

POLITECNICO DI MILANO
MSc Automation and Control Engineering

Software Engineering

Design Document

For the Application of Medication Reminder



Prof. Matteo Giovanni Rossi

Avinash Umesh Sarma 951410
Dineshkumar Ramasamy 964069
Satyabrata Dash 961528

ACADEMIC YEAR 2021/2022

Contents

1. Introduction.....	3
1.1 Scope.....	3
2. Software Architecture.....	3
2.1 Overview.....	3
2.2 Functionalities.....	4
2.3 Class Diagram	5
2.4 Description	5
3. Sequence Diagrams.....	8
3.1 Add Medication.....	8
3.2 Show Medication.....	9
3.3 Modify Medication	10
3.4 Delete Medication	11
3.5 Turn ON the medicine reminder	12
3.6 Turn ON the stock reminder	13
3.7 Restock	14
4. User Interface Design	15
5. MIT App Inventor	16
5.1 Introduction	16
5.2 Overview.....	16
5.3 Internal Architecture.....	18
5.4 Components	18
5.5 Behavior.....	19
6. References	19

1. Introduction

From studying the requirements for the medicine reminder, the Feasibility Study document provides evident justifications for developing the same. The development of the mobile applications follows different phases of documentation, starting from the Feasibility Study Document which explains the feasibility of the concept. Further, the next phase involves the study of requirements to develop the concept that makes the Requirement Analysis and Specifications Document (RASD) while the study of software architecture to follow is enclosed in Design Document (DD). The other phases include the Implementation Document, Testing Document, and User Manual, where the information regarding the documents would be mentioned correspondingly in the later phases of documentation.

Here, this illustrates the Design Document (DD) of the project of the medication reminder abbreviated and named “MedRem”. The application allows the creation of medicine and stock reminders round the clock with a display of the medicines. Design Document discusses the software architecture with all the necessary diagrams to establish the interaction between the components. Furthermore, this document describes a set of design characteristics required for the implementation by introducing constraints and quality attributes. A short note on the MIT App Inventor's development platform is also mentioned at the end of the document.

1.1 Scope

MedRem is an easy-to-use application that helps people to keep track of their medications by providing reminders. It has a simplistic design and vintage approach to the features. This application consists of adding, showing, modifying, and deleting the medicine information. Other than that, two kinds of reminders can be added to the medicines, namely a medicine reminder and a stock reminder. The user can set and modify them according to their intake of the medicines. Restock is yet another feature that is added to the application, allowing the user to update the current stock value.

2. Software Architecture

2.1 Overview

The architecture of the application is structured according to three logic layers:

The **presentation layer** (GUI) handles the interaction with the users. It contains interfaces that can communicate with them and is responsible for rendering the information. Its scope is to make the application's functions understandable to the customers.

The **business or application layer** (Manager) takes care of the functions to be provided for the users. It also coordinates the work of the application, making logical decisions and moving data between the other two layers.

The **data access layer** cares for the management of the information, with the corresponding access to the databases. It picks up useful information for the database users and passes it along to the other layers.

2.2 Functionalities

Some of the major functionalities of the medicine reminder, “MedRem” includes the following:

- **Add/Modify medicines:** The application needs the user to add the medicines, to successfully set reminders according to required schedules and the necessary dosage at the right time. Also, in cases of change in prescriptions, users must be able to modify the information on the medicine itself and further, the timings of reminders.
- **Showing/Deleting the medicines:** The medicines that are added to the application must be shown as a list alphabetically, in a display, for the user to be sure of which medicines they are taking. It should be listing out all the information regarding the medicine including the name, dosage, and frequency. Also, it shows whether the user sets reminders or not, but the list of medicines should be displayed, nevertheless. Once the user has already finished with the intake of a particular medicine, to avoid confusion and save space, they can delete the medicine from the application.
- **Stock Management:** During the time of adding medicines or modifying them, a user could be able to mention the number of pills or the stock of it, to keep in account how many medications are remaining with them. It can be further added or restocked in case of a change in prescriptions.
- **Reminders Management:** The addition of medicines to the application can be accompanied by setting multiple reminders according to the prescription for each medicine. A medicine reminder can be set to keep track of the medicine intake daily once it reaches the pre-set time

by the user. As part of the stock management, a stock reminder can also be set, for it turns on when the number of pills reaches a minimum quantity. Both the reminders use an alarm at the specified time or interval to notify the user.

