

**Discipline of Information Technology, Media and Communication College of Art, Business, Law and Social Sciences (ABLSS)**

**ICT365 ASSIGNMENT 1**

**CHECK LIST**

**Surname (Family Name):** Wong

**Given Names:** Chung Lun

**Student Number:** 34134484

**Tutor’s Name:** Mr. Loke Kai Fatt

**Assignment Due Date:** 5th March 2022 **Date Submitted:** 5th March 2022

**Your assignment should meet the following requirements. Please confirm this (by ticking boxes) before submitting your assignment.**

* All details above are complete.
* I acknowledge and agree that the assessor of this assignment may, for the purpose of assessing this assignment, reproduce this assignment and provide a copy to another academic staff member.
* I am aware of Murdoch University’s assessment policy 8.1.8 that states that “Unit Coordinators have the right to submit any assignment to the University’s plagiarism detection software if they suspect plagiarism.”
* I have read and understood the requirements for submission of assignments specified in the Unit Information and Learning Guide.
* I have read and understood the requirements for documenting and submitting this assignment that are specified in the assignment question sheet of this assignment.
* You understand that the zip file must be submitted to ICT365 Unit LMS.
* You have kept a copy of this assignment, including the zip file.

**Detailed Description**

Purpose of the lifelogging application aims to provide user the interface and application logic to store the past events and life activities which can aids Dementia Sufferers to fight mental inability in recognition. As part of the core function of the application, users will be able to view all the past events that are saved and loaded into the application in an interactive manner. These events are plotted on the map canvas with icons representing each type of event. These events can then be retrieved and the relevant event details such as tweet messages, photo, video, Facebook text will be displayed to the user to assist them in memory rebuild. Other than that, users can also add new events into the application as and when by filling in the necessary information.

As Dementia patients are the target user pools of the application, the GUI is designed to be friendly, easy to use, and intuitive. For instance, while adding a new event, users are guided to fill in required details starting from selecting the event type to add. Subsequently, the application can render different input field and component based on the event type to the user and collect information accordingly. Other than that, a line to the nearest event or marker will also be plotted while user click on the canvas map to assist users in creation of related and linked events. In short, the integrated GUI metaphors in the application will enhance the user experience and provide convenience in adding/retrieving past events.

On top of the core features and core functionality as discussed above, there are also some additional features that can be integrated to the application to increase the interaction with users. First and foremost, events that are tightly related might aids user in recognizing the past events and memory. In this context, each event is linked with a set of events where there are relations. Since the relationship of the event linkage is unique to each other, a HashSet data structure is used to store the related events. The sets increase in size over time as more events are added and linked together, resulting in more events linkage which typically assist in mental cognitive and memory.

**Self-Diagnosis, Evaluation, & Declaration**

**Features that are fully implemented**

* Serialize the XML Event element and load into the application as Event class using LINQ to XML method
* Events are added into the lifelog-events.xml file with total count of 20 and at least 2 tracklog events are added.
* C# Dictionary has been used to store the Event class which contains different Event sub-classes, representing different Event Type
* Map XML Element to appropriate Event sub-classes (E.g., Twitter/Facebook)
* Display appropriate Event markers as intuitive icon on the GMap Canvas respectively, differentiate by the Event Type
* Retrieve nearest Event details (with line drawn) and display on new form
* Display Event details on marker clicked or Retrieve Event button clicked
* Render different Event details/attributes depend on Event type
* Display photo and play video files in sub-form
* Add new Event and display required input field according to Event type
* Save files (E.g., photo, videos, gpx) into project directory respectively
* Plot and add new Event on GMap Canvas
* View newly added Event detail and information on the GMap Canvas
* New created event can be added to the Dictionary that keeps the event data
* Link newly created event to existing event
* Incorporated design pattern (Factory method) into the application
* Unit testing of base and sub-classes, and Factory class

**Features that are not fully working**

* Unable to serialize “end-time” XML element into the application using deserialization
* Possible causes:
* Might be due to duplicated “datetimestamp” element in the XML file which causes difficulty in reading “end-time” element, especially it comes after the “start-time”

**Features that are not implemented**

* Unable to load .gpx file and display Tracklog event into the GMap Canvas
* Possible causes:
* Requires external library to convert .gpx file before loading into C# application
* Unable to append new XML Event element into the lifelog-events.xml file within SOAP-BODY

Possible causes:

* Design of the Event base and sub- classes might not align with the provided XML context, and hence serialization does not generate similar Event element structure.
* SOAP-ENV namespace is not accounted and hence the insertion of new XML Event element does not append into the SOAP-ENV Body
* \*\**Please refer to code Line 223 of FillEventDetailForm.cs for the attempt to serialize Event object into XML file.*

**Brief Description**

*XML Serialization & Deserialization*

To load and save the XML files into/to the application as the event data, LINQ to XML method is used to perform serialization and deserialization. Each event sub-classes are annotated with the XElement type and their respective element name before the serialization and deserialization happen. Without annotating the class attributes as XElement, the serialization will not map the responding XML element to the class attributes. On the other hand, the query function (e.g., *from, in, select*) of LINQ is also used to select the XML element in each event. The returning IEnumerable result then provides the capability to iterate through the desired event types and mapping can be done seamlessly to the corresponding event sub-classes. However, the serialization of event class to XML element had failed to attempt.

