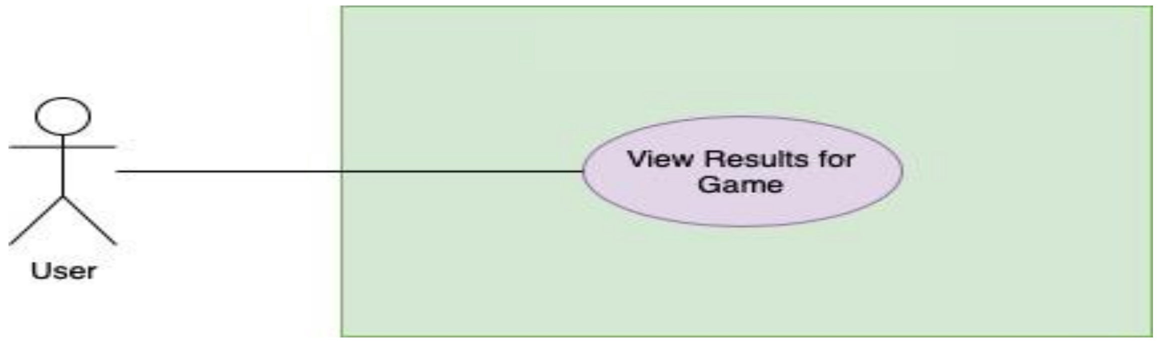
Users	**Steps**
• Application User	• Parse word letter by letter. • Match the letter written on top of the screen with one of the given options. • Press the correct option, do this till the word is spelled correctly

1.4.9 Game Level Hard



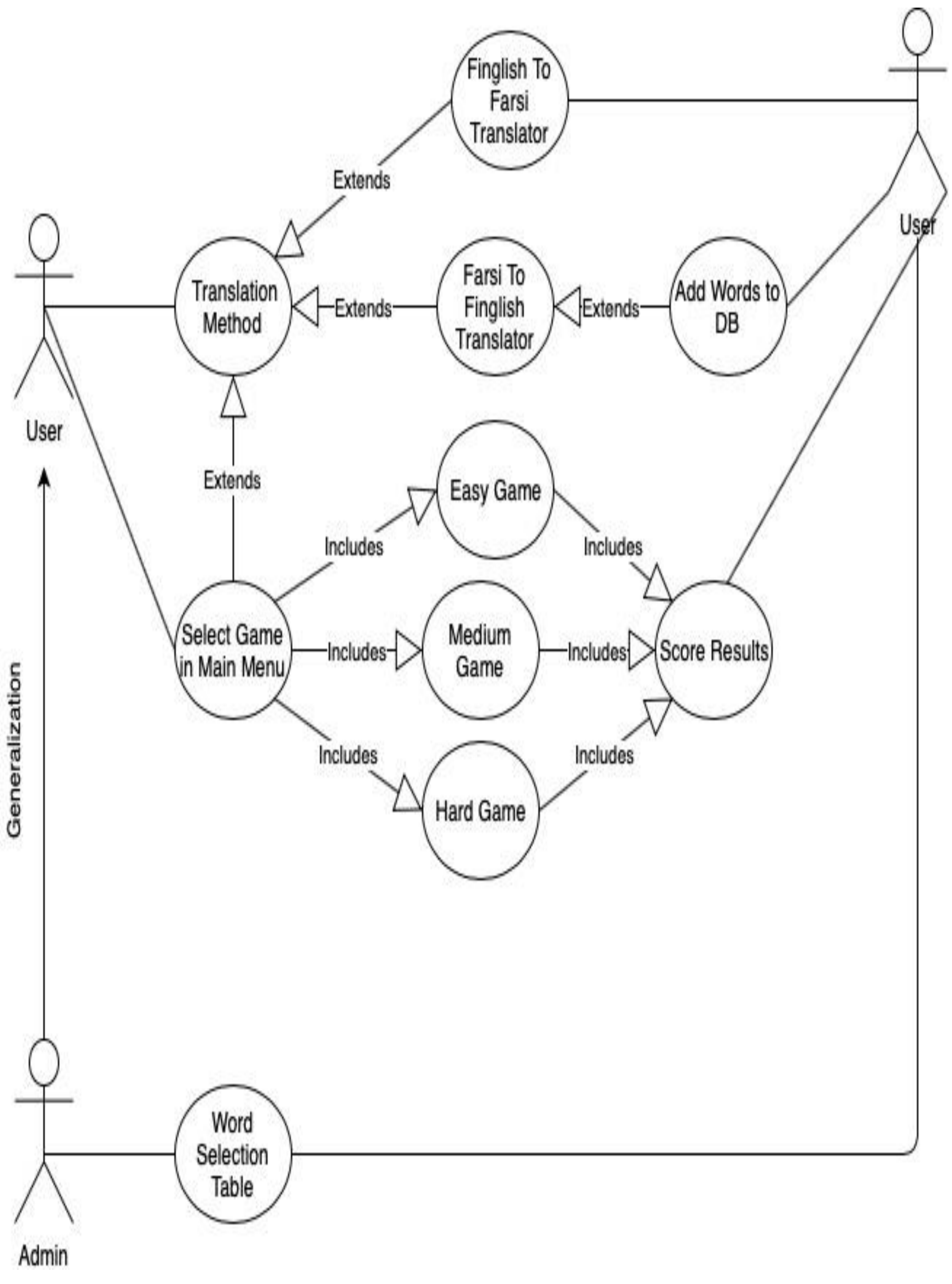
Users	**Steps**
• Application User	• Parse the given word letter by letter • Spell the word in Farsi using keyboard. • Do same for next word. • Repeat for 10 rounds.

1.4.10 Game Results



Users	**Steps**
• Application User	• Once the game session has ended a score is shown on the screen. • View score for your game session • Hit exit button to go back to applications main menu.

1.4.11 Use Case Diagram



1.5 Functional Requirements

- The User should be able to select whether they want to translate Farsi to Finglish or Finglish to Farsi upon the app’s startup.
- The User should be able to enter text via keyboard input or via copy and pasting the desired text into the dialogue box.
- Once the User presses the translate button there should be an almost instantaneous translation in the output dialogue box.
- There will be a functionality incorporated into the system in which all punctuation is removed and the font is translated into a default.
- The User should be able to copy the outputted text into their iOS’s clipboard.
- Users should be able to add to the database of words.
- The user will have three game modes designed to improve the users Farsi reading and writing skills.
    - The first game will show the user a word in Finglish, the user then has multiple options to select the correct word in Farsi.
    - The second game will be a little harder as it will show users a word and give the user a list of letters to choose from in order to spell out the word.
    - The third game will be harder in the sense that it will not give the users predefined letters to build the given word with it will only give a keyboard with all the possible alphabetical options.
    - The game will give a score for each round.

1.6 Non-Functional Requirements

- This application will be supported on iOS based devices
- The application shall be reliable in a sense that it will have less than 1 crash per 100 uses, the application will also have as accurate as a translation as possible for the user.
- The application should be able to support many users simultaneously, because the database access of each user should be independent of other users.
- The response time throughout the application should be under a second.
- The flow and user interface of the application shall be intuitive and easy to follow. Due to the application being meant to be distributed to a variety of age groups, all users should be able to use the application without much guidance.

1.7 Keywords

Finglish: Persian(Farsi) sentences typed in English. Mostly used in texting.

1.8 Constraints

The main constraint of this system is due to the grammar syntax of the Farsi language. When writing Farsi vowels are not written the user memorizes the word and pronounces the word accordingly. Because vowels are not written they are not translated into Finglish in the application. If we wanted to get the exact translation, we would have to have a database of every single word in the language mapped to its Finglish counterpart. We have established a database to store each word however the user must be connected to the internet when using this feature. If there is no established internet connection the application will use the most recent database, it has downloaded previously.

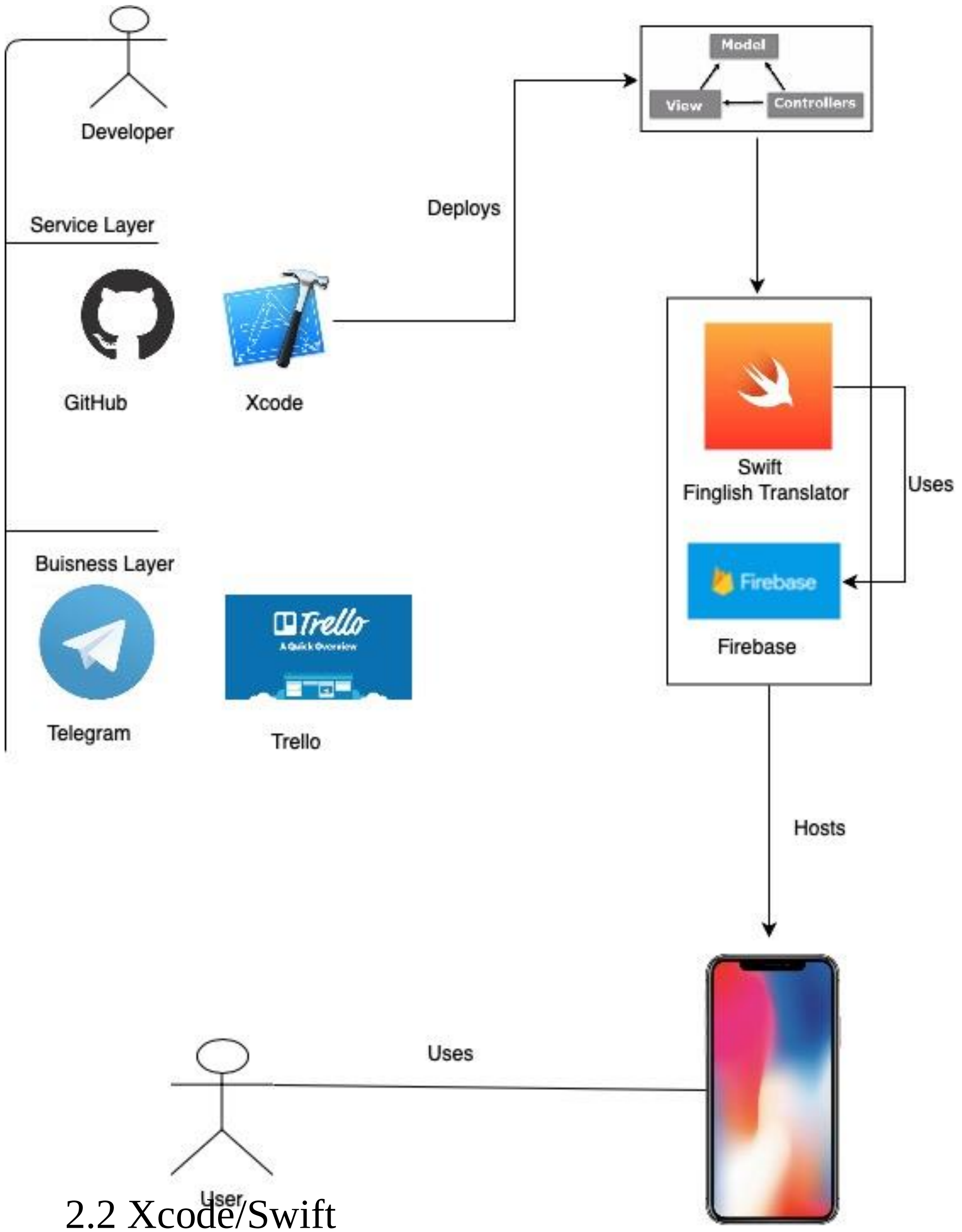
An example of the Farsi language constraint can be seen here, سلام this word literally spells out SLM because the vowels are not written. The correct pronunciation would be salam, which means hello. Hence the two a’s are missing, and must be added when you are translating farsi to Finglish.