2.3 Class Diagram

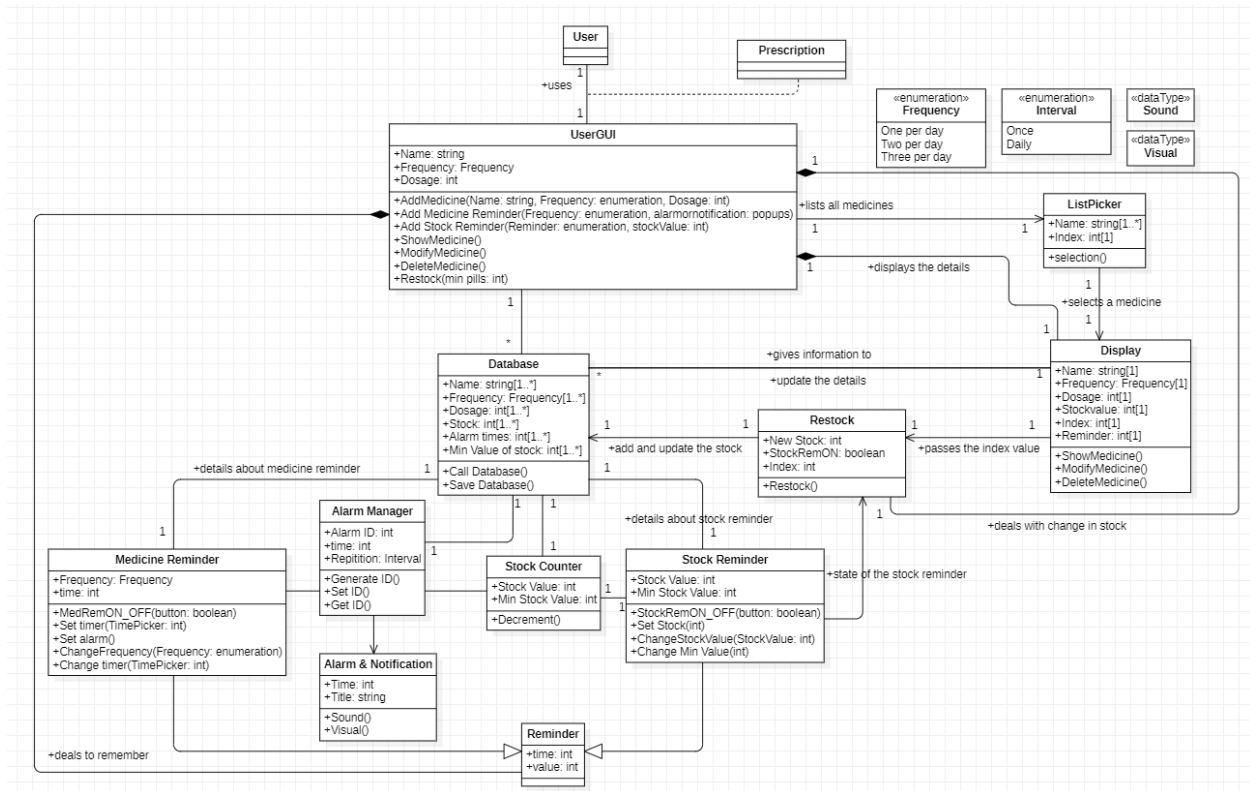


Figure: UML class diagram

2.4 Description

The UML diagram mentioned above is the descriptive version of the system to be developed. It contains various features as mentioned previously. Since it revolves around the user himself, they are the starting point as well as the ending point.

The **User** is the point of origin of our application. He/she has chosen the application to keep a track of their medicines and give reminders to them in case they forget. The user GUI becomes the medium between the user and the application. The prescriptions that the user already have is filled manually into the application which contains various fields like name, frequency, and dosage of the medicine(s).

The **User GUI** is the most important part of our application. After the user downloads the application, the User Graphical User Interface becomes the bridge between the user and the application. Through

the user GUI only, the user can access the application's features like the show and delete medication and perform various operations like adding or modifying medications.

When the user decides to add a medication, the **user GUI** interprets the command and shows a screen to fill in the details like *name*, *frequency*, and *dosage* of the medication using the necessary components like Textboxes. The *frequency* here is an enumeration spinner limited to a maximum of Thrice a day, making it a constraint for ease of development. Further, the user can also set the Reminders, namely the **Medicine Reminders** and **Stock Reminders**, by adding their corresponding attributes. Users can save the details of the medicines, even without setting the reminders by toggling off the respective reminder switches. Once the data is saved, all the information will be stored in the **Database**.

The **Reminder** forms an essential component for the **user GUI**, which includes two types: **Medicine Reminders** and **Stock Reminders**.

A **Medicine Reminder** is a feature assigned by the user to a particular medication for providing a reminder at a specific time or specific periodicity. It contains the following fields as inputs: *frequency* and *time*. The medicine reminder activates only if the medicine reminder button is toggled ON. The frequency represents the periodic nature of the reminder while the time is pre-set by the user, by adjusting the TimePicker component. This feature stores the time in the **Database** corresponding to the same medicine. The information about the time is passed on to the **Alarm Manager**, that deals with the **Alarm**. In case of the time modification for the medicine reminders, the user also has the option to change it, accordingly, saving the new values to the **Database** as well.

A **Stock Reminder** is a feature assigned by the user to a particular medication for providing a reminder only if a minimum threshold value has been reached. The stock reminder contains the following fields as inputs: *stock value* and *minimum stock value*. The stock reminder activates only if the set reminder button is toggled ON. Once the stock is set, the information is again stored in the **Database** and the same data is used to initialize a **Stock Counter**. The stock reminder is also connected with **Restock** module when the *current stock value* is needed to be updated.

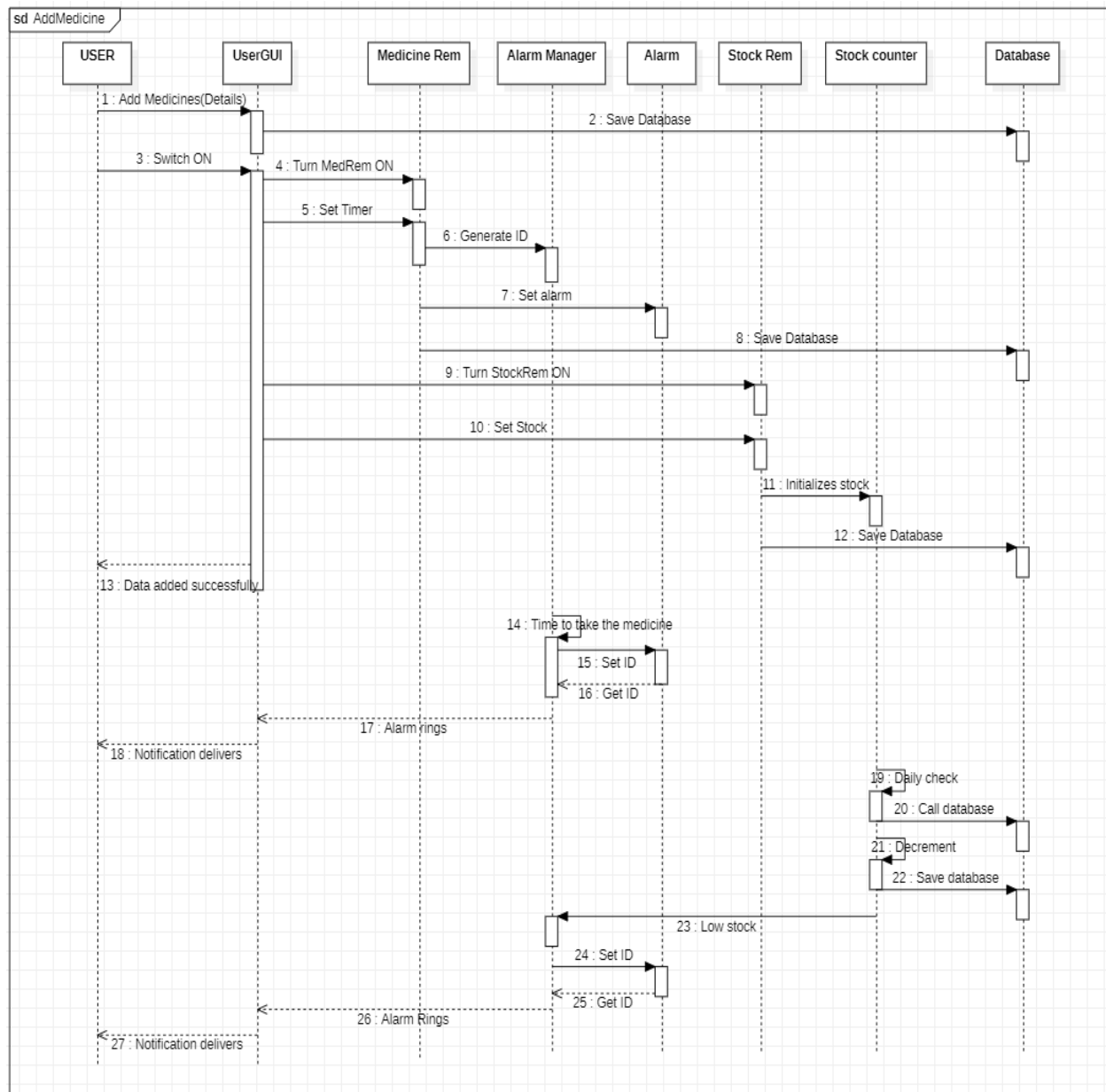
Stock Counter is a module that deals with decrement counting of the *current stock value* based upon the *dosage* and *frequency* which is also compared with the *minimum value* of the corresponding medicines, calling, and saving back to the **Database**. In case of low stock of the medicine, the module communicates with the **Alarm Manager** to give an **Alarm**. If the user needs to renew the stock value of a medicine, the restock module can be made to use that edits the stock value and save it back to the **database**.

