CSP400 Software Delivery Project

Design Document

Benjamin Sampson
914545@swansea.ac.uk
May 8th, 2020

Contents

1 Design Document **3**
1.1 Introduction . . . 3
1.2 UI Design . . . 3
1.3 New Application Addition . . . 3
1.3.1 Settings UI Requirements . . . 3
1.3.2 Settings UI Specifications . . . 4
1.3.3 Settings UI Requirements and Specifications Cross Reference . . . 4
1.4 Workflow Map . . . 4
1.5 Class Overview . . . 4
1.5.1 Settings . . . 7
1.5.2 Frame Creation . . . 10
1.5.3 Sequence Creation . . . 15
1.5.4 Saved Frames . . . 19
1.5.5 Saved Sequences . . . 21
1.5.6 Export Frames/Sequences . . . 23
1.5.7 Util . . . 25
1.5.8 Main Activity . . . 25

1 Design Document

1.1 Introduction

In this document, I will be providing details in relation to the overall design of my application. This includes detailed information about each class and method along with a generic overview of how the application works and interacts with everything as shown [Figure 2].

1.2 UI Design

The overall design for the UI has changed slowly over the course of development, from being stepped and ordered, to giving the user the ability to select whatever they want whenever they want. This specific change was made to give the user more freedom when it came to creating their Frames or Sequences, and also to provided them with the ability to edit Frames or Sequences on the fly without any issues. The UI layout itsself is basic and simple although cluttered, however the UI will be modified according to user data taken from pilot studies. This information can help bring the application closer to being more user friendly.

1.3 New Application Addition

When further developing the application, it was discussed with my project supervisor, to add in a "Settings" area of the application. This area would primarily be used for assistance with future user studies and tests that will be conducted using my application and its associated hardware. Following this, I have subsequently detailed a new set of requirements and specifications that this area must include. These will be further evaluated using testing methods, the results of which will be presented in the Testing Document.

1.3.1 Settings UI Requirements

Requirement ID	Requirement Description
SEUIREQ1	The Settings UI must include switches for the user to select which user they are.
SEUIREQ2	The Settings UI must include switches for the user to select which task they wish to complete

Table 1: Table showing the Settings UI Requirements

1.3.2 Settings UI Specifications

Specification ID	Specification Description
SEUISPEC1	Each switch for selecting which user they are, must lock all other "user" switches.
SEUISPEC2	Each switch for selecting a user, will change values which change the location to load Frames or export Frames and Sequences too.
SEUISPEC3	Each switch for selecting the task, must lock all other "task" switches.
SEUISPEC4	Each switch for selecting a task, will change values which will change the location to load Frames or export Frames and Sequences too.

Table 2: Table showing the Settings UI Specifications

1.3.3 Settings UI Requirements and Specifications Cross Reference

Requirement ID	Specification ID
SEUIREQ1	SEUISPEC1, SEUISPEC2
SEUIREQ2	SEUISPEC3, SEUISPEC4

Table 3: Table showing the Settings UI Specifications

1.4 Workflow Map

[Figure 1] shows a map detailing the overall flow of the application when being used by a user. They have the ability to open a menu and navigate around the application from different areas, along with being able to create, view and export Frames or Sequences.

1.5 Class Overview

[Figure 2] depicts how each class links to a "Util" class which holds all the main data storage and boolean operators. This class will be explained further on. It also depicts how all the classes come together to form the finished application. Originally in my design, there was no "Util" class, which meant that every class interacted with each other continuously from the "MainActivity" but introducing a "Util" class to store this data ensured that there was less looping and easier communication between different areas of the application.