1.9 Acceptance Testing

Most of the acceptance testing will be done while I am in Iran this break. I will also be giving the app to my relatives in Iran for them to use and report any bugs they encounter while using the application. I will also be using my own basic knowledge of the Farsi idiom in order to do some acceptance testing of the application.

2 Software Architecture

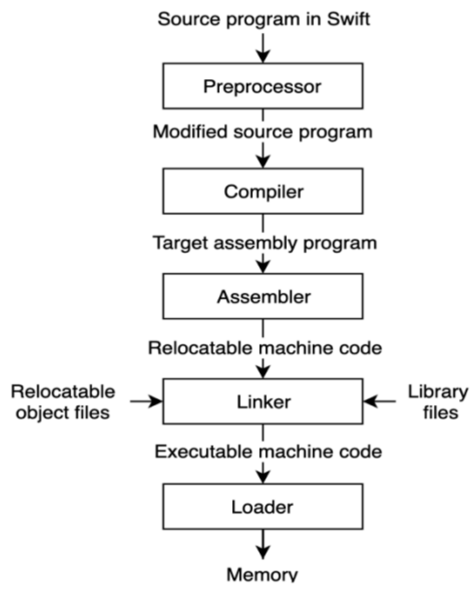
2.1 Technology chart



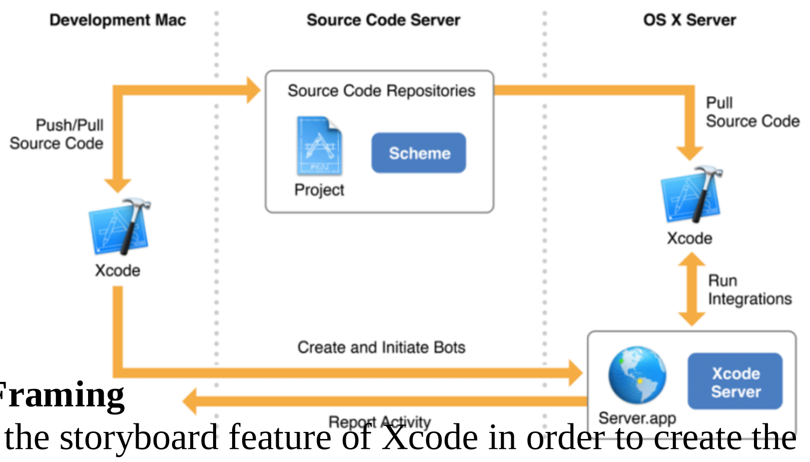
2.2 Xcode/Swift

Xcode had multiple uses in my application. I used it as an editor and a build tool, I also used many of its features for wireframing the components needed for my application.

Build
Xcode runs a number of tools and passes dozens of arguments between them, handles their order of execution. The majority of language processing systems consists of 5 parts; Processor, Compiler, Assembler, Linker, Loader.



Although I will not be using much continuous integration into my application Xcode allows for easy continuous integration through its Xcode Server.



WireFraming
I used the storyboard feature of Xcode in order to create the wireframes, of the application.

Model View Controller
Xcode uses something similar to the Model View Controller architecture however the major differences are the View aspect and the Controller aspect are formed into one entity known as the ViewController. In my application the ViewController classes contain components for getting data via user input. Once the data is obtained through the various view controllers, my data is then passed to Model classes. The Model classes allow for the data to be processed and stored into the Firebase database. The most important aspect of my Model classes is the Translate class which does the processing for the translations as well as the database look ups.

Swift
Swift is a language developed by Apple, it is based on C++ and is object oriented. My application is written of Swift which also incorporates a Firebase API which is used for communicating to the Cloud based database.

2.3 Database

There are two databases that this application uses one is cloud based while the other is internal. When there is a word that is not present in the Firebase database the user can use a functionality called **Add Words**. Add Words will allow the user to enter the new word and its translation into the Pending database in Firebase. The Admin can then opt to move that word into the Confirmed section in Firebase if he or she believes it’s an accurate translation.

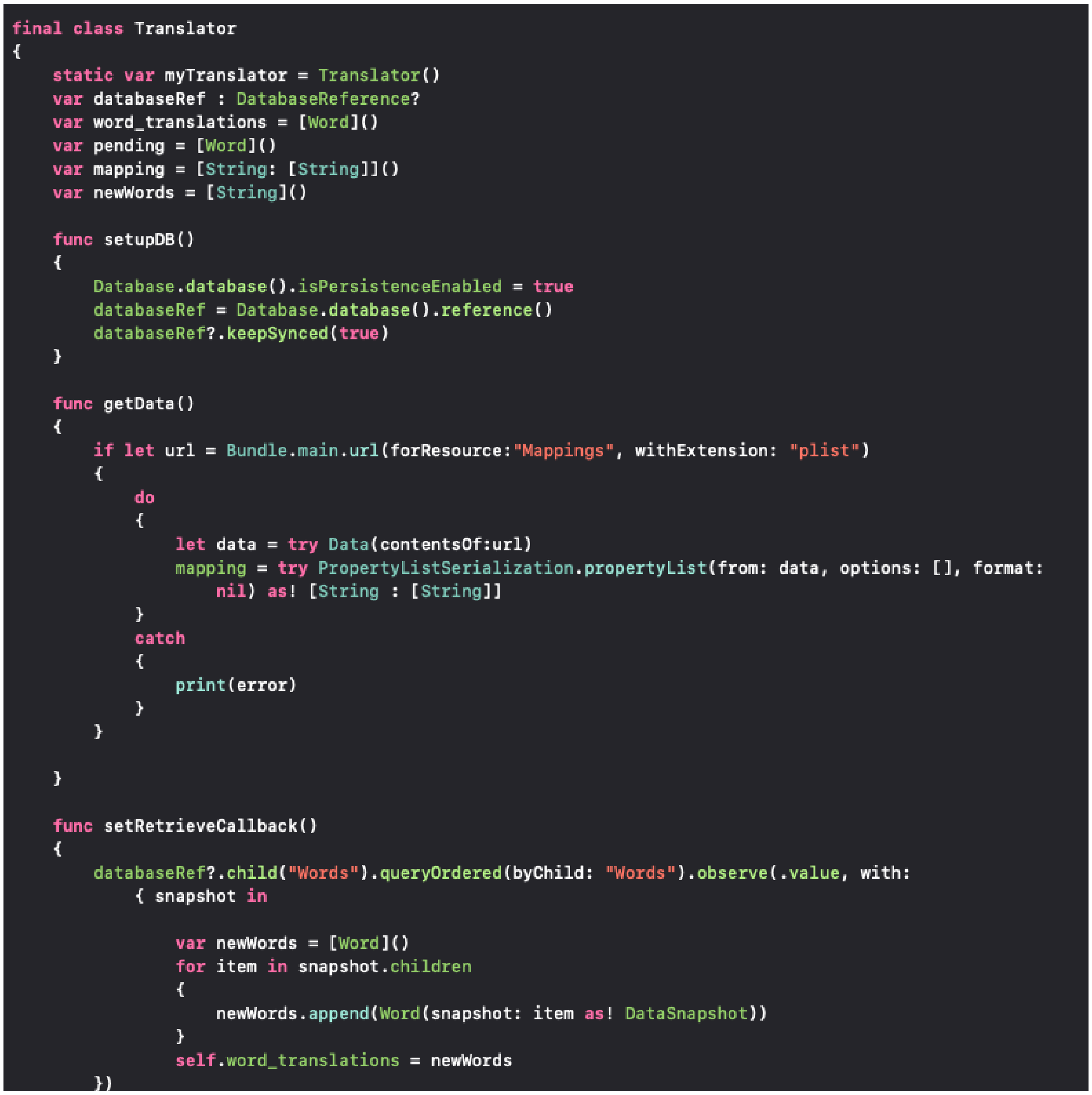
Firebase
The Firebase databases consists of two sections a pending section and a confirmed section, both the sections are structured the same. Each consists of a list which consists of Word objects. Each Word object has two strings one storing a Finglish word one storing a Farsi word.