The **Alarm Manager** deals with all the *alarms* that needed to be set for **medicine reminders** or **stock reminders** through the **stock counter**. In the case of medicine reminders, once the user sets an alarm, unique *IDs* are generated for each of the alarms based upon the *time*, which are further passed on to the **Alarm Clock**, where the alarms would be turned on according to the *IDs*.

If the user wants to see the details of all the medicine, the **ListPicker** shows the list of medicines that are already added to the **Database**. Further, if the user needs to see detailed information about a particular medicine, the user can also pick the name from the list which takes the user to a **Display** where the entire information about the selected medicine would be displayed. The **ListPicker** also helps to get the *index* of the selected medicine that is again used in **Display** to retrieve the information of medicine in the database. **Display**, being a part of the requirement, is also the instance where the user wants to proceed with modifying, deleting, or even restocking the medicine details. In case of modification, the user can update the details, which would be replaced in the Database, while changing the reminders take the users through the respective reminder modules to change the corresponding attributes. The deletion of data is performed by removing the entry from the Database.

3. Sequence Diagrams

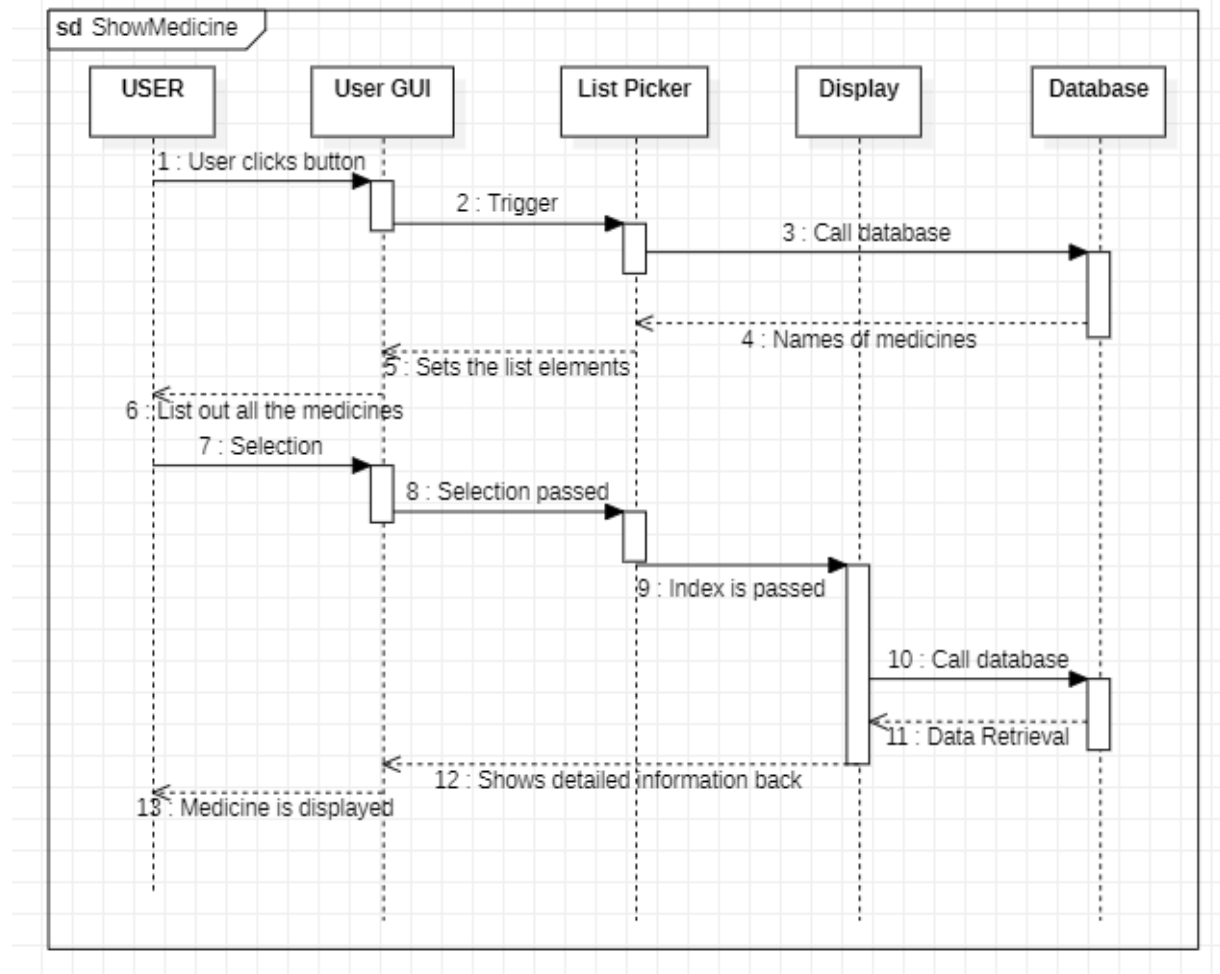
3.1 Add Medication



In the sequence diagram mentioned above, once the user clicks on add medicine, they can fill in the details such as the name, frequency, and dosage value. After the user toggles ON the medicine reminder and stock reminder button, the time value based on the frequency and the stock value and minimum stock value based on the dosage is set to the medication. All the data is stored successfully