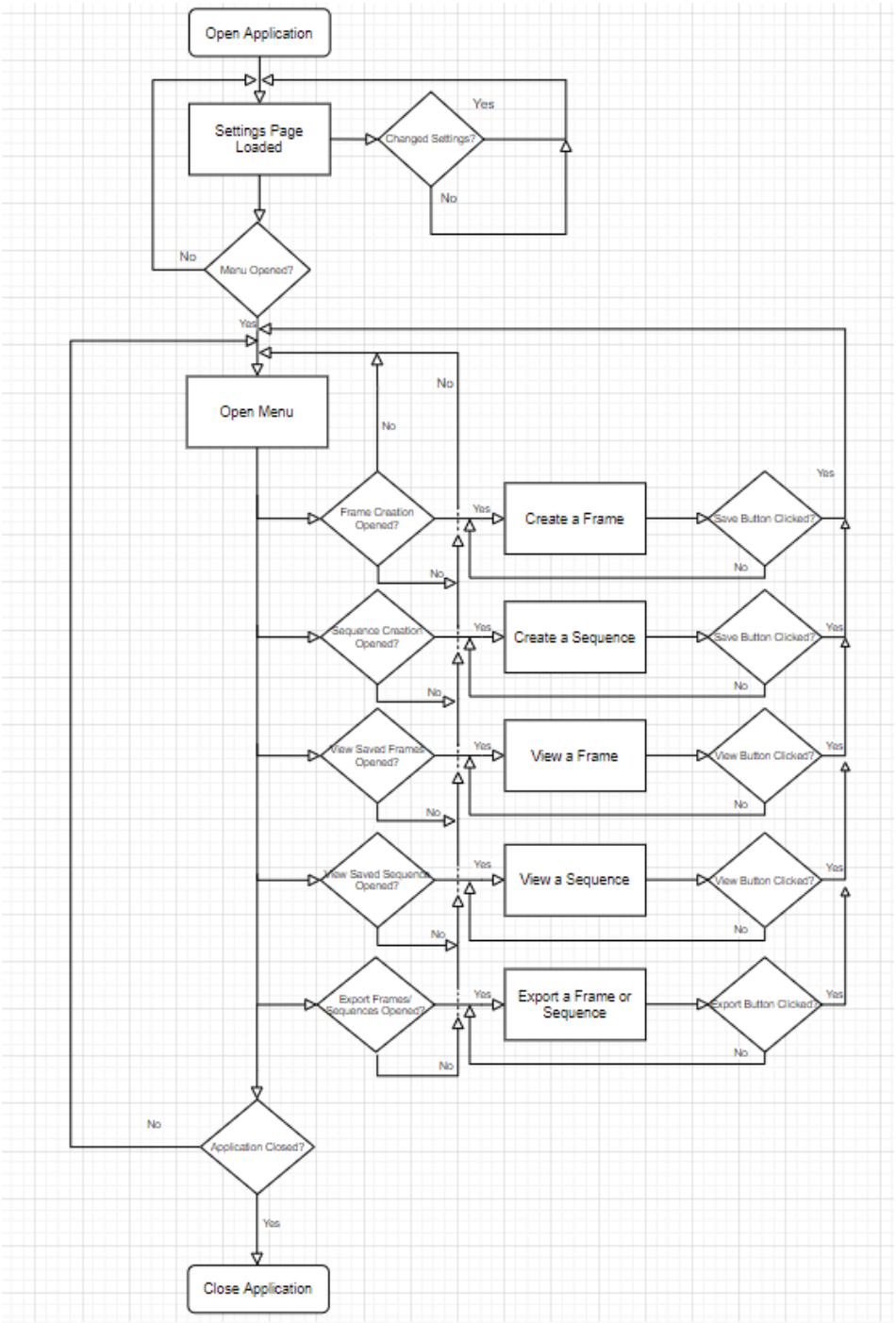

Figure 1: Workflow Map

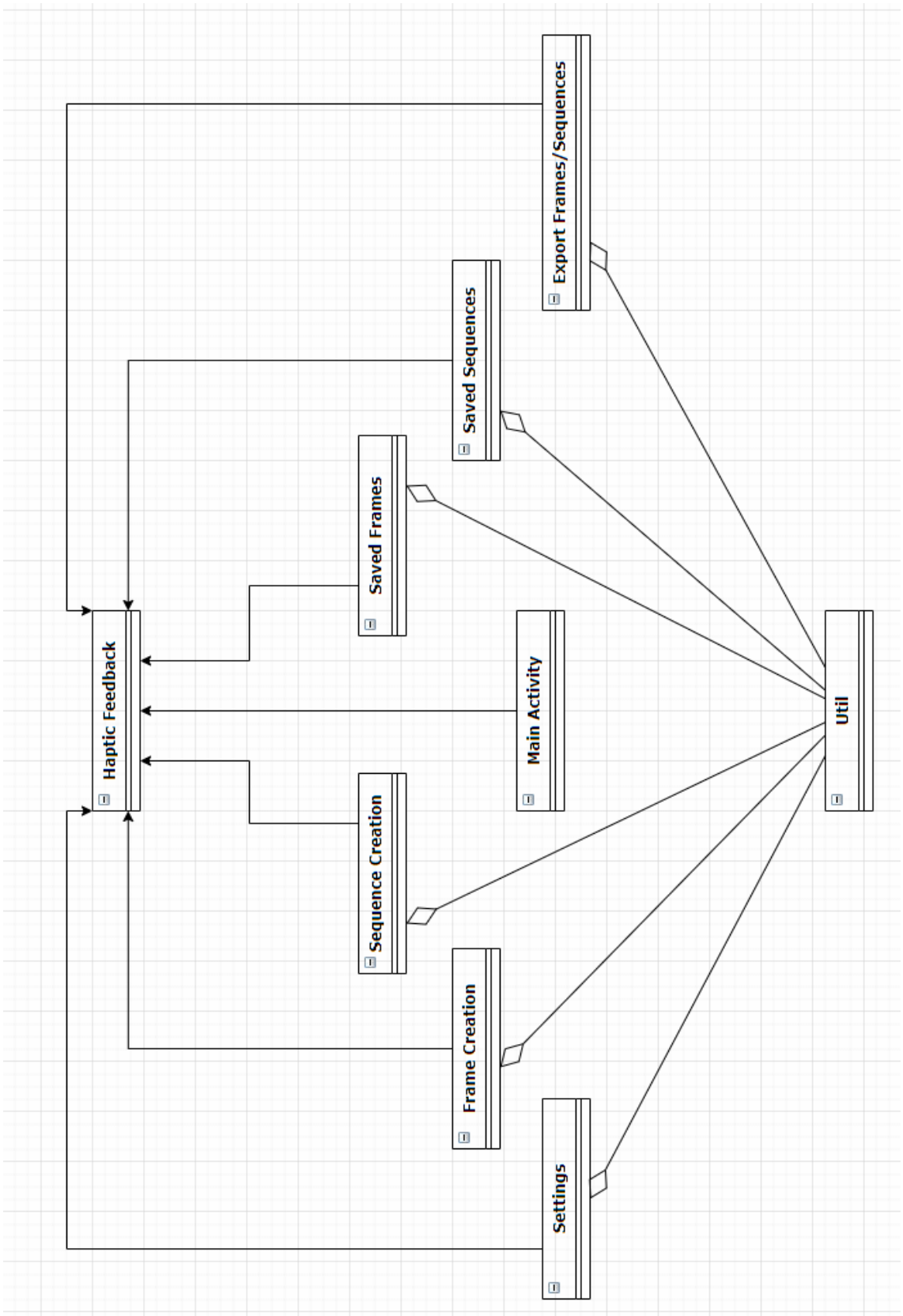

Figure 2: Class Overview

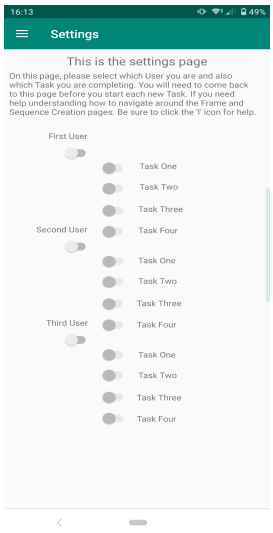

Figure 3: Settings UI

1.5.1 Settings

[Figure 4] shows the class diagram associated with the Settings. At the start of the application, the user will be presented with a "Settings" page. This page will dictate what user they are. This page was created to allow the user to set which task they are going to complete, and also allow user interactions and testing to be conducted. This class contains the methods; firstUser, secondUser, thirdUser, UserTaskOne, UserTaskTwo, UserTaskThree, UserTaskFour. This class also makes use of the "Util" class, which contains getters and setters for all the booleans being used, to indicate which switch has been selected.

The first three methods that are being used are: firstUser, secondUser, thirdUser, and all of these methods are relatively similar in structure. They are attached to the user switches as shown in [Figure 3] and contain boolean information that, whenever the appropriate switch is flicked, it sets that boolean to "true" and disables the rest. This ensures that the user can only select one user at a time, this also allows communication across the rest of the application as each saving and loading method will detect which user has been selected, so it knows where to load and save files from.

The second set of methods that are included in this class are: userTaskOne, userTaskTwo, userTaskThree, userTaskFour, and like the user methods, are all of similar structure. They contain boolean operators that are assigned true or false if they have been checked or not. This works the same way as users, where if one is ticked, the rest are disabled. The application will use these settings to decide where to load and save files during usage.

