

\_

**SCHOOL OF ENGINEERING AND TECHNOLOGY**

FINAL ASSESSMENT FOR THE BSC (HONS) INFORMATION TECHNOLOGY; BSC (HONS) COMPUTER SCIENCE; YEAR 2

ACADEMIC SESSION 2021; SEMESTER 3 PRG2104: OBJECT ORIENTED PROGRAMMING

Project DEADLINE: Week 14

INSTRUCTIONS TO CANDIDATES

* This assignment will contribute 50% to your final grade.
* This is an individual assignment.

**IMPORTANT**

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorized late submission of work.

- Coursework submitted after the deadline will be awarded 0 marks

-

**Lecturer’s Remark** (Use additional sheet if required)

I....N...g...J...ih....B...i.n......... (Name) 1...8..0...9..3..6...7..4..std. ID received the assignment and read the

comments..........J......i..h.....B.....i..n......... (Signature/date)

**Academic Honesty Acknowledgement**

“I ....N....g...J..i.h....B...i.n...................(student name). verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties *(refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme)* for any kind of copying or collaboration on any assignment.”

Ji…h...B.....i..n......1..4../.1..1../..2..0..2..1. (Student’s signature / Date)

Table of Contents

**[1.](#_Toc89123628)****[Class Diagram](#_Toc89123628)** [1](#_Toc89123628)

**[2.](#_Toc89123629)****[Program Description](#_Toc89123629)** [1](#_Toc89123629)

**[2.1](#_Toc89123630)****[Splash Screen](#_Toc89123630)** [1](#_Toc89123630)

**[2.2](#_Toc89123631)****[Adding Task](#_Toc89123631)** [3](#_Toc89123631)

**[2.3](#_Toc89123632)****[Viewing Task](#_Toc89123632)** [4](#_Toc89123632)

**[2.4](#_Toc89123633)****[Editing Task](#_Toc89123633)** [5](#_Toc89123633)

**[2.5](#_Toc89123634)****[Deleting Task](#_Toc89123634)** [6](#_Toc89123634)

**[2.6](#_Toc89123635)****[Database](#_Toc89123635)** [6](#_Toc89123635)

**[2.7](#_Toc89123636)****[Menu Bar](#_Toc89123636)** [6](#_Toc89123636)

[**2.7.1** **Close** 6](#_Toc89123637)

[**2.7.2** **App Instruction** 7](#_Toc89123638)

[**2.7.3** **App Detail** 8](#_Toc89123639)

[**2.8** **Responsive Design** 9](#_Toc89123640)

[**3.** **Personal Reflection** 10](#_Toc89123641)

[**3.1** **OOP Concepts Applied** 10](#_Toc89123642)

[**3.2** **Challenges Faced** 10](#_Toc89123643)

[**3.2.1** **Datepicker Implementation** 10](#_Toc89123644)

[**3.2.2** **Splash Screen Implementation** 10](#_Toc89123645)

[**3.3** **Strengths and Weaknesses of the App** 11](#_Toc89123646)

[**3.3.1** **Strengths** 11](#_Toc89123647)

[**3.3.2** **Weaknesses** 11](#_Toc89123648)

[**4.** **Conclusion** 13](#_Toc89123649)

[**5.** **References** 14](#_Toc89123650)

# **Class Diagram**

**Diagram

Description automatically generated**

# **Program Description**

The To Do List is an application that help users to stay organized and manage their day-to-day. The To Do List can be used to make users day plans that include task lists, shopping lists, and so on, which can increase productivity and focus for users. To Do List’s modern and easy-to-use experience makes users’ lists unique and special.

Program Feature:

1. Splash screen
2. Instruction Page
3. Add Task
4. Delete Task
5. Edit Task
6. Database
7. Menu Bar
8. Responsive Design

## **Splash Screen**

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 1: Splash Screen

To begin with, the To Do List app has a welcome screen to introduce the app to its users. As shown in Figure 1, the welcome screen has the logo of the app with the title of the app which is “TO DO LIST”. The welcome screen is a splash screen whereby the welcome screen will first fade in then fade out to the next screen.