Local Mapping

The local mapping is stored in the application itself. The local mapping is a dictionary that maps each letter of the Farsi language to its equivalent in the English language.

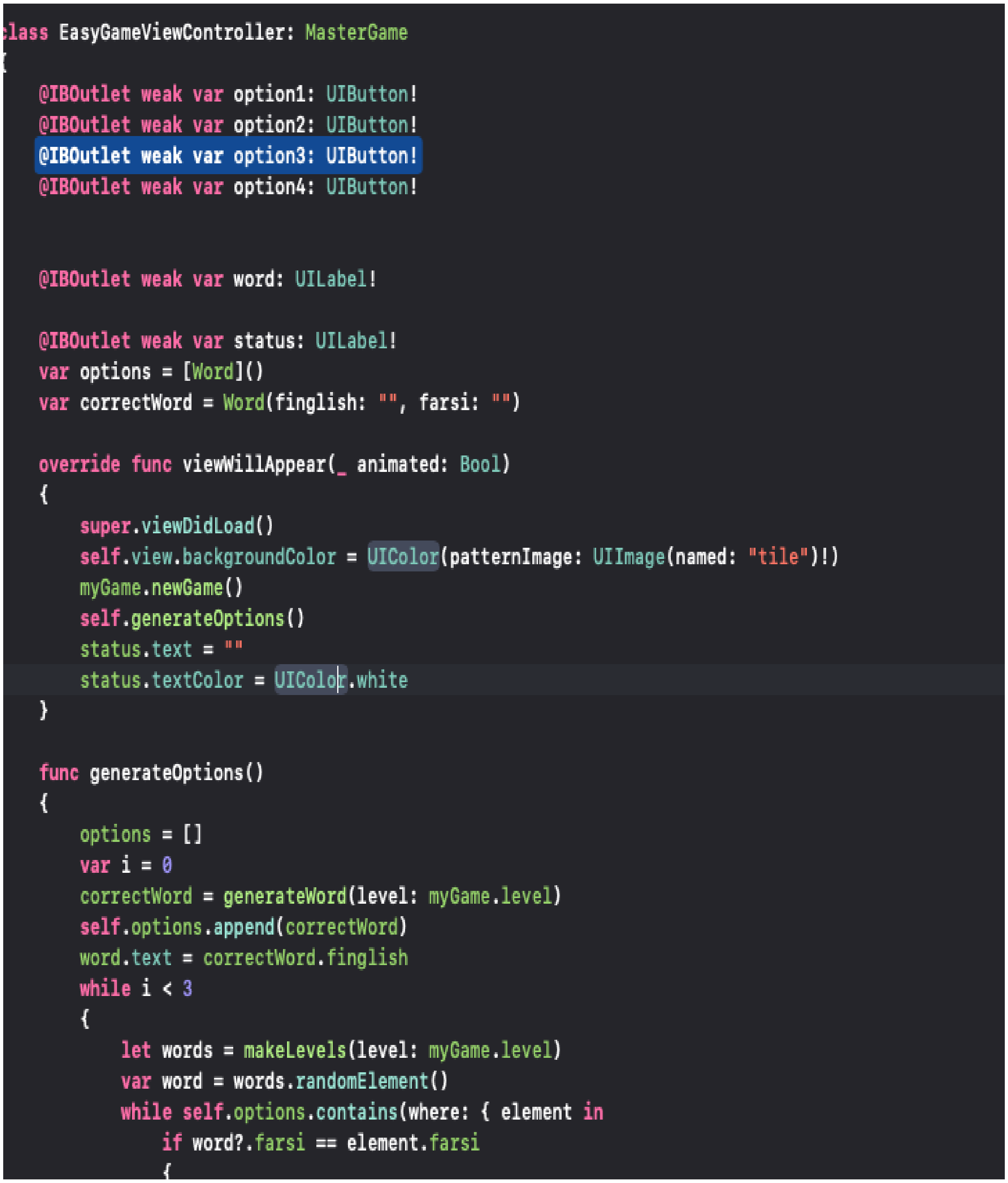
2.4 Design Patterns

Singleton: A design pattern that restricts the instantiation of a class to one object. I used a singleton in my translation class which mainly does database lookups

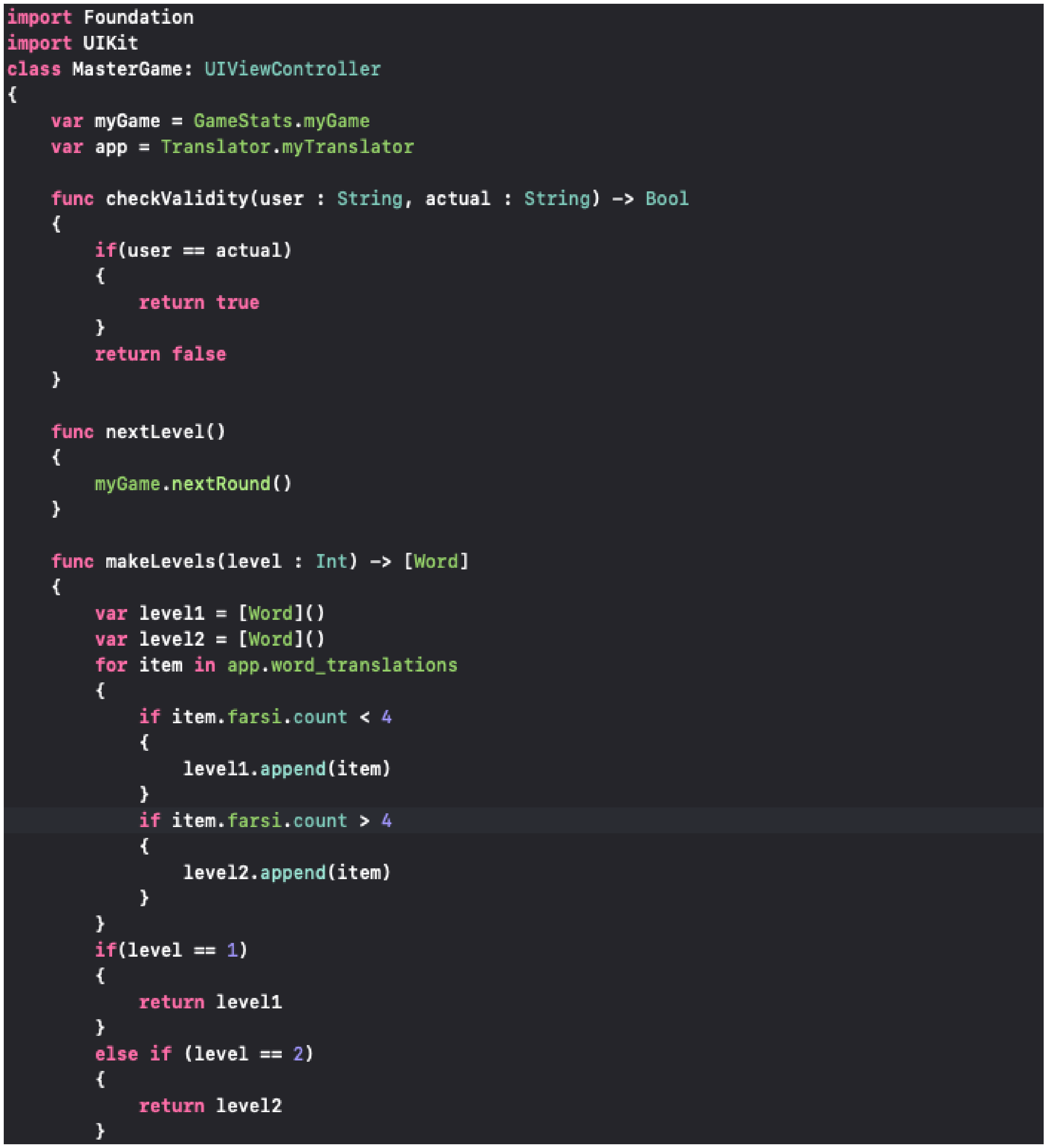


```
final class Translator
{
    static var myTranslator = Translator()
    var databaseRef : DatabaseReference?
    var word_translations = [Word]()
    var pending = [Word]()
    var mapping = [String: [String]]()
    var newWords = [String]()

    func setupDB()
    {
        Database.database().isPersistenceEnabled = true
        databaseRef = Database.database().reference()
        databaseRef?.keepSynced(true)
    }

    func getData()
    {
        if let url = Bundle.main.url(forResource:"Mappings", withExtension: "plist")
        {
            do
            {
                let data = try Data(contentsOf:url)
                mapping = try PropertyListSerialization.propertyList(from: data, options: [], format:
                    nil) as! [String : [String]]
            }
            catch
            {
                print(error)
            }
        }

    }

    func setRetrieveCallback()
    {
        databaseRef?.child("Words").queryOrdered(byChild: "Words").observe(.value, with:
            { snapshot in

                var newWords = [Word]()
                for item in snapshot.children
                {
                    newWords.append(Word(snapshot: item as! DataSnapshot))
                }
                self.word_translations = newWords
        })
```

Factory Design Pattern: The Factory design pattern creates an object without exposing the logic to the client and refer to a newly created object using a common interface. The use of this is mostly seen when I create a game object; each level of game is dependent on the functionalities of another class called MasterGame



```
lass EasyGameViewController: MasterGame

    @IBOutlet weak var option1: UIButton!
    @IBOutlet weak var option2: UIButton!
    @IBOutlet weak var option3: UIButton!
    @IBOutlet weak var option4: UIButton!


    @IBOutlet weak var word: UILabel!

    @IBOutlet weak var status: UILabel!
    var options = [Word]()
    var correctWord = Word(finglish: "", farsi: "")

    override func viewWillAppear(_ animated: Bool)
    {
        super.viewDidLoad()
        self.view.backgroundColor = UIColor(patternImage: UIImage(named: "tile")!)
        myGame.newGame()
        self.generateOptions()
        status.text = ""
        status.textColor = UIColor.white
    }

    func generateOptions()
    {
        options = []
        var i = 0
        correctWord = generateWord(level: myGame.level)
        self.options.append(correctWord)
        word.text = correctWord.finglish
        while i < 3
        {
            let words = makeLevels(level: myGame.level)
            var word = words.randomElement()
            while self.options.contains(where: { element in
                if word?.farsi == element.farsi
```