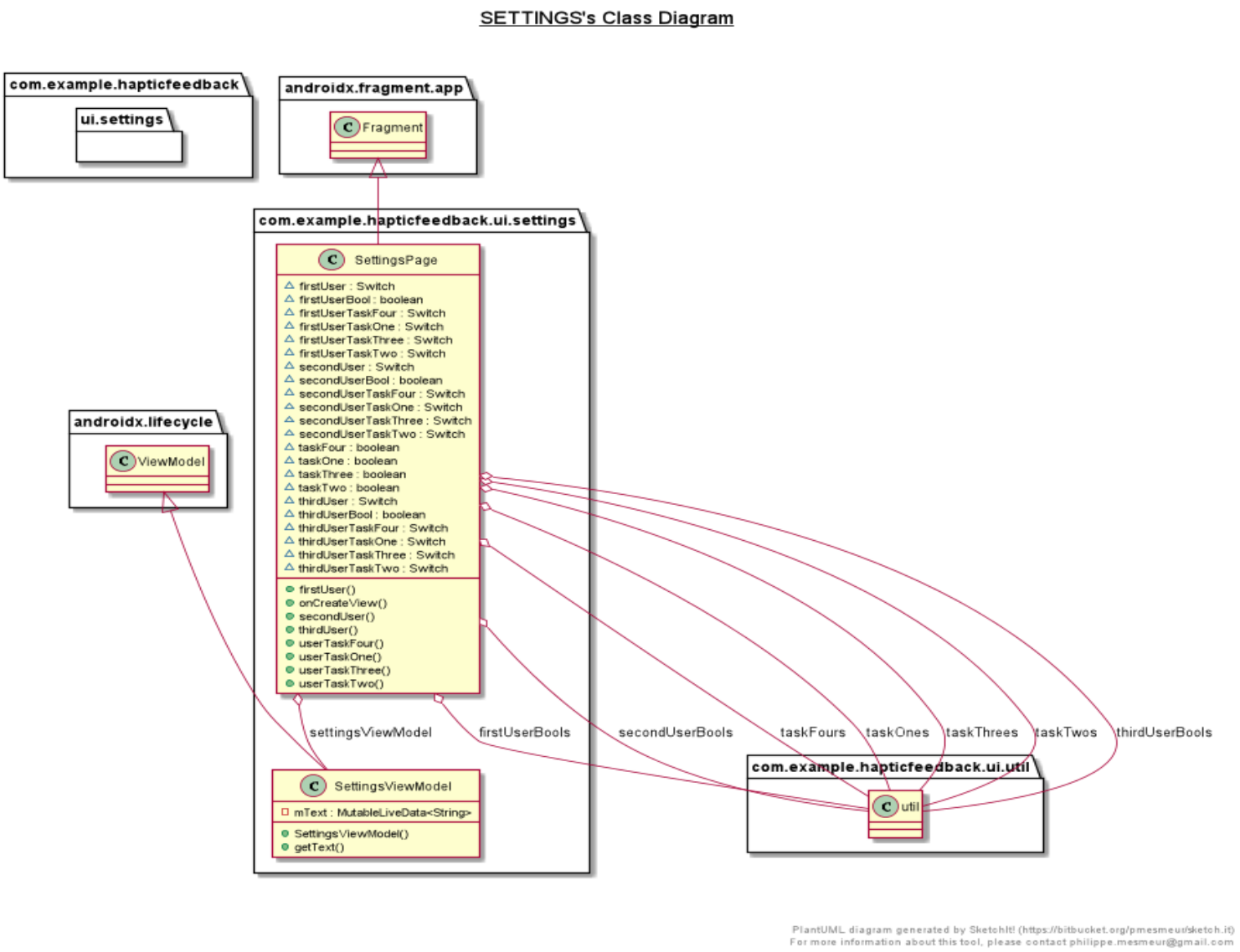

Figure 4: Settings Overview

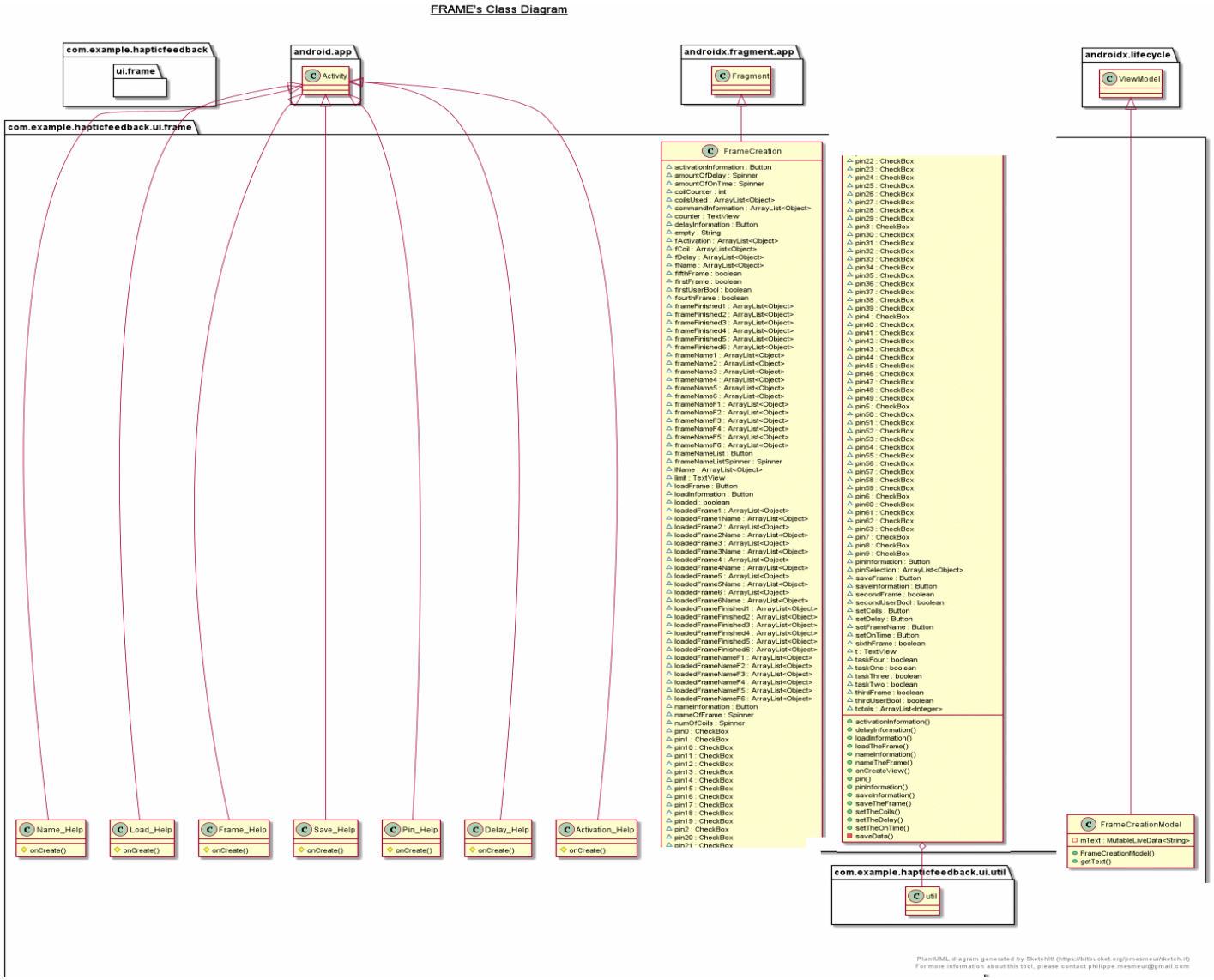

Figure 5: Frame Creation Overview

1.5.2 Frame Creation

[Figure 5] shows the class diagram associated with creating a Frame. The 'ui.frame' folder contains all the classes that add functionality to creating a Frame, where the main functionality takes place inside the 'FrameCreation' class. This class contains the following methods: activityInformation, delayInformation, loadInformation, nameInformation, pinInformation, saveInformation, loadTheFrame, nameTheFrame, pin, saveTheFrame, saveData, setTheCoils, setTheDelay, setTheOnTime. This class also makes use of the "Util" class, which contains getters and setters for all the booleans and ArrayLists being used, to allow for storing and retrieving of data across the application.