*Interface and Processing*

Interface of the application, hereby WinForm components are organized by each interface component. For instance, the MainForm consist of the core map canvas with event markers plotted on top, while other forms are created to serve only a single purpose (e.g., RetrieveEventForm is used to display event details, AddEventForm is used to allow user to select event types to create, etc). There are also a minimal amount of application logic incorporated in the Form component such as finding the nearest distance between 2 locations and find the nearest event marker in the canvas map (as GetDistance() and FindNearestMarker() function).

*Design Pattern*

The application involves loading of different event types such as Twitter, Facebook, Video, Photo, and Tracklog events. To facilitate the creation of these events, the factory method design pattern is incorporated into the application, namely the EventFactory class. While deserializing the XML file and loading data into the application, the GetEvent method in the EventFactory class is used. In this context, each event which is uniquely identified by the XElement of the respective event name is used to get the event types. The event types are then assigned to an enumeration of EventType which is declared in the EventFactory class. Subsequently, the corresponding Event sub-classes are returned to the main control and events are created accordingly. With this approach, the events that are created from serialization is transparent to the user during events creation.

*Refactoring*

The refactoring of the application first focuses on removing duplicate codes by creating a private method. For instance, the PlotMarkerOnMap() method is a product of refactoring to create markers on the map canvas. Without refactoring, marker plotting codes are duplicated upon creation of each event. In addition, the enumeration of EventType in EventFactory class is created to represent the state of each EventType. Since each event sub-classes consist of an EventType attribute, the event types can be validated before type casting and retrieval of event sub-class’s attribute. Class modifiers have also been refactored to ensure only required access is granted to the application. These includes changing the event’s attribute to protected and function used in respective form to private access as they will only be used within their class context. Moreover, try catch blocks are added to ensure exceptions are properly documented and acknowledged by the user. Several examples here are photo/video files that are faulty or unavailable are caught, XML files corruption or data changes which caused the application to load data are also caught, etc.

*Unit Testing*

To minimize maintenance and errors debugging, unit test of each important components such as event base, sub, and factory classes are unit tested individually. Particularly of events classes which represent the core component of the application, unit testing is conducted and focused on. For instance, the inheritance relationship between the base Event class, and sub- Twitter, Facebook, Photo, Video, and Tracklog events are tested by checking if each event is an instance of the sub-classes. Furthermore, for sub-classes which consist of DateTime data, the serialization of XML file reads the value as string and a separate method is used to return the DateTime type. These methods are also tested thoroughly to ensure the method can return accurate DateTime data type which represented by the string datetimestamp value. Finally, the EventFactory class which is used to create respective events are also unit tested to ensure the events that are created and returned from the factory are of the correct event sub-classes.

**Test Evidence**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Test Description** | **Expected Result** | **Actual Result** | **Outcome** |
| 1 | Map canvas showing on Winform and Events are marked as per the Event Type accordingly | Markers plotted on the canvas map | *Refer to Appendix A* | PASS |
| 2 | Select nearest event/marker on the map when user click on any location | A line is drawn from the clicked location to the nearest event, and coordinates are listed in the left panel | *Refer to Appendix B* | PASS |
| 3 | Retrieve the nearest event when user click on any location | A line is drawn, and user can click on the Retrieve Event button to view Event details | *Refer to Appendix C* | PASS |
| 4 | Display event information according to respective event details | Different value is shown to the user while retrieving different type of events | *Refer to Appendix D* | PASS |
| 5 | Form and value changes according to the Event Type when retrieving event | Photo event should show different attributes compare to Video events | *Refer to Appendix E* | PASS |
| 6 | Ability to select and add different Event | A new form should pop up to collect user input | *Refer to Appendix F* | PASS |
| 7 | Form input field changes according to different event type | Photo event fields should show different attributes than Video event input fields | *Refer to Appendix G* | PASS |
| 8 | Upload file while creating new Event, and save file to project directory respectively | Filepath should be read and displayed to user before upload, and file is saved in project directory. Filename is renamed with convention of “EventID+FileName” | *Refer to Appendix H* | PASS |
| 9 | Validate and prompt user to select location before adding new event | A message box should show up to prompt user accordingly | *Refer to Appendix J* | PASS |
| 10 | Validate and prompt user to select an event/nearest event before retrieving event | A message box should show up to prompt user accordingly | *Refer to Appendix K* | PASS |
| 11 | Show media files (e.g., photo/video) and allow user to open the file accordingly | Photo and video should show/play in a new form | *Refer to Appendix L* | PASS |

**Appendices**

Map

Description automatically generated*Appendix A*

*Appendix B*

Map

Description automatically generated

*Appendix C*

Map

Description automatically generated

*Appendix D*

Map

Description automatically generated

Map

Description automatically generated

*Appendix E*

Map

Description automatically generated

Map

Description automatically generatedMap

Description automatically generated

Map

Description automatically generated

Map

Description automatically generated*Appendix F*

*Appendix G*

Map

Description automatically generated

Map

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidenceGraphical user interface, website

Description automatically generated*Appendix H*

Map

Description automatically generated

Map

Description automatically generated*Appendix J*

Map

Description automatically generated*Appendix K*

*Appendix L*

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

**Source Code Listing**