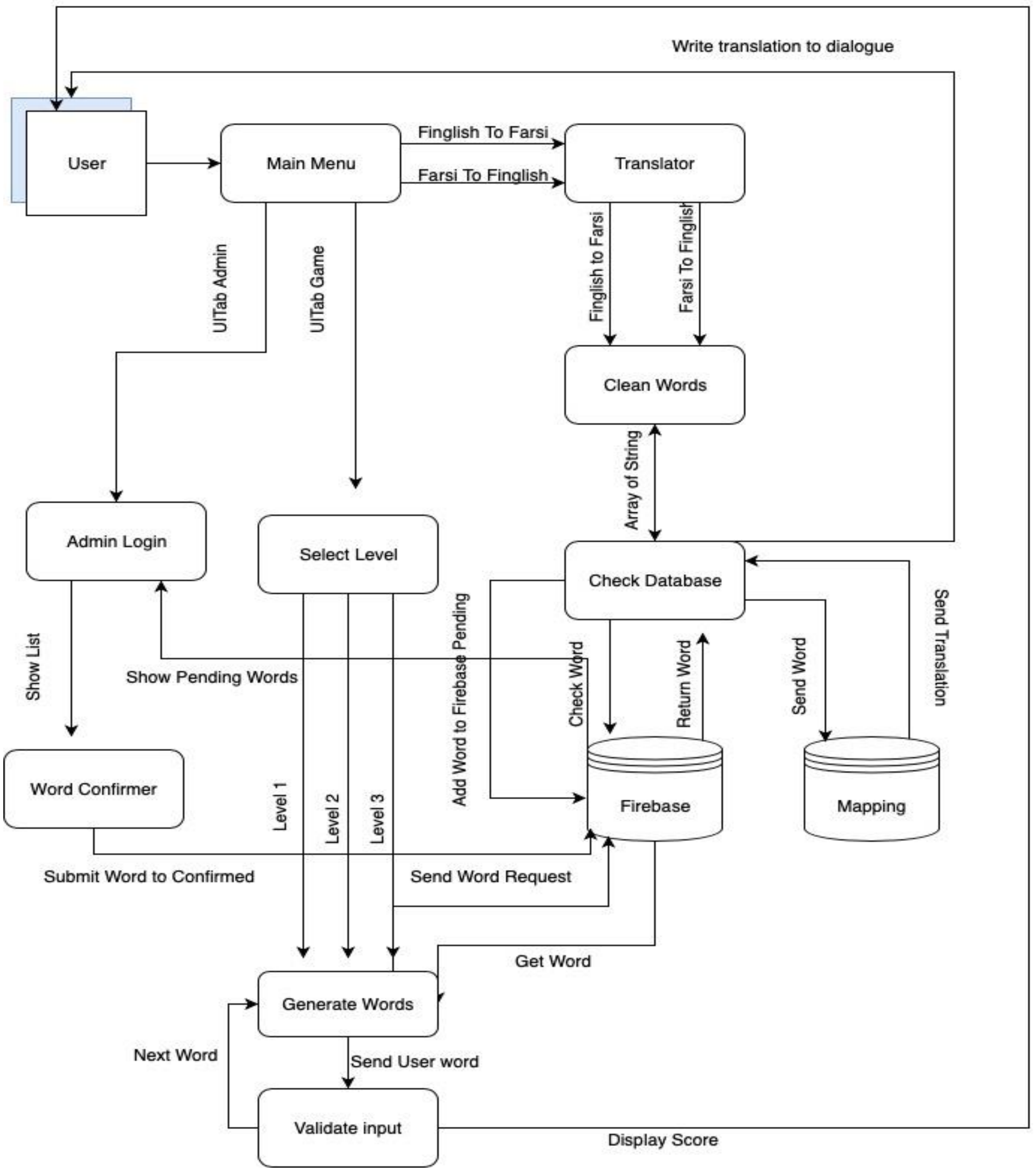


```
import Foundation
import UIKit
class MasterGame: UIViewController
{
    var myGame = GameStats.myGame
    var app = Translator.myTranslator

    func checkValidity(user : String, actual : String) -> Bool
    {
        if(user == actual)
        {
            return true
        }
        return false
    }

    func nextLevel()
    {
        myGame.nextRound()
    }

    func makeLevels(level : Int) -> [Word]
    {
        var level1 = [Word]()
        var level2 = [Word]()
        for item in app.word_translations
        {
            if item.farsi.count < 4
            {
                level1.append(item)
            }
            if item.farsi.count > 4
            {
                level2.append(item)
            }
        }
        if(level == 1)
        {
            return level1
        }
        else if (level == 2)
        {
            return level2
        }
```

Observer Pattern: A pattern in which used for one to many relationships where one object notifies one or more other objects that its state changed. This pattern is used numerous times internally in swift specially in the User Interface, object like buttons and scroll menus are observing for changes in states.