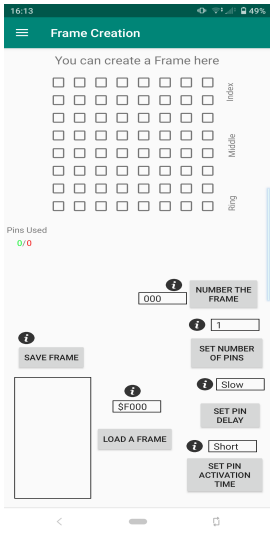

Figure 6: Frame Creation UI

As shown in [Figure 6], there are 6 'i' icons; these provide the user with detailed information about the specific functions available to them and also information on how to create a Frame. These classes; activityInformation, delayInformation, loadInformation, nameInformation, pinInformation, saveInformation, are all similar. They all contain a simple onCreate method that, when their respective 'i' icon is clicked, it displays an image pop-up showing the different options and information in regards to that feature. This will help the user better understand the creation of a Frame during use and allow the user to work closely with a provided User Manual.

The 'nameTheFrame' method works in unison with a spinner which gives the users a range of numbers to name their Frame. These numbers are unique to that Frame and are set up in a way so that the user is prevented from using the same name twice. Once the button has been clicked, it will be added to a creation view and also a subsequent ArrayList that stores this data. The data is stored in 3 locations; fName for verification, frameName1-6 names used in Frame creation and frameNameF1-6 for lower-case names used in Sequence creation. The data stored in frameName1-6 is then added into the commandInformation block, which will be the header of the Frame data. Anything inside this block will be saved as a finished Frame.

The 'setTheCoils' method works with a spinner which provides the user with number values. These values indicate the amount of pins the user wishes to use when making their Frame. This number can be changed at any time if they wish to add or remove pins, however the user selection must be met. The data stored in this method goes into 1 location; fCoil for verification to check that the coil amount has been met. The data selected goes straight into the commandInformation block and adds in the second index after frameName. This method also makes use of the 'loaded' boolean, which means that if the switch associated is flicked, they will be editing a loaded Frame. In user tests, having the ability to compare between old and new Frames is useful; this is saved as a different Frame despite being allowed the same name to ensure that data is not overwritten.

The 'setTheDelay' method works with a spinner which provides data for the delay between pin activation. The value that the user selects can be changed at any time and also makes use of the 'loaded' boolean if they are editing a loaded Frame and not creating a new Frame. The data that this method uses is stored in 1 location; fDelay for verification, ensuring the delay time has been set and not missed out. This data is then directly passed into the commandInformation block as the third index.

The 'setOnTime' method works with a spinner which provides the user information on how long they want a pin to be activated for. The value that the user selects can be changed at any time and also makes use of the 'loaded' boolean if they are editing a loaded Frame and not creating a new Frame. The data used in this method is stored in 1 location; fActivation for verification, ensuring the activation time has been set and not missed out. This data is then passed into the commandInformation block as the fourth index. With this finial passing of information into the commandInformation Array, the header for creating a frame is complete.

The 'pin' method works with the 8x8 check box selection at the top of the page. This works in unison with the 'setTheCoils' method, and can't be used until a user has set the amount of coils they wish to use in Frame creation. This method contains a 64 switch case statement which individually programs each check box with their own counter and pin code. They have to be individually programmed as the user can add or remove them in any order along with each pin having their own unique code. The data that this method uses is stored in 2 locations; totals to indicate how many more pins can be selected before the desired amount is reached, and pinSelection, which adds the unique pin code to an Array containing the information of all pins selected.

The 'saveData' method uses the 'saveTheFrame' method to actually save data. However, the method runs several checks before it saves data. It uses several verification Arrays (fName, fCoil, fDelay, fActivation), to ensure that the correct user inputs have been made, alongside ensuring that commandInformation is filled with correct data and all pinSelections have been chosen. The method also makes use of the 'loaded' boolean to determine if the user has edited a loaded frame or not. Once the button is clicked, it attaches pinSelections to the end of commandInformation, ensuring that there is the required order to the finished Frame.

If the user has edited a Frame, it uses the totals Array to ensure that the correct amount of pins have been selected before saving data into 3 different Arrays as follows; loadedFrameFinished1-6 is used to save the edited Frame into a new Array using data from commandInformation and pinSelections, loadedFrame1-6Name which contains an upper-case value of the name, which will be used to load or export the Frame later and loadedFrameNameF1 which contains an lower-case value which will be used when creating a Sequence. An ending value is also then saved at the end of the finished Frame, which will ensure successful termination of the Frame when used in conjunction with the hardware.

If the user is creating a new Frame, it uses the totals Array to ensure that the correct amount of pins have been selected, before saving data into 1 Array; frameFinished1-6 which will combine commandInformation, pinSelections and an ending value into 1 Array which can be viewed, edited, loaded, exported or used later on in the application.

Once the button has been clicked, and a successful Frame has been saved, it will read data associated to that Frame, and provide a coloured addition to specific pins. This shows the user the previous Frame that they have made, and will make it easier to create patterns without leaving the application. The button will also clear all Arrays to allow for new Frames to be made whilst also un-ticking any ticked check boxes, resetting the Frame Creation page.

The last method used when creating a Frame is 'loadTheFrame' which will take boolean data from the Settings page. It will search what user and task they are on, then provide the ability for the user to load Frames from that task. This will then read the imported file data and display it in not only the text view but also colour the 8x8 grid of check boxes, showing what the user has made in the Frame they have just loaded. From this, they will be able to edit the number of pins used, activation time, coil on time and also select new pins if they wish.