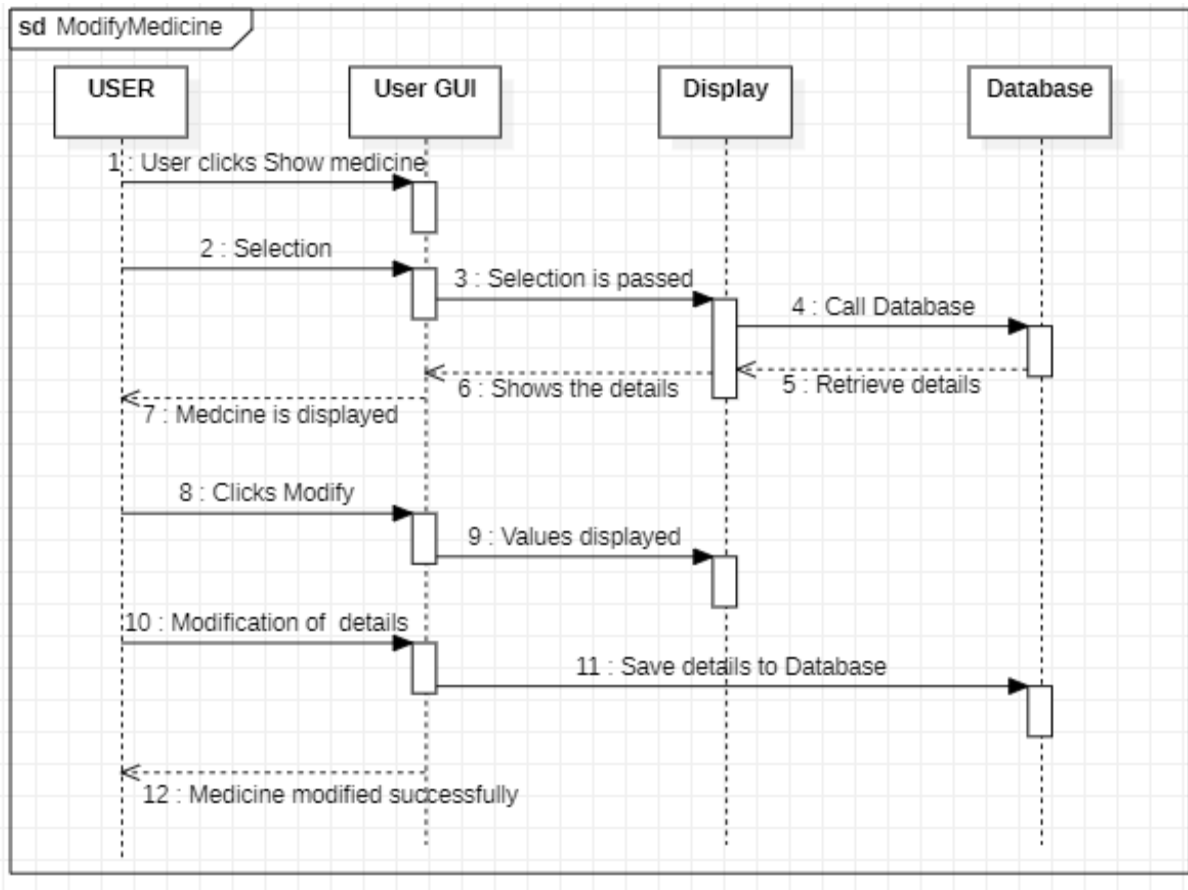
in the database. The medicine reminder generates unique IDs for each reminder and the alarm manager sets them. Notification for the medication intake is provided through an alarm to the user. The stock counter performs a daily check on the stock value and decrements it based on the dosage value and saves it in the database. If the current stock value becomes less than the specified minimum stock value, it reminds the user in form of an alarm notification.

3.2 Show Medication



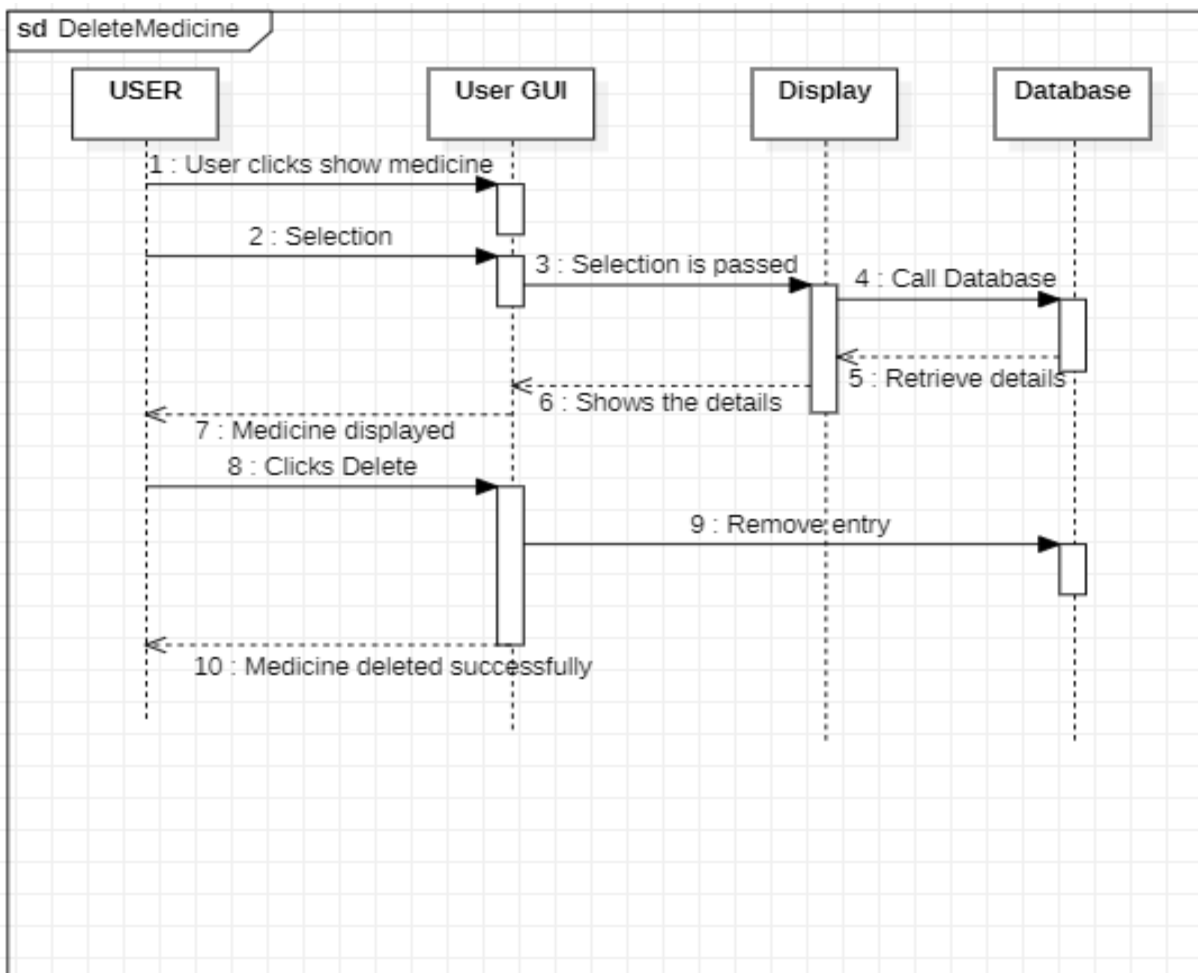
The sequence diagram represents showing of a medication. The user clicks on the button and is directed to the next screen. The trigger generated by the user lets them see the added medicines in the form of a list. The ListPicker acts as a compressed version of the database, where only the index value containing names is called. When the user selects a medication, the selection and the index values are passed to the database. The database returns the related information regarding the selected medicine, thus showcasing it.