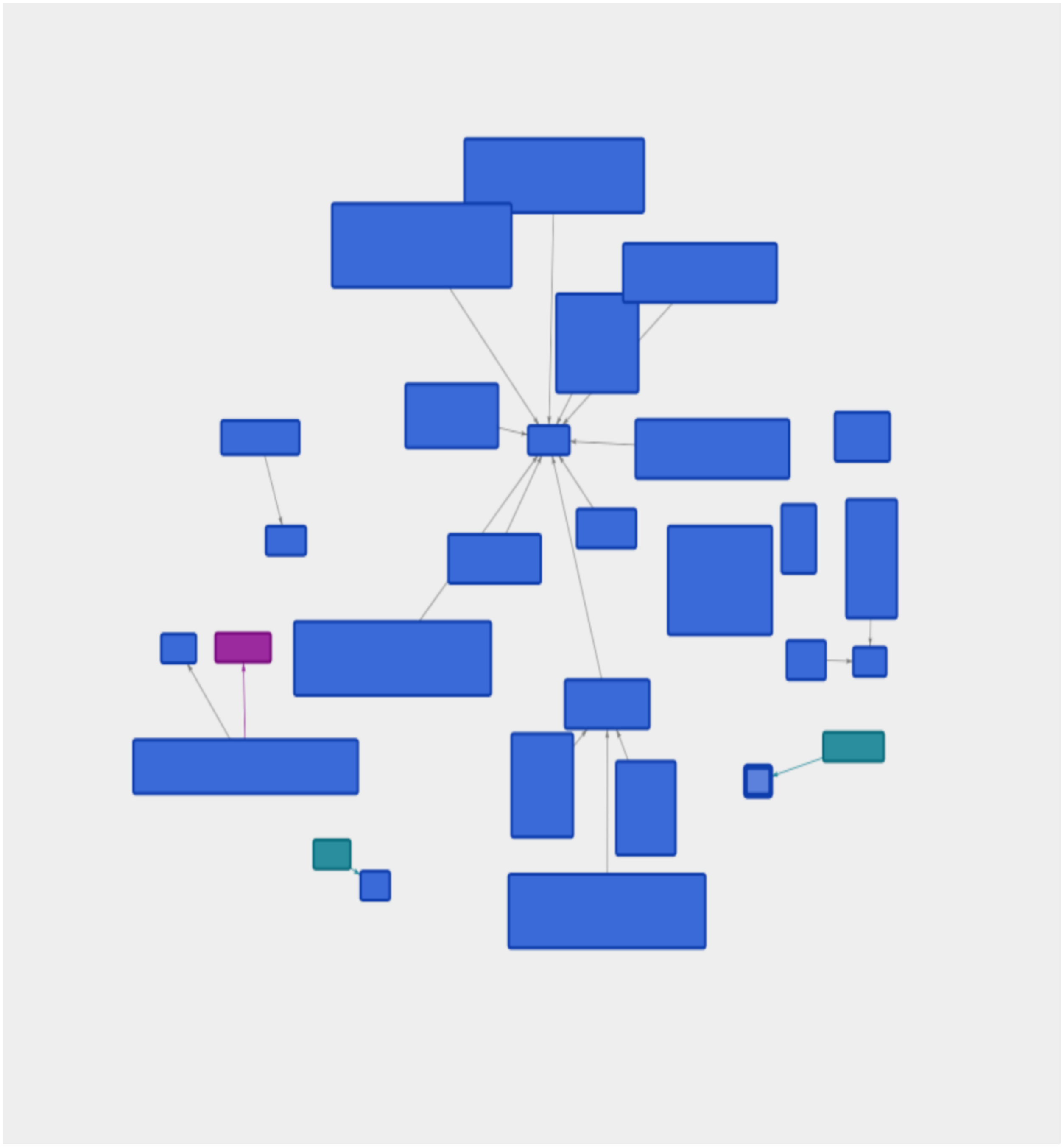
3 DESIGN DECISIONS

3.1 Data Flow Diagram



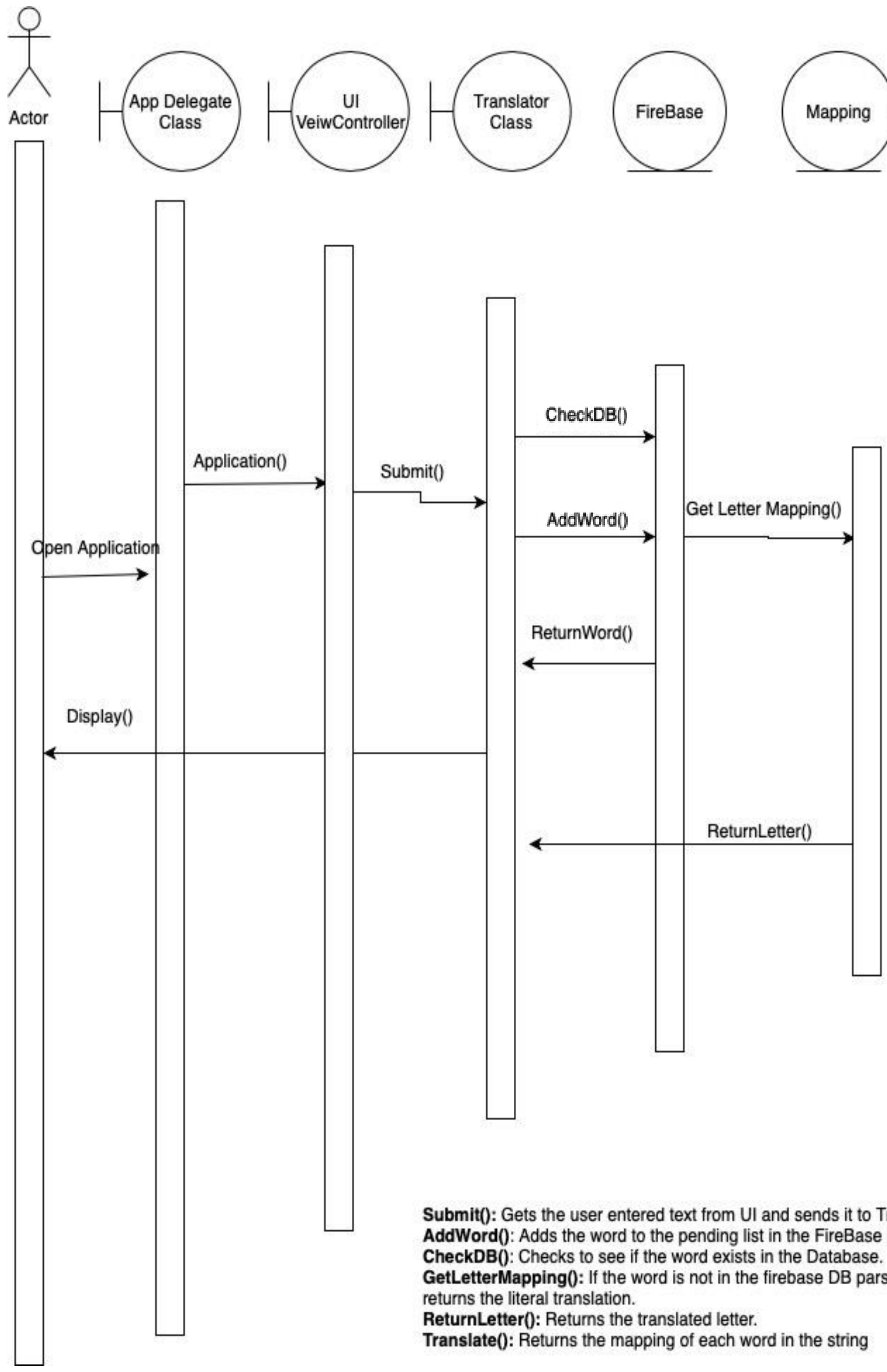
3.2 Class Diagram

- Main Diagram is linked as an HTML file.



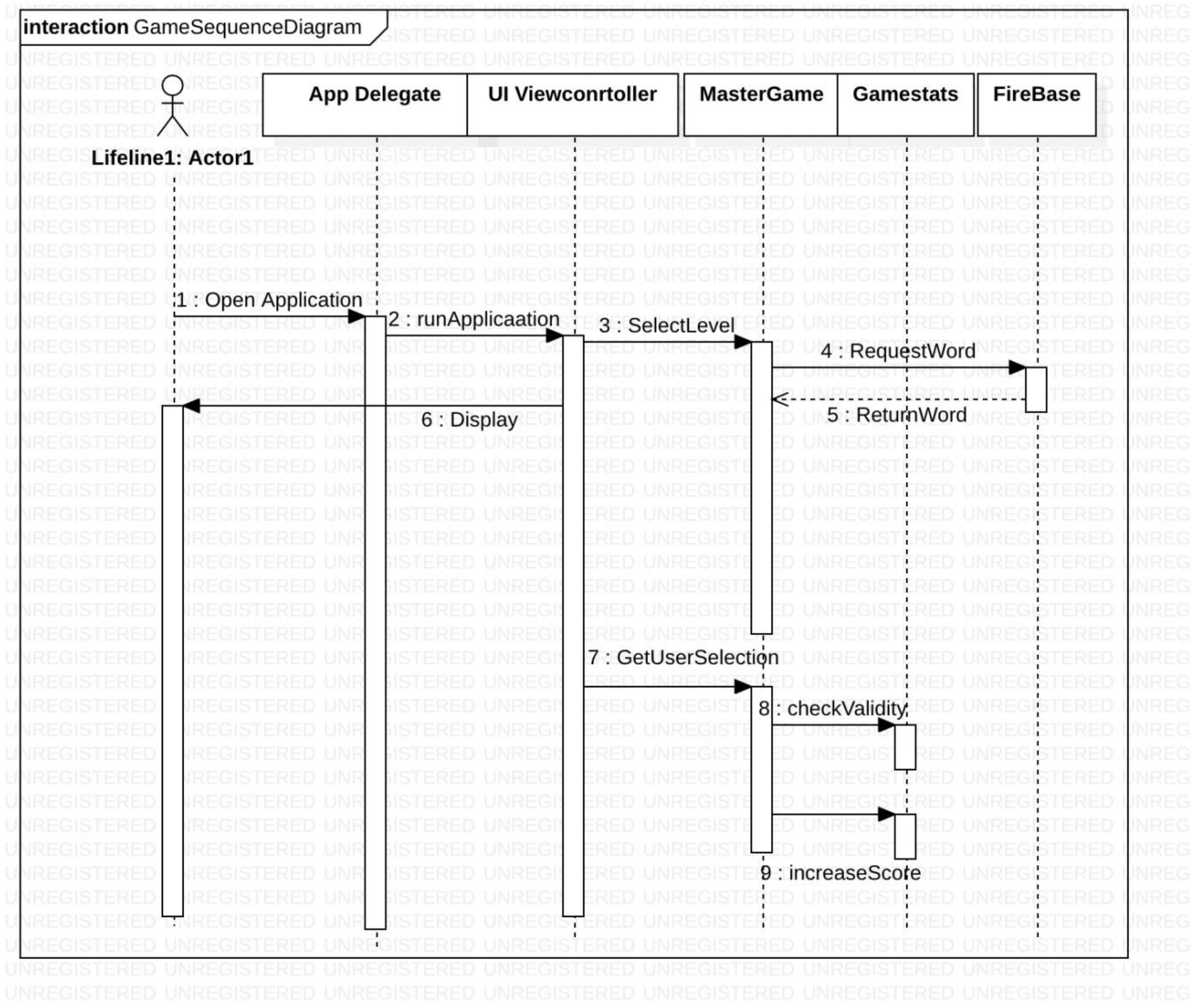
3.3) Sequence Diagrams

Translation Sequence Diagram



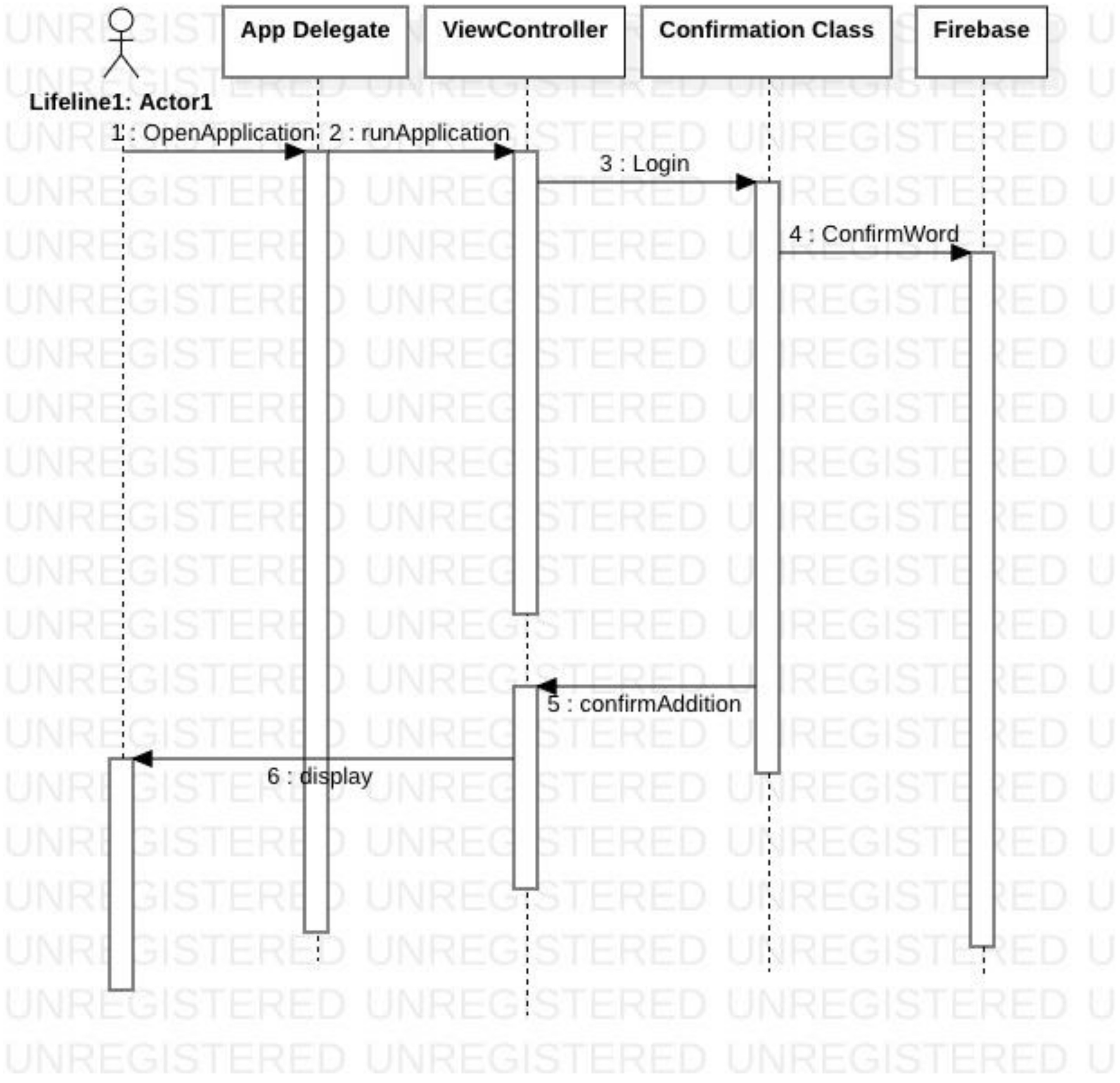
Submit(): Gets the user entered text from UI and sends it to Translator class.
AddWord(): Adds the word to the pending list in the FireBase Database
CheckDB(): Checks to see if the word exists in the Database.
GetLetterMapping(): If the word is not in the firebase DB parses the word and returns the literal translation.
ReturnLetter(): Returns the translated letter.
Translate(): Returns the mapping of each word in the string