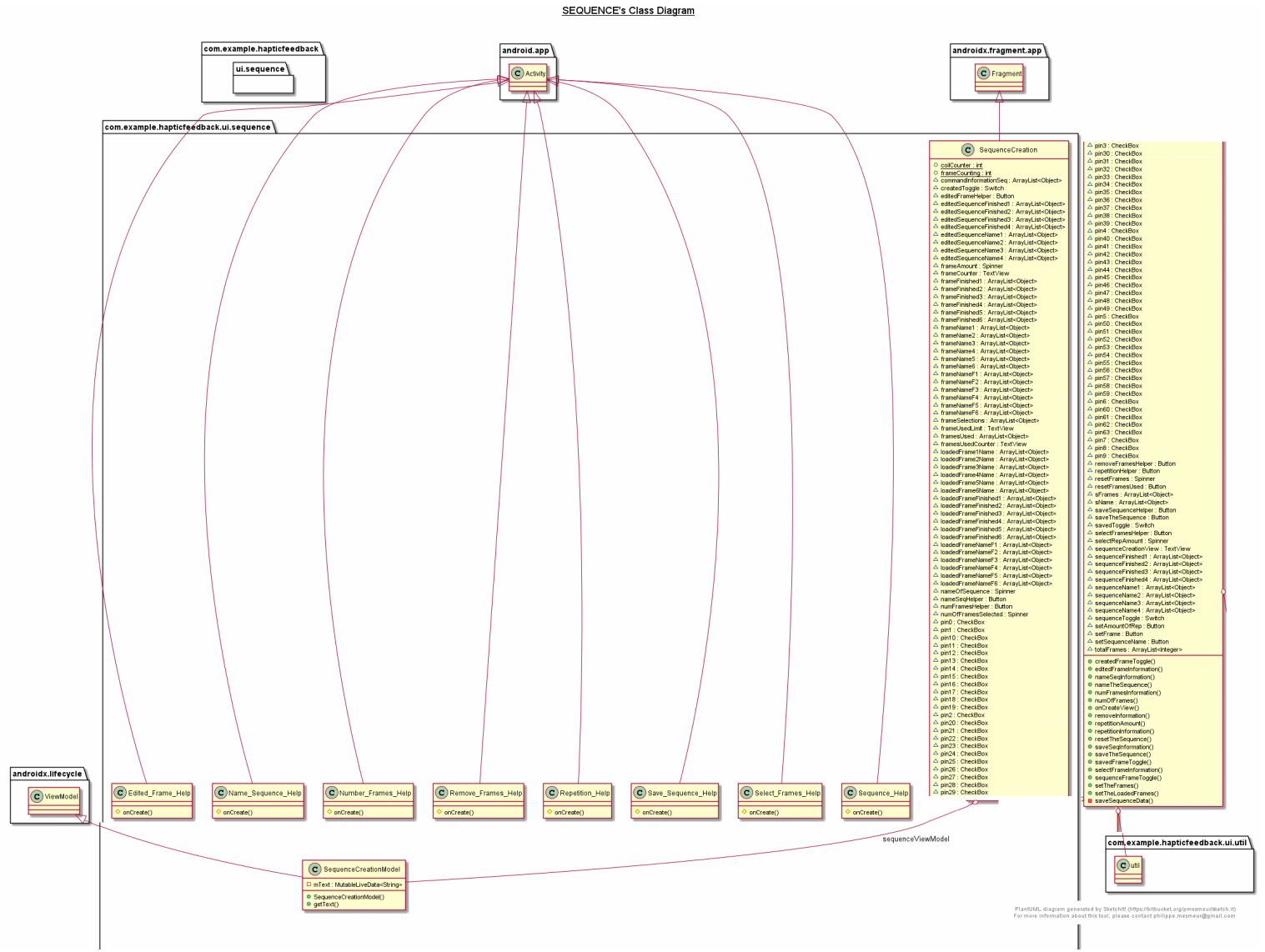

Figure 7: Sequence Creation Overview

1.5.3 Sequence Creation

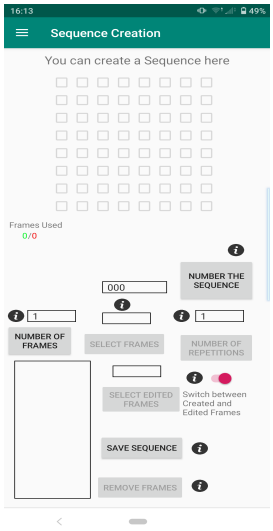

Figure 8: Sequence Creation UI

[Figure 7] shows the class diagram associated with creating a Sequence. The 'ui.sequence' folder contains all the classes that add functionality to creating a Sequence, where the main functionality takes place inside the 'SequenceCreation' class. This class contains the following methods: createdFrameToggle, editedFrameInformation, nameSeqInformation, nameTheSequence, numFramesInformation, numOfFrames, removeInformation, repetitionAmount, repetitionInformation, resetTheSequence, saveSeqInformation, saveTheSequence, selectFrameInformation, setTheFrames, setTheLoadedFrames, saveSequenceData. This class also makes use of the "Util" class, which contains getters and setters for all the booleans and ArrayLists being used, to allow for storing and retrieving of data across the application.

As shown in [Figure 8], there are 7 'i' icons that provide the user with detailed information about the specific functions available to them and also information on how to create a Sequence. These classes; editedFrameInformation, nameSeqInformation, numFramesInformation, removeInformation, repetitionInformation, saveSeqInformation, selectFrameInformation, are all similar. They all contain a simple onCreate method that, when their respective 'i' icon is clicked, displays an image pop-up showing the different options and information in regards to that feature. This will help the user better understand the creation of a Sequence during use and is corroborated via the User Manual.

The 'savedFrameToggle' method works with a switch and gives the user the ability to swap between two spinners. One spinner contains all the Frames they have made recently, during their usage of the application, the other contains Frames that they have recently edited. This is to split up both the created and edited Sequences so a comparison between the two can be conducted in user studies.

The 'nameTheSequence' method works with a spinner and provides the user with a list of selections they can use, to number their frame. This works in the same way as the 'nameTheFrame' method for Frame Creation. These numbers are unique to that Sequence and are set in a way so that the user cannot use the same name twice. Once the button has been clicked, it will be added to a creation view and also subsequent ArrayLists that stores this data. The data is stored in 3 locations; sName for verification, sequenceName1-6 used for names in Sequence Creation and editedSequenceName1-6 if the user is using edited Frames to make a Sequence. The data stored in edited/sequenceName1-6 is then added to the commandInformationSeq block, which will be the header of Sequence data. Anything inside this block will be saved as a finished Sequence. This method also makes use of a 'sequenceToggle' switch, which is used to indicate whether they are using edited or created Frames, which will be used to save their Sequence in a different location.

The 'numOfFrames' method works with a spinner which provides the user with the ability to select how many Frames they wish to use, when creating their Sequence. This number can be changed at any time if the user wishes to add more Frames. The data for this method is stored in 1 location; sFrames for verification to make sure the user has selected their desired number of Frames. This data is then stored in commandInformationSeq in the second index of making a Sequence. This method uses both frameFinished1-6 and loadedFrameFinished1-6 to load spinners with Frames they have either edited or created recently. This data can also be changed whenever the user likes, giving them the ability to add or remove how many Frames they want in the Sequence without a full restart.

The methods 'setTheFrames' and 'setTheLoadedFrames' are both used in a similar way. Once the user has selected the number of Frames they wish to use, depending on the toggle at the time, they will be able to select Frames from either the created or edited Frame spinners. It makes use of totalFrames to ensure that the maximum number of Frames has not been reached. All data is stored in 1 location; frameSelections, which holds all their selected Frames that will be used for the creation of a Sequence. As Sequences use a lower-case letter to identify Frames, it uses frameNameF1-6 to do this. The data contained in frameSelections is then stored below commandInformationSeq to be used when creating a Sequence. Depending on the Frames that the user selects, their data will be read and shown graphically on the 8x8 grid of check boxes, allowing the user to see the Sequences as they are making them.

The 'repetitionAmount' method works with a spinner and is activated once the user has inputted the Frame they wish to use. This indicates how many repetitions that Frame will make before proceeding onto the next Frame in the Sequence. The data used in this method is stored in 1 location; frameSelections which will pair the repetition amount to the Frame. The data is then stored in frameSelections, which is then displayed to the user. This method also uses the toggle, to ensure it is saving the repetition amount of a Frame to its desired location.