3.3 Modify Medication



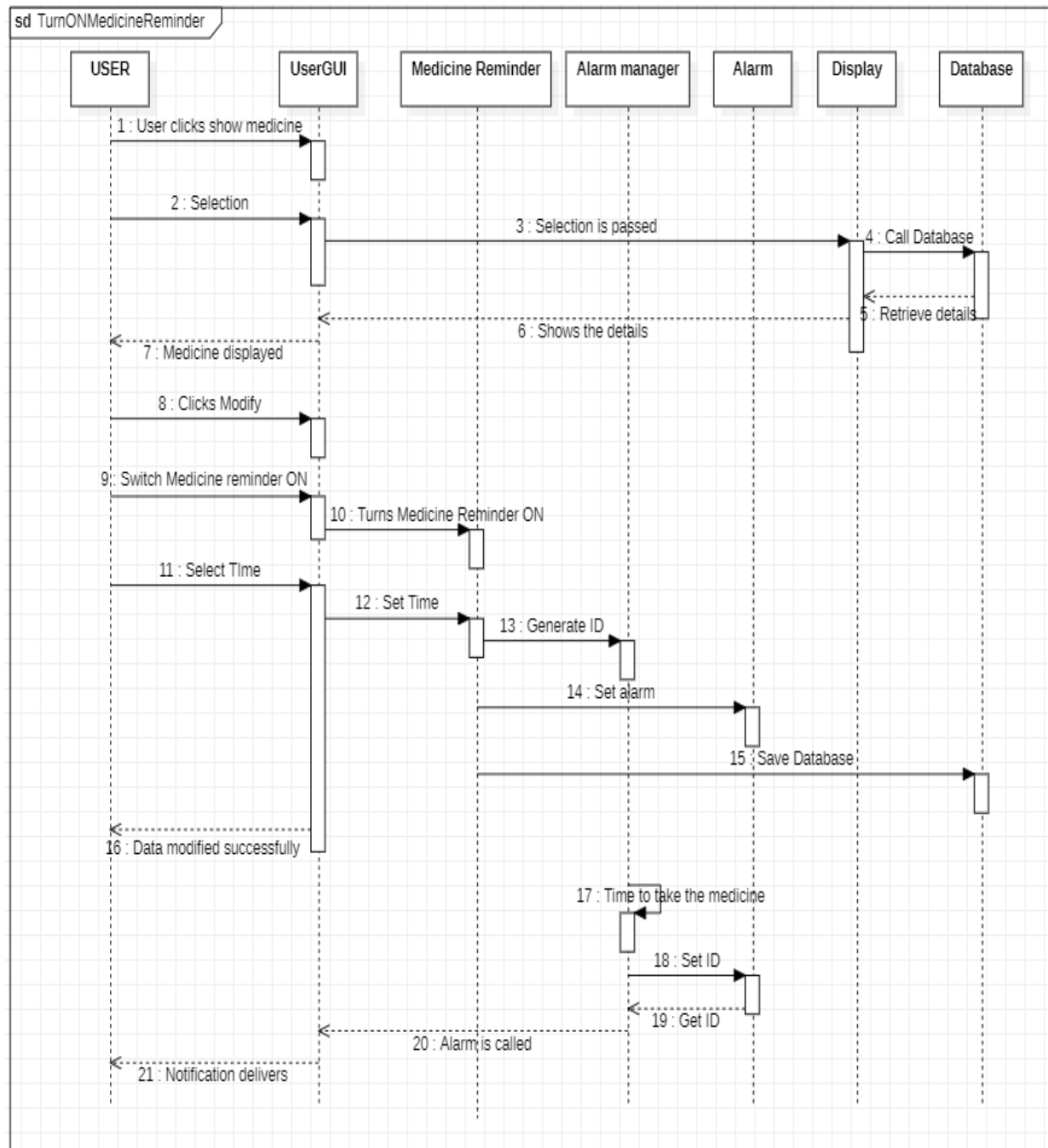
This sequence diagram describes the medicine modification. The user clicks on show medicine. From the GUI, the user is directed to the display, and further to the database. Once the data is returned, the user clicks on the modify. All the previously stated fields are displayed to be modified. Modification of previous details is performed by the user and then saves the new data to the database. Then, the new data is saved and displayed.

3.4 Delete Medication



This sequence diagram describes the medicine deletion. The user clicks on show medicine. From the GUI, the user is directed to the display, and further to the database. Once the data is returned, the user clicks on the delete. As soon as the GUI receives the command to delete the medicine, the operation is successful.