Game Sequence Diagram



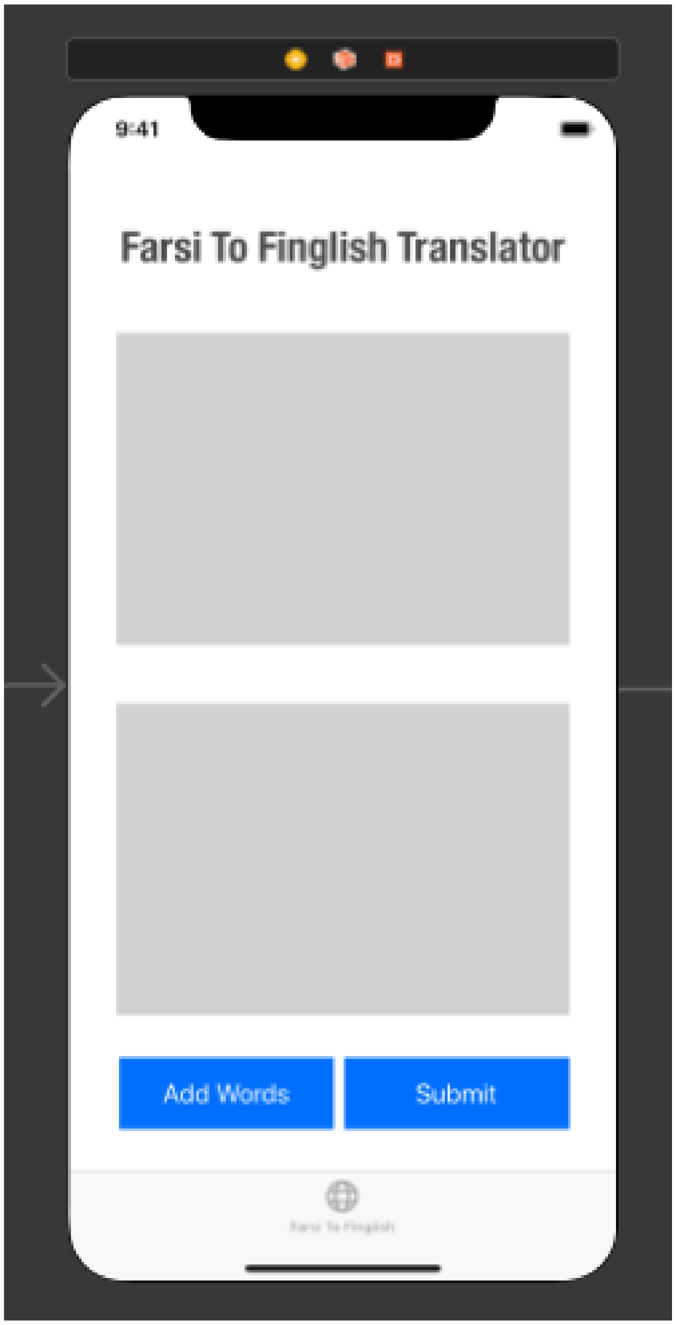
SelectLevel(): Allows the user to select a level.
IncreaseScore(): Increases the score of the game session
RequestWord(): Gets a word from Firebase according to the selected Level.
ReturnWord(): Returns the word from the database
CheckValidty(): Compares users answer with the actual answer
Translate(): Returns the mapping of each word in the string