The 'resetTheSequence' method makes use of a button and gives the user the ability to remove all currently added Frames to their Sequence. This method clears all data that has been stored in relation to frameSelections and can be used if the user wants to remove Frames. This button does not remove any data stored in commandInformationSeq.

The 'saveSequenceData' method uses the 'saveTheSequence' method to actually save data. This method makes sure that the correct information has been entered that is required when making a Sequence, and also ensures that the user has added the required number of Frames to a Sequence The method also makes use of the 'sequenceToggle' boolean to determine if the user has used previously edited Frames or not. Once the button is clicked, it attaches frameSelections to the end of commandInformationSeq, ensuring that there is the required order to the finished Sequence.

If the user has used loaded Frames, it will save data similarly to Frame Creation, to ensure that you can compare Sequences that the user may have edited or created. If the user is creating a new Sequence using created Frames, then the data is stored separately to if they were using edited Frames. Both of these Sequences can be viewed by the user whenever they like.

Once the button has been clicked, and a successful Sequence has been saved, it will read data associated to that Sequence, and provide a coloured addition to specific pins. This shows the user previous Sequences that they have made, and will make it easier to create patterns without leaving the application. The button will also clear all Arrays to allow for new Sequences to be made and will reset the Sequence Creation page.

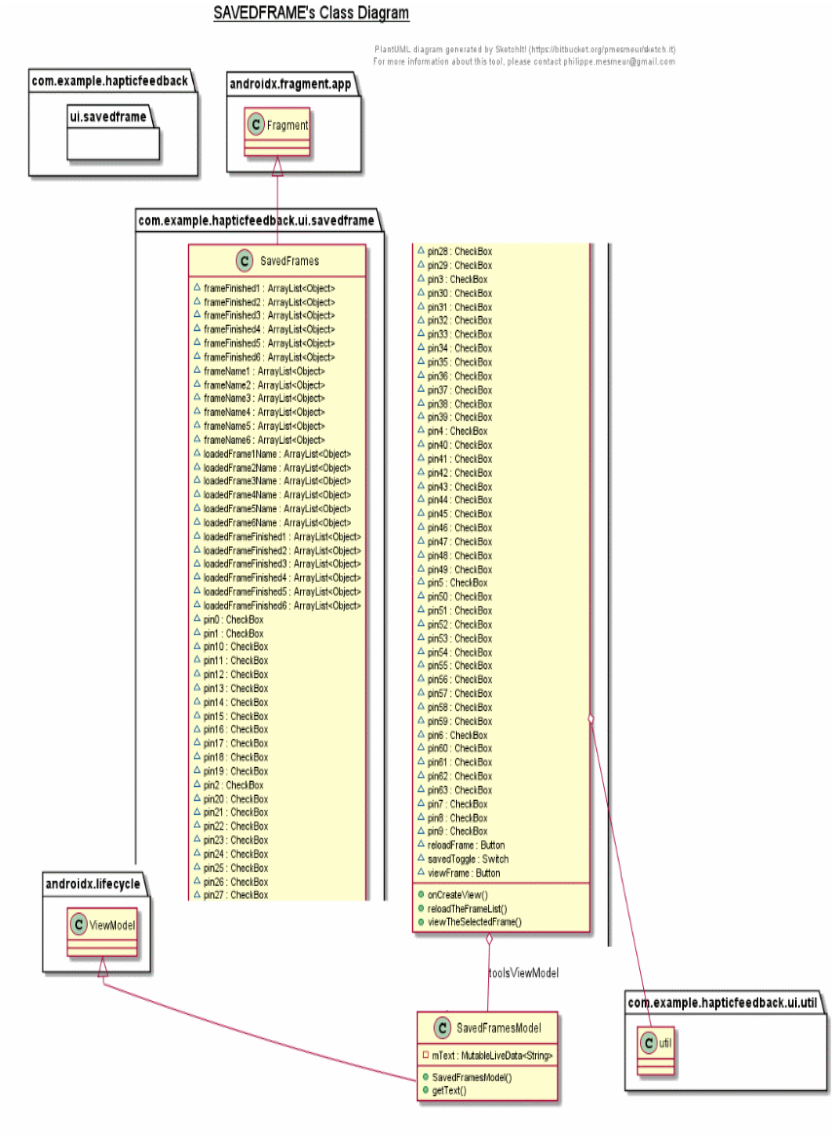

Figure 9: Saved Frame Overview

1.5.4 Saved Frames

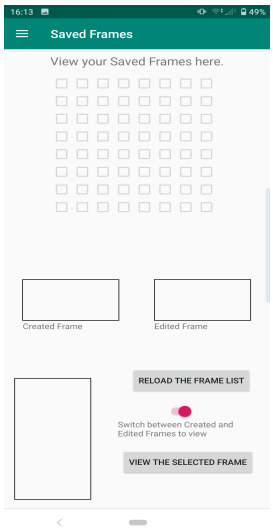

Figure 10: Saved Frame UI

[Figure 9] shows the class diagram associated with viewing a Saved Frame. The 'ui.savedframe' folder contains all the classes that add functionality to viewing a Saved Frame, where the main functionality takes place inside the 'SavedFrames' class. This class contains the following methods: reloadTheFrameList, viewTheSelectedFrame. This class also makes use of the "Util" class, which contains getters and setters for all the ArrayLists being used, to allow for storing and retrieving of data across the application.

[Figure 10] shows the Saved Frame UI. The 'reloadTheFrameList' method works with a button and 2 spinners. Once this button is clicked, it will fill both spinners with their respective data - this could be Frames that the user has made whilst on the application, or Frames that they have edited. It uses frameName1-6 and loadedFrameName1-6 Arrays to populate the spinners with whatever the user has done.

The 'viewTheSelectedFrame' method works with a button that reads the contents of either selected spinner, finds the finished or edited Frame data, then displays it in a text view. It works in unison with the savedToggle switch, which will swap between edited or created Frames, so the user can only view one at a time. Depending on what Frame they selected, this will subsequently change the 8x8 grid of check boxes, providing the user with visual aid for each Frame. This also gives the user the ability to differentiate between the Frames they select.