Chart, treemap chart

Description automatically generated

Figure 2: The Main Screen of the To Do List App

Graphical user interface, text, application

Description automatically generated

Figure 3: Add & Edit Alert Dialog

## **Adding Task**

To add a task, we must click on the button called “New” as shown in Figure 2. Once the “New” button is clicked on, the Add & Edit alert dialog will appear as shown in Figure 3. In this alert dialog, user will be prompt to enter the task and 7 available subtasks they desire. The date is set to the current date as the default date, but users can also select a date they desire using the date picker. User can then select the OK button or click the enter key in their keyboard to add the task to the Main Screen.

Treemap chart

Description automatically generated

Figure 4: Main Screen After Addition of a Task

After clicking the OK button in the Add & Edit alert dialog, the task name will be added into the TableView as shown in Figure 4.

## **Viewing Task**

To view a task, users must select the task they want in the TableView as shown in Figure 4. Once clicked, the right side of the application will update on the task name, date, and the subtasks as shown in Figure 5.

Graphical user interface, table

Description automatically generated

Figure 5: View Task in the Main Screen

## **Editing Task**

Graphical user interface, table

Description automatically generated

Figure 6: Edited Task

To edit or update a task, users need to click on the “Edit” button after selecting the task they want to edit which will prompt up the same alert dialog as shown in Figure 3. An example of this can be seen with the comparison of Figure 5 and Figure 6. In Figure 6, the date of the task “Clean Room” is changed from 30/11/2021 to 03/12/2021, and the second subtask of the task “Clean Room” is also erased when compared to the pre-edited task shown in Figure 5.

## **Deleting Task**

To delete a task in the app, the user must select the desired task to be deleted on the TableView and click on the “Delete” button. If no task is selected but the “Delete” button is clicked an alert dialog will appear mentioning that there is no task selected as shown in Figure 7.

Graphical user interface, text, application, email

Description automatically generated

Figure 7: No Task Selected During Deletion

## **Database**

A database is implemented for the purpose of saving the data in the app. Without the database, all tasks added and edited will not appear in the application again after the program is closed and reopened.

## **Menu Bar**

For the menu bar, there are few features like closing the app, opening a dialog for the instructions of the application, as well as the app details.



Figure 8: Menu Bar

### **Close**



Figure 9: Close with Shortcut CTRL+C

With this feature, the user would be able to close the app by clicking “File” on the menu bar and then click “Close” as shown on Figure 9, this will shut down the app. Another method to shut down the app is by using the shortcut method where users would just click CTRL+C on their keyboard as shown on Figure 9 to exit the app.

### **App Instruction**



Figure 10: App Instruction with Shortcut CTRL+I

For this menu bar feature, users would be able to view the instruction on how to use the application. As shown on Figure 10, users will need to click on “Instruction” on the menu bar then click on the “App Instruction” to view the app instruction. Another way to achieve this is the shortcut version where users would click CTRL+I to achieve the same outcome.

Graphical user interface, text, application, email

Description automatically generated

Figure 11: App Instruction Dialog

When the user clicked on the necessary method to view the app instruction, a dialog that a contains the app instruction will be prompted for the user to view.

### **App Detail**



Figure 12: App Detail with Shortcut CTRL+H

For this menu bar feature, users would be able to view the application details. As shown on Figure 12, users will need to click on “Help” on the menu bar then click on the “About” to view the app details. Another way to achieve this is the shortcut version where users would click CTRL+H to achieve the same outcome.

Text, letter

Description automatically generated

Figure 13: App Detail Dialog

When the user clicked on the necessary method to view the app detail, a dialog that a contains the program description will be prompted for the user to view.

## **Responsive Design**

Chart, treemap chart

Description automatically generated

Figure 14: Application after enlargement

To accommodate with users that have different screen size, the application will also be responsive as shown in Figure 14 when compared to the default size shown in Figure 2.

