UNIVERSITI TUNKU ABDUL RAHMAN

ASSIGNMENT 4 (15%)

**UECS3263 iOS APPLICATION DEVELOPMENT**

BACHELOR OF SCIENCE (HONOURS) SOFTWARE ENGINEERING

|  |  |
| --- | --- |
| Name (as stated in Student Card) | Student ID |
| Gervin Fung Da Xuen | 1801655 |
| Programme | Submission Date |
| Software Engineering | 8/8/2021 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | | **Total Marks** | **Marks Given** |
| User Interface Components and Property List – CO2 (4 marks) | |  |  |
|  | Display of list of fruits/vegetables along with the calorie count (per 100g) | 1 |  |
|  | Entry of word to search | 1 |  |
|  | Display the fruit/vegetable found | 1 |  |
|  | Property list | 1 |  |
| App Construction and Execution – CO3 (8 marks) | |  |  |
|  | Requirements implemented | 3 |  |
|  | Code design (including code readability) | 3 |  |
|  | Executes successfully | 1 |  |
|  | Screenshot of simulator (showing list of fruits/vegetables and the search result) | 1 |  |
| App Design – CO1 (3 marks) | | | |
|  | Explanation of code for searching | 3 |  |
| Total | | 15 |  |

Contents

[**Application Design** 4](#_Toc79335296)

[**Explanation on Searching** 6](#_Toc79335297)

[**Code Listing** 8](#_Toc79335298)

# **Application Design**

|  |  |
| --- | --- |
|  |  |
| Figure 1.1 *Initial Layout* | Figure 1.2 *Click Table Cell* |

Figure 1.1 show the initial layout when user click the application, notice the label is “Chosen fruit/vegetable” to indicate that the user did not click any fruit/vegetable. Figure 1.2 shows that when user click “Arugula” the label changed immediately to provide feedback to users to show that it indeed changes when user click on different fruit/vegetables.

|  |  |
| --- | --- |
| Graphical user interface, text, application, Teams  Description automatically generated | Graphical user interface, application, Teams  Description automatically generated |
| Figure 1.3 *Return of Search Bar Result* | Figure 1.4 *Click Table Cell* |

Referring to the Figure 1.3 above, the searching algorithm works by checking if the name of the vegetables/fruits contain the word entered by the user and return the list that contains the word (detailed explanation in the next section). Notice that in Figure 1.4, the label change as user taps “Green Olives”.

# **Explanation on Searching**

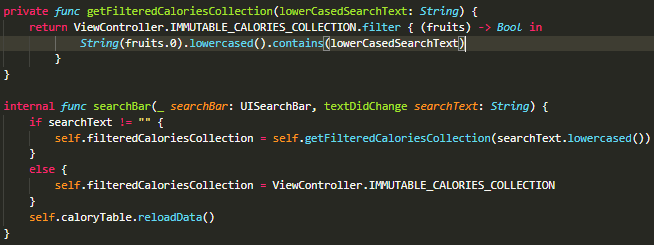


Figure 1.5 *Entire Substring search algorithm*

Referring to Figure 1.5, it’s evident I did not write a real searching algorithm and used *contains* function provided by Swift library because library method provide by *Swift* is definitely better and has been optimized for performance. Hence it would be redundant to write one myself when there’s a function that does the same thing with better performance.

The substring search I implemented for this assignment will check if searchText variable is consists of empty string or not. If it’s empty, it will be assigned to the original dictionary read from the property list. Otherwise, it will run and check if there’s any key from the dictionary that contains the lowercased search text, to do that each key (String) in the dictionary is also transformed to lowercased text. This is done so that the searching algorithm will ignore the case of each letter in the search text as well as the key in the dictionary. For example, if user enter ‘GREEN” but the list has 2 key that contains the word “Green”, then the search result return by the search bar is of empty list because of differences in case of letter. However, if we transform the word “GREEN” to “green” and the 2 keys into lowercase letter, then the search result returned by the search bar will not be empty, hence this approach is more user friendly.

|  |
| --- |
|  |
| Figure 1.6 *Substring search algorithm* |
|  |
| Figure 1.7 *Another* *Substring search algorithm* |

Referring to Figure 1.6, which is the substring search I implemented for this assignment, while Figure 1.7 would be another substring search algorithm pseudocode which I have written for comparison.

One would argue that the method used in Figure 1.6 is more intuitive as that is how we have been taught of using for-loop and hence would be easier to reason with. However, if one were to dive in deeper to the concept of *“Referential Transparency”* and the concept of immutability*,* which emphasis on the concept that a pure function should receive input, produce an output with no side-effect – any mutation or unpredictable behavior, one would know the method in Figure 1.6 didn’t break the concept of “*Referential Transparency”* whereas the method in Figure 1.7 broke it.

Hence, I personally believe that the method shown in Figure 1.6 is better. This is because the method shown in Figure 1.6 receive 1 input “*lowerCasedSearchText”* as parameter*,* and returns one output, a dictionary which its key (String) contain the String assigned to “*lowerCasedSearchText”* with no side effect, unlike the method shown in Figure 1.6, where the variable *“list”* is mutated by performing append, hence the variable *list* is mutated by adding tuple into the list, that is the side-effect.