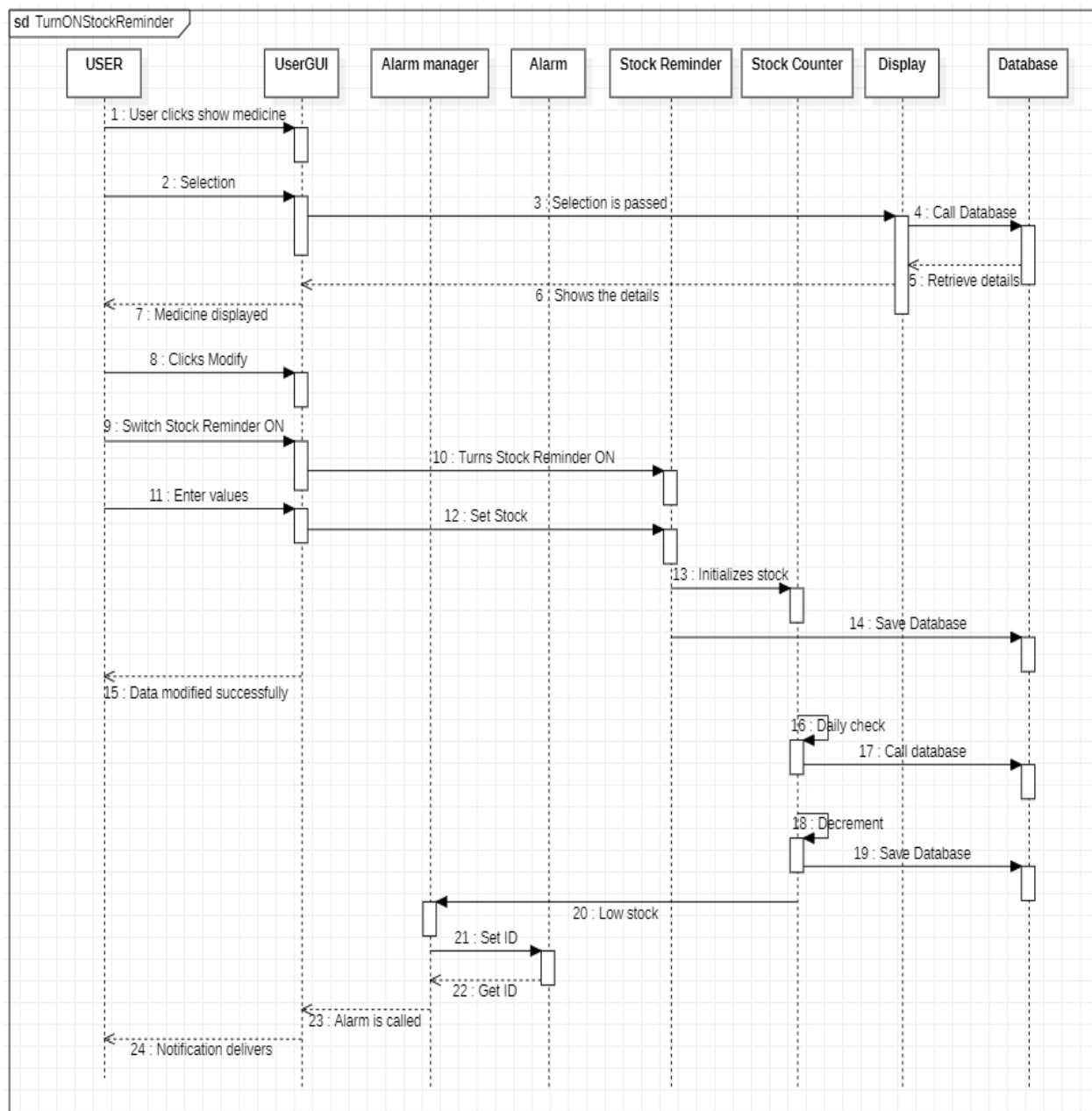
3.5 Turn ON the medicine reminder



For the sequence diagram shown above, the user must have already saved medicine details in the database with the reminders turned off. Thus, the sequence diagram starts with all the necessary operations to show the list of medicines and make the selection of the required medicine to see the complete details. The user can then go on with the modify button to proceed with turning ON the toggle switch for Medicine Reminder and set the time for the reminder with the corresponding

TimePickers. On saving the details to the database, the alarms would be set according to the uniquely generated ID by the Alarm Manager, which is passed on to the Alarms at the event of reaching the time, ringing, and pushing notifications to the user to take medicine.

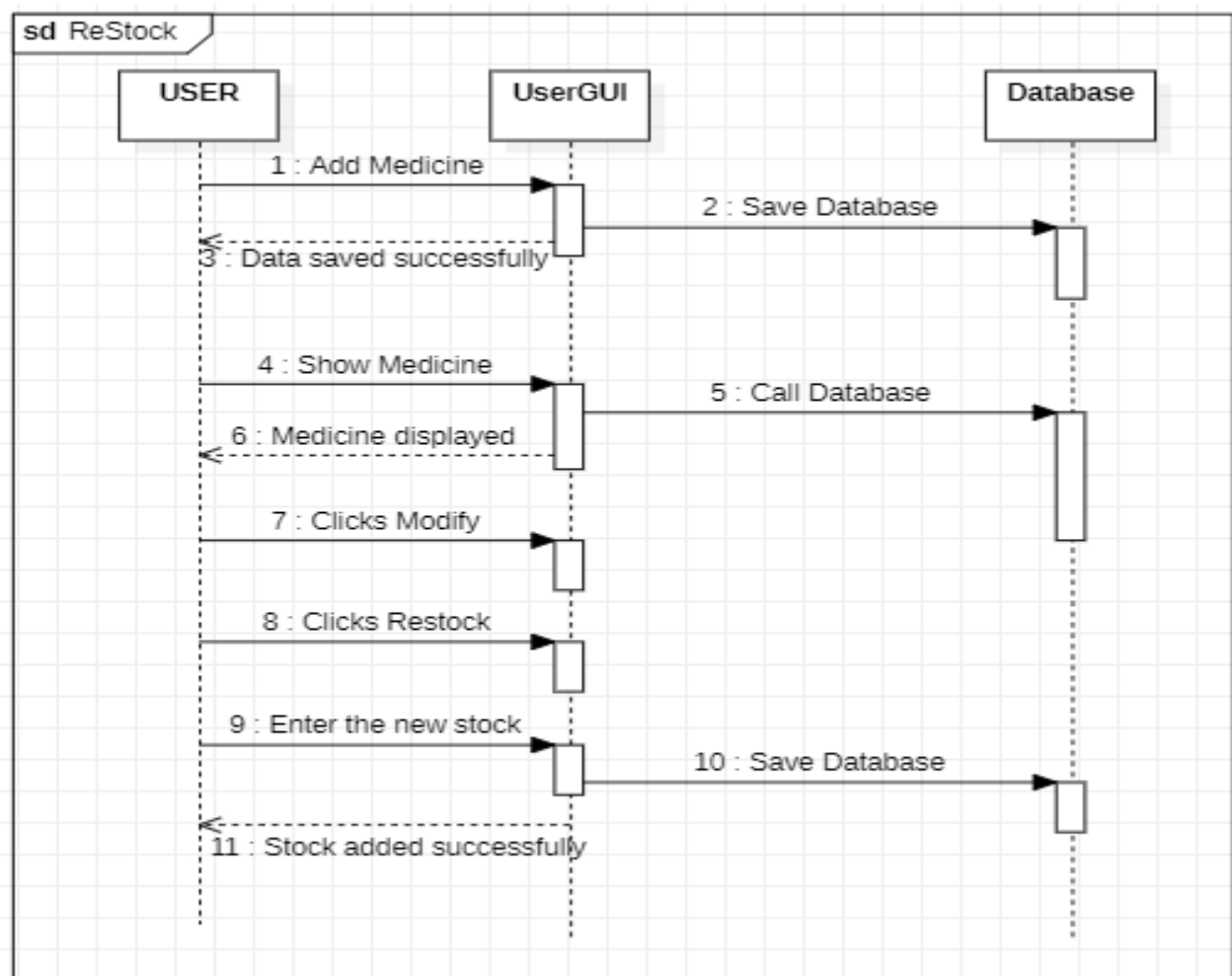
3.6 Turn ON the stock reminder



For the diagram shown above, the user must have already saved medicine details in the database with the reminders turned off. Thus, the sequence diagram starts with all the necessary operations to show