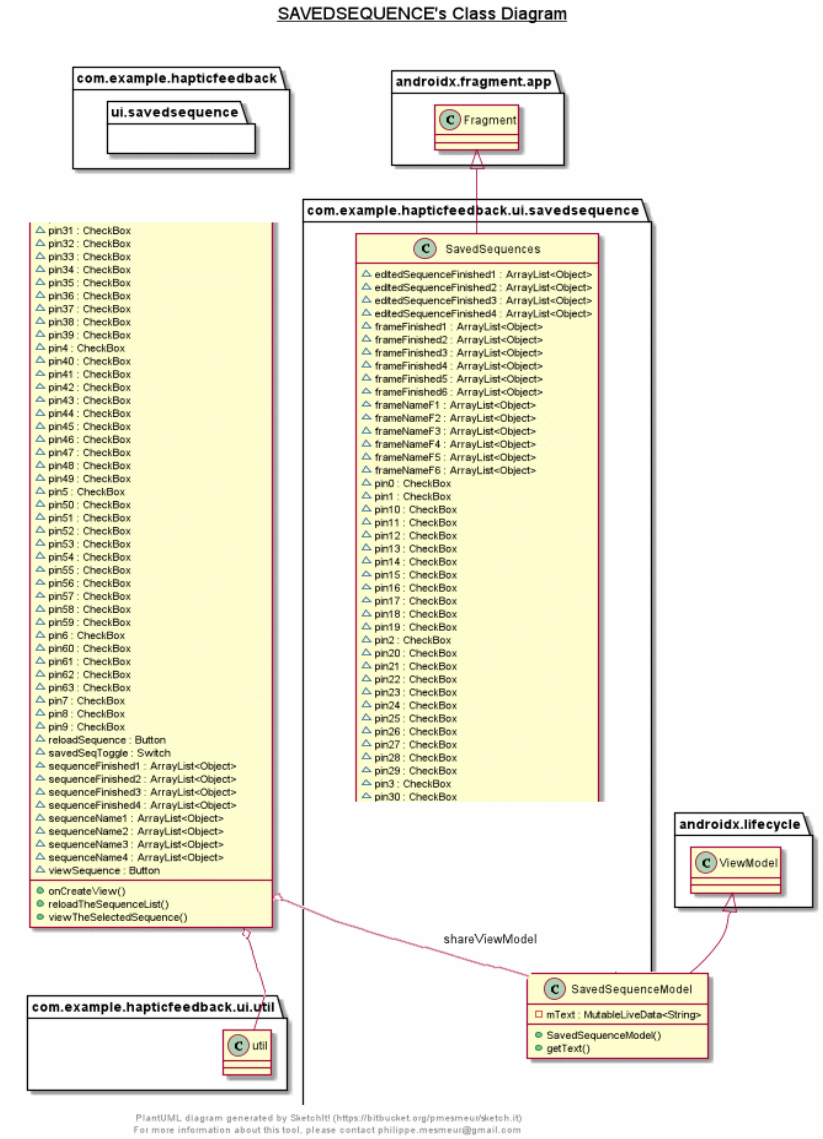

Figure 11: Saved Sequence Overview

1.5.5 Saved Sequences

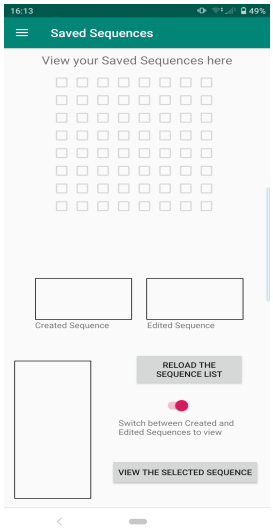

Figure 12: Saved Sequence UI

[Figure 11] shows the class diagram associated with viewing a Saved Sequence. The 'ui.savedsequence' folder contains all the classes that add functionality to viewing a Saved Sequence, where the main functionality takes place inside the 'SavedSequences' class. This class contains the following methods: reloadTheSequenceList, viewSelectedSequence. This class also makes use of the "Util" class, which contains getters and setters for all the ArrayLists being used, to allow for storing and retrieving of data across the application.

[Figure 12] shows the Saved Sequence UI. The 'reloadTheSequenceList' method works with a button and 2 spinners. Once this button is clicked, it will fill both spinners with their respective data, this could be Sequences that the user has made whilst on the application, or Sequences that contain edited Frames. It uses the sequenceName1-4 Array to populate the spinners with whatever the user has done.

The 'viewTheSelectedSequence' method works with a button that reads the contents of either selected spinner, finds the finished or edited Sequence data, then displays it in a text view. It works in unison with the savedSeqToggle switch, which will swap between Sequences made, or Sequence containing edited Frames, so the user can only view one at a time. Depending on what Sequence they selected, this will subsequently change the 8x8 grid of check boxes, providing the user with visual aid for each Frame within that Sequence. This also gives the user the ability to differentiate between the different Sequences they select.

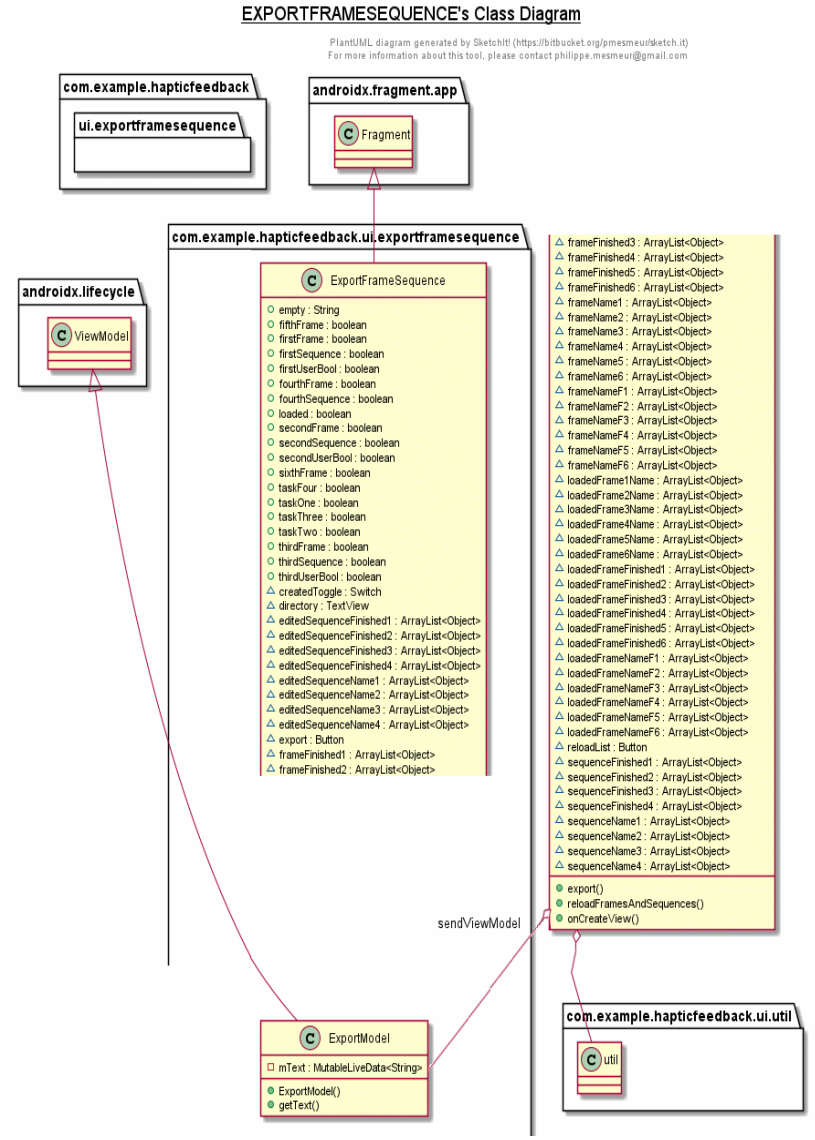

Figure 13: Export Frame/Sequence Overview

1.5.6 Export Frames/Sequences

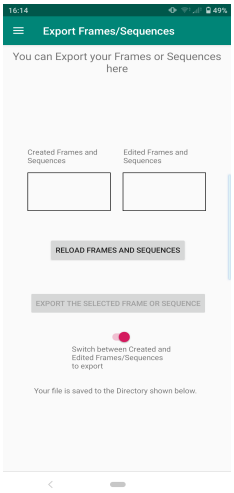

Figure 14: Export Frame/Sequence UI

[Figure 13] shows the class diagram associated with exporting a Frame or Sequence. The 'ui.exportframesequence' folder contains all the classes that add functionality to exporting a Frame or Sequence, where the main functionality takes place inside the 'ExportFrameSequence' class. This class contains the following methods: export, reloadFrameSequenceList. This class also makes use of the "Util" class, which contains getters and setters for all the Booleans and ArrayLists being used, to allow for storing and retrieving of data across the application.