Of course, by no means the method used in Figure 1.7 is bad, it’s just a matter of personal preferences. In a simple program the effect of mutation will not have great impact, this is not the case for a robust program. Furthermore, the code written in Figure 1.6 is cleaner and more readable than that of in Figure 1.7. Hence, I would try to reduce mutation whenever possible, it’s just a matter of personal practices.

# **Code Listing**

//

//  ViewController.swift

//  asm4

//

//  Created by Guest User on 8/8/21.

//  Copyright © 2021 Guest. All rights reserved.

//

import UIKit

final *class* ViewController: UIViewController, UISearchBarDelegate, UITableViewDataSource, UITableViewDelegate {

    private static let CELL\_IDENTIFIER: *String* = "cell"

    private static let IMMUTABLE\_CALORIES\_COLLECTION: *Dictionary*<*String*, *String*> = ViewController.readPropertyList()

    private static let SORTED\_KEY = *Array*(ViewController.IMMUTABLE\_CALORIES\_COLLECTION.keys).sorted()

    private var filteredCaloriesCollection: *Dictionary*<*String*, *String*> = ViewController.IMMUTABLE\_CALORIES\_COLLECTION

    @IBOutlet weak var calorySearchBar: UISearchBar!

    @IBOutlet weak var caloryTable: UITableView!

    @IBOutlet weak var caloryLabel: UILabel!

    override *func* viewDidLoad() {

        super.viewDidLoad()

        self.caloryTable.dataSource = self

        self.caloryTable.delegate = self

        self.calorySearchBar.delegate = self

        self.caloryTable.register(UITableViewCell.self, forCellReuseIdentifier: ViewController.CELL\_IDENTIFIER)

        self.caloryLabel.textAlignment = .center

        self.caloryLabel.text = "Chosen fruit/vegetable"

    }

    private static *func* readPropertyList() -> *Dictionary*<*String*, *String*> {

        if let filePath = Bundle.main.path(forResource: "caloriesInfo", ofType: "plist") {

            if let plistData = FileManager.default.contents( atPath: filePath) {

                do {

                    let plistObject = try PropertyListSerialization.propertyList(from: plistData, options: PropertyListSerialization.ReadOptions(), format: nil)

                    let caloriesDict = plistObject as? *Dictionary*<*String*, *String*>

                    return caloriesDict!

                } catch {

                    print("Error serializing data from property list")

                }

            } else {

                print("Error reading data from property list file")

            }

        } else {

            print("Property list file does not exist")

        }

        return [String: *String*]()

    }

    private *func* getLabelTextTuple(index: *Int*) -> (*String*, *String*) {

        let key = ViewController.SORTED\_KEY[index]

        let value = self.filteredCaloriesCollection[key]! + " / 100g"

        return (key, value)

    }

    internal *func* tableView(\_ *tableView*: UITableView, numberOfRowsInSection *section*: *Int*) -> *Int* {

        return self.filteredCaloriesCollection.count

    }

    internal *func* numberOfSections(in *tableView*: UITableView) -> *Int* {

        return 1

    }

    internal *func* tableView(\_ *tableView*: UITableView, cellForRowAt *indexPath*: IndexPath) -> UITableViewCell {

        let tuple = self.getLabelTextTuple(index: indexPath.row)

        let text = tuple.0

        let subtitle = tuple.1

        let newCell = UITableViewCell(style: UITableViewCell.CellStyle.subtitle, reuseIdentifier: ViewController.CELL\_IDENTIFIER)

        newCell.textLabel!.text = text

        newCell.detailTextLabel!.text = subtitle

        return newCell

    }

    internal *func* tableView(\_ *tableView*: UITableView, didSelectRowAt *indexPath*: IndexPath) {

        let tuple = self.getLabelTextTuple(index: indexPath.row)

        let text = tuple.0

        let subtitle = tuple.1

        self.caloryLabel.text = "You search for \(text) with calories \(subtitle)"

    }

    private *func* getFilteredCaloriesCollection(lowerCasedSearchText: *String*) -> *Dictionary*<*String*, *String*> {

        return ViewController.IMMUTABLE\_CALORIES\_COLLECTION.filter { (fruits) -> *Bool* in

*String*(fruits.0).lowercased().contains(lowerCasedSearchText)

            }

    }

    internal *func* searchBar(\_ *searchBar*: UISearchBar, textDidChange *searchText*: *String*) {

        if searchText != "" {

            self.filteredCaloriesCollection = self.getFilteredCaloriesCollection(lowerCasedSearchText: searchText.lowercased())

        }

        else {

            self.filteredCaloriesCollection = ViewController.IMMUTABLE\_CALORIES\_COLLECTION

        }

        self.caloryTable.reloadData()

    }

    internal *func* searchBarSearchButtonClicked(\_ *searchBar*: UISearchBar) {

        searchBar.resignFirstResponder()

    }

}