the list of medicines and make the selection of the required medicine to see the complete details. The user can then go on with the modify button to proceed with turning ON the toggle switch for Stock Reminder. On entering the details of stock value and the minimum threshold value, the stock reminder would be set, and the values are stored in the database. Furthermore, the values are also used to initialize the stock counter to decrement the stock daily based on dosage and frequency. When the stock reaches less than the minimum value, the alarm manager is triggered to generate a unique ID to set an alarm for the Low stock of medicine.


3.7 Restock



The sequence diagram for the restock feature starts with the assumption that the user has already added the medicine to the database. From the Home Screen in the GUI, the user clicks on the Show Medicine button to list out all the medicines added. On selecting the medicines to perform the restock operation,

the entire information about the medicine saved in the database is shown through the display, where the user can click on Modify button. Further, the user can proceed by clicking on Restock button to enter the new stock value to be added to the current stock and saved in the database.

4. User Interface Design



23:39	23:39	23:39
Home Screen	Add Medication	Set_Reminder
	Name Medication Name	Medicine Reminder <input checked="" type="checkbox"/>
	Dosage Enter dosage	First Intake Time <input type="text"/> Set Time
	Frequency ▼	Second Intake Time <input type="text"/> Set Time
	Set Reminder	Third Intake Time <input type="text"/> Set Time
		Stock Reminder <input checked="" type="checkbox"/>
		No.of Pills <input type="text" value="Enter the actual quantit"/>
		Minimum Quantity <input type="text" value="Enter The Minimum Qua"/>
MedRem	Cancel	Save
ADD MEDICATION		
SHOW MEDICATION		

23:41

91%

SHOW MEDICATION

Name	Panadol
Dosage	1
Frequency	Three per day
First Intake	2340
Second Intake	2347
Third Intake	2352
Modify	Delete

Modify Medication

Name	Panadol
Dosage	1
Frequency	Three per day

Medicine Reminder

First Intake

2340

Change Time

Second Intake

2347

Change Time

Third Intake

2352

Change Time

Stock Reminder

No.of Pills

60

Cancel

Save

5. MIT App Inventor

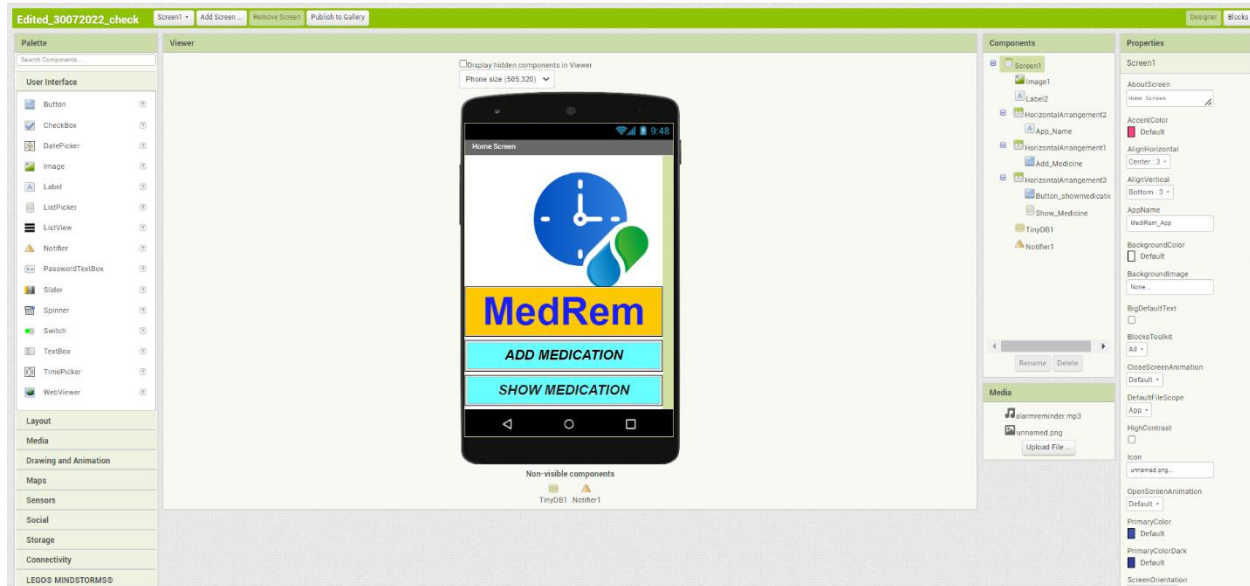
5.1 Introduction

MIT App Inventor is an online development platform that anyone can leverage to solve real-world problems. It provides a web-based “What you see is what you get” (WYSIWYG) editor for building mobile phone applications targeting the Android and iOS operating systems. It uses a block-based programming language built on Google Blockly (Fraser, 2013) and inspired by languages such as StarLogo TNG (Begel & Klopfer, 2007) and Scratch (Resnick et al., 2009; Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010), empowering anyone to build a mobile phone app to meet a need.