Admin Sequence Diagram

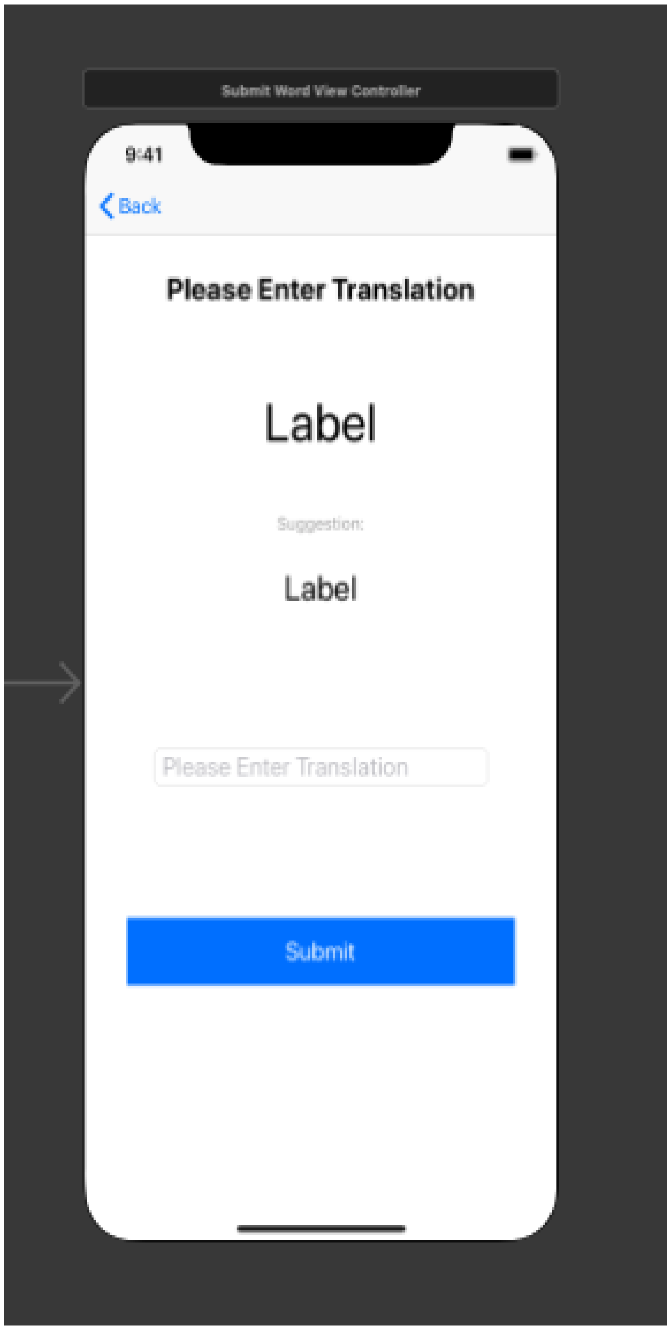


3.4) Wireframes

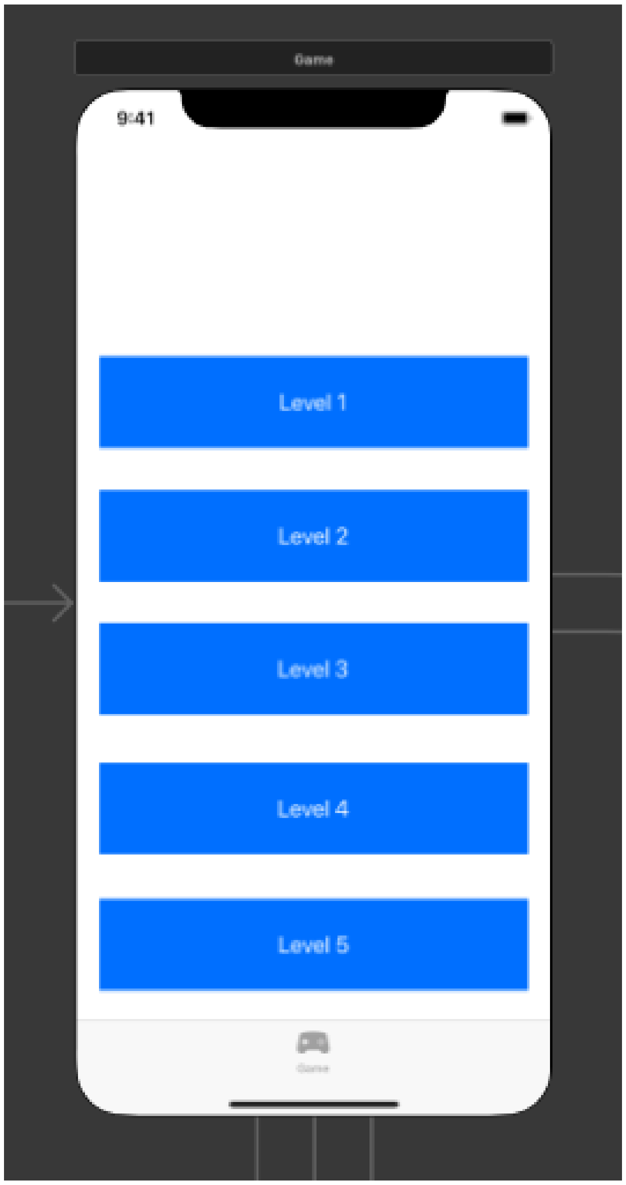
Translation Page



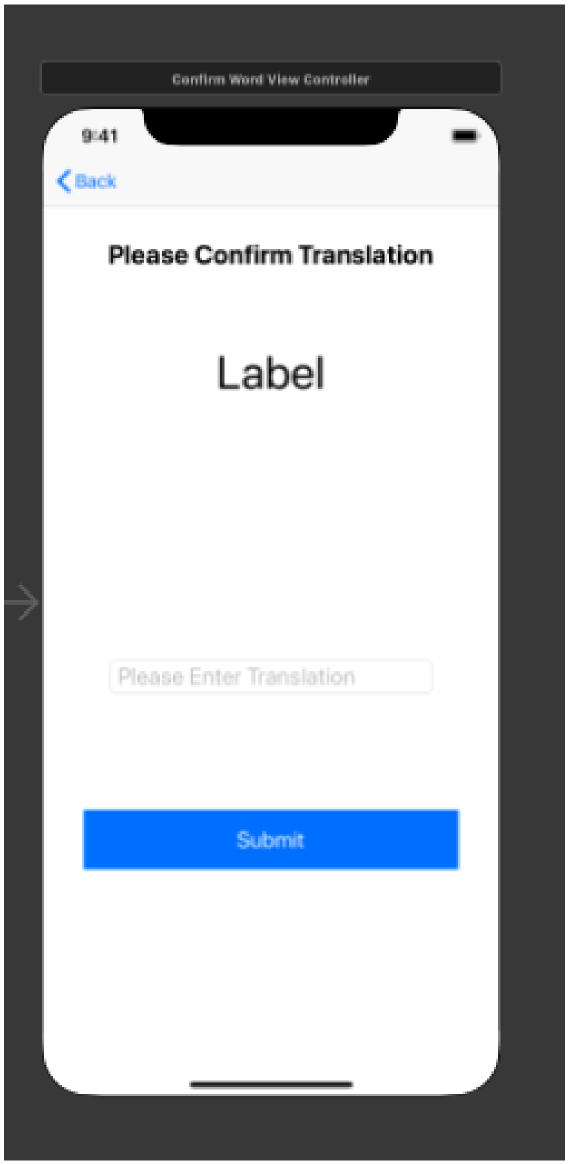
Add New Word



Game Level Selection



Admin Confirm Word



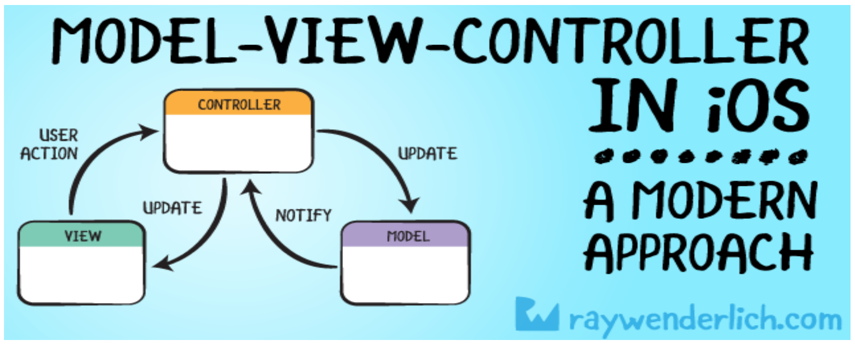
4.0 DESIGN

4.1) Design Patterns

For this project I decided to use the MVC design pattern which is a little bit different when it comes to the swift implementation. The Swift implementation has combined the View and the Controller into one aspect called the ViewController. I then have Model classes that are accessible by all the ViewControllers in the project. The MVC allows for a maintainable and easy to understand design.

For both the Finglish to Farsi and Farsi to Finglish aspects of the project I decided to use a singleton pattern. I create a singleton of the translator class, the singleton class has all the mappings for each letter as well as the required methods to get and set the mappings. Creating a singleton class allows the changes made in the Settings class to be consistent throughout the translation process. Every time one of the classes needs to conduct a translation they are able to do so with the most current translation instance.

For the game classes I will be using a Factory Design Pattern. Due to the fact that Game 1, Game 2 and Game 3 have very similar properties they will be sharing a lot of functionality. I have created a mother class called MasterGame that has all the common functionalities and properties of both the game versions. MasterGame also uses a singleton called GameStats that keeps track of the users progress in the game. Then I have three classes called game 1, game 2 and game 3 these classes inherit all the functionalities from the game class, but implement their own relative functionalities as well.



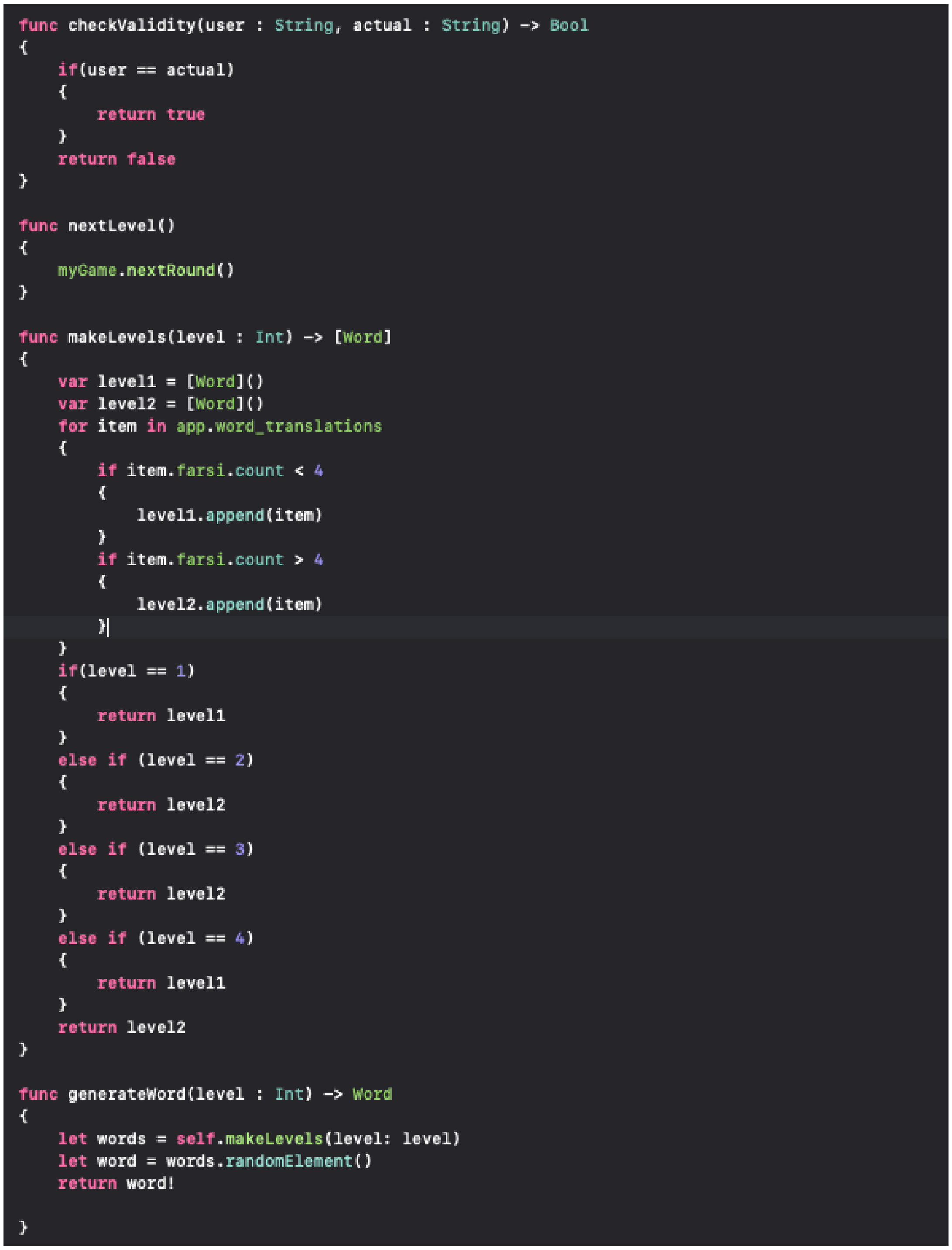
4.2) SOLID Design

S: Single Responsibility
O: Open/Closed Principle
L: Liskuv Substitution
I: Interface Segregation
D: Dependency Inversion

In order to make this software more maintainable, testable, and understandable I will be using the SOLID Design principle. I will be implementing the single responsibility principle in order to make sure my classes are very maintainable. All my classes will have one responsibility, and one responsibility only. This will allow for easier testing as well as allow my classes to be understandable by others in the future. All my classes will also follow the open closed principle, which means my classes will be designed in a way that they are very extendable, but there is no need to alter the main class in any way. This will be mostly apparent in my game class where there is a main Game class and the three sub game classes extend those classes. Liskov Substitution principles will not be as apparent as I will not have many subclasses in my project, so there will not be a need for the objects in the program to be replaceable with their subtypes. I will heavily be integrating Interface segregation as I will have many interfaces that are tailored for each specific task that needs to be completed by the program. I also do not believe I will be using much of dependency inversion because I will not have many low level modules based on higher level modules that should be based on abstractions. I will mostly have one layer that are all based on abstractions.

Uses of Solid in Application:

Single Responsibility
Each one of my classes has a single responsibility, which allows me to constantly reuse each aspect as well as make testing more affective. Below is a snippet of my MasterGame class which the functions all have single responsibilities.