[Figure 14] shows the Export Frame/Sequence UI. The 'reloadFrameSequenceList' method works with a button and 2 spinners. Once this button is clicked, it will fill both spinners with their respective data, this could be Frames and Sequences that the user has made whilst on the application, or Frames and Sequences that have been edited or contain edited data. It uses the frameName1-6, loadedFrame1-6Name, sequenceName1-4 and editedSequenceName1-6 Arrays to populate the spinners with whatever the user has done in that session.

The 'export' method works with a button that reads the contents of either selected spinner, finds the finished or edited data depending if the user has a Frame or Sequence selected. It works in unison with the createdToggle switch, which will swap between created and edited Frames or Sequences so the user can only export one at a time. Once the user clicks the button, it will read booleans associated to which user and task they have previously selected, along with if they are exporting a created or edited Frame or Sequence, and then will subsequently export the data to a specific directory on the user's device.

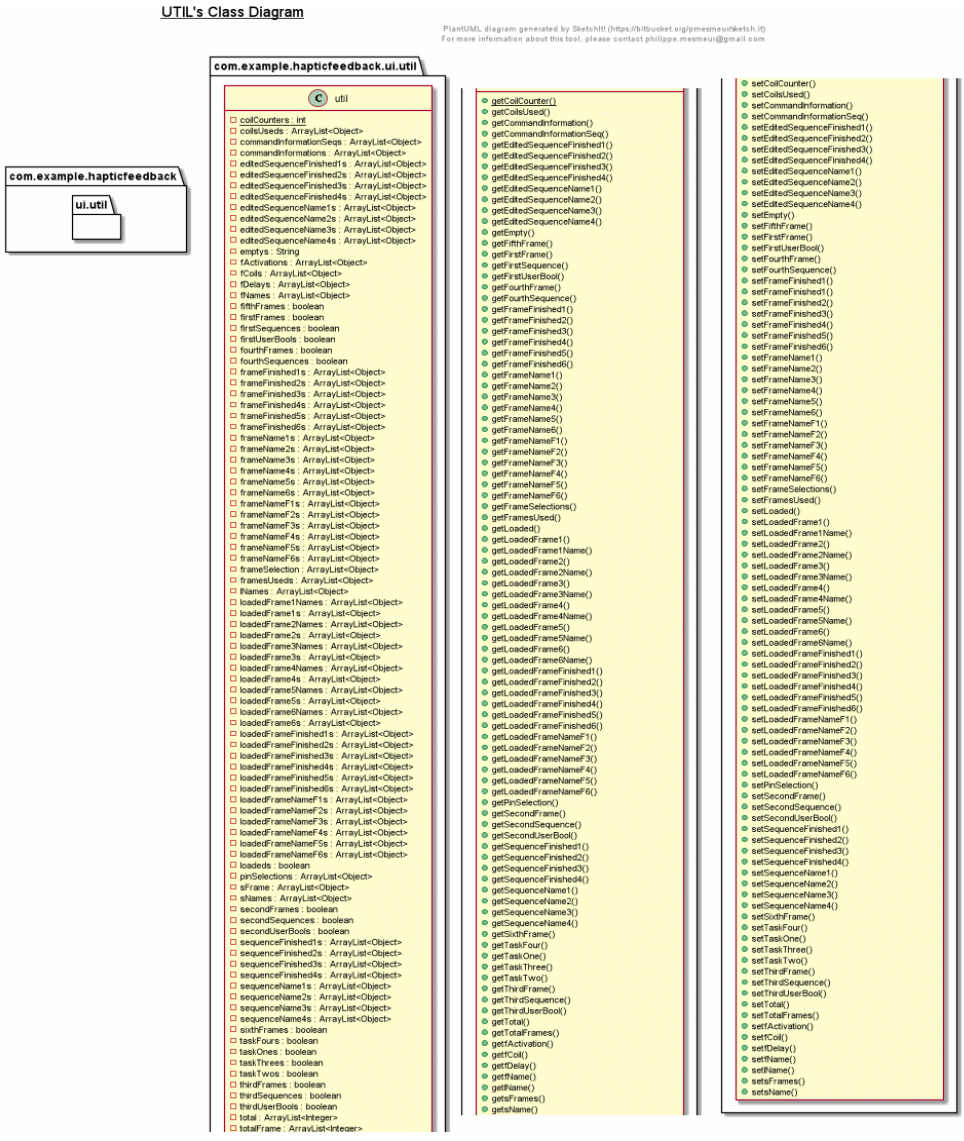

Figure 15: Util Overview

1.5.7 Util

[Figure 15] shows the contents of the util class. This class provides all the getters and setters, along with initial Boolean values and all the ArrayLists that are used throughout the system. It was chosen to put all of these global variables in one class, as it meant that each class could refer back to util in order to retrieve or save data without the need for multiple duplications of data.

1.5.8 Main Activity

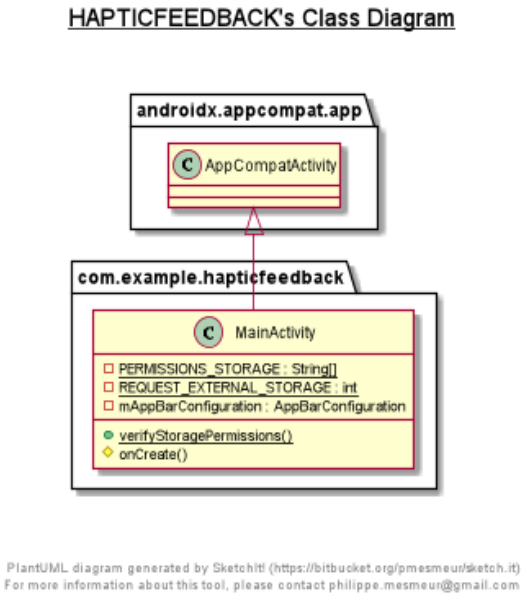

Figure 16: Main Activity Overview

[Figure 16] shows what goes on inside the Main Activity class. This class's main function is to hold the navigation bar that will provide access to the user to other areas of the application. This class also creates the initial application from start-up and requests permissions on saving and accessing external storage. This is important due to the fact that my application saves and loads data to or from the user's device.