# **Personal Reflection**

## **OOP Concepts Applied**

This system utilizes the model-view-controller (MVC) pattern, which is used to implement the user interfaces, data, and controlling logic. In this system, each view is assigned with its own controller that has different functions that different functions that can be assigned to elements in the view.

The model class or ToDo.scala, contains the variables of the data collected from the system that can be changed. Then an instance object of the class ToDo is created by calling a generic ObservableBuffer with the data type of ToDo, “val toDoData = ObservableBuffer[ToDo]()”. This object causes the data inputted by the users to be stored into a list that allows listeners to track changes when occurred. This data is then saved inside a database that has a table named “ToDo” by having inheritance done between the class ToDo and Database trait.

## **Challenges Faced**

### **Datepicker Implementation**

The Datepicker implementation would be the most challenging problem I have faced during this assignment. When implementing the Datepicker, the problem of the application not being able to retrieve the date picked from the Datepicker to display it on the label in main screen of the application.

This problem was solved by going through the ScalaFX API on Datepicker value member to determine which function to use to pass through the date picked to the label in the main screen of the application

### **Splash Screen Implementation**

Another problem I faced is during the implementation of the Splash Screen as the welcoming page of the application. I had problem making the welcome page to fade transition out that will show lead to the next screen of the application which is the main screen shown in Figure 2.

This problem is resolved after vigorous research and try and error which ultimately led me to a YouTube video as mentioned on [1]. However, with the video being done with Javafx, I had to go through ScalaFX API on Fade Transition to convert the code on [1] to ScalaFX code.

## **Strengths and Weaknesses of the App**

### **Strengths**

The strength of this system is that it includes all the necessary function for a to do list app. It allows users to manage their tasks with functions like adding new tasks, edit tasks, and deleting new tasks. The application also comes with a completion text where users can check the checkbox in the add edit dialog which will then prompt the label in the main screen to says, “Task Completed”.

Other than that, the system also has task list on the left side of the application which will list down all the tasks added by the users. This is done to allow an easy access to all the tasks as well as a simple looking task list that can be viewed by the users.

### **Weaknesses**

There are a few weaknesses of the system, mainly concerning the subtasks of each task. Firstly, the system only allows seven subtasks maximum for each task which is a negative prospect when users would like to create a grocery list which requires many different items. Beside the maximum subtasks problem, the system also has a problem when deleting a subtask when editing a task. When a subtask is removed, the subtask below the deleted subtask does not move up to replace the deleted subtask position, leaving a gap in the between subtasks as shown in Figure 15.

Graphical user interface, table

Description automatically generated

Figure 15: Subtask Removal Problem

Other than that, there is also a problem regarding the duplication of the tasks. The program allows users to add a new task that has a same name as other tasks that are already mentioned as shown in Figure 16. This causes a problem of duplication which will take up space and memory.

Graphical user interface, table

Description automatically generated

Figure 16: Task Duplication

# **Conclusion**

In short, the To Do List app is done using object-oriented programming and GUI programming. An UML Class diagram is used to describes the relationship between classes of the system. The system also has all the required functions of an To Do List application like adding new tasks, editing task, deleting task, and viewing task for users to record their desired tasks to do. Beside that, a user interface is also created using an extensive GUI library from ScalaFX and JavaFx that will provide a better user experience.

# **References**

[1] Genuine Coder, *JavaFX Splash Screen / Welcome Screen (Inside Same Window)*, Aug. 27, 2016. Accessed on: Nov. 29, 2021. [Video file]. Available: https://www.youtube.com/watch?v=muz6QLIgrC0&ab\_channel=GenuineCoder

[2] ScalaFX, FadeTransition, ScalaFX API 8.0.102-R11, n.d. Accessed on Aug. 29, 2021. [Online]. Available: http://www.scalafx.org/api/8.0/index.html#scalafx.animation.FadeTransition

[3] ScalaFX, DatePicker, ScalaFX API 8.0.102-R11, n.d. Accessed on Aug. 29, 2021. [Online]. Available: http://www.scalafx.org/api/8.0/index.html#scalafx.scene.control.DatePicker