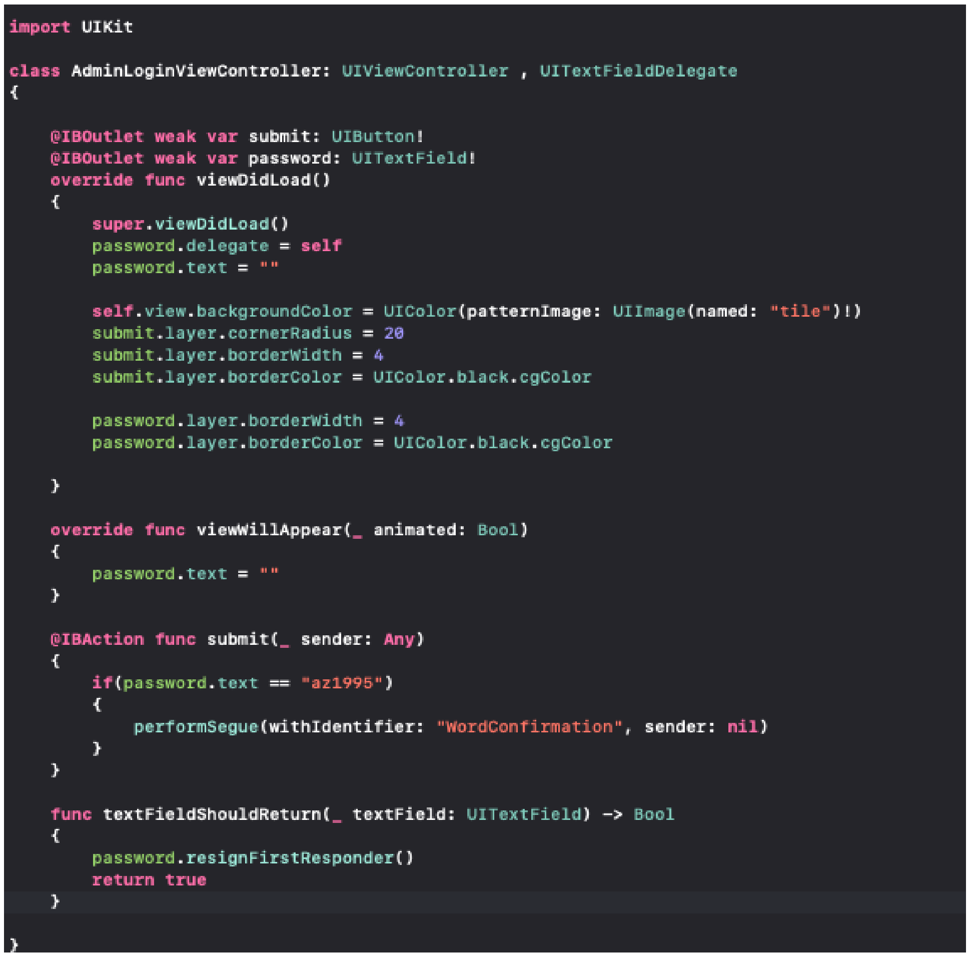


```
func checkValidity(user : String, actual : String) -> Bool
{
    if(user == actual)
    {
        return true
    }
    return false
}

func nextLevel()
{
    myGame.nextRound()
}

func makeLevels(level : Int) -> [Word]
{
    var level1 = [Word]()
    var level2 = [Word]()
    for item in app.word_translations
    {
        if item.farsi.count < 4
        {
            level1.append(item)
        }
        if item.farsi.count > 4
        {
            level2.append(item)
        }
    }
    if(level == 1)
    {
        return level1
    }
    else if (level == 2)
    {
        return level2
    }
    else if (level == 3)
    {
        return level2
    }
    else if (level == 4)
    {
        return level1
    }
    return level2
}

func generateWord(level : Int) -> Word
{
    let words = self.makeLevels(level: level)
    let word = words.randomElement()
    return word!

}
```

Open Closed Principle
Open closed principle is apparent in my use of the factory design pattern specially in my Game classes. My classes are able to be extended by other children classes however the children classes are not able to modify the parent class. An example of this can be seen above in the factory design pattern.

Liskov Substitution
Not much Liskov substitution was used in my programming due to the fact that I do not have many subclasses, so there is no need to replace an object with its subtype and have it work perfectly normal.

Interface Segregation
Interface Segregation allows the software to not implement interfaces that he or she does not intend to use. This is mostly done internally in swift in my application, I am able to implement most UI aspect of the application using a simple UIViewCotnroller Protocol however sometimes I am in need of more functionality in which I then have to introduce more protocols.



```
import UIKit

class AdminLoginViewController: UIViewController , UITextFieldDelegate
{

    @IBOutlet weak var submit: UIButton!
    @IBOutlet weak var password: UITextField!
    override func viewDidLoad()
    {
        super.viewDidLoad()
        password.delegate = self
        password.text = ""

        self.view.backgroundColor = UIColor(patternImage: UIImage(named: "tile")!)
        submit.layer.cornerRadius = 20
        submit.layer.borderWidth = 4
        submit.layer.borderColor = UIColor.black.cgColor

        password.layer.borderWidth = 4
        password.layer.borderColor = UIColor.black.cgColor

    }

    override func viewWillAppear(_ animated: Bool)
    {
        password.text = ""
    }

    @IBAction func submit(_ sender: Any)
    {
        if(password.text == "az1995")
        {
            performSegue(withIdentifier: "WordConfirmation", sender: nil)
        }
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool
    {
        password.resignFirstResponder()
        return true
    }

}
```

Dependency Inversion

There is not much dependency inversion in the application as there are not many senders and receivers in the application. Most of the dependency inversions take form as button listeners in the swift internal architecture.

5) Agile Milestones

Quarter 2:

Week 1-2:

Work on Translation Class.

Week 3-4:

Have the Farsi to Finglish and Finglish to Farsi functionalities completely done.

Week 5-6:

Have the Game aspects for both levels completely done.

Week 7-8

Due all the testing needed.

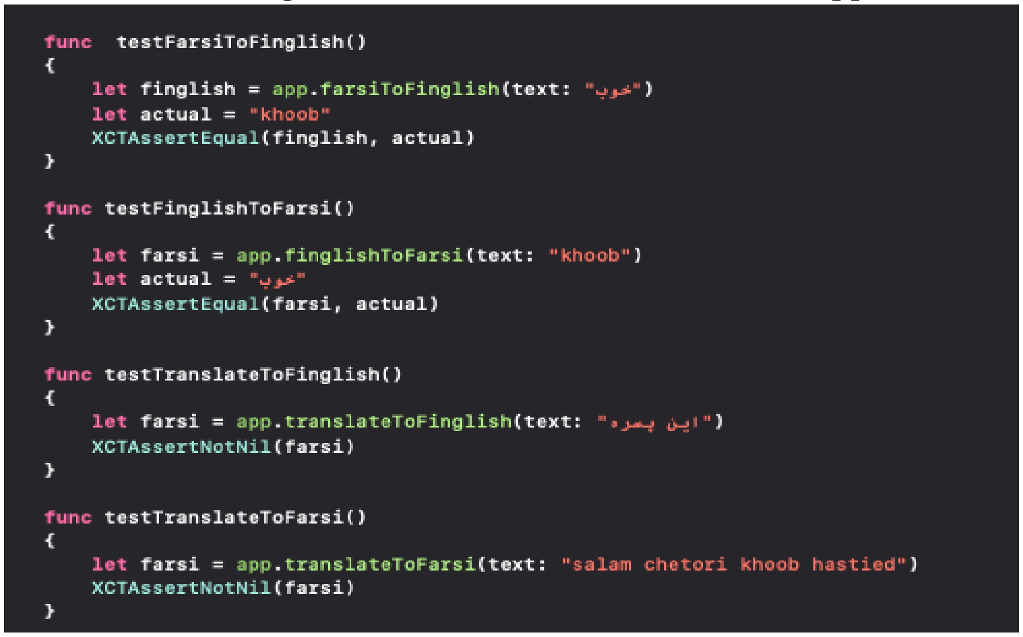
Week 9-10

Finish up all the documentation.

6) Testing

XCTest

I will be using xctest to the functionalities of the app



```
func  testFarsiToFinglish()
{
    let finglish = app.farsiToFinglish(text: "خوب")
    let actual = "khoob"
    XCTAssertEqual(finglish, actual)
}

func testFinglishToFarsi()
{
    let farsi = app.finglishToFarsi(text: "khoob")
    let actual = "خوب"
    XCTAssertEqual(farsi, actual)
}

func testTranslateToFinglish()
{
    let farsi = app.translateToFinglish(text: "این پسره")
    XCTAssertNotNil(farsi)
}

func testTranslateToFarsi()
{
    let farsi = app.translateToFarsi(text: "salam chetori khoob hastied")
    XCTAssertNotNil(farsi)
}
```

System Testing

Farsi to Finglish Translation

1. Open application.
2. Go to Farsi to Finglish Tab
3. Type سلام
4. Hit the Submit button
5. A salam should be in bottom translation box in black.
6. Now type خيلي in the above box.
7. Hit Submit.
8. Now kheili should be typed in red in the lower box

Add Words

1. Open application.
2. Go to Farsi to Finglish Tab
3. Type خيلي in the above box.
4. Hit Submit.
5. kheili should be typed in red in the lower box
6. Now press the Add Words button.
7. Find خيلي in the bar, and click on it
8. Enter the desired translation, click submit.
9. Type خيلي in the above box.
10. Now kheilie should be typed in green.

Admin Confirm Word

1. Open application.
2. Go to Admin Tab, type in the admin password
3. Click a word on the list
4. Confirm the word on the next page, and hit submit.
5. Now go to the Farsi to English translation page, the word you confirmed should now show up as black text when translated

7) Notes

Initially while creating this application, I did not believe I would need to incorporate a database aspect except for the letter mappings. However, after testing the application I saw that many of the translations would be very inaccurate if I was to solely base my app on mappings. I decided to add a database component that would be crowd sourced, users could add words to the database in order to make the translator more effective.