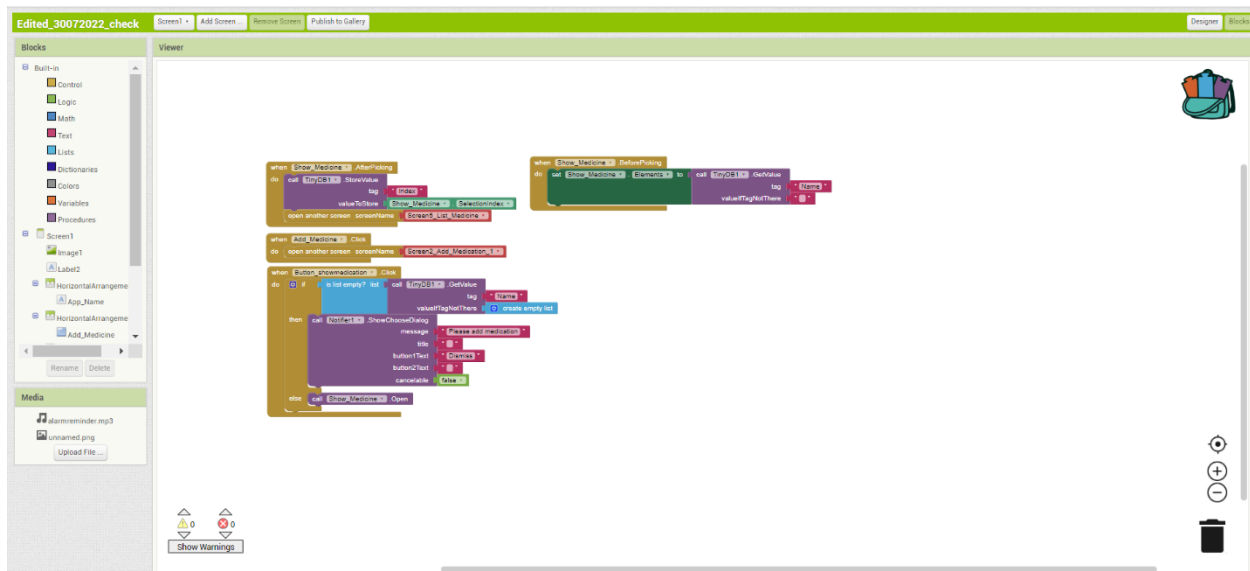
5.2 Overview

The MIT App Inventor user interface includes two main editors: the design editor and the blocks editor. The design editor or designer is a drag-and-drop interface to lay out the elements of the application’s user interface (UI). The blocks editor is an environment in which app inventors can visually lay out the logic of their apps using color-coded blocks that snap together like puzzle pieces to describe the

program. To aid in development and testing, App Inventor provides a mobile app called the App Inventor Companion (or just “the Companion”) that developers can use to test and adjust the behaviour of their apps in real-time. In this way, anyone can quickly build a mobile app and immediately begin to iterate and test.



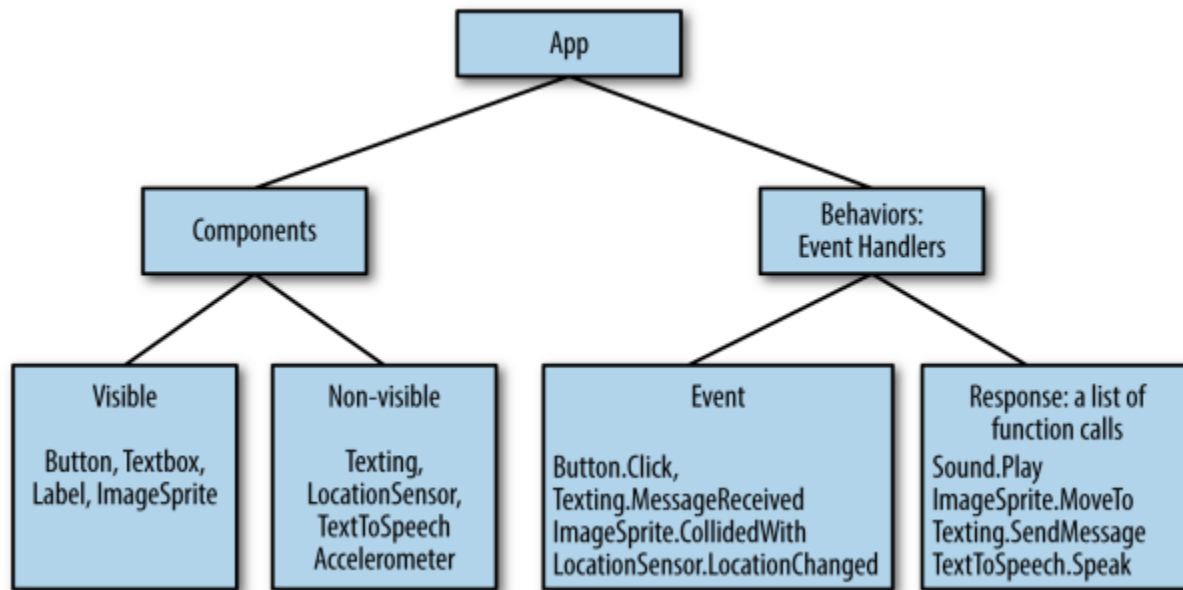
App inventor’s Design editor. App inventors drag components out from the palette (far left) to the viewer (center left) to add them to the app. Inventors can change the properties of the components (far right). An overview of the screen’s components and project media are also displayed (center right)



App Inventor’s Block Editor. Blocks code is typically read left to right, top to bottom.

5.3 Internal Architecture

One way to describe an app's internals is to break it into two parts, its components, and its behaviors. Roughly, these correspond to the two main windows you use in App Inventor: you use the Component Designer to specify the objects (components) of the app, and you use the Blocks Editor to program how the app responds to the user and external events (the app's behavior).



Internal architecture of the App Inventor's app

5.4 Components

There are two main types of components in an app: visible and non-visible. The app's visible components are those that you can see when the app is launched—buttons, text boxes, and labels. These are often referred to as the app's user interface. Non-visible components are those that you can't see, so they're not part of the user interface. Instead, they provide access to the built-in functionality of the device; for example, the Texting component sends and processes SMS texts, the LocationSensor component determines the device's location and the TextToSpeech component talks. The non-visible components are the technology within the device—little worker bees that do jobs for your app. Both visible and non-visible components are defined by a set of properties. Properties are memory slots for storing information about the component. Visible components like buttons and labels have properties such as Width, Height, and Alignment, which together define how the component

looks. Component properties are like spreadsheet cells: you modify them in the Component Designer to define the initial appearance of a component. You can also change the values with blocks.

5.5 Behavior

App components are generally straightforward and easy to understand: a text box is for entering information, a button is for clicking, and so on. An app's behavior, on the other hand, is conceptually difficult and often complex. The behavior defines how the app should respond to events, both user-initiated (e.g., a button clicks) and external (e.g., an SMS text arriving on the phone). The difficulty of specifying such interactive behavior is why programming is so challenging. Fortunately, App Inventor provides a high-level blocks-based language for specifying behaviors. The blocks make programming behaviors more like plugging puzzle pieces together, as opposed to traditional text-based programming languages, which involve learning and typing vast amounts of code. And App Inventor is designed to make specifying event-response behaviors especially easy. The following sections provide a model for understanding app behavior and how to specify it in App Inventor.

6. References

https://link.springer.com/chapter/10.1007/978-981-13-6528-7_3

<http://www.appinventor.org/bookChapters/chapter14